2nd Joint Advanced Student School 2004
Course 1
Complexity Analysis of String Algorithms

St. Petersburg, March 28th – April 7th 2004

Preface

The State University St. Petersburg, the Steklov Institute St. Petersburg, and the Technische Universität München organized the second Joint Advanced Student School (JASS 2004) in St. Petersburg from March 28th through April 7th. It was financed by the Bavarian Ministry of Economics, by Siemens and by Infineon. Four courses with different topics were offered. This booklet contains the papers prepared by the students of course 1 "Complexity Analysis of String Algorithms".
The course covered advanced techniques for the analysis of string algorithms applied to fundamental algorithms in this area. There was a special focus on average case analysis and exact analysis (as initiated by D. Knuth). The following topics were coverd by St. Petersburg and Munich participants.

1. **Data Structures for Pattern Matching.** Suffix trees and suffix arrays are a basic data structure in pattern matching.

   Article: [Ukk95, KS03]. Book chapter: 6 of [Gus97].

   Additionally: [MM93]

   Presented by: Olga Sergeeva

2. **Sub-linear Approximate String Matching.** A classical algorithm based on the use of suffix trees, lowest common ancestor queries. The complexity analysis uses basic techniques such as the Chernoff bound.

   Article: [CL94]. Book chapter: 2 of [Szp00], 6 and 8 of [Gus97]. Additionally: [BFC00] (lca)

   Presented by: Robert West

3. **Approximate Text Indexing.** Using simple mathematical arguments the matching probabilities in the suffix tree are bound and by a clever division of the search pattern sub-linear time is achieved.

   Article: [NBY00]. Book chapter: 6 of [Gus97]

   Presented by: Alexander Vahitov

4. **Compressed Suffix Arrays.** This space efficient index structure is analyzed on a bit level to reveal linear size.

   Article: [GV00, Sad00]. Book chapter: 6 of [Szp00].

   Presented by: Fabian Pache

5. **Asymptotic Properties of Suffix Trees.** Analysis of height and feasible path length using probabilistic tools such as 2nd Moment Method.

   Article: [Szp93]. Book chapter: 4 of [Szp00].

   Presented by: Ivan Kazmenko

6. **Sequential Pattern Matching.** Analysis of Knuth-Morris-Pratt type algorithms using the Subadditive Ergodic Theorem.

   Article: [RS98a]. Book chapter: 5 of [Szp00].

   Presented by: Tobias Reichl

7. **Greedy Algorithms for the SCS Problem.** Analysis of some greedy algorithms using tools from information theory such as asymptotic equity property (AEP).

   Article: [FS98]. Book chapter: 6 of [Szp00].

   Presented by: Anton Nesterov

8. **Analysis of Pattern Occurances.** Analysis of the number of occurances of patterns in text using generating functions.

   Article: [GO81, RS98b]. Book chapter: 7 of [Szp00].

   Presented by: Roland Aydin

9. **Rice's Integrals.** Rice's Integrals (already known by Nörlund) are another basic and successful technique to analyze asymptotic behavior of trees.

   Article: [FS95]. Book chapter: 8 of [Szp00].

   Presented by: Thomas Preu

10. **Digital Search Trees.** Analysis of different digital trees with Rice's integrals.

    Article: [FS86a]. Book chapter: 8 of [Szp00].

    Presented by: Nicolai Baron von Hoyningen-Huene

11. **The Mellin Transform.** A very popular technique for the analysis of digital trees and the like.

    Article: [FGD95]. Book chapter: 9 of [Szp00].

    Presented by: Ilja Posov

12. **Automaton Searching on Tries.** Analysis of a general search technique using automatons in tries, based on the eigenvalues of the automatons adjacency matrix and Mellin transform.

    Article: [BYG96]. Book chapter: 9 of [Szp00].

    Presented by: Mikhail Lakunin

Contents

Chapter 1

Data Structures for Pattern Matching

Olga Sergeeva

For many applications, efficient string processing is crucial. Searching for a substring or a subsequence in a string; searching for common substrings of a set of strings; finding out the number of direct repetitions these are just a few important examples of the problems often appearing in work with strings. In the applications, there is need in solving these problems as efficient (in asymptotic) as possible, and also to store the strings 'economically'. So, there arises a question of efficient strings representations, both being compact and providing good bases for algorithms.

This paper is concerned with two representations, in some sense revealing the structure of the initial string and thus meeting many demands: suffix trees and suffix arrays.

1.1 Suffixes

Definition 1.1. Let s be a string. s' is called a suffix of s, if $s = as'$ for some a. Note, that an empty string is a suffix of any string.

It turns out, that if the suffixes are well structured , the resulting construction can be very informative and can be a good base for developing efficient algorithms. Also, both structures we are to discuss can be built in time, linear in the length of the initial string (in this paper, always n).

Why is it possible to build representations, based on suffix relations, efficienly? The very simple (but also very general) idea is that the suffixes are closely related, being parts of each other. Maybe it s worth looking at how this idea refracted in combination with other ideas, and what it lead to in different algorithms, as they historically appeared. But we in this paper will discuss mostly those which made the most of it.

One of the string representations of interest is suffix tree.

1.2 Suffix trees

1.2.1 Definitions, examples, background

Definition 1.2. Suffix tree for a string s is a rooted tree with edges, marked with substrings of s, having the following properties:

1. Any concatenation of the marks along each path from the root to a leaf forms a suffix and every suffix appears once.

2. The marks on the edges, having a common root, begin with different symbols of the alphabet.

From the definition, you can see that there must be as many leaves in the suffix tree as there are non-empty suffixes in s, i.e. n.
What is good in such a representation? The following bright example can give some comprehension of the advantages of suffix trees.

Example 1.1 (Substring search). Suppose that we have a string s and a (shorter) string p, for which we are to answer if it occurs in s.
In fact, even this is not enough detailed description of the problem. The context is important: will we work with s and p once? Or we have a constant pattern, which we are to search in many strings? Or we have a fixed string s, and are to answer many queries about different p? Answers to such questions of course greatly influence the structures and the algorithms preferred.
Suppose that this is s which is fixed (some encyclopedia, for example). So it is s to be preprocessed how can we answer if p is a substring of s, having its suffix tree S?
If p is a substring, then it s a beginning of some suffix. In S, every suffix must appear as a concatenation of marks from the root to a leaf, so if p begins with a symbol, with which no branch from the root begins, we can definitely say that p is not a substring. Otherwise, we should search only in the subtree, corresponding to the branch beginning with this symbol (there is only one such branch from the root!) Reasoning the same for the consequent vertices, we will answer our question, making in each vertex a constant number of comparisons (the alphabet is constant), and there will be no more then the length of p (denote: $|p|$) steps. So overall time is $O(|p|)$ operations plus $O(|s|)$ to build the suffix tree.
Note that for our case the distribution of work is very successful: although for the first query we spend $O(|p| + |s|)$ time, for all the consequent queries time will be linear in the length of the pattern! So, although for the first query the time is the same with, for example, Knuth-Moris-Pratt algorithm, but in that algorithm p needs $|s|$ time preprocessing, not s. So in fact we have an algorithm which, after initial preprocessing, answers the queries in time, linear in the length of p. Mention, that Donald Knuth did not believe in existence of such an algorithm, until suffix trees were invented.

Suffix trees have many other applications, for instance, in search for the longest common substring of a pair (and, further a set of) strings; for repetitions; for longest common ancestor.
It is also like a 'bridge' to much more difficult and important problem of inexact matching when given strings may contain errors and in practice they will contain them!

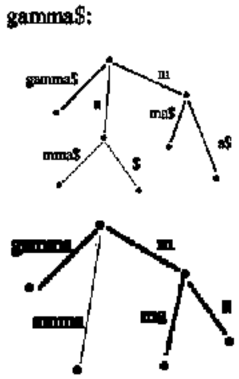


Figure 1.1: Suffix tree and implicit tree for 'gamma'.

Note. We gave a definition of suffix tree, but didn t check correctness of the definition: does suffix tree exist for an arbitrary s? Indeed, if we want to have a leaf for each suffix, then it is impossible to build a suffix tree for a string, in which one suffix s_1 is a prefix of another suffix s_2 we will inevitably spell out the shorter suffix, spelling out the longer one. This problem is easy to solve: if we add a symbol not from alphabet (say, $'\$'$) to the end of s, then no suffix will be a prefix of another suffix.

We will call a tree for a string without additional symbol an implicit tree for S. In implicit tree, there is exactly one path spelling out any suffix, but the ends of some suffixes are not marked anyhow.

1.2.2 Algorithms

The suffix tree can be built naively , adding suffixes to the tree one by one, beginning from the longest suffix (which is the string itself). To add a new suffix, we try to spell it out in the tree, and at the point it s no more possible, create a new branch, marked with the not spelled remainder. This approach demands $O(|s|^2)$ time.
The first linear-time algorithm, by Weiner, appeared in 1973, in his article Linear Pattern Matching Algorithms . Although it was linear in time, it was space-consuming. A less complex and less space-consuming algorithm was invented in 1976 (McCreight. A Space-Economial Suffix Tree Construction Algorithm). Eventually, in 1993, Ukkonen in his On-Line Construction of Suffix Trees introduced his simpler algorithm, having several nice features.

1.2.3 Ukkonen's algorithm

Description

We will start with a simple but not efficient algorithm, and then in several steps, suggested by common sense ('how not to do excessive work?'), will transform it to linear.
Ukkonen s algorithm is on-line: it is split up into $|s|$ phases, after each of them there is an implicit(!) tree for a prefix of s. We begin with the string, containing

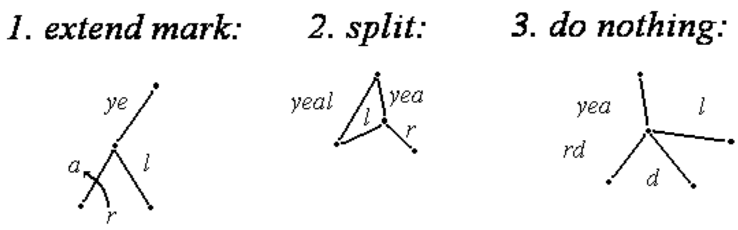


Figure 1.2: Extending 'yea' to 'year' in different trees.

only the first symbol of s, and each phase increase the length of the processed string by one. Phase $i+1$ is itself split into $i+1$ *extensions*, one extension for each from the $i+1$ suffixes of $s[1..i+1]$. In the extension j in the phase $i+1$ algorithm finds the end of the path, marked with $s[j..i]$. It then extends this substring, adding $s(i+1)$ to its end, if it doesn't exist in the tree. So, in the phase $i+1$ it one by one inserts the strings $s[1..i+1], s[2..i+1], .., s[i+1]$ into the tree, if they do not exist.
T_1 consists of one edge, marked with $s(1)$.
After the last phase, we will carry out one more phase , adding symbol \$. The resulting implicit tree for $s\$$ will be a real suffix tree, as we already pointed out. The formal definition of the algorithm is as follows.

Algorithm.
Build T_1.
for i from 1 to $m-1$ do begin {phase $i+1$}
  for j from 1 to $i+1$ begin {extension j}
    find in the current tree the end of the path with mark $S[j..i]$.
    If needed, extend the path with $s(i+1)$, providing existence
    of the string $s[j..i+1]$.
  end;
end;

Let's look closely on the possible cases of extention (see Fig. 2).

1. We are just to extend a mark on an edge
2. When we are to add an edge (and maybe to split an existing edge into two)
3. When nothing is needed to be done.

We will refer to these cases as the first rule, the second rule and the third rule for the algorithm.
What do we spend time for? For looking for the end of the current suffix in a tree after this, we spend constant to prolong it. So, it is essential how we search for the ends of the suffixes in T_i.
We can find the end of a suffix s in $O(|s|)$, walking from the root each time. In this case, we will build the T_{i+1} from T_i in $O(i^2)$, so the final tree will appear

after $O(n^3)$ operations, comparing to $O(n^2)$ in the naive algorithm! We ll reduce this to $O(n)$ using some observations and techniques. Each of them is (just!) a useful heuristics, which cannot qualitatively change estimation of time for the worst case, but applied together they result in essential speeding-up.

Suffix links

Definition 1.3. Suffix link is a pointer from an inner vertex v with the path mark $x\alpha$ to a vertex $s(v)$ with mark α, if it exists in the tree, where x is a symbol, and α is a string. Notice, that if α is empty, suffix link points to the root.

It is easy to see that there is a vertex $s(v)$ for every inner vertex of the tree. Moreover, the following statement is true: if a vertex v with path mark $x\alpha$ is added to the tree in the extension j of the phase $i+1$ (that is, the second rule is applied), then the vertex $s(v)$ either already exists in the tree of will be created in the next extension, in this phase. In both cases, we will find it incidentally, so adding a pointer will not require additional search.
Using suffix links can substantially reduce amount of search.
First, we will maintain a pointer to the end of the longest suffix $s[1..i]$, so for the first extension we need only to prolong a mark on the given edge - no search.
To proceed with the next one, we will not move from the root every time we search for the end of the next suffix. Instead, we will get from the available ending point up to the first inner vertex v (if it s not an inner vertex itself), walk along its suffix link $s(v)$ and search only in the subtree of $s(v)$. If our current suffix can be written in the form $x\alpha\beta$, where $x\alpha$ is the mark of the path to v, then $s(v)$ is marked with α, so trying to spell out the next suffix (which is $\alpha\beta$), we would surely have come to $s(v)$. Adding new suffix links after transformation of the tree in each extension (if needed), we will easily maintain the tree in this pleasant for processing form.

Note. This note will help us to estimate the algorithm complexity.
Denote by depth of a vertex the number of edges on the way from the root to that vertex. Then the moment we are moving along the suffix link, the depth of v, exceeds the depth of $s(v)$ by no more then 1. The reason is that the prefixes of the shorter string (α) will occur in the string more often, so there will be more bifurcations along the way corresponding to α. And in the case no bifurcations are added comparing to the way corresponding to $x\alpha$, $d(x\alpha) - d(\alpha) = 1$. So, getting along the suffix link, we will not get much nearer to the root. Later this will help us to estimate how many times we could descend, and then the overall processing time.

Not check, but search

Still, there are unnecessary steps. The end of the next suffix to prolong surely exists in the tree (in the subtree from $s(v)$, as we cleared up), so we are not to check its existence, but only to find its end. We do not have to compare every symbol on en edge, once having chosen it. If it is shorter then the not spelled out remainder of the suffix, we can skip over it to the next vertex; if it is longer split it at the needed point. Doing this way, we spend for descending each edge constant time. All we need for this style of work is to know the number of symbols on the edge (which is a detail of implementation) and to be able to extract a symbol in s by its number in constant time. Along with the notice on the vertex depths correlation, this yields in the following

Theorem 1.1. *In the improved algorithm, each phase takes $O(n)$ time.*

Proof. Summing up, what steps we perform during one extension? Get up to the nearest inner vertex no more than one edge; go along the suffix link; descend some vertices; apply one of the rules of suffix extension; maybe add a new suffix link. All these actions, except descending, take constant time.
We need to estimate how many times we descend during one phase. Let s pay our attention to the current vertex depth changes. Raising, we decrease it by no more than 1; the same with walking along the suffix link; and while descending, we increase current depth that s why overall depth increment over the phase in not more than $3 * n$. □

Corollary 1.1. *The current version of Ukkonen s algorithm terminates in $O(n^2)$.*

Question. We spent so much effort, and got nothing in comparison with the naive algorithm?

The last touches

Improving an algorithm, one can encounter the following problem: if the output is large, the algorithm cannot be very fast, for working time is no less then output size. This is the case: if we don't change the format of representation the tree, we will not get further optimization, because in general case the overall length of labels on the edges doesn't have to be linear.

Example 1.2. Suffix tree for the string 'abcd...xyz' consists of 26 branches, with marks having 26, 25, ... 1 letters on them. So, the overall length of marks is $26 * 27/2$. Although for arbitrary long strings there will not exist such an example, because the size of the alphabet is considered constant, this example gives notion on how much redundancy can the marks contain.

Example 1.3. Tree for $(a)^n(b)^n(a)^{n-1}b^{n-1}...a^2b^2ab$.

We will modify algorithm in a few touches, taking the following observations:

1^{st}*touch* If we replace the labels with indexes (the beginning and the end of the substring in s), we'll have two numbers, corresponding to each edge, and as the number of edges is less then $2*m-1$, linear space is spent. This also simplifies maintaining the length of the edge (the detail, which we need), making it more consequent. After mentioning that, we can immediately forget that we have not symbols, but numbers, because working with them is the same.

2^{nd}*touch* If $x\alpha\beta$ appeared in the tree, then definitely $\alpha\beta$ appeared also. Then when we are to apply the third rule, we can complete with the phase. So a phase is a consequency of extensions, which use the first (prolonging a mark) and the second (branching off) rules.

$3^{rd}touch$ A leaf cannot become an inner vertex, because the three 'rules' algorithm uses do not transform leaves anyhow.

During the phase, we add the same symbol to edge marks. In terms of indices, we increment by one ending indices, setting them to i on the phase i.

We will split or do nothing only with the suffixes, not processed in the previous phase (those, to which we applied the do nothing rule).

If in the previous phase we ended in extension j, in the beginning of phase we already know that all we have to do with the first j suffixes is to increment by one their last end. That s why it s efficient to write on such edges the mark of infinity, in the phase i implying infinity is i, and to replace them only after the tree is built. This means, that we build the tree 'in one path', moving only 'forward'.

The last observation yealds the

Theorem 1.2. *Ukkonen s algorithm terminates in* $O(n)$.

Difficulties

We saw that suffix trees can make string processing much simpler. At the same time, this is a complicated structure, which is sometimes difficult to implement. The reasons are

1. No "locality" – bad for paging
2. Dependency on the length of the alphabet (Σ): the 'constant' for choosing the right branch gets bigger with grouth of $|\Sigma|$.
3. Number of "children" ranges for different vertices – no general ways of representation: the vertices near the root have many children (almost $|\Sigma|$), and for them arrays provide good representation; the vertices near leaves have almost no children, arrays would be too rarefied, and for these vertices linked lists are preferred; and for 'middle' vertices balanced trees and hashing are better (linked lists would increase time for search, in case $|\Sigma|$ is large).

Due to these reasons, in a number of applications a simpler, although a bit less convenient for algorithms, structure is preferred.

1.3 Suffix arrays

1.3.1 The very first algorithm

Definition 1.4. Suffix array for a string s is an array, containing the suffixes of s in lexicographic order.

The idea to alphabetically order the suffixes belongs to Udi Manber and Gene Myers (1993, "Suffix Arrays: a New Method For On-Line String Searches").
They proposed an algorithm of direct constructing the array (i.e. construction, not based on firstly built suffix tree) in $O(n*logn)$ time. This algorithm not only

builds the array, but on the way gathers some additional information (useful for algorithms). In their article, Manber and Myers also presented an algorithm of search, using this information, for a pattern P in $O(|P| + logn)$ time.
The clear advantage of suffix arrays in comparison with suffix trees is that they are much less complicated structure. So, they were preferred in several fields even though there were no known linear time algorithms for direct suffix array construction. Such algorithms (in amount of three, simultaniously and independently!) appeared in 2003, and we will touch upon one of them.

Note. Search time on suffix arrays ($O(|P| + logn)$) seems to be great loss in comparison with $O(|P|)$ for suffix trees. But in practice, these values ($O(|P|)$ and $O(logn)$) are usually comparable, and search time does not decrease dramatically.

As we mentioned, suffix array can be built (in linear time) from a corresponding suffix tree. This approach has obvious disadvantages. Here is an idea of an algorithm, based on different approach.

Algorithm. As with the trees, we build the array inductively, greatly using it's structure. Initially, we have an array with unordered suffixes. Beginning with sorting the suffixes by the first symbol (which is linear in the string's length - radix sort), every phase we twice the number the suffixes are sorted on.
After the phase H, the suffixes are organized into buckets, holding suffixes with the same H first symbols.

If A(i) is the suffix in the first bucket, A(i-H) should be first in its 2H-bucket. We can move it to the beginning of its 2H-bucket, and mark this fact. For every bucket, we need to know the number of suffixes in this bucket that have already beer moved and placed in 2H-order. The algorithm basically scans the suffixes as they appear in the H-order and for each A(i) it moves A(i-H) (if it exists) to the next available place in its bucket.
The number of phases is logarithmical, so the overall running time is $O(n*logn)$. The implementation is quite interesting, see [MM93].

1.3.2 Skew algorithm

In 2003, independently and in parallel, three different direct linear time suffix array construction algorithms were introduced (by Kim; by Ko and Aluru; and the one we are to consider - 'Skew' algorithm by Juha Karkkainen and Peter Sanders.)
Before getting to the idea of the *skew algorithm*, we need to refer to some background, and thus to give another glance at the history of suffix trees development. Together with the line of algorithms of Weigner, McCreight, Ukkonen, in which the suffix tree is built inductively, with use in some way of the tight relations between suffixes which, provided with insight into these relations, make it possible to organize induction very efficiently - together with this line, there existed another line, introduced by Martin Farach. Farach's suffix tree construction was based on quite different idea: construct separate trees for suffixes

starting at odd positions (recursively) and for the remaining suffixes (using the results of the first step). Merge the two suffix trees into one. Merging, being a difficult procedure, relies on structural properties of suffix trees that are not available in suffix arrays. (Worth mentioning that Farach's approach has an important advantage of not being dependent on the alphabet size.) Kim (the author of another linear algorithm) managed to perform similar merging with suffix arrays, but the procedure is still very complicated.
The skew algorithm has a similar structure:

1. Construct the suffix array of the suffixes starting at positions $i mod 3 \neq 0$. This is done by reduction to the suffix array construction of a string of two thirds the length, which is solved recursively.

2. Construct the suffix array of the remaining suffixes using the result of the first step

3. Merge the two suffix arrays into one.

The use of two thirds instead of half of the suffixes in the first step makes the last step indeed easy: a simple comparison-based merging is sufficient. For example, to compare suffixes strating at i and j with $i mod 3 = 0$ and $j mod 3 = 1$, we first compare the initial characters, and if they are the same, we compare the suffixes starting at $i + 1$ and $j + 1$ whose relative order is already known from the first step (the situation here is quite similar to the situation in the algorithm in previous section, see picture).
The figure 3 gives an example of algorithm. In [KS03], together with useful ideas and theoretical observations, there is a (short and easy understandable) implementation in C++.

Theorem 1.3. *The skew algorithm can be implemented to run in time $O(n)$.*

Proof. The second step is easy, due to the idea, same with the one for step 3. Again: the suffixes S_i with $i mod 3 = 0$ are sorted by sorting the pairs $(s[i], S_{i+1})$, where s is the initial string. So, it's not difficult to see that the second and the third steps require linear time, and the execution time obeys the reccurence $T(n) = O(n) + T(2n/3)$, $T(n) = O(1)$ for $n < 3$. This reccurence has the solution $T(n) = O(n)$. □

Note. In the previous section, we mentioned that generally suffix array should be built together with collecting some additional information (an array of longest common prefixes of suffixes that are adjacent in the suffix array), which makes it much more valuable for algorithms. Referring to [KS03] for details, we will just mention that this information can be gathered in the course of the *skew algorithm* as well.

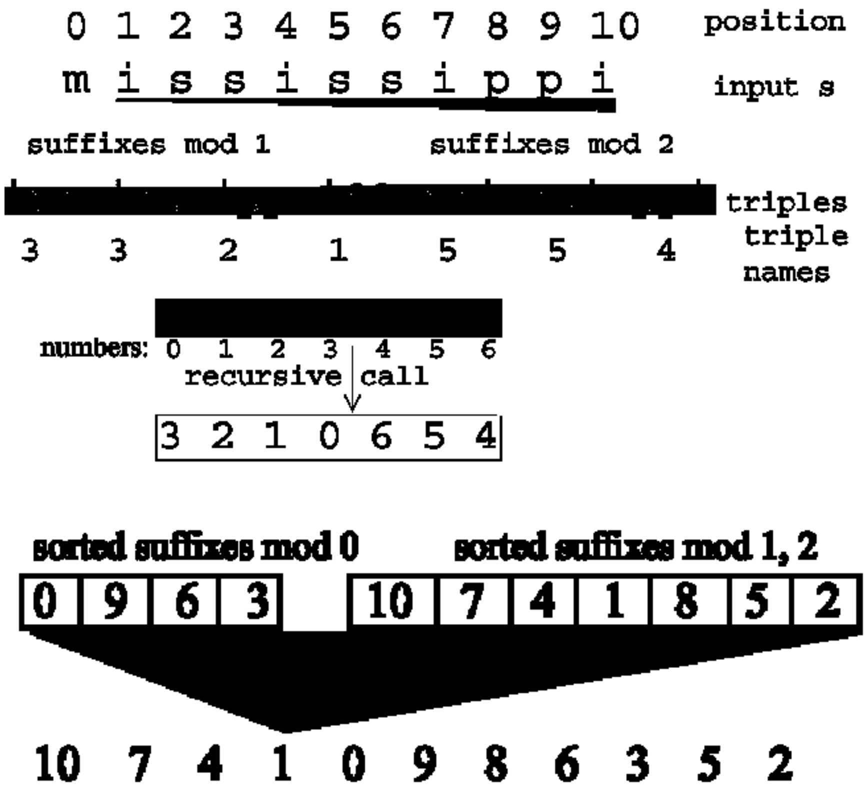


Figure 1.3: Skew algorithm, applied to string 'mississippi'.

Chapter 2

Sublinear Approximate String Matching

Robert West

The present paper deals with the subject of approximate string matching and demonstrates how Chang and Lawler [CL94] conceived a new sublinear time algorithm out of ideas that had previously been known.
The problem is to find all locations in a text of length n over a b-letter alphabet where a pattern of length m occurs with up to k differences (substitutions, insertions, deletions).
The algorithm will run in $\mathcal{O}(\frac{n}{m}k\log_b m)$ time when the text is random and k is bounded by the threshold $m/(\log_b m + \mathcal{O}(1))$. In particular, when $k = o(m/\log_b m)$ the expected running time is $o(n)$.

2.1 Introduction

> I never have found the perfect quote. At best I have been able to find a *string* of quotations which merely circle the ineffible idea I seek to express.
>
> *Caldwell O'Keefe*

2.1.1 What?

In order to be able to explain why and how we will solve a problem, we are bound to exactly define what the problem is.

Definition 2.1. Given a text string T of length n and a pattern string P of length m over a b-letter alphabet, the *k-differences approximate string matching problem* asks for all locations in T where P occurs with at most k differences (substitutions, insertions, deletions).

The following examply will clarify the matter:

Example
```
TORTEL LINI
   YELTSIN
   *  **
```

"YELTSIN" matches in "TORTELLINI" with three differences: The 'Y' in "YELTSIN" is replaced by a 'T', the 'T' in "YELTSIN" is deleted and the 'S' in "YELTSIN" is replaced by an 'L'.

2.1.2 Why?

Approximate string matching as a generalisation of exact string matching has been made necessary due to one major reason:
Genetics is the science that has, in the last years, conjured up a set of new challenges in the field of string processing. (e.g. the search for a string like "GCACTT..." in a huge gene database)
Sequencing techniques, however, are not perfect: The experimental error is up to 5–10%.
Moreover, gene mutation (leading to polymorphism; cf. Darwin's ideas) is a *condicio sine qua non*, it is the mother of evolution. Thus matching a piece of DNA against a database of many individuals must allow a small but significant error.

2.1.3 How?

We will first gather the ingredients (suffix trees, matching statistics, lowest common ancestor retrieval, edit distance) and then merge the ingredients to form the algorithm: We will develop the linear expected time algorithm (LET) in detail and will then obtain the sublinear expected time algorithm (SET) after some modifications.
We will mainly stick to the paper of Chang and Lawler [CL94]; since it is very concise I added material from further sources where it seemed necessary for reasons of understanding.

2.2 The Auxiliary Tools

> Hunger, work and sweat are the best spices.
>
> *Icelandic proverb*

– Not in our case! Our spices are suffix trees, matching statistics, lowest common ancestor retrieval and edit distance. (An apt and detailed introduction to all of these concepts can be found in [Gus97].)

2.2.1 Suffix Trees

Suffix trees may be called *the* data structure for all sorts of string matching. Thus it takes small wonder that they will extensively be used in our case, too.
We will denote the suffix tree of a string $P[1..m]\$$ (\$ is the terminal symbol that appears nowhere but at the last position) as $\mathfrak{S}_P$.
A string α is called a *branching word* (of $P\$$) iff there are different letters x and

y such that both αx and αy are substrings of $P\$$.
Quite obviously, the following correlations hold:

$$\begin{array}{rcl} \text{root} & \longleftrightarrow & \varepsilon \text{ (empty string)} \\ \{\text{internal nodes}\} & \longleftrightarrow & \{\text{branching words}\} \\ \{\text{leaves}\} & \longleftrightarrow & \{\text{suffixes}\} \end{array}$$

Next we define floor(α) to be the longest prefix of α that is a branching word and ceil(α) to be the shortest extension of α that is a branching word or a suffix of $P\$$. Note that α is a branching word iff $\text{floor}(\alpha) = \text{ceil}(\alpha) = \alpha$.
To make matters simpler, we introduce the 'inverted string': Let $\beta^{-1}\alpha$ be the string α without its prefix β; of course this only makes sense, if β really is a prefix of α.
The nodes of the tree will be labelled with the correlating branching words or suffixes respectively, while an edge (β, α) will be labelled with the triple (x, l, r) such that $P\$[l] = x$ and $\beta^{-1}\alpha = P\$[l..r]$.
The following are intuitionally clear: $\text{son}(\beta, x) := \alpha$; $\text{len}(\beta, x) := r - l + 1$. Furthermore, let $\text{first}(\beta, x) := l$ (the position of the first letter in $P\$$, not the letter itself, which is already known to be x).
At last, let shift(α) be α without its first letter (if $\alpha \neq \varepsilon$); so applying the shift function means following a suffix link (cf. [Ukk95, Gus97]).

2.2.2 Matching Statistics

The next crucial data structure to be used is matching statistics. It will store information about the occurrence of a pattern in a text; this is put more precisely by the following

Definition 2.2. The *matching statistics* of text $T[1..n]$ with respect to pattern $P[1..m]$ is an integer vector $\mathfrak{M}_{T,P}$ together with a vector $\mathfrak{M}'_{T,P}$ of pointers to the nodes of $\mathfrak{S}_P$, where $\mathfrak{M}_{T,P}[i] = l$ if l is the length of the longest substring of $P\$$ (anywhere in $P\$$) matching exactly a prefix of $T[i..n]$ and where $\mathfrak{M}'_{T,P}[i]$ points to $\text{ceil}(T[i..i+l-1])$.
More shortly we will write $\mathfrak{M}$ and $\mathfrak{M}'$.

Our goal is to find an $\mathcal{O}(n + m)$ time algorithm for computing the matching statistics of T and P in a single left-to-right scan of T using just $\mathfrak{S}_P$. (We will follow [CL94].)
Brief description of the algorithm: The longest match starting at position 1 in T is found by walking down the tree, matching one letter a time. Subsequent longest matches are found by following suffix links and carefully going down the tree. (cf. Ukkonen's construction of the suffix tree: "skip-and-count trick", [Ukk95, Gus97])

What follows now is some notes that shall serve to clarify the pseudo-code given later:

- i, j, k are indices into T:
  - The i-th iteration computes $\mathfrak{M}[i]$ and $\mathfrak{M}'[i]$.
  - Position k of T has just been scanned.

  - j is some position between i and k.

- Invariants:

  - At all times true:
    (1) $T[i..k-1]$ is a substring of P; $T[i..j-1]$ is a branching word of P.

  - After step 3.1 the following becomes true:
    (2) $T[i..j-1] = \text{floor}(T[i..k-1])$ and corresponds to node α.

  - After step 3.2 the following becomes true as well:
    (3) $T[i..k]$ is not a word.

- If $j < k$ after step 3.1, then $T[i..k-1]$ is not a branching word (2), so neither is $T[i-1..k-1]$.
  So, as substrings of P they must have the same single-letter extension.
  We know from iteration $i-1$ that $T[i-1..k-1]$ is a substring of P (1) but $T[i-1..k]$ is not (3), so $T[k]$ cannot be this letter. Hence the match cannot be extended.

- Together invariants (1) and (3) imply $\mathfrak{M}[i] = k - i$.

- i, j, k never decrease and are bounded by n: $i + j + k \leq 3n$. For every constant amount of work in step 3, at least one of i, j, k is increased. The running time is therefore $\mathcal{O}(n)$ for step 3, and $\mathcal{O}(m)$ for steps 1 and 2 (use e.g. Ukkonen's [Ukk95, Gus97] or McCreight's [Gus97] algorithm to construct the suffix tree), yielding together the desired $\mathcal{O}(n+m)$.

After the above explanations the following code, which computes the matching statistics of T with respect to P, should be more easily understandable:

```
1      construct 𝔖_P in 𝒪(m) time
2      α := root; j := k := 1
3      for i := 1 to n do
3.1        while (j < k) ∧ (j + len(α, T[j]) ≤ k) do     // "skip and count"
               α := son(α, T[j]);
               j := j + len(α, T[j])
           elihw
3.2        if j = k then     // extend the match
               while son(α, T[j]) exists ∧ T[k] = P$[first(α, T[j]) + k − j] do
                   k := k + 1
                   if k = j + len(α, T[j]) then
                       α := son(α, T[j]);
                       j := k fi
               elihw
           fi
3.3        𝔐[i] := k − i
           if j = k then 𝔐′[i] := α
               else 𝔐′[i] := son(α, T[j]) fi
3.4        if (α is root) ∧ (j = k) then
               j := j + 1;
               k := k + 1 fi
           if (α is root) ∧ (j < k) then
               j := j + 1 fi
           if (α is not root) then
               α := shift(α) fi
       rof
```

2.2.3 Lowest Common Ancestor

Having introduced the notion of suffix trees, we will at some points while deriving the main algorithm be interested in the node that is an ancestor to two given nodes of the tree and that is 'minimal' with that quality, meaning it is the node furthest from the root. More formally:

Definition 2.3. For nodes u, v of a rooted tree $\mathfrak{T}$, $\textsc{lca}(u, v)$ is the node furthest from the root that is an ancestor to both u and v.

Our goal is a constant time LCA retrieval after some preprocessing; it is achieved by reducing the LCA problem to the *range minimum query (RMQ)* problem as proposed in [BFC00]. RMQ operates on arrays and is defined as follows.

Definition 2.4. For an array $\mathfrak{A}$ and indices i and j, $\textsc{rmq}_{\mathfrak{A}}(i, j)$ is the index of the smallest element in the subarray $\mathfrak{A}[i..j]$.

For ease in notation we will say that, if an algorithm has preprocessing time $p(n)$ and query time $q(n)$, it has complexity $\langle p(n), q(n)\rangle$.
The following main lemma states how closely the complexities of RMQ and LCA are connected.

Lemma 2.1. *If there is a $\langle p(n), q(n)\rangle$-time solution for RMQ on a length n array, then there is a $\langle \mathcal{O}(n) + p(2n-1), \mathcal{O}(1) + q(2n-1)\rangle$-time solution for LCA in a tree with n nodes.*

The $\mathcal{O}(n)$ term will come from the time needed to create the soon-to-be-presented arrays. The $\mathcal{O}(1)$ term will come from the time needed to convert the RMQ answer on one of these arrays to the LCA answer in the tree.

Proof. The LCA of nodes u and v is the shallowest (i.e. closest to the root) node between the visits to u and v encountered during a depth first search (DFS) traversal of $\mathfrak{T}$ (n nodes; labels: $1, ..., n$).

Therefore, the reduction proceeds as follows:

1. Let array $\mathfrak{D}[1..2n-1]$ store the nodes visited in a DFS of $\mathfrak{T}$. $\mathfrak{D}[i]$ is the label on the i-th node visited in the DFS.
2. Let the *level* of a node be its distance from the root. Compute the level array $\mathfrak{L}[1..2n-1]$, where $\mathfrak{L}[i]$ is the level of node $\mathfrak{D}[i]$.
3. Let the *representative* of a node be the index of its first occurrence in the DFS. Compute the representative array $\mathfrak{R}[1..n]$, where $\mathfrak{R}[w] = \min\{j \mid \mathfrak{D}[j] = w\}$.

All of this is feasible during a single DFS; thus the running time is $\mathcal{O}(n)$.
Now the LCA may be computed as follows (suppose u is visited before v): The nodes between the first visits to u and v are stored in $\mathfrak{D}[\mathfrak{R}[u]..\mathfrak{R}[v]]$. The shallowest node in this subtour is found at index $\text{RMQ}_{\mathfrak{L}}(\mathfrak{R}[u], \mathfrak{R}[v])$. The node at this position and thus the output of $\text{LCA}(u, v)$ is $\mathfrak{D}[\text{RMQ}_{\mathfrak{L}}(\mathfrak{R}[u], \mathfrak{R}[v])]$.
The time complexity is as claimed in the lemma: Just $\mathfrak{L}$ (size $2n-1$) must be proprocessed for RMQ. So the total preprocessing time is $\mathcal{O}(n) + p(2n-1)$. For the query one RMQ in $\mathfrak{L}$ and three constant time array lookups are needed. In total we get $\mathcal{O}(1) + q(2n-1)$ query time. □

What about RMQ's complexity? – After procomputing (at least a crucial part of) all possible queries, lookup time is $q(n) = \mathcal{O}(1)$.
Preprocessing time $p(n)$ is $\mathcal{O}(n^3)$ by a brute force algorithm: For all possible index pairs, search the minimum.
It is $\mathcal{O}(n^2)$ if we fill the table by dynamic programming. This is, however, still naive.
A far better algorithm runs in $\mathcal{O}(n \log n)$ time: Precompute only queries for blocks of a power-of-two length; remaining answers may then be inferred in constant time at the moment of query.
Finally, a really clever $\mathcal{O}(n)$ time solution was proposed by Bender and Colton: It makes use of the fact that adjacent elements in $\mathfrak{L}$ differ by exactly ± 1; so only solutions for the few generic ± 1-patterns are precomputed. It can be shown that there are only $\mathcal{O}(\sqrt{n})$ such patterns and that preprocessing is then $\mathcal{O}(\sqrt{n}(\log n)^2) = \mathcal{O}(n)$.
For details regarding the two latter cases, I refer to [BFC00]; dwelling upon them would prolong this report in an undue way.

2.2.4 Edit Distance

We will now introduce another data structure called 'edit distance', which will be needed in the main algorithm:

Definition 2.5. The *edit distance* (or Levenshtein distance) between two strings S_1 and S_2 is the minimum number of edit operations (insertions, deletions, substitutions) needed to transform S_1 into S_2.

Such a transformation may be coded in an *edit transcript*, i.e. a string over the alphabet $\{I, D, S, M\}$, meaning "insertion", "deletion", "substitution" or "match" respectively. The following example will clarify this (S_1 is to be transformed to S_2). Note there may be more than one transformation – even more than one optimal transformation.

Example
```
SIMDMDMMI
v intner  = S1
wri t ers = S2
```

In the now to be presented 'naive' dynamic programming solution we need not use the auxiliary tools already gathered (suffix trees, matching statistics, LCA), but later on – in the Landau–Vishkin algorithm – we will do so to obtain some lower complexity.

Lemma 2.2. *The edit distance is computable using dynamic programming.*

Proof. What we want to do is to build the table $\mathfrak{E}$ where $\mathfrak{E}[i, j]$ denotes the edit distance between $S_1[1..i]$ and $S_2[1..j]$. Then $\mathfrak{E}[n_1, n_2]$ will be the edit distance between the complete strings $S_1[1..n_1]$ and $S_2[1..n_2]$.
The base conditions are: $\mathfrak{E}[i, 0] = i$ (all deletions); $\mathfrak{E}[0, j] = j$ (all insertions)
The recurrence scheme is:

$$\mathfrak{E}[i, j] = \min\{\mathfrak{E}[i, j-1] + 1, \mathfrak{E}[i-1, j] + 1, \mathfrak{E}[i-1, j-1] + I_{ij}\},$$

where $I_{ij} = 0$, if $S_1[i] = S_2[j]$, and $I_{ij} = 1$ otherwise.

The last letter of an optimal transcript is one of $\{I, D, S, M\}$. The recurrence selects the minimum of these possibilities. For example, if the last letter of an optimal transcript is I then $S_2[n_2]$ is appended to the end of S_1, and what remains to be done is to transform S_1 to $S_2[1..n_2 - 1]$. The edit distance between these two strings is already known, and the first of the three possibilities is chosen. □

Here is some part of the table for the example given above (The arrows point to the cells from which a particular cell may be computed and thus represent one of $\{I, D, S, M\}$). That there is often more than one arrow makes clear once more that there may be more than one optimal way to transform one string into another. Each trace that follows a sequence of arrows from cell (n_1, n_2) to cell $(1, 1)$ represents an optimal transcript.

$\mathfrak{E}[i,j]$	S_2		w	r	i	t	e	r	s
S_1		0	1	2	3	4	5	6	7
	0	0	$\leftarrow 1$	$\leftarrow 2$	$\leftarrow 3$	$\leftarrow 4$	$\leftarrow 5$	$\leftarrow 6$	$\leftarrow 7$
v	1	$\uparrow 1$	$\nwarrow 1$	$\nwarrow\leftarrow 2$	$\nwarrow\leftarrow 3$	$\nwarrow\leftarrow 4$	$\nwarrow\leftarrow 5$	$\nwarrow\leftarrow 6$	$\nwarrow\leftarrow 7$
i	2	$\uparrow 2$	$\nwarrow\leftarrow 2$	$\nwarrow 2$	$\nwarrow 2$	*			
n	3	$\uparrow 3$							
t	4	$\uparrow 4$							
n	5	$\uparrow 5$							
e	6	$\uparrow 6$							
r	7	$\uparrow 7$							

The complexity is $\mathcal{O}(|S_1| \cdot |S_2|)$.
Also note that rows, columns and diagonals are non-decreasing and differ by at most one.

We need, however, some slightly different thing: We need the minimum number of operations to transform $P[1..m]$ so that it *occurs* in $T[1..n]$, not that it actually *is* T; i.e. we want starting spaces to be "free".
Thus we need a table $\mathfrak{D}$, where

$$\mathfrak{D}[i, j] := \min_{1 \leq l \leq j} \{\text{edit distance between } P[1..i] \text{ and } T[l..j]\}$$

We achieve this by simply changing the base conditions: $\mathfrak{D}[i, 0] = i$ (as before: all deletions); $\mathfrak{D}[0, j] = 0$ (ε ends anywhere).
There is a match if row m is reached and if the value there is $\leq k$.

It is possible to reduce the complexity from $\mathcal{O}(mn)$ to $\mathcal{O}(kn)$; the method is called the *Landau–Vishkin algorithm (LV)*:

Call cell $\mathfrak{D}[i,j]$ an entry of diagonal $j-i$ (range: $-m,...,n$).
LV makes use of one fact: When computing $\mathfrak{D}$, we may not omit whole rows or whole columns, but we may in fact omit whole diagonals. That is why we will not compute $\mathfrak{D}$ but, column by column, the $(k+1)\times(n+1)$ "meta table" $\mathfrak{L}$ where $\mathfrak{L}[x,y]$ (with x ranging from 0 to k and y from 0 to n) is the row number of the last (i.e. deepest) x along diagonal $y-x$.
$-k \le y-x \le n$, so all relevant diagonals and thus solutions are represented because $\mathfrak{D}[k+1,0]=k+1>k$ and diagonals are non-decreasing.
There is a solution if row m is reached in $\mathfrak{D}$, i.e. if $\mathfrak{L}[x,y]=m$; then there is a match ending at position $m+y-x$ with x differences.

First, define $\mathfrak{L}[x,-1]=\mathfrak{L}[x,-2]:=-\infty$; this is sensible because every cell of diagonal $-1-x$ is at least $\mathfrak{D}[x+1,0]=x+1>x$.
Now fill row 0: $\mathfrak{L}[0,y]=\text{jump}(1,y+1)$, where jump$(i,j)$ is the longest common prefix of $P[i..m]$ and $T[j..n]$, i.e.

$$\text{jump}(i,j) := \min\{\mathfrak{M}[j], \text{length of word } \textsc{lca}(\mathfrak{M}'[j], \text{leaf } P\$[i..m])\}.$$

Consider some part of $\mathfrak{L}$: $\begin{array}{c|c|c|c|} & \multicolumn{3}{l}{y \rightarrow} \\ \cline{2-4} x & \alpha & \beta & \gamma \\ \cline{2-4} \downarrow & & & \mathfrak{L}[x,y] \\ \cline{2-4} \end{array}$

$\alpha=\mathfrak{L}[x-1,y-2]$, $\beta=\mathfrak{L}[x-1,y-1]$ and $\gamma=\mathfrak{L}[x-1,y]$ denote respectively the row numbers of the last $x-1$ on diagonals $y-x-1$, $y-x$ and $y-x+1$. Along diagonal $y-x$ the cells at rows α, $\beta+1$ and $\gamma+1$ are at most x (due to the non-decreasingness and difference by at most one). The cell in row $\beta+1$ must be exactly x. So it follows from the non-decreasingness of diagonals that the deepest of these three cells (in row $t:=\max\{\alpha,\beta+1,\gamma+1\}$) must have value x, too. To find the row where the *last* x occurs on diagonal $y-x$, we go down this diagonal as long as the value does not increase to $x+1$. Thus cell (x,y) may be computed as follows:

$$\mathfrak{L}[x,y]=t+\text{jump}(t+1,t+1+y-x)$$

2.3 The Algorithm

Virtus in usu sui tota posita est.

Marcus Tullius Cicero, De re publica

– "The value of virtue is entirely in its use." So let us apply all the "virtual" assets we have gathered up to now and make use of them in the core algorithm.

2.3.1 Linear Expected Time

The algorithm me develop first will run in $\mathcal{O}(n)$ time when the following two conditions hold:

1. $T[1..n]$ is a uniformly random string over a b-letter alphabet.
2. The number of differences allowed in a match is

$$k < k^* = \frac{m}{\log_b m + c_1} - c_2.$$

(the constants c_i are to be specified later; m is the pattern length)

Pattern P need not be random.

The algorithm is named after those who invented it – Chang and Lawler (CL) – and is given in pseudo-code below:

$s_1 := 1$; $j := 1$
do
  $s_{j+1} := s_j + \mathfrak{M}[s_j] + 1$; *// compute the "start positions"*
  $j := j + 1$
until $s_j > n$
$j_{max} := j - 1$
for $j := 1$ to j_{max} **do**
  if $(j + k + 2 \leq j_{max}) \wedge (s_{j+k+2} - s_j \leq m - k)$ **then**
    apply LV to $T[s_j..s_{j+k+2} - 1]$ **fi** *// "work at s_j"*
rof

It is not presently obvious that the algorithm is correct. So we will have a closer look at it:

If $T[p..p + d - 1]$ matches P and $s_j \leq p \leq s_{j+1}$, then this string can be written in the form $\zeta_1 x_1 \zeta_2 x_2 ... \zeta_{k+1} x_{k+1}$, where each x_l is a letter or empty, and each ζ_l is a substring of P. As may be shown by induction, for every $0 \leq l \leq k + 1$, $s_{j+l+1} \geq p + \text{length}(\zeta_1 x_1 ... \zeta_l x_l)$. So in particular $s_{j+k+2} \geq p + d$, which implies $s_{j+k+2} - s_j \geq d \geq m - k$. So CL will perform work at start position s_j and thereby detect there is a match ending at position $p + d - 1$.

If we can show the probability to perform work at s_1 is small, this will be true for all s_j's because they are all stochastically independent and equally distributed (because any knowledge of all the letters before s_j is of no use when "guessing" s_{j+1}).

Since $s_{k^*+3} - s_1 \geq s_{k+3} - s_1$ and $m - k \geq m - k^*$, the event $s_{k+3} - s_1 \geq m - k$ implies the event $s_{k^*+3} - s_1 \geq m - k^*$.

So $\Pr[s_{k^*+3} - s_1 \geq m - k^*] \geq \Pr[s_{k+3} - s_1 \geq m - k]$ and it suffices to prove the following lemma.

Lemma 2.3. *For suitably chosen constants c_1 and c_2, and $k^* = \frac{m}{\log_b m + c_1} - c_2$,*

$$\Pr[s_{k^*+3} - s_1 \geq m - k^*] < 1/m^3.$$

Proof. For the sake of easiness, let us assume (i) $b = 2$ ($b > 2$ gives slightly smaller c_i's) and (ii) k^* and $\log m$ are integers ($\log m := \log_2 m$).

Let X_j be the random variable $s_{j+1} - s_j$.

Note that $s_{k^*+3} - s_1 = X_1 + ... + X_{k^*+2}$ (telescope sum).

There are $m2^d$ different strings of length $\log m + d$, but at most m such substrings of P. Together with the fact that $X_1 = \mathfrak{M}[1] + 1$, we have

$$\Pr[X_1 = \log m + d + 1] < 2^{-d} \quad \text{for all integer } d \geq 0 \tag{2.1}$$

Note that all the X_j are stochastically independent and equally distributed (you do not gain any knowledge if you know how big the gap between the previous start positions was). So $\mathbf{E}[X_j] = \mathbf{E}[X_1]$ for all j. We will now show that $\mathbf{E}[X_j] < \log m + 3$:

$$
\begin{aligned}
\mathbf{E}[X_j] = 1 + \mathbf{E}[\mathfrak{M}[1]] &= 1 + \log m + \mathbf{E}[\mathfrak{M}[1] - \log m] \\
&= 1 + \log m + \sum_{d=1}^{\infty} d \Pr[\mathfrak{M}[1] - \log m = d] \\
&= 1 + \log m + \sum_{d=1}^{\infty} d \Pr[X_1 = \log m + d + 1] \\
&< 1 + \log m + \sum_{d=1}^{\infty} d \left(\frac{1}{2}\right)^d \\
&= 1 + \log m + \frac{1}{2} / \left(1 - \frac{1}{2}\right)^2 = \log m + 3
\end{aligned}
$$

Now let $Y_i := X_i - \frac{m-k^*}{k^*+2}$ and
apply Markov's inequality: $\Pr[X \geq h] \leq \mathbf{E}[X]/h$, for all $h > 0$ ($t > 0$):

$$
\begin{aligned}
\Pr[X_1 + ... + X_{k^*+2} \geq m - k^*] &= \Pr[Y_1 + ... + Y_{k^*+2} \geq 0] \\
&= \Pr[e^{t(Y_1 + ... + Y_{k^*+2})} \geq e^{t \cdot 0}] \\
&\leq \mathbf{E}[e^{t(Y_1 + ... + Y_{k^*+2})}]/1 \\
&= \mathbf{E}[e^{tY_1} \cdot ... \cdot e^{tY_{k^*+2}}] \\
&= \mathbf{E}[e^{tY_1}] \cdot ... \cdot \mathbf{E}[e^{tY_{k^*+2}}] \\
&= \mathbf{E}[e^{tY_1}]^{k^*+2}
\end{aligned}
$$

Note that we used the independence and equal distribution of the variables Y_j, which follows from the independence and equal distribution of the variables X_j.
Inequality (2.1): $\Pr[X_1 = \log m + d + 1] < 2^{-d}$, is equivalent to

$$
\Pr[Y_1 = \log m + d + 1 - \frac{m - k^*}{k^* + 2}] < 2^{-d} \quad \text{for all integer } d \geq 0
$$

So, the theorem of total expectation implies, for all $t > 0$ ($\alpha := \log m + 1 - \frac{m-k^*}{k^*+2}$),

$$
\begin{aligned}
\mathbf{E}[e^{tY_1}] &= \mathbf{E}[e^{tY_1} | Y_1 \leq \alpha] \cdot \underbrace{\Pr[Y_1 \leq \alpha]}_{\leq 1} + \\
&\quad + \sum_{d=1}^{\infty} \mathbf{E}[e^{tY_1} | Y_1 = \alpha + d] \cdot \Pr[Y_1 = \alpha + d] \\
&\leq e^{t\alpha} + \sum_{d=1}^{\infty} e^{t(\alpha+d)} \cdot \Pr[Y_1 = \alpha + d] \\
&< \sum_{d=0}^{\infty} e^{t(\alpha+d)} \cdot 2^{-d}
\end{aligned}
$$

So we have up to now:

$$
\begin{aligned}
\Pr[s_{k^*+3} - s_1 \geq m - k^*] &\leq \mathbf{E}[e^{tY_1}]^{k^*+2} \\
&< (\sum_{d=0}^{\infty} e^{t(\alpha+d)} \cdot 2^{-d})^{k^*+2},
\end{aligned}
$$

and choosing $t = \frac{\log_e 2}{2}$ and doing some algebra will yield:

$$
\begin{aligned}
&= (\sqrt{2}^{\alpha} \sum_{d=0}^{\infty} \sqrt{2}^{-d})^{k^*+2} \\
&\leq (\sqrt{2}^{\alpha+3.6})^{k^*+2} \\
&\leq \sqrt{2}^{(k^*+2)\log m-(m-k^*)+4.6(k^*+2)} \\
&< \sqrt{2}^{(5.6-c_1)(k^*+2)-(c_2-2)(c_1+\log m)}
\end{aligned}
$$

which is less than $1/m^3$ if $c_1 = 5.6$ and $c_2 = 8$.
So the probability to perform work at position s_1 and thus at each position is less than $1/m^3$. Thus LV is applied with a probability of less than $1/m^3$. The text it is applied to is supposed to have length $(k+2)\mathbf{E}[X_1] < (k+2)(\log m + 3) = \mathcal{O}(k \log m)$, and LV has complexity $\mathcal{O}(kl)$, if l is the length of the input string. Also recall that $k = \mathcal{O}(\frac{m}{\log m})$. So the average expected work for any start position s_j is

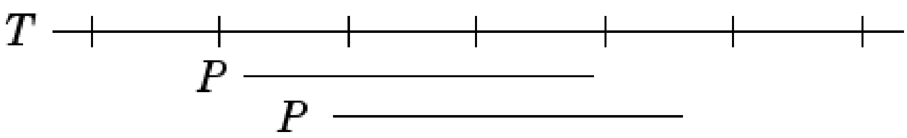
$$
\begin{aligned}
m^{-3}\mathcal{O}(k^2 \log m) &= m^{-3}\mathcal{O}(\frac{m^2}{(\log m)^2} \log m) \\
&= \mathcal{O}(\frac{1}{m \log m}) \\
&= \mathcal{O}(\lambda n.\lambda m.1)
\end{aligned}
$$

Hence the total expected work is $\mathcal{O}(n)$. $\square$

2.3.2 Sublinear Expected Time

Now an algorithm is derived from LET that is sublinear in n (when $k < k^*/2 - 3$; k^* as before).

The trick is to partition T into regions of length $\frac{m-k}{2}$. Then any substring of T that matches P must contain the whole of at least one region:



Now, starting from the left end of each region R, compute $k+1$ "maximum jumps" (using $\mathfrak{M}$, as in LV), say ending at position p. If p is within R, there can be no match containing the whole of R. If p is beyond R, apply LV to a stretch of text beginning $\frac{m+3k}{2}$ letters to the left of R and ending at p.
A variation of the proof for LET yields that

$$\Pr[p \text{ is beyond } R] < 1/m^3.$$

So, similarly to the analysis of LET, the total expected work is:

$$
\begin{aligned}
m^{-3} \underbrace{\frac{2n}{m-k}}_{\sharp \text{ regions}} \underbrace{[(k+1)(\log m + \mathcal{O}(1)) + \mathcal{O}(m)]}_{\text{exp. work at region examined}} = \ldots &= \mathcal{O}(n/m^3) \\
&= o(n)
\end{aligned}
$$

$\square$

To understand what an asset the algorithm is, consider the following facts:
A combination of LET (for $k \geq k^*/2 - 3$) and SET (for $k < k^*/2 - 3$) runs in $\mathcal{O}(\frac{n}{m} k \log m)$ expected time. In a 16-letter alphabet, k^* may be up to 25% of m, in a 64-letter alphabet even 35%.

2.4 Conclusion

> Gut gekaut ist halb verdaut.
>
> *German proverb*

The problem we have dwelt upon over the last few pages demonstrates in a beautiful manner how important a thorough preprocessing is in the business of algorithm design: Having gathered all the ingredients (data structures and auxiliary algorithms), merging them into the core algorithm was (although not obvious) quite short a task. – But that has been common knowledge for centuries: “A good chewing is half the digestion,” as goes the translation of the above saying.

Chapter 3

Approximate string indexing: comments to slideshow

Alexander Vahitov

Using simple mathematical arguments the matching probabilities in the suffix tree are bound and by a clever division of the search pattern sub-linear time is achieved.

The report is based on the article of G. Navarro and R. Baeza-Yates 'A Hybrid Indexing Method For Approximate String Matching'

3.1 Introduction

First Section 3.2 is about the task of the algorithm with some simple examples. Next Section 3.3 will tell you some basic ideas and algortihms used in the main resulting algorithm of the report, which is presented in Section 3.4. Then comes Section 3.5 where you will find some ideas to prove the average-case complexity of the algorithm (full complexity analysis can be found in the issue of G. Navarro and R. Baeza-Yates). The last part of my report is Section 3.6 with conclusions and future directions of scientific work in this field.

3.2 Our Task

We have a long text T and a short pattern (P). Our task is to find substrings from T which match our pattern approximately. Approximate matching means that we can accept some errors during the searching process (you can imagine that these are transportation errors). These errors are differences between the pattern and the founded text piece (*ocuurence*). We define 3 kinds of differences between strings: *insertion*, *replacement* and *deletion*.

If we have a pattern *abc* and a text *adbc*, then there are some obvious examples of differences between this strings :

$$adc = a + \underline{d} + b + c \text{ - } \textit{insertion}$$
$$abd = a + b + \underline{c \to d} \text{ - } \textit{replacement}$$
$$ab = a + b + \underline{c \to \varnothing} \text{ - } \textit{deletion}$$

We call the minimum number of such changes needed to transform one string to another as *edit distance* (abbreviated *ed*) between the strings. For example, all the mentioned above strings have the edit distance equal to 1 (because we needed for transformation only 1 change).
If S can be transformed to S' with x changes, then S' can be transformed to S also with x changes (we remove deletions with insertions, and vice versa).
The resulting algorithm, which will be presented later, has to solve the approximate searching problem. There are some different approaches to this problem, and at first I will tell you some of them. The first variant is to build a suffix trie for the text and search in $O(n)$ time for the occurences. We can descend by the branches of the trie till the level where we can understand that this branch does not contain an occurence. If we use *suffix array* structure, we will free much memory (tis problem arises because suffix trees have very big memory requirements). This method of search is called Depth-First Search. We can generate a set of *viable prefixes* (possible prefixes for the pattern occurence) and search for them in our trie.
The second way is to build an on-line filtering algorithm. It can use some sort of index (for example, many algorithms are based on storing text *q-grams* - pieces of the text with length equal to some number q).But it is obvious that the number of errors in such algorithms is strongly bounded, nd it can be incompatible with many practical situations.
There is another outstanding algorithm by Myers based on the mixture of the two approaches. It uses q-grams. It divides a pattern in such pieces that their length is less than $q - k$ (where k is error-level, error number divided by pattern length). Then it generates for each text piece all the strings that can appear from the pattern when some errors are done. All this strings are searched then in the q-gram set, and the last step is merging founded piece occurences and searching for the occurence of the whole pattern.
Our algorithm at first will divide our pattern into some pieces. Then it will use approximate search to find the occurences of this pieces in the text, and then verify whether the occurence of some pattern piece can be continued to the occurence of the whole pattern.

3.3 Basic Ideas and Algoritms

3.3.1 Lemma: dividing the pattern

Dividing the pattern is useful for our algorithm. If we have two strings A and B,their edit distance $ed \leq k$ and we divide A intoj substrings, then at least one of the substrings appears in B with at most $\lfloor k/j \rfloor$ errors. This is obvious because we have to change A k times to transform it to j. Each change is applied to one of the substrings. The average number of changes per substring is $\frac{k}{j}$. So it is easy to see that there is at least one of A_i that has less than or equal to $\frac{k}{j}$ changes.

Example:

$$ed('he\_likes', 'they\_like') = 3 = k;$$
$$A_1 =' he\_', A_2 =' likes' \Rightarrow j = 2;$$
$$ed('he\_', 'they\_') = 2; ed('likes', 'like') = 1 = \lfloor \tfrac{k}{j} \rfloor.$$

3.3.2 Computing edit distance

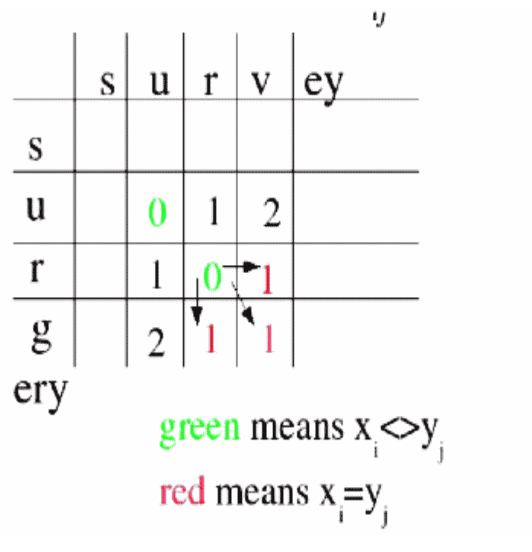
Computing edit distance is also useful for approximate string matching. There is a classical dynamic programming algorithm solving this problem. We have 2 strings: x and y with characters x_i and y_j. Let's consider C_{ij} as edit distance between $x_1..x_i$

and $y_1..y_j$. Let's fill the matrix of C_{ij} with such algorithm: $C_{i,0} = i$ because you need i insertions to the empty string to change it to $x_1..x_i$. And also $C_{0,j} = j$ by the same cause. $C_{ij} = C_{i-1,j-1}$ if $x_i = y_j$. Another case is when $x_i \neq y_j$. You need one deletion of the character y_j from the string $y_1...y_j$ to make it matching $x_1...x_i$ with $C_{i,j-1}$ errors. Also you may delete x_i to make C_{ij} equal to $C_{i-1,j+1}$. And you can replace x_i with y_j to make C_{ij} equal to $C_{i-1,j-1}$. So if $x_i \neq y_j$ then $ed(x_1..x_i, y_1..y_j) = 1 + min\{C_{i-1,j}; C_{i,j-1}; C_{i-1,j-1}\}$.

An example shows how does the algorithm work with $'survey'$ and $'surgery'$ strings. The cases when $x_i = y_j$ are marked with green, and other cases - with red. There are arrows from the minimal matrix element (left, upper or diagonal) which summed with 1 gives us C_{ij} to make it matching $x_1..x_i$ with $C_{i,j-1}$ errors.

$$x = x_1x_2\dots x_m; y = y_1y_2\dots y_n; x_p, y_q \in \Sigma$$
$$C_{ij} = ed(x_1\dots x_i, y_1\dots y_j);$$
$$(C) \text{ is a matrix filled with } C_{ij}$$
$$C_{0,j} = j; C_{i,0} = i;$$

$$C_{i,j} = \begin{cases} C_{i-1,j-1} & x_i = y_j \\ 1 + \min\{C_{i-1,j}, C_{i-1,j-1}, Ci, j-1\} & \text{else;} \end{cases}$$



Now let's consider y as the text and x as the pattern. Let's construct the algorithm to search the substrings from the text, approximately matching the pattern. The only different between this algorithm and previous one is that we initialize $C_{0,j}$ with 0 instead of j because the pattern matching process can start from every text position.

These are fullfilled matrices by the algorithm that simply computed the edit distance and the algorithm that searched the pattern in the text.

		s	u	r	g	e	r	y
	0	1	2	3	4	5	6	7
s	1	0	1	2	3	4	5	6
u	2	1	0	1	2	3	4	5
r	3	2	1	0	1	2	3	4
v	4	3	2	1	1	2	3	4
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

3.3.3 NFA construction

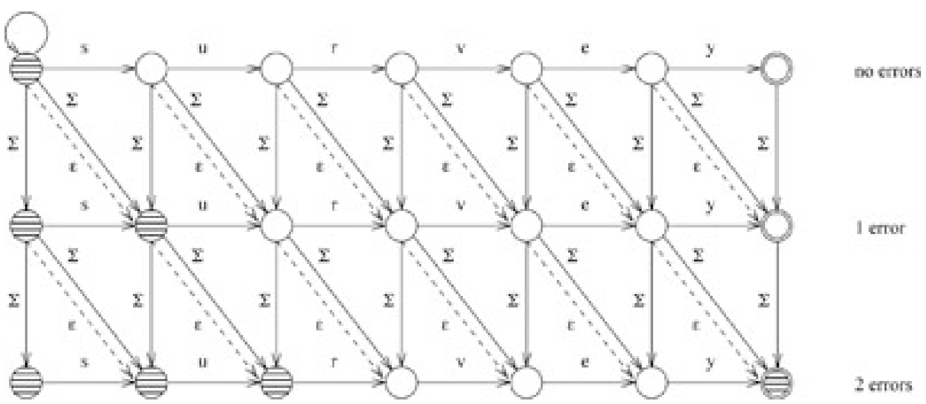
Here is presented *Nondeterministic Finite Automaton*. We will use it to search for approximate matches of pattern P in the text T. The initial state of the automaton corresponds to 0 errors in matching process and to the first character of the pattern. The automaton will go by pattern and text characters simultaneously. When it reaches the last pattern character, it means that we have found an occurence. It is presented lower in the picture as a table. In columns pattern characters are written, and each row is used to represent errors in matching. We search for pattern occurence accepting some errors, and transition between rows means accepting one error.
There are 4 kinds of transitions between states (we make changes with pattern and the text is remaining the same):

- if current text and pattern characters are the same;
- if current pattern character is replaced with the text one;
- if current text character is inserted to the pattern;
- if current pattern character is deleted.

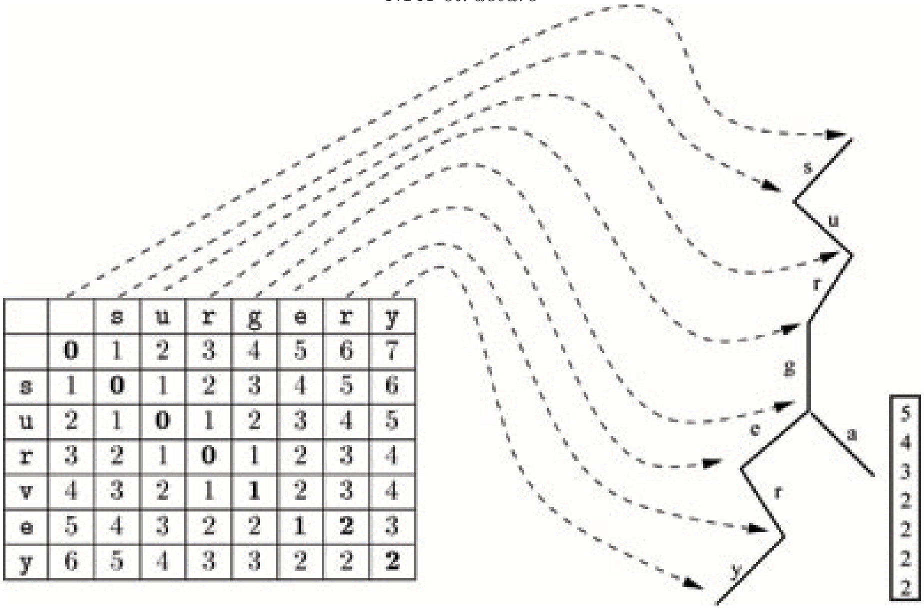
Look at the illustration of the automaton which searches the text for approximate matching the pattern *'survey'* with at most 2 errors. The rows correspond to the errors which are already made, and the columns correspond to the characters of the pattern already reached by the automaton.
The transitions are:

- horizontal, *if current text and pattern characters are the same*;
- solid diagonal, *if current pattern character is replaced with the text one*;
- vertical, *if current text character is inserted to the pattern*;
- dashed diagonal, *if current pattern character is deleted*.

The automaton finish states are the right ones.



NFA structure



		s	u	r	g	e	r	y
	0	1	2	3	4	5	6	7
s	1	0	1	2	3	4	5	6
u	2	1	0	1	2	3	4	5
r	3	2	1	0	1	2	3	4
v	4	3	2	1	1	2	3	4
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

How NFA searches 'surga' in 'surgery'

3.3.4 Depth-First Search technique

And the last is the algorithm of the depth-first search. Here we define the $U_k(P)$ which is a set of the strings, matching to P with at most k errors. It is possible to search this string in the text, but the complexity of this search is quite large. We can use a suffix tree and search in it the strings from $U_k^t(P)$. These are the neighborhood elements which are not prefixes of other neighborhood elements.

3.4 Main algorithm

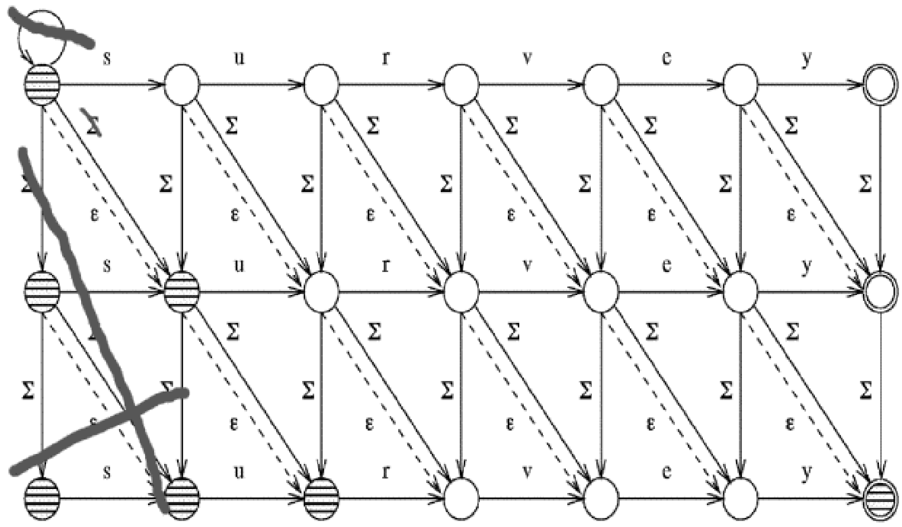
Our algorithm, searching in the suffix tree, has to start from the root, consider some string x incrementally,determine when $ed(x,P) \leq k$ and determine when adding of any character makes the ed greater than k.

In the picture you can see how the algorithm works with the input of the suffixes from the suffix tree. It fills the matrix like in the example above and analyzes it's elements.The algorithm increments x and updates the last column in O(m) time. When the element in the last row (last column element) is $\leq k$, the match is detected. Otherwise, if all the values in the column are $\geq k$, the match cannot be detected.

This is the illustration of the algorithm working in the suffix tree with 2 suffixes, matching the pattern $'surgery'$. It is shown the column of a matrix for the suffix $'surga'$.

The cost of the suffix tree search is exponential in m and k, so it's better to perform j searches of patterns of length $[\frac{m}{j}]$ and $\frac{k}{j}$ errors (remember the lemma about dividing!). That's why we divide patterns. So, we divide our pattern into j pieces and search them using the above algorithm. Then, for each match found ending at text position i we check the text area $[i-m-k..i+m+k]$. But the larger j, the more text positions need to be verified, and the optimal j will be found soon.

Now we have to adapt our NFA for searching in the suffix tree. At first, we'll search for matching from the beginning ot the suffix, so we don't need *initial self-loop*. Second, we don't need *initial insertions* to the pattern - because if te suffix matches with such insertions, we will find the suffix matching without these insertions. So, we remove the down-left triangle of the automaton, under diagonal. And at last, we can start match with $k + 1$ first characters of the pattern (because we removed initial insertions, only initial deletions remain, initial replacement is the same as initial deletion).



This is the illustration to the changes of our NFA from the previous example.

3.4.1 Suffix Arrays

Here I can say something about using suffix arrays instead of suffix trees. Suffix arrays have less space requirements, but the time complexity of the search in the case of suffix array should be multiplied by $\log n$. Suffix array replaces nodes with intervals and traversing to the node is going to the interval. If there is a node and it's children, then the node interval contains children intervals. More information on suffix arrays was in Olga Sergeeva's report.

3.5 Complexity Analysis

Let's analyse the algorithm to determine it's complexity and the best variant of partitioning the pattern. Now we'll find the average number of nodes at level l. If we are working with random text, then the number of suffixes in level l is σ^l, and for small l the number of suffixes longer than l is nearly n. In the probability model of n balls thrown into σ^l urns we find that the average number of filled urnes is $\theta(\min\{\sigma^l, n\})$. If $l \leq m'$, at least $l - k$ text characters must match the pattern, and if $l > m'$, at least $m' - k$ pattern characters must match the text. There is no difference, which exactly is the length of the pattern prefix. So, we sum all the probabilities for different pattern prefix lengths:

$$\sum_{m'=l-k}^{l} \frac{1}{\sigma^{l-k}} C_l^{l-k} C_{m'}^{l-k} + \sum_{m'=l+1}^{l+k} \frac{1}{\sigma^{m'-k}} C_{m'-k}^{l} C_{m'}^{m'-k}$$

In the first sum the largest term is first one: $\frac{1}{\sigma^{l-k}}C_l^k$, and we can bound the whole sum with $(l-k)\frac{1}{\sigma^{l-k}}C_l^k$. By Stirling's approximation we have

$$C_l^k = \left(\frac{e^l\sqrt{2\pi l}}{k^k(l-k)^{l-k}\sqrt{2\pi k}\sqrt{2\pi(l-k)}}\right)^2\left(1+O(\frac{1}{l})\right)$$

And the whole first sum is $(l-k)\gamma(\beta)^l O(\frac{1}{l})$, where

$$\gamma(\beta) = \frac{1}{\sigma^{1-x}x^{2x}(1-x)^{2(1-x)}}$$

Here you see that $(l-k)\gamma(\beta)^l O(\frac{1}{l}) = O(\gamma(\beta)^l)$. The first sum exponentially decreases when $\gamma(\beta) < 1$, it means that:

$$\sigma > \left(\frac{1}{\beta^{2\beta}(1-\beta)^{2(1-\beta)}}\right)^{\frac{1}{1-\beta}} = \frac{1}{\beta^{\frac{2\beta}{1-\beta}}(1-\beta)^2} > \frac{e^2}{(1-\beta)^2} \Leftrightarrow \beta < 1 - \frac{e}{\sqrt{\sigma}},$$

because $e^{-1} < \beta^{\frac{\beta}{1-\beta}}$ if $\beta \in [0,1]$. The second summation can be also bounded with $O(\gamma(\beta)^l)$, and the probability of processing a given node at depth l is $O(\gamma(\beta)^l)$. In practice, e should be replaced by $c = 1.09$ (founded experimentally) because we have found only upper bound for the probability, but not the exact upper bound.
Using the formulas bounding the probability of matching, let's consider that in levels

$$l \le L(k) = \frac{k}{1-\frac{c}{\sqrt{\sigma}}} = O(k)$$

all the nodes are visited, and in levels $l > L(k)$ nodes are visited with probability $O(\gamma(\frac{k}{l})^l)$. Remember that the average number of visited nodes at the level l (for small l) is $\theta(\min\{n, \sigma^l\})$.
Now we will speak about searching of a single pattern in the text using our automaton with depth-first search technique. We can define three cases of analysis of this search process:

- $L(k) \ge \log_\sigma n$, $n \le \sigma^{L(k)}$ - 'small n', online search is preferable and no index is needed (since the total work is n); It shows that the indexing technique does not work for very small texts.
- $m + k < \log_\sigma n, n > \sigma^{m+k}$ - 'large n', the total search cost is

  $$\sigma^{L(k)} + \frac{\sigma^k(1+\beta)^{2(m+k)}}{\beta^{2k}},$$

  independent of n;
- $L(k) < \log_\sigma n \le m + k$, 'intermediate n', the search is sublinear of n in time if error level $\beta < 1 - \frac{e}{\sqrt{\sigma}}$.

Now let's add to our analysis the pattern partitioning mechanism. Remember that j here is a number of pieces in which pattern is divided. After dividing, the mechanism analised above is used. With pattern partitioning,

- First case conditions are:

  $$\frac{k}{1-\frac{c}{\sqrt{\sigma}}} \ge j\log_\sigma n, n \le \sigma^{L(\frac{k}{j})},$$

  complexity is $O(n)$.

- Here $m + k < j\log_\sigma n, n > \sigma^{\frac{m+k}{j}}$, if $\beta = \frac{k}{l} < 1 - \frac{e}{\sqrt{\sigma}}$ the complexity is $O(n^{1-\log_\sigma \frac{1}{\gamma(1+\beta)}})$. This is sublinear of n, we use $j = \frac{m+k}{\log_\sigma n}$. This j is simply the smallest of possible j's in this case (with j less than $\frac{m+k}{\log_\sigma n}$ we get into the first case).
- Third case has it's own j for the minimum of complexity, but it can get over the bounds of this case, so in most cases we can simply use such j as in second case, and we will get sublinear of n time complexity.

3.6 Conclusions

- The splitting technique balances between traversing too many nodes of the suffix tree and verifying too many text positions
- The resulting index has sublinear retrieval time $O(n^\lambda), 0 < \lambda < 1$ if the error level is moderate.
- In future there can appear more exact algorithms to determine the correct number of pieces in which the pattern is divided and there are (and may appear in future) some better algorithms for verifying after matching a piece of pattern.

Chapter 4

Compressed Suffix Arrays

Fabian Pache

In this work I present Compressed Suffix Arrays from A to Z, starting with ordinary Suffix Arrays, covering Compressed Suffix Arrays as described in [GV00] by Grossi and Vitter in-depth and finishing with an outline of further improvements on Compressed Suffix Arrays developed by K. Sadakane described in [Sad00]

4.1 Introduction

Merely having a certain text usually is not very satisfying. Before long one wants to find the occurences of a smaller text within the larger text. This is called an *enumerative* query. If we are interested only in the number of occurences it is an *counting* query, while *existential* queries only return if there is at least one occurence of the subtext. The terms *larger text* and *smaller text* can be interpreted very liberally. While one of the obvious applications would be using something like this paper as the larger text and for instance 'Suffix Array' as the smaller text there are other applications. The human genome can be seen as a text, admittedly one with a rather small alphabet, with any subsequence being a word or rather a *pattern*.

4.2 Suffix Arrays

The entire idea of a suffix array is to find a certain pattern P within a text T as fast as possible, using as little additional space as possible. A suffix array SA for a text T has as many entries as T has characters. Each entry i of the suffix array points to the position of the i-smallest suffix of T. 'Smallest suffix' in this case refers to the lexicographic ordering of all suffices of T. The order in turn is defined by the alphabet Σ which contains all characters of T. Note that the suffix array is created independant of the pattern P or the length of the pattern. Therefore a suffix array can be used for multiple sequential queries using different patterns efficiently. Searching only once for a single instance of a certain pattern can be done more efficiently using other algorithms. Suffix arrays excel for multiple queries on a static text.

4.2.1 Algorithms

A suffix array in its basic form provides no more than the information where the smallest suffix, second smallest suffix, third smallest suffix and so on starts. It does

not, in itself, provide a function that given a certain pattern, returns the position of the pattern in the text.
However such an alogrithm is quickly outlined once one remembers that the sought pattern is a subset the text. Each subset can in turn be seen as a prefix of a suffix. This is where the suffix array comes in. Each and every suffix is referenced exactly once by the corresponding suffix array. The suffix array contains all suffixes in their lexicographic order therefore all entries of the suffix array pointing to occurences of the pattern are in an unbroken sequence. Note that the entries in the suffix array are sorted, the suffixes in turn usually are not.
Since the occurences of P are in sequence, finding all occurences of P boils down to finding the first and the last occurence. Everything in between matches the pattern as well.

4.2.2 Complexity

Since the only intrinsic function of a suffix array is *lookup*(i), returning the i-smallest suffix in T by table lookup, time complexity for one operation is $O(1)$. Space consumption at this point is considered $O(n)$ for both the text and the suffix array

4.3 Compressed Suffix Arrays

Considering that every text will be stored binary in a digital environment it seems prudent to reduce the alphabet to binary as well. This also gives the highest possible degree of freedom for pattern seeking operations. But it introduces a problem in the size analysis of the text and its accompanying suffix array. A text of n characters can be seen as a bit sequence of $O(m)$ bits where $m = n \log_2 n$. But for each bit, a entry in the suffix array is required and each entry must be able to adress one the $O(m)$ suffixes. Therefore the suffix array 'grows' to $O(m \cdot \log m)$ bits as $\log m$ bits are required to uniquely address one of m entries.
Grossi and Vitter pointed out a clever way of reducing the space requirement back to $O(m)$ without incuring a great speed penalty.

4.3.1 Compression of the Suffix Array

The basic idea of Grossi and Vitter is a recursive divide and conquer algorithm. For each step of the recursion, half the entries of the suffix array are retained for the next step while the other half are stored implicitly.
For now I will only describe the compression of the suffix array itself. The observant reader might note that additional data is required to recover the implictly stored data. Compression of this information is not trivial and will be covered later in section 4.3.2.

Remember that each suffix of T, and therefore each index of $1, \ldots, m$ is refered to only once in SA, the suffix array can be interpreted as a permutation. It follows that the initial steps are applicable to all permutations. However section 4.3.2 will show that it is not a generic method that can be applied to all permutations.
As mentioned before, we will compress the suffix array recursively. In each step of the recursion we remove half the entries. The original suffix array is stored at level $k = 0$ and the recursion is applied often enough so that the suffix array shrinks back to m bits. Since the size of the initial suffix array SA_0 was m entries of $\log m$ bits and the number of bits for each entry is assumed constant Grossi and Vitter reduce the

number of explicitly stored entries. The number of levels K required therefore is

$$\begin{aligned} \frac{m}{2^K} \cdot \log m &= m \\ \log m &= 2^K \\ &\Rightarrow \\ K &= \log \log m \end{aligned}$$

Each step of the recursion removes the odd values and keeps the even values of SA_k. For reasons discussed in section 4.3.2 the kept entries are divided by two in each step. As a side effect of this division the new array SA_{k+1} again contains odd and even values, so the recursion can be applied again.

How to recover the elements removed from SA_k? For now, do not consider the required space, only keep in mind that we want to retain constain time access for each level. Therefore Grossi and Vitter introduce 3 new arrays on each level:

1. A bit vector B_k where

$$B_k[i] = \begin{cases} 1 & \text{if } SA_k[i] \text{ is even} \\ 0 & \text{if } SA_k[i] \text{ is odd} \end{cases}$$

2. A integer mapping ψ_k that is only required to be defined for all i where $B_k[i] = 0$ i.e the entry *will not* be kept in the recursion. The mapping $j = \psi_k[i]$ is used to find the entry $SA_k[j]$ that is one less than $SA_k[i]$. In other words

$$SA_k[i] = SA_k[\psi_k(i)] + 1 \text{ iff } B_k[i] = 0$$

3. A integer vector $rank_k$ where $rank_k[i]$ contains the number of 1s in $B[0..i]$. Like ψ this array is not required to be defined for all entries, but only for those where $B_k[i] = 1$ i.e the entry *will* be kept in the recursion. This array denotes the position of the halfed entry in the next level.

Using these arrays, it is possible to recover SA_k, using SA_{k+1}, B_k, $rank_k$ and ψ_k as follows:

$$SA_k(i) = \begin{cases} 2 \cdot SA_{k+1}[rank_k[i]] & \text{iff } B_k[i] = 1 \\ 2 \cdot SA_{k+1}[rank_k[\psi_k[i]]] + 1 & \text{iff } B_k[i] = 0 \end{cases}$$

The evaluation of $rank_k$ in the second statement is possible since $B_k[\psi[i]] = 1$ iff $B_k[i] = 0$

Grossi and Vitter combine the above formulas by filling the unneccessary entries ($rank_k[i]$ for $B_k[i] = 0$ and ψ_k for $B_k[i] = 1$) with neutral operations and are therefore able to put both cases in a single statement. While this is mathematically very sophisticated it is not a representation that makes the compression and reconstruction scheme easier to understand. Considering modern CPU architectures that have a operations pipeline that can reach its full potential primarily on code that is executed without conditions a unified statement might have advantages in the implementation. Please refer to the original work of Grossi and Vitter [GV00] for details.

This scheme is obviously applied only until the last level of the compression is reached. At this point the values of SA are stored explicitly. Figure 4.1 shows a suffix array for a binary text of length 32. Therefore there are levels $k = 0 \ldots \lceil \log \log 32 \rceil = 0 \ldots 3$. Entries of *rank* and *psi* that are not required in the decompression are left blank.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
T	a	b	b	a	b	b	a	b	b	a	b	b	a	b	a	a	a	b	a	b	a	b	b	a	b	b	b	a	b	b	a	#
SA_0	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14	30	12	18	27	9	6	3	20	23	29	11	26	8	5	2	22	25
B_0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	1	1	0
$rank_0$		1					2	3		4			5	6	7	8	9	10			11		12				13	14		15	16	
ψ_0	2		14	15	18	23			28		30	31							7	8		10		13	16	17			21			27

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SA_1	8	14	5	2	12	16	7	15	6	9	3	10	13	4	1	11
B_1	1	1	0	1	1	1	0	0	1	0	0	1	0	1	0	0
$rank_1$	1	2		3	4	5			6			7		8		
ψ_1			9				1	6		12	14		2		4	5

	1	2	3	4	5	6	7	8
SA_2	4	7	1	6	8	3	5	2
B_2	1	0	0	1	1	0	0	1
$rank_2$	1			2	3			4
ψ_2		5	8			1	4	

	1	2	3	4
SA_3	2	3	4	1

Figure 4.1: Example of a compressed suffix array

4.3.2 Compression of the Auxiliary Arrays

Meanwhile we compressed the suffix array down to an acceptable size of m bits. On the other hand we optained 3 new array of which only B_k can be stored in m bits.

$rank_k$ is larger by far, requiring $m \log m$ bits on each level. But there is no need to store $rank_k$ explicitly as Guy Jacobson developed a method described in the thesis paper for [Jac89] that can be based on B_k and requires only $o(m)$ bits. The basic idea is to store a two level dictionary that allows for constant access time in the cost model used here.

Compression of ψ_k is more involved than compressing $rank_k$. This is also the point where the compression becomes inapplicable to ordinary permutations. For each level $k \in \{0 \ldots K-1\}$we create 2^{k+1} list. Each of the lists is labeled with a unique binary string of length $k+1$. For each entry $\psi_k(i)$ with $B_k(i) = 0$ we determine the array to store the entry in by looking up a substring t of T. t is defined as a prefix of a suffix in T. The suffix is the one pointed to by the corresponding entry in SA_k. The length of the prefix is determined by the current level of recursion. $t(i) = T((2^k \cdot SA[i]) \ldots (2^k \cdot SA[i] + 2^k - 1))$. We append $j = psi_k(i)$ to the list labeled $t(i)$.

Continuing the example of figure 4.1 the lists are shown in figure 4.2. Note that each of the lists is sorted and the maximum entry in Level k is $\frac{m}{2^k}$ due to the division by two of each entry in SA on each level of the recursion. Thus each of the 2^{k+1} lists can be stored using a bit-vector of length $\frac{m}{2^k}$. The space requirement (without assisting structures to optimize access) is therefore

$$2^{k+1} \cdot \frac{m}{2^k} = O(m)$$

In order to access the compressed entry $j = \psi_k(i)$ we have to determine the number of 0s preceeding entry i in B_k as this is the index to the concatenated lists for level k. This can be done by calculating $h = i - rank_k(i)$ using the technique based on [Jac89] for $rank$ outlined above. Finding the hth entry in the lexicographically ordered lists is not described in detail in [GV00], but Grossi and Vitter claim constant access time while using no more than $O(m)$ bits for access optimzing structures

Level 0:

a list:	2	14	15	18	23	28	30	31
b list:	7	8	10	13	16	17	21	27

Level 1:

aa list:				
ab list:	9			
ba list:	1	6	12	14
bb list:	2	4	5	

Level 2:

$aaaa$ list:		$aaab$ list:		
$aaba$ list:		$aabb$ list:		
$abaa$ list:		$abab$ list:		
$abba$ list:		$abbb$ list:	5	8
$baaa$ list:		$baab$ list:		
$baba$ list:	1	$babb$ list:	4	
$bbaa$ list:		$bbab$ list:		
$bbba$ list:		$bbbb$ list:		

Figure 4.2: Lists for ψ of figure 4.1

4.3.3 Complexity in general

On each level $0 \leq k \leq \log\log m$ we store B_k, $rank_k$ and the tables for psi_k in $O(m)$ bits. Therefore the entire structure can be stored in $m \log\log m$ bits. Since time is constant for each level $SA(i)$ can be accessed in $O(\log\log m)$.

4.3.4 Complexity optimization

With a small penalty in time requirement it is possible to further reduce the size of the structure to $O(m)$ bits. This is done by using only a subset of the levels. As long as the size of the subset is constant the asymptotic size remains at $O(m)$. Level $k = \log\log m$ is always stored explicitly. Of the other array it suffices to store $1 \leq L < \log\log m$ levels with minor modifications if these L levels are $l(j) = \left\lceil \frac{j}{L} \log\log m \right\rceil$ for $0 \leq j \leq L-1$. To be able to reconstruct $SA_{l(j+1)}$ from $SA_{l(j)}$ it is neccessary modify $B_{l(j)}(i)$ to be 1 only if $SA_{l(j)}(i)$ can be found in $SA_{l(j+1)}(rank_{l(j)}(i))$ instead of $SA_{l(j)+1}(rank_{l(j)}(i))$. Note that $rank$ needs not be modified if based solely on B.

$\psi_{l(j)}$ still has to be defined for all entries where $B_{l(j)} = 0$, only now there are more then half of the entries defined.

This modifactions enable us to use ψ to seek an entry of $B = 1$. The length of the seeking process in turn determines the time requirements. The process is bounded by longest possible sequence that has to be investigated. For a compressed suffix array with L levels this sequence can be no longer than the sequence required to move from level $l(0)$ to $l(1)$. Since all upper bound of sequences on the same level are of equal length, the longest seeking process is

$$\begin{aligned}
\sum_{j=0}^{L-1} \frac{\|l(j)\|}{\|l(j+1)\|} &= \sum_{j=0}^{L-1} \frac{\frac{m}{2^{l(j)}}}{\frac{m}{2^{l(j+1)}}} \\
&= \sum_{j=0}^{L-1} \frac{2^{l(j+1)}}{2^{l(j)}} \\
&= \sum_{j=0}^{L-1} \frac{2^{\frac{j+1}{L} \log\log m}}{2^{\frac{j}{L} \log\log m}} \\
&= \sum_{j=0}^{L-1} \frac{(2^{\log\log m})^{\frac{j+1}{L}}}{(2^{\log\log m})^{\frac{j}{L}}} \\
&= \left(2^{\log\log m}\right)^{\frac{1}{L}} \sum_{j=0}^{L-1} 1 \\
&= (\log m)^{\frac{1}{L}} \cdot L \\
&= O(\log^{\epsilon} m) \text{ with } \epsilon = \frac{1}{L}
\end{aligned}$$

4.4 Extensions of Compressed Suffix Arrays

Leaving the most general case of suffix arrays on binary texts, Sadakane proposes some extensions of suffix arrays on human readable texts. For a more in-depth description of Sadakanes datastructures and algorithms please refer to [Sad00]. Here I can only give a short summary of his work and what it might imply.

4.4.1 Operations

While the original suffix array offers no more functionality than a mere lexicagraphic ordering new operations are included.

Inverse Suffix Array A suffix array answers the question ‘At which position i does the j-smallest suffix begin?’. Or more concise $i = SA[j]$ To answer the inverse question ‘What is the order j of the suffix starting at position i?’($j = SA^{-1}[i]$) we have no methods available up to now. Sadakane proposes a structure that contains SA^{-1} that has the same time and space requirements as SA.

Searching There are algorithms that allow for searching operations in suffix arrays. Sadakane augments the structure so that searching is an intrinsic feature of the suffix array.

Decompression Using only the suffix array and the introduced extensions but without the original text, recover a substring defined by first and last index in the original text.

The last two functions are based on something Sadakane calls ‘the inverse of the array of cumulative frequencies’. The abstract does not elaborate enough on the subject to make a further description possible in this paper.

4.4.2 Complexity

Obviously Sadakane requires more memory to store the suffix array itself, but on the other hand removes the need to store the text. His claim is that the entire search index can be reduced to below the size of the original text for certain texts.

4.5 Conclusion

We started out with a large text and an even larger suffix array. In a first step Grossi and Vitter drastically reduced the size of the suffix array without unreasonable penalties in usability. Sadakane took this process even further and removed the neccessity for storing the text, at the same time adding to the functionality of the datastructure. Meanwhile we are storing neither the text nor the suffix arrays at all. All we retained is a set of hints and literally pointers to both the text and the suffix array. This is a fascinating evolution of a once seeming unchangable structure.

Chapter 5

Asymptotic Properties of Suffix Trees.

Ivan Kazmenko

Unlike the previous chapters, this one is not going to introduce a new sophisticated suffix tree construction algorithm, dig into its properties and prove that it works fast and fine. Instead, we'll consider one of the most dumb algorithms of suffix tree construction and find out that under certain conditions on the text, it almost surely works rather well, meaning that we can find almost sure upper and lower bound for the complexity of new suffix insertion while the size of the text tends to infinity.

This chapter is based on the article *Asymptotic Properties of Data Compression And Suffix Trees* by Wojciech Szpankowski [Szp93] and the book *Average case analysis of algorithms on sequences* [Szp00] of the same author.

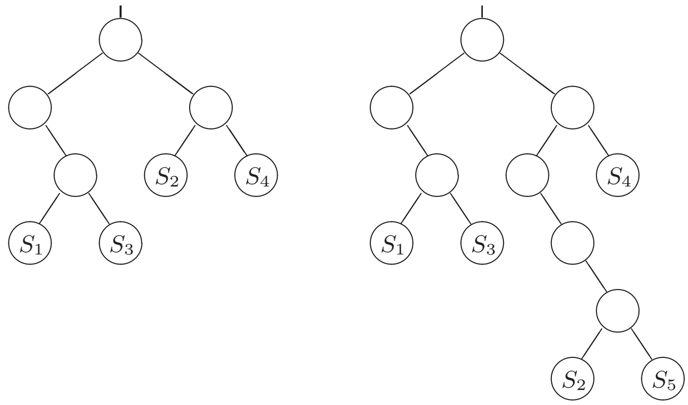
5.1 Suffix Tree Construction

Let's start with some common symbol definitions which we will use in the entire chapter.
Let Σ is a finite alphabet of size $|\Sigma| = V$, $\{X_k\}_{k=1}^{\infty}$ be a stationary ergodic sequence of symbols generated from Σ, and $X_m^n = (X_m, ..., X_n)$ for $m < n$ be a partial sequence of the whole sequence $\{X_k\}_{k=1}^{\infty}$.
We shall now consider a very simple algorithm of suffix tree construction. A node of our tree can be either internal, i. e. branching node, or external node storing one of the suffixes $S_i = \{X_k\}_{k=i}^{\infty}$. Each edge is labeled by some symbol from Σ. When adding a suffix, we start from the root of our tree and try to 'align' the suffix to the tree, that is, move by the edge corresponding to the current symbol of our suffix and change the current symbol to the next one in the suffix. That procedure continues until we find no such edge at the vertex we are currently in. We then add that edge and create a new vertex at its end storing the suffix we were adding.
More formally, consider a digital tree built in the following way:
Step 0. At the beginning, the tree consists of its root only.
Step 1. Consider a tree $\mathcal{T}_n$ built for the partial sequence $X_1^n = (X_1, ..., X_n)$.
Step 2. Set current vertex to root.
Step 3. Starting with $j = n + 1$, we either

(A) move by the edge marked by X_j from the current vertex if it exists thus changing the current vertex and increase j by 1, or
(B) construct a new edge marked with symbol X_j from the current vertex to a new vertex marked with our suffix X_{n+1}^{∞} and proceed to *Step 1* with n increased by 1 otherwise.
Note that $j - n$ is the number of case *(A)* occurences during a single *Step 3*.
The picture shows an example of a single loop of our algorithm.

Let $X_1^{10} = (0, 1, 0, 1, 1, 0, 1, 1, 1, 0)$.

Four inserted suffixes. Fifth suffix insertion.



$S_1 = 0101101110$
$S_2 = 101101110$
$S_3 = 01101110$
$S_4 = 1101110$
$S_5 = 101110$

We do not formalize our 'splitting policy', that is, the way how we split an external node that becomes internal during some other suffix insertion. The natural way to do the 'splitting' is shown on the picture. We can consider all previous suffix marks to be infinite branches of our tree to make the algorithm formally correct.
We are interested in the complexity of a single loop of our algorithm. Formally, our main questions regarding the algorithm described will be the following:
What is the typical height of $\mathcal{T}_n$?
What is the typical difference $j - n$ when Step 3 is finished?
What is the typical minimal possible difference $j - n$ at the end of Step 3 for the tree $\mathcal{T}_n$?
In the next section, we will present some assumptions on the sequence $\{X_k\}_{k=1}^{\infty}$ that, being not too restrictive, will get us some bounds on the value in question.

5.2 Depth of Insertion in a Suffix Tree

As we study our sequence $\{X_k\}_{k=1}^{\infty}$ in a probabilistic framework, its most important characteristic is nth order probability distribution $P(X_1^n) = Pr\{X_k = x_k, 1 \leqslant k \leqslant n, x_k \in \Sigma\}$. The entropy of our sequence is the limit $h = \lim\limits_{n\to\infty} \frac{E\{-\log P(X_1^n)\}}{n}$. It is known that $h \leqslant \log V$. All logarithms are natural ones in this chapter.
Another characteristic of much interest is the parameter L_n which is the smallest integer $L > 0$ such that $X_m^{m+L-1} \neq X_{n+1}^{n+L}$ for all $1 \leqslant m \leqslant n$. Informally, it has the following meaning: when we insert the suffix S_{n+1}, we will require exactly L_n steps *(A)* to do it.
Returning to our example, let $X_1^{10} = (0, 1, 0, 1, 1, 0, 1, 1, 1, 0)$. Here $L_1 = 1$, $L_2 = 3$, $L_3 = 2$, and $L_4 = 5$ since $X_5^8 = X_2^5 = (1, 0, 1, 1)$ and therefore $L_4 > 4$.

So, what will be our assumptions on the sequence $\{X_k\}$? Below we introduce the mixing condition - a weakened form of independence.
Remember that $\{X_k\}$ is called an independent sequence if for every set of indexes $I = \{i_1, \ldots, i_r\}$ the probablity of $\{X_k\}_{k\in I}$ being in $\bigotimes\{A_k\}_{k=1}^r$ is equal to the product of the corresponding probabilities: $Pr\{X_{i_1} \in A_1, \ldots, X_{i_r} \in A_r\} = Pr\{X_{i_1} \in A_1\} \ldots Pr\{X_{i_r} \in A_r\}$. Somewhat weaker is pairwise independent condition which takes only the sets I of size 2 into consideration, stating that $Pr\{X_{i_1} \in A_1, X_{i_2} \in A_2\} = Pr\{X_{i_1} \in A_1\}Pr\{X_{i_2} \in A_2\}$. The independence itself can be also written in pairwise form with events being not subsets of a single copy of Σ, but elements of a more complex σ-field.
Let F_m^n be a σ-field (also known as σ-algebra) generated by $\{X_k\}_{k=m}^n$ for $m \leqslant n$. Independence means that for every pair of events $A \in F_0^m$ and $B \in F_{m+1}^\infty$ it is true that $Pr\{AB\} = Pr\{A\}Pr\{B\}$. The mixing condition creates a gap of size d between our σ-fields so that $A \in F_0^m$ and $B \in F_{m+d}^\infty$ and transforms our equality into two inequalities bounding the left term with the right one multiplied by some constants from both sides. The strong α-mixing condition substitutes that constants by functions tending to 1 from both sides while the gap size d tends to infinity. The formal definitions follow.
We say that $\{X_k\}$ satisfies the *mixing condition* if and only if there exist constants $0 < c_1 \leqslant c_2$ and an integer d such that for all $A \in F_0^m$, $B \in F_{m+d}^\infty$ and $0 \leqslant m \leqslant m+d \leqslant n$ the following condition is true: $c_1 Pr\{A\}Pr\{B\} \leqslant Pr\{AB\} \leqslant c_2 Pr\{A\}Pr\{B\}$.
Now let α be a function of d such that $\alpha(d) \xrightarrow[d\to\infty]{} 0$. $\{X_k\}$ satisfies the *strong α-mixing condition* if and only if for all $A \in F_0^m$, $B \in F_{m+d}^\infty$ and $0 \leqslant m \leqslant m+d \leqslant n$ the following condition is true: $(1-\alpha(d))Pr\{A\}Pr\{B\} \leqslant Pr\{AB\} \leqslant (1+\alpha(d))Pr\{A\}Pr\{B\}$.
We define two new parameters of $\{X_k\}$. They are parameters h_1 and h_2:
$h_1 = \lim\limits_{n\to\infty} \frac{\max\{\log P^{-1}(X_1^n),\ P(X_1^n)>0\}}{n} = \lim\limits_{n\to\infty} \frac{\log(1/\min\{P(X_1^n),\ P(X_1^n)>0\})}{n}$,
$h_2 = \lim\limits_{n\to\infty} \frac{\log(E\{P(X_1^n)\})^{-1}}{2n} = \lim\limits_{n\to\infty} \frac{\log(\sum_{X_1^n} P^2(X_1^n))^{-1}}{2n}$.
The relationship with entropy h is as follows: $0 \leqslant h_2 \leqslant h \leqslant h_1$. The values h_1 and h_2 are also known as Rényi entropy of order $-\infty$ and 2, respectively.
The formulas are complex, so we could use a simple example, Bernoulli model, to see what these values are like.
Assume that symbols X_i are generated indepenently, and ith symbol is generated according to the probability p_i. Thus, $h = \sum\limits_{i=1}^{V} p_i \log(p_i^{-1})$, $h_1 = \log(1/p_{min})$ and $h_2 = 2\log(1/P)$ where $p_{min} = \min\limits_{1\leqslant i\leqslant V}\{p_i\}$ is the probability of least probable symbol occurence and $P = \sum\limits_{i=1}^{V} p_i^2$ can be interpreted as a probability of a match between any two symbols.
Now, we are ready to present our main result, Theorem 5.1. It proposes the conditions under which we can find almost sure lower and higher bounds for L_n, the value we are interested in. An important finding is that we not only know how it behaves (its behavior is logarithmic with respect to n), but also find the range of the constant by that logarithm.

Theorem 5.1. *Consider stationary ergodic sequence $\{X_k\}_{k=-\infty}^\infty$.*
1. Assume strong α-mixing condition.
2. Let $h_1 < \infty$ and $h_2 > 0$.
$(\ast)$ $\exists\rho : 0 < \rho < 1, \exists\beta$ such that $\alpha(d) = O(d^\beta \rho^d)$ for $d \to \infty$.
Then
(1) $\liminf\limits_{n\to\infty} \frac{L_n}{\log n} = \frac{1}{h_1}$ (a.s.),
(2) $\limsup\limits_{n\to\infty} \frac{L_n}{\log n} = \frac{1}{h_2}$ (a.s.).

How restrictive is the condition $(*)$? Many practically occuring cases fit it, for example, in Bernoulli model, $\alpha(d) = 0$ because of independence of X_k, and if the sequence $\{X_k\}$ is Markovian, $\alpha(d)$ decays exponentially fast. In general, statement (1) of Theorem 5.1 does not hold without the $(*)$ condition.

5.3 Height and Shortest Feasible Path in a Suffix Tree

In this section, we will introduce yet another bundle of auxiliary definitions to formulate our Theorem thm:kaz-2, and then prove Theorem 5.1 using Theorem 5.2. The proof of Theorem 5.2 itself will not be given due to its complexity, however, a short overview of its proof techniques will be done in Section 5.4.
Let us define some more depth characteristics. Let $\mathcal{T}_n$ be a suffix tree constructed from the first n suffixes of $\{X_k\}$. *mth depth* $L_n(m)$ is the depth of the ith suffix in $\mathcal{T}_n$; note that $L_n = L_{n+1}(n+1)$. *Average depth* D_n is the depth of a randomly selected suffix, that is, $D_n = \frac{1}{n}\sum\limits_{m=1}^{n} L_n(m)$.
Height and shortest feasible path are defined as follows. *Height* H_n is the length of the longest path in $\mathcal{T}_n$; $H_n = \max\limits_{1\leqslant m\leqslant n}\{L_n(m)\}$. *Available node* is a node which does not belong to $\mathcal{T}_n$ but its predecessor does, that is, a node that could be inserted in $\mathcal{T}_{n+1}$ at the next insertion with no other nodes added. *Shortest feasible path* s_n is the length of the shortest path from the root to an available node.
For each two suffixes, we can find their longest common prefix by walking down the tree along them till they part. *Self-alignment* $C_{i,j}$ is the length of the longest common prefix of S_i and S_j.
One can easily prove the following relations of self-alignment to other suffix tree parameters:
$L_n(m) = \max\limits_{1\leqslant k\leqslant n, k\neq m}\{C_{k,m}\} + 1$;
$H_n = \max\limits_{1\leqslant i<j\leqslant n}\{C_{i,j}\} + 1$;
$L_n = \max\limits_{1\leqslant m\leqslant n}\{C_{m,n+1}\} + 1$.
Returning to our example, let $X_1^{10} = (0,1,0,1,1,0,1,1,1,0)$. Consider suffix tree $\mathcal{T}_4$ built from first 4 suffixes. $L_4(1) = 3$, $L_4(2) = 2$, $L_4(3) = 3$, $L_4(4) = 2$. $H_4 = 3$, $s_4 = 2$. But $L_4 = L_5(5) = 5$.
Note that the S_5 node of $\mathcal{T}_5$ is not an available node in $\mathcal{T}_4$ since it requires auxiliary internal nodes to be inserted. In $\mathcal{T}_5$, $H_5 = 5$, and $s_5 = 2 = s_4$.
Digging into the properties of $C_{i,j}$ gives the proof of Theorem 5.2 formulated below. It is a variant of Theorem 5.1 with L_n substituted by s_n and H_n. As we already observed, the statement (2) of the theorem does not need $(*)$ condition to hold.

Theorem 5.2. *Consider stationary ergodic sequence $\{X_k\}_{k=1}^{\infty}$.*
1. Assume strong α-mixing condition.
2. Let $h_1 < \infty$ and $h_2 > 0$.
Then
(1) $\lim\limits_{n\to\infty}\frac{s_n}{\log n} = \frac{1}{h_1}$ *(a.s.) when $(*)$ holds,*
(2) $\lim\limits_{n\to\infty}\frac{H_n}{\log n} = \frac{1}{h_2}$ *(a.s.) when $\alpha(d)$ satisfies the following:* $\sum\limits_{d=0}^{\infty}\alpha^2(d) < \infty$.

Proof of Theorem 5.1 by Theorem 5.2: For each of the two statements, we will bound the left side of equality by the right side from both sides.
(1): $\limsup\limits_{n\to\infty}\frac{L_n}{\log n} \leqslant \lim\limits_{n\to\infty}\frac{H_n}{\log n}$ (a.s.) simply holds by definition as $L_n \leqslant H_n$; let's prove that $\limsup\limits_{n\to\infty}\frac{L_n}{\log n} \geqslant \lim\limits_{n\to\infty}\frac{H_n}{\log n}$ (a.s.). Note that H_n is a non-decreasing sequence;

$L_n = H_n$ when $H_{n+1} > H_n$, and that occurs infinitely often since $H_n \to \infty$ and $\{X_k\}$ is an ergodic sequence, so $Pr\{L_n = H_n\ i.o.\} = 1$ and there exists a subsequence $n_k \to \infty$ such that $L_{n_k} = H_{n_k}$. It is clear now that the upper limit of L_n in not less than the limit of H_n with an arbitrary common denominator, which is equal to $\log n$ in our case.
(2) can be proved in a similar way: s_n is a non-decreasing sequence also.

□

5.4 Proof Techniques

In this section, we will throw a short glance on the tools used to prove Theorem 5.2 itself. The whole proof is complex and technically hard.
One of the methods used in the proof is a technique called *String-Ruler Approach*. According to it, the correlation between different substrings is measured using another string ω called a string-ruler. To illustrate it, we shall find the longest common prefix of two independent strings $\{X_k(1)\}_{k=1}^{\infty}$ and $\{X_k(2)\}_{k=1}^{\infty}$. Let its length be $C_{1,2}$. The following equivalence is obvious:
$C_{1,2} \geqslant k \Longleftrightarrow \exists\omega$ of length k: $X_1^k(1) = \omega = X_1^k(2)$.
We then construct a set $\mathcal{W}_k = \{\omega \in \Sigma^k : |\omega| = k\}$ and estimate the probabilities $P(\omega_k) = P(X_{m+1}^{m+k} = \omega_k)$ for a fixed position m in our sequence $\{X_k\}$.
Another important method is a probabilistic one, called *Second Moment Method*. The version by Chung and Erdös of this method states that for a sequence of events A_i we have
$Pr\{\bigcup_{i=1}^{n} A_i\} \geqslant \frac{(\sum_{i=1}^{n} Pr\{A_i\})^2}{\sum_{i=1}^{n} Pr\{A_i\} + \sum_{i \neq j} Pr\{A_i \cap A_j\}}$.
We then apply it to the sets $A_{i,j} = \{C_{i,j} \geqslant k\}$.
The reasoning of the latter method is elementary. Let us remember Markov's inequality
$Pr\{X \geqslant t\} \leqslant \frac{E\{X\}}{t}$
and Chebyshev's inequality
$Pr\{|X - E\{X\}| \geqslant t\} \leqslant \frac{Var\{X\}}{t^2}$.
After some trivial calculations we get First Moment Method:
for integer-valued nonnegative random variable X
$Pr\{X > 0\} \leqslant E\{X\}$
and Second Moment Method (Chebyshev's version):
$Pr\{X = 0\} \leqslant \frac{Var\{X\}}{(E\{X\})^2}$,
respectively. The version by Chung and Erdös is derived from the latter one.

5.5 Summary

In our main result, Theorem 5.1, we have shown that, given a stationary ergodic sequence generated over a finite alphabet, under strong α-mixing condition on the sequence, the depth of the nth suffix insertion into a partial suffix tree of that sequence using simple and natural algorithm specified above can be described by the expression $c \log n$ where c almost surely lies between $1/h_1$ and $1/h_2$ and the parameters h_1 and h_2 can be found explicitly.

Chapter 6

Sequential Pattern Matching – Analysis of Knuth-Morris-Pratt Type Algorithms Using the Subadditive Ergodic Theorem

Tobias Reichl

Based on an article by Mireille Règnier and Wojciech Szpankowski this report outlines the complexity analysis of Knuth-Morris-Pratt type algorithms using the Subadditive Ergodic Theorem, Martingales and Azuma's Inequality.

Using the Subadditive Ergodic Theorem we will prove the existence of a linearity constant for worst and average case. Although the Subadditive Ergodic Theorem doesn't indicate a way to compute the linearity constant, we may use Azuma's Inequality to show that the number of comparisons done is well concentrated around its mean value.

6.1 Pattern Matching

6.1.1 Conventions

Before starting we have to introduce some conventions in nomenclature: a pattern p of length m, denoted p_1^m, is matched against a text t of length n, denoted t_1^n.
We have to define some kind of counting function:

$$M(l,k) = \begin{cases} 1 & t[l] \text{ is compared to } p[k] \\ 0 & \text{otherwise} \end{cases} .$$

A position in the text is called an *alignment position* (AP) if starting from it comparisons between text and pattern are done, or more formally

$$M(AP + (k-1), k) = 1 \qquad \text{for some k.}$$

6.1.2 Defining Sequential Algorithms

We will classify algorithms by a property we call *sequentiality*.

1. **Semi-sequential:** The sequence of alignment positions used by the algorithm is non-decreasing.
2. **Strongly semi-sequential:** (1) and the comparisons $M(l_i, k_i)$ define non-decreasing text-positions l_i.
3. **Sequential:** (1) and $M(l,k) = 1 \Rightarrow t_{l-(k-1)}^{l-1} = p_1^{k-1}$, so: text-pattern comparisons $M(l,k)$ are only done as long as there is a prefix of the pattern to the left of the text position to be compared next.
4. **Strongly sequential:** (1), (2) and (3).

6.1.3 Naive / Brute Force Algorithm

In short we may outline the *naive* or *brute force algorithm* as follows:

- Every text position is an alignment position.
- The aligned pattern is matched against the text from left to right until either a mismatch occurs or the pattern is found.
- The pattern is then shifted by one and the next matching is started.

The brute force algorithm is a sequential algorithm: the APs are non-decreasing and the condition $M(l,k) = 1 \Rightarrow t_{l-(k-1)}^{l-1} = p_1^{k-1}$ holds: no more comparisons are done after a mismatch is found, so every alignment is used only as long as prefixes of the pattern are found in the text.
The sequence of text positions l_i defined by the sequence of comparisons $M(l_i, k_i)$, however, may include 'jumping backwards', i.e. if a mismatch occurs, the AP is shifted by one and comparisons again start at the beginning of the pattern.

6.1.4 Knuth-Morris-Pratt

Idea: (Morris-Pratt) Disregard APs if we already know that there cannot be a prefix of the pattern, namely the ones that safisfy $t_{l+i}^{l+k-1} \neq p_1^{k-i}$ for all i. Or equivalently $p_{1+i}^{k} \neq p_1^{k-1}$ as the already processed text has to be identical to the corresponding prefix of the pattern.
This knowledge can be obtained by a preprocessing of the pattern. The specific shifting functions can formally be described as following:

Morris-Pratt-Variant (MP):

$$S = \min\{k-1; \min\{s > 0 : p_{1+s}^{k-(s+1)}\}\}$$

Knuth-Morris-Pratt-Variant (KMP):

$$S = \min\{k; \min\{s : p_{1+s}^{k-(s+1)} \quad \text{and} \quad p_k^k \neq p_{k-s}^{k-s}\}\}$$

MP and KMP differ in the amount of information used from the pattern. Both are strongly sequential algorithms, because from the definition of the shift function it is automatic that the sequentiality condition (2) holds, there is no 'jumping backwards'.

6.1.5 Defining Complexity

The complexity in matching a pattern p against a text portion t_r^s can be defined as the number of comparisons needed:

$$c_{r,s}(t,p) = \sum_{l\in[r,s],k\in[1,m]} M(l,k) \tag{6.1}$$

Overall complexity $c_{1,n}$ is denoted as c_n. If either the text or the pattern is a realization of a random sequence we shall write C_n.
To look at KMP we have to introduce two probabilistic tools: the Subadditive Ergodic Theorem and Azuma's Inequality.

6.2 Subadditive Ergodic Theorem

6.2.1 Fekete's Theorem

Assume a deterministic sequence $\{x_n\}_{n=0}^{\infty}$ satisfies the so called *subadditivity property*, that is

$$x_{m+n} \leq x_n + x_m \tag{6.2}$$

for all integers $m, n \geq 0$. We may fix $m \geq 0$ and write

$$n = km + l \quad \Leftrightarrow \quad \frac{k}{n} = \frac{1}{m} - \frac{l}{mn} \ . \tag{6.3}$$

Then by successive application of the subadditivity property arrive at

$$x_n = x_{km+l} \leq x_m + x_m + \cdots + x_m + x_l = kx_m + x_l \ . \tag{6.4}$$

Now dividing by n and considering $n \to \infty$ resp. $k/n \to 1/m$, cf. (6.3) we get

$$\limsup_{n\to\infty} \frac{x_n}{n} \leq \inf_{m\geq 1} \frac{x_m}{m} \leq \alpha \ . \tag{6.5}$$

To complete the derivation we may use the definition of $\liminf$ and get the following:

$$\liminf_{n\to\infty} \frac{x_n}{n} = \sup_{n\geq 0} \left\{ \inf_{k\geq n} \frac{x_n}{n} \right\} = \alpha \tag{6.6}$$

Thus we just derived the theorem of Fekete.

Theorem 6.1 (Fekete 1923). *If a sequence of real numbers satisfies the* subadditive property

$$x_{m+n} \leq x_n + x_m \tag{6.7}$$

for all integers $m, n \geq 0$, then

$$\lim_{n\to\infty} \frac{x_n}{n} = \inf_{m\geq 1} \frac{x_m}{m} \ . \tag{6.8}$$

If the subadditvity property (6.7) is replaced by the superadditvity property

$$x_{m+n} \geq x_n + x_m \tag{6.9}$$

for all integers $m, n \geq 0$, then

$$\lim_{n\to\infty} \frac{x_n}{n} = \sup_{m\geq 1} \frac{x_m}{m} \ . \tag{6.10}$$

Example 6.1 (Longest Common Subsequence). The *longest common subsequence* (LCS) problem is a special case of the *edit distance* problem. Two ergodic stationary sequences $X = X_1, X_2, \ldots, X_n$ and $Y = Y_1, Y_2, \ldots, Y_n$ are given, then let

$$L_n = max\{K : X_{i_k} = Y_{j_k} \text{ for } 1 \le k \le K, \quad \begin{array}{ll} \text{where} & 1 \le i_1 < i_2 < \cdots < i_K \le n, \\ \text{and} & 1 \le j_1 < j_2 < \cdots < j_K \le n\} \end{array}$$

be the length of the longest common subsequence. Observe that

$$L_{1,n} \ge L_{1,m} + L_{m,n} \ . \tag{6.11}$$

The LCS in the region $(1, n)$ may cross the boundary of X_1^m, Y_1^m and X_m^n, Y_m^n. Hence it may be bigger than the sum of the LCSs in each subregion $(1, m)$ and (m, n) and so $a_n = \mathbf{E}[L_{1,n}]$ is superadditive:

$$\lim_{n\to\infty} \frac{a_n}{n} = \alpha = \sup_{m\ge 1} \frac{\mathbf{E}[L_m]}{m} \ . \tag{6.12}$$

But here you can already see the cavity: Fekete's Theorem[1] only states the existence of the linearity constant, but neither tells us its value nor even how to compute it. For the LCS problem here Steele in 1982 conjectured $\alpha \approx 0.8284$.

Theorem 6.2 (DeBruijn and Erdös 1952). *The subadditivtiy property can be relaxed to include a sequence $c_n = o(n)$*

$$x_{n+m} \le x_n + x_m + c_{n+m} \tag{6.13}$$

where

$$\sum_{k=1}^{\infty} \frac{c_k}{k^2} < \infty \ . \tag{6.14}$$

Then, too

$$\lim_{n\to\infty} \frac{x_n}{n} = \inf_{m\ge 1} \frac{x_m}{m} \ . \tag{6.15}$$

6.2.2 Subadditive Ergodic Theorem

As Fekete's Theorem only applies to deterministic sequences, effort has been taken to generalize it to sequences of random variables.

Theorem 6.3 (Kingman and Liggett). *Let $X_{m,n}$ $(m < n)$ be a sequence of random variables satisfying the following properties:*

1. $X_{0,n} \le X_{0,m} + X_{m,n}$ *(subadditivity property)*
2. *For every k, $\{X_{nk,(n+1)k}, n \ge 1\}$ is a stationary sequence.*
3. *The distribution of $\{X_{m,m+k}, k \ge 1\}$ does not depend on m.*
4. $\mathbf{E}[X_{0,1}] < \infty$ *and for each n,* $\mathbf{E}[X_{0,n}] \ge c_0 n$ *where $c_0 > -\infty$.*

Then

$$\lim_{n\to\infty} \frac{\mathbf{E}[X_{0,n}]}{n} = \inf_{m\ge 1} \frac{\mathbf{E}[X_{0,m}]}{m} := \alpha \ , \tag{6.16}$$

$$\lim_{n\to\infty} \frac{X_{0,n}}{n} = X \qquad (a.s) \ . \tag{6.17}$$

[1] And the Subadditive Ergodic Theorem, as as we will see later.

Theorem 6.4 (Deriennic). *Similar to subadditivity with deterministic sequences, subaddititvity with random sequences can be relaxed to include a sequence A_n*

$$X_{0,n} \leq X_{0,m} + X_{m,n} + A_n \tag{6.18}$$

such that $\lim_{n\to\infty} \mathbf{E}[A_n/n] = 0$. *Then, too*

$$\lim_{n\to\infty} \frac{X_{0,n}}{n} = X \qquad (a.s) \ . \tag{6.19}$$

6.3 Martingales and Azuma's Inequality

6.3.1 Basic Properties of Martingales

Martingale is a standard tool in probabilistic analysis. A sequence

$$Y_n = f(X_1, X_2, \ldots, X_n), \qquad n > 0 \tag{6.20}$$

is a martingale with respect to the *filtration*

$$\mathcal{F}_n = (X_1, X_2, \ldots, X_n) \tag{6.21}$$

if for all $n \geq 0$ the following hold:

1. $\mathbf{E}[|Y_n|] < \infty$ and
2. $\mathbf{E}[Y_{n+1} \mid X_0, X_1, \ldots, X_n] = \mathbf{E}[Y_{n+1} \mid \mathcal{F}_n] = Y_n$

So $\mathbf{E}[Y_{n+1} \mid \mathcal{F}_n]$ defines a random variable depending on the *k*nowledge contained in $(X_1, X_2, \ldots, X_n)$. Now let's define the *martingale difference* as

$$D_n = Y_n - Y_{n-1} \tag{6.22}$$

so that

$$Y_n = Y_0 + \sum_{i=1}^{n} D_i \qquad \Leftrightarrow \qquad \sum_{i=1}^{n} D_i = Y_n - Y_0 \ . \tag{6.23}$$

Then we may rewrite the martingale difference as

$$D_i = Y_i - Y_{i-1} = \mathbf{E}[Y_n \mid \mathcal{F}_i] - \mathbf{E}[Y_n \mid \mathcal{F}_{i-1}] \tag{6.24}$$

This is possible as the realization of the martingale sequence Y_n depends on the knowledge contained in $\mathcal{F}_i$, so the difference between neighbouring elements depends on the difference in knowledge about X_i. Now observe:

$$\mathbf{E}[Y_n \mid \mathcal{F}_n] = Y_n \qquad \text{and} \qquad \mathbf{E}[Y_n \mid \mathcal{F}_0] = \mathbf{E}[Y_n] \ .$$

Note: $\mathcal{F}_n$ completely defines Y_n, while $\mathcal{F}_0$ contains no information about Y_n.
Interestingly we are now able to rewrite the martingale diffence sum, cf. (6.23), as

$$\sum_{i=1}^{n} D_i = Y_n - \mathbf{E}[Y_0] \ . \tag{6.25}$$

And this is what we are interested in: the deviation of Y_n from its mean value. To further assess it we will now introduce Hoeffding's Inequality.

6.3.2 Hoeffding's Inequality and Azuma's Inequality

Theorem 6.5 (Hoeffding's Inequality). *Let $\{Y_n\}_{n=0}^{\infty}$ be a martingale and let there exist a constant c_n such that*

$$|Y_n - Y_{n-1}| = |D_n| \leq c_n \tag{6.26}$$

Then

$$Pr\left\{|Y_n - Y_0| \geq x\right\} = Pr\left\{\left|\sum_{i=1}^{n} D_i\right| \geq x\right\} \leq 2\exp\left(-\frac{x^2}{2\sum_{i=1}^{n} c_i^2}\right) \quad . \tag{6.27}$$

By now, we know how to use the martingale difference sum $\sum_{i=1}^{n} D_i$ for assessing the deviation from the mean. We also know how to assess this martingale difference sum, provided D_i is bounded.
What we still need is to establish bounds on D_i.
The trick: let $\hat{X}_i$ be an independent copy of X_i. Then

$$\begin{aligned}
\mathbf{E}\left[f_n(X_1,\dots,X_i,\dots,X_n) \mid \mathcal{F}_{i-1}\right] &= \\
\mathbf{E}\left[f_n(X_1,\dots,\hat{X}_i,\dots,X_n) \mid \mathcal{F}_{i-1}\right] &= \\
\mathbf{E}\left[f_n(X_1,\dots,\hat{X}_i,\dots,X_n) \mid \mathcal{F}_{i}\right] &
\end{aligned}$$

because both X_i and $\hat{X}_i$ share the same distribution, but $\mathcal{F}_i$ in respect to $\mathcal{F}_{i-1}$ doesn't contain additional information about $\hat{X}_i$. Hence we may rewrite the martingale difference again as

$$\begin{aligned}
D_i &= \mathbf{E}\left[Y_n \mid \mathcal{F}_i\right] - \mathbf{E}\left[Y_n \mid \mathcal{F}_{i-1}\right] \\
&= \mathbf{E}\left[f_n(X_1,\dots,X_i,\dots,X_n) \mid \mathcal{F}_i\right] - \mathbf{E}\left[f_n(X_1,\dots,X_i,\dots,X_n) \mid \mathcal{F}_{i-1}\right] \\
&= \mathbf{E}\left[f_n(X_1,\dots,X_i,\dots,X_n) \mid \mathcal{F}_i\right] - \mathbf{E}\left[f_n(X_1,\dots,\hat{X}_i,\dots,X_n) \mid \mathcal{F}_i\right]
\end{aligned}$$

Taking into account both terms only differ in including X_i resp. $\hat{X}_i$ we are able to postulate the existence of a constant d_i with $|D_i| \leq d_i$ and using Hoeffding's Inequaltity we finally arrive at Azuma's Inequality.

Theorem 6.6 (Azuma's Inequality). *Let $\{Y_n\}_{n=0}^{\infty}$ be a martingale and let there exist a constant c_n such that*

$$\left|f_n(X_1,\dots,X_i,\dots,X_n) - f_n(X_1,\dots,\hat{X}_i,\dots,X_n)\right| \leq c_i \tag{6.28}$$

where $\hat{X}_i$ is an independent copy of X_i. Then

$$\begin{aligned}
& Pr\left\{\left|f_n(X_1,\dots,X_i,\dots,X_n) - \mathbf{E}\left[f_n(X_1,\dots,\hat{X}_i,\dots,X_n)\right]\right| \geq x\right\} \\
= \quad & Pr\left\{|Y_n - \mathbf{E}\left[Y_n\right]| \geq x\right\} \quad \leq \quad 2\exp\left(-\frac{x^2}{2\sum_{i=1} nc_i^2}\right)
\end{aligned}$$

6.4 Application to KMP

6.4.1 Establishing m-Convergence

An alignment position in the text is called *unavoidable alignment position* if for any $r \leq i$ and any $l \geq i + m$ it's an alignment position when the algorithm is run on t_r^l. KMP-like algorithms share the same set of unavoidable alignment positions

$$\mathcal{U} = \bigcup_{l=1}^{n} \{U_l\} \tag{6.29}$$

where

$$U_l = \min\{\min_{1\le k\le l}\{t_k^l \preceq p\}, l+1\} \ . \tag{6.30}$$

This equation on the one hand specifies starting positions of pattern prefixes as unavoidable alignment positions (no positions inside those prefixes, as those would be jumped over) and on the other hand specifies steps of size one if there is no pattern prefix.
Interstingly this property seems to be uniquely limited to Morris-Pratt type algorithms – e.g. the Boyer-Moore algorithm does not have this property.

An algorithm is said to be l-convergent if there exists an increasing sequence of unavoidable alignment positions $\{U_l\}_{i=1}^{n}$ satisfying

$$U_{i+1} - U_i \le l \ . \tag{6.31}$$

Thus, l-convergence indicates the maximum size 'jumps' for an algorithm. For example, the brute force algorithm is 1-convergent and – what we are interested in more – KMP-like algorithms are m-convergent.

Proof. Let l be a text position and let r be any text position with $r \le U_l$. Then let $\{A_i\}$ be the set of APs when the algorithm is run on T_r^m. Note: $r \in \{A_i\}$ as the algorithm inevitably aligns at the starting position.
Then we may define the last alignment position A_J before U_l as

$$A_J = \max\{A_i : A_i < U_l\} \ . \tag{6.32}$$

So we have $A_{J+1} \ge U_l$. Using an adversary argument we will show that $A_{J+1} > U_l$ cannot be true, thus $A_{J+1} = U_l$. We define

$$y = \max\{k : M(k, (k - A_J) + 1) = 1\} \ , \tag{6.33}$$

so y is the rightmost position in the text we can do a comparison at when starting at A_J. Observe: $y \le l$. Otherwise, when comparisons would be done further, $T_{A_J}^l \preceq H$ would have to hold – and this in turn contradicts the definition of U_l.
Since KMP-like algorithms are strongly sequential, the text-pattern comparisons define non-decreasing sequences of text positions. For pattern-text comparisons at text position $y+1$ the pattern cannot be aligned at A_J, it has to be aligned at the next alignment position A_{J+1} with $A_{J+1} \le y+1 \le l+1$.
The definition of U_l leaves two possibilities: $U_l \le l$ if there is a prefix of the pattern, or $U_l = l+1$ if there is no prefix. The above equation $A_{J+1} \le l+1$ together with the second possibility $U_l = l+1$ contradicts the assumption $U_l < A_{J+1}$, so we may assume the first possibility $U_l \le l$ – this then implies that $H_{U_l}^l \preceq H$.
An occurence of the whole pattern is consistent with the available information. We – as we want to create a contradiction – may assume this is the case. As the sequence $\{A_i\}$ is non-decreasing and $A_{J+1} > U_l$ this occurence will be 'jumped over' and not be detected by the algorithm. Thus $A_{J+1} = U_l$ as needed.

Taking this a bit further we may combine $A_{J+1} \le y+1$ and $y \le A_J + m - 1$ to $A_{J+1} \le y+1 \le A_J + m + 1 - 1 = A_J + m$. So we have shown $A_{J+1} - A_J \le m$ for any pair (A_J, A_{J+1}) of APs in the text, thus KMP-like algorithms are m-convergent.
$\square$

6.4.2 Establishing Subadditivity

If c_n, the number of comparisons, is subadditive we may use the Subadditive Ergodic Theorem to prove linear complexity of algorithms. To achieve this we have to show that c_n is (almost) subadditive

$$c_{1,n} \leq c_{1,r} + c_{r,n} + a \ . \tag{6.34}$$

After rearranging the equation it suffices to prove the existence of an a such that

$$|c_{1,n} - (c_{1,r} + c_{r,n})| \leq a \ . \tag{6.35}$$

Let U_r be the smallest unavoidable aligment position greater than r. Then we are able to split $c_{1,n} - (c_{1,r} + c_{r,n})$ into $c_{1,n} - (c_{1,r} + c_{U_r,n})$ and $c_{r,n} - c_{U_r,n}$.
For the first part we have to count either:

- S_1: comparisons done after position r with alignment positions before r. Those only contribute to $c_{1,n}$ but neither to $c_{1,r}$ (the algorithm won't compare after r) nor $c_{U_r,n}$ (the algorithm doesn't align before U_r, thus not before r, too).
- S_2: comparisons done with alignment positions between r and U_r. Those also only contribute to $c_{1,n}$ but neither to $c_{1,r}$ nor $c_{U_r,n}$ (the algorithm in those cases only aligns before r resp. after U_r).

Summing up we arrive at

$$S_1 = \sum_{AP<r} \sum_{i \geq r} M(i, i - AP + 1) \leq m^2 \ . \tag{6.36}$$

This sum is bounded as there are at maximum m alignment positions before r with comparisons done after r. And for each aligment position there are at maximum m comparisons done.

$$S_2 = \sum_{r \leq AP < U_r} \sum_{i \geq r} M(AP + (i-1), i) \leq lm \tag{6.37}$$

This sum is bounded: because of the l-convergence of sequential algorithms there are at maximum l text positions between r and U_r, each with at maximum m comparisons done. Note: with m-convergent KMP-like algorithms this would resolve to m^2, too.
For the second part we have to count comparisons done with alignment positons before U_r (thus between r and U_r). Those contribute to $c_{r,n}$ only as $c_{U_r,n}$ starts comparing at position U_r.

$$S_3 = \sum_{r \leq AP < U_r} \sum_{i \geq r} M(AP + (i-1), i) \leq lm \tag{6.38}$$

This is the same sum as S_2, hence bounded for the same reasons. Finally we are able to put the parts together:

$$|c_{1,n} - (c_{1,r} + c_{r,n})| \leq |S_1 + S_2 - S_3| \leq m^2 + lm = a \ . \tag{6.39}$$

So by now we have show subadditivity

$$c_{1,n} \leq c_{1,r} + c_{r,n} + a \tag{6.40}$$

and are able to apply the Subadditive Ergodic Theorem.

6.4.3 Applying the Subadditive Ergodic Theorem

Before continuing we have to develop some modelling assumptions about the structure of text and pattern.

- **Deterministic Model:** Both text and pattern are non-random.[2] In this case we have to maximize complexity over all possible texts.
- **Semi-Random Model:** The text is a realization of stationary and ergodic sequence, the pattern is given, thus non-random. In this case we use average complexity over all texts.
- **Stationary Model:** Both text and pattern are a realization of a stationary and ergodic sequence, so we use average complexity over all texts and patterns.

Applying the Subadditve Ergodic Theorem yields similar results for worst and average case:

Deterministic Model: $$\lim_{n\to\infty} \frac{max_t(c_n(t,p))}{n} = \alpha_1(p)$$

Semi-Random Model: $$\lim_{n\to\infty} \frac{\mathbf{E}_t\left[C_n(p)\right]}{n} = \alpha_2(p)$$

Stationary Model: $$\lim_{n\to\infty} \frac{\mathbf{E}_{t,p}\left[C_n\right]}{n} = \alpha_3$$

6.4.4 Applying Azuma's Inequality

Even if we cannot determine the linearity constants α_1 to α_3, we still can show that C_n is concentrated around its mean.
We may assume that the text t is generated by a memoryless source, and C_n is a function of this random text $t = t_1, t_2, \ldots, t_n$. By flipping a single character we may change C_n by at most $2m^2$ comparisons, so C_n satisfies the condition for applying Azuma's Inequality:

$$\left|C_n\left(t_1, t_2, \ldots, t_i, \ldots, t_n\right) - C_n\left(t_1, t_2, \ldots, \hat{t}_i, \ldots, t_n\right)\right| \leq 2m^2 \tag{6.41}$$

Theorem 6.7. *Let t be a random text of length m generated by a memoryless source and let the pattern p of length m be given. Then the number C_n of comparisons made by the Knuth-Morris-Pratt algorithm is concentrated around its mean*

$$\mathbf{E}\left[C_n\right] = \alpha_2 n\left(1 + o(n)\right). \tag{6.42}$$

Equally

$$Pr\left\{\left|C_n - \alpha_2 n\right| \geq \epsilon n\right\} \leq 2\exp\left(-\frac{(\epsilon n)^2}{2\cdot n\cdot\left(2m^2\right)^2}\left(1+o(n)\right)\right)$$
$$= 2\exp\left(-\frac{\epsilon^2 n}{4m^4}\left(1+o(n)\right)\right) \tag{6.43}$$

for any $\epsilon > 0$.

[2]Applying Murphy's Law we may asume text and/or pattern to be exactly what you do not want them to be...

6.5 Concluding Remarks

The Subadditive Ergodic Theorem proves the existence of the linearity constant under quite general probabilistic assumptions. The main prerequisite is the existence of so called unavoidable alignment positions, a property that seems to be uniquely limited to Knuth-Morris-Pratt like algorithms.

Although we have not been able to compute this constant, we have been able to show that the number C_n of comparisons done is concentrated around its mean value $\alpha_2 n$.

Chapter 7

Greedy Algorithms for the Shortest Common Superstring Problem

Anton Nesterov

This paper is based on a article by A. Frieze and W.Szpankowski. It presents some greedy algorithms that solve Shortest Common Superstring Problem and their analysis in probabylistic framework. Also graph algorithms for equivalent problems are presented.

Various versions of the Shortest common superstring (in short SCS) problem play important role in data compression and DNA sequensing.
Problem Formulation. Given a collection of strings, say x^1, $x^2, \ldots, x^n$ over an alphabet Σ, find the shortest string z such that each of x^i appears as substring (a consecutive block) of z. In the DNA seqquencing another formulation of the problem may be of even greater interest. We call it an approximate SCS and one asks for a superstring that contains approximately (e.g in the Hamming distance sense) the original strings as x^1, $x^2, \ldots, x^n$ as substrings.
Our results are about some greedy approximations of the SCS but in a probabilistic framefork. We prove that several greedy algorithms for the SCS problem are asymptotically optimal in the sence that thay produce a total overlap of SCS that differs from the optimal (maximum) overlap by a quantity that is an order of magnutude smaller than the leading term of the overlap.
We assume, that the strings are generated independently. We first consider the so-called Bernoulli model in which symbols of the alphabet Σ are generating independently within a string. Later we extends our results to other models: Markovian model and Mixing Model, which is generalization of previous ones.

7.1 Definitions

Before presenting some results, we introduce some notation and a framefork for describing our algorithms.
Suppose $x = x_1 x_2 \ldots x_3$ and $y = y_1 y_2 \ldots y_3$ are strings over the same finite alphabet $\Sigma = (\omega_1, \omega_2, ..., \omega_M)$ where M is the size of the alphabet. We define their overlap

$$o(x, y) = \max\{j : y_i = x_{r-j+i}, 1 \leq i \leq j\}.$$

If $x \neq y$ and $k = o(x, y)$, then

$$x \oplus y = x_1 x_2 ... y_{k+1} y_{k+2} ... y_s.$$

Let S be a set of all superstrings built over strings $x^1, .., x^n$. Then

$$O_n^{opt} = \sum_{i=1}^{n} |x|^i - \min_{z \in S} |z|.$$

We assume that the input strings are independently generated. We analyse the Bernoully model, that is, each $x = x^j = x_1 x_2 ... x_{i-1}$ is the same length l and and x_i is generated independently of x_1, $x_2 \ldots x_{i-1}$. Futhermore, $P(x_i = \omega_j) = p_j > 0$ for $1 \leq j \leq M$. Let

$$H = \sum_{i=1}^{m} p_i \log p_i$$

be the associated entropy for the Bernoully model (i.e., memoryless source).

7.2 Greedy algorithms

We study the following algorithm: its input is the strings $x^1, x^2, .., x^n$ over Σ. It outputs a string z which is a superstring of the input.

Generic greedy algorithm.
1. $I \leftarrow \{x^1, x^2, ...x^n\}; O_n^{gr} \leftarrow 0;$
2. **repeat**
3. choose $x, y \in I$; $z = x \oplus y$
4. $I \leftarrow (I \setminus \{x, y\})$
5. $O^g r_n \leftarrow O_n^{gr} + o(x, y)$
6. **until** $|I| = 1$

We consider two variants:
GREEDY. In Step 3 choose $x \neq y$ in order to maximize $o(x, y)$.
RGREEDY. In Step 3 x is the string z produced in the previous iteration, while y is chosen in order to maximize $o(x, y) = o(z, y)$. Our initial choice for x is x^1. Thus in RGREEDY we have one "long" string z grows by addition of strings at the right-hand end.

7.3 Results

Consider the SCS problem under the Bernoulli model. Let $P = \sum_{j=1}^{M} p_i^2$. Then, with high probability,

$$\lim_{n\to\infty} \frac{O_n^{opt}}{n \log n} = \frac{1}{H}$$

$$\lim_{n\to\infty} \frac{O_n^{gr}}{n \log n} = \frac{1}{H}$$

provided

$$|x| > -\frac{4}{\log P} \log n$$

for all $1 \leq i \leq n$
In many applications, notably for data compression and the DNA recombination problem, the Bernoully model assumption is too unrealistic. Therefore, we extend this theorem to the case when there is some dependency among symbols withing a string. However we still assume that strings x^1,$x^2, \ldots, x^n$ are statically independent. But we restrict somewhat the dependency among symbols of each string, that is, we desribe a main ideas of the mixing model.
Mixing Model. During the generation of string, each symbol depends on all previous ones, but the farther the symbol the lesser the dependence on it.

7.4 Compression

The SCS can be used to compress strings. Indeed, instead of storing all strings of total length nl we can store the SCS and n pointers indicating the beginning of an original string plus length of all strings. However, this does not provide optimal compression (which is known to be the entropy H). Show this, compute the compressionn ratio C_n which is defined as the ratio of the number of bits needed to transmit the compression code to the length of the original set of strings. It is easy to see that

$$C_n = \frac{nl - (1/H)n \log n + n \log_2(nl - (1/H)n \log n)}{nl}$$

where the first term of the numerator represents the length of the Shortest superstring and the second term corresponds to the number of bits needed to encode the pointers. Observe that when the length of a string l grows faster than $\log n$, then $C_n \to 1$, that means no compression. When $l = O(\log n)$ some compression might take place. The fact that SCS does not compress well is hardly surprising: in the construction of

SCS we do not use all available redundancy of all strings but only that contained in suffixes/preffixes of original strings.

7.5 Graph processes

In this section some graph algorithms, that correspond to GREEDY and RGREEDY are presented. But before them,
First, we show that a pair i, j such that $o(x^i, x^j) \geq l/2$ unlikely exists. Let ϵ denote the event that there is no such pair. If $l = K \log n$, then

$$P(\neg\epsilon) \leq \frac{n(n-1)}{2} \sum_{k=l/2}^{l} P^k = O(n^{2+(K \log P)/2}) = o(1)$$

provided $K \geq -4/\log P$.

7.5.1 RGREEDY

Consider a tree process that is equal to RGREEDY. Tree T be an infinite rooted M-ary tree. M (size of an alphabet) edges leading down from each vertex will be labeled with $\omega_1, \omega_2, ...\omega_M$. Thus, each vertex of depth d is identified with string of length d. Also, label each vertex v with an integer $v(v)$, number of strings that have the prefix associated with this vertex.
We model the process of RGEEDY in the following way: particle Z starts at the root. Then at a vertex v it moves to v's ω_j descendent with probability p_j. The particle stops at depth $l/2$. Let $\omega = s_k s_{k-1}...s_1$be the lowest vertex on the path traversed that has a nonzero v. This process models the computation of the largest suffix $s_k s_{k-1}...s_1$ of z which can be merged with a prefix of a^i.
Then we model the deletion of $a^t = a_1 a_2 ... a_{l/2}$ which has the prefix $a_1 a_2 ... a_k$. Let $\omega_i = a_1 a_2 ... a_i$. Put $v(\omega_i) = \max\{0, v(\omega_i) - 1\}$ for $1 \leq i \leq l/2$.
We iterate the above process $n - 1$ times.

7.5.2 GREEDY

Let D be the digraph $([n], A)$ with edge weights $\omega_{i,j} = o(b^i, a^j)$ for $i, j \in [n]$.
Sort the edges A into $e_1, e_2, ..., e_N$, where $N = n^2$, so that $\omega(e_i) \geq \omega(e_{i+1})$;
$S_G \leftarrow \emptyset$;
For $i = 1$ **to** N **do**: **if** $S_G \cup \{e_i\}$ contains **in** D neither a vertex of outdegree or indegree at least 2 in S_G, nor a directed cycle, **then** $S_G \leftarrow S_G \cup \{e_i\}$.
After termination S_G contains $n - 1$ edges of a Hamilton path of D and corresponds to superstring x^1,$x^2, \ldots, x^n$. The selection of an edge weight (b^i, a^j) corresponds to overlaping x^i to the left of x^j.

Chapter 8

Analysis of Pattern Occurances

Roland Aydin

This paper will summarize the proof for the formula to compute the expected number of occurrences of a given pattern H in a text of size n. The intuitive solution of $E[O_n(H)] = P(H)(n - m + 1)$ will be verified utilising generating functions. Frequency analysis will rely on the decomposition of the text T onto languages, the so-called initial, minimal, and tail languages. Going from there to their generating functions both for a Markovian and a Bernoulli environment, the formula will be shown to work due to properties of the respective generating functions.

8.1 Preliminaries

Markov sequence

A sequence $X_1, X_2, ...$ of random variates is called a *Markov sequence* of order 1 iff, for any n,

$$F(X_n|X_{n-1}, X_{n-2}, ...X_1) = F(X_n|X_{n-1})$$

i.e., if the conditional distribution F of X_n, assuming $X_{n-1}, X_{n-2}, ...X_1$ equals the conditional distribution F of X_n assuming only X_{n-1}.

Markov chain

If a Markov sequence of random variates X_n take the *discrete values* $a_1, ..., a_N$ then

$$P(x_n = a_{in}|x_{n-1} = a_{in-1}, ..., x_1 = a_{i1}) = P(x_n = a_{in}|x_{n-1} = a_{in-1})$$

and the sequence x_n is called a *Markov chain* of order 1.

Correlation of patterns

A *correlation* of two patterns X (size m) and Y is a string, denoted by XY, over the set $\Omega = \{0, 1\}$.

$$|XY| = |X|$$

Each position i can be computed as

$$i = 1 \Leftrightarrow \text{place } Y \text{ at } X_i \wedge \text{ all overlapping pairs are identical} \text{else} \qquad i = 0$$

Example of pattern correlation

Let $\Omega = \{M, P\}$, $X = MPMPPM$ and $Y = MPPMP$. Then XY can be deduced in the following manner:

```
X: HTHTTH
Y: HTTHT          0
     HTTHT        0
       HTTHT      1
         HTTHT    0
           HTTHT  0
             HTTHT 1
```

whilst YX can be shown to equal 00010

Representation of the correlation

Other representations of either string:

1. as a number in some base t. Thus, e.g. $XY_2 = 9$
2. as a polynomial. Thus, e.g. $XY_t = t^3 + 1$

Autocorrelation

Furthermore, *autocorrelation* of X can be defined as XX. It represents the periods of X, i.e. those shifts of X that cause that pattern to overlap itself. Using $Y = MPPMP$ from our previous example, YY evaluates to 10010 Using $A = MMM$, AA evaluates to 111

Autocorrelation set

Given a string H, the autocorrelation *set* A_{HH} or just A is defined as

$$A_{HH} = \{H_{k+1}^m : H_1^k = H_{m-k+1}^m\}$$

Example of an autocorrelation set

Let $H = SOS$ The autocorrelation reveals to be

$$HH = 101$$

whereas the autocorrelation set in that case is

$$A = \{\epsilon, 01\}$$

Let's play a game

The Penny game - invented by Penney.
Each player chooses a pattern.
They then flip a coin until the pattern comes up consecutively. The player who chooses only one symbol (k times), has a chance to win of at least 0.5 This is because of the "optimal" autocorrelation.

8.2 Sources

Bernoulli

A *Bernoulli Source*, or *memoryless source*, generates text randomly.
Every subsequent symbol (of a finite alphabet) is created independently of its predecessors, and the probability of each symbol is not necesserily the same.
If it is, the Source is called a *symmetric*, or *unbiased* Bernoulli Source.
If text over an alphabet S is generated by a Bernoulli Source, then each symbol $s \in S$ *always* occurs with probability $P(s)$.

Markovian Source

A *Markovian Source* generates symbols based not on the *a priori* probability of each symbol.
Instead, it only heeds a (finite) set of predecessors to ascertain the probability of each next symbol.
In order to do so, it requires a *memory* of previously emitted symbols.
Text generated by a Markovian Source is a realization of a Markov sequence of order K.
K denotes the number of previous symbols that the probability of the next symbol depends on.
In our application, this sequence will be stationary and $K = 1$, i.e. a first-order Markov sequence.
When computing the next symbol, we only need to observe the last symbol.
In our case ($K = 1$), the transition matrix is defined by

$$P = \{p_{i,j}\}_{i,j \in S}$$

where

$$p_{i,j} = \text{Probability } (t_{k+1} = j | t_k = i)$$

The matrix entry (i, j) denotes the conditional probability of the next symbol being j if the current symbol is i.

8.3 Generating functions of languages

What is a language, after all

A language L is a collection of words.
This collection must satisfy certain properties to belong to a specific language.
Thus, we can associate with a language L its generating function $L(z)$.

Generating functions

Given a sequence $\{a_n\}_{n\geq 0}$, we know its generating function is defined as

$$A(z) = \sum_{n\geq 0} a_n z^n$$

For sinister purposes, we represent it differently as

$$A(z) = \sum_{\alpha\in S} z^{w(\alpha)}$$

where S is a set of objects (words ...) and $w(\alpha)$ is a weight function.
Henceforth we will interpret it as the size of α, i.e. $w(\alpha) = |\alpha|$
The equivalence becomes evident when we set a_n to be the number of objects α satisfying $w(\alpha) = n$. Now we have a more combinatorial view

Generating function of a language

Now, for any language L, we define its generating function $L(z)$ as

$$L(z) = \sum_{w\in L} P(w) z^{|w|}$$

where $P(w)$ is the probability of word w's occurence and $|w|$ is the length of w.
So the coefficient of $z^{|w|}$ is the sum of the probabilites all words of that length.
In addition, we assume that $P(\epsilon) = 1$. So every language includes the empty word (as we know).

Conditional generating function

In addition, the H-conditional generating function of L is given as

$$\begin{aligned} L_H(z) &= \sum_{w\in L} P(w|w_{-m} = h_1 \dots w_{-1} = h_m) z^{|w|} \\ &= \sum_{w\in L} P(w|w_{-m}^{-1} = H) z^{|w|} \end{aligned}$$

where w_{-i} is the symbol preceding the first character of w at distance i.
We use this definition for Markovian sources, where the probability depends on the previous symbols.

Example: autocorrelation generating function

In our previous example, the autocorrelation set was

$$A = \{\epsilon, 01\}$$

The generating function of the set is

$$A(z) = 1 + \frac{z^2}{4}$$

given a Bernoulli source, and

$$A_{SOS}(z) = 1 + p_{SO} p_{OS} z^2$$

given a Markovian source of order one.

Formulating our objective

We will now formulate the special generating functions whose closed form we will later strive to compute:

1. $T^{(r)}(z) = \sum_{n\geq 0} Pr(O_n(H) = r)z^n$
2. $T(z,u) = \sum_{r=1}^{\infty} T^{(r)}(z)u^r = \sum_{r=1}^{\infty}\sum_{n=0}^{\infty} Pr(O_n(H) = r)z^n u^r$

8.4 Declaring languages

Introduction

Let H be a given pattern.

- The *initial language* R is the set of words containing only **one** occurrence of H, located at the **right** end.
- The *tail language* U is defined as the set of words u such that Hu has exactly **one** occurrence of H, which occurs at the **left** end.
- The *minimal language* M is the set of words w such that Hw has exactly **two** occurrences of H, located at its **left** and **right** ends.

Component languages

We differentiate several special languages, given a pattern H. "$\cdot$" stands for concatenation of words.

1. $R = \{r : r \in T_1 \wedge H \text{ occurs at the right end of} r\}$
2. $U = \{u : H \cdot u \in T_1\}$
3. $M = \{w : H \cdot w \in T_2 \wedge H \text{ occurs at the right end of } H \cdot w\}$

8.5 Language relationships

Qualities of T_r

At first, we will try to describe the languages T and T_r in terms of R, M and U:
$\forall r \geq 1$:

$$T_r = R \cdot M^{r-1} \cdot U$$

Composition proof (T_r)

Proof:
First occurance of H in a T_r word determines the prefix p
which is in R.
From that prefix on, we look onward until the next occurance of H.
The found word w is $\in M$.
After $r-1$ iterations, we add a H-devoid suffix, which is in U, because its prefix has H at the end.

□

Qualities of T

The "extended" version of T_r, its words including an arbitrary number of H occurrences, can be composed similarily:

$$T = R \cdot M^* \cdot U$$

where $M^* := \bigcup_{r=0}^{\infty} M^r$

Composition proof (T)

Proof:
A word belongs to T, if for some $1 \leq r < \infty$ it belongs to T_r.
As $\bigcup_{r=1}^{\infty} M^{r-1} = \bigcup_{r=0}^{\infty} M^r = M^*$, the assertion is proven.

□

Four language relationships

Analyzing the relationships between M, U and R further, we introduce

1. W, the set of all words
2. S, the alphabet set
3. the operators "+" and "-", which denote disjoint union and language subtraction

Four language relationships I

$$\bigcup_{k \geq 1} M^k = W \cdot H + (A - \{e\})$$

Proof:
$\leftarrow$:
Let k be the number how often H occurs in $W \cdot H$.
$k \geq 1$.
The *last* occurrence of H in every included word is on the right.
That means, that $W \cdot H \subseteq \bigcup_{k \geq 1} M^k$.
$\rightarrow$:
Let $w \in \bigcup_{k \geq 1} M^k$.
Iff $|w| \geq |H|$, then surely the inclusion is correct.
Iff $|w| < |H|$ (how can that be?), then $w \notin W \cdot H$.
But then, necessarily, $w \in A - \{\epsilon\}$, because the second H in Hw overlaps with the first H by definition (it is element of M^k), so w must be in the autocorrelation set A.

□

Four language relationships II

$$U \cdot S = M + U - \{e\}$$

Proof:
All words of S consist of a single character s.
Given a word $u \in U$ and concatenating them, we differentiate two cases.
If Hus contains a second occurrence of H, it is clearly at the right end. Then $us \in M$.
If Hus does contain only a single H, then us must be non-empty word of U.

□

Four language relationships III

$$H \cdot M = S \cdot R - (R - H)$$

Proof: $\rightarrow$: Let sw be a word in $H \cdot M$, $s \in S$ (we can write every such word in this way WLOG).
sw contains exactly two times H, evidently at its left, and also at its right end. Thus, sw is also $\in S \cdot R$
$\leftarrow$: If a word swH from $S \cdot R$ is not in R, then because it contains a second H starting at the left end of sw, because $wH \in R$. Of course, in that case it is $\in H \cdot M$.

$\square$

Four language relationships IV

$$T_0 \cdot H = R \cdot A$$

Proof:
Let wH be $\in T_0 \cdot H$. Then there can be either be one or more occurences of H in wH, one of which is at the right end.
If there is no second one, then wH is $\in R$ by definition of R
If, however, there is a second one, then it overlaps somehow with the first one.
So we view the word until the end of the *first* H, which is in R. Due to the overlapping, the remaining part is $\in A$.

$\square$

One more

Combining relationships II and III yields

$$H \cdot U \cdot S - H \cdot U = (S - \epsilon)R$$

No proof is necessary, as we have validated both ingredients.
Using II, the left side is $H(U \cdot S - U) = H \cdot M$
The right side is

$$S \cdot R - R = S \cdot R - (R \cap S \cdot R) \qquad = S \cdot R - (R - H)$$

Together, that is just relationship III.

8.6 Languages & Generating Functions

in the bernoulli environment

We will now transcend from languages to their generating functions. Given any language L_1, we know its generating function to be

$$A_1(z) = \sum_{w \in L_1} P(w) z^{|w|}$$

So what is the the result of multiplying two languages (i.e. concatenating them) in respect to their gen. func.? What is $L_3 = L_1 \cdot L_2$?

$$\begin{aligned}
A_3(z) &= \sum_{w \in L_3} P(w) z^{|w|} \\
&= \sum_{w \in L_1 \wedge w \in L_2} P(w_1)P(w_2) z^{|w_1|+|w_2|} \\
&= \sum_{w \in L_1} P(w_1) z^{|w_1|} \sum_{w \in L_2} P(w_2) z^{|w_2|} \\
&= A_1(z) A_2(z)
\end{aligned}$$

! The assumption $P(wv) = P(w)P(v)$ only holds true with a memoryless source.

Special Cases

A few particular cases:

- S (alphabet set) $\Rightarrow S(z) = \sum_{s \in S} P(s) z^{|s|} = z$
- $L = S \cdot L_1 \Rightarrow L(z) = zL_1(z)$
- $\{\epsilon\} \Rightarrow E(z) = \sum_{w \in \{\epsilon\}} P(w) z^{|w|} = 1 \cdot 1 = 1$
- $H \Rightarrow H(z) = \sum_{w=H} P(H) z^{|H|} = P(H) z^m$
- W (the set of *all* words) $\Rightarrow W(z) = \sum P(w) z^{|k|} = \sum_{k \geq 0} z^k = \frac{1}{1-z}$

8.7 Looking for Generating Functions

Translating I

We will now attempt to translate our known language relationships into generating functions: In case I only, the formula we derive is correct just for a memoryless source.

$$\begin{aligned}
\bigcup_{k \geq 1} M^k &= W \cdot H + (A - \{e\}) \\
\sum_{k=1}^{\infty} M_H(z)^k &= W(z) \cdot P(H) z^m + A_H(z) - 1 \\
\sum_{k=0}^{\infty} M_H(z)^k - 1 &= \frac{1}{1-z} \cdot P(H) z^m + A_H(z) - 1 \\
\frac{1}{1 - M_H(z)} &= \frac{1}{1-z} \cdot P(H) z^m + A_H(z)
\end{aligned}$$

Translating II

$$\begin{aligned}
U \cdot S &= M + U - \{e\} \\
U \cdot S - U &= M - \{e\} \\
U_H(z) z - U_H(z) &= M_H(z) - 1 \\
U_H(z)(z-1) &= M_H(z) - 1 \\
U_H(z) &= \frac{M_H(z) - 1}{(z-1)}
\end{aligned}$$

Translating III

$$\begin{aligned} H \cdot M &= S \cdot R - (R - H) H \cdot M - H \qquad = S \cdot R - R \\ P(H)z^m M_H(z) - P(H)z^m &= S(z) \cdot R(z) - R(z) \\ P(H)z^m (M_H(z) - 1) &= R(z)(z-1) \\ R(z) &= P(H)z^m \frac{M_H(z) - 1}{z - 1} \\ R(z) &= P(H)z^m U_H(z) \end{aligned}$$

8.8 Main findings I

$T^{(r)}(z)$

We remember, that for $r \geq 1$

$$T_r = R \cdot M^{r-1} \cdot U$$

We have now gleaned every component, and can translate it (for $r \geq 1$) into

$$T^{(r)}(z) = R(z)M^{r-1}(z)U_H(z)$$

$T(z,u)$

We do also remember, that

$$T = R \cdot M^* \cdot U$$

As T is the language with *any* number of Hs, its generating function is indeed ...

$$T(z,u) = R(z)\frac{u}{1 - uM_H(z)}U_H(z)$$

8.9 On to other shores

What is left to do?

We still have no formula of gathering $O_n(H)$, i.e. the frequency of H-occurrences ($|H| = m$) in random text of length n over an alphabet S with $|S| = V$.
Let us make an educated guess, though. What we do not know, is how important *overlapping* is. Assuming to disregard that topic, the answer *could* be

$$E[O_n(H)] = P(H)(n - m + 1)$$

It is.
But why?

Using derivatives

Looking at our bivariate generating function of T,

$$T(z,u) = \sum_{r=1}^{\infty} \sum_{n=0}^{\infty} Pr(O_n(H) = r) z^n u^r$$

we notice that we would like the two sums to be reversed. Deriving it after u ...

$$T_u(z,u) = \sum_{r=1}^{\infty}\sum_{n=0}^{\infty} Pr(O_n(H) = r) z^n r \text{ (=\#Occ) } u^{r-1}$$

... and setting u to 1 leads to ...

$$T_u(z,1) = \sum_{n=0}^{\infty}(\sum_{r=1}^{\infty} Pr(O_n(H) r) z^n$$

Proof Preparations

To shorten things, we introduce

$$D_H(z) = (1-z)A_H(z) + z^m P(H)$$

and rewrite $M_H(z)$ as

$$M_H(z) = 1 + \frac{z-1}{D_H(z)}$$

as well as

$$U_H(z) = \frac{1}{D_H(z)}$$

and

$$R(z) = z^m P(H)\frac{1}{D_H(z)}$$

Deriving the closed form formula (1)

$$\begin{aligned} T_u(z,u) &= R(z)U_H(z)\frac{u}{(1-uM_H)}\frac{d}{du} \\ &= R(z)U_H(z)\frac{(1-uM)+uM}{(1-uM_H)^2} \\ &= R(z)U_H(z)\frac{1}{(1-uM_H)^2} \end{aligned}$$

Deriving the closed form formula (2)

u is now set to 1 due to the previous calculus:

$$\begin{aligned} T_u(z,1) &= R(z)U_H(z)\frac{1}{(1-M_H)^2} \\ &= R(z)U_H(z)(1-1+\frac{z-1}{D_H(z)})^{-2} \\ &= R(z)U_H(z)\frac{D_H(z)^2}{(z-1)^2} \\ &= R(z)\frac{1}{D_H(z)}\frac{D_H(z)^2}{(z-1)^2} \\ &= z^m P(H)\frac{1}{D_H(z)}\frac{D_H(z)}{(z-1)^2} \\ &= \frac{z^m P(H)}{(z-1)^2} \end{aligned}$$

Main findings II

As the text has length n, we are extracting the nth coefficient of $T_u(z,1)$, and *voilà*

$$\begin{aligned} E[O_n] &= [z^n]T_u(z,1) \\ &= P(H)[z^n]z^m(1-z)^{-2} \\ &= P(H)[z^{n-m}](1-z)^{-2} \\ &= (n-m+1)P(H) \end{aligned}$$

About certainty

the variance of $E(O_n(H)$ is, for a $r > 1$:

$$Var[O_n(H)] = nc_1 + c_2 + O(r^{-n})$$

where

$$c_1 = P(H)(2A_H(1) - 1 - (2m-1)P(H) + 2P(H)E_1))$$

$$\begin{aligned} c_2 \quad &= P(H)((m-1)(3m-1)P(H) - (m-1) \\ &(2A_H(1) - 1) - 2A'_H(1)) - 2(2m-1) \\ &(P(H)^2E_1 + 2E_2P(H)^2 \end{aligned}$$

E_1, E_2 are

$$E_1 = \frac{1}{\pi_{h1}}[(P-\Pi)Z]_{h_m,h_1}E_2 \qquad = \frac{1}{\pi_{h1}}[(P^2-\Pi)Z^2]_{h_m,h_1}$$

Without going into detail (cf. literature references), we see that the Variance depens mainly on the length of the text plus a constant.

Chapter 9

Rice's integrals – a method for solving generalized differences

Thomas Preu

Often in the analysis of algorithm and data structures we have the need to estimate the asymptotic growth of differences and sequences defined by recurrence equations. But often we don't know the explicit representation of the solutions. Therefor we need methods, which we can establish asymptotics without knowing the exact representation. Rice's integral is such a method. In this paper we will introduce basics in complex analysis and develope the mathematic foundations needed in theoretical computer science. The paper is based on the lecture "Analysis 4" held by Prof. W. Heise in 2003 at the TU München and an article by Flajolet et al. [FS95].

9.1 Basics of Complex Analysis

9.1.1 Complex Differentiability

In the whole of this paper, we will have to deal massivly with complex analysis. As complex analysis is often seen as a discipline of pure mathematics, most people working in computer science, even in theoretical computer science, have not heard much about it. So I will give a rough introduction to it and present the needed theorems. It is assumed that the reader has basic knowledge of real analysis, e.g., knows what a real function is, what continuity and differentiablity means, and , what complex numbers are.

First of all we consider a complex mapping on an open set E

$$f : E \subset \mathbb{C} \to \mathbb{C}, z = x + yi \mapsto f(z) = u(x, y) + iv(x, y) \tag{9.1}$$

This mapping is said to be continous, if the correspondig 2-dimensional mapping

$$g : E' \subset \mathbb{R}^2 \to \mathbb{R}^2, (x, y) \mapsto (u(x, y), v(x, y)) \tag{9.2}$$

is continous in the sense of multidimensional real analysis.

If g is differentiable at (x_0, y_0), we have the derivative

$$A := \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix}_{x=x_0, y=y_0} \tag{9.3}$$

and A is a linear mapping with the property

$$\forall (x, y) \in E' : g(x, y) = g(x_0, y_0) + A \cdot \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} + B(x - x_0, y - y_0) \tag{9.4}$$

where B is a mapping with $B(0, 0) = (0, 0)^T$ and $\lim\limits_{(x,y)\to(0,0)} \frac{B(x,y)}{|(x,y)|} = (0, 0)^T$ which, of course, implies continuity at $(0, 0)$

As a linear mapping $E \subset \mathbb{C} \to \mathbb{C}$ is in fact a complex multiplication, the natural way to introduce complex derivatives, is trying to find a complex number $f'(z_0)$, which acts the same way as A in (9.4):

$$\forall z \in E : f(z) = f(z_0) + f'(z_0) \cdot ((x - x_0) + i(y - y_0)) + b((x - x_0) + i(y - y_0)) \tag{9.5}$$

where b is a complex mapping with $b(0) = 0$ and $\lim\limits_{z\to 0} \frac{b(z)}{|z|} = 0$. If such a number $f'(z_0)$ exists, f is said to be complex differentiable at z_0.

Now we compare the matrix-vector-multiplication with the product of two complex numbers:

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a_{1,1}x_1 + a_{1,2}x_2 \\ a_{2,1}x_1 + a_{2,2}x_2 \end{pmatrix} \tag{9.6}$$

$$(a_1 + ia_2)(x_1 + ix_2) = (a_1x_1 - a_2x_2) + i(a_2x_1 + a_1x_2) \tag{9.7}$$

we get the correspondence $a_{1,1} = a_1 = a_{2,2}$ and $a_{1,2} = -a_2 = a_{2,1}$

From this, it is plausible that a real function $\mathbb{R}^2 \to \mathbb{R}^2$ understood as a complex function is complex differentiable, iff $u_x = v_y$ and $u_y = -v_x$. These equations are known as the Cauchy-Riemann Differential Equation (CRDE).

As a result, we have that every complex differentiable function is differentiable in a real sense, but a real differentiable function is only complex differentiable, if the CRDEs are satisfied. So complex differentiability is somewhat stronger.

Definition 9.1. A complex mapping f, as introduced in (9.1), is said to be differentiable at a point $z_0 \in E$ with derivative $f'(z_0)$, iff (9.5) holdes for some appropriate b.

Definition 9.2. The complex mapping f is said to be holomorphic at a point z_0 on E, iff f is complex differentiable at any point of a neighbourhood $F \subset E$ of z_0, which is more than just complex differentiable at z_0.

Definition 9.3. The complex mapping f is said to be holomorphic on an open set $F \subset E$, iff f is holomorphic at any point $z_0 \in F$.

In fact, when evaluating complex derivatives, not many changes occur; e.g. $(z^n)' = nz^{n-1}$, $(e^z)' = e^z$ or $(\sin(z))' = \cos(z)$

9.1.2 Integration

In real analysis we integrate over intervalls, where the integral is a limit of sums. These sums take into account the values of the function and the length of the parts of the discretisation of the intervall. The discretisation Z gets "finer" in the sense, that the longest part tends to 0:

$$\int_a^b f(x)dx = \lim_{n\to\infty, |Z_n|\to 0} \sum_{k=0, x_{n,k}\in\left[z_{n,k}, z_{n,k+1}\right]}^{n} f(x_{n,k})(z_{n,k+1} - z_{n,k}) \tag{9.8}$$

where $z_{n,0} = a < z_{n,1} < \ldots < z_{n,n} < z_{n,n+1} = b$ for every n
In some way, we walk along the intervall from a to b, picking some points, where we examine the function closer, and get a number from this process. Of course, it shouldn't be important, whether we are faster or slower while "walking". This is expressed by the substitution formula:

$$\int\limits_a^b g(\gamma(s))\gamma'(s)ds = \int\limits_c^d g(t)dt \tag{9.9}$$

where $\gamma : [a,b] \to [c,d]$ is piecewise differentable and monotonous and $g : [c,d] \to \mathbb{R}$ is piecewise continous. In γ we have the information how fast we are on the intervall at each point of the parametrisation.
In a similar way we can integrate "along" a piecewise differentiable curve $\gamma : \mathbb{R} \to \mathbb{C}; t \mapsto \gamma(t)$; we can think of $\mathbb{C}$ as $\mathbb{R}^2$. In this situation γ is called an integration path. An integral of a complex function is the sum of the integrals of the real and the imaginary part, so for a complex function f we get[1]:

Definition 9.4. The integral of a complex function f along a curve γ is:

$$\int_\gamma f(z)dz = \int_a^b f(\gamma(t)) \cdot \gamma'(t)dt = \int_a^b \Re\left(f(\gamma(t)) \cdot \gamma'(t)\right) dt + i \int_a^b \Im\left(f(\gamma(t)) \cdot \gamma'(t)\right) dt \tag{9.10}$$

For example we have the curve $\gamma : [0, 2\pi] \to \mathbb{C}; t \mapsto e^{it}$ which is a circle of radius 1 and center 0 and $f(z) = \frac{1}{z}$.

$$\int_\gamma \frac{dz}{z} = \int_0^{2\pi} \frac{1}{e^{it}} \cdot ie^{it}dt = \int_0^{2\pi} idt = 2\pi i \tag{9.11}$$

Some properties of real integrals can be copied almost literarily:
The length of a path of integration is

$$\Lambda(\gamma) = \int_a^b \left|\gamma'(t)\right| dt = \int_a^b \sqrt{(\Re\gamma'(t))^2 + (\Im\gamma'(t))^2}dt \tag{9.12}$$

For a piecewise continous $f : [a,b] \to \mathbb{C}$:

$$\left|\int_a^b f(t)dt\right| \leq \int_a^b |f(t)|\, dt \tag{9.13}$$

For f, continous on the image of a path of integration γ and M the maximum absolute value of f on that image, we have:

$$\left|\int_\gamma f(t)dt\right| \leq M \cdot \Lambda(\gamma) \tag{9.14}$$

So far we have always considered curves and not their images as domain of integration. But the following result shows, that somehow only the domain is important. For this purpose we call a surjective, piecewise continous differentiable, real function ϕ a parameter transform, iff for all points of its domain, where ϕ is differentiable, the derivative is greater than 0.

Theorem 9.1. *If $\phi : [c,d] \to [a,b]$ is a parameter transform and $\gamma_1 : [a,b] \to \mathbb{C}$ an integration path, then $\gamma_2 = \gamma_1 \circ \phi : [c,d] \to \mathbb{C}$ is also a path of integration and for a function f continous on the image of γ_1 we have:*

$$\int_{\gamma_1} f(z)dz = \int_{\gamma_2} f(z)dz \tag{9.15}$$

[1] Note, that we integrate along a curve in the first place and not the image of the curve.

Although the proof is simple, we won't state it here. The essence of this theorem is, that the speed, at which we follow the line defined by a curve, is not important for the value of the integral – as long as we follow the line "smooth" enough, i.e. we don't "jump" around and we don't stay to long at one point.

Definition 9.5. For a continous function f on an open set $E \subset \mathbb{C}$ we call $F : E \to \mathbb{C}$ the antiderivative of f, if F is holomorphic on E and $F' = f$. f is said to have local antiderivatives, if for each $z_0 \in E$ there exists a neighbourhood $G \subset E$ of z_0, so that there exists a holomorphic F with $F' = f$ on G.

9.1.3 Holomorphic functions and the Cauchy Integral Theorem

Holomorphic functions do have many nice properties. The proofs are often sophisticated and technical, so we will again omit them.

Definition 9.6. A complex function f is said to be analytic at a point z_0 if there exists a neighbourhood G of z_0, for example an open circle, such that f has a powerseries expansion, which converges for some radius R. f is said to be analytic on an open set, if it is analytic on every point of this set.

Theorem 9.2. *For a continous function f on a domain[2] E the following statements are equivalent:*

1. *f is holomorphic*
2. *f has local antiderivatives*
3. *f is differentiable in the real sense and obeys the CRDEs*
4. *f is analytic*

Since powerseries can be differentiated without any loss in their convergent domain, this shows, that holomorphic function are arbitrarily often differentiable – at least in a local sense; but since we can cover domains with circles we can connect the domains of definition of these derivatives and get one derivative for the whole domain of a holomorphic function. This is an outstandig property of holomorphic functions: in real analysis a function can be differentiable but doesn't have to be for a second time; in complex analysis a function, which is differentiable on an open set, is arbitrarily often differentiable.
Since powerseries are more or less Taylor series, holomorphic functions are completly defined by only one point – if you know all higher order derivatives at just one point. Another interesting property similar to that is, when you know the function at countable many points, which cluster at one point, then the holomorphic functions is also defined uniquely. All this has the consequence, that whenever we have a holomorphic function on a certain domain, then possible holomorphic extensions are again unique – so restrictions of functions to a domain which is too small for our purpose is no problem because we just extend the function to wherever we want, at least if no poles hinder us.

Theorem 9.3. *Let E be a convex domain and $Z \subset E$ a set of points without cluster points. If $f : E \to \mathbb{C}$ is continous and holomorphic on $E \setminus Z$ then for every closed (meaning $\gamma(a) = \gamma(b)$) path of integration $\gamma : [a, b] \to \mathbb{C}$ the following holds:*

$$\int_\gamma f(z)dz = 0 \tag{9.16}$$

[2] a domain is a connected, open set; those, who have never heard of connectivity, can imagine this as a set, in which from every point to every other point there exists a path – despite this is called path connectivity and is stronger than sole connectivity, it is enough to get an idea of it.

This is the Cauchy integral theorem[3] and is something like the fundametal theorem of integral and differential calculus: if you have two paths from x to y you can put these two together to form a closed path, by altering the direction of one of those paths; so their integral is zero. Since the alternation means a change in sign[4], this means that the integrals along the two paths for each of them have the same value, and the integral itself does only depend on the starting and end point of integration.
At least if the function is holomorphic "enough" and the domain is convex – the later can be extend to so called simple connected domains, witch is roughly speaking connected and "without holes".

9.1.4 Cauchy integral formula and residue calculus

Next we will state the Cauchy integral formulas[5] and formulate the residue theorem, which will be the central tools used in solving Rice's integrals.

Theorem 9.4. *Let U be an open disc in a domain E with $\overline{U} \subset E$[6] and we denote a positivly oriented boundary of U with ∂U[7]. Let $f : E \to \mathbb{C}$ be holomorphic, then the following statement hold:*

$$\forall z \in U : \quad f(z) = \frac{1}{2\pi i} \int_{\partial U} \frac{f(\zeta)}{\zeta - z} d\zeta \tag{9.17}$$

$$\forall z \in U : \quad f^{(n)}(z) = \frac{n!}{2\pi i} \int_{\partial U} \frac{f(\zeta)}{(\zeta - z)^{n+1}} d\zeta \tag{9.18}$$

$$\forall z \in E \setminus \overline{U} : \quad 0 = \frac{1}{2\pi i} \int_{\partial U} \frac{f(\zeta)}{\zeta - z} d\zeta \tag{9.19}$$

There are some generalisations of the CIT and CIF for general paths (not only circles and points can be encircled more than once). Their mathematical exact presentation would need some unnecesary complicated definitions. The main result is however, that the circle in the CIF can be deformated "continously" arbitrarily as long as the path doesn't cross the singularity z. We can even encircle z more than once, but the integral will than be an integer multiple of integral, whose path does encircle z only once, according to the number how often and in what direction z is encircled.
Untill now we had only considered polynomial singularities with the CIF. But the concept of integrating around singularities can be extended.
As stated above every function that is holomorphic can be expressed as a powerseries. There exists a generalisation of this concept, which is called Laurent series. Laurent series have the form

$$\sum_{n=-\infty}^{\infty} a_n (z - z_0)^n \tag{9.20}$$

It can be shown that under certain conditions f can be expressed within some disc partialy containing E and whose boundary is in E entirely as a Laurent series. Here f must be defined on a domain E for every point z_0, for which a path in E exists that encircles z_0.
In the following we consider holomorphic functions f which have isolated singularities; this means roughly speaking, that the domain of definition doesn't have any holes beside some points and these points do not cluster.

[3] further denoted with CIT
[4] this can easily be checked by considering the substition formula
[5] further denoted with CIF
[6] $\overline{U}$ indicates the closure of U in the standard topology on $\mathbb{C}$
[7] a positivly oriented boundary of a disc is a path (this means a special curve and not a set) starting from one point at the boundary then encircles the disc counterclockwise untill it returns for the first time to the starting point – this means it is a closed path

Definition 9.7. The coefficient at $n = -1$ of a Laurent series of a holomorphic function f with isolated singularities of type (9.20) is called the residue at z_0: $\operatorname*{Res}_{z_0}(f) = a_{-1}$

It can be shown[8] that there exists an ϵ, such that a circle $U_\epsilon(z_0)$ with radius ϵ small enough and center z_0 has the property $\overline{U_\epsilon(z_0)} \setminus \{z_0\} \subset E$ and that

$$\operatorname*{Res}_{z_0}(f) = \frac{1}{2\pi i} \int_{\partial U_\epsilon(z_0)} f(z)dz \tag{9.21}$$

Taking into account the CIT this yields the residue theorem[9]:

Theorem 9.5. *Let E be an open set and U an open disc with $E \subset U$. Let, for some $n \in \mathbb{N}_0$, f be holomorphic on $E \setminus \{z_1, \ldots, z_n\}$. Then*

$$\frac{1}{2\pi i} \int_{\partial U} f(z)dz = \sum_{k=0}^{n} \operatorname*{Res}_{z_k}(f) \tag{9.22}$$

This is an extension of the CIT. It can even be extended to the case, where U is not a circle in U and even if U is not bounded, as long as the singularities are isolated in the encircled domain. This finishes our short introduction to complex analysis.

9.2 Other mathematical formulas

9.2.1 The Gamma function

The Gamma function is a generalisation of the factorial to complex numbers – one of its definitions is

$$\Gamma(s) := \int_0^\infty e^{-t} t^{s-1} dt \tag{9.23}$$

It satisfies the relations:

$$\forall n \in \mathbb{N}_0 : \Gamma(n) = (n-1)! \quad \forall x \in \mathbb{C} \setminus -\mathbb{N}_0 : \Gamma(x+1) = x\Gamma(x) \tag{9.24}$$

A direct consequence is:

$$\prod_{i=0}^{n} (s-i) = \frac{\Gamma(s+1)}{\Gamma(s-n)} \tag{9.25}$$

The Gamma function is holomorphic on its domain $\mathbb{C} \setminus -\mathbb{N}_0$; at the negativ integers it diverges.
An estimate for the growth of the Gamma function is the so called Stirling formula:

$$\Gamma(x) = \sqrt{2\pi} x^{x-\frac{1}{2}} e^{-x} \left(1 + O(\frac{1}{n})\right) \tag{9.26}$$

The following formula, which is a direct consequence of the Stirling formula is the standard estimate:

$$\frac{\Gamma(n+1)}{\Gamma(n+1-\alpha)} = n^\alpha \left(1 + O(\frac{|a|^2}{n})\right) \tag{9.27}$$

It can be shown that

$$\lim_{n\to\infty} \left(\sum_{i=1}^{n} \frac{1}{i} - \ln(n) \right) =: \gamma \approx 0.577 \tag{9.28}$$

[8] in fact, this is a direct consequence of the structure of the Laurent series and the fact that $\forall n \in \mathbb{Z} \setminus \{-1\} : \int_\gamma z^n = 0$ if γ is a closed path

[9] again the proof is quiet obvious and the idea simple, if you are used to complex analysis, but if exactly formulated very technical

This γ is known as Eulers constant.
Using this, another result is:

$$\frac{\Gamma'(x)}{\Gamma(x)} = -\gamma - \frac{1}{x} - \sum_{k=1}^{\infty}\left(\frac{1}{x+k} - \frac{1}{k}\right) \tag{9.29}$$

The proofs of all the statements above are wonderful perls of mathematical analysis. But as with all perls, you have to dive deep to find them, so the proofs are far from trivial and we omit them all. The Gamma function does have many other nice properties – but we won't need them.

9.2.2 Zeta functions and Modified Bell Polynomials

The so called incomplete Hurwitz ζ function is:

$$\zeta_n(r,\beta) = \sum_{i=0}^{n-1} \frac{1}{(i+\beta)^r} \tag{9.30}$$

$\zeta_n(r,1)$ defines the generalized harmonic numbers $\zeta_n(r)$ and their limit ($n \to \infty$) is the famous Riemann ζ function.
From (9.29) it follows, that

$$\zeta_{n+1}(1,\beta) = \ln(n) - \frac{\Gamma'(\beta)}{\Gamma(\beta)} + O(\frac{1}{n}) \tag{9.31}$$

The modified Bell polynomials $\mathbf{L_m} = \mathbf{L_m}(x_1, x_2, \ldots, x_m)$ are defined as

$$\exp\left(\sum_{k=1}^{\infty} x_k \frac{t^k}{k}\right) = 1 + \sum_{m=1}^{\infty} \mathbf{L_m} t^m \tag{9.32}$$

It is rather technical than difficult to proof that in general

$$\mathbf{L_m}(x_1, x_2, \ldots) = \sum_{1m_1+2m_2+\ldots=m} \frac{1}{m_1!m_2!\ldots} \left(\frac{x_1}{1}\right)^{m_1} \left(\frac{x_2}{2}\right)^{m_2} \ldots \tag{9.33}$$

and to get an idea of them, we have

$$\begin{aligned} \exp\left(\sum_{k=1}^{\infty} x_k \frac{t^k}{k}\right) &= 1 + x_1 t + \left(\frac{x_2}{2} + \frac{x_1^2}{2}\right) t^2 + \\ &\left(\frac{x_3}{3} + \frac{x_1 x_2}{2} + \frac{x_1^3}{6}\right) t^3 + \left(\frac{x_4}{4} + \frac{x_1 x_3}{3} + \frac{x_2^2}{8} + \frac{x_2 x_1^2}{4} + \frac{x_1^4}{24}\right) t^4 + \ldots \end{aligned} \tag{9.34}$$

9.3 Motivation and Basic Integrals

First we will introduce generalized differences for a sequence $\{f_k\}_{k\in\mathbb{N}_0}$:

$$\Delta f_n = f_{n+1} - f_n \qquad \Delta^n f_0 = \sum_{k=0}^{n} \binom{n}{k} (-1)^{n-k} f_k = (-1)^n D_n[f] \tag{9.35}$$

The differences D_n arise often in the average case analysis of some data structures as search trees or tries. As a naive bound we get:

$$|D_n[f]| \le 2^n \max_{0\le k\le n} |f_n| \tag{9.36}$$

But for many sequences f_n that come across in data structure analysis this bound is way to rough and polynomial bounds can be found – this phenomenon is called exponential cancelation.
In the analysis of recurrent sequences generating functions are often used to simplify and solve the relations. For example the exponential generating is

$$f(z) = \sum_{n=0}^{\infty} f_n \frac{z^n}{n!} \tag{9.37}$$

and Poisson generating function is defined by

$$\hat{f}(z) = \sum_{n=0}^{\infty} f_n e^{-z} \frac{z^n}{n!} \tag{9.38}$$

We consider the transform $f_n \mapsto g_n = D_n[f]$. Substitution in the exponetial generating function or the Poisson generating function respectively yields the equations

$$g(z) = e^z f(-z) \quad \hat{g}(z) = e^{-z} \hat{f}(-z) \tag{9.39}$$

So it can be supposed, that when these transforms induce drastical simplifications of recurences or difference equations, high order differences as D_n may play a significant role.
We assume in the following, that a holomorphic function $\phi(x)$ interpolates the values of the sequence f_n, which means $\forall k \in \mathbb{N}_0 : f_k = \phi(k)$.

Lemma 9.1. *Let ϕ be a holomorphic function in a domain that contains the half-line $[n_0, \infty[$ and $\mathcal{C}$ is a positivly oriented closed path in the domain of ϕ, which encircles $[n_0, n]$ and does not include any of the integers $0, 1, \ldots, n_0 - 1$ nor a point, where ϕ is not holomorphic. Then the following holds*

$$\sum_{k=n_0}^{n} \binom{n}{k} (-1)^k \phi(k) = \frac{(-1)^n}{2\pi i} \int_{\mathcal{C}} \phi(s) \frac{n!}{s(s-1)\ldots(s-n)} ds \tag{9.40}$$

Proof. We apply the residue theorem (9.22). Since the only points where the integrand is not holomorphic are the integers $n_0, n_0 + 1, \ldots, n$, we only have to consider these integers. Let k be such an integer; the residues can be evaluated according to the CIF (9.17). Then we have:

$$\operatorname*{Res}_{s=k} \phi(s) \frac{n!}{s(s-1)\ldots(s-n)} =$$

$$\operatorname*{Res}_{s=k} \frac{1}{s-k} \left(\phi(s) \frac{n!}{s(s-1)\ldots(s-k+1)(s-k-1)\ldots(s-n)} \right) = \phi(k) \frac{(-1)^{n-k} n!}{k!(n-k)!} \tag{9.41}$$

Simple summation while taking the sign into account yields the proposition. □

For the further discussion the definition of polynomial growth will be important:

Definition 9.8. A function ϕ in an unbounded domain Ω is said to have polynomial growth, if for some r the formula $|\phi(s)| = O(|s|^r)$ holds as $s \to \infty$ in Ω. r is called the degree of ϕ

If the function in (9.40) is of polynomial growth in the half-plain $\Re(s) > n_0 - \epsilon$ for some $\epsilon > 0$ and n is sufficiently large, then we have for some $n_0 > c > \max\{n_0 - \epsilon, n_0 - 1\}$ the representation

$$\sum_{k=n_0}^{n} \binom{n}{k} (-1)^k \phi(k) = -\frac{(-1)^n}{2\pi i} \int_{c-i\infty}^{c+i\infty} \phi(s) \frac{n!}{s(s-1)\ldots(s-n)} ds \tag{9.42}$$

Take as contour of integration the path

$$\gamma_j : \left[-j, j+\frac{1}{2}\right] \to \mathbb{C}; \begin{cases} c+ix & -j \le x \le j \\ c+je^{-2\pi i(\frac{1}{4}-x)} & j \le x \le j+\frac{1}{2} \end{cases}$$

where $j > n$. This path is negatively oriented (so the sign changes) and we can apply (9.40). Since for any allowed j (9.40) holds, we consider the limit $j \to \infty$. The first part of the path yields the integral in (9.42) and the second can be estimated as $O(|j|^{-n-1+r+1})$ using formula (9.14), since the integrand and therefor its maximum has asymtotical growth $O(|j|^{-n-1+r})$ and the length of the path is $\frac{2\pi j}{2}$. So if n is sufficiently large the second part of the integral vanishes.

9.4 Integrals of Functions with Poles and Representation of Sumes

9.4.1 Rational functions

Theorem 9.6. *Let ϕ be a rational function holomorphic in a domain that contains the half-line $[n_0, \infty[$. If n is big enough we have*

$$\sum_{k=n_0}^{n} \binom{n}{k} (-1)^k \phi(k) = -(-1)^n \sum_s \operatorname*{Res}_s \left(\phi(s) \frac{n!}{s(s-1)\dots(s-n)}\right) \tag{9.43}$$

where the sum is taken over all poles of ϕ and over $0, 1, \dots, n_0 - 1$

Proof. First we use Lemma 9.1 and take as path of integration a circle of radius R big enough to encircle all poles. When $R \to \infty$ and $n > r$ the integral on the right side of (9.40) tends to 0 by a similar argument used for (9.42). Applying once again the residue theorem (9.22), we find that the sum on the righthand side of (9.43) minus its lefthand side is 0, which directly yields (9.43) □

As a next step we try to express the residues. For this purpose we will need the incomplete Hurwitz zeta function and the modified Bell polynomials introduced in (9.30) and (9.33). As every rational function can be expressed as a linear combination of terms of the form $A(x-a)^{-r}$, where $r \in \mathbb{N}_0$, we only have to consider functions ϕ of this type.

Lemma 9.2. *When $\alpha \in \mathbb{C} \setminus \mathbb{N}_0$, then*

$$I_n(\alpha) = (-1)^n n! \operatorname*{Res}_{s=\alpha} \left(\frac{1}{(s-\alpha)^r} \frac{1}{s(s-1)\dots(s-n)}\right) \tag{9.44}$$

has the following asymptotic

$$I_n(\alpha) = -\Gamma(-\alpha) n^{\alpha} \frac{(\ln n)^{r-1}}{(r-1)!} \left(1 + O\left(\frac{1}{\ln n}\right)\right) \tag{9.45}$$

In the following we use the symbol $\langle\phi(s)\rangle_{k,\alpha}$ to denote the kth coefficient in the Laurent series at a certain point $\alpha \in \mathbb{C}$.

Proof.

$$
\begin{aligned}
I_n(\alpha) = &- n!\left\langle (s-\alpha)^r \frac{1}{(-s)(1-s)\dots(n-s)} \right\rangle_{-1,\alpha} = \\
&- n!\left\langle \frac{1}{(-s)(1-s)\dots(n-s)} \right\rangle_{r-1,\alpha} = \\
&- n!\left\langle \frac{1}{(-\alpha-s)(1-\alpha-s)\dots(n-\alpha-s)} \right\rangle_{r-1,0} = \\
&- n!\left\langle \exp\left(-\ln\left(\prod_{j=0}^{n}(j-\alpha-s)\right)\right) \right\rangle_{r-1,0} = \\
&- n!\left\langle \exp\left(-\sum_{j=0}^{n}\ln(j-\alpha-s)\right) \right\rangle_{r-1,0}
\end{aligned}
\tag{9.46}
$$

Since we have $\ln(j-\alpha-x) = \ln\left((j-\alpha)\left(1+\frac{-x}{j-\alpha}\right)\right) = \ln(j-\alpha)+\ln\left(1+\frac{-x}{j-\alpha}\right)$, we can apply the series expansion $\ln(1+x) = \sum_{m=1}^{\infty}(-1)^{m+1}\frac{x^m}{m}$ to get

$$
\begin{aligned}
&= -n!\exp\left(-\sum_{j=0}^{n}\ln(j-\alpha)\right)\left\langle \exp\left(\sum_{j=0}^{n}\left(\sum_{m=1}^{\infty}\frac{1}{m}\left(\frac{s}{j-\alpha}\right)^m\right)\right)\right\rangle_{r-1,0} = \\
&- n!\frac{1}{(-\alpha)(1-\alpha)\dots(n-\alpha)}\left\langle \exp\left(\sum_{m=1}^{\infty}\left(\sum_{j=0}^{n}\frac{1}{(j-\alpha)^m}\frac{s^m}{m}\right)\right)\right\rangle_{r-1,0} \overset{(9.30)}{=} \\
&- n!\frac{1}{(-\alpha)(1-\alpha)\dots(n-\alpha)}\left\langle \exp\left(\sum_{m=1}^{\infty}\zeta_{n+1}(m,-\alpha)\frac{s^m}{m}\right)\right\rangle_{r-1,0} \overset{(9.33)}{=} \\
&- \frac{\Gamma(n+1)\Gamma(-\alpha)}{\Gamma(n+1-\alpha)}\mathbf{L_{r-1}}(\zeta_{n+1}(1,-\alpha),\zeta_{n+1}(2,-\alpha),\dots\zeta_{n+1}(r-1,-\alpha)) \overset{(9.27)}{=} \\
&- \frac{\Gamma(n+1)\Gamma(-\alpha)}{\Gamma(n+1-\alpha)}\mathbf{L_{r-1}}(\ln n - \frac{\Gamma'(-\alpha)}{\Gamma(-\alpha)} + O(1/n),\zeta_{n+1}(2,-\alpha),\dots\zeta_{n+1}(r-1,-\alpha)) =
\end{aligned}
\tag{9.47}
$$

Since the incomplete Hurwitz zeta function fullfills $\zeta_n(r,\beta) = O(1)$ for $n\to\infty$ and $r\in\mathbb{N}\setminus\{1\}$ and since beside the first coefficients of the modified Bell polynomials $\mathbf{L_m}$ all coefficent are of degree smaller than m all other coefficient can be neglected.

$$
\begin{aligned}
&= -\frac{\Gamma(n+1)\Gamma(-\alpha)}{\Gamma(n+1-\alpha)}\frac{1}{(r-1)!}(\ln n - \frac{\Gamma'(-\alpha)}{\Gamma(-\alpha)} + O(1/n))^{r-1} = \\
&- \frac{\Gamma(n+1)\Gamma(-\alpha)}{\Gamma(n+1-\alpha)}\frac{(\ln n)^{r-1}}{(r-1)!}\cdot\left(1+O\left(\frac{1}{\ln n}\right)\right) \overset{(9.31)}{=} \\
&- \Gamma(-\alpha)n^\alpha\left(1-O\left(\frac{1}{n}\right)\right)\frac{(\ln n)^{r-1}}{(r-1)!}\cdot\left(1+O\left(\frac{1}{\ln n}\right)\right) = \\
&- \Gamma(-\alpha)n^\alpha\frac{(\ln n)^{r-1}}{(r-1)!}\cdot\left(1+O\left(\frac{1}{\ln n}\right)\right)
\end{aligned}
\tag{9.48}
$$

□

As a first example we analyze for an $m\in\mathbb{N}$ the asymtotic growth of the sum

$$
S_n(m) = \sum_{k=1}^{n}\binom{n}{k}\frac{(-1)^k}{k^m} \tag{9.49}
$$

Here we can use the function $\phi(s) = \frac{1}{s^m}$ to interpolate the sequence and we set $n_0 = 1$. We have only one pole not in $\{n_0, n_0+1, \ldots, n\}$, that is 0, which is of the order $m+1$. We have to modify our calculations yielding (9.2), since the pole is in $\mathbb{N}_0$:

$$\begin{aligned} S_n(m) = &-\operatorname*{Res}_{s=0}\left(\frac{1}{s^{m+1}}\frac{n}{s-n}\frac{n-1}{s-n+1}\cdots\frac{2}{s+2}\frac{1}{s+1}\right) = \\ &-\operatorname*{Res}_{s=0}\left(\frac{1}{s^{m+1}}\left(\left(1-\frac{s}{1}\right)\left(1-\frac{s}{2}\right)\cdots\left(1-\frac{s}{n}\right)\right)^{-1}\right) = \\ &-\left\langle\left(\left(1-\frac{s}{1}\right)\left(1-\frac{s}{2}\right)\cdots\left(1-\frac{s}{n}\right)\right)^{-1}\right\rangle_{m,0} \end{aligned} \tag{9.50}$$

Similar to the calculations taken out in (9.46) and (9.47) with the generalized harmonic numbers $\zeta_n(k)$ this is

$$-\left\langle \exp\left(\sum_{k=1}^{\infty}\zeta_n(k)\frac{s^k}{k}\right)\right\rangle_{m,0} \tag{9.51}$$

Now we can use once again the modified Bell polynomials and the facts, that $\zeta_n(k) = \zeta(k) + O\left(\frac{1}{n^{k-1}}\right)$ for $k \geq 2$ and $\zeta_n(1) = \ln(n) + \gamma + O(1/n)$, which follows from (9.31) with $\beta = 1$ and $\Gamma'(1) = \gamma\Gamma(1)$ to get:

$$\begin{aligned} -S_n(m) = &\sum_{1m_1+2m_2+\ldots=m}\frac{1}{m_1!m_2!\ldots}\left(\frac{\zeta_n(1)}{1}\right)^{m_1}\left(\frac{\zeta_n(2)}{2}\right)^{m_2}\left(\frac{\zeta_n(3)}{3}\right)^{m_3}\cdots = \\ &\left(1+O\left(\frac{1}{n}\right)\right)\sum_{1m_1+2m_2+\ldots=m}\frac{1}{m_1!m_2!\ldots}\cdot \\ &\left(\ln(n)+\gamma+O\left(\frac{1}{n}\right)\right)^{m_1}\left(\frac{\zeta(2)}{2}\right)^{m_2}\left(\frac{\zeta(3)}{3}\right)^{m_3}\cdots \end{aligned} \tag{9.52}$$

Since the $\zeta(k)$ are constants we have for a polynomial P_m of degree m the asymtotics

$$-S_n(m) = P_m(\ln(n)) + O\left(\frac{(\ln(n))^m}{n}\right) \tag{9.53}$$

Using the values of the ζ function, we get for the first values of m

$$-S_n(1) = \ln(n) + \gamma + O\left(\frac{1}{n}\right) \tag{9.54}$$

$$-S_n(2) = \frac{1}{2}(\ln(n))^2 + \gamma\ln(n) + \frac{\gamma}{2} + \frac{\pi^2}{12} + O\left(\frac{\ln(n)}{n}\right) \tag{9.55}$$

Moreover for $m = 1$ we get the exact result

$$\sum_{k=1}^{n}\binom{n}{k}\frac{(-1)^{k-1}}{k} = -S_n(1) = \zeta_n(1) \tag{9.56}$$

The above asymptotic equation can be generalized for $m \notin \mathbb{N}$, as we will see later. Another example is the sequence

$$T_n = \sum_{k=0}^{n}\binom{n}{k}\frac{(-1)^k}{k^2+1} \tag{9.57}$$

This sequence obeys the recurrence

$$T_0 = 1 \quad T_1 = \frac{1}{2} \quad T_n = \frac{n}{n^2+1}((2n-1)T_{n-1} - (n-1)T_{n-2}) \tag{9.58}$$

which seams hard to solve or even estimate by conventional methods.
Since the sequence underlaying this differences can be interpolated by $\phi(s) = (1 + s^2)^{-1} = \frac{1}{s-i} + \frac{1}{s+i}$, this allows us to directly applay (9.2) using trigonometric identities and the fact that $|\Gamma(z)| = |\Gamma(\overline{z})|$ to get

$$\Gamma(-i)n^i\left(1 + O\left(\frac{1}{n}\right)\right) + \Gamma(i)n^{-i}\left(1 + O\left(\frac{1}{n}\right)\right) =$$
$$(\Gamma(-i)e^{i\ln(n)} + \Gamma(i)e^{-i\ln(n)})\left(1 + O\left(\frac{1}{n}\right)\right) = \rho \cdot \cos(\ln(n) + \theta) + o(1) \quad (9.59)$$

for some θ and $\rho = 2\,|\Gamma(i)| = 2\sqrt{\pi/\sinh(\pi)} \approx 1.04313$.
This example shows, that complex poles introduce periodic behavior in the asymtotics of a sequence.

9.4.2 Meromorphic functions

Meromorphic functions are generalisations of rational function. Meromorphic functions are holomorphic on an certain domain except isolated singularities.

Theorem 9.7. *Let ϕ be a function holomorphic in a domain that contains the half-line $[n_0, \infty[$. If n is big enough we have*

1. *If ϕ is meromorphic on $\mathbb{C}$ and of polynomial growth, then*

$$\sum_{k=n_0}^{n} \binom{n}{k} (-1)^k \phi(k) = -(-1)^n \sum_s \operatorname*{Res}_s \left(\phi(s)\frac{n!}{s(s-1)\dots(s-n)}\right) \quad (9.60)$$

where the sum is taken over all poles of ϕ and over $0, 1, \dots, n_0 - 1$

2. *If ϕ is meromorphic on the half-plane defined by $\Re(s) \geq d$ for some $d < n_0$ and of polynomial growth in this set, then*

$$\sum_{k=n_0}^{n} \binom{n}{k} (-1)^k \phi(k) = -(-1)^n \sum_s \operatorname*{Res}_s \left(\phi(s)\frac{n!}{s(s-1)\dots(s-n)}\right) + O(n^d) \quad (9.61)$$

where the sum is taken over all poles but $n_0, n_0 + 1, \dots, n$

Proof. Since in both cases the function is meromorphic, the number of poles are countable. So in the first case we can find positively oriented, concentric circles γ_j whose radii tend to ∞ and do not come across any pole. In the second case we can find for any $\epsilon > 0$ a $d < d' < d + \epsilon$, such that the pathes defined by $[d' - iR_j, d' + iR_j]$ and the half circle with center d' and radius R_j don't cross a pole and R_j tends to infinity. Since d' is arbitrarily close to d, we can assume $d = d'$. Of course, if the theorem is used in practice, other types of paths can be used, when they have the essential properties stated and used here.
Now we integrate along these curves and get using the residue theorem similar to theorem 9.6 the results above. Since ϕ is of polynomial growth[10] we can use the arguments at (9.42) to get the asymtotics of the integrals over the "infinite" paths. In the first case we get 0 for n big enough to overwhelm the polynomial growth. In the second case the $O(n^d)$ comes from the integral along the parallel to the imaginary

[10] in fact, it is only necessary to have polynomial growth on the union of the paths of integration – so we don't have to bother about poles or even infinitly many poles on our compact set, because we just circumnavigate them; in the second case the polynomial growth is even only needed beside a compact set, because the path of integration is not allowed to cross a pole and then holomorphic functions do attain their maximum on a compact set, so they can be estimated by a constant, which is trivially of polynomial growth

axes, as this argument illustrates (It's not a proof – it's just a plausibility argument; we asume $|\phi(s)| = O(|s|^r)$ for an integer r):

$$\left|\frac{(-1)^n}{2\pi i}\int_{d-i\infty}^{d+i\infty}\phi(s)\frac{n!}{s(s-1)\dots(s-n)}ds\right| \overset{(9.13)}{\leq}$$

$$\frac{1}{2\pi}\int_{d-i\infty}^{d+i\infty}\left|\phi(s)\frac{n!}{s(s-1)\dots(s-n)}\right| ds \leq$$

$$\frac{1}{2\pi}\int_{d-i\infty}^{d+i\infty}|s|^r\left|\frac{n!}{s(s-1)\dots(s-n)}\right| ds =$$

$$\frac{1}{2\pi}\int_{d-i\infty}^{d+i\infty}\frac{|s|^r}{|s(s-1)\dots(s-r+1)\cdot(s-r)(s-r-1)|}\,|(n(n-1)\dots(n-d+1))|$$

$$\left|\frac{(n-d)(n-d-1)\dots(r+2-d)}{(n-s)(n-s-1)\dots(r+2-s)}\right| |(r+2-d)(r+1-d)(r-d)\dots 2\cdot 1|^{-1}\,ds \leq$$

$$O(n^d)\int_{d-i\infty}^{d+i\infty}\frac{1}{|s|^2}ds = O(n^d) \tag{9.62}$$

This rough approximation holds for $r+1-d<0$. For the other case, this argument is not applicable, although I think, that $O(n^d)$ holds even in this case. □

As our next example we want to analyze the recurrence relation

$$f_n = a_n + 2\sum_{k=0}^{n}\binom{n}{k}\frac{1}{2^n}f_k \tag{9.63}$$

Further we will assume that $a_0 = a_1 = 0$; this is without loss of generality. Then we use exponential generating function introduced in (9.37) to get

$$f(z) = \sum_{n=0}^{\infty} f_n\frac{z^n}{n!} = \sum_{n=0}^{\infty}\left(a_n + 2\sum_{k=0}^{n}\frac{1}{2^n}\binom{n}{k}f_k\right)\frac{z^n}{n!} =$$

$$\sum_{n=0}^{\infty}a_n\frac{z^n}{n!} + \sum_{n=0}^{\infty}\left(2\sum_{k=0}^{n}\frac{1}{2^n}\binom{n}{k}f_k\frac{z^n}{n!}\right) =$$

$$a(z) + 2\sum_{n=0}^{\infty}\left(\sum_{k=0}^{n}\frac{1}{2^{n-k}(n-k)!}\frac{f_k}{2^k k!}z^{k+(n-k)}\right) =$$

$$a(z) + 2\left(\sum_{n=0}^{\infty}\frac{1}{2^n}\frac{z^n}{n!}\right)\left(\sum_{n=0}^{\infty}\frac{f_n}{n!}\left(\frac{z}{2}\right)^n\right) = a(z) + 2e^{z/2}f\left(\frac{z}{2}\right) \tag{9.64}$$

This easily translates into the Poisson generating function via multiplication the whole equation by e^{-z} to get

$$\hat{f}(z) = \hat{a}(z) + 2\hat{f}\left(\frac{z}{2}\right) \tag{9.65}$$

so for the coefficients $\hat{f}_n = n!\langle\hat{f}(z)\rangle_{n,0}$ we have

$$\hat{f}_n = \hat{a}_n + 2\frac{1}{2^n}\hat{f}_n \Rightarrow \hat{f}_n = \frac{\hat{a}_n}{1-2^{1-n}} \tag{9.66}$$

Since the equality

$$f_n = \sum_{k=0}^{n}\binom{n}{k}\hat{f}_n \tag{9.67}$$

holds, we get the identity respecting $a_0 = a_1 = 0$ and hence $\hat{a}_0 = \hat{a}_1 = 0$

$$f_n = \sum_{k=0}^{n}\binom{n}{k}\frac{\hat{a}_k}{1-2^{1-k}} = \sum_{k=2}^{n}\binom{n}{k}\frac{\hat{a}_k}{1-2^{1-k}} \tag{9.68}$$

To proof (9.67) we first show:

$$\hat{f}_n = n!\langle \hat{f}\rangle_{n,0} = n!\left\langle \sum_{n=0}^{\infty} f_n e^{-z}\frac{z^n}{n!}\right\rangle_{n,0} = n!\left\langle \sum_{n=0}^{\infty} f_n \sum_{k=0}^{\infty}(-1)^k \frac{z^k}{k!}\frac{z^n}{n!}\right\rangle_{n,0} =$$

$$n!\left\langle \sum_{i=0}^{\infty}\sum_{j=0}^{i}(-1)^j \frac{1}{j!} f_{i-j}\frac{z^j z^{i-j}}{(i-j)!}\right\rangle_{n,0} = n!\left\langle \sum_{n=0}^{\infty}\sum_{k=0}^{i}\frac{1}{k!}\frac{1}{n-k!}(-1)^k f_{n-k} z^n\right\rangle_{n,0} =$$

$$(-1)^n\left\langle \sum_{n=0}^{\infty}\sum_{k=0}^{i}\frac{n!}{k!(n-k)!}(-1)^k f_k z^n\right\rangle_{n,0} = (-1)^n \sum_{k=0}^{i}\binom{n}{k}(-1)^k f_k \tag{9.69}$$

Next we examine the righthand side of (9.67):

$$\sum_{k=0}^{n}\binom{n}{k}\hat{f}_k = \sum_{k=0}^{n}\binom{n}{k}(-1)^n\sum_{j=0}^{k}\binom{n}{j}(-1)^j f_j = \sum_{l=0}^{n}\sum_{m=l}^{n}(-1)^{l+m}\binom{n}{m}\binom{m}{l} f_l \tag{9.70}$$

Since we want this sum to be f_n, we have to show, that the inner sum is $\delta_{l,n}$[11]

$$\sum_{m=l}^{n}(-1)^{l+m}\binom{n}{m}\binom{m}{l} = \sum_{m=l}^{n}(-1)^{l+m}\frac{n!}{m!(n-m)!}\frac{m!}{l!(l-m)!} =$$

$$\sum_{m=l}^{n}(-1)^{l+m}\frac{n!}{(n-m)!(l-m)!l!} = \frac{n!}{l!}\sum_{m=l}^{n}(-1)^{l+m}\frac{1}{(n-m)!(l-m)!} =$$

$$\frac{n!}{l!}\sum_{k=0}^{n-l}(-1)^{2l+k}\frac{1}{(n-l-k)!k!} = \frac{n!}{l!(n-l)!}\sum_{k=0}^{n-l}(-1)^k\frac{(n-l)!}{(n-l-k)!k!} =$$

$$\binom{n}{l}\sum_{k=0}^{n-l}\binom{n-l}{k}(-1)^k(1)^{n-l-k} = \binom{n}{l}(1+(-1))^{n-l} = \binom{n}{l}\delta_{l,n} = \delta_{l,n} \tag{9.71}$$

Since (9.63) appears in the analysis of tries, the asymptotics of

$$U_n = \sum_{k=2}^{n}\binom{n}{k}\frac{\hat{a}_k}{1-2^{1-k}} \tag{9.72}$$

are of great interest. The $\hat{a}_k$ are usualy simple; for $n \geq 2$ we get for the $a_n = n-1$, which appears taking a closer look to tries, $\hat{a}_n = (-1)^n$. So we can apply theorem 9.7 with $f_k = (2^{k-1}-1)^{-1}$. We have infinitly many poles at $\chi_k = 1 + \frac{2\pi i k}{\ln 2}$.
We choose as path of integration circles centered at the origin, that avoid the poles and let the radius tend to ∞. The whole analysis is carried out in Knuth's "Art of computer programming"; we don't carry it out here, but the result is:

$$U_n = \frac{n}{\ln 2}\left(\ln(n) + \gamma - 1 - \frac{\ln 2}{2} + \sum_{k\in\mathbb{Z}\setminus\{0\}}\Gamma\left(-1-\frac{2\pi i k}{\ln 2}\right)e^{2\pi i k\ln(n)/\ln 2}\right) + O(1) \tag{9.73}$$

Since the Γ function decreases rapidly along the imaginary axes the effects of the sum can almost be neglected and we get the simplyfied asymptotic

$$n\log_2(n) + nP(\log_2(n)) + O(1) \tag{9.74}$$

More generally regular spaced poles introduce disturbances, which asymptoticaly behave like Fourier series in $\ln(n)$, as seen her with P.

[11] This is the widly used Dirac δ symbol

Another example is the sum $V_n = \sum_{k=1}^{n-1} \binom{n}{k} \frac{B_k}{2^k-1}$, which arises in the analysis of Patricia tries. Since $B_k = 0$ for $k \in 2\mathbb{N}+1$, the sign is not necessary, and using $B_k = -k\zeta(1-k)$ for $k \in 2\mathbb{N} \cup \{1\}$, we have to analyze the integral

$$V_n = \frac{(-1)^n}{2\pi i} \int_{1/2-i\infty}^{1/2+i\infty} \frac{n!}{(s-1)(s-2)\dots(s-n)} \frac{\zeta(1-s)}{2^s-1} ds \tag{9.75}$$

The path of integration is the infinite rectangle from $\frac{1}{2}-\infty$ to $\frac{1}{2}+\infty$ and from $n-\frac{3}{4}+\infty$ to $n-\frac{3}{4}-\infty$; but it can be shown that the integral of the second path is identical 0 for each n and so the rectangle can be extended to ∞ therefor is a variant of the second part of theorem 9.7, where the residues are not yet evaluated.
Now we have to consider the double pole at 0 (from the ζ function and from $(2^s-1)^{-1}$) and all the simple poles at $\chi_k = 2\pi ik/\ln(2)$, which yields analog to the example befor

$$V_n = \frac{\ln(n)}{\ln(2)} - \frac{1}{2} - \frac{1}{\ln(2)} \sum_{k \in \mathbb{Z}\setminus\{0\}} \zeta(1-\chi_k)\Gamma(1-\chi_k)e^{2\pi ik\ln(n)/\ln(2)} + O(1) \tag{9.76}$$

There are also some other examples, where Rice's integrals can be used succesfully; for example digital trees or quad trees used for multidimensional searching.
In the last example the extrapolating function was just given to us, but you can imagine that this would otherwise be a difficult task – especially with such not everyday occuring numbers as the Bernoulli numbers. If we have coefficients which are sums or products of other sequence α_k and we can interpolate the elements of this sequence by $\alpha(s)$, then we can use

$$A_n = \prod_{k=1}^{n} \alpha_k \Rightarrow A(s) = \prod_{k=1}^{\infty} \frac{\alpha(k)}{\alpha(k+s)} \quad A_n = \sum_{k=1}^{n} \alpha_k \Rightarrow A(s) = \sum_{k=1}^{\infty} (\alpha(k)-\alpha(k+s)) \tag{9.77}$$

9.4.3 Functions with Algebraic and Logarithmic Singularities

Now we turn to general algebraic and logarithmic functions. The problem with these function is that they can not to defined on $\mathbb{C}$, even not with some pointwise exceptions, but only with some uncountable exceptions. For example the complex extension of the logarithm and the root are defined on $\mathbb{C} \setminus]-\infty, 0]$[12]. For this reason we can not simply integrate around the points, where the functions are not defined, because every circle would be "slashed", but we have to use the so called Hankel contours to get our integration done.
Because of the difficulties that arise with this class of functions, we will only consider examples; but the first in more detail. Since we are familiar with the sequence $\frac{1}{k^m}$ from (9.49), where m was an integer, we try to generalise this to arbitrary λ, as we indicated before.

Theorem 9.8. *For any nonintegral λ, the sum*

$$S_n(\lambda) = \sum_{k=1}^{n} \binom{n}{k} (-1)^k k^{-\lambda} \tag{9.78}$$

has an asymptotic expansion in descending powers of $\ln(n)$ *of the form*

$$-S_n(\lambda) = (\ln(n))^{\lambda} \sum_{j=0}^{\infty} (-1)^j \frac{\Gamma^{(j)}(1)}{j!\Gamma(1+\lambda-j)} \frac{1}{(\ln(n))^j} \tag{9.79}$$

[12] People having done some complex analysis know, that there are some means to make this restriction a little bit more flexible, but you will never get rid of a malicious slash, which cuts into the complex plane

Proof. As seen in (9.50) we can represent the sum as an integral in an analoge way

$$S_n(\lambda) = \frac{1}{2\pi i}\int_{\mathcal{C}} \omega_n(s)\frac{ds}{s^{\lambda+1}} \text{ with } \omega_n(s) = \left(\left(1-\frac{s}{1}\right)\left(1-\frac{s}{2}\right)\dots\left(1-\frac{s}{n}\right)\right)^{-1} \tag{9.80}$$

Here $\mathcal{C}$ may be the vertical line $\Re(s) = \frac{1}{2}$ as in (9.42). Despite all, these integrals would be hard to evaluate; since we are only interested to encircle the poles (here only $1, 2, \dots, n$) we can deformate the path of integration as we like unless we don't encounter any new pole or the slash. For this reason we introduce the Hankel contour:

$$\begin{aligned}
\mathcal{C} &= \mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_3 + \mathcal{C}_4 \\
\mathcal{C}_1 &= \left\{ s \,\middle|\, |s| = R \wedge \left(|\Im(s)| \geq \frac{1}{\ln(n)} \vee \Re(s) > 0 \right) \right\} \\
\mathcal{C}_2 &= \left\{ s \,\middle|\, s = \frac{i-t}{\ln(n)} \wedge t \geq 0 \wedge |s| \leq R \right\} \\
\mathcal{C}_3 &= \left\{ s \,\middle|\, s = \frac{e^{\theta i}}{\ln(n)} \wedge \theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \right\} \\
\mathcal{C}_4 &= \left\{ s \,\middle|\, s = \frac{-i-t}{\ln(n)} \wedge t \geq 0 \wedge |s| \leq R \right\}
\end{aligned}$$

This is a circle of radius $R > n$ around the origin, that circumvents the slash by leaving some space around it; but as n increases the slash is left less space, so that if n tends to ∞ any point in the domain of $s^{-\lambda}$ will be encircled.
Now we split the integral in three parts $S_n(\lambda) = J_1 + J_2 + J_3$; we will see that for the asymptotic J_1 and J_2 can be neglected.

1. Let J_1 be the part, that belongs to the outer "circle" $\mathcal{C}_1$. Of course $s^{-\lambda-1}$ is of polynomial growth, so we can once more use our estimate for polynomial growth, to see that J_1 is $O(R^{-n-\lambda})$ and in the end, when $R \to \infty$, we get $J_1 = 0$

2. Now we will estimate the parts of $\mathcal{C}_2$ and $\mathcal{C}_4$ with $\Re(s) < -\frac{1}{\sqrt{\ln(n)}} =: -t_0$. For simplification, we forget about the term $s^{-\lambda-1}$ and asume we integrate along the negative real axes till we reach $-t_0$[13]. This simplifications do not touch the character of our estimate, but make it simpler. Additionaly, we mirror everything on the imaginary axes, to get:

$$\mu(n) := \int_{t_0}^{\infty} \frac{dt}{(1+\frac{t}{1})\dots(1+\frac{t}{n})} \tag{9.81}$$

As next step we split the new path into the intervalls $[t_0, 1]$, $\left[1, n^{1/3}\right]$ and $\left[n^{1/3}, \infty\right]$, to get the integrals $\mu_1(n)$, $\mu_2(n)$ and $\mu_3(n)$

[13]We asume $\lambda + 1 \geq 0$ and then it is quiet obvious, that we can neglect the powers of s for the asymptotic analysis, because they would make the asymptotic only smaller; but for example if λ are negative integers we get the Stirling numbers of second kind, which have normaly exponential growth – our derivation can not be applayed in this case

(a)

$$\mu_1(n) = \int_{t_0}^{1} \frac{dt}{(1+t/1)\dots(1+t/n)} = \int_{t_0}^{1} \frac{n!dt}{(1+t)(2+t)\dots(n+t)} =$$
$$\int_{t_0}^{1} \frac{\Gamma(n+1)\Gamma(t+1)}{\Gamma(n+t+1)} dt \overset{(9.26)}{=}$$
$$\int_{t_0}^{1} \frac{\sqrt{2\pi}(n+1)^{n+1-1/2}e^{-n-1}\sqrt{2\pi}(t+1)^{t+1-1/2}e^{-t-1}}{\sqrt{2\pi}(n+t+1)^{n+t+1-1/2}e^{-n-t-1}} \left(1+O\left(\frac{1}{n}\right)\right) dt \leq$$
$$\int_{t_0}^{1} \frac{\sqrt{2\pi}(t+1)^{t+1-1/2}}{e(n+t+1)^t} \left(1+O\left(\frac{1}{n}\right)\right) dt = O(1) \int_{t_0}^{1} \frac{(t+1)^{t+1-1/2}}{(n+t+1)^t} dt =$$
$$O(1) \int_{t_0}^{1} \frac{1}{(n)^t} dt \qquad (9.82)$$

The last few transformation can be carried out because $t \in [t_0, 1]$, so is small compared to n and can be ignored or estimated by a constant respectivly.

$$O(1) \int_{t_0}^{1} \frac{1}{(n)^t} dt = O(1) \left(\frac{-1+n^{1-t_0}}{n\ln(n)}\right) = O(e^{-t_0 \ln(n) + \ln(\ln(n))})$$
$$= O(e^{-1/2\sqrt{\ln(n)}}) \qquad (9.83)$$

(b) In the next part we use, that $1 \leq t$

$$\mu_2(n) = \int_{1}^{n^{1/3}} \frac{n!}{(1+t)\dots(n+t)} dt =$$
$$\int_{1}^{n^{1/3}} 1 \cdot 2 \cdot \frac{3}{2+t} \frac{4}{3+t} \cdots \frac{n}{n-1+t} \frac{1}{(1+t)(n+t)} dt \leq$$
$$\int_{1}^{n^{1/3}} O(1) \frac{1}{(1+t)(n+t)} dt = O\left(\frac{\ln(n^{1/3}+n)}{n-1}\right) = O(n^{-2/3}) \qquad (9.84)$$

(c) Similar to the estimates done in the formula above we get, for n large enough

$$\mu_3(n) = \int_{n^{1/3}}^{\infty} \frac{1}{(1+t/1)\dots(1+t/n)} dt \leq$$
$$\int_{n^{1/3}}^{\infty} \frac{1}{e^{(1/2)t}} dt = O(e^{-(1/2)n^{1/3}}) \qquad (9.85)$$

In the whole we have the result $J_2 = O(e^{(1/2)\sqrt{\ln(n)}})$, so it is of smaller order than any negativ power of $\ln(n)$

3. Now we have to estimate J_3; this is the integral along the contour of $\mathcal{C}_2 \cup \mathcal{C}_3 \cup \mathcal{C}_4$, for which $\Re(s) \geq -t_0$, denoted in the following by $\mathcal{C}^0$.
Again we use Stirlings formula (9.26) to get the asymptotic:

$$\omega_n(s) = n^s \Gamma(1-s) \left(1 + O\left(\frac{\ln(n)}{n}\right)\right) \qquad (9.86)$$

Since s is very small on $\mathcal{C}^0$ the part with $O\left(\frac{\ln(n)}{n}\right)$ can be estimated easily by $O(e^{(1/2)\sqrt{\ln(n)}})$ and this yields

$$J_3 = J^0 + O(e^{(1/2)\sqrt{\ln(n)}}) \text{ with } J^0 = \frac{1}{2\pi i} \int_{\mathcal{C}^0} n^s \Gamma(1-s) \frac{ds}{s^{\lambda+1}} \qquad (9.87)$$

Now we use the transformation $z = s\ln(n)$ with $\mathcal{D}^0$ is the image under this transform of $\mathcal{C}^0$ and have

$$J^0 = (\ln(n))^\lambda \frac{1}{2\pi i}\int_{\mathcal{D}^0} e^z \Gamma\left(1 - \frac{z}{\ln(n)}\right)\frac{dz}{z^{\lambda+1}} \tag{9.88}$$

By the transform we get $|z| = O(\sqrt{\ln(n)})$ on $\mathcal{D}^0$; this is the reason, we can expand the Gamma function around 1 in a power series and after changing summation and integration, we get

$$J^0 = (\ln(n))^\lambda \sum_{m=0}^{\infty}(-1)^m \frac{\Gamma^{(m)}(1)}{m!}\frac{1}{(\ln(n))^m}\frac{1}{2\pi i}\int_{\mathcal{D}^0} e^z z^{m-\lambda-1}dz \tag{9.89}$$

Now we have to estimate the remaining integrals; this can be done by the so called Laplace method, where we extend the contour to $-\infty$ to get $\mathcal{L}$. We will only present the result here:

$$\frac{1}{2\pi i}\int_{\mathcal{L}} e^z z^{m-\lambda-1}dz = \frac{1}{\Gamma(1-m+\lambda)} \tag{9.90}$$

Now, taking together all the parts, we have prooven the theorem. □

As an application, we can examine the sum

$$X_n = \sum_{k=1}^{n}\binom{n}{k}\frac{(-1)^k}{\sqrt{1+k^2}} \tag{9.91}$$

We have the local behvior $(s \pm i)^{-1/2}$ at the "problem points", so we get an asymptotic growth of $\sqrt{\ln(n)}$. Similar to (9.59) we have for some ρ and θ_0

$$X_n = \rho\sqrt{\ln(n)}\cos(\ln(n) + \theta_0) + O((\ln(n))^{-1/2}) \tag{9.92}$$

Other examples and direct applications are

$$-S_n(-1/2) = \frac{1}{\sqrt{\pi\ln(n)}} - \frac{\gamma}{2\sqrt{\pi(\ln(n))^3}} + O((\ln(n))^{-5/2}) \tag{9.93}$$

$$-S_n(1/2) = 2\sqrt{\frac{\ln(n)}{\pi}} + \frac{\gamma}{\sqrt{\pi\ln(n)}} + O((\ln(n))^{-3/2}) \tag{9.94}$$

In general the coefficients are rational expressions of terms as γ, $\Gamma(-\lambda)$ and $\zeta(2)$, $\zeta(3)$, ...
For the rest of this part, we will only state some more examples, where the method of Rice's integrals (perhaps with use of the Hankel contour) can be applied succesfully.

Theorem 9.9. *For the logarithmic differences we have the asymptotics*

$$Y_n = \sum_{k=1}^{n}\binom{n}{k}(-1)^k\ln(k) = \ln(\ln(n)) + \gamma + \frac{\gamma}{\ln(n)} - \frac{\pi^2+6\gamma^2}{12(\ln(n))^2} + O\left(\frac{1}{(\ln(n))^3}\right) \tag{9.95}$$

The method can also be used for entire functions, which have no poles at all (despite the artificial ones introduced by the kernel ...). For example for

$$Z_n = \sum_{k=0}^{n}\binom{n}{k}\frac{(-1)^k}{k!} \tag{9.96}$$

which obeys the recurrence $Z_{n+2} = (2-2/n)Z_{n+1} + (1-1/n)Z_n$ can be extrapolated by the entire function $1/\Gamma(s)$, and after carrying out Rice's method it reveals for some constants c and θ

$$Z_n = cn^{-1/4}\sin(2n^{1/2}+\theta) + o(n^{-1/4}) \tag{9.97}$$

We have seen many cases were heavy use of complex analysis can resolve the asymptotics of recurrences, generalized differences and sums. We summarize all this in the table below.

Some types of singularities and the asymptotics they introduce in the corresponding difference

singularity	asymptotics
singularity of $\phi(s)$ at $s_0 = \sigma_0 + i\tau$	approximately $n^{s_0} = n^{\sigma_0}e^{i\tau_0 \ln(n)}$
simple pole: $(s-s_0)^{-1}$	$-\Gamma(-s_0)n^{s_0}$
multiple pole: $(s-s_0)^{-r}$	$-\Gamma(-s_0)n^{s_0}\frac{(\ln(n))^{r-1}}{(r-1)!}$
algebraic singularity: $(s-s_0)^{\lambda}$	$-\Gamma(-s_0)n^{s_0}\frac{(\ln(n))^{-\lambda-1}}{\Gamma(-\lambda)}$
logarithmic singularity: $(s-s_0)^{\lambda}(\ln(s-s_0))^r$	$-\Gamma(-s_0)n^{s_0}\frac{(\ln(n))^{-\lambda-1}}{\Gamma(-\lambda)\ln(\ln(n))^r}$

9.5 Mellin Transforms and Rice's Integrals

The Mellin transform of a function and its inverse have the form

$$\phi(z) = \int_0^\infty t^{z-1}f(t)dt \qquad f(t) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} t^{-z}\phi(z)dz \tag{9.98}$$

If we take a closer look to the integral (9.42) and take into account the asymptotic of $\omega_n(s)$ (9.86)[14] we get:

$$\frac{1}{2\pi i}\int_{d-i\infty}^{d+i\infty}\phi(s)\frac{(-1)^n n!}{s(s-1)\dots(s-n)}ds \approx \frac{1}{2\pi i}\int_{d-i\infty}^{d+i\infty}\phi(s)\Gamma(-s)n^s ds \tag{9.99}$$

If we would change the sign of the variable and then compare the result with the inverse Mellin transform we observe a definite analogy. Without stating it formally, in cases, were we want to evaluate Rice's integrals and we get stuck with it, it can be worth a try to evaluate the corresponding inverse Mellin transform to get an idea about the size of the asymptotics.

But the similarity can be stated formally as the Poisson-Mellin-Newton cycle.

Theorem 9.10. *The coefficients of a Poisson generating function are expressible as a Rice's integral of a Mellin transform of the Poisson generating function.*

$$\{f_n\} \xrightarrow{\textit{Poisson GF}} \hat{f}(t) = \sum_{n=0}^{\infty} f_n e^{-t}\frac{t^n}{n!} \xrightarrow{\textit{Mellin transform}} \hat{f}^*(s) = \int_0^\infty \hat{f}(t)t^{s-1}dt \xrightarrow{\textit{Rice's integral}} \{f_n\} \tag{9.100}$$

Proof. We take a closer look to the Mellin transform and state a Newton series to have

$$\hat{f}^*(s) = \int_0^\infty \hat{f}(t)t^{s-1}dt = \sum_0^\infty \frac{f_n}{n!}\int_0^\infty e^{-t}t^{s+n-1}dt \overset{(9.23)}{=} \Gamma(s)\left(f_0 + f_1\frac{s}{1!} + f_2\frac{s(s+1)}{2!} + \dots\right) \tag{9.101}$$

[14] don't forget about the additional s in the denominater, so we realy get $\Gamma(-s)$ and not $\Gamma(1-s)$, after changing sign

By differencing[15] we get to the following formula

$$f_n = \sum_{k=0}^{n} \binom{n}{k} (-1)^k \frac{\hat{f}^*(-s)}{\Gamma(-s)} \tag{9.102}$$

But these differences are exactly of the Rice type, so we conclude the corresponding equations

$$f_n = \frac{(-1)^n}{2\pi i} \int_{\mathcal{C}} \left(\frac{\hat{f}^*(s)}{\Gamma(-s)} \frac{n!}{s(s-1)\dots(s-n)} \right) ds \tag{9.103}$$

$$\hat{f}^*(s) = \int_0^\infty \left(e^{-t} \sum_{n=0}^{\infty} f_n \frac{t^n}{n!} \right) t^{s-1} dt \tag{9.104}$$

This are the relations, which were stated. □

For example if we carry out the Mellin transform of the Poisson generating function (9.65) we have $\hat{f}^*(s) = \frac{\hat{a}^*(s)}{1-2^{1+s}}$ and from the formulas above we get the result

$$f_n = \sum_{k=0}^{n} \binom{n}{k} \frac{\hat{a}^*(-k)}{\Gamma(k)} \frac{(-1)}{1-2^{1-k}} \tag{9.105}$$

This is formally the same result as if we had carried out the Rice's method.
There are some other examples as digital search trees, were the Poisson-Mellin-Newton cycle can be applied. This is a hint that the formal result we get from the cycle can be developed to an actual result by the Rice's integrals. The cycle is also an explanation for some other phenomena, but this would lead too far here.

9.6 Summary

Despite complex analysis seems to be part of pure mathematics, it can be applied for finding asymptotics of solutions of difference equations and generalized differences, which are needed in the analysis of algorithms. By the method of Rice's integrals we can tackle the average case analysis of tries, digital search trees, multidimensional searching and other datastructures and algorithms of great practical use. In most cases a detailed calculation of the asymptotics is in fact much too complex, but you can get a first estimate of the growth by comparing the problem with the examples outlined here and the table given at the end of the section befor the last.

[15] normaly we would differenciate, but here we have not a series in powers of s like s^n but a series in $s(s+1)(s+2)\dots$; since differencing leads to success with $s(s-1)(s-2)\dots$, we have to play a little bit with the sign and get a result

Chapter 10

Digital Search Trees Average case analysis of digital search trees and tries

Nicolai Baron von Hoyningen-Huene

10.1 Introduction

A very common problem in computer science is to search for the appearance of a string inside a text. There exist algorithms that use suffix trees, which are often implemented as digital trees to optimize the performance of the search. In this article some properties of those data structures are analyzed, which can be used to calculate the average performance and space complexity of algorithms using digital trees. For an introduction to Rice's method (see Chapter 9) is recommended, but the mathematical derivation in this article is largely based on [FS86b].
In the first part basic terms and structures are defined. Subsequently there will be a detailed analysis of the internal path length and external internal nodes for digital search trees. The following derivation for properties of tries are just sketched because of similarity to the precedent part. Then the binary trees are generalized to M-ary trees and examined. Concluding, a general framework for analysis of properties of digital trees is presented.

10.2 Trees

First of all we look at rudimentary definitions for the sake of completeness and later usage, the definitions are taken from [CLR90]. In computer science data has to be stored in an intelligent way. Often there exist keys which are related to data, thus a special data structure is needed.

Definition 10.1. A **dictionary** is defined as an abstract data type storing items, or keys, associated with values. Basic operations are insert, find, and delete.

The operations new(), insert(i, v, D), and find(i, D) may be defined as follows:

- *new* () returns a dictionary

- $find\,(i, insert\,(i, v, D)) = v$
  $find\,(i, insert\,(j, v, D)) = find\,(i, D)$ if $i \neq j$ where i and j are items or keys, v is a value, and D is a dictionary. The operation $find\,(i, new\,())$ is not defined.
- The modifier function $delete\,(i, D)$ may be defined as follows.
  $delete\,(i, new\,()) = new\,()$
  $delete\,(i, insert\,(i, v, D)) = delete\,(i, D)$
  $delete\,(i, insert\,(j, v, D)) = insert\,(j, v, delete\,(i, D))$ if $i \neq j$

We can define $find\,(i, new\,())$ using a special value: $fail$. This only changes the return type of $find$. $find\,(i, new\,()) = fail$
A tree is a commonly used structure for implementation of dictionaries:

Definition 10.2. A **tree** is defined as a data structure accessed beginning at the root node. Each node is either a leaf or an internal node. An internal node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings. Contrary to a physical tree, the root is usually depicted at the top of the structure, and the leaves are depicted at the bottom.

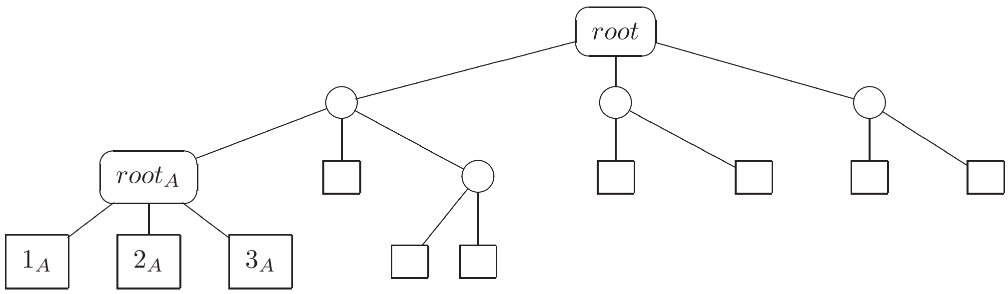


Figure 10.1: Example for a trinary tree

And we have a special property of a tree:

Definition 10.3. The **depth** of a node in a tree is the distance from this node to the root of the tree.

We can constrain trees to optimize performance for some purpose:

Definition 10.4. A **search tree** is a tree where every subtree of a node has values less than any other subtree of the node to its right. The values in a node are conceptually between subtrees and are greater than any values in subtrees to its left and less than any values in subtrees to its right.

In the binary world of a computer, trees with binary properties are widely supported:

Definition 10.5. A **binary tree** is either empty (no nodes), or has a root node, a left binary tree, and a right binary tree.

10.3 Digital Search Trees

10.3.1 Data Structure of Binary Search Trees

Definition 10.6. A **binary search tree** is a binary tree and also a search tree. A new node is added as a leaf.

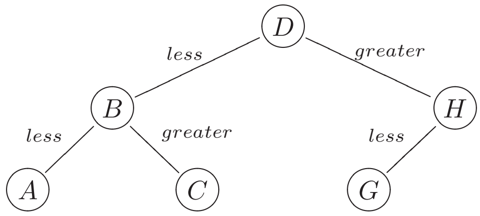


Figure 10.2: Example for a binary search tree with lexical ordering

The worst case for search is in the order of the number of keys N stored in a binary search tree, since the tree can be degenerated to a linear list. This arise, when keys are inserted in a a- or descending order. Instead the average case for successful search in a binary search tree is logarithmic, because the worst case is very unlikely. It evaluates to

$$2\left(1+\frac{1}{N}\right)H_N-3=(2\ln 2)\lg N+2\gamma-3+O\left(\frac{\log N}{N}\right).$$

10.3.2 Data Structure of Digital Search Trees

Keys are always handled by a computer as binary data. Therefore we can use the digital properties of the keys.

Definition 10.7. A **digital tree** is a tree for storing a set of strings where nodes are organized by substrings common to two or more strings.

The ordering of the keys is intuitively: we follow the tree by the bits descending from the first bit of the key represented as a binary number until we come to a leaf, a zero directs us to the left, a one to the right. In Figure 10.3 and 10.4 you can see an example tree, the binary coding of each letter is written next to it. Note, that the structure of the digital search tree depends of the order of the input of the keys. We define N as the number of keys stored in this dictionary. The number of nodes is limited by the number of bits in the keys and larger than $\lg N$ but likely less than a constant factor for many natural situations.

Definition 10.8. A **digital search tree** is a dictionary implemented as a digital tree which stores keys in internal nodes, so there is no need for extra leaf nodes to store the keys.

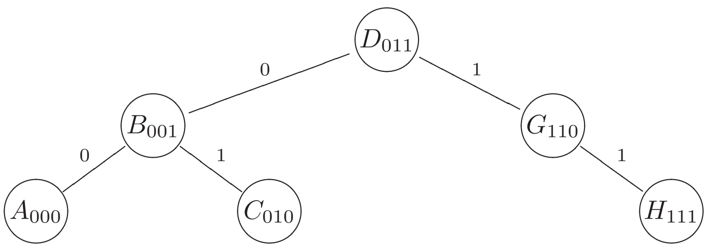


Figure 10.3: Example of a digital search tree with internal path length = 8

The worst case is the same as for binary search trees in a similar pathological case. But the average case for successful search in a digital search tree is improved:

$$\lg N+\frac{\gamma-1}{\ln 2}+\frac{3}{2}-\alpha+\delta(N)+O\left(\frac{\log N}{N}\right).$$

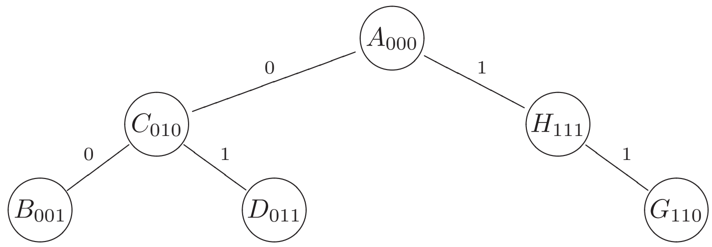


Figure 10.4: Another example of a digital search tree with same keys

10.3.3 Internal Path Length

In this chapter the first property of a digital search tree is analyzed.

Definition 10.9. The internal path length of a tree is the sum of the depth of every node of the tree.

This property is directly related to the complexity of the data structure: The number of nodes examined during a successful search in a search tree with N nodes is the path length of this node and in average case this counts as one plus the internal path length normalized through division by N.

Let A_N be the average internal path length of a digital search tree built from N (sufficiently long) keys comprised of random bits. Then we have the fundamental recurrence relation

$$A_N = N - 1 + \sum_{k=0}^{\infty} \frac{1}{2^{N-1}} \binom{N-1}{k} \left(A_k + A_{N-1-k}\right), \qquad N \geq 1 \tag{10.1}$$

with $A_0 := 0$.

The internal path length of any tree of N nodes is the sum of the internal path lengths of the subtrees of the node plus $N-1$, that are the ones missing for the distance from each node to the root of the whole tree. We count for each possible partition of the nodes to both subtrees and weight the sum by the number of all possibilities. The subtrees are randomly built.

We strike now for the goal to approximate A_N to get an explicit useful term. By symmetry A_k is equal to A_{N-1-k} and we get

$$A_N = N - 1 + \sum_{k=0}^{\infty} \frac{1}{2^{N-1}} \binom{N-1}{k} \left(A_k + A_k\right)$$

This equation is transformed into a functional one on the exponential generating function $A(z) = \sum_{N=0}^{\infty} A_N \frac{z^N}{N!}$ with $A'(z) = \sum_{N=1}^{\infty} A_N \frac{z^{N-1}}{(N-1)!}$ by multiplying both sides by $\frac{z^{N-1}}{(N-1)!}$ and summing for $N \geq 1$:

$$\begin{aligned}
\sum_{N=1}^{\infty} \frac{A_N z^{N-1}}{(N-1)!} &= \sum_{N=1}^{\infty} \frac{(N-1)\,z^{N-1}}{(N-1)!} + 2\sum_{N=1}^{\infty}\sum_{k=0}^{\infty} \frac{1}{2^{N-1}} \binom{N-1}{k} A_k \frac{z^{N-1}}{(N-1)!} \\
&= z\sum_{N=2}^{\infty} \frac{z^{N-2}}{(N-2)!} + 2\sum_{k=0}^{\infty}\sum_{N=k+1}^{\infty} \frac{A_k}{2^{N-1}} \frac{(N-1)!}{k!\,(N-1-k)!} \frac{z^{N-1}}{(N-1)!} \\
&= z\sum_{t=0}^{\infty} \frac{z^t}{t!} + 2\sum_{k=0}^{\infty} \frac{A_k}{k!} \sum_{N=k+1}^{\infty} \frac{z^{N-1}}{2^{N-1}\,(N-k-1)!} \\
&= ze^z + 2\sum_{k=0}^{\infty} \frac{A_k}{k!} \sum_{N=k+1}^{\infty} \left(\frac{z}{2}\right)^{N-1} \frac{1}{(N-k-1)!} \\
&= ze^z + 2\sum_{k=0}^{\infty} \frac{A_k}{k!} \sum_{N=0}^{\infty} \left(\frac{z}{2}\right)^{(N+k+1)-1} \frac{1}{((N+k+1)-k-1)!} \\
&= ze^z + 2\sum_{k=0}^{\infty} \frac{A_k}{k!} \left(\frac{z}{2}\right)^{k} \sum_{N=0}^{\infty} \left(\frac{z}{2}\right)^{N} \frac{1}{N!} \\
&= ze^z + 2\sum_{k=0}^{\infty} \frac{A_k}{k!} \left(\frac{z}{2}\right)^{k} e^{\frac{z}{2}} \\
A'(z) &= ze^z + 2A\left(\frac{z}{2}\right) e^{\frac{z}{2}}
\end{aligned}$$

We simplify this by substituting $e^z B(z)$ for $A(z)$ with

$$B(z) = \sum_{N=0}^{\infty} B_N \frac{z^N}{N!}$$

and

$$B'(z) = \sum_{N=1}^{\infty} B_N \frac{z^{N-1}}{(N-1)!}$$

That is, $A(z) = e^z B(z)$ and $A'(z) = e^z B'(z) + e^z B(z)$. One can say that $B(z)$ is the expectation of the internal path length, if the number of keys is Poisson with parameter z. So the equation reads in terms of $B(z)$ as

$$e^z B'(z) + e^z B(z) = ze^z + 2B\left(\frac{z}{2}\right) e^{\frac{z}{2}} e^{\frac{z}{2}}$$

$$B'(z) + B(z) = z + 2B\left(\frac{z}{2}\right)$$

$$\sum_{N=1}^{\infty} B_N \frac{z^{N-1}}{(N-1)!} + \sum_{N=0}^{\infty} B_N \frac{z^N}{N!} = z + 2\sum_{N=0}^{\infty} B_N \frac{\left(\frac{z}{2}\right)^N}{N!}$$

This corresponds to a simple recurrence on the coefficients

$$B_N + B_{N-1} = \frac{1}{2^{N-2}} B_{N-1}$$

$$B_N = -\left(1 - \frac{1}{2^{N-2}}\right) B_{N-1}$$

for $N \geq 3$ with $B_2 = 1$.
And leads us to an explicit formula for B_N:

$$B_N = (-1)^N \prod_{j=1}^{N-2} \left(1 - \frac{1}{2^j}\right)$$

So we can get an explicit formula for A_N:

$$\begin{aligned}
A(z) &= e^z B(z) \\
&= e^z \sum_{N=0}^{\infty} B_N \frac{z^N}{N!} \\
&= \left(\sum_{N=0}^{\infty} \frac{z^N}{N!}\right)\left(\sum_{N=0}^{\infty} B_N \frac{z^N}{N!}\right) \\
&= \sum_{N=0}^{\infty}\left(\sum_{k=0}^{N} \frac{1}{(N-k)!}\frac{B_k}{k!}\right) z^N \\
&= \sum_{N=0}^{\infty}\left(\sum_{k=0}^{N} B_k \binom{N}{k}\right)\frac{z^N}{N!}
\end{aligned}$$

$$A_N = \sum_{k=0}^{N} \binom{N}{k} B_k \tag{10.2}$$

We want to analyse this sum by Rice's integral resp. a theorem for meromorphic function, which is derived in 9:

Theorem 10.1. *Let ϕ be a function holomorphic in a domain that contains the half-line $[n_0, \infty[$. If n is big enough we have:*
If ϕ is meromorphic on the half-plane defined by $\Re(s) \geq d$ for some $d < n_0$ and of polynomial growth in this set, then

$$\sum_{k=n_0}^{n} \binom{n}{k} (-1)^k \phi(k) = -\sum_{s} (-1)^n Res_s \left(\phi(s)\frac{n!}{s(s-1)\dots(s-n)}\right) + O(n^d) \tag{10.3}$$

where the sum is taken over all poles but $n_0, n_0+1, \dots, n$.

Therefore we introduce a new series Q_N to come closer to the preceding equation:

$$Q_N = \prod_{j=1}^{N}\left(1 - \frac{1}{2^j}\right)$$

So we can get $B_N = (-1)^N Q_{N-2}$and by substitution:

$$A_N = \sum_{k=2}^{N} \binom{N}{k} (-1)^k Q_{k-2} \tag{10.4}$$

Q_N is defined only for integers, so we have to find a meromorphic function to extend Q_N to the complex plane. We choose the following function

$$Q(x) = \prod_{j=1}^{\infty}\left(1 - \frac{x}{2^j}\right)$$

with obviously $Q(1) = Q_\infty$ and you can see clearly that $\frac{Q(1)}{Q(2^{-N})}$ is a correct extension:

$$Q_N = \prod_{j=1}^{N}\left(1 - \frac{1}{2^j}\right) = \frac{\prod_{j=1}^{\infty}\left(1 - \frac{1}{2^j}\right)}{\prod_{j=N}^{\infty}\left(1 - \frac{1}{2^j}\right)} = \frac{Q(1)}{\prod_{j=1}^{\infty}\left(1 - \frac{1}{2^{j+N}}\right)} = \frac{Q(1)}{\prod_{j=1}^{\infty}\left(1 - \frac{2^{-N}}{2^j}\right)} = \frac{Q(1)}{Q(2^{-N})}$$

Our equation now has the form:

$$A_N = \sum_{k=2}^{N} \binom{N}{k} (-1)^k \frac{Q(1)}{Q(2^{-k+2})}$$

$Q(x)$ is obviously meromorphic on the half-plane defined by $\Re(s) \geq d$ with $d := \frac{1}{2}$ and the function is also polynomial. So we get the following equation by theorem 10.1:

$$A_N = -\sum_{z} (-1)^N Res_z \left(B(N+1, -z) \frac{Q(1)}{Q(2^{-z+2})} \right) + O\left(N^{\frac{1}{2}}\right) \qquad (10.5)$$

We proceed to evaluate the poles at $z \leq 1$ for $B(N+1, -z) \frac{Q(1)}{Q(2^{-z+2})}$:

- $B(N+1, -z)$ is singular at $z = 0, 1$ because the Γ function is zero in the denominator.
- $z = j \pm \frac{2\pi ik}{\ln 2}$ for $j = 1, 0, -1, ...$ and all $k \geq 0$ are poles since at these points $2^{-z+j} = 1$ which causes one of the factors of $Q\left(2^{-z+2}\right)$ to vanish.

Only the poles for $z \geq \frac{1}{2}$ are within the region of interest.
So we can approximate the residues at the poles. Beginning with $z = 1$:

$$-B(N+1, -z) \frac{Q(1)}{Q(2^{-z+2})} = -B(N+1, -z) \frac{1}{1 - 2^{-z+1}} \frac{Q(1)}{Q(2^{-z+1})}$$

First analyze $-B(N+1, -z)$:

$$\begin{aligned}
-B(N+1, -z) &= -\frac{\Gamma(N+1)\Gamma(-z)}{\Gamma(N+1-z)} \\
&= -\frac{N!(-z-1)!}{(N-z)!} \\
&= (-1)^N \frac{N!}{\prod_{k=0}^{N}(z-k)} \\
&= (-1)^N \frac{N!}{z\prod_{k=1}^{N} k\left(\frac{z}{k}-1\right)} \\
&= (-1)^N \frac{N!}{zN!\prod_{k=1}^{N}\left(\frac{z}{k}-1\right)} \\
&= \left((-1)^N z \prod_{k=1}^{N} -\left(1-\frac{z}{k}\right)\right)^{-1} \\
&= -\left(z \prod_{k=1}^{N}\left(1-\frac{z}{k}\right)\right)^{-1} \\
&= \left(z(z-1)\prod_{k=2}^{N}\left(1-\frac{z}{k}\right)\right)^{-1}
\end{aligned}$$

We want to approximate this by a Taylor series expansion. To do this the following lemma is helpful:

Lemma 10.1. *If $F(z) = \prod_{j \in R} \frac{1}{1-f_j(z)}$ for some index set R, then the Taylor series expansion of F at a, if it exists, is given by*

$$F(z) = F(a)\left(1 + \sum_{j \in R} \frac{f_j'(a)}{1-f_j(a)}(z-a) + O\left((z-a)^2\right)\right)$$

Proof.

$$G(z) = \prod_{j\in R} g_j(z)$$
$$G'(z) = \sum_{j\in R} g_j'(z) \prod_{k\in R\neq j} g_k(z)$$
$$\frac{G'(z)}{G(z)} = \frac{\sum_{j\in R} g_j'(z) \prod_{k\in R\neq j} g_k(z)}{\prod_{j\in R} g_j(z)} = \sum_{j\in R} \frac{g_j'(z)}{g_j(z)}$$

$$F(z) = F(a)\left(1 + \frac{F'(a)}{F(a)}(z-a) + O\left((z-a)^2\right)\right) = F(a) + F'(a)(z-a) + O\left((z-a)^2\right)$$

□

At $a = 1$ we have the expansion for the Beta function:

$$\begin{aligned} -B(N+1,-z) &= \frac{1}{1-z} z^{-1} \prod_{j=2}^{N} \left(1 - \frac{z}{j}\right)^{-1} \\ &= \frac{1}{1-z} N \left(1 + (H_{N-1} - 1)(z-1) + O\left((z-1)^2\right)\right) \\ &= \frac{N}{1-z} - N(H_{N-1} - 1) + O(z-1) \\ &= -\frac{N}{z-1} - N(H_{N-1} - 1) + O(z-1) \end{aligned}$$

with $H_{N-1} = \gamma + \ln N - O\left(\frac{1}{N}\right)$. So we get

$$-B(N+1,-z) = -\frac{N}{z-1} - N(\gamma + \ln N - 1) + O(z-1)$$

Approximation of $\frac{1}{1-2^{-z+1}}$ with the series expansion for $\frac{1}{e^x-1}$ leads to:

$$\begin{aligned} \frac{1}{1-2^{-z+1}} &= \frac{1}{1-e^{\ln 2(-z+1)}} = -\frac{1}{(-z+1)\ln 2} + \frac{1}{2} - \frac{-z+1}{12} + O\left((-z+1)^3\right) \\ &= \frac{1}{(z-1)\ln 2} + \frac{1}{2} + O(z-1) \end{aligned}$$

The missing part $\frac{Q(1)}{Q\left(2^{-z+1}\right)}$ is analyzed similarly to $-B(N+1,-z)$ by using the Taylor expansion for the Q function:

$$\begin{aligned} \frac{Q(1)}{Q(2^{-z+1})} &= Q(1) \prod_{j<1} \left(1 - 2^{-z+j}\right)^{-1} \\ &= 1 - \ln 2 \sum_{j<1} \frac{2^{j-1}}{1-2^{j-1}} (z-1) + O\left((z-1)^2\right) \\ &= 1 - \alpha \ln 2 (z-1) + O\left((z-1)^2\right) \end{aligned}$$

with $\alpha = 1 + \frac{1}{3} + \frac{1}{7} + \frac{1}{15} + ... \approx 1,606695$.
Connecting the analyzed parts, the integrand is approximately:

$$\begin{aligned} -B(N+1,-z) \frac{Q(1)}{Q(2^{-z+2})} &= \left(-\frac{N}{z-1} - N(H_{N-1} - 1) + O(z-1)\right) \\ &\times \left(\frac{1}{(z-1)\ln 2} + \frac{1}{2} + O(z-1)\right) \\ &\times \left(1 - \alpha \ln 2 (z-1) + O\left((z-1)^2\right)\right) \end{aligned}$$

The residue at $z = 1$ is the Laurent coefficient a_{-1} of $\frac{1}{z-1}$ in this product:

$$\begin{aligned} Res_{z=1} &= -\frac{N}{\ln 2}\left(H_{N-1} - 1\right) + N\left(\alpha - \frac{1}{2}\right) \\ &= -N \lg N - N\left(\frac{\gamma - 1}{\ln 2} - \alpha + \frac{1}{2}\right) + O\left(1\right) \end{aligned}$$

Then approximate the residues at the other poles $z = 1 \pm \frac{2\pi ik}{\ln 2}$:

$$\begin{aligned} Res_{z=1\pm\frac{2\pi ik}{\ln 2}} &= -\frac{1}{\ln 2}\sum_{k\neq 0} B\left(N+1, -1-\frac{2\pi ik}{\ln 2}\right) \\ B\left(N+1, -1-\frac{2\pi ik}{\ln 2}\right) &= \frac{\Gamma\left(N+1\right)\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)}{\Gamma\left(N-\frac{2\pi ik}{\ln 2}\right)} \\ &= N\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)\frac{\Gamma\left(N\right)}{\Gamma\left(N-\frac{2\pi ik}{\ln 2}\right)} \end{aligned}$$

Now the standard approximation formula for the Γ function is used to simplify this term.

$$\begin{aligned} \frac{\Gamma\left(N+1\right)}{\Gamma\left(N+1-\alpha\right)} &= N^{\alpha}\left(1 + O\left(\frac{1}{N}\right)\right) \\ Res_{z=1\pm\frac{2\pi ik}{\ln 2}} &= N\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)\left(N-1\right)^{\frac{2\pi ik}{\ln 2}}\left(1 + O\left(\frac{1}{N}\right)\right) \\ &= N\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)N^{\frac{2\pi ik}{\ln 2}}\left(1 + O\left(\frac{1}{N}\right)\right) \\ &= N\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)e^{\frac{2\pi ik \lg N}{\ln 2}}\left(1 + O\left(\frac{1}{N}\right)\right) \end{aligned}$$

The sum of residues at the points $z = z = 1 \pm \frac{2\pi Ik}{\ln 2}$ is found to be

$$Res_{z=1\pm\frac{2\pi Ik}{\ln 2}} = -N\delta\left(N\right) + O\left(1\right)$$

where

$$\delta\left(N\right) = \frac{1}{\ln 2}\sum_{k\neq 0}\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)e^{2\pi ik \lg N}$$

is a small oscillatory term. So finally we insert these results in (10.5) and get the following theorem:

Theorem 10.2. *The average internal path length of a digital search tree built from N records with keys from random bit stream is*

$$A_N = N \lg N + N\left(\frac{\gamma - 1}{\ln 2} - \alpha + \frac{1}{2} + \delta\left(N\right)\right) + O\left(N^{\frac{1}{2}}\right)$$

Proof. This follows from the derivation above. □

10.3.4 External Internal Nodes

A property of trees of some interest is the number of internal nodes which have both links null. An alternate storage representation could be used for such nodes to save space.

Theorem 10.3. *The average number of nodes with both links null in a digital search tree built from N records with keys from random bit streams is*

$$N\left(\beta+1-\frac{1}{Q_\infty}\left(\frac{1}{\ln 2}+\alpha^2-\alpha\right)+\delta^*(N)\right)+O\left(N^{\frac{1}{2}}\right)$$

where the constants involved have the values
$\alpha=1+\frac{1}{3}+\frac{1}{7}+\frac{1}{15}+...\approx 1.606695...,$
$Q_\infty=\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{7}{8}+...\approx .288788...$ *and*
$\beta=\frac{1\cdot 2^2}{1}\left(\frac{1}{1}\right)+\frac{2\cdot 2^3}{1\cdot 3}\left(\frac{1}{1}+\frac{1}{3}\right)+\frac{3\cdot 2^4}{1\cdot 3\cdot 7}\left(\frac{1}{1}+\frac{1}{3}+\frac{1}{7}\right)+...\approx 7.74313...$
The function $\delta^*(N)$ is a periodic function in $\lg N$, with $|\delta^*(N)|<10^{-6}$. The approximate value of the coefficient of the leading term is $0.372046812...$.

Proof. As before, we use a simple transform with generating functions to derive an explicit sum, then use Rice's method to evaluate this sum. The number of external internal nodes is

$$C_N=\sum_{k=0}^{\infty}\frac{1}{2^{N-1}}\binom{N-1}{k}\left(C_k+C_{N-1-k}\right),\qquad N\geq 2 \tag{10.6}$$

with $C_1=1$ and $C_0=0$. This follows from the fact that the number of nodes with both links null in a tree is exactly the sum of the numbers of such nodes in the two subtrees of the root while the tree has more than one node.
In terms of the exponential generating function $C(z)=\sum_{N=0}^{\infty}\frac{C_N z^N}{N!}$ by multiplying both sides by $\frac{z^{N-1}}{(N-1)!}$ and summing for $N\geq 1$, we have:

$$\begin{aligned}
C_1+\sum_{N=2}^{\infty}\frac{C_N z^{N-1}}{(N-1)!} &= 1+\sum_{N=2}^{\infty}\left(2\sum_{k=0}^{\infty}\frac{1}{2^{N-1}}\binom{N-1}{k}C_k\frac{z^{N-1}}{(N-1)!}\right)\\
\sum_{N=1}^{\infty}\frac{C_N z^{N-1}}{(N-1)!} &= 1+\sum_{N=2}^{\infty}\left(2\sum_{k=0}^{\infty}\frac{1}{2^{N-1}}\binom{N-1}{k}C_k\frac{z^{N-1}}{(N-1)!}\right)
\end{aligned}$$

This leads, similar to A_N, to the equation

$$C'(z)=1+2C\left(\frac{z}{2}\right)e^{\frac{z}{2}} \tag{10.7}$$

Again we introduce a new generating function $D(z)=\sum_{N=0}^{\infty}\frac{D_N z^N}{N!}$ defined by $D(z)=e^{-z}C(z)$ to get a somewhat more manageable form:

$$D'(z)+D(z)=e^{-z}+2D\left(\frac{z}{2}\right)$$

By the recurrence on the coefficients we get:

$$D_N+D_{N-1}=(-1)^{N-1}+\frac{1}{2^{N-2}}D_{N-1}$$

$$D_N=(-1)^{N-1}-\left(1-q^{N-2}\right)D_{N-1},\qquad N\geq 2 \tag{10.8}$$

with $D_1=1$, $D_0=0$.
We define the constant $q:=\frac{1}{2}$ (we will see that only this constant changes for M-ary trees). This recurrence is inhomogeneous, so we get a somewhat more complicated explicit form:

$$D_N=(-1)^{N-1}\sum_{i=1}^{N-1}\prod_{j=i}^{N-2}\left(1-q^j\right)$$

Rewriting this in terms of

$$\begin{aligned}
R_N &= \sum_{i=1}^{N} \prod_{j=i}^{N} \left(1 - \frac{1}{2^j}\right) \\
&= \sum_{i=1}^{N} \frac{\prod_{j=1}^{N} \left(1 - \frac{1}{2^j}\right)}{\prod_{j=1}^{i} \left(1 - \frac{1}{2^j}\right)} \\
&= Q_N \sum_{i=2}^{N} \frac{\prod_{j=1}^{N} \left(1 - \frac{1}{2^j}\right)}{\prod_{j=1}^{i} \left(1 - \frac{1}{2^j}\right)} \\
&= Q_N + \sum_{i=1}^{N} \frac{Q_N}{Q_i} \\
&= Q_N \left(1 + \sum_{k=1}^{N} \frac{1}{Q_k}\right)
\end{aligned}$$

and transforming like (10.2) back:

$$\begin{aligned}
D_N &= (-1)^N R_{N-2} C_N \\
&= N - \sum_{k=2}^{N} \binom{N}{k} D_N
\end{aligned}$$

We have the following explicit sum for the desired quantity:

$$C_N = N - \sum_{k=2}^{\infty} \binom{N}{k} (-1)^k R_{k-2} \tag{10.9}$$

This sum is similar to (10.4) but more difficult to evaluate because R_N is more complicated than Q_N. Because $R(z)$ does not extend R_N for positive integers, we get by Taylor expansion that $R_N = N + 1 - \alpha + R_N^*$. R_N^* satisfies a simple recurrence, converges very quickly and is polynomial bounded, so we extend R_N^* by the meromorphic function $R^*(z)$

$$\begin{aligned}
R_N^* &= \frac{(N + 1 - \alpha) q^{N+1}}{1 - q^{N+1}} + \frac{1}{1 - q^{N+1}} R_{N+1}^* \\
R^*(z) &= \sum_{i=2}^{\infty} \frac{(z + 1 + i - \alpha) q^{z+1+i}}{\prod_{j=0}^{i} (1 - q^{z+1+j})}
\end{aligned}$$

Substituting, we have

$$C_N = N - \sum_{k=2}^{\infty} \binom{N}{k} (-1)^k (R_{k-2}^* + k + 1 - \alpha)$$

After applying the elementary identities of Pascal's triangle

$$\sum_{k=0}^{\infty} \binom{N}{k} (-1)^k = \sum_{k=0}^{\infty} \binom{N}{k} \alpha (-1)^k = 0,$$

we have the simplified result

$$C_N = (N - 1)(\alpha + 1) - \sum_{k=2}^{\infty} \binom{N}{k} (-1)^k R_{k-2}^* \tag{10.10}$$

Now, by Theorem 10.1 and again looking only at the half-plane to the right of the line $z = \frac{1}{2}$, we know that

$$C_N - (N-1)(\alpha+1) = -\sum_z (-1)^N Res_z \left(B(N+1, -z) R^*(z-2)\right) + O\left(N^{\frac{1}{2}}\right) \tag{10.11}$$

In this case we look at the poles for $R^*(z-2)$ at $1 \pm \frac{2\pi i k}{\ln 2}$ and see that they are all single poles. The main term is given by $N \lim_{z\to 1} R^*(z-2)$; the poles for $k \neq 0$ add a small oscillatory term.

Lemma 10.2.

$$\sum_{n=0}^{\infty} \frac{u^n}{\prod_{k=1}^{n}(1-q^k)} = \frac{1}{\prod_{k=0}^{\infty}(1-q^k u)}$$

Proof. The coefficient of $u^n q^m$ on both sides is the number of ways to write n as the sum of m nonnegative integers. □

The method of calculating $R^*(-1)$ is to express $R^*(z)$ in terms of generating functions, which generalizes the function of Lemma 10.2, then to expand that function and exploit certain properties of its derivatives. Specifically, we define

$$F(u,v) = \sum_{j=1}^{\infty} \frac{q^j u^j}{\prod_{i=1}^{j}(1-q^i v)}$$

This is the generating function for restricted partitions, the coefficients of $u^n v^m q^k$ is the number of ways to partition k into m parts not exceeding n. By Lemma 10.2 we get:

$$F(u,1) = \sum_{n=0}^{\infty} \frac{q^n u^n}{\prod_{k=1}^{n}(1-q^k)}$$
$$F(u,1) + 1 = \sum_{n=0}^{\infty} \frac{u^n}{\prod_{k=1}^{n}(1-q^k)}$$
$$F(u,1) = \frac{1}{\prod_{k=0}^{\infty}(1-q^k u)} - 1$$
$$F(1,1) = Q_\infty^{-1} - 1.$$

Also we have

$$F_1'(u,1) = \frac{1}{\prod_{i=1}^{\infty}(1-q^i u)} \sum_{(k=1)}^{\infty} \frac{q^k}{(1-q^k u)}$$

so that $F_1'(1,1) = \frac{\alpha}{Q_\infty}$. Furthermore, we have

$$F\left(1, q^{z+1}\right) = \sum_{j=1}^{\infty} \frac{q^j}{\prod_{i=1}^{j}(1-q^{z+1+i})}$$

$$F_1'\left(1, q^{z+1}\right) = \sum_{j=1}^{\infty} \frac{j q^j}{\prod_{i=1}^{j}(1-q^{z+1+i})},$$

which finally gives the following expansion:

$$R^*(z) = \frac{q^{z+1}}{1-q^{z+1}} \left((z+1-\alpha)\left(F\left(1,q^{z+1}\right)+1\right) + F_1'\left(1,q^{z+1}\right)\right)$$

From this formulation, a Taylor expansion around $z = -1$ is straightforward:

$$\frac{q^{z+1}}{1-q^{z+1}} = \frac{1}{(z+1)\ln q} - \frac{1}{2} + O(z+1)$$

$$F\left(1, q^{z+1}\right) = F(1,1) + (z+1)\ln q F_2'(1,1) + O(z+1)^2$$
$$F_1'\left(1, q^{z+1}\right) = F_1'(1,1) + (z+1)\ln q F_{12}'(1,1) + O\left((z+1)^2\right),$$

so that

$$R(z) = -\frac{F(1,1)+1}{\ln q} + \alpha F_2'(1,1) - F_{12}''(1,1) + O(z+1)$$

$$F_2'(1,1) = \sum_{j=1}^{\infty}\left(\frac{q^j}{\prod_{i=1}^{j}(1-q^i)}\left(\sum_{k=1}^{j}\frac{q^k}{1-q^k}\right)\right) \tag{10.12}$$

$$F_{12}''(1,1) = \sum_{j=1}^{\infty}\left(\frac{jq^j}{\prod_{i=1}^{j}(1-q^i)}\left(\sum_{k=1}^{j}\frac{q^k}{1-q^k}\right)\right) \tag{10.13}$$

Actually, we can relate $F_2'(1,1)$ to α and Q_∞. Because

$$F_1'(1,1) = F_2'(1,1) + F(1,1)$$

there is $F_2'(1,1) = \frac{\alpha-1}{Q_\infty}+1$. There does not seem to be an easy way to express $F_{12}''(1,1)$ in terms of α and Q_∞, so we denote that constant simply by β. Collecting terms, we have shown that the residue of the integrand at $z=1$ is

$$N\left(\beta + 1 - \frac{1}{Q_\infty}\left(\alpha^2 - \alpha - \frac{1}{\ln q}\right)\right)$$

It remains to calculate the residues of the integrand at the other singularities. This calculation is straightforward: the residue of $\frac{1}{1-q^{z+1}}$ at $z = -1 \pm \frac{2\pi i k}{\ln q}$ is $-\frac{1}{\ln q}$, and the other terms in $R^*(z)$ contribute a factor of $\frac{2\pi i k}{Q_\infty \ln q}$. The factor for $B(N+1,-z)$ is expanded exactly as in the preceding Taylor expansion for the internal path length; thus we have the oscillatory term

$$\delta^*(N) = \frac{1}{Q_\infty \ln q}\sum_{k\neq 0}\frac{2\pi i k}{\ln q}\Gamma\left(-1-\frac{2\pi i k}{\ln q}\right)e^{2\pi i k \lg N}$$

This completes the calculation of the coefficient of the linear term. □

10.4 Digital Search Tries

10.4.1 Data Structure of Tries

Definition 10.10. A digital search trie is a digital tree for storing a set of strings in which there is one node for every prefix of every string in the set.

The name of this data structure comes from the word re<u>trie</u>val. The word retrieval is stressed, because a trie has a lookup time that is equivalent to the length of the string being looked up. Again we represent the strings as keys in binary form. It may be convenient to assume that the strings are all of same (binary) length, but the method is also appropriate for varying length strings, if no string is a prefix of another.
Digital search tries compared to trees have much improved worst case performance. Their average case performance is asymptotically optimal. If N records with keys from random bit streams are inserted into an initially empty trie, then the average number of nodes examined during successful search reads as

$$\lg N + \frac{\gamma}{\ln 2} + \frac{1}{2} + \delta(N) + O\left(\frac{1}{N}\right)$$

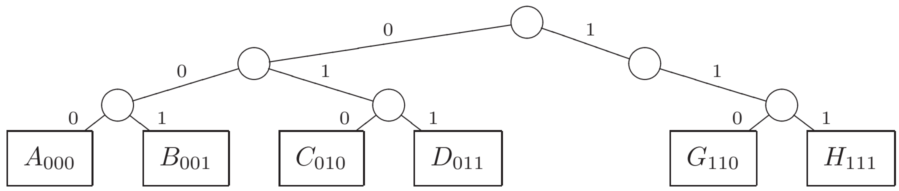


Figure 10.5: Example for a digital search trie with external path length = 18

10.4.2 Data Structure of Patricia Tries

You can optimize the performance of a trie constructed with N keys by ensuring that this trie has just $N - 1$ internal nodes by collapsing one-way branches on internal nodes and get the so called Patricia tries:

Definition 10.11. A Patricia tree is defined as a compact representation of a digital search trie where all nodes with one child are merged with their parent.

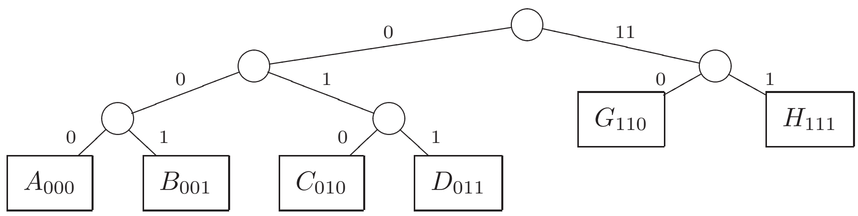


Figure 10.6: Example for a Patricia trie

The average number of nodes examined during successful search is one less than for standard tries.

10.4.3 External Path Length

Definition 10.12. The external path length of a tree is the sum of the depth of every leaf of the tree.

The fundamental recurrence for the average external path length of a binary trie is

$$A_N^{[T]} = N + \sum_{k=0}^{\infty} \frac{1}{2^N} \binom{N}{k} \left(A_k^{[T]} + A_{N-k}^{[T]} \right), \qquad N \geq 2 \tag{10.14}$$

with $A_0^{[T]} = A_1^{[T]} = 0$. This is the number of nodes examined during all successful searches. Note that since no key is stored at the root, the subtrees have a total of N keys. The resulting functional equation on the exponential generating function is not a difference-differential but simply a difference equation:

$$A^{[T]}(z) = z\left(e^z - 1\right) + 2A^{[T]}\left(\frac{z}{2}\right) e^{z-2}$$

It is still convenient to transform the equation with $A(z) = e^z B(z)$ to get the equation

$$B^{[T]}(z) = z\left(1 - e^{-z}\right) + 2B^{[T]}\left(\frac{z}{2}\right)$$

This yields directly

$$B^{[T]}(z) = \frac{N(-1)^N}{1 - \left(\frac{1}{2}\right)^{N-1}}$$

and

$$A_N^{[T]} = \sum_{k=2}^{\infty} \binom{N}{k} (-1)^k \frac{k}{1 - \left(\frac{1}{2}\right)^{k-1}}$$

This can be handled directly by Rice's method or Mellin transform techniques, as described in full detail in [Szp00].

The fundamental recurrence for the average external path length of a Patricia trie is

$$A_N^{[P]} = N\left(1 - \frac{1}{2^{N-1}}\right) + \sum_{k=0}^{\infty} \frac{1}{2^N} \binom{N}{k} \left(A_k^{[P]} + A_{N-k}^{[P]}\right), \qquad N \geq 1 \tag{10.15}$$

with $A_0^{[P]} = 0$. The external path length is the sum of the ones of the subtries of the root plus the number of nodes in the subtries (N) unless one of the subtries is empty which has the probability $\frac{1}{2^{N-1}}$. The resulting functional equation on the exponential generating function is

$$A^{[P]}(z) = z\left(e^z - e^{\frac{z}{2}}\right) + 2A^{[P]}\left(\frac{z}{2}\right) e^{\frac{z}{2}}$$

with transformed version

$$B^{[P]}(z) = z\left(1 - e^{-\frac{z}{2}}\right) + 2B^{[P]}\left(\frac{z}{2}\right)$$

which yields directly

$$B^{[P]}(z) = \frac{N(-1)^N}{2^{N-1} - 1}$$

and

$$A_N^{[P]} = \sum_{k=2}^{\infty} \binom{N}{k} \frac{k(-1)^k}{2^{k-1} - 1} = A_N^{[T]} - N$$

Given the result for binary tries the average external path length for Patricia tries is obvious.

10.4.4 External Internal Nodes

The average number of internal nodes with both sons external are computed for Patricia tries. The derivation is similar to the one for digital search trees, so we only sketch it here. We start with the recurrence

$$C_N^{[P]} = \sum \frac{1}{2^N} \binom{N}{k} \left(C_k^{[P]} + C_{N-k}^{[P]}\right), \qquad N \geq 3 \tag{10.16}$$

with $C_0^{[P]} = C_1^{[P]} = 0$ and $C_2^{[P]} = 1$. This corresponds to the functional equation

$$C^{[P]}(z) = \left(\frac{z}{2}\right)^2 + 2C^{[P]}\left(\frac{z}{2}\right) e^{\frac{z}{2}}$$

which transforms to

$$D^{[P]}(z) = \left(\frac{z}{2}\right)^2 e^{-z} + 2D^{[P]}\left(\frac{z}{2}\right)$$

and eventually gives the sum

$$C_N^{[P]} = \frac{1}{4} \sum_{k=2}^{N} \binom{N}{k} (-1)^k \frac{k(k-1)}{1 - \frac{1}{2}^{k-1}}$$

Knuth gives specific evaluations of such sums. The eventual result is that the proportion of nodes in Patricia tries with both sons external is $\frac{1}{4 \ln 2} = .3606...$ plus a small oscillatory term. Thus, according to this measure, digital search trees are (slightly) more balanced than Patricia tries.

10.5 Multiway Branching

Above only binary trees have been analyzed. But we can generalize the average analysis to M-ary trees, where each node contains M links to other nodes, numbered from 0 to $M-1$. It turns out that the analysis given above survives largely intact for the M-ary case. For example, to find the average number of nodes in a M-ary digital search tree with all links null, we begin with the fundamental recurrence:

$$C_N^{[M]} = \sum_{k_1+k_2+...+k_M=N-1} \frac{1}{M^{N-1}} \binom{N-1}{k_1, k_2, ..., k_M} \left(C_{k_1}^{[M]} + C_{k_2}^{[M]} + ... + C_{k_M}^{[M]}\right), \quad N \geq 2 \qquad (10.17)$$

with $C_1^{[M]} = 1$ and $C_0^{[M]} = 0$. The argumentation is the same as for the digital search tree. The number of nodes with all links null in a tree is exactly the number of such nodes in all the subtrees of the root, unless the tree has just one node. Again the partitions of $N-1$ nodes without the root into M subtrees weighted by all possibilities are examined. All the subtrees are randomly built according to the same model.
By symmetry, (10.17) is equivalent to

$$C_N^{[M]} = M \sum_{k_1+k_2+...+k_M=N-1} \frac{1}{M^{N-1}} \binom{N-1}{k_1, k_2, ..., k_M} C_{k_1}^{[M]}, \qquad N \geq 2$$

with $C_1^{[M]} = 1$ and $C_0^{[M]} = 0$. Now we introduce the exponential generating function $C^{[M]}(z) = \sum_{N=0}^{\infty} \frac{C_N^{[M]} z^N}{N!}$ and derive the following difference-differential equation:

$$C^{[M]\prime}(z) = 1 + MC^{[M]}\left(\frac{z}{M}\right)\left(e^{\frac{z}{M}}\right)^{M-1} = 1 + MC^{[M]}\left(\frac{z}{M}\right)\left(e^{\left(1-\frac{1}{M}\right)z}\right) \qquad (10.18)$$

For $M = 2$, this is exactly the equation derived from (10.7); moreover none of the manipulations used for solving it depend in an essential way on the value of that constant.

Corollary 10.1. *The average number of nodes with all links null in an M-ary search tree (for $M \geq 2$) built from N records with keys from random bit streams is*

$$N\left(\beta^{[M]} + 1 - \frac{1}{Q_\infty^{[M]}}\left(\frac{1}{\ln M} + \alpha^{[M]2} - \alpha^{[M]}\right) + \delta^{[M]}(N)\right) + O\left(N^{\frac{1}{2}}\right)$$

where the constants involved are given by
$\alpha^{[M]} = \sum_{k=1}^{\infty} \frac{1}{M^k-1}$,
$Q_\infty^{[M]} = \prod_{k=1}^{\infty}\left(1 - \frac{1}{M^k}\right)$,
$\beta^{[M]} = \sum_{k=1}^{\infty} \frac{kM^{k+1}}{\prod_{i=1}^{k}\left(M^i-1\right)} \sum_{j=1}^{k} \frac{1}{M^j-1}$
and the oscillatory term is
$\delta^{[M]}(N) = \frac{1}{Q_\infty \ln M} \sum_{k\neq 0} \frac{2\pi ik}{\ln M}\Gamma\left(-1 + \frac{2\pi ik}{\ln M}\right) e^{2\pi ik \lg N}$.

10.6 General Framework

The methods that we have used in the previous sections can be applied to study many other properties of digital trees. If $X(T)$ and $x(T)$ are parameters of trees satisfying

$$X(T) = \sum_{subtrees\, T_j\, of\, the\, root\, of\, T} X(T_j) + x(T) \qquad (10.19)$$

then the exponential generating functions for the expectations X_N and x_N for an M-ary digital search tree built from N records with keys from random bit streams satisfy

$$X'(z) = MX\left(\frac{z}{M}\right) e^{\left(1-\frac{1}{M}\right)z} + x(z)$$

This is derived in exactly the same manner as (10.18). Now in terms of the generating functions $Y(z) = e^{-z}X(z)$ and $y(z) = e^{-z}x(z)$ this becomes

$$Y'(z) + Y(z) = MY\left(\frac{z}{M}\right) + y'(z) + y(z) \tag{10.20}$$

This leads to a nonlinear recurrence like (10.8) satisfied by Y_N, with the solution sought given by $X_N = \sum_{k=0}^{N} \binom{N}{k} Y_k$. If the quantity $(-1)^k Y_k$ is sufficiently well behaved, we can study its asymptotics and find a function Y_k^* which

(i) is simply related to Y_k so that $\sum_{k=0}^{N} \binom{N}{k} \left(Y_k - (-1)^k Y_k^*\right)$ is easily evaluated,

(ii) satisfies a recurrence of the form $Y_{N+1}^* = (1 - g(M, N)) Y_N^* + f(M, N)$,

(iii) goes to zero quickly as $N \to \infty$.

Depending on the nature of $g(M, N), f(M, N)$ and the speed of convergence, conditions (ii) and (iii) may allow the recurrence to be turned around to extend Y_N^* to the complex plane and so allow the desired expectation to be computed by evaluating the sum $\sum_{k=0}^{N} \binom{N}{k} \left(Y_k - (-1)^k Y_k^*\right)$ as detailed in the previous sections.

The same type of generalization applies to the study of tries (and Patricia tries), and the simpler nature of the recurrences follows through the generalization. For example, the exponential generating functions for the expectations X_N and x_N of parameters of trees satisfying (10.19) for a random trie built from N records from random bit stream is

$$X(z) = MX\left(\frac{z}{M}\right) e^{\frac{z}{M}} + x(z) \tag{10.21}$$

which is considerably easier to deal with. The equation can be solved by Rice's method and also by Mellin transform techniques.

This general framework allows quite full analysis of the types of trees considered, and they clearly expose the fundamental differences and similarities among the analyses.

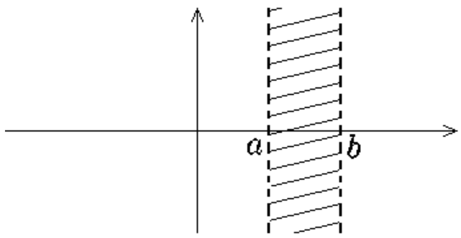
Chapter 11

Mellin transforms and asymptotics: Harmonic sums

Ilja Posov

This survey presents a unified and essentially self-contained approach to the asymptotic analysis of a large class of sums that arise in combinatorial mathematics, discrete probabilistic models, and the average-case analysis of algorithms. It relies on the Mellin transform, a close relative of the integral transforms of Laplace and Fourier. The method applies to harmonic sums that are superpositions of rather arbitrary "harmonics" of a common base function. Its principle is a precise correspondence between individual terms in the asymptotic expansion of an original function and singularities of the transformed function. Here no theorem is proved, and even not every theorem is completely formulated. For precise presentation of the theory reader is refered to the original paper.

We have to deal a lot with complex variable functions and I'll remind you some basic concepts about them. The first concept about complex variable functions is holomorphic function. Function is called holomorphic in some area, if it has complex derivative in every point of this area.
The second concept is 'analitic function'. Function is analitic in some point z_0 of complex plane, if it can be expanded into Taylor series in this point, i.e. $f(z) = \sum_{n=0}^{\infty} c_n(z-z_0)^n = c_0 + c_1(z-z_0) + c_2(z-z_0)^2 + \cdots$. Similary, the function is called analitic in an area, if it is analitic in every point of that area. One of the central result of the complex variable function theory is the theorem, that every holomorphic in some area function is analitic there. The converse statement holds too. We'll use the word 'analitic' a lot.
Except holomorphic functions there are meromorphic functions. Meromorphic in an area function is a function, that is holomorphic there except discrete set of points that are called poles. Discrete set means a set, every point of which can be isolated from other points. For example, every finite set is discrete.
Consider a function $f(z) = \frac{1}{z(z-1)}$. It is analitic (and therefore holomorphic) in $\mathbb{C} \setminus \{0,1\}$, but it is meromorphic in entire $\mathbb{C}$ with poles $z = 0$ and $z = 1$.
The last concept is 'open strip'. Open strip $\langle a, b\rangle = \{z = x + iy \mid a < y < b\}$ is a set of points in a complex plane that looks like:



Open strip can be infinite, if a or b is infinity, and it's obvious that $\langle -\infty, \infty\rangle = \mathbb{C}$

11.1 Mellin transform definition

Robert Hjalmar Mellin (1854-1933) was Finnish mathematitian who studied the transform which now bears his name and established its reciprocal properties. Now we are finally going to learn what Mellin transform is.

Definition 11.1. Let $f(x)$ be real function defined on $(0, +\infty)$. Then its Mellin transform is complex valued function that is defined by equality

$$\mathfrak{M}\left[f(x); s\right] = f^*(s) = \int_0^{+\infty} f(x)x^{s-1}dx$$

Of course, integral from definition usually converges not for all $s \in \mathbb{C}$, but it usually converges for all s from some open strip, which in this case is called 'fundamental' strip.

Proposition 11.1. *If $f(x) = \mathrm{O}(x^u)$ as $x \to 0$ and $f(x) = \mathrm{O}(x^v)$ as $x \to +\infty$, then the integral from Mellin transform definition converges for every $s \in \langle -u, -v\rangle$ and defines an analitic function in this strip.*

Example 11.1. If $f(x) = x^k$, then $f(x) = \mathrm{O}(x^k)$ as $x \to 0$ and $f(x) = \mathrm{O}(x^k)$ as $x \to +\infty$. Proposition states that transform of $f(x)$ (that is $f^*(s)$) exists in the open strip $\langle -k, -k\rangle$, but this strip is empty. In fact, transform of x^k simply doesn't exist, i.e. $\int_0^\infty x^k x^{s-1}dx$ doesn't converge for every $k \in \mathbb{R}$ and $s \in \mathbb{C}$. One can simply check it.

Now I present examples of functions that do have Mellin transforms.

Example 11.2. Let $f(x) = \frac{1}{1+x}$. $f(x) = \mathrm{O}(1) = \mathrm{O}(x^0)$ as $x \to 0$, and $f(x) = \mathrm{O}(x^{-1})$ as $x \to +\infty$. Now we can make use of proposition 11.1, here $u = 0$ and $v = -1$. Proposition states, that in this case Mellin transform $f^*(s) = \int_0^{+\infty} \frac{1}{1+x}x^{s-1}dx$ exists in the fundamental strip $\langle 0, 1\rangle$. The integral can be evaluated and it occurs, that $f^*(s) = \frac{\pi}{\sin \pi s}$. But the equality holds only for $s \in \langle 0, 1\rangle$, for other s integral simply doesn't converge. By the way, function $\frac{\pi}{\sin \pi s}$ by itself can be evaluated practically in entire $\mathbb{C}$, except, may be, integer points.

Example 11.3 (Gamma function). Now we consider the function $f(x) = e^{-x}$. $f(x) = \mathrm{O}(1) = \mathrm{O}(x^0)$ as $x \to 0$, and $\forall M{>}0$ $f(x) = \mathrm{O}(x^{-M})$ as $x \to +\infty$. Again, after using the proposition 11.1 we obtain, that Mellin transform of $f(x)$ exists in the open strip $\langle 0, M\rangle$ for every $M > 0$. It means, that the fundamental strip of this Mellin transform is $\langle 0, +\infty\rangle$. Now let's evaluate the transform. $f^*(s) = \int_0^{+\infty} e^{-x}x^{s-1}dx$. This integral is called Gamma function and notation is $\Gamma(s)$. There is a well known functional equation on gamma function which states that $s\Gamma(s) = \Gamma(s + 1)$. This equation allows us to evalute gamma function not only in the right half of complex plane, but also in every other point of complex plane. For example, $(-\frac{1}{2})\Gamma(-\frac{1}{2}) = \Gamma(\frac{1}{2}) = \sqrt{\pi}$, so $\Gamma(-\frac{1}{2}) = -2\sqrt{\pi}$. The problem is only with nonpositive integers.

Statement $0\Gamma(0) = \Gamma(1)$ doesn't allow us to evaluate gamma function in zero, it demonstrates, that there is a pole of gamma function in zero. Every negative integer is a pole of gamma function for the same reason.

Gamma function occurs frequently in Mellin transforms. But now we look on the last example of Mellin transform before going to learn basic properties of Mellin transform.

Example 11.4 (Transform of step function).

$$H(x) = \begin{cases} 1, x \in (0,1) \\ 0, x \in (1, +\infty) \end{cases}$$

As in the previous example and for the same reasons, fundamental strip of the transformed function is $\langle 0, +\infty \rangle$. Here the transform can be simply evaluated. $H^*(s) = \int_0^{+\infty} H(x)x^{s-1}dx = \int_0^1 x^{s-1}dx = \frac{1}{s}$. As in the all previous examples, we see that transformed function is defined not only in the fundamental strip. Here fundamental strip is $\langle 0, +\infty \rangle$, but $\frac{1}{s}$ may be evaluated in $\mathbb{C} \setminus \{0\}$. Fundamental strip is only the place where an integral converges.
If we had considered another step function $\overline{H}(x) = 1 - H(x)$, we would have obtained the transform $\overline{H}^*(s) = -\frac{1}{s}$ with fundamental strip $\langle -\infty, 0 \rangle$.

11.2 Mellin transform basic properties

All basic properties of Mellin transform can be simply obtained by means of such methods as integration by parts and change of variable. Here they are summarized in a table.

	$f(x)$	$f^*(s)$	$\langle \alpha, \beta \rangle$	
(1)	$x^\nu f(x)$	$f^*(s+\nu)$	$\langle \alpha - \nu, \beta - \nu \rangle$	
(2)	$f(x^\rho)$	$\frac{1}{\rho} f^*(\frac{s}{\rho})$	$\langle \rho\alpha, \rho\beta \rangle$	$\rho > 0$
(3)	$f(\frac{1}{x})$	$-f^*(-s)$	$\langle -\beta, -\alpha \rangle$	
(4)	$f(\mu x)$	$\frac{1}{\mu^s} f^*(s)$	$\langle \alpha, \beta \rangle$	$\mu > 0$
(5)	$\sum_k \lambda_k f(\mu_k x)$	$\left(\sum_k \frac{\lambda_k}{\mu^s}\right) f^*(s)$		
(6)	$f(x) \log x$	$\frac{d}{ds} f^*(s)$	$\langle \alpha, \beta \rangle$	
(7)	$\Theta f(x)$	$-sf^*(s)$	$\langle \alpha', \beta' \rangle$	$\Theta = x\frac{d}{dx}$
(8)	$\frac{d}{dx} f(x)$	$-(s-1)f^*(s-1)$	$\langle \alpha' + 1, \beta' + 1 \rangle$	
(9)	$\int_0^x f(t)dt$	$-\frac{1}{s} f^*(s+1)$		

The most interesting are the fourth and the fifth properties. The fourth one is called 'separation property' and the fifth property is its generalisation. If the sum $\sum_k \lambda_k f(\mu_k x)$ is finite, fifth property is obvious because of linearity of Mellin transform. But if the sum is infinite, the fifth property holds only if function $f(x)$ and series $\sum_k \frac{\lambda_k}{\mu^s}$ satisfy some additional conditions. Anyway, we'll use this property in this paper for infinite sums without paing attention to the problem. All studied functions are good enough and fifth property holds for them.

Example 11.5 (Zeta function). Here we'll use the fifth property to introduce zeta function. Consider the function

$$g(x) = \frac{e^{-x}}{1-e^{-x}} = e^{-x} + e^{-2x} + e^{-3x} + \cdots$$

The series converges for every $x > 0$. Now we apply the fifth property. $\lambda_k = 1$, $\mu_k = k$ and $f(x) = e^{-x}$, so

$$g^*(s) = \left(\frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \cdots\right) \mathfrak{M}\left[e^{-x}; s\right] = \zeta(s)\Gamma(s)$$

Series $\zeta(s) = \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \cdots$ converges for every $s \in \langle 1, +\infty\rangle$. Fundamental strip of the transform is $\langle 1, +\infty\rangle$ too, and why it is so we'll discuss later.

Now we'll summarize in a table a number of Mellin transforms, some of them were obtained earlier, some of them can be obtained by means of Mellin transform basic properties.

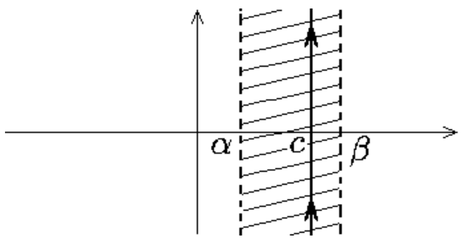
$f(x)$	$f^*(s)$	$\langle\alpha, \beta\rangle$
e^{-x}	$\Gamma(s)$	$\langle 0, +\infty\rangle$
$e^{-x} - 1$	$\Gamma(s)$	$\langle -1, 0\rangle$
e^{-x^2}	$\frac{1}{2}\Gamma(\frac{1}{2}s)$	$\langle 0, +\infty\rangle$
$\frac{e^{-x}}{1-e^{-x}}$	$\zeta(s)\Gamma(s)$	$\langle 1, +\infty\rangle$
$\frac{1}{1+x}$	$\frac{\pi}{\sin \pi s}$	$\langle 0, 1\rangle$
$\log(1+x)$	$\frac{\pi}{s \sin \pi s}$	$\langle -1, 0\rangle$
$H(x) \equiv 1_{0<x<1}$	$\frac{1}{s}$	$\langle 0, +\infty\rangle$
$x^\alpha (\log x)^k H(x)$	$\frac{(-1)^k k!}{(s+\alpha)^{k+1}}$	$\langle -\alpha, +\infty\rangle$ $k \in \mathbb{N}$

Here the most interesting is in the first two lines, we see two different functions having the same Mellin transform. The only diffence is in fundamential strips of the transforms. And now it's a good moment to formulate a theorem about reconstruction of initial function having only its Mellin transform.

Theorem 11.1. *Let f(x) have Mellin transform $f^*(s)$ with fundamental strip $\langle\alpha, \beta\rangle$. Let $\alpha < c < \beta$ and $f^*(c + it)$ is integrable. Then the equality*

$$\frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} f^*(s)x^{-s}dx = f(x)$$

holds almost everywhere.



The picture presents a patch of integration used in the theorem.

11.3 Singularities

Definition 11.2. Laurent expansion of function $\phi(s)$ in point s_0 is an equality:

$$\phi(s) = \sum_{k \geq -r}^{+\infty} c_k (s - s_0)^k$$

Here $c_{-r} \neq 0$. If $r > 0$, then s_0 is called a pole of order r. If $r = 1$, then pole is called simple. If $r = 2$, pole is double.

If $r \leq 0$, then function is analitic in s_0, because Laurent series in this case degenerates into Taylor series.

Example 11.6. Consider the function $\frac{1}{s^2(s+1)}$, it has two poles on the complex plane. Double pole is at $s_0 = 0$ and simple pole is at $s_0 = -1$:

$$\frac{1}{s^2(s+1)} = \frac{1}{s+1} + 2 + 3(s+1) + \cdots \qquad s_0 = -1$$

$$\frac{1}{s^2(s+1)} = \frac{1}{s^2} - \frac{1}{s} + 1 - s + \cdots \qquad s_0 = 0$$

Definition 11.3. A singular element (s.e.) of $\phi(s)$ at s_0 is an initial sum of Laurent expansion truncated at terms of O(1) or smaller.

Example 11.7. We consider the same function $\phi(s) = \frac{1}{s^2(s+1)}$ as in the previous example. Singular elements at $s_0 = 0$ are:

$$\left[\frac{1}{s^2} - \frac{1}{s}\right], \left[\frac{1}{s^2} - \frac{1}{s} + 1\right], \ldots$$

Here we can trancate the Laurant expansion wherever we want, but we are to include terms with negative degree of s. The same is about singular elements in $s_0 = 1$. They are:

$$\left[\frac{1}{s+1}\right], \left[\frac{1}{s+1} + 2\right], \left[\frac{1}{s+1} + 2 + 3(s+1)\right], \ldots$$

We are to include all items with negative degree of $s+1$, other items we can include if we want, but usually there is no need to do it.

Definition 11.4. Let $\phi(s)$ be meromorphic in some area Ω with $\mathfrak{S}$ including all the poles of $\phi(s)$ in Ω. A singular expansion of $\phi(s)$ in Ω is a formal sum of singular elements of $\phi(s)$ at all points of $\mathfrak{S}$. Notation: $\phi(s) \asymp E$.

Example 11.8.

$$\frac{1}{s^2(s+1)} \asymp \left[\frac{1}{s+1}\right]_{s=-1} + \left[\frac{1}{s^2} - \frac{1}{s}\right]_{s=0} + \left[\frac{1}{2}\right]_{s=1} \qquad s \in \mathbb{C}$$

It is a singular expansion of function $\phi(s) = \frac{1}{s^2(s+1)}$ in all complex plane. $\mathfrak{S} = \{-1, 0, 1\}$. There is no pole at point $s_0 = 1$, but we can include a singular element at it in a singular expansion, if we want. $\mathfrak{S}$ is to include all the poles of the function, but it also may include any other points. However there is usually no sense in it.

Singular expansion is only a formal sum, we are not trying to evaluate it or even to simplify. This sum only shows us which poles does function have and what singular elements are there.

Example 11.9 (Singular expansion of gamma function). I'll remind you that gamma function is defined by the equality

$$\Gamma(s) = \int_0^{+\infty} e^{-x} x^{s-1} dx$$

Integral converges for $s \in \langle 0, +\infty \rangle$, but functional equation $s\Gamma(s) = \Gamma(s+1)$ allows to make a continuation of the gamma function to all complex plane except nonpositive integers. (It has been already noticed in example 11.3) Now we are going to obtain a singular expansion of gamma function in $\mathbb{C}$.
First af all, functional equation on gamma function implies

$$\Gamma(s) = \frac{\Gamma(s+m+1)}{s(s+1)(s+2)\dots(s+m)}, \quad m \in \mathbb{N} \cup \{0\}$$

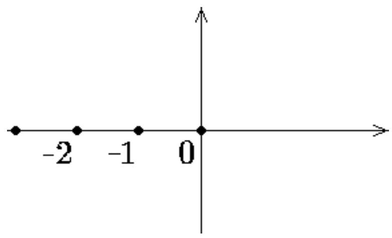
It demonstrates again, that gamma function has poles in nonpositive integers, but now we can learn much more about them. After making a kind of substitution of $-m$ for s we obtain

$$\Gamma(s) \sim \frac{(-1)^m}{m!} \frac{1}{s+m}, \quad \text{as } s \to -m$$

This means that left side divided by right side tends to 1 as s tends to $-m$. But we can understand it as that the right side is a singular element of gamma function in point $s = -m$. Now we know singular elements in all the poles of gamma function and thus we can write a singular expansion:

$$\Gamma(s) \asymp \sum_{k=0}^{+\infty} \frac{(-1)^k}{k!} \frac{1}{s+k}, \quad m \in \mathbb{C}$$

As it was already said, we are not trying to evaluate the sum, we only look on it and see what poles does Gamma function have, and what singular elements are there. For example we see that all poles of gamma function are simple.



Here the arrangement of gamma function poles is demonstrated at the picture.

11.4 Direct mapping

We have already seen in proposition 11.1, that asymptotics of function $f(x)$ in zero results on the leftmost boundary of the fundamental strip of Mellin transform $f^*(s)$. The same is with the asimptotics in infinity. It results on the rightmost boundary of the fundamental strip. Direct mapping is a theorem about what information can we obtain about Mellin transform, if we know a more detailed asymptotics of inital function $f(x)$ in zero and infinity.

Theorem 11.2 (Direct mapping). *Let $f(x)$ have a transform $f^*(s)$ with nonemty fundamental strip $\langle \alpha, \beta \rangle$. Let*

$$f(x) = \sum_{(\xi,k) \in A} c_{\xi,k} x^{\xi} (\log x)^k + \mathrm{O}(x^{\gamma}), \quad x \to 0$$

where $k \in \mathbb{N} \cup \{0\}$, $-\gamma < -\xi \leq \alpha$. Then $f^*(s)$ is continuable to a meromorphic function in the strip $\langle -\gamma, \beta \rangle$, where it admits the singular expansion

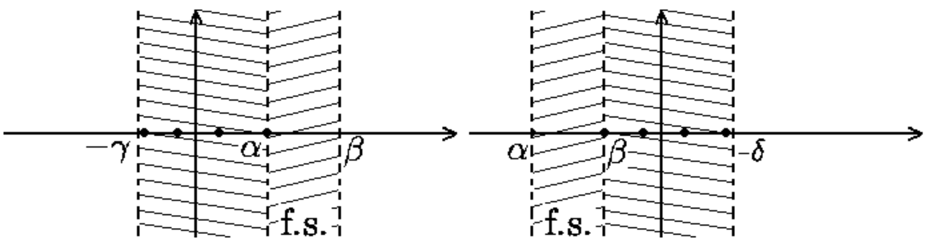
$$f^*(s) \asymp \sum_{(\xi,k)\in A} c_{\xi,k} \frac{(-1)^k k!}{(s+\xi)^{k+1}}, \quad s \in \langle -\gamma, \beta \rangle$$

If asymptotic expansion is given at infinity, then the similar result holds. The only difference is, that meromorphic continuation is to the right of the fundamental strip and there is an additional minus sign in singular expansion of transformed function. Look for explanation in two tables given below.
Singular expansion of transformed function presented in the theorem may seem to be confusing, but in real life there are no logarithms in asymptotic expansions. If there is one, it comes in the first degree. In this cases k equals 0 or 1, which makes singular expansion much more simple.
Let's put in a table some information that we know about connection between asymptotic expansion of initial function and properties of transformed one.

$f(x)$	$f^*(s)$
Order at 0: $\mathrm{O}(x^{-\alpha})$	Leftmost boundary of f.s. at $\Re(s) = \alpha$
Order at $+\infty$: $\mathrm{O}(x^{-\beta})$	Rightmost boundary of f.s. at $\Re(s) = \beta$
Expansion till $\mathrm{O}(x^{\gamma})$ at 0	Meromorphic continuation till $\Re(s) = -\gamma$
Expansion till $\mathrm{O}(x^{\delta})$ at $+\infty$	Meromorphic continuation till $\Re(s) = -\delta$

The next table contains information about connections between terms in asymptotic expansion of initial function and singularities of its Mellin transform.

$f(x)$	$f^*(s)$
Term $x^a(\log x)^k$ at 0	Pole with s.e. $\frac{(-1)^k k!}{(s+a)^{k+1}}$
Term $x^a(\log x)^k$ at $+\infty$	Pole with s.e. $-\frac{(-1)^k k!}{(s+a)^{k+1}}$
Term x^a at 0	Pole with s.e. $\frac{1}{s+a}$
Term $x^a \log x$ at 0	Pole with s.e. $-\frac{1}{(s+a)^2}$



Information contained in two tables is visualized on these two pictures.

Example 11.10. We have already obtained the singular expansion of gamma function by means of functional equation on it. Now we'll obtain the same result, but by applying the direct mapping theorem.
It was shown in example 11.3, that function $f(x) = e^{-x}$ has Mellin transform $f^*(s) = \Gamma(s)$. Asymptotic expansion of $f(x)$ at 0 is as follows:

$$f(x) = e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots = \sum_{j=0}^{M} \frac{(-1)^j}{j!} x^j + \mathrm{O}(x^{M+1})$$

Theorem states, that $f^*(s)$ is meromorphicaly continuable to $\langle -M-1, +\infty \rangle$. (And we do know it already) The asymptotic expansion of continued function is:

$$f^*(s) \asymp \sum_{j=0}^{M} \frac{(-1)^j}{j!} \frac{1}{s+j} \qquad s \in \langle -M-1, +\infty \rangle$$

The last holds for every positive M, so we can rewrite it in the following way:

$$f^*(s) \asymp \sum_{j=0}^{+\infty} \frac{(-1)^j}{j!} \frac{1}{s+j} \qquad s \in \mathbb{C}$$

and this is an asymptotic expansion in entire $\mathbb{C}$. This expansion we have already seen in example 11.9.

Example 11.11. In example 11.5 we intoduced the Mellin pair

$$g(x) = \frac{e^{-x}}{1-e^{-x}}; \quad g^*(s) = \zeta(s)\Gamma(s)$$

with fundamental strip $\langle 1, +\infty \rangle$. The asymptotic expansion of $g(x)$ at 0 is

$$g(x) = \frac{e^{-x}}{1-e^{-x}} = \sum_{j=-1}^{+\infty} B_{j+1} \frac{x^j}{(j+1)!}$$

where B_j are so-called Bernoulli numbers, we can suppose that this asymptotic expansion is a definition of Bernoully numbers. $B_0 = 1$, $B_1 = -1/2$. As in the previous example, asymptotic expansion is complete, i.e. we can write, that $g(x) = \ldots + \mathrm{O}(x^M)$ for any positive M we want. So, direct mapping theorem states, that transform $g^*(s)$ has meromorphic continuation to strip $\langle -\infty, +\infty \rangle = \mathbb{C}$. The singular expansion there is

$$g^*(s) = \zeta(s)\Gamma(s) \asymp \sum_{j=-1}^{+\infty} \frac{B_{j+1}}{(j+1)!} \frac{1}{s+j}, \qquad s \in \mathbb{C}$$

If we compare it with singular expansion of gamma funcion

$$\Gamma(s) = \sum_{j=0}^{+\infty} \frac{(-1)^j}{j!} \frac{1}{s+j}, \qquad m \in \mathbb{C}$$

we can extract the singular expansion of zeta function

$$\zeta(s) = \left[\frac{1}{s-1}\right]_{s=1} + \sum_{j=0}^{+\infty} \left[(-1)^j \frac{B_{j+1}}{j+1}\right]_{s=-j}$$

We see, that there are no poles in nonpositive integers, so we have obtained a result, that zeta function in meromorphic in entire $\mathbb{C}$ with the only pole in $s_0 = 1$. Since singular expansion is a sum of initial sums of Laurent expansions, we can extract information about values of zeta function in nonpositive integers.

$$\zeta(-m) = (-1)^m \frac{B_{m+1}}{m+1}, \qquad m \in \mathbb{N} \cup \{0\}$$

By the way, $B_{2k+1} = 0$ for $k \in \mathbb{N}$, so zeta function is zero in even negative integers, $\zeta(0) = -1/2$, and

$$\zeta(-2m+1) = -\frac{B_{2m}}{2m}, \qquad m \in \mathbb{N}$$

Example 11.12. There was another Mallin pair

$$f(x) = \frac{1}{x+1}; \quad f^*(s) = \frac{\pi}{\sin \pi s}$$

with fundamental strip $\langle 0, 1\rangle$. Asymptotic expansion at 0

$$f(x) = \frac{1}{1+x} = \sum_{n=0}^{+\infty} (-1)^n x^n, \quad x \to 0$$

implies posibility of continuation of transformed function to the left of fundamental strip, namly to $\langle -\infty, 1\rangle$. Singular expansion there is

$$f^*(s) \asymp \sum_{n=0}^{+\infty} \frac{(-1)^n}{s+n}, \quad s \in \langle -\infty, 1\rangle$$

If we consider asymptotic expansion at $+\infty$

$$f(x) = \frac{1/x}{1+1/x} = \sum_{n=0}^{+\infty} (-1)^{n-1} x^{-n}, \quad x \to +\infty$$

we learn about meromorphic continuation of transformed function to the right of fundumantal strip. Singular expansion to the right of fundamental strip is

$$f^*(s) \asymp -\sum_{n=1}^{+\infty} \frac{(-1)^{n-1}}{s-n}, \quad s \in \langle 0, \infty\rangle$$

Minus sign, as it has been already said, comes from the fact, that we consider the asymptotic expansion at infinity.
Now two singular expansions we can conbine into one and it gives us singular expansion of transformed function in entire $\mathbb{C}$

$$f^*(s) \equiv \frac{\pi}{\sin \pi x} \asymp \sum_{n \in \mathbb{Z}} \frac{(-1)^n}{s+n}, \quad s \in \mathbb{C}$$

As we see, expansion is right, but we could have obtained it much easier.

11.5 Converse mapping

Direct mapping theorem was a method of obtaining information about transformed function given the initial function. But usually it is not very interesting. We are interested in information about initial function and not about its transform. So now we are ready to introduce converse mapping theorem. It will be given exact formulation, but further we will apply the theorem without checking if exemined function satisfies all conditions.

Theorem 11.3 (Converse mapping). *Let f(x) be continious on $(0, +\infty)$ function, that has a transform $f^*(s)$ with nonempty fundammental strip $\langle \alpha, \beta\rangle$. Let $f^*(s)$ be meromorphically continuable to $\langle -\gamma, \beta\rangle$ with a fininte number of poles there, and be analytic on $\Re(s) = -\gamma$. Let $f^*(s) = \mathrm{O}(|s|^{-r})$ with $r > 1$ when $|s| \to +\infty$ in $\langle -\gamma, \beta\rangle$. If*

$$f^*(x) \asymp \sum_{(\xi,k) \in A} d_{\xi,k} \frac{1}{(s-\xi)^k}, \quad s \in \langle -\gamma, \beta\rangle$$

Then an asymptotic expansion of $f(x)$ at 0 is

$$f(x) = \sum_{(\xi,k)\in A} d_{\xi,k} \left(\frac{(-1)^{k-1}}{(k-1)!} x^{-\xi} (\log x)^{k-1} \right) + \mathrm{O}(x^{\gamma})$$

Converse mapping theorem is a theorem, that derives asymptotics of initial function from singularities of transformed. Next table includes some explanation of the theorem.

$f^*(f)$	$f(x)$
Pole at ξ	Term in asymptotic expansion $\approx x^{-\xi}$
left of f.s.	expansion at 0
right of f.s.	expansion at $+\infty$
Simple pole	
left: $\frac{1}{s-\xi}$	$x^{-\xi}$ at 0
right: $\frac{1}{s-\xi}$	$-x^{-\xi}$ at $+\infty$
Multiple pole	logarifmic factor
left: $\frac{1}{(s-\xi)^{k+1}}$	$\frac{(-1)^k}{k!}x^{-\xi}(\log x)^k$ at 0
right: $\frac{1}{(s-\xi)^{k+1}}$	$-\frac{(-1)^k}{k!}x^{-\xi}(\log x)^k$ at $+\infty$

Example 11.13. Consider a function $\phi(s) = \frac{\Gamma(s)\Gamma(\nu-s)}{\Gamma(\nu)}$. It is analytic in strip $\langle 0, \nu\rangle$. We know the singular expansion of gamma function an we can use this knwoledge to obtain the singular expansion of $\phi(s)$ in the strip $\langle -\infty, \nu\rangle$:

$$\phi(s) \asymp \sum_{j=0}^{+\infty} \frac{(-1)^j}{j!} \frac{\Gamma(\nu+j-1)}{\Gamma(\nu)} \frac{1}{s+j}, \quad s \in \langle -\infty, \nu\rangle$$

This $\phi(s)$ is a Mellin transform of some function $f(x)$, which can be obtained by applying of theorem 11.1

$$f(x) = \frac{1}{2\pi i}\int_{\nu/2-i\infty}^{\nu/2+i\infty} \phi(s)x^{-s}ds$$

Singularities of $\phi(s)$ in strip $\langle -\infty, \nu\rangle$ encode asymptotics for $f(x)$ at 0

$$f(x) = \sum_{j=0}^{+\infty} \frac{(-1)^j}{j!} \frac{\Gamma(\nu+j-1)}{\Gamma(\nu)} x^j, \quad x \to 0$$

One could have remembed binomial theorem and noticed, that function $\overline{f}(x) = (1+x)^{-\nu}$ has the same expansion at 0. This means, that difference $\varpi(x) = f(x) - \overline{f}(x)$ decase to zero faster, than any power of x, i.e. $\varpi(x) = \mathrm{O}(x^M)$, $\forall M > 0$. In fact, $\varpi(x) \equiv 0$, and we have indirectly obtained a new Mellin pair

$$f(x) = (1+x)^{-\nu}, \quad f^*(s) = \frac{\Gamma(s)\Gamma(\nu-s)}{\Gamma(\nu)}$$

Example 11.14. Consider a function $\phi(s) = \Gamma(1-s)\frac{\pi}{\sin \pi s}$. It is analitic in strip $\langle 0, 1\rangle$. We know singular expansion of every factor in this function and thus we can write singular expansion

$$\phi(s) \asymp \sum_{0}^{+\infty} (-1)^n n! \frac{1}{s+n}, \quad s \in \langle -\infty, 1\rangle$$

Applying of converse mapping theorem yields an asymptics for initial function $f(x)$

$$f(x) \sim \sum_{n=0}^{+\infty}(-1)^n n! x^n, \quad x \to 0$$

Symbol '$\sim$' is written instead of '$=$' to make an emphasis, that expansion is only asymptotic, i.e. series from the right side doesn't converge for any $x > 0$, but we can write, that $f(x) = \sum_{n=0}^{M}(-1)^n n! x^n + \mathrm{O}(x^M)$ as $x \to 0$ for any positive M. In fact, $f(x) = \int_0^{+\infty} \frac{e^{-t}}{1+xt} dt$.

11.6 Harmonic sums

Definition 11.5. A sum of the type $G(x) = \sum_k \lambda_k f(\mu_k x)$ is called harmonic sum with base function $g(x)$, frequencies μ_k and amplitudes λ_k. Series $\Lambda(s) = \sum_k \frac{\lambda_k}{\mu^s}$ is called the Dirichlet series.

Now we are going to discuss when the fifth basic property of Mellin transform holds for infinite harmonic sums. Next proposition is not formulated fully, base function and Dirichlet series are to satisfy some certain conditions about speed of growth, but all this is skipped here for simplicity.

Proposition 11.2. *The Mellin transform of the harmonic sum $G(x) = \sum_k \lambda_k f(\mu_k x)$ is defined in the intersection of the the fundamental strip of $g(x)$ and the domain of absolute convergence of the Dirichlet series $\Lambda(s) = \sum_k \frac{\lambda_k}{\mu^s}$ which is of the form $\Re(s) > \sigma_a$ (or $\Re(s) < \sigma_a$) for some real σ_a. In addition, $G^*(s) = \Lambda(s) g^*(s)$.*

In next (and last) two examples we'll derive two well known asymptotics. One is an asymptotic of harmonic numbers, and the second is Stirling's formula. But here we'll obtain complete asymptotics, which is not known so well.

Example 11.15 (Harmonic numbers). Consider the function

$$h(x) = \sum_{k=1}^{+\infty}\left[\frac{1}{k} - \frac{1}{k+x}\right] = \sum_{k=1}^{+\infty}\frac{1}{k}\frac{x/k}{1+x/k}$$

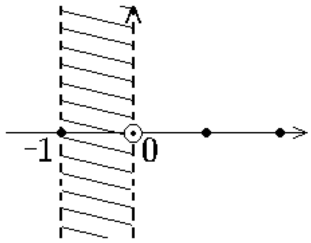
It is usuall harmonic sum with frequencies $\mu_k = 1/k$, amplitudes $\lambda_k = 1/k$ and base function $g(x) = \frac{x}{1+x}$. Function $h(x)$ is connected with harmonic numbers in very simple way: $h(n) = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = H_n$ for any $n \in \mathbb{N}$. Now we are going to evaluate Mellin transform of $G(x)$. To do this, we are to evaluate the Dirichlet series $\Lambda(s)$ and the tranform of base functon $g(x)$. $\Lambda(s) = \sum_k \frac{\lambda_k}{\mu^s} = \sum_{k=1}^{+\infty} k^{-1+s} = \zeta(1-s)$. Transform of base function is $g^*(s) = -\frac{\pi}{\sin \pi s}$ with fundamental strip $\langle -1, 0\rangle$, which is the result of applying of the first base property of Mellin transform. Notice, that the Dirichlet series $\Lambda(s) = \zeta(1-s)$ converges absolutely in the strip $\langle -1, 0\rangle$
Now we are ready to write the transform of $h(x)$, which is

$$h^*(s) = \Lambda(s) g^*(s) = -\zeta(1-s)\frac{\pi}{\sin \pi s}, \quad s \in \langle -1, 0\rangle$$

Before trying to write a singular expansion, we are to study zeta function some more. We know that $\zeta(s) \sim \frac{1}{s-1}$, it's a begining of Laurent expansion of zeta function at 1, but it's not enough for us. We want to know a coeficient of the zero degree in the Laurant expansion. Let it be γ, so we can write $\zeta(s) = \frac{1}{s-1} + \gamma + \cdots$ as $s \to 1$. This γ is so-called Euler constant, which is approximatly equal to 0.577215664901532860606512090082402431042159. Keeping all of this in mind, we can write singular expansion of $h^*(s)$:

$$h^*(s) \asymp \left[\frac{1}{s^2} - \frac{\gamma}{s}\right] - \sum_{k=1}^{+\infty}(-1)^k \frac{\zeta(1-k)}{s-k}, \quad s \in \langle -1, +\infty\rangle$$

Fundamental strip of the transform and its poles to the right of the fundamental strip are presented at the picture:



Double pole at zero is marked with a big circle.
Now we can finally apply converse mapping theorem and we obtain wishful asympotics:

$$H_n \sim \log n + \gamma + \sum_{k\geq 1} \frac{(-1)^k B_k}{k}\frac{1}{n^k} = \log n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \cdots$$

Example 11.16 (Stirling's formula). In this example we are to use the product decomposition of gamma function, that looks like

$$\log \Gamma(x+1) + \gamma x = \sum_{n=1}^{+\infty} \left[\frac{x}{n} - \log\left(1 + \frac{x}{n}\right)\right]$$

Let's denote the right side as $\ell(x)$. This is a harmonic sum with amplitudes $\lambda_n = 1$, frequencies $\mu_n = 1/n$ and a base function $g(x) = x - log(1+x)$. The Dirichlet series is $\Lambda(s) = \sum_{n=1}^{+\infty} \lambda_n \mu_n^{-s} = \sum_{n=1}^{+\infty} n^s = \zeta(-s)$. Transform of $g(x)$ can be evaluated by means of the first and the ninth basic properties of Mellin transform. The result is $g^*(s) = -\frac{\pi}{s\sin\pi s}$ with fundamental strip $\langle -2, -1\rangle$. As in the previous example, the Dirichlet series $\Lambda(s) = \zeta(-s)$ converges absolutely in the fundamental strip of transform of base function. So, fundamental strip of transform of $\ell(s)$ is $\langle -2,-1\rangle$ too.

$$\ell^*(s) = -\zeta(-s)\frac{\pi}{s\sin\pi s}, \quad s \in \langle -2,-1\rangle$$

We want to obtain an asymptotics in infinity, so we look on meromorphic continuation to the right of the fundamental strip. Laurant expansion of zeta function at zero is $\zeta(s) = \frac{1}{2} - \frac{1}{2}\log(2\pi) + \mathrm{O}(s)$, so singular expansion of $\ell^*(s)$ to the right of fundamental strip is

$$\ell^*(s) \asymp \left[\frac{1}{(s+1)^2} + \frac{1-\gamma}{(s+1)}\right] + \left[\frac{1}{2s^2} - \frac{\log\sqrt{2\pi}}{s}\right] + \sum_{n=1}^{+\infty} \frac{(-1)^{n-1}\zeta(-n)}{n(s+n)}, \quad s \in \langle -2, +\infty\rangle$$

Now we can apply converse mapping theorem and derive an asymptotics of function $\ell(s)$, which is

$$\ell(x) = \Big[x\log x - (1-\gamma)x\Big] + \left[\frac{1}{2}\log x + \log\sqrt{2\pi}\right] + \sum_{n=1}^{+\infty} \frac{B_{2n}}{2n(2n-1)}\frac{1}{x^{2n-1}}, \quad x \to +\infty$$

Here every item from singular expansion was converted to an item in asymptotic expansion without any simplification, but now we do some, keeping in mind, that $\Gamma(x+1) = x!$, so

$$\log(x!) = \log\left(x^x e^{-x}\sqrt{2\pi x}\right) + \sum_{n=1}^{+\infty} \frac{B_{2n}}{2n(2n-1)}\frac{1}{x^{2n-1}}, \quad x \to +\infty$$

This dazzling formula is named after Stirling and this is a good reason to finish the paper right here.

Chapter 12

Automaton Searching on Tries.

Mikhail Lakunin

Above there were considered a lot of algorithms that allow us to search for a pattern in a string and a few powerful methods to give asymptotic estimations of such algorithms. There is an extension of such algorithms, not for one pattern but for a regular expression. The algorithms that are to be reviewed there run on the preprocessed text (we use Patricia tree) and are of logarithmic *expected* time of the size of the text for the stricted class of regular expressions and in a sublinear *expected* time for all regular expressions.

It's the first such algorithm to be found with this complexity.

The main purpose of the text is to give understanding of what's going on, and therefore some of the technicals could be omitted. For the precise explanation the reader is referred to the original paper.

12.1 Structure of the paper

1. What we actually want to do or our task
2. The algorithms in a "few words"
3. Introducing indexing structures such as suffix tries and Patricia trees and a notation used in the Regular Languages theory(the Theory of formal languages)
4. Consider the simple algorithm for the stricted class of a regular expression that runs in logarithmic time.
5. Consider more complicated general algorithm.
6. Give estimation of the general algorithm considering sketch of technicals.
7. How to apply "general estimation theorem" to the particular cases.
8. Consider some heuristic for improving size of the query.
9. Open problems and conclusion (generalization of what we has spoken about)

12.2 Our task or what we want to do

We want to find efficient algorithm of finding all occurrences of a query given in the form of a regular language searching on the static preprocessed text. By finding all the occurrences we mean that we are going to find all the positions of the text from where one substring from our query set could start. In general queries are not of the big length, so there and below we state that the size of the query is bounded by the constant. We use there automaton that searches through the indexing tree and we state that our algorithm is pretty efficient, i.e. works in sublinear expected time (in average case). Moreover there exist small stricted subset of such queries that the expected time will be logarithmic. Besides we consider a powerful tool for estimating expected time of given automaton, the theorem that allows it on basis of some knowledge about incidence matrix of the automaton. And after all of that we suggest an heuristic for enhancing or for clarifying the query that helps to work all the algorithm faster.

12.3 Algorithm in a "few words"

As in previous parts we've got a large input text that is static or in other words we can construct indexing structure on it and we don't bother about how to reconstruct it if the text will change. So we construct a trie (from word retrieval) that is indexing binary tree (we suppose our alphabet is binary; if it's not we just use some encoding to obtain binary text and hence binary alphabet) was discussed by Olga Sergeeva. It becomes obvious to us that we can find arbitrary substring in the text in height of indexing tree length. In the case of perfectly balanced tree it's $\log_2 n$, where n is the length of the string.
Now we're able to find a single substring in a large text in logarithmic of the size of the text expected time (in average case, considering uniform distribution) Let's complicate the task. If set of all the substrings we are to find in the text could be represented as a prefix from *minimal preword set* concatenated with any continuation. For such a set we can state that it's possible to search for that query in a time $o(\|query\|)$ independently of the size of the answer, i.e. in logarithmic expected time of size of the text.
Then we consider general regular expression query, construct Deterministic Automaton for it and traverse it on the index trie. It could be showed that Automaton don't visit all the vertices of the index trie (the quantity of which is $O(n)$),but only a part of them in average case, that's the main idea of sublinear expected time.

12.4 Basic index structures and notation

To approach crystal understanding we need to introduce or just to help to recall some basics facts we are going to manage with soon after.
As it has been said we consider only binary alphabet, any other alphabet could be easily converted to the binary one with some encoding

12.4.1 Indexing structures

Definition 12.1. Trie is a binary index tree for a text(tree each edge of that is marked with a character), that satisfy following conditions:

1. each path of it (from the root to a leaf) consist of the prefix of one and only one of the suffixes of the text
2. paths are as short as possible satisfying first condition
3. each leaf of it is marked with the position of the first character of the corresponded suffix

The main problem of the pure trie is that the upper bound of the number of inner nodes is $O(n^2)$ that approaches in unbalanced tries, that's why in practice enhanced structure is of often use. It's so-called Patricia tree.

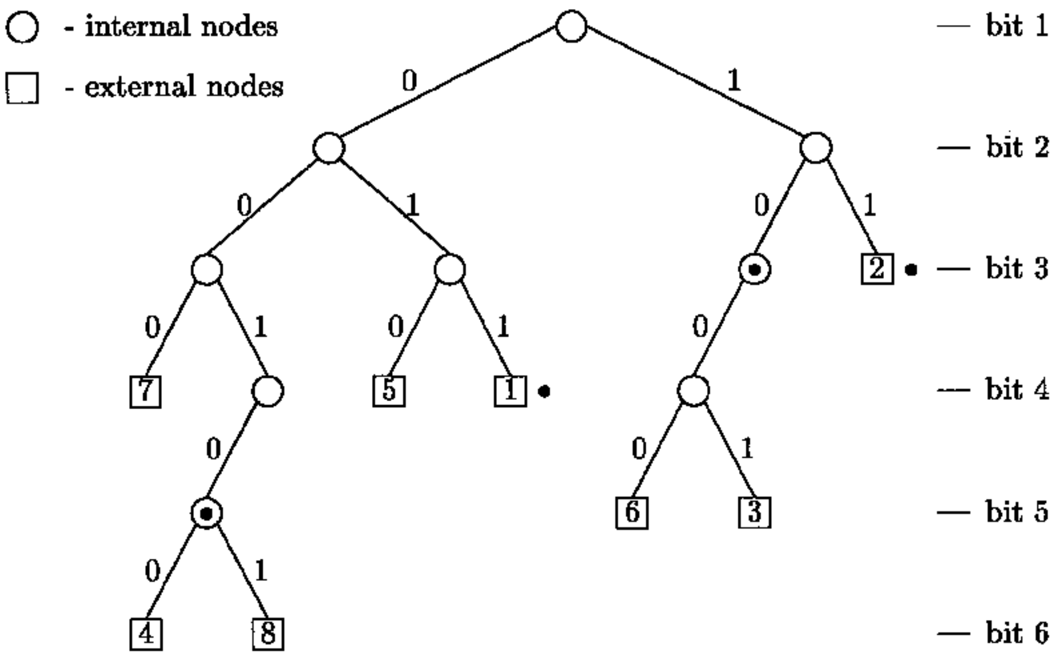


Figure 12.1: Binary trie (external node label indicates position in the text) for the 01100100010111

Definition 12.2. Patricia tree constructed on the basis of the trie, but with one enhancement. We exclude such nodes that have only one descendant and one ancestor, in other words we exclude nodes that are not leafs and that are not nodes of the "bifurcation". To not miss the correspondence with the initial text we remember the quantity of steps we need to omit to get to the next "bifurcation" node.

Note. Asymptotically in average case:
Height of the trie is: $2log_2(n) + O(log_2(n))$ [Reg81]
Height of the Patricia tree: $log_2(n) + O(log_2(n))$ [Reg81]

12.4.2 Basic definitions of the Theory of Formal Languages

General definition and notation.

1. $\mathbf{\Sigma}$ is a set of all characters (Alphabet), in some cases one character from the set
2. ε is an empty string
3. $\mathbf{xy}$ is concatenation of strings x and y
4. $\mathbf{w = xyz}$ if w is concatenation of strings x, y, z then:
   (a) $\mathbf{x}$ is a prefix of w
   (b) $\mathbf{z}$ is a suffix of w
   (c) $\mathbf{y}$ is a substring of w

Let's define operations we are to use specifying regular expression (RE).

1. $\mathbf{r}, \mathbf{q}$ are sets of strings
2. $\mathbf{r} + \mathbf{q}$ is a unite of these sets $(r + q = r \cup q)$
3. $\mathbf{r}?$ is one or zero occurrences of r ($r? = \varepsilon + r$ in our notation)
4. $\mathbf{r^k}$ is k occurrences of r
5. $\mathbf{r^{\leq k}}$ is from zero to k occurrences of r ($r^{\leq k} = \sum_{i=0}^{k} r^i$)
6. $\mathbf{r}^*$ is Kleene closure,i.e. any number of occurrences of r $(r^* = \sum_{i=0}^{+\infty} r^i)$
7. $\mathbf{r}^+$ is one or more occurrences of r $(r^+ = \sum_{i=1}^{+\infty} r^i)$

12.5 Algorithm for a restricted class of regular expression

There we consider algorithm of searching for the stricted class of queries. It was briefly reviewed above, there we define it more exactly.
Let's introduce new expression class so-called *prefixed regular expressions* that is subclass of regular expressions class that was defined above.

Definition 12.3. Prefixed regular expression (PRE) is:

1. $\emptyset$ is a PRE (the empty set)
2. ε is a PRE($\{\varepsilon\}$ the set of empty string)
3. for each $a \in \Sigma$, a is a PRE ($\{a\}$)
4. if $p, q \in$ PRE and $r \in$ RE (such that $\varepsilon \in r$) and $x \epsilon \Sigma$ then:
   (a) $p + q$ is a PRE (union)
   (b) xp is a PRE (concatenation with a character on the left)
   (c) pr is a PRE (concatenation with ε-regular expression on the right)

(d) $p*$ is a PRE

Example 12.1. This is PRE

$$ab(bc^* + d^+ + f(a+b))$$

Example 12.2. This is not PRE

$$a\Sigma^* b$$

Example 12.3. This is not PRE too

$$(a+b)(c+d)(e+f)$$

Statement. There exist such **unique** and **finite** subset(we call it *preword* set) of the (given) PRE set that satisfy following conditions:

1. for every word in the PRE set there exist unique prefix from *preword* set
2. for every word from the *preword* set there's no other prefix in the set

To search a PRE query, we construct a tree (so-called Complete Prefix Trie) in similar way as we construct trie, but we use not suffixes of the text as we do in case of trie but all the strings in *preword* set of the query. And then traverse through that tree and through our index trie simultaneously for an answer.

Definition 12.4. Complete Prefix Trie (CPT) is the trie of the set of the strings such that:

1. there are no truncated paths(as in Patricia tree), that is, every word corresponds to a complete path in the trie
2. if a word x is a prefix of another word w, then only x is stored in the trie.

The second rule has appeared since the search for x is sufficient to also find the occurrences of w. We are going to find only starting positions, aren't we?
We've constructed the Complete Prefix Trie so we are now able to traverse through that.
We traverse simultaneously, if possible, the complete prefix trie(CPT) and the trie (Patricia tree) of all suffixes of the text (the index). That is, follow at the same time a 0 or 1 edge(we consider binary alphabet), if they exist in both the trie and the CPT. All the subtrees in the index associated with terminal nodes in the complete prefix trie are the answer. As in prefix searching, because we may skip bits while traversing the index, a final comparison with the actual text is needed to verify the result.
Let's formulate the algorithm.

Algorithm. Searching for a query:

1. Construct the Complete Prefix Trie from the query
2. Traverse simultaneously the complete prefix trie and Patricia tree to obtain an answer.

Note. Since we may skip bits while traversing the index(if we used Patricia tree), a final comparison with the actual text is needed to verify the result.

Because the number of nodes of the complete prefix trie of the *preword* set is $O(|query|)$, the search time is also $O(|query|)$.

Conclusion. It is possible to search a PRE-query in time $O(|query|)$ independently of the size of the answer.

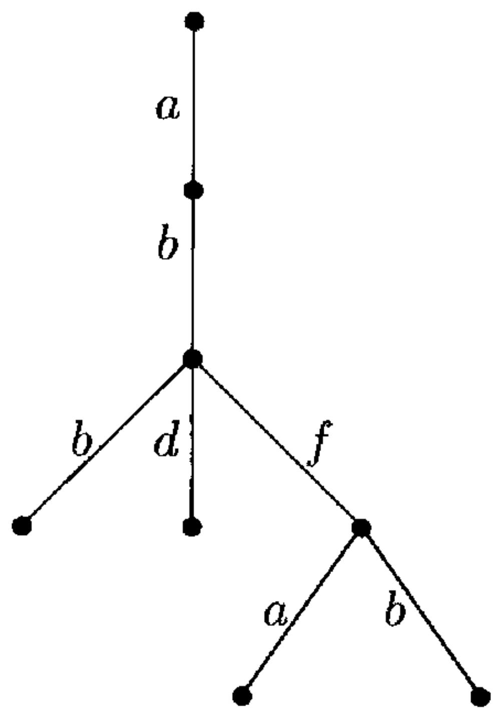


Figure 12.2: Complete Prefix Trie for a query $ab(bc^* + d^+ + f(a+b))$

12.6 General algorithm

The extension of the previously discussed idea leads to general algorithm of searching query given as regular expression.

12.6.1 What we want to do there

There we present the algorithm that can search for arbitrary regular expression in time sublinear in n on the average. For this, we simulate a DFA (Deterministic Finite Automaton) in a binary trie built from all the suffixes of a text.
Since the situation is pretty similar with the previous algorithm. Let's start from presenting algorithm itself.

Algorithm. General Automaton Search

1. Convert the regular expression(query) into minimized DFA(independent of the size of the text)
2. Eliminate outgoing transitions from final states
3. Convert character DFA into binary DFA (Each state will then have at most two outgoing transitions, one labeled with 0 and one labeled with 1), i.e. applying some encoding
4. Simulate binary DFA on the binary trie of all suffixes we've constructed(See pictures of DFA and binary index trie).
   That is:
   (a) root of the tree with initial state.
   (b) for any internal node associated with state i, associate its left descendant with state j, if $i > j$ for a bit 0, and associate its right descendant with state k, if $i > k$ for a bit 1.
5. For every node of the index associated with a final state, accept the whole subtree and halt.
6. On reaching an external node, run the remainder of the automaton on the single string determined by this external node.

12.7 Efficiency Estimation for the General Algorithm

The precise average case analysis of the above algorithm is not simple. Some explanations were given in "few words" in the start of the paper, There I'll try to clarify the situation a bit, for precise explanation reader is referred to the original paper.

12.7.1 The structure of the proof

The situation is as follows. We have Patricia indexing trie with $O(n)$ nodes and the DFA(Deterministic finite automaton) and we want to calculate how many nodes of the Patricia indexing trie will be visited by automaton. We want to prove that this quantity is less then $O(n)$ (asymptotically).
There I want to give a sketch of the proof on step by step basis.

1. We introduce N_n^i, i.e. the quantity of nodes of our index Patricia tree that was visited by Automaton when the quantity of suffixes in the tree is n and we've engaged our automaton on the i-th node.

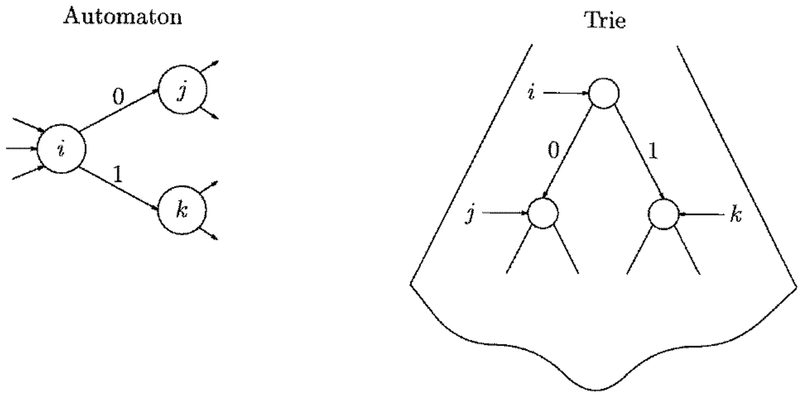


Figure 12.3: DFA with corresponded index trie

4

2. We note that our automaton from each node can go only in 2 directions in the index tree and taking into account all the possible combinations of the ways suffixes could go in i-th node we can say , therefore we can say that:

$$N_n^i = 1 + \frac{1}{2^n}\left(\sum_{l=0}^{n}\binom{n}{l}\left(N_l^j + N_{n-l}^k\right)\right)$$

where j, k are states DFA goes on $0, 1$ correspondingly (see picture with DFA and trie)

3. Now it looks like equation could be solvable with generating function. We apply generate function,
4. and convert it to matrix equation writing it for all the starting nodes in a unit.
5. Then we "solve" equation and find the generating function. The generating function equals to the matrix series, where the matrix is incidence matrix of DFA.
6. Then we convert it to a Jordan form and decompose matrix equation looking at the single values in a matrix.
7. Then we apply Mellin transform method to get some knowledge about this single values series and about matrix series
8. Using that we got some estimation of eigenvalues and information about N, i.e. about number of nodes was visited and about number of comparison DFA made (in average case)
9. Finally we obtain following theorem(taking in account number of steps we need for verifying in case of Patricia tree):

   Theorem 12.1. *The expected number of comparisons performed by a minimal DFA for a query q represented by its incidence matrix H while searching the trie of n random strings is sublinear, and given by:*

   $$O\left((log_2 n)^t \, n^r\right)$$

   ,where $r = log_2\lambda$,$\lambda = max_i(|\lambda_i|)$,$t = max_i(m_i - 1$,such that $|\lambda| = \lambda_i)$, and the λ_i are the eigenvalues of H with multiplicities m_i.

12.8 How to apply "general estimation theorem" to the particular cases

To analyze average time of searching for particular query or particular type of the query ,we need to consider DFA of the query and its incidence matrix and eigenvalues of the matrix. And then we can apply "the main theorem" to obtain estimation.

Example 12.4. For example, DFAs having only cycles of length 1, have a largest eigenvalue equal to 1, but with multiplicity proportional to the number of cycles, obtaining a complexity of $O(polylog(n))$.

Example 12.5. For the regular expression $(0(011)0)^*1$, the eigenvalues are:

$$2^{1/3}, -\frac{1}{2}(2^{1/3} - 3^{1/2}2^{1/3}i), -\frac{1}{2}(2^{1/3} - 3^{1/2}2^{1/3}i), 0$$

, and the first 3 have the same modulus. The solution in this case is $N_n^1 = O(n^{1/3})$

12.9 Heuristic for optimizing the query

12.9.1 What we mean by optimizing

In this section, we present a general heuristic, which we call **substring analysis**, to plan what algorithms and order of execution should be used for a generic pattern matching problem, which we apply to regular expressions.
The aim of this section is to find from every query a set of necessary conditions that have to be satisfied.

12.9.2 Substring graph

Definition 12.5. Substring graph of a regular expression is an acyclic directed graph such that each node is labeled by a string. And its defined recursively by the following rules:

1. $G(\varepsilon)$ is a single node labeled ε.
2. $G(x)$ for any $x \in \Sigma$ is a single node labeled with x.
3. $G(s+t)$ is the graph built from $G(s)$ and $G(t)$ with an ε-labeled node with edges to the source nodes and an ε-labeled node with edges from the sink nodes.
4. $G(st)$ is the graph built from joining the sink node of $G(s)$ with the source node of $G(t)$, and relabeling the node with the concatenation of the sink label and the source label.
5. $G(r^+)$ are two copies of $G(r)$ with an edge from the sink node of one to the source node of the other.
6. $G(r^*)$ is two ε-labeled nodes connected by an edge .

12.9.3 How to Use

reducing the size of the query

After building $G(q)$, we search for all node labels in $G(q)$ in our index of suffixes, determining whether or not that string exists in the text ($O(|q|)$ time). For all nonexistent labels, we remove:

1. the corresponding node,

2. adjacent edges,
3. and any adjacent nodes (recursively) from which all incoming edges or all outgoing edges have been deleted.

This reduces the size of the query.

estimating final answer size

From the number of occurrences for each label we can obtain an upper bound on the size of the final answer to the query:

1. for adjacent nodes (serial, or "and" nodes) we multiply both numbers
2. for parallel nodes ("or" nodes) we add the number of occurrences

Note. ε-nodes are treated in special way.

Managing ε-nodes:

1. consecutive serial ε-nodes are replaced by a single ε-node.
2. chains that are parallel to a single ε-node, are deleted
3. the number of occurrences in the remaining ε-nodes is defined as 1

After the simplifications,ε-nodes are always adjacent to non-ε-nodes, since ε was assumed not to be a member of the query

12.10 Open problems and conclusion

We have shown that using a trie or Patricia tree, we can search for many types of string searching queries in logarithmic average time, independently of the size of the answer. We also show that automaton searching in a trie is sublinear in the size of the text on average for any regular expression, this being the first algorithm found to achieve this complexity. Similar ideas have been used since for approximate string searching by simulating dynamic programming over a digital tree [Gonnet et al. 1992; Ukkonen 1993], also achieving sublinear time on average. In particular, Gonnet et al. [1992] have used this algorithm for protein matching.

In general, however, the worst case of automata searching is linear. For some regular expressions and a given algorithm it is possible to construct a text such that the algorithm must be forced to inspect a linear number of characters. The pathological cases consist of periodic patterns or unusual pieces of text that, in practice, are rarely found.

Finding an algorithm with logarithmic search time for any RE query is still an open problem [Galil 1985]. Another open problem is to derive a lower bound for searching REs in preprocessed text.

Bibliography

[BFC00] Michael A. Bender and Martin Farach-Colton, *The lca problem revisited*, Latin American Theoretical INformatics, may 2000, pp. 88–94.

[BYG96] Ricardo A. Baeza-Yates and Gaston H. Gonnet, *Fast text searching for regular expressions or automaton searching on tries*, Journal of the ACM **43** (1996), no. 6, 915–936.

[CL94] William I. Chang and Eugene L. Lawler, *Sublinear approximate string matching and biological applications*, Algorithmica **12** (1994), 327–344.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to algorithms*, MIT Press, 1990.

[FGD95] Philippe Flajolet, Xavier Gourdon, and Philippe Dumas, *Mellin transforms and asymptotics: Harmonic sums*, Theoretical Comp. Science **144** (1995), no. 1–2, 3–58.

[FS86a] Philippe Flajolet and Robert Sedgewick, *Digital search trees revisited*, SIAM J. Comput. **15** (1986), no. 3, 748–767.

[FS86b] ———, *Digital search trees revisited*, SIAM J. Comput. **15** (1986), no. 3, 748–767.

[FS95] ———, *Mellin transforms and asymptotics: Finite differences and rice's integral*, Theoretical Computer Science **144** (1995), no. 1-2, 101–124.

[FS98] A. Frieze and W. Szpankowski, *Greedy algorithms for the shortest common superstring that are asymptotically optimal*, Algorithmica **21** (1998), 21–36.

[GO81] L. J. Guibas and A. M. Odlyzko, *String overlaps, pattern matching, and nontransitive games*, J. of Combinatorial Theory, Series A **30** (1981), 183–208.

[Gus97] Dan Gusfield, *Algorithms on strings, trees, and sequences: Comp. science and computational biology*, Cambridge University Press, 1997.

[GV00] Roberto Grossi and Jeffry Scott Vitter, *Compressed suffix arrays and suffix trees with applications to text indexing and string matching*, 32nd Symp. on Theory of Comput., 2000, pp. 397–406.

[Jac89] Guy Jacobson, *Sapce-efficient static trees and graphs*, IEEE Symposium on Foundations of Computer Science (1989), 549–554.

[KS03] Juha Kärkkäinen and Peter Sanders, *Simple linear work suffix array construction*, Proc. 30th Int. Colloq. on Automata, Languages and Programming (ICALP), LNCS, vol. 2719, Springer, 2003, pp. 943–955.

[MM93] Udi Manber and Gene Myers, *Suffix arrays: A new method for on-line string searches*, SIAM J. Comput. **22** (1993), no. 5, 935–948.

[NBY00] G. Navarro and R. Baeza-Yates, *A hybrid indexing method for approximate string matching*, Journal of Discrete Algorithms (JDA) **1** (2000), no. 1, 205–209, Special issue on Matching Patterns.

[Reg81] M. Regnier, *On the average height of trees in digital search and dynamic hashing*, Inf. Proc.Lett **13** (1981), 64–67.

[RS98a] M. Regnier and W. Szpankowski, *Complexity of sequential pattern matching algorithms*, Proc. RANDOM, LNCS, vol. 1518, Springer, 1998, pp. 187–199.

[RS98b] Mireille Regnier and Wojciech Szpankowski, *On pattern frequency occurrences in a markovian sequence*, Algorithmica **22** (1998), 631–649.

[Sad00] Kunihiko Sadakane, *Compressed text databases with efficient query algorithms based on the compressed suffix array*, Proc. ISAAC, LNCS, vol. 1969, Springer, 2000, pp. 410–421.

[Szp93] Wojciech Szpankowski, *Asymptotic properties of data compression and suffix trees*, IEEE Transactions on Information Theory **39** (1993), no. 5, 1647–1659.

[Szp00] ———, *Average case analysis of algorithms on sequences*, 1 ed., Wiley-Interscience, 2000.

[Ukk95] E. Ukkonen, *On-line construction of suffix trees*, Algorithmica **14** (1995), 249–260.