Technische Universität München
Department of Informatics
Chair for Efficient Algorithms
Prof. Dr. Ernst W. Mayr/Dr. Jens Ernst
Johannes Nowak

Summer Term 2005
Exercise Sheet 2
3. Mai 2005

Selected Topics in Computational Biology

Due: 10.05.2005 **after** *the lecture*

Exercise 1 (10 points)

Construct a suffix tree for the string $CTGCCTGA$ with Ukkonen's algorithm. Describe all steps of the construction in detail.

Exercise 2 (10 points)

Show that the size of a suffix tree for a string of length n is $\Theta(n^2)$ in the worst case if the edge labels are written explicitly.

Exercise 3 (10 points)

Let $S = \{S_1, \ldots, S_k\}$ be a set of k strings of total length n. A *generalized* suffix tree for S is a compact Σ^+-tree that represents all the suffixes of strings in S_k.

a) Give an $O(n)$-algorithm that constructs a generalized suffix tree for S.

b) Consider a generalized suffix tree in a dynamic setting, that is, arbitrary strings can be added to and deleted from S. How can the generalized suffix tree be efficiently maintained? Why is it easier to delete a whole string of length m than deleting m arbitrary suffixes from the tree?

Exercise 4 (10 points)

Let $S = \{S_1, \ldots, S_k\}$ be a set of k strings of total length n. Develop an $O(n)$-algorithm that finds every string S_i that is a substring of some other string S_j with $1 \leq i, j \leq k$, $i \neq j$.

Programming Task

Implement Ukkonen's algorithm for suffix tree construction and a function that tests in time $O(m)$ whether a pattern P of length m is a substring of T or not. If the pattern occurs your program should return the number of occurrences, the starting position of the first occurrence, and the starting position of the last occurrence of the pattern in the text. Note that there might be $\Omega(n)$ occurrences of a pattern. Thus, the tree has to be augmented with additional information for efficiently answering such queries.

Please respect the following hints and technical details:

- Your implementation should be in plain ANSI C.

- The input strings consist of arbitrary ASCII symbols (thus, the alphabet size is 256). The text and the pattern to search for are specified in a file. The symbol $ is used as an end marker, both for the text and the patterns. Your program gets the name of the file as command line argument. The first string in the file is the text for which a suffix tree has to be constructed. The other strings correspond to patterns and should be searched for one by one, see the example below.
- You can use the example files available at the exercises homepage to test your implementation. There are six input files along with the corresponding correct output.
- Try to use a space efficient representation of suffix trees, i.e. pay attention to the constants.
- The output of your program should follow the example below.

Example for input and output of the program

Input:

```
Heute programmieren wir
einen Algorithmus zur
Suche in Texten.$
te$
wir
ein$
xx$
```

Output:

```
Suffix tree: time 0.5 sec
Pattern 1: 2 occurrences, first pos. 4, last pos. 59, time: 0.1 sec
Pattern 2: 1 occurrences, pos. 21, time: 0.2 sec
Pattern 3: 0 occurrences, time: 0.1 sec
```