INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR EFFIZIENTE ALGORITHMEN

Skriptum

zur Vorlesung

Internet-Algorithmik

gehalten im Sommersemester 2004

von

Sven Kosub

Erstellt unter Mithilfe von:

Moritz G. Maaß und Mojca Miklavec

29. Januar 2005

Version 1.0

Vorwort

Dieses Skriptum bietet die Grundlage für die Vorlesung „Internet-Algorithmik“, die erstmalig im Sommersemester 2004 an der Technischen Universität München stattfand. Die Vorlesung hat einen Umfang von vier Semesterwochenstunden und die zugehörige Übung einen Umfang von zwei Semesterwochenstunden.

Viele der Materialien sind noch nicht in Monographie- oder Lehrbuchform veröffentlicht. Deshalb sind die folgenden Literaturangaben nur als Referenz für die algorithmischen oder technologischen Grundtechniken, die im Rahmen der Vorlesung vorausgesetzt werden, geeignet.

Standardlehrbücher für den Entwurf und Analyse von Algorithmen:

- [GT02] Michael T. Goodrich und Roberto Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples.* 2. Auflage, John Wiley & Sons, Inc., Chichester, UK, 2002.
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein. *Introduction to Algorithms.* 2. Auflage, The MIT Press, Cambidge, MA, 2001.

Standardlehrbuch für den Bereich der Netzwerkarchitekturen:

- [KR03] James F. Kurose und Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* 2. Auflage, Addison-Wesley, Boston, MA, 2003.

  Eine deutsche Version der ersten Auflage ist unter dem Titel *Computernetze: Ein Top-Down- Ansatz mit Schwerpunkt Internet* ist 2001 bei Pearson Eduction in München erschienen.

Inhaltsverzeichnis

1 Einleitung

Goodrich und Tamassia [GT02] verstehen unter Internet-Algorithmik das Studium des Entwurfs und der Analyse von Algorithmen und Datenstrukturen für kombinatorische Probleme, die primär durch das Internet und Internet-Anwendungen motiviert sind.

1.1 Internet: Begriffe und Definitionen

Was ist das Internet?

1. Eine soziologische Interpretation des Internets wäre sozialer Raum, öffentlicher Raum oder Cyberspace. Allen gemeinsam scheint die Auffassung, dass das Internet aufgefasst wird als Menge aller Rechner, Nutzer und Handlungen der Nutzer, die durch Rechner bzw. Rechentechnik vermittelt werden.

2. Unter infrastrukturellen Gesichtspunkten könnten wir das Internet als Menge aller Rechner (und Verbindungsleitungen) auffassen.

3. Unter dem Aspekt der Netzwerkarchitektur ist das Internet die Menge aller Rechner, die Nachrichten gemäß vereinbarter Protokolle austauschen.

Wir werden alle drei Aspekte in unsere Betrachtungen einbeziehen.

Grundlegend für alle Aspekte ist die Verwendung von Protokollen.

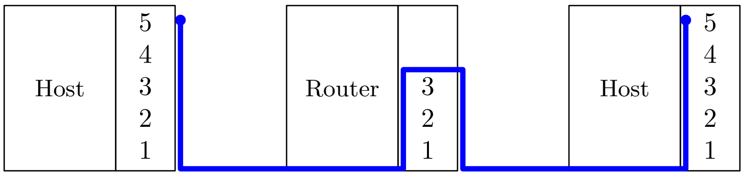
Was ist ein Protokoll?

Nach einer gängigen Definition ist ein Protokoll eine Definition von Format und Reihenfolge der Nachrichten, die zwischen zwei oder mehr kommunizierenden Einheiten ausgetauscht werden, sowie die Handlungen, die bei Übertragung oder Empfang einer Nachricht oder dem Auftreten eines Ereignisse unternommen werden.

Im Allgemeinen müssen immer mehrere Protokolle gleichzeitig berücksichtigt werden. Wir sprechen dann auch von Protokollstapel. Je nach Abstraktionsgrad werden mehrere Ebenen im OSI-Schichtmodell zusammengefasst. Der von uns verwendete Protokollstapel ist:

Schicht	Stack	Dateneinheit	Beispiele
5	Anwendung (Application Layer)	Nachricht	HTTP (Web) SMTP (E-Mail) FTP (Dateiübertragung)
4	Transport (Transport Layer)	Segment	TCP (mit Verbindungsaufbau) UDP (verbindungslos)
3	Vermittlung (Network Layer)	Datagramm	IP BGP (Routing)
2	Sicherung (Data Link Layer)	Rahmen	Ethernet PPP
1	Bitübertragung (Physical Layer)	Signal (1-PDU)	je nach Medium

Zum Beispiel kann Datenfluss zwischen Anwendungen, die auf zwei Hosts laufen, wie folgt aussehen.



1.2 Internetanwendungen

Typische Motivationen für Internetauftritt:

- Geschäftsmodelle / Geschäftserfolg (Ebay, Amazon, Banken, ...)
- Unterhaltung (Musik, Spiele, ...)
- Kommunikation (Email, Internettelephonie, ...)
- Resourcenerweiterung (SETI@home, ...)
- Interessengruppen (Foren, Flash Mobs / Smart Mobs, ...)

Rücksicht in dieser Vorlesung:

- Technologien, die auf die globale Verfügbarkeit von Wissensressourcen abzielen

- Internet als Mittel zur Demokratisierung des Wissenszugangs

Charakteristika der Technologien:

- Handhabung hoher Datendichte
- dezentrale (anarchische) Datenverteilung
- unstrukturierte Informationen

Beispiele:

- Suchmaschinen
- Diagnoseinstrumente
- ...

1.3 Kombinatorische Probleme

Typische Probleme:

- Optimierungsprobleme (vor allem auf Netzwerken)
- Statistiken
- Inferenzprobleme
- ...

Verwenden einen Top-Down-Zugang (d.h. absteigend im Abstraktionsgrad)

1.4 Algorithmische Techniken

Klassische Techniken, die in der Vorlesung auftauchen:

- Asymptotische Analyse (von Laufzeit- und Platzmaßen)
- NP-Vollständigkeit
- Kompetitive Analyse
- Approximation
- Randomisierung

Neuere Technik: Algorithmische Implementierungstheorie

- Analyse und Entwurf von Mechanismen (= Protokoll + Berechnungsanteil)
- Spieltheoretische (mikroökonomische) Analyse

1.5 Themenliste

Geplante Themen:

1. Internet-Analytik (betrifft alle Schichten im OSI-Modell)
2. Gestaltung des Internetbetriebs (betrifft Schichten 3 und 4 im OSI-Modell)
3. Informationsverteilung im Internet (betrifft die Anwendungsschicht)
4. Informationsgewinnung im Internet (betrifft die Anwendungsschicht)

Nicht behandelte Themen:

1. Algorithmische Aspekte der Internet-Ökonomie
2. Internet-Sicherheit
3. Formale Modelle für das Internet
4. Formale Aspekte von XML
5. Mobiles Internet

Datenanalyse 2

Ziel der Datenanalyse: Entwurf und Analyse guter Algorithmen zur Analyse großer und dichter Datenmengen

2.1 Texte

Wesentlicher Inhalt: Suche „kurzer“ Zeichenfolgen in „langen“ Zeichenfolgen

2.1.1 Definitionen und Notationen

Wir führen zunächst eine Reihe notwendiger Begriffe ein:

- Eine Alphabet Σ ist eine endliche Menge von Symbolen (Buchstaben).
- Ein Wort (String, Zeichenkette) s über Σ ist eine endliche Folge von Symbolen aus Σ.
- Σ^* ist die Menge aller Wörter über Σ.
- Für $s \in \Sigma^*$ wird mit $|s|$ die Länge von s bezeichnet, d.h. für $s = s_0 \ldots s_{n-1}$ gilt $|s| = n$.
- Das leere Wort wird mit ε bezeichnet, d.h. es gilt $|\varepsilon| = 0$.

Definition 2.1 *Es sei $s = s_0 s_1 \ldots s_{n-1}$ ein Wort über Σ.*

1. *Ein Wort $s' \in \Sigma^*$ der Länge m heißt* Teilwort *von s, falls ein $i \in \{0, 1, \ldots, n-1\}$ existiert mit $s_i s_{i+1} \ldots s_{i+m-1} = s'$.*
2. *Ein Wort $s' \in \Sigma^*$ der Länge m heißt* Präfix *von s, falls $s' = s_0 s_1 \ldots s_{m-1}$ gilt. Ist s' ein Präfix von s mit $s' \neq s$, so heißt s'* echtes Präfix *von s.*
3. *Ein Wort $s' \in \Sigma^*$ der Länge m heißt* Suffix *von s, falls $s' = s_{n-m} s_{n-m+1} \ldots s_{n-1}$ gilt. Ist s' ein Suffix von s mit $s' \neq s$, so heißt s'* echtes Suffix *von s.*
4. *Eine Wort $s' \in \Sigma^*$ heißt* Rand *von s, falls s' sowohl Präfix als auch Suffix von s ist. Der* eigentliche Rand $\partial(s)$ *von s ist der längste Rand von s, der verschieden von s ist.*

Algorithmus: SimpleMatching
Eingabe: Suchwort s mit $|s| = m$ und Wort t mit $|t| = n$
Gesucht: Alle Positionen, ab denen s als Teilwort in t vorkommt

```
WHILE i ≤ n − m
    WHILE j ≤ m AND s_j = t_{i+j}
        j := j + 1
        IF j = m
            Gebe i zurück
    i := i + 1
    j := 0
```

Abbildung 2.1: Einfacher Algorithmus für exakte Suche

2.1.2 Ein einfacher Algorithmus für exakte Suche

Wir betrachten folgendes Problem: Suche ein Teilwort $s = s_0 s_1 \ldots s_{m-1}$ in einem Wort $t = t_0 t_1 \ldots t_{n-1}$. Genauer sind wir daran interessiert, alle Positionen des Vorkommens von s in t zu bestimmen.

Der Ausgangspunkt für die nachfolgenden Überlegungen ist der in Abbildung 2.1 beschriebene naive Algorithmus. Eine offentischtliche Analyse des Algorithmus ergibt folgende Abschätzung der Laufzeit an Hand der benötigten Vergleiche von Wortpositionen.

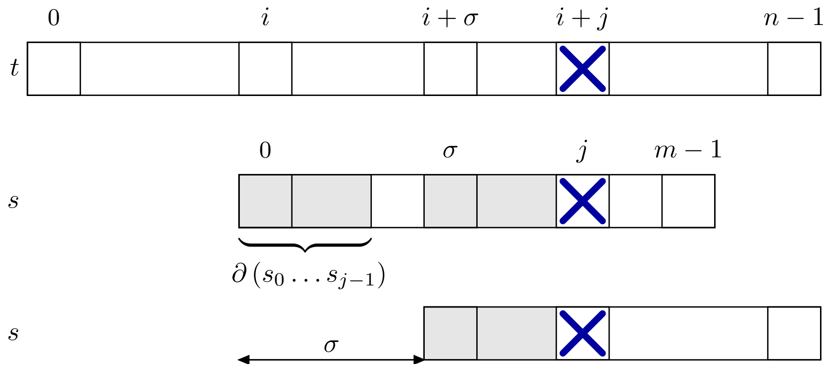
Proposition 2.2 *Der naive Algorithmus benötigt sowohl zum Finden eines Vorkommens als auch zum Finden aller Vorkommen eines Wortes der Länge m in einem Wort der Länge n im schlechtesten Fall $O(n \cdot m)$ Vergleiche.*

Im Folgende wird unser Ziel sein, Algorithmen für das Suchproblem zu entwickeln, die mit $O(n + m)$ Vergleichen auskommen, die also mithin in Linearzeit laufen.

2.1.3 Der Algorithmus von Knuth, Morris und Pratt

Ein klassischer Linearzeitalgorithmus ist der Algorithmus von Knuth, Morris und Pratt. Dieser Algorithmus basiert auf dem im vorangegangenen Abschnitt dargelegten naiven Algorithmus, benutzt aber eine verfeinerte Berechnung der Verschiebung des Suchwortes.

Dazu führen wir eine Analyse der Unverträglichkeit von Positionen durch. Wir betrachten dazu folgendes Schema.



Ein Paar (i, j) heißt *Unverträglichkeit* (im Englischen *mismatch*) von s und t, falls der Vergleich $s_j = t_{i+j}$ in der zweiten Zeile des naiven Algorithmus in Abbildung 2.1 negativ ausfällt. Damit haben wir folgende Situation:

$$s_0 \dots s_{j-1} = t_i \dots t_{i+j-1} \text{ und } s_j \neq t_{i+j}.$$

Eine Erhöhung von i heißt *Verschiebung* (im Englischen *shift*). Eine Verschiebung von i auf $i + \sigma$ heißt zulässig, falls gilt:

$$t_{i+\sigma} \dots t_{i+j-1} = s_0 \dots s_{j-1-\sigma}$$

Damit müssen wir bei einer Verschiebung erst ab Position $j - \sigma$ Vergleiche durchführen.

Proposition 2.3 *Es sei eine Unverträglichkeit (i, j) aufgetreten. Dann ist die kürzeste zulässige Verschiebung die von i auf $i + j - |\partial(s_0 \dots s_{j-1})|$.*

Beweis: Die Verschiebung von i auf $i + j$ ist immer zulässig. Ist i auf $i + \sigma$ eine zulässige Verschiebung bei Unverträglichkeit (i, j), so gilt:

$$s_0 \dots s_{j-1} = t_i \dots t_{i+j-1} \text{ und } s_0 \dots s_{j-1-\sigma} = t_{i+\sigma} \dots t_{i+j-1}$$

Damit ist das Wort $s_0 \dots s_{j-1-\sigma}$ ein Rand von $s_0 \dots s_{j-1}$. Daraus folgt, dass $\partial(s_0 \dots s_{j-1})$ die kürzeste zulässige Verschiebung erzeugt, nämlich um $j - |\partial(s_0 \dots s_{j-1})|$. ■

Mit dieser Einsicht können wir wie folgt fortfahren. Angenommen wir bekommen als zusätzliche Eingabe ein Feld B mit allen Ränderlängen, d.h. $B[0] = -1$ und für $1 \leq j \leq m$ gilt $B[j] = |\partial(s_0 \dots s_{j-1})|$. Dann können wir den Algorithmus von Knuth, Morris und Pratt wie in Abbildung 2.2 beschrieben formulieren.

Algorithmus: Knuth-Morris-Pratt
Eingabe: Wörter s, t über Σ mit $|s| = m$ und $|t| = n$
Gesucht: Alle Positionen, ab denen s als Teilwort in t vorkommt

1. WHILE $i \leq n - m$
2.   WHILE $j \leq m$ AND $s_j = t_{i+j}$
3. $j := j + 1$
4.   IF $j = m$
5.   Gebe i zurück
5. $i := i + j - B[j]$
7. $j := \max\{0, B[j]\}$

Abbildung 2.2: Algorithmus von Knuth, Morris und Pratt (mit gegebenen Ränderlängen im Feld B)

Für die Anzahl der Vergleiche bei gegebenen Ränderlängen erhalten wir folgende Aussage.

Lemma 2.4 *Der Algorithmus von Knuth, Morris und Pratt benötigt maximal $2n - m + 1$ Vergleiche, um ein Wort der Länge m in einem Wort der Länge n zu suchen, wenn die Ränderlängen für das gesuchte Wort gegeben sind.*

Beweis: Wir unterscheiden in der Analyse zwischen erfolglosen und erfolgreichen Versuchen beim Vergleich $s_j = t_{i+j}$ in der zweiten Zeile des Algorithmus.

1. Für die Anzahl der *erfolglosen* Vergleiche ergibt sich $n - m + 1$ als einfache obere Schranke.

2. Die Anzahl *erfolgreicher* Versuche lässt sich wie folgt abschätzen: Nach jedem Durchlauf der äußeren WHILE-Schleife (erste bis siebte Zeile) wird $i+j$ nicht kleiner, damit wird $i + j$ wegen $j - B[j] > 0$ und $B[j] \geq 0$ für $j > 0$ auf jeden Fall erhöht. Wegen $0 \leq i + j \leq (n - m) + m \leq n$ werden maximal n erfolgreiche Vergleiche ausgeführt.

Somit ergeben sich insgesamt höchstens $n - m + 1 + n = 2n - m + 1$ Vergleiche. ∎

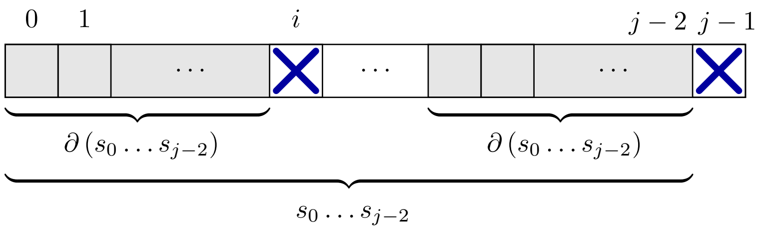
Wir müssen uns nun noch damit beschäftigen, wie wir das Feld B möglichst effizient berechnen können. Dazu führen wir weitere Vorüberlegungen durch. Angenommen wir hätten die Feldelemente $B[0], \ldots, B[j-1]$ bereits berechnet und wollen nun das Element

Algorithmus: ComputeBoundaries
Eingabe: Wort s mit $|s| = m$
Gesucht: Feld B mit allen Ränderlängen von s

$B[0] := -1$
$B[1] := 0$
FOR $j := 2$ TO m
  WHILE $k \geq 0$ AND $s_k \neq s_{j-1}$
    $k := B[k]$
  $B[j] := k + 1$
  $k := k + 1$

Abbildung 2.3: Algorithmus zur Berechnung der Ränderlängen

$B[j] = |\partial(s_0 \ldots s_{j-1})|$ berechnen. Das Szenario ist in folgender Abbildung zusammengefasst:



Wir unterscheiden hier zwei Fälle.

1. Ist $s_{B[j-1]} = s_{j-1}$, so gilt klarerweise $B[j] = B[j-1] + 1$.

2. Ist $s_{B[j-1]} \neq s_{j-1}$, so verfahren wir folgt: Wir müssen in diesem Fall ein kürzeres Präfix von $s_0 \ldots s_{j-2}$ finden, das auch Suffix von $s_0 \ldots s_{j-2}$ ist. Das nächstkürzere Präfix mit dieser Eigenschaft ist $\partial(\partial(s_0 \ldots s_{j-2}))$. Wir testen nun, ob sich dieser Rand zu einem Rand von $s_0 \ldots s_{j-1}$ erweitern lässt. Sollte dies nicht der Fall, so betrachten wir $\partial(\partial(\partial(s_0 \ldots s_{j-2})))$ usw. usf.

Diese Überlegungen lassen sich nun in die Form des Algorithmus aus Abbildung 2.3 bringen.

Für die Anzahl der Vergleiche zur Bestimmung der Ränderlängen erhalten wir folgende Aussage.

Lemma 2.5 *Das Feld B mit den Ränderlängen von s lässt sich mit höchstens $2m - 1$ Vergleichen berechnen.*

Beweis: Auch hier unterscheiden wir wieder zwischen erfolglosen und erfolgreichen Versuchen bei den ausgeführten Vergleichen $s_k = s_{j-1}$ in der vierten Zeile.

1. Die Anzahl *erfolgreicher* Vergleiche lässt sich folgendermaßen abschätzen: Bei einem erfolgreichen Vergleich ($s_k = s_{j-1}$) wird k in der siebten Zeile erhöht; anschließend aber auch j in der dritten Zeile. Wegen $j \in \{2, \ldots, m+1\}$ kann k maximal $(m-1)$-mal erhöht werden. Somit sind maximal $m-1$ erfolgreiche Vergleiche möglich.

2. Bei der Anzahl *erfolgloser* Vergleiche ergibt sich folgendes: Bei einem erfolglosen Vergleich ($s_k \neq s_{j-1}$) wird k um mindestens 1 verringert. Es gilt jedoch in jedem Fall $k \geq -1$. Da außerdem zu Beginn $k = 0$ galt und k nur um so viel verringert werden kann, wie vorher zugefügt worden ist, kann k höchstens $((m-1)+1)$-mal verringert werden. Es sind also maximal m erfolglose Vergleiche möglich.

Insgesamt ergeben sich also höchstens $(m-1) + m = 2m - 1$ Vergleiche. ■

Setzen wir die beiden Algorithmen aus den Abbildungen 2.2 und 2.3 zusammen, so erhalten wir den vollständigen Algorithmus von Knuth, Morris und Pratt. Die Laufzeit dieses Algorithmus lässt sich nun leicht aus den bewiesenen Lemmata gewinnen.

Theorem 2.6 *Der Algorithmus von Knuth, Morris und Pratt benötigt im schlechtesten Fall höchstens $2n+m$ Vergleiche, um ein Wort der Länge m in einem Wort der Länge n zu suchen (und zu finden).*

Beweis: Folgt durch Addition unmittelbar aus Lemma 2.4 und Lemma 2.5. ■

Bemerkungen:

1. Der Algorithmus von Knuth, Morris und Pratt kann so abgeändert werden werden, dass alle Positionen, ab denen ein Wort der Länge m in einem Wort der Länge n vorkommt, mit $O(n+m)$ Vergleichen gefunden werden können.

2. Der Algorithmus von Knuth, Morris und Pratt kann erweitert werden, um eine Menge von Wörtern einem Text zu suchen (Algorithmus von Aho und Corasick).

2.1.4 Der Algorithmus von Boyer und Moore

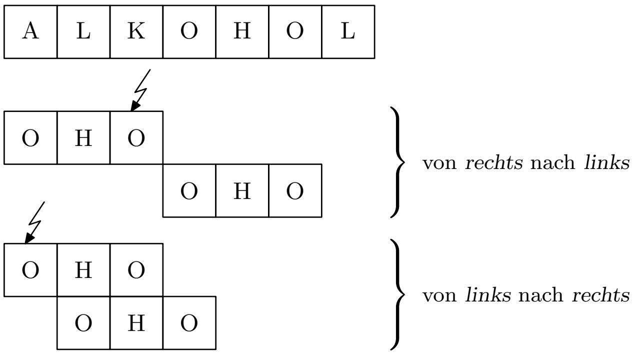
Wir beschäftigen uns mit einem weiteren Algorithmus für das Problem der exakten Suche in Texten, der mit einer linearen Anzahl von Vergleichen auskommt. Im Gegensatz zu dem Algorithmus von Knuth, Morris und Pratt, bei dem der Test auf das Vorkommen des Suchwortes im Text immer von links nach rechts durchgeführt wird, besteht die Idee beim Algorithmus von Boyer und Moore gerade darin, den Test von rechts nach links durchzuführen. Ein einfacher Algorithmus, der dies realisiert ist in Abbildung 2.4 dargestellt.

Algorithmus: Right-to-Left-Naive
Eingabe: Wörter s und t mit $|s| = m$ und $|t| = n$
Gesucht: Position, ab der s in t vorkommt

```
WHILE i ≤ n − m
    j := m − 1
    WHILE s_j = t_{i+j}
        j := j − 1
        IF j < 0
            Gebe i zurück
    i := i + 1
```

Abbildung 2.4: Einfacher Algorithmus mit Rechts-Links-Test

Warum ist dieser Ansatz überhaupt interessant? Für große Alphabete (z.B. bei der Verarbeitung von Web-Dokumenten) ist die Wahrscheinlichkeit groß, dass eine Unverträglichkeit (i, j) durch ein Symbol ausgelöst wird, das im Suchwort s nicht vorkommt, wie infolgendem Beispiel zu sehen ist.



In diesem Fall können wir bei der Verschiebung i gleich auf $i + j$ setzen.

Natürlich benötigt der Algorithmus aus Abbildung 2.4 im schlechtesten Fall $O(n \cdot m)$ Vergleiche für die Suche. Wie beim Algorithmus von Knuth, Morris und Pratt werden wir jedoch auch hier eine Verschiebungstablle S verwenden, von der wir uns noch überzeugen müssen, dass wir sie mit linearer Anzahl von Vergleich bestimmen können, um insgesamt einen Linearzeit-Algorithmus zu erhalten. Ohne an dieser Stelle bereits näher auf die Vorberechnung der Verschiebungstablle S einzugehen, lässt sich der Algorithmus von Boyer und Moore wie in Abbildung 2.5 gezeigt formulieren.

Zur Berechnung der Verschiebungstabelle führen wir wieder eine Unverträglichkeitsanalyse durch. Dazu vergleichen wir $s_0 \ldots s_{m-1}$ mit $t_i \ldots t_{i+m-1}$. Es sei eine Unverträglichkeit bei (i, j) aufgetreten, d.h. es gilt

$$s_{j+1} \ldots s_{m-1} = t_{i+j+1} \ldots t_{i+m-1} \text{ und } s_j \neq t_{i+j}.$$

Algorithmus: Boyer-Moore
Eingabe: Wörter s und t mit $|s| = m$ und $|t| = n$
Gesucht: Position, ab der s in t vorkommt

```
Berechne Verschiebungstablle S für das Wort s
j := m − 1
WHILE i ≤ n − m
    WHILE s_j = t_{i+j}
        j := j − 1
        IF j < 0
            Gebe i zurück
    i := i + S[j]
    j := m − 1
```

Abbildung 2.5: Algorithmus von Boyer und Moore

Hier ist zu beachten, dass die Unverträglichkeit von rechts kommend auftritt. Für eine zulässige Verschiebung σ müssen wir zwei Fälle unterscheiden.

1. Im Fall $\sigma \leq j$ muss

   $$s_{j+1-\sigma} \dots s_{m-1-\sigma} = t_{i+j+1} \dots t_{i+m-1} = s_{j+1} \dots s_{m-1} \text{ sowie } s_j \neq s_{j-\sigma}$$

   gelten.

2. Für $\sigma > j$ muss $s_0 \dots s_{m-1-\sigma} = s_\sigma \dots s_{m-1}$ erfüllt sein.

Bildlich lassen sich die beiden Fälle wir folgt ausdrücken:

Bild zur Verschiebungskonstellation bei Unverträglichkeit

Eine einfache Beobachtung ist nun, dass die schraffierten Bereiche gerade Ränder sind. Wir definieren also für ein Wort s die Randmenge $R(s)$ als

$$R(s) =_{\text{def}} \left\{ s' \mid s' \text{ ist Rand von } s \right\}.$$

Damit sind die Bedingungen für eine zulässige Verschiebung σ wie folgt formulierbar:

$$\sigma \leq j \ \wedge \ s_{j+1} \dots s_{m-1} \in R(s_{j+1-\sigma} \dots s_{m-1}) \ \wedge s_j \neq s_{j-\sigma} \tag{2.1}$$
$$\sigma > j \ \wedge \ s_0 \dots s_{m-1-\sigma} \in R(s_0 \dots s_{m-1}) \tag{2.2}$$

Wir definieren die Tabelle (Array) S mit den kürzesten zulässigen Verschiebungen für $0 \leq j \leq m$ als

$$S[j] =_{\text{def}} \min \{ \sigma \mid (\sigma, j) \text{ erfüllt Bedingung (2.1) oder Bedingung (2.2) } \} .$$

Diese Regel zur Berechnung von S heißt *good suffix rule*. Die Berechnung von S gemäß dieser Regel erfolgt wie in Abbildung 2.6 beschrieben und in folgende Abbildung illustriert:

Bild zur Berechnung gemäß *good suffix rule*

Wir geben das Resultat der Laufzeitanalyse ohne Beweis an.

Theorem 2.7 *Der Algorithmus von Boyer und Moore benötigt im schlechtesten Fall maximal* $3n - \frac{3(n-m+1)}{m+2}$ *Vergleiche, um ein Wort der Länge* m *in einem Wort der Länge* n *zu suchen.*

Bemerkung. Der Algorithmus von Boyer und Moore benötigt im schlechtesten Fall $3n - o(n)$ Vergleiche für $n/m \to \infty$ und $m \to \infty$. Damit ist er asmyptotisch schlechter als der Algorithmus von Knuth, Morris und Pratt.

Wir wollen uns noch überlegen, wie wir die Motivation für den Rechts-Links-Vergleich gewinnbringend in den Algorithmus von Boyer und Moore integrieren können, ohne die Laufzeit entscheidend zu erhöhen. Dazu führen wir eine neue Regel ein, die *bad character rule* genannt wird. Wir betrachten eine Unverträglichkeit bei (i, j) mit dem Symbol $t_{i+j} = x$. Zwei Fälle sind möglich:

1. Es gibt $0 \leq r \leq j - 1$ mit $s_r = x$. Dann definieren wir $\sigma =_{\text{def}} j - r$.
2. Für alle $0 \leq r \leq j - 1$ ist $s_r \neq x$. Dann definieren wir $\sigma =_{\text{def}} j + 1$.

Algorithmus:	ComputeShifts
Eingabe:	Wort s mit $\|s\| = m$
Gesucht:	Tablle S der kürzesten zulässigen Verschiebungen

```
FOR i := 0 TO m
    S[i] := m

/* Verschiebungsberechnung gemäß Bedingung (2.1) */

H[0] := −1
H[1] := 0
FOR j := 2 TO m
    WHILE k ≥ 0 AND s_{m−k−1} ≠ s_{m−j}
    σ := j − k − 1
    S[m − k − 1] := min{S[m − k − 1], σ}
    k := H[k]
H[j] := k + 1
k := k + 1

/* Verschiebungsberechnung gemäß Bedingung (2.2) */

B := ComputeBoundaries(s)                          /* siehe Abbildung 2.3 */
j := 0
i := B[m]
WHILE i ≥ 0
    WHILE j < m − i
        S[j] := min{S[j], m − i}
        j := j + 1
    i := B[i]
    j := 0
```

Abbildung 2.6: Algorithmus zur Berechnung der Verschiebungstabelle

Es ist nun leicht einzusehen, dass wir *good suffix rule* und *bad character rule* einfach durch das Maximum der Verschiebungen verknüpfen können.

Bemerkungen.

1. Der Algorithmus von Boyer und Moore (in obiger Version) benötigt $O(n + r \cdot m)$ Vergleiche, um alle r Vorkommen von s in t zu finden (im Gegensatz zum Algorithmus von Knuth, Morris und Pratt), z.B. für $s = a^m$ und $t = a^n$.

2. Der eben angeführte Nachteil kann jedoch durch weitere einfache Veränderungen der Verschiebungstabelle S auf $O(n + m)$ verbessert werden (Erweiterung von Galil).

2.1.5 Approximative Suche in Texten

Bisher haben wir das Suchproblem stets so betrachtet, dass das gesuchte Wort s exakt als Teilwort in t vorkommen muss. Nunmehr lassen wir eine begrenzte Anzahl von Fehlern zu. Das gesuchte Wort s muss jetzt als Teilwort in t bis auf höchstens k Fehler vorkommen. Wir definieren das Suchproblem zunächst formal.

Es sei Σ ein endliches Alphabet. Es seien $s = s_1 \dots s_m$ und $t = t_1 \dots t_n$ Wörter über Σ. Es sei $k \in \mathbb{R}_+$ die maximal erlaubte Fehlerzahl. Es sei $d : \Sigma^* \times \Sigma^* \to \mathbb{R} \cup \{\infty\}$ eine Abstandsfunktion (wie wir sie später definieren werden).

Das Problem der approximativen Suche in Texten besteht nun im folgenden: Gegeben s, t, k und $d(\cdot, \cdot)$ finde alle Positionen j in t, so dass ein $i \leq j$ existiert mit $d(s, t_i \dots t_j) \leq k$. Hierbei ist zu beachten, dass wir die Endposition ausgeben und nicht die Anfangsposition, ab der das Suchwort vorkommt. Durch Betrachtung der umgekehrten Wörter sind jedoch auch diese Position bestimmbar.

Wir werden uns hier nur auf solche Abstandsfunktionen beschränken, die auf Transformationen von Wörtern beruhen.

Eine endliche Menge $R \subseteq \Sigma^* \times \Sigma^*$ heißt *Operationenmenge*, die Elemente von R heißen *Operationen*. Es seien $s, t \in \Sigma^*$ gegeben. Sei $(x, y) \in R$ eine Operation. Dann *konvertiert* (x, y) das Wort s in das Wort t (symbolisch $s \overset{(x,y)}{\Longrightarrow} t$), falls Wörter $v, w \in \Sigma^*$ existieren mit $s = vxw$ und $t = vyw$. Eine *Transformation* von s in t ist eine Folge von Operationen $(o_1, \dots, o_\ell)$ mit $o_i \in R$, so dass Wörter $w_1, \dots, w_{\ell-1}$ existieren mit $s \overset{o_1}{\Longrightarrow} w_1, w_1 \overset{o_2}{\Longrightarrow} w_2, \dots, w_{\ell-1} \overset{o_\ell}{\Longrightarrow} t$.

Es sei R eine Operationenmenge und $c : R \to \mathbb{R}_+$ eine Kostenfunktion. Die Kosten einer Transformation $(o_1, \dots, o_\ell)$ sind definiert als

$$c(o_1, \dots, o_\ell) =_{\text{def}} \sum_{j=1}^{\ell} c(o_j).$$

Wir definieren eine *Abstandsfunktion* $d : \Sigma^* \times \Sigma^* \to \mathbb{R}_+$ für $s, t \in \Sigma^*$ wie folgt:

$$d(s,t) =_{\text{def}} \begin{cases} \text{minimale Kosten einer Transformation von } s \text{ in } t, & \text{falls eine existiert} \\ \infty, & \text{sonst} \end{cases}$$

Bemerkungen

1. Abstandsfunktion ist im Allgemeinen unberechenbar (wegen Unentscheidbarkeit von generellen Ersetzungssystemen).

2. Ein Ausweg aus der Unberechenbarkeit: Wir betrachten nur Transformationen, bei denen ein einmal konvertiertes Teilwort nicht noch einmal verändert werden darf. Dies wird bei uns immer der Fall sein.

3. Ist R symmetrisch und gilt $c(x,y) = c(y,x)$ für alle $(x,y) \in R$, so ist d symmetrisch, d.h. es gilt $d(s,t) = d(t,s)$ für alle $s,t \in \Sigma^*$. Außerdem gilt $d(s,t) \geq 0$ und $d(s,t) \leq d(s,w) + d(w,t)$ für alle $s,t,w \in \Sigma^*$. Somit ist eine symmetrische Abstandsfunktion eine Metrik.

Wenn nicht anders vermerkt, sind bei uns die Kosten für alle zugelassenen Operationen gleich 1.

Typische Operationen

- Einfügen (INS): Formal definiert als (ε, a) für $a \in \Sigma$; interpretiert als „Füge Buchstaben a an beliebiger Position ein“

- Loschen (DEL): Formal definiert als (a, ε) für $a \in \Sigma$; interpretiert als „Entferne ein beliebiges Vorkommen des Buchstaben a“

- Substitution (SUB): Formal definiert als (a,b) für $a,b \in \Sigma$; interpretiert als „Ersetze ein Vorkommen des Buchstabens a durch Buchstaben b“

- Transposition (TRANS): Formal definiert als (ab, ba) für $a,b \in \Sigma, a \neq b$; interpretiert als „Vertausche einmalig benachbarte Buchstaben a und b“

Häufig benutzte Abstände

1. Der *Editierabstand* (Levenshtein-Abstand) verwendet als Operationenmenge die Menge {INS, DEL, SUB} und definiert eine symmetrische Abstandsfunktion $\mathrm{ed}(\cdot,\cdot)$. Es gilt $0 \leq \mathrm{ed}(x,y) \leq \max\{|x|,|y|\}$.

2. Die *Hamming-Distanz* verwendet als Operationenmenge die Einermenge {SUB} und definiert eine symmetrische Abstandsfunktion $d_H(\cdot,\cdot)$. Es gilt $0 \leq d_H(x,y) \leq |x|$ für $|x| = |y|$ und $d(x,y) = \infty$ sonst.

3. Der *Episodenabstand* verwendet als Operationenmenge die Einermenge {INS}. Die Abstandsfunktion $d_E(\cdot,\cdot)$ ist nicht symmetrisch und hat für alle x, y die Eigenschaft $d_E(x,y) \in \{|y| - |x|, \infty\}$.

Beispiel. Es seien die Wörter $s =$ INTERNET und $t =$ ZISTERNE gegeben. Dann gilt für die Hamming-Distanz $d_H(s,t) = \|\{i \mid s_i \neq t_i\}\| = 8$. Für den Editierabstand ergibt sich $\mathrm{ed}(s,t) = 3$, wie sich leicht durch die Transformation

$$\begin{array}{rcl} \text{INTERNET} & \overset{(\varepsilon,Z)}{\Longrightarrow} & \textbf{Z}\text{INTERNET} \\ \text{ZI}\textbf{N}\text{TERNET} & \overset{(N,S)}{\Longrightarrow} & \text{ZI}\textbf{S}\text{TERNET} \\ \text{ZISTERNE}\textbf{T} & \overset{(T,\varepsilon)}{\Longrightarrow} & \text{ZISTERNE} \end{array}$$

Im Folgenden interessieren wir uns nur noch für den Editierabstand.

Die erste Frage, mit der wir uns beschäftigen müssen, ist: Wie berechnen wir überhaupt den Editierabstand?

Dazu betrachten wir einen (generischen) Ansatz mittels Dynamischer Programmierung. Es seien $s, t \in \Sigma^*$ gegeben. Wir definieren eine $(1 + |s|) \times (1 + |t|)$-Matrix D_{st} wie folgt:

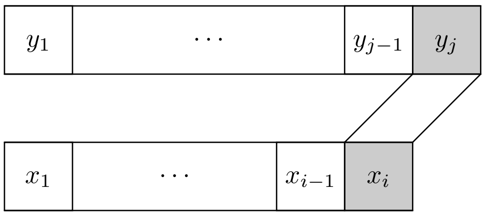
$$D_{st}[i,j] =_{\mathrm{def}} \mathrm{ed}(s_1 \dots s_i, t_1 \dots t_j)$$

Die Matrizeneinträgen stehen untereinander wie folgt in Zusammenhang.

Proposition 2.8 *Es seien s und t Wörter über Σ. Dann gelten für $i \in \{0, 1, \dots, |s|\}$ und $j \in \{0, 1, \dots, |t|\}$ folgende Aussagen.*

1. $D_{st}[0,j] = j$.
2. $D_{st}[i,0] = i$.
3. *Ist $s_i = t_j$, so gilt $D_{st}[i,j] = D_{st}[i-1,j-1]$.*
4. *Ist $s_i \neq t_j$, so gilt $D_{st}[i,j] = 1 + \min\{D_{st}[i-1,j], D_{st}[i,j-1], D_{st}[i-1,j-1]\}$.*

Beweis: Die ersten drei Aussagen sind klar. Für die vierte Aussage gelte $s_i \neq t_j$. Wir unterscheiden drei Fälle, je nachdem welche Operation die Transformation von $s_1 \dots s_i$ in $t_1 \dots t_j$ vollendet. Zur Veranschaulichung betrachten wir folgendes Schema.



Algorithmus: ComputeEditDistance
Eingabe: Wörter s und t mit $|s| = m$ und $|t| = n$
Gesucht: Editierabstand $\mathrm{ed}(s,t)$

```
FOR i := 0 TO m
    D[i,0] := i
FOR j := 0 TO n
    D[0,j] := j
FOR j := 1 TO n
    FOR i := 1 TO m
        IF s_i = t_j
            D[i,j] := D[i-1,j-1].
        ELSE
            D[i,j] := 1 + min {D[i-1,j], D[i,j-1], D[i-1,j-1]}
Gebe D[m,n] zurück
```

Abbildung 2.7: Dynamische Programmierung für Berechnung des Editierabstandes

Wir führen den Beweis durch Weglassen bzw. Hinzufügen der Buchstaben s_i oder t_j.

1. Löschen von s_i schließt die Transformation von $s_1 \dots s_i$ in $t_1 \dots t_j$ ab. Dann gilt $D_{st}[i,j] \leq 1 + D_{st}[i-1,j]$.

2. Einfügen von t_j am Ende von $s_1 \dots s_i$ schließt die Transformation von $s_1 \dots s_i$ in $t_1 \dots t_j$ ab. Dann gilt $D_{st}[i,j] \leq 1 + D_{st}[i,j-1]$.

3. Substitution von s_i durch t_j schließt die Transformation von $s_1 \dots s_i$ in $t_1 \dots t_j$ ab. Dann gilt $D_{st}[i,j] \leq 1 + D_{st}[i-1,j-1]$.

Insgesamt gilt also $D_{st}[i,j] \leq 1 + \min\{D_{st}[i-1,j], D_{st}[i,j-1], D_{st}[i-1,j-1]\}$. Da die obigen Operationen die einzigen möglichen Operatione für die Vollendung der Transformation sind, erhalten wir Gleichheit in der Ungleichung. ■

Damit ergibt sich auf kanonische Weise der in Abbildung 2.7 beschriebene Algorithmus auf der Basis einer Dynamischen Programmierung für die Berechnung des Editierabstandes.

Beispiel. Wir berechnen die Abstandsmatrix D_{st} für unser Beispiel mit den Wörtern s = INTERNET und t = ZISTERNE.

	D_{st}		Z	I	S	T	E	R	N	E
		0	1	2	3	4	5	6	7	8
	0	0	1	2	3	4	5	6	7	8
I	1	1	1	1	2	3	4	5	6	7
N	2	2	2	2	2	3	4	5	5	6
T	3	3	3	3	3	2	3	4	5	6
E	4	4	4	4	4	3	2	3	4	5
R	5	5	5	5	5	4	3	2	3	4
N	6	6	6	6	6	5	4	3	2	3
E	7	7	7	7	7	6	5	4	3	2
T	8	8	8	8	8	7	6	5	4	**3**

Theorem 2.9 *Der Algorithmus zur Berechnung des Editierabstandes zwischen Wörtern der Längen m und n mittels Dynamischer Programmierung benötigt sowohl im schlechtesten Fall als auch im besten Fall $\Theta(n \cdot m)$ Schritte. Er kann mit dem Speicherplatz $O(\min\{n, m\})$ implementiert werden.*

Beweis: Die Laufzeitabschätzung ist offensichtlich. Für die Implementierung mit linearem Platz genügt die Beobachtung, dass für eine zeilenweise Berechnung der Matrix jeweils nur die darüber liegende Zeile bzw. für eine spaltenweise Berechnung der Matrix jeweils nur die links daneben liegende Spalte gespeichert werden muss. ■

Bemerkung. Es gibt einen Algorithmus, der den Editierabstand zwischen s und t in der Zeit $O(\mathrm{ed}(s, t)^2)$ (bzw. $O(k^2)$ bei vorgegebenem Fehler) bestimmt (Algorithmus von Ukkonen).

Wir müssen uns nun noch darum kümmern, wie wir den Algorithmus zur Berechnung des Editierabstandes adaptieren können, um ihn auch für die approximative Suche zu verwenden. Die Idee besteht hier einfach darin, für jede Position in t den Editierabstand zu s neu zu berechnen. Dafür können wir einfach die Matrix D_{st} in abgeänderter Weise initialisieren, nämlich als $D_{st}[0, j] = 0$ für alle $j \in \{0, 1, \ldots, n\}$. Dies führt unmittelbar zu dem in Abbildung 2.8 gezeigten Algorithmus. Zu beachten ist hierbei, dass die Matrix spaltenweise entwickelt wird. Im Array D ist deshalb die aktuelle Spalte enthalten, während Array A die linke Nachbarspalte abspeichert.

Theorem 2.10 *Der Algorithmus von Sellers benötig $O(n \cdot m)$ Schritte und $O(m)$ Platz, um alle Vorkommen eines Wortes der Länge m in einem Wort der Länge n mit bis zu k Editierfehlern zu bestimmen*

Beweis: Offensichtlich. ■

Algorithmus: SELLERS
Eingabe: Wörter s und t mit $|s| = m$ und $|t| = n$, Parameter $k \in \mathbb{N}$
Gesucht: alle Positionen j mit $\mathrm{ed}(s, t_i \dots t_j) \leq k$ für ein geeignetes i

```
FOR i := 1 TO m
    A[i] := i
FOR j := 1 TO n
    FOR i := 1 TO m
        IF s_i = t_j
            D[i] := A[i − 1].
        ELSE
            D[i] := 1 + min {D[i − 1], A[i], A[i − 1]}
    IF D[m] ≤ k
        Gebe j zurück
Kopiere D nach A
```

Abbildung 2.8: Algorithmus von Sellers für approximative Suche in Texten

Bemerkungen.

1. Der Algorithmus von Sellers kann für viele andere Abstandsfunktionen herangezogen werden.

2. Es gibt einen Algorithmus für Suche in Texten mit bis zu k Fehlern, der mit Zeit $O(k \cdot n)$ auskommt (Algorithmus von Landau und Vishkin).

2.2 Hypertexte

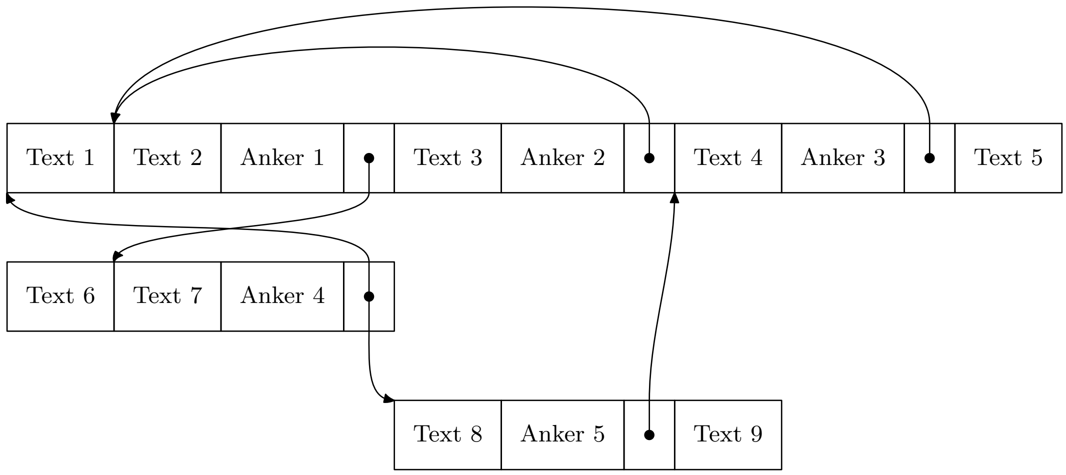
Hypertext ist ein „nichtlinearer" Text, der als Netzwerk von Informationseinheiten aufgebaut ist. Wesentliche Bestandteile von Hypertexten sind:

- *Knoten* stellen die Informationseinheiten dar und enthalten typischerweise Text.

- *(Hyper)Links* sind Querverweise zwischen Knoten. Es gibt verschiedene Varianten, z.B. uni- und bidirektional, ein oder mehrere Ziele (*fat links*), eine oder mehrere Quellen usw. usf.

- *Anker* sind die Kopplungsstellen zwischen Links und Knoten.

Eine Hypertext(-Dokument) ist eine Sammlung von Knoten und Links. Die folgende Abbildung zeigt die wesentlichen Ausdrucksmöglichkeit in einem Hypertext.



2.2.1 Das Modell von Manber und Wu

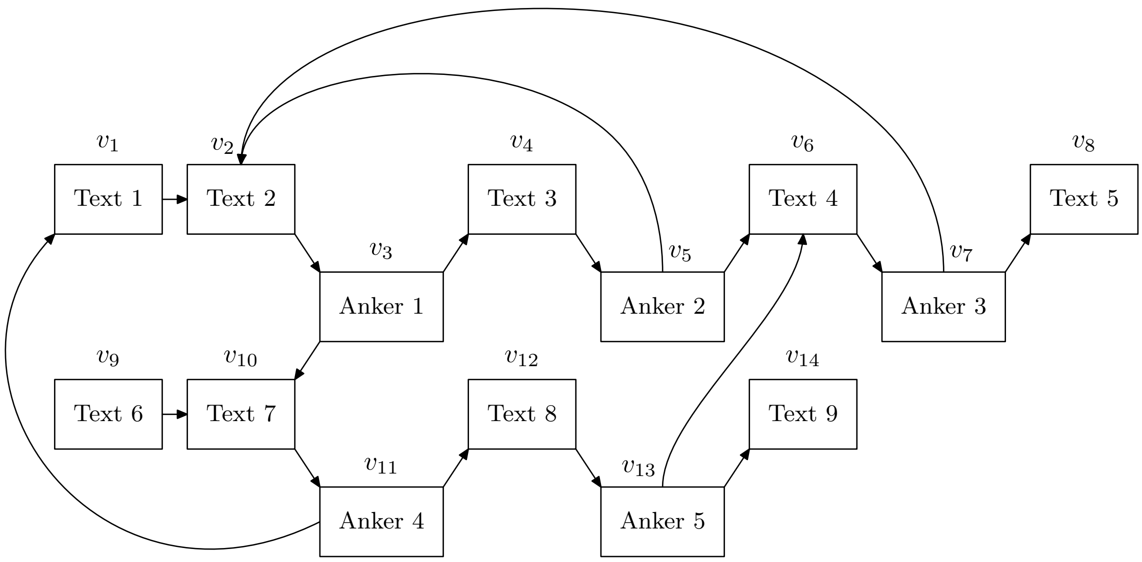
Im Folgenden betrachten wir ein einfaches graphentheoretisches Modell für Hypertext, das als Basis für die Behandlung von Suchproblemen in Hypertext dient.

Definition 2.11 *Es sei Σ ein endliches Alphabet.*

1. *Ein Tripel $H = (V, E, T)$ heißt* Hypertext*, falls (V, E) ein gerichteter Graph und T eine Abbildung von V nach Σ^+ sind.*

2. *Ein Hypertext $H = (V, E, T)$ heißt* elementar*, falls $|T(v)| = 1$ für alle $v \in V$ gilt.*

Unser Hypertextbeispiel lässt sich im Modell von Manber und Wu wie folgt ausdrücken:



Wir verwenden folgende Konventionen für einen Hypertext $H = (V, E, T)$:

- $n =_{\text{def}} \|V\|$
- $N =_{\text{def}} \sum_{v \in V} |T(v)|$ (Gesamttextlänge)

Klarerweise gilt $n = N$ für elementaren Hypertext.

Wir gehen nunmehr zu einer Formalisierung des Suchproblemes in Hypertext über.

Definition 2.12 *Es seien $H = (V, E, T)$ ein Hypertext (über dem Alphabet Σ) und s ein Wort aus Σ^*. Es sei $(v_1, \ldots, v_k)$ ein gerichteter Weg in (V, E) mit $k \geq 2$, d.h. es gilt $(v_i, v_{i+1}) \in E$ für alle $i \in \{1, \ldots, k-1\}$.*

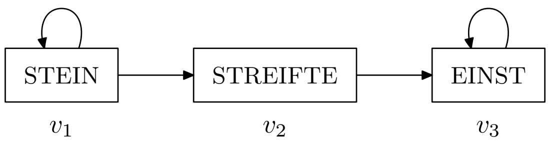
1. *Das Wort s ist* verträglich *mit dem Weg $(v_1, \ldots, v_k)$ in H (ab Position j in v_1), falls ein Suffix $R(v_1)$ von $T(v_1)$ und ein Präfix $L(v_k)$ von $T(v_k)$ existieren, so dass $s = R(v_1)T(v_2)\ldots T(v_{k-1})L(v_k)$ gilt.*

2. *Das Wort s ist* Teilwort *des Hypertextes H, falls eine der folgende Bedingungen erfüllt ist:*

   (a) Es gibt einen Knoten $v \in V$, so dass s Teilwort von $T(v)$ ist.

   (b) Es gibt einen Weg $(v_1, \ldots, v_k)$ in H mit $k \geq 2$, so dass s verträglich mit $(v_1, \ldots, v_k)$ ist.

Beispiel. Wir betrachten folgenden Hypertext



zusammen mit dem Wort s = EINSTEINSTREIFTEEINSTEIN. Dann ist mit nachfolgendem Schema leicht einzusehen, dass s verträglich ist mit dem Weg $(v_1, v_1, v_2, v_3, v_3)$:

$R(v_1)$	$T(v_1)$	$T(v_2)$	$T(v_3)$	$L(v_3)$
EIN	STEIN	STREIFTE	EINST	EIN

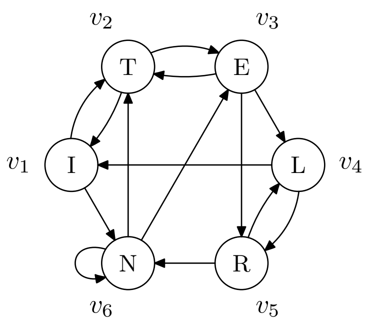
2.2.2 Exakte Suche in Hypertext

Das Problem der exakten Suche im Hypertext lässt sich wie folgt formulieren: Gegeben sei ein Hypertext $H = (V, E, T)$ und ein Suchwort $s \in \Sigma^*$. Kommt s als Teilwort in H vor? (Eigentlich lautet auch hier die Aufgabe: Finde alle Positionen, ab denen s verträglich mit passendem Weg in H ist.)

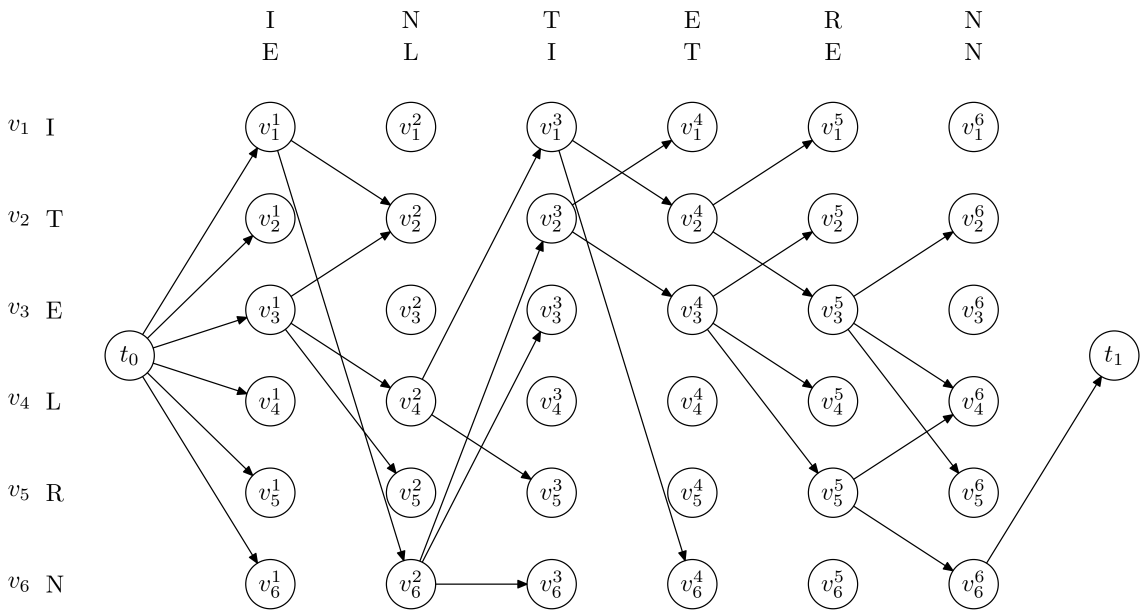
Wir betrachten zunächst das Problem auf elementarem Hypertext. Dafür seien ein elementarer Hypertext $H = (V, E, T)$ und ein Suchwort $s = s_1 \ldots s_m \in \Sigma^*$ gegeben. Es gilt mithin $N = \|V\| = n$. Die algorithmische Idee besteht darin, zu H einen Hilfsgraph (V_s^H, E_s^H) zu konstruieren, um das Suchproblem auf ein Erreichbarkeitsproblem im Hilfsgraph zurückzuführen. Dafür definieren wir:

$$
\begin{aligned}
V_s^H \; &=_{\text{def}} \; \left\{ v^i \mid v \in V, 1 \le i \le m \right\} \; \cup \; \{t_0, t_1\} \\
E_s^H \; &=_{\text{def}} \; \left\{ (t_0, v^1) \mid v \in V \right\} \; \cup \\
&\quad\; \cup \; \left\{ (v^i, u^{i+1}) \mid v, u \in E, T(v) = s_i \right\} \; \cup \\
&\quad\; \cup \; \{ (v^m, t_1) \mid T(v) = s_m \}
\end{aligned}
$$

Beispiel. Wir wollen die Wörter INTERNET und ELITEN in folgendem Hypertext suchen:



Unsere Konstruktion führt auf den folgenden Graphen, wobei beide Hilfsgraphkonstruktionen zusammen in einem Graphen dargestellt sind:



Lemma 2.13 *Es sei H ein elementarer Hypertext. Ein Wort s ist genau dann verträglich mit dem Weg $(v_1, \ldots, v_m)$ in H, wenn $(t_0, v_1^1, v_2^2, \ldots, v_m^m, t_1)$ ein Pfad in (V_s^H, E_s^H) ist.*

Beweis: Nach Konstruktion von (V_s^H, E_s^H). ■

Der Algorithmus in Abbildung 2.9 löst auf Basis von Lemma 2.13 das Suchproblem mit Hilfe einer Tiefensuche auf dem Hilfsgraph.

Theorem 2.14 *Der Algorithmus zur Suche eines Wortes der Länge m in einem elementaren Hypertext $H = (V, E, T)$ benötigt im schlechtesten Fall $O(mN + m\|E\|)$ Schritte.*

Beweis: Die Konstruktion von (V_s^H, E_s^H) ist in einer zur Größe von (V_s^H, E_s^H) proportionalen Anzahl von Schritten möglich, d.h. in $O(mN + m\|E\|)$ Schritten. Tiefensuche und Markieren benötigen $O(\|V_s^H\| + \|E_s^H\|) = O(mN + m\|E\|)$ Schritte. Die dritte Zeile des Algorithmus in Abbildung 2.9 benötigt $O(N)$ Schritte. ■

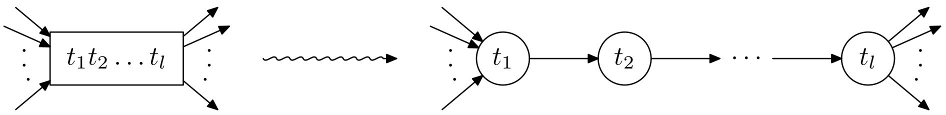
Bemerkung. Der Algorithmus zur exakten Suche in elementaren Hypertext kann so implementiert werden, dass lediglich $O(n)$ zusätzlicher Speicher benötigt wird.

Algorithmus:	SEARCHELEMENTARYHYPERTEXT
Eingabe:	elementarer Hypertext $H = (V, E, T)$ und Suchwort $s \in \Sigma^*$
Gesucht:	alle Knoten $v \in V$, ab denen s auf einem Weg in H vorkommt

1. Konstruiere (V_s^H, E_s^H) aus H und s
2. Führe Tiefensuche von t_0 aus in (V_s^H, E_s^H) durch; markiere dabei alle Knoten in V_s^H, von denen aus Knoten t_1 erreicht wird
3. FOR $v^1 \in V_s^H$
4.   IF v^1 ist markiert
5.   Gebe v zurück

Abbildung 2.9: Algorithmus für exakte Suche in elementarem Hypertext

Wir gehen nunmehr zur Betrachtung des Suchproblems auf allgemeinem Hypertext über. Der nahe liegende Ansatz ist hier, den Hypertext in elementaren Hypertext umzuwandeln vermöge der folgenden Ersetzung von Knoten v mit $|T(v)| > 1$ durch Pfade:



Durch die Transformation ergeben sich folgende Veränderungen in den Größen. Es seien $H = (V, E, T)$ der ursprüngliche Hypertext und $H' = (V', E', T')$ der korrespondierende elementare Hypertext. Dann gilt:

$$\begin{aligned}
\|V'\| &= \|V\| + \sum_{v \in V} (|T(v)| - 1) = N \\
\|E'\| &= \sum_{v \in V} (\mathrm{out} - \deg(v) + |T(v)| - 1) = \|E\| + N - \|V\| \\
N' &= \|V'\| = N
\end{aligned}$$

Wir modifizieren den Algorithmus SEARCHELEMENTARYHYPERTEXT aus Abbildung 2.9 so, dass für einen gegebenen Hypertext zunächst der elementare Hypertext konstruiert wird.

Korollar 2.15 *Der modifizierte Algorithmus zur Suche eines Wortes der Länge m in einem Hypertext $H = (V, E, T)$ mit der Gesamtlänge N benötigt im schlechtesten Fall $O(mN + m\|E\|)$ Schritte.*

2.2.3 Approximative Suche in elementarem Hypertext

Wir beschränken uns auf elementaren Hypertext. Damit lässt sich das Problem der approximativen Suche in Hypertext (bezüglich eine Abstandsfunktion $d(\cdot, \cdot)$) wie folgt formulieren. Gegeben seien ein elementarer Hypertext $H = (V, E, T)$, ein Suchwort $s \in \Sigma^*$ und eine

Algorithmus:	APPROXIMATIVESEARCH - VERSION H
Eingabe:	elementarer Hypertext $H = (V, E, T)$, Suchwort $s \in \Sigma^*$ und Parameter $k \in \mathbb{N}$
Frage:	Kann Hypertext H mit höchstens k Substitutionen so abgeändert werden, dass s verträglich mit einem Weg in H wird?

1. Rate für jeden Knoten in H eine konkrete Substitution (a, b) mit $a, b \in \Sigma$
2. $C :=$ Anzahl geratener Substitutionen (a, b) mit $a \neq b$
3. IF $C > k$ THEN Antworte mit „Nein“
4. ELSE
5. $\quad H'$:=Hypertext nach Ausführung der geratenen Substitutionen auf H
4. $\quad$ SEARCHELEMENTARYHYPERTEXT(H', s)

Abbildung 2.10: Nicht-deterministischer Algorithmus für Version H des approximativen Suchproblems

Fehleranzahl $k \in \mathbb{N}$. Gibt es einen Weg $(v_1, \ldots, v_l)$ in H mit $d(s, T(v_1)T(v_2)\ldots T(v_l)) \leq k$ gilt? (Eigentlich: Finde alle Knoten, ab denen s bis auf den Abstand k verträglich mit einem passenden Weg ist.)

Im Hypertext-Kontext müssen wir jedoch die folgende drei Versionen unterscheiden, je nachdem wo die Operationen durchgeführt dürfen:

- **Version H**: Können wird durch höchstens k Operationen auf dem Hypertext das Wort s verträglich mit einem Weg in H machen?
- **Version S**: Können wir durch höchstens k Operationen auf dem Suchwort ein zu einem Weg in H verträgliches Suchwort erhalten?
- **Version H+S**: Können wir durch höchstens k Operationen auf dem Hypertext und dem Suchwort ein zu einem Weg in H verträgliches Suchwort erhalten?

Theorem 2.16 *Es seien $\mathcal{A} = \{\mathrm{SUB}\}$ die die Hamming-Distanz definierende Operationenmenge und $\mathcal{B} = \{\mathrm{SUB}, \mathrm{INS}, \mathrm{DEL}\}$ die den Editierabstand definierende Operationenmenge.*

1. *Für $\mathcal{A}$ und $\mathcal{B}$ ist die Version H des approximativen Suchproblems in Hypertext* **NP***-vollständig.*

2. *Für $\mathcal{A}$ und $\mathcal{B}$ ist die Version H+S des approximativen Suchproblems in Hypertext* **NP***-vollständig.*

Beweis: Wir führen den Beweis lediglich für die erste Aussage und für die Operationenmenge $\mathcal{A}$ durch. Dazu müssen wir zeigen, dass

(a) das Problem in der Klasse **NP** liegt und dass

(b) das Problem mindestens so schwierig wie ein anderes (bekanntes) **NP**-vollständiges Problem ist.

Wir beweisen die Aussage einzeln.

(a) Die Idee zum Nachweis des Enthaltenseins des Problems in **NP** besteht darin, k Substitutionen auf dem Hypertext zu raten und das exakte Suchrpoblem im entstandenen Hypertext zu lösen. Der in Abbildung 2.10 beschriebene nicht-deterministische Algorithmus realisiert genau dies. Eine einfache Analyse unter Zuhilfenahme von Theorem 2.14 ergibt, dass der Algorithmus pro geratener Substitutionenfolge im schlechtesten Fall $O(\|\Sigma\|^2 \cdot N + mN + m\|E\|)$ Schritte benötigt und mithin in nicht-determinstischer polynomieller Zeit läuft.

(b) Zum Nachweis der Härte verwenden wir als bekanntes **NP**-vollständiges Problem Directed Hamiltonian Path. Diese Problem besteht darin, in einem gerichteter Graphen $G = (V, E)$ einen gerichteten Hamilton-Pfad zu finden, d.h. einen gerichteten Pfad, der alle Knoten des Graphen genau einmal durchläuft.

Unsere Aufgabe ist es nun, für einen (beliebigen) gegebenen gerichteten Graphen G einen Hypertext H^G, ein Suchwort s^G und einen Parameter k^G zu konstruieren (in $O(|G|^r)$ Schritten für irgendein $r \in \mathbb{N}$), so dass gilt

$$\begin{aligned} &G \text{ enthält Hamilton-Pfad} \\ &\qquad\iff \quad s^G \text{ kommt in } H^G \text{ mit höchstens } k^G \text{ Subsitutionen auf } H^G \text{ vor.} \end{aligned}$$

Es sei nun $G = (V, E)$ ein gerichteter Graph, $n = \|V\|$. Dann betrachten wir als Instanz für das approximative Suchproblem

- den Hypertext $H^G =_{\text{def}} (V, E, T)$ über $\Sigma = \{a_1, \ldots, a_n\}$ mit $T(v) = a_1$ für alle $v \in V$,
- das Suchwort $s^G =_{\text{def}} a_1 a_2 \ldots a_n$ (wobei $a_i \neq a_j$ für $i \neq j$ gilt) und
- den Fehlerparameter $k^G =_{\text{def}} n - 1$.

Wir müssen zwei Richtungen zeigen.

Sei $(v_1, \ldots, v_n)$ ein Hamilton-Pfad in G. Wir ersetzen für jeden Knoten v_i den Inhalt $T(v_i)$ durch a_i. Dies sind $n - 1$ Substitutionen, da wir den Inhalt eines Knotens unverändert lassen. Damit gilt $d_H(s^G, T(v_1)T(v_2)\ldots T(v_n)) \leq n - 1 = k^G$.

Sei $(v_1, \ldots, v_n)$ ein Weg in H mit $d_H(s^G, T(v_1)T(v_2)\ldots T(v_n) \leq k^G$, d.h. es gibt Ersetzung von höchstens k^G Knoteninhalten, so dass gerade das Wort s^G entsteht. Wegen $n = \|V\|$ und $a_i \neq a_j$ für $i \neq j$ muss somit $v_1 \neq v_j$ für $i \neq j$ auf dem Weg gelten. Somit ist $(v_1, \ldots, v_n)$ ein Hamilton-Pfad.

Damit ist das Theorem bewiesen. ■

Korollar 2.17 *Unter der Annahme* $\mathbf{P} \neq \mathbf{NP}$ *gibt es keinen Algorithmus, der ein Suchwort der Länge* m *in einem Hypertext* $H = (V, E, T)$ *bis auf* k *Veränderungen im Hypertext findet und dafür stets mit* $O\left((m + N + \|E\| + k)^r\right)$ *Schritten für irgendein* $r \in \mathbb{N}$ *auskommt.*

Bemerkung. Theorem 2.16 (und damit Korollar 2.17) gilt nicht für azyklischen Hypertext.

2.2.4 Der Algorithmus von Navarro

Wir betrachten nun die Version S des approximativen Suchproblems für den Editierabstand, d.h. Einfügungen, Löschungen und Substitutionen dürfen nur im Suchwort vorgenommen werden.

Dazu führen wir zunächst eine Adaption des Algorithmus von Sellers durch. Hierbei müssen die Fehlerpropagationsregel (für die Dynamische Programmierung) aus Proposition 2.8 anpassen. Zur Erinnerung:

$$D_{st}[i,j] = \begin{cases} D_{st}[i-1,j-1], & \text{falls } s_i = t_j \\ 1 + \min\{\underbrace{D_{st}[i-1,j]}_{\text{Lösche } s_i}, \underbrace{D_{st}[i,j-1]}_{\text{Füge } t_j \text{ an } s_i \text{ an}}, \underbrace{D_{st}[i-1,j-1]}_{\text{Ersetze } s_i \text{ durch } t_j}\}, & \text{falls } s_i \neq t_j \end{cases}$$

Der Algorithmus von Sellers durchläuft die Matrix D_{st} spaltenweise (aus Gründen der Platzeffizienz). Dieses Vorgehen funktioniert hier nicht mehr, da der Hypertext nicht linear ist. Wir müssen die Matrix D_{st} zeilenweise durchlaufen. Damit erhalten wir als Fehlerpropagationsregel (immer noch auf zwei Texte bezogen):

$$D[j] = \begin{cases} A[j-1], & \text{falls } s_i = t_j \\ 1 + \min\{\underbrace{A[j]}_{\text{Lösche } s_i}, \underbrace{D[j-1]}_{\text{Füge } t_j \text{ an } s_i \text{ an}}, \underbrace{A[j-1]}_{\text{Ersetze } s_i \text{ durch } t_j}\}, & \text{falls } s_i \neq t_j \end{cases}$$

Dabei ist D die aktuelle Zeile und A die in der Matrix darüber liegende Zeile.

Der Eintrag $D[j]$ bei der i-ten Iteration ist als minimaler Editierabstand von $s_1 \dots s_i$ zu einem Suffix $t_1 \dots t_j$ zu interpretieren. Uminterpretiert auf Hypertext würde der Eintrag $D[j]$ bei der i-ten Iteration dem minimalen Editierabstand von $s_1 \dots s_i$ zu einem Suffix von $T(u_1)T(u_2)\dots T(u_{j-1})T(v_j)$ entlang irgendeines Weges $(u_1, u_2, \dots, u_{j-1}, v_j)$, der im Knoten v_j endet, entsprechen.

Wir erhalten somit als neue Fehlerpropagationsregel für den Hypertext $H = (V, E, T)$:

$$f(v,i) = \begin{cases} \min(\{A[u] \mid (u,v) \in E\} \cup \underbrace{\{i-1\}}_{\text{falls in-deg}(v)=0}), & \text{falls } s_i = T(v) \\ 1 + \min\left\{A[v], \min\limits_{(u,v)\in E} D[u], \min\limits_{(u,v)\in E} A[u]\right\}, & \text{falls } s_i \neq T(v) \end{cases}$$

Algorithmus:	FirstAlgorithm
Eingabe:	elementarer Hypertext $H = (V, E, T)$, Suchwort $s \in \Sigma^*$ und Parameter $k \in \mathbb{N}$
Ausgabe:	Knoten $v \in V$, so dass Weg in H existiert, der in v endet und ein Suffix entlang des Weges Editierabstand höchstens k zu s hat (wobei Operationen nur auf s ausgeführt werden dürfen)

```
1.   FOR v ∈ V
2.       A[v] := 0
3.       D[v] := 0
4.   FOR i := 1 TO m
5.       FOR v ∈ V
6.           D[v] := f(v, i)
7.       FOR v ∈ V
8.           A[v] := D[v]
9.       FOR v ∈ V
10.          IF D[v] ≤ k
11.              Gebe v zurück
```

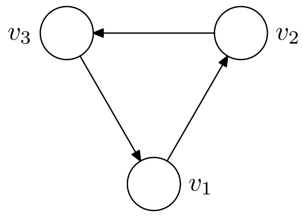
Abbildung 2.11: Ein erster Algorithmus für approximative Suche in Hypertext

Als direkte Übertragung des Algorithmus von Sellers mit Hilfe der neuen Fehlerpropagationsregel erhalten wir den Abbildung 2.11 dargestellten Algorithmus FirstAlgorithm.

Die Laufzeit von FirstAlgorithm beträgt offensichtlich $O(mN + m\|E\|)$. Allerdings besteht noch das Problem festzulegen, in welcher Reihenfolge die Knoten durchlaufen werden (in der dritten Zeile). Für azyklischen Hypertext sollte dies klarerweise gemäß der topologischen Sortierung der Knoten erfolgen.

Proposition 2.18 *Der Algorithmus* FirstAlgorithm *(mit Verwendung der topologischer Sortierung) benötigt für die Suche eines Wortes der Länge m in einem elementaren azyklischen Hypertext $H = (V, E, T)$ mit bis zu k Editierfehlern im Suchwort im schlechtesten Fall $O(mN + m\|E\|)$ Schritte.*

Unser Ziel ist natürlich, auch für Hypertexte mit Kreisen über einen (korrekten) Algorithmus mit Laufzeit $O(mN + m\|E\|)$ zu verfügen. Der kritische Fall hierbei sind Einfügeoperationen, da auf die Einträge in D zurückgegriffen werden muss, die noch nicht ihre aktuellen Werte erhalten haben. Exemplarisch ist dies am Beispiel



zu sehen, denn es gilt bei dreimaliger Einfügung im Suchwort:

$$\begin{aligned} D[v_1] &= 1 + D[v_3] \\ D[v_2] &= 1 + D[v_1] \\ D[v_3] &= 1 + D[v_2] \end{aligned}$$

Mit der Festlegung einer speziellen Reihenfolge für die Fehlerpropagation ist das Problem der Widersprüchlichkeit derartiger Gleichungssysteme nicht zu beheben.

Eine erste Idee für einen Ausweg ist, Einfügungen durch ein verändertes Suchwort zu vermeiden. Dazu wählen wir einen neuen Buchstaben $\square \notin \Sigma$ mit den folgenden Editierkosten:

$$\begin{aligned} c(\square, \varepsilon) &= 0 \quad \text{(d.h. Löschen von } \square \text{ zählt nicht)} \\ c(\square, a) &= 1 \quad \text{(d.h. Ersetzen von } \square \text{ zählt als ein Fehler)} \end{aligned}$$

Wir betrachten nun das Suchwort $s' =_{\mathrm{def}} \square^k s_1 \square^k s_2 \square^k \cdots \square^k s_m \square^k$. Als Anschauung steht dahinter, dass wenn wir in s einen Buchstaben einfügen wollen, so ersetzen wir in s' den Platzhalter $\square$ durch diesen Buchstaben. Maximal können wir jedoch nur k Einfügungen vornehmen.

Als neue Fehlerpropagationsregel erhalten wir nunmehr:

$$f'(v,i) = \begin{cases} \min(\ \{A[u] \mid (u,v) \in E\} \ \cup \ \{i-1\}), & \text{falls } s'_i = T(v) \\ \min\left\{ c_{\mathrm{DEL}}(v,i), 1 + \min\limits_{(u,v)\in E} A[u] \right\}, & \text{falls } s'_i \neq T(v) \end{cases}$$

wobei

$$c_{\mathrm{DEL}}(v,i) = \begin{cases} A[v], & \text{falls } s'_i = \square \\ 1 + A[v], & \text{sonst} \end{cases}$$

Wir modifizieren FirstAlgorithm durch Einsetzen der neuen Fehlerpropagationsregel und lassen ihn auf s' an Stelle von s laufen. Die Reihenfolge, in der die Propagationsregeln auf die Knoten angewendet wird, ist unerheblich.

Theorem 2.19 *Der Algorithmus* FirstAlgorithm *(bei Verwendung der modifizierten Fehlerpropagationsregel) benötigt für die Suche eines Wortes der Länge m in einem elementaren Hypertext $H = (V, E, T)$ mit bis zu k Editierfehlern im Suchwort im schlechtesten Fall $O(mk(N + \|E\|)$ Schritte (und $O(n)$ Platz).*

Beweis: Offensichtlich. ■

Wir müssen nun noch den Faktor k aus der Laufzeitschranke in Theorem 2.19 eliminieren. Dazu führen wir die Fehlerpropagation durch, ohne das Symbol $\square$ tatsächlich in das

Suchwort s einzufügen. D.h. nach Aufdatierung (ohne Einfügen) müssen wir k-mal über alle Kanten $(u, v) \in E$ iterieren und folgende Regel anwenden:

$$D[v] := \min(\underbrace{A[v]}_{\text{Löschen}} , \underbrace{1 + A[u]}_{\text{Ersetzen}}) \tag{2.3}$$

Mit diesem Vorgehen benötigen wir jedoch immer noch $O(mk(N + \|E\|))$ Schritte. Deshalb werden wir Buch darüber führen, auf welche Kanten sich die Regel (2.3) überhaupt auswirkt: nur auf solche Kanten mit $A[v] > 1 + A[u]$. Auf diesen Kanten führen wir die Fehlerpropagation gemäß der Regel (2.3) aus.

Generell verwenden wir folgende Fehlerpropagationsregel:

$$g(v, i) =_{\text{def}} \begin{cases} \min(\ \{ A[u] \mid (u, v) \in E\} \ \cup \ \{i - 1\} \ ), & \text{falls } s_i = T(v) \\ 1 + \min(\overbrace{A[v]}^{\text{Löschen}} , \overbrace{\min_{(u,v) \in E} A[u]}^{\text{Ersetzen}}), & \text{falls } s_i \neq T(v) \end{cases}$$

Der Algorithmus von Navarro in Abbildung 2.12 fasst unsere Analyse zusammen.

Theorem 2.20 *Der Algorithmus von Navarro benötigt für die Suche eines Wortes der Länge m in einem elementaren Hypertext $H = (V, E, T)$ mit bis zu k Editierfehlern im schlechtesten Fall $O(mN + m\|E\|)$ Schritte.*

Beweis: Wir führen zunächst die Komplexitätsanalyse für die Einfügepropagation durch. (Im Prinzip ist erst noch zu klären, ob der Propagationsprozess überhaupt abbricht.) Dazu verwenden wir die amortisierte Analyse für jeden Schleifendurchlauf von $i \in \{1, \ldots, m\}$. Wir führen folgende Bezeichnungen ein für $i \in \{1, \ldots, m\}$ und $v \in V$:

$A_0^i[v]$	$=_{\text{def}}$	Wert von $A[v]$ im i-ten Schleifendurchlauf nach Ausführung der Zeilen 7–8
$A_1^i[v]$	$=_{\text{def}}$	Wert von $A[v]$ im i-ten Schleifendurchlauf nach Ausführung der Zeile 9–10 (d.h. nach erfolgter Fehlerpropagation)

Wir definieren die Einzelpotenziale Γ und Φ für $i \in \{0, 1, \ldots, m\}$ wie folgt:

$$\Gamma(i) =_{\text{def}} \sum_{(u,v) \in E} A_0^i[u]$$

$$\Phi(i) =_{\text{def}} \sum_{(u,v) \in E} A_1^i[u]$$

Folgende Aussagen gelten:

1. $\Phi(0) = \Gamma(0) = 0$.
2. Für alle $1 \leq i \leq m$ gilt $\Gamma(i) \leq \Phi(i - 1) + \|E\|$.

Algorithmus: NAVARRO
Eingabe: elementarer Hypertext $H = (V, E, T)$, Suchwort $s \in \Sigma^*$ und Parameter $k \in \mathbb{N}$
Ausgabe: Knoten $v \in V$, so dass Weg in H existiert, der in v endet und ein Suffix entlang des Weges Editierabstand höchstens k zu s hat (wobei Operationen nur auf s ausgeführt werden dürfen)

```
FOR v ∈ V
    A[v] := 0
    D[v] := 0
FOR i := 1 TO m
    FOR v ∈ V
        D[v] := g(v, i)
    FOR v ∈ V
        A[v] := D[v]
    FOR (u, v) ∈ E
        PROPAGATE(u, v)
FOR v ∈ V DO
    IF D[v] ≤ k
        Gebe v zurück
```

Algorithmus: PROPAGATE(u, v)

```
(a) IF (u, v) ∈ E AND A[v] > 1 + A[u]
(b)     A[v] := 1 + A[u]
(c)     FOR (v, z) ∈ E
(d)         PROPAGATE(v, z)
```

Abbildung 2.12: Der Algorithmus von Navarro

3. Für alle $1 \leq i \leq m$ gilt $\Phi(i-1) \leq \Phi(i) + \|E\|$.

Wir überprüfen die Aussagen einzeln.

1. Trivial.

2. Der Unterschied zwischen $A_1^{i-1}[u]$ und $A_0^i[u]$ wird durch die Zeile 5–6 bestimmt. Damit gilt $A_0^i[u] \leq 1 + A_1^{i-1}$ für alle $(u,v) \in E$. Folglich ergibt sich

$$\Gamma(i) = \sum_{(u,v)\in E} A_0^i[u] \leq \sum_{(u,v)\in E} (A_1^{i-1}[u] + 1) = \Phi(i-1) + \|E\|.$$

3. Der Wert $A_1^i[u]$ gibt den Editierabstand zwischen $s_1 \ldots s_i$ und einem Suffix entlang eines Weges mit Endpunkt u an. Damit gilt $A_1^{i-1}[u] \leq 1 + A_1^i[u]$ (Begründung: Füge s_i an $s_1 \ldots s_{i-1}$ an). Folglich ergibt sich

$$\Phi(i-1) = \sum_{(u,v)\in E} A_1^{i-1}[u] \leq \sum_{(u,v)\in E} (A_1^i[u] + 1) = \Phi(i) + \|E\|.$$

Insgesamt erhalten wir, dass für die Anzahl R der durch die Zeile 5–6 dekrementierten A-Werte gilt

$$R \leq \Gamma(i) - \Phi(i) \leq \Phi(i-1) + \|E\| - (\Phi(i-1) - \|E\|) = 2\|E\|.$$

Für die Gesamtlaufzeit des Algorithmus erhalten wir:

$$O(\underbrace{m}_{\text{Zeile 4}} \cdot (\underbrace{N}_{\text{Zeilen 5–6}} + \underbrace{N}_{\text{Zeilen 7–8}} + \underbrace{2\|E\|}_{\text{Zeilen 9–10}} + \underbrace{N}_{\text{Zeilen 1–3}} + \underbrace{N}_{\text{Zeilen 11–13}})) = O(mN + m\|E\|).$$

■

Bemerkungen.

1. Der Platzbedarf des Navarro-Algorithmus ist $O(n)$ (da die Tiefe des Rekursionsbaumes für die Propagate-Aufrufe durch $O(m)$ beschränkt ist).

2. Der Algorithmus von Navarro kann auf allgemeinen Hypertext erweitert werden mit der Laufzeit $O(mn + m(\|E\| + mN) + kn)$.

3. Der Algorithmus von Navarro kann auch für die exakte Suche im Hypertext angewendet werden.

2.3 Datenströme

Ziel dieses Abschnittes ist es, effiziente Algorithmen für die Ermittlung von Statistiken bei hoher Datendichte und eingeschränktem Zugriff auf die Daten zu entwerfen.

Typische Anwendungen:

- Internet-Archive
- Web-Log-Analyse
- Router-Statistiken („Wem gehören die meisten Pakete?“ usw. usf.)

2.3.1 Verarbeitungsmodelle und Komplexitätsmaße

Es sei $\Sigma = \{a_1, \ldots, a_n\}$ ein Alphabet der Größe n. (Wir betrachten aber auch unbeschränkte Alphabete.) Ein *Datenstrom* s ist eine endliche Folge $s = (s_1, \ldots, s_i, \ldots, s_N)$ mit $s_i \in \Sigma$.

Interpretation eines Verarbeitungsmodells für Datenströme:

In einem Datenstrom s wird jedes Element s_i nur einmal gelesen und zwar in aufsteigender Reihenfolge der Indizes.

Ein Beispiel für eine Datenstromverarbeitung liegt bei Back-Ups, die auf Bändern hinterlegt sind, vor.

Komplexitätsmaße in solchen Verarbeitungsmodellen:

- Anzahl der Pässe: „Wie oft liest ein Algorithmus den Datenstrom?“Algorithmen, die nur ein Pass über den Datenstrom benötigen, heißen *Online-Algorithmen*.
- Speicherplatz (als Funktion von n, N)
- Verarbeitungszeit pro Datenstromelement

Charakteristik guter Algorithmen auf Datenströmen:

- möglichst ein Pass (oder nur sehr wenige)
- sublinearer Speicherplatz (möglichst logarithmisch)
- $O(1)$ amortisierte Verarbeitungszeit pro Datenstromelement (d.h. insgesamt lineare Verarbeitungszeit)

Beispiel. Die Algorithmen von Knuth, Morris und Pratt bzw. Boyer und Moore für exakte Suche in Texten sind so, wie sie beschrieben wurden, keine Online-Algorithmen, da bei der Bestimmung der Tabelle der Ränderlängen mehrmals auf Elemente zugegriffen wird. Die Ausführung der Suche selbst geschieht in einem Pass über das Wort t, d.h. für ein festes Suchwort sind beide Algorithmen tatsächlich Online-Algorithmen mit konstantem Platzbedarf.

2.3.2 Der Algorithmus von Fischer und Salzberg

Wir betrachten das folgende wichtige Auswertungsproblem: Gegeben sei ein Datenstrom $s = (s_1, \ldots, s_N)$ über einem Alphabet Σ mit $\|\Sigma\| = n$. Bestimme die Element in s, die mit Häufigkeit ϑ vorkommen, d.h. geben die Menge

$$H(s, \vartheta) =_{\text{def}} \{ \ a \in \Sigma \mid |s|_a > \vartheta \cdot N \ \}$$

aus, wobei $\vartheta \in [0, 1)$ eine reelle Zahl und $|s|_a$ die Anzahl der Vorkommen von a in s ist.

Theorem 2.21 *Jeder Online-Algorithmus zur Bestimmung der Menge $H(s, \vartheta)$ für einen Datenstrom s benötigt im schlechtesten Fall $\Omega(n \log \frac{N}{n})$ Speicherplatz.*

Beweis: Es sei $N > 4n > \frac{16}{\vartheta}$. Es sei A ein Online-Algorithmus, der die Menge $H(s, \vartheta)$ bestimmt. Wir betrachten den Zustand von A in dem Moment, wenn das Element s_M mit $M = \lfloor \frac{N+1}{2} \rfloor$ gelesen wird. Da A nicht mehr die Möglichkeit hat, auf die Datenstromelemente s_i mit $i \leq M$ zuzugreifen, muss der Algorithmus A alle Datenstrompräfixe mit verschiedenen Elementhäufigkeiten unterscheiden können, d.h. alle $s' = (s'_1, \ldots, s'_M)$ und $s'' = (s''_1, \ldots, s''_M)$ mit $|s'|_a \neq |s''|_a$ für irgendein $a \in \Sigma$. Anderenfalls könnten die zweiten Hälften der Datenströme mit $u = (u_{M+1}, \ldots, u_N)$ ergänzt werden, so dass $a \in H(s'u, \vartheta)$ und $a \notin H(s''u, \vartheta)$ gilt, aber A für beide Datenströme entweder a ausgibt oder a nicht ausgibt. In beiden Fällen wäre A nicht korrekt. Dieses Argument funktioniert jedoch nur, wenn $|s'|_a \leq \vartheta \cdot N$ und $|s''|_a \leq \vartheta \cdot N$ für alle $a \in \Sigma$ gilt. Die Menge dieser Datenstrompräfixe lässt sich durch die folgende Menge K_N repräsentieren:

$$K_N =_{\text{def}} \{(k_1, \ldots, k_n) \in \mathbb{N}^n \mid k_i \leq \vartheta \cdot N, \textstyle\sum_{j=1}^n k_j = \lfloor \frac{N}{2} \rfloor\}.$$

Der Algorithmus A benötigt deshalb mindestens $\log \|K_N\|$ Bits als Speicherplatz. Wir müssen nun also $\|K_N\|$ nach unten abschätzen.

Wenn wir $\lfloor \frac{N}{2} \rfloor$ wie folgt auffassen

$$\left\lfloor \frac{N}{2} \right\rfloor = \underbrace{1 + \cdots + 1}_{k_1} + \underbrace{1 + \cdots + 1}_{k_2} + \cdots + \underbrace{1 + \cdots + 1}_{k_n},$$

so lässt sich jedes Tupel aus K_N interpretieren als ein Tupel $(b_0, b_1, \ldots, b_n)$ mit den Eigenschaften $0 = b_0 \leq b_1 \leq \cdots \leq b_n = \lfloor \frac{N}{2} \rfloor + n$ und $k_i = b_i - b_{i-1} - 1$ für alle $1 \leq i \leq n$. Die Werte b_i sind nichts anderes als die Nummer der $+$-Zeichen zwischen den zu k_{i-1} und k_i

gehörenden Blöcken, d.h. die Positionen der Grenzen zwischen den Blöcken $i-1$ und i. Wir betrachten nun die Menge

$$K'_N =_{\text{def}} \{\ (k_1, \ldots, k_n) \in \mathbb{N}^n) \mid \text{für alle } i \text{ gilt } k_i = \lfloor \tfrac{N}{2n} \rfloor \text{ oder } k_i = \lceil \tfrac{N}{2n} \rceil \}$$

Für Tupel aus K'_N können jedoch für jedes $i \in \{1, \ldots, n-1\}$ die zugehörigen Grenzen zu den Blöcken $i-1$ bzw. $i+1$ unabhängig um bis zu $\lfloor \frac{N}{4n} \rfloor$ nach links und rechts verschoben werden und es gilt $\lceil \frac{N}{2n} \rceil + \lfloor \frac{N}{4n} \rfloor \leq \vartheta \cdot N$. Mithin gilt

$$\|K_N\| \geq \left(2\left\lfloor \frac{N}{4n} \right\rfloor + 1\right)^{n-1}.$$

Folglich erhalten wir $\log \|K_N\| = \Omega(n \log \frac{N}{n})$ als untere Schranke für den Speicherplatz von A. ■

Wir gehen nunmehr zur Betrachtung von Zwei-Pass-Algorithmen über; zunächst jedoch nur für $\vartheta = \frac{1}{2}$. D.h. gegeben ein Datenstrom $s = (s_1, \ldots, s_N)$ bestimme dasjenige Element, das mit absoluter Mehrheit in s vorkommt (Mehrheitselement).

Als Idee verwenden wir folgende einfache Beobachtung:

> *Wenn wir alle in einem Datenstrom vorkommenden Elemente so umordnen können, dass benachbarte Elemente verschieden sind, so kann ein Element höchstens $\lceil \frac{N}{2} \rceil$-mal im Datenstrom vorkommen.*

Der in Abbildung 2.13 beschriebene Algorithmus von Fischer und Salzberg setzt gerade diese Idee um.

Theorem 2.22 *Der Algorithmus von Fischer und Salzberg benötigt für die Bestimmung des Elements mit absoluter Häufigkeit mindestens $\lfloor \frac{N+1}{2} \rfloor$ aus einem Datenstrom der Länge N höchstens $\lceil \frac{3}{2}N \rceil - 2$ Vergleiche.*

Beweis: Die Korrektheit des Algorithmus ergibt sich aus folgenden Überlegungen:

1. Während des ersten Passes ist folgende Invariante nach jedem Schleifendurchlauf erfüllt: In Q_2 sind nur Elemente enthalten, die dem obersten Element in Q_1 entsprechen. Somit folgt: Gibt es ein Mehrheitselement in s, so ist dies a.

2. Im zweiten Pass gilt stets, dass wann immer ein Paar von Elemente aus $Q_1 \cup Q_2$ entfernt wird (Zeilen 12-14; Zeilen 18, 20,21; Zeilen 18, 28), ist ein Element gerade a und das andere Element verschieden von a. Wir unterscheiden zwei Fälle:

   (a) Der zweite Pass bricht vorzeitig in Zeile 26 ab. Dann war Q_2 leer, d.h. höchstens die Hälfte der Elemente waren gleich zu a, d.h. a ist kein Mehrheitselement.

Algorithmus: Fischer-Salzberg
Eingabe: Datenstrom $s = (s_1, \ldots, S_N)$ über Σ
Ausgabe: Element $a \in \Sigma$ mit $|s|_a \geq \lfloor \frac{N+1}{2} \rfloor$

```
/* Erster Pass */

a := s_1
Push(a, Q_1)
FOR i := 2 TO N
    IF s_i ≠ a
        a := s_i
        Push(a, Q_1)
        IF Q_2 ist nicht leer
            a :=Pop(Q_2)
            Push(a, Q_1)
    ELSE
        Push(s_i, Q_2)

/* Zweiter Pass */

b :=Pop(Q_1)
IF Q_1 ist nicht leer
    Pop(Q_1)
ELSE
    Push(b, Q_2)
WHILE Q_1 ist nicht leer
    b :=Pop(Q_1)
    IF a = b
        IF Q_1 ist nicht leer
            Pop(Q_1)
        ELSE
            Push(b, Q_2)
    ELSE
        IF Q_2 ist leer
            Gebe „Kein Mehrheitselement vorhanden“ aus
        ELSE
            Pop(Q_2)
IF Q_2 ist nicht leer
    Gebe a aus
```

Abbildung 2.13: Der Algorithmus von Fischer und Salzberg

(b) Der zweite Pass wird vollständig durchlaufen. Damit bleiben nur Elemente in Q_2 übrig.

Insgesamt gilt also:

$$a \text{ ist Mehrheitselement von } s \iff Q_2 \text{ ist nicht leer nach Zeile 28}$$

Genau dieses Kriterium wird in Zeile 29 getestet. Mithin ist der Algorithmus von Fischer und Salzberg korrekt.

Eine einfache Analyse der Anzahl der Vergleiche ergibt:

- Für die erste Phase sind höchstens $N-1$ Vergleiche nötig.
- Für die zweite Phase sind höchstens $\lceil \frac{N}{2} \rceil - 1$ Vergleiche nötig (ein Vergleich pro entferntem Elementpaar und eventuell zusätzlich ein Vergleich für ei Element, das von Q_1 in Q_2 wechselt).

Es werden also insgesamt höchstens $N - 1 + \lceil \frac{N}{2} \rceil - 1 = \lceil \frac{3}{2} N \rceil - 2$ benötigt. ■

Bemerkungen.

1. Der Algorithmus von Fischer und Salzberg in obiger Form ist eigentlich ein Online-Algorithmus mit linearer Platzkomplexität.
2. Es kann gezeigt werden, dass kein Algorithmus für die Bestimmung des Mehrheitselementes mit weniger als $\lceil \frac{3}{2} N \rceil - 2$ Vergleichen auskommt.

Wir betrachten nun den allgemeinen Fall $\vartheta \in [0, 1)$. Dabei wissen wir, dass $\|H(s, \vartheta)\| < \frac{1}{\vartheta}$ gilt. Wir verallgemeinern die Idee, die dem Algorithmus von Fischer und Salzberg zu Grunde liegt:

> *Wenn wir alle Elemente im Datenstrom so anordnen können, dass immer Blöcke der Größe $\lfloor \frac{1}{\vartheta} \rfloor$ mit paarweise verschiedenen Elementen entstehen, so kann ein Element höchstens $\lfloor \vartheta \cdot N \rfloor$-mal im Datenstrom vorkommen.*

Für einen Datenstrom s und ein festes $\vartheta \in [0, 1)$ berechnen wir mit Hilfe dieser Idee zunächst in einem Pass eine Kandidatenmenge K mit $\|K\| \leq \lfloor \frac{1}{\vartheta} \rfloor$ und $H(s, \vartheta) \subseteq K$. Dies erfolgt mit dem Algorithmus ComputeCandidates wie in Abbildung 2.14 beschrieben.

Proposition 2.23 *Der Algorithmus* ComputeCandidates *ist korrekt.*

Algorithmus: ComputeCandidates
Eingabe: Datenstrom $s = (s_1, \ldots, S_N)$ über Σ
Ausgabe: K mit $\|K\| \leq \lfloor \frac{1}{\vartheta} \rfloor$ und $H(s, \vartheta) \subseteq K$

```
1.    K := ∅
2.    FOR i := 1 TO N
3.      IF s_i ∈ K
4.        C[s_i] := C[s_i] + 1
5.      ELSE
6.        Füge s_i in K ein
7.        C[s_i] := 1
8.      IF ||K|| > 1/ϑ
9.        FOR a ∈ K
10.         C[a] := C[a] − 1
11.       IF C[a] = 0
12.         Lösche a aus K
13.   Gebe K aus
```

Abbildung 2.14: Berechnung der Kandidatenmenge für $\vartheta \in [0, 1)$

Beweis: Es sei $a \in \Sigma$ ein Element, dass von ComputeCandidates nicht ausgegeben wird, d.h. $a \notin K$. Jedes Vorkommen von a wurde gelöscht mit zusammen mindestens $\frac{1}{\vartheta} - 1$ Vorkommen anderer Symbole (Zeilen 8–12). Insgesamt werden mehr als $\frac{|s|_a}{\vartheta}$ Vorkommen von Buchstaben entfernt. Mithin gilt $\frac{|s|_a}{\vartheta} \leq N$ bzw. $|s|_a \leq \vartheta \cdot N$. Es folgt $a \notin H(s, \vartheta)$. ■

Für die Komplexität von ComputeCandidates ergibt sich zunächst bei Implementierung von K mittels Hash-Tabelle und C als dynamischem Array:

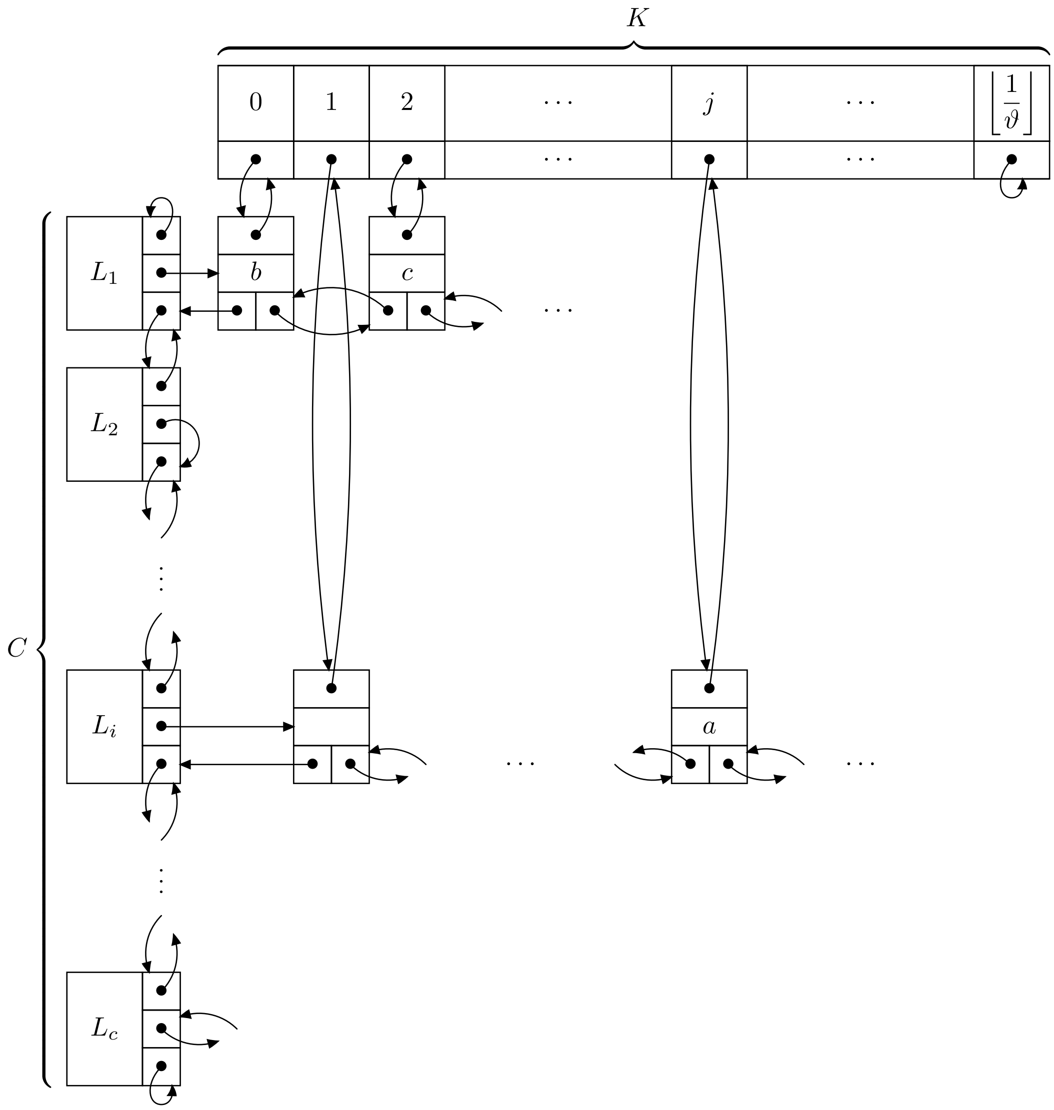
- $O(\frac{1}{\vartheta})$ Speicherplatz
- $O(1)$ amortisierte Laufzeit pro Datenstromelement (ohne Hashing)

Verfeinerte Datenstrukturen für K und C:

1. Operationen:
   - INSERT(a, K): Füge neues Element a in K ein
   - INC(a, K): Inkrementiere Zähler von Element $a \in K$
   - DEC-ALL(K): Dekrementiere alle Zähler für Elemente $a \in K$ (und entferne Elemente mit Zählerstand 0)
2. Implementierungen:
   - K als Hash-Tabelle

- C als Linkstruktur $(L_1, \dots, L_c)$ mit folgender Interpretation:
  - L_i ist Linkstruktur $(a_{i_1}, \dots, a_{i_r})$ mit $a_{i_j} \in K$ und $C[a_{i_j}] = i$
  - L_c enthält Elemente mit höchstem aktuell vorkommenden Zählerstand
- INSERT(a, K): Füge a in K und L_1 ein
- INC(a, K): Entferne a aus L_i und füge a in L_{i+1} ein (wenn $i = c$, so erzeuge neue Liste L_{i+1})
- DEC-ALL(K): Entferne L_1 und alle $a \in L_1$ aus K.

Damit ergibt sich folgendes Schema für die verfeinerte Implementierung von K und C:



Für die Komplexitätsanalyse erhalten wir nun:

- Speicherplatz $O(\frac{1}{\vartheta}) + O(c)$ (der additive Term $O(c)$ kann eliminiert werden, indem leere Listen nicht verwaltet werden und stattdessen in L_i immer die Anzahl k leerer Listen bis zur nächsten nicht-leeren Liste L_{i+k+1} notiert wird)
- Laufzeit $O(1)$ pro Datenstromelement im schlechtesten Fall (ohne Hashing)

Theorem 2.24 *Es gibt einen Zwei-Pass-Algorithmus für die Bestimmung von $H(s, \vartheta)$ auf einem Datenstrom s der Länge N für $\vartheta \in [0, 1)$, der mit $O(\frac{1}{\vartheta})$ Platz und $O(1)$ Schritten (ohne Hashing) pro Datenstromelement im schlechtesten Fall auskommt.*

Beweis: Der Algorithmus führt im ersten Pass den Algorithmus COMPUTE CANDIDATES aus und bestimmt im zweiten Pass $|s|_a$ für alle $a \in K$ und gibt alle $a \in K$ mit $|s|_a > \vartheta \cdot N$ aus. Korrektheit folgt aus Proposition 2.23. Die Platz- und Laufzeitschranken aus obiger Implementierung. ∎

Bemerkung. Für die Laufzeiten liegt das uniforme Kostenmodell zu Grunde.

2.3.3 Schätzung von Häufigkeitsmomenten

Wir interessieren uns weiter für die statistische Auswertung ganzer Datenströme. Dazu betrachten wir folgende statistischen Größen.

Definition 2.25 *Es seien Σ ein endliches Alphabet der Größe n und $s = (s_1, \dots, s_N)$ ein Datenstrom der Größe N über Σ.*

1. *Für $k \in \mathbb{N}$ ist das* Häufigkeitsmoment *(statistische Moment) $F_k(s)$* k-ter Ordnung *von s definiert als*
$$F_k(s) =_{\text{def}} \sum_{a \in \Sigma} |s|_a^k.$$
2. *Das* Häufigkeitsmoment $F_\infty(s)$ der Ordnung ∞ *von s ist definiert als*
$$F_\infty(s) =_{\text{def}} \max_{a \in \Sigma} |s|_a.$$

Beispiele.

- $F_0(s) = \|\{s_1, \dots, s_N\}\|$ (Anzahl verschiedener Elemente in s).
- $F_1(s) = N$.
- $F_2(s)$ heißt auch Homogenitätsindex von Gini (wird z.B. auch benutzt, um Einkommensverteilung von Volkswirtschaften zu beurteilen).

Im Folgenden beschäftigen wir uns mit dem Problem, wie Platz-effizient die Momente berechnet werden können?

Proposition 2.26 *Die Häufigkeitsmomenten für einen Datenstrom der Länge N über einem Alphabet der Größe n lassen sich exakt mit $O(n \log N)$ Platz berechnen.*

Beweis: Benutze Binärzähler für jeden möglichen Buchstaben und berechne Ergebnis aus den Zählungen. ■

Bemerkung. Für festes Alphabet (und variables k) ist dies Platzschranke optimal für exakte Online-Algorithmen (verwende sogenannte Spurenmethode aus Theorem 2.21)

Um die lineare Platzschranke zu brechen, versuchen wir Ansätze mit randomisierten Algorithmen.

Erster Ansatz (Morris): *Probabilistisches Zählen in Binärzähler der Größe* $O(\log \log N)$

Dazu speichern wir in einem Zähler C für eine Zahl m nicht die Zahl m sondern die Repräsentierung $\log(1+m)$. Dabei entsteht folgendes Problem:

$$\begin{array}{rcl}
\text{Zählerinhalt} & & \text{repräsentierte Zahl} \\
c & \longrightarrow & m_c = 2^c - 1 \\
 & & \quad \downarrow \text{Inkrementierung} \\
c' = \log(1+2^c) \notin \mathbb{N} & \longleftarrow & m_c + 1 = 2^c
\end{array}$$

Ein Ausweg aus diesem Problem besteht darin, den Wert c im Zähler nur mit Wahrscheinlichkeit Δ zu inkrementieren und mit Wahrscheinlichkeit $1-\Delta$ unverändert zu lassen. Ein gute Wahl für Δ ist

$$\frac{1}{\Delta} =_{\text{def}} m_{c+1} - m_c.$$

Das Algorithmenfragment sieht dann wie folgt aus:

```
RANDOM x ∈ [0,1]
IF x ≤ Δ
    c' := c + 1
ELSE
    c' := c
```

Für die erwartete repräsentierte Zahl im Zähler ergibt sich dann

$$\Delta \cdot m_{c+1} + (1-\Delta) \cdot m_c = \Delta(m_{c+1} - m_c) + m_c = 1 + m_c.$$

Im Erwartungswert steht also die richtige inkrementierte Zahl im Zähler. Die folgende Proposition belassen wir als Übungsaufgabe.

Proposition 2.27 *Es sei X_n der Wert im Zähler nach n-facher Imkrementierung. Dann gilt $\mathbb{E}X_n = n$ und $\operatorname{Var} X_n = \binom{n}{2}$.*

Algorithmus: SampleCount
Eingabe: Parameter N, Datenstrom $s = (s_1, \ldots, S_N)$ über $\Sigma = \{a_1, \ldots, a_n\}$
Ausgabe: (Schätzung für das) Moment $F_k(s)$

```
Parameter m_1 und m_2 geeignet initialisieren
FOR i := 1 TO m_2
   FOR j := 1 TO m_1
      RANDOM p IN {1, ..., N}
      r := 0
      a := s_p
      FOR q := p TO N
         IF s_q = a
            r := r + 1
      X[i, j] := N · (r^k − (r − 1)^k)
   FOR j := 1 TO m_1
      Y[i] := Y[i] + 1/m_1 · X[i, j]
Gebe den Median von {Y[1], ..., Y[m_2]} aus
```

Abbildung 2.15: Der Algorithmus SampleCount

Damit können wir mit Hilfe der Ungleichung von Chebyshev abschätzen:

$$\text{Prob}[\ |X_n - n| \geq \lambda n\ ] \leq \frac{\text{Var}X_n}{\lambda^2 n^2} \leq \frac{1}{4\lambda^2}.$$

Dabei belassen wir unsere Analyse des Ansatzes von Morris und wenden uns einem zweiten tragfähigerem Ansatz zu.

Zweiter Ansatz (Alon, Matias, Szegedy): *Sampling*

Wir betrachten ein zufälliges Element s_i in $s = (s_1, \ldots, s_N)$ und zählen die Vorkommen von Elementen, die gleich s_i sind, erst ab der Position i. An Hand dieser Anzahl geben wir eine Schätzung für den gesamten Datenstrom ab. Für eine geeignete Fehlerkonzentration müssen jedoch mehrere Variablen verwendet werden.

Zunächst setzen wir voraus, dass die Länge des Datenstroms bekannt ist. Eine Umsetzung des eben beschriebenen Ansatzes ist im Algorithmus SampleCount aus Abbildung 2.15 zu sehen.

Für den Platzbedarf von SampleCount in Abhängigkeit von m_1 und m_2 ergibt sich:

- Berechnung von $X[i, j]$ benötigt $O(\log N + \log n)$ Platz.
- Darstellung von $X[i, j]$ benötigt $O(k \log N)$ Platz.
- Es müssen $m_1 \cdot m_2$ Variablen verwaltet werden.

Insgesamt erfordert dies $O(m_1 \cdot m_2 \cdot (k \cdot \log N + \log n))$ Platz. Wie sind nun aber m_1 und m_2 zu wählen, damit sowohl Platzbedarf als auch Fehlerwahrscheinlichkeit klein gehalten werden kann?

Proposition 2.28 *Für jeden Datenstrom $s = (s_1, s_2, \ldots, s_N)$ sind die Zufallsvariablen $X[i,j]$ aus Zeile 10 des Algorithmus* SampleCount *für alle $1 \leq i \leq m_2$ und $1 \leq j \leq m_1$ unabhängig und identisch verteilt. Außerdem gilt $\mathbb{E}X[i,j] = F_k(s)$.*

Beweis: Unabhängigkeit und identische Verteilung der Variablen $X[i,j]$ ist offensichtlich. Wir berechnen nun den Erwartungswert von $X = X[i,j]$ wie folgt:

$$\mathbb{E}X = \frac{1}{N} \cdot \sum_{i=1}^{n} \sum_{j=1}^{|s|_{a_i}} N \cdot (j^k - (j-1)^k) = \sum_{i=1}^{n} |s|_{a_i}^k = F_k(s).$$

■

Lemma 2.29 *Es sei X die Zufallsvariable, die von* SampleCount *in Zeile 10 für den Datenstrom s berechnet wird. Dann gilt $\mathbb{E}X^2 \leq k \cdot F_1(s) \cdot F_{2k-1}(s)$.*

Beweis: Wir rechnen wie folgt aus:

$$\begin{aligned} \mathbb{E}X^2 &= \frac{1}{N} \cdot \sum_{i=1}^{n} \sum_{j=1}^{|s|_{a_i}} \Big(N \cdot (j^k - (j-1)^k) \Big)^2 \\ &= N \cdot \sum_{i=1}^{n} \sum_{j=1}^{|s|_{a_i}} (j^k - (j-1)^k) \cdot (j^k - (j-1)^k) \end{aligned}$$

An dieser Stelle verwenden wir folgende Ungleichung, die für beliebige $x > y > 0$ gilt:

$$x^k - y^k = (x-y) \cdot (x^{k-1} + x^{k-2}y + \cdots + xy^{k-2} + y^{k-1}) \leq (x-y) \cdot k \cdot x^{k-1}$$

Damit ergibt sich

$$\begin{aligned} \mathbb{E}X^2 &\leq N \cdot \sum_{i=1}^{n} \sum_{j=1}^{|s|_{a_i}} k \cdot j^{k-1} \cdot (j^k - (j-1)^k) \\ &\leq N \cdot \sum_{i=1}^{n} \sum_{j=1}^{|s|_{a_i}} k \cdot j^{2k-1} \\ &= k \cdot N \cdot F_{2k-1}(s) \\ &= k \cdot F_1(s) \cdot F_{2k-1}(s) \end{aligned}$$

■

Lemma 2.30 *Für alle Zahlen $x_1, \ldots, x_n \in \mathbb{R}_+$ gilt*

$$\left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{i=1}^{n} x_i^{2k-1}\right) \leq n^{1-\frac{1}{k}} \cdot \left(\sum_{i=1}^{n} x_i^k\right)^2 .$$

Beweis: Es sei $M = \max_{1 \leq i \leq n} x_i$. Dann gilt $M^k \leq \sum_{i=1}^{n} x_i^k$. Damit ergibt sich

$$\begin{aligned}
\left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{i=1}^{n} x_i^{2k-1}\right) &\leq \left(\sum_{i=1}^{n} x_i\right) \cdot \left(M^{k-1} \cdot \sum_{i=1}^{n} x_i^k\right) \\
&\leq \left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{i=1}^{n} x_i^k\right)^{\frac{k-1}{k}} \cdot \left(\sum_{i=1}^{n} x_i^k\right) \\
&\leq \left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{i=1}^{n} x_i^k\right)^{\frac{2k-1}{k}} \\
&\leq n^{1-\frac{1}{k}} \cdot \left(\sum_{i=1}^{n} x_i^k\right)^{\frac{1}{k}} \cdot \left(\sum_{i=1}^{n} x_i^k\right)^{\frac{2k-1}{k}} \\
&= n^{1-\frac{1}{k}} \cdot \left(\sum_{i=1}^{n} x_i^k\right)^2
\end{aligned}$$

Bei der Abschätzung in der vorletzten Zeile benutzt, dass die Funktion $f(x) = x^k$ konvex ist und somit die Jensensche Ungleichung angewendet werden kann. Wir erhalten $n(\frac{1}{n}\sum_{i=1}^{n} x_i)^k \leq n(\frac{1}{n}\sum_{i=1}^{n} x_i^k)$. Folglich gilt $\sum_{i=1}^{n} x_i \leq n^{1-\frac{1}{k}}(\sum_{i=1}^{n} x_i^k)^{\frac{1}{k}}$. ■

Korollar 2.31 *Es sei $Y[i]$ die Zufallsvariable, die von dem Algorithmus* SampleCount *in Zeile 12 für einen Datenstrom s berechnet wird. Dann gilt*

$$\mathbb{E}Y[i] = F_k(s) \quad \textit{und} \quad \mathrm{Var}Y[i] \leq k \cdot n^{1-\frac{1}{k}} \cdot \frac{F_k(s)^2}{m_1}.$$

Beweis: Mit Hilfe von Proposition 2.28 erhalten wir für den Erwartungswert

$$\mathbb{E}Y[i] = \mathbb{E}\frac{1}{m_1} \cdot \sum_{j=1}^{m_1} X[i,j] = \frac{1}{m_1} \cdot \sum_{j=1}^{m_1} \mathbb{E}X = F_k(s).$$

Für die Varianz ergibt sich

$$\mathrm{Var}Y[i] \quad = \quad \mathrm{Var}\left(\frac{1}{m_1}\sum_{j=1}^{m_1} X[i,j]\right)$$

$$
\begin{aligned}
&= \frac{1}{m_1} \cdot \mathrm{Var} X && \text{(mit Hilfe von Proposition 2.28)} \\
&\leq \frac{1}{m_1} \cdot \mathbb{E} X^2 && \\
&\leq \frac{1}{m_1} \cdot k \cdot F_1(s) \cdot F_{2k-1}(s) && \text{(Lemma 2.29)} \\
&\leq \frac{k \cdot n^{1-\frac{1}{k}}}{m_1} \cdot F_k(s)^2 && \text{(Lemma 2.30)}
\end{aligned}
$$

■

Unser Ziel ist es, die Fehlerwahrscheinlichkeit von SampleCount klein zu halten. Für diese gilt mit Korollar 2.31

$$\mathrm{Prob}[\ |Y[i] - F_k(s)| \geq \delta \cdot F_k(s)\ ] \leq \frac{\mathrm{Var} Y[i]}{\delta^2 F_k^2(s)} \leq \frac{k \cdot n^{1-\frac{1}{k}}}{m_1 \cdot \delta^2}.$$

Damit ist für $m_1 \geq c \cdot \frac{k \cdot n^{1-\frac{1}{k}}}{\delta^2}$ die Fehlerwahrscheinlichkeit höchstens $\frac{1}{c}$.

Theorem 2.32 *Werden für $k \geq 1$, $\delta > 0$ und $\varepsilon > 0$ die Parameter m_1 und m_2 im Algorithm* SampleCount *(siehe Abbildung 2.15) als*

$$m_1 = 8 \cdot \frac{k \cdot n^{1-\frac{1}{k}}}{\delta^2} \qquad \text{und} \qquad m_2 = 2 \cdot \log\left(\frac{1}{\varepsilon}\right)$$

gesetzt, so berechnet SampleCount *für einen Datenstrom s mit bekannter Länge N (über $\Sigma = \{a_1, \ldots, a_n\}$) einen Wert Y, der von $F_k(s)$ mit Wahrscheinlichkeit höchstens ε um mehr als $\delta F_k(s)$ abweicht, und benötigt dafür*

$$O\left(\frac{k^2 \log\left(\frac{1}{\varepsilon}\right)}{\delta^2} \cdot n^{1-\frac{1}{k}} \cdot (\log N + \log n)\right)$$

Speicherplatz.

Beweis: Die Aussagen des Theorems sind klar bis auf die Wahrscheinlichkeitsaussage. Diese folgt aus Fehlerabschätzungen für den Median von $\{Y[1], \ldots, Y[m_2]\}$ mit Hilfe von Chernoff-Schranken (ohne Beweis). ■

Als eine schöne Folgerung aus diesem Theorem erhalten wir eine $O(\sqrt{n}(\log N + \log n))$ Platzschranke für die Bestimmung des zweiten Häufigkeitsmomentes.

Betrachten wir nun den Fall, dass uns die Datenstromlänge N nicht bekannt ist. Wie passen wir dann SampleCount an? Zwei Aspekte müssen wir dabei beachten:

1. Jedes neu gelesene Datenstromelement könnte das letzte Element sein.

Algorithmus: ALON-MATIAS-SZEGEDY
Eingabe: Datenstrom $s = (s_1, \ldots, S_N)$ über $\Sigma = \{a_1, \ldots, a_n\}$
Ausgabe: (Schätzung für das) Moment $F_k(s)$

```
1.  Parameter m_1 und m_2 geeignet initialisieren /* siehe Theorem 2.32 */
2.  FOR i := 1 TO m_2
3.     FOR j := 1 TO m_1
4.        A[i,j] := s_1
5.        R[i,j] := 1
6.  L := 2
7.  WHILE Element s_L existiert
8.     FOR i := 1 TO m_2
9.        FOR j := 1 TO m_1
10.          RANDOM p IN {1,...,L}
11.       IF p = L
12.          A[i,j] := s_p
13.          R[i,j] := 1
14.       ELSE
15.          IF s_L = A[i,j]
16.             R[i,j] := R[i,j] + 1
17.       X[i,j] := L · (R[i,j]^k − (R[i,j] − 1)^k)
18.    L:=L+1
19. Berechne Y[i] als Durchschnitt über X[i,1],...,X[i,m_1]
20. Gebe den Median von {Y[1],...,Y[m_2]} aus
```

Abbildung 2.16: Der Algorithmus von Alon, Matias und Szegedy

2. Alle Positionen im Datenstrom, ab denen wir die Vorkommen von Elementen zählen, müssen mit gleicher Wahrscheinlichkeit gewählt werden können. Diese sinkt jedoch mit zunehmender Datenstromlänge.

Der in Abbildung 2.16 beschrieben Algorithmus von Alon, Matias und Szegedy gibt eine Lösung für diese Probleme an.

Korollar 2.33 *Der Algorithmus von Alon, Matias und Szegedy berechnet das Häufigkeitsmoment $F_k(s)$ für einen Datenstrom $s = (s_1, \ldots, s_N)$, ohne die Gesamtlänge zu kennen, mit den gleichen Fehler- und Platzschranken wie der Algorithmus* SAMPLECOUNT.

Beweis: Die Übertragung der Platzschranken ist klar. Die Fehlerschranken übertragen sich, falls die Wahrscheinlichkeit dafür, dass eine Position p als Beginn für die Zählung im Array R gewählt wird (Zeilen 11–13) gerade $\frac{1}{N}$ ist. Dies ist jedoch wie folgt einzusehen:

1. Die Wahrscheinlichkeit, dass s_p neu gewählt wird, ist $\frac{1}{p}$ (die geschieht nur im Falle $L = p$).

2. Die Wahrscheinlichkeit, dass s_p auch nach Schritt $p+1$ gewählt bleibt, ist

$$\frac{1}{p} \cdot \left(1 - \frac{1}{p+1}\right) = \frac{1}{p} \cdot \frac{p}{p+1} = \frac{1}{p+1}$$

Aus diesen Überlegungen ergibt sich also insgesamt, wenn $X_{p,N}$ für $1 \leq p \leq N$ das zufällige Ereignis repräsentiert, dass eine Zählung bei Position p beginnt:

$$\text{Prob}[X_{p,N}] = \frac{1}{p} \cdot \prod_{i=1}^{N-p} \left(1 - \frac{1}{p+1}\right) = \frac{1}{p} \cdot \prod_{i=1}^{N-p} \frac{p}{p+i} = \frac{1}{p} \cdot \frac{p}{N} = \frac{1}{N}$$

■

Da der Algorithmus von Alon, Matias und Szegedy nicht für $k = 0$ funktioniert, müssen wir uns für diesen Fall etwas anderes überlegen.

Dritter Ansatz (Flajolet und Martin): *Probabilistisches Zählen mit Hashing (zur Bestimmung von $F_0(s)$, d.h. Anzahl unterschiedlicher Elemente in s)*

Wir führen dazu folgende Vorbetrachtung durch: Angenommen wir verfügen über eine reellwertige Zufallsvariable $X : \Omega \to [0,1]$ mit der Eigenschaft $\text{Prob}[X \in [\ell, r)] = r - \ell$, d.h. die der Gleichverteilung auf $[0,1]$ genügt. Weiter angenommen wir haben n unabhängige Kopien $X_1, \ldots, X_n$ von X und wir betrachten die Zufallsvariable

$$Y_n =_{\text{def}} \min\{X_1, \ldots, X_n\}.$$

Welchen Erwartungswert hat Y_n? Eine gute Intuition ist die folgende: Wenn wir n Punkte *gleichzeitig* in das Interval $[0,1]$ werfen würden, dann sollte zu erwarten sein, dass sie das Intervall in $n+1$ gleiche Teile zerlegen. Mithin wäre die minimale Position eines Punktes in $[0,1]$ im Erwartungswert $\frac{1}{n+1}$. Das folgende Lemma liefert den formalen Beweis, dass uns unsere Intuition nicht getäuscht hat.

Lemma 2.34 *Es gilt* $\mathbb{E}Y_n = \dfrac{1}{n+1}$.

Beweis: Für $n = 1$ gilt klarerweise $\mathbb{E}X_1 = \int_0^1 t\ dt = \frac{1}{2}$. Für $n \geq 2$ berechnet sich der Erwartungswert gemäß

$$\mathbb{E}Y_n = \int_0^1 \cdots \int_0^1 \min(t_1, \ldots, t_n)\ dt_1 \ldots dt_n = \int_0^1 \cdots \int_0^1 p_0(\min(t_1, \ldots, t_n))\ dt_1 \ldots dt_n$$

mit $p_0(x) =_{\text{def}} x$ für alle $x \in \mathbb{R}$. Führen wir einen Schritt zur Auflösung der Integralfolge aus, so erhalten wir:

$$\mathbb{E}Y_n \;=\; \int_0^1 \cdots \int_0^1 \left(\int_0^{\min(t_2,\ldots,t_n)} t_1\ dt_1 + \int_{\min(t_2,\ldots,t_n)}^1 \min(t_2, \ldots, t_n)\ dt_1 \right) dt_2 \ldots dt_n$$

$$= \int_0^1 \cdots \int_0^1 \frac{1}{2}\min^2(t_2,\ldots,t_n) + \min(t_2,\ldots,t_n)(1-\min(t_2,\ldots,t_n))\ dt_2\ldots dt_n$$
$$= \int_0^1 \cdots \int_0^1 p_1(\min(t_2,\ldots,t_n))\ dt_2\ldots dt_n$$

mit $p_1(x) =_{\text{def}} \frac{1}{2}x^2 - x(1-x)$ für alle $x \in \mathbb{R}$. Wie leicht zu sehen ist, besteht zwischen p_0 und p_1 der folgende Zusammenhang: $p_1(x) = \int_0^x p_0(t)\ dt\ +\ p_0(x)\cdot(1-x)$ für alle $x \in \mathbb{R}$.

Lösen wir nun alle Integrale der Reihe nach von innen nach außen auf, so erhalten wir eine Folge von Polynomen $p_0, p_1, \ldots, p_n$ mit

$$\mathbb{E}Y_n = \int_0^1 \cdots \int_0^1 p_m(\min(t_{m+1},\ldots,t_n,1))\ dt_m\ldots dt_n \qquad \text{für alle } 1 \le m \le n,$$

die folgender Rekursion genügen:

$$\begin{aligned} p_0(x) &= x && \text{für alle } x \in \mathbb{R} \\ p_m(x) &= \int_0^x p_{m-1}(t)\ dt\ +\ p_{m-1}(x)(1-x) && \text{für } m \ge 1 \text{ und alle } x \in \mathbb{R}. \end{aligned}$$

Mittels vollständiger Induktion über n können wir nun zeigen, dass

$$p_n(x) = \frac{1-(1-x)^{n+1}}{n+1}$$

gilt, woraus sofort $\mathbb{E}Y_n = p_n(1) = \frac{1}{n+1}$ und somit die Behauptung der Proposition folgt.

Der Induktionsanfang $n = 1$ ist offensichtlich: $p_1(x) = x = 1-(1-x)$. Es sei nun $n \ge 2$. Weiterhin gelte die Aussage bereits für p_{n-1}. Dann erhalten wir

$$\begin{aligned} p_n(x) &= \int_0^x p_{n-1}(t)\ dt\ +\ p_{n-1}(x)(1-x) \\ &= \int_0^x \frac{1-(1-t)^n}{n}\ dt\ +\ \frac{1-(1-x)^n}{n}\cdot(1-x) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{(nach Induktionsvoraussetzung)} \\ &= \frac{x}{n} - \left[\frac{1}{n}\cdot\frac{1}{n+1}\cdot(1-t)^{n+1}\cdot(-1)\right]_{t=0}^{t=x} + \frac{1-x}{n} - \frac{(1-x)^{n+1}}{n} \\ &= \frac{1}{n}\cdot\left(\frac{1}{n+1}\cdot(1-x)^{n+1} - \frac{1}{n+1} + 1 - (1-x)^{n+1}\right) \\ &= \frac{1}{n}\cdot\left(\frac{n}{n+1} - (1-x)^{n+1}\cdot\frac{n}{n+1}\right) \\ &= \frac{1-(1-x)^{n+1}}{n+1} \end{aligned}$$

■

Algorithmus: FLAJOLET-MARTIN
Eingabe: Datenstrom $s = (s_1, \ldots, s_N)$ über festem Alphabet $\Sigma = \{a_1, \ldots, a_n\}$
Ausgabe: (Schätzung für das) Moment $F_0(s)$

1. Wähle eine zufällige Hash-Funktion $h : \Sigma \to [0, 1]$, die die Symbole aus Σ gleichverteilt über $[0, 1]$ streut
2. $Y := 1$
3. $i := 1$
4. WHILE Element s_i existiert
5. $\quad Y := \min(Y, h(s_i))$
6. $\quad i := i + 1$
7. Gebe $\frac{1}{Y} - 1$ zurück

Abbildung 2.17: Der Algorithmus von Flajolet und Martin

Mit Hilfe von Lemma 2.34 ergibt sich nun als Idee für einen Algorithmus, dass wir uns jeden im Datenstrom vorkommenden Buchstaben als einen zufällig gewählten Punkt im Intervall $[0, 1]$ vorstellen und deshalb einfach das Minimum aller dieser Punkte bestimmen. Im Erwartungswert ist das Minimum μ gerade $\frac{1}{|s|_0+1}$, woraus sich $\frac{1}{\mu} - 1$ durch Umstellung nach $|s|_0$ als gute Schätzung für $|s|_0$ ergibt.

Der in Abbildung 2.17 angegebene Algorithmus von Flajolet und Martin realisiert diese Idee mit Hilfe idealer Hash-Funktion. Problematisch für die Implementierung dieser algorithmischen Idee sind zwei Aspekte:

- Wie gehen wir mit reellen Zahlen um?
- Wie bestimmen wir eine Familie $\mathcal{H}$ von Hashfunktionen mit der idealen Eigenschaft: $\mathrm{Prob}_{\mathcal{H}}[\ h(a) \text{ liegt im Intervall } [\ell, r)\ ] = \ell - r$ für alle $a \in \Sigma$ und $0 \leq \ell \leq r \leq 1$?

Der erste Aspekt wird von den Schwierigkeiten bei der Beantwortung der zweiten Fragen überlagert, da bisher nicht bekannt ist wie eine solche Funktionenfamilien zu konstruieren wäre.

Auswege bieten sich mit der Verwendung von Hash-Funktionen mit eingeschränkteren Eigenschaften an wie z.B. universelle Hash-Funktionen oder paarweise unabhängige Hash-Funktionen.

Paarweise unabhängige Hash-Funktionen sind wie folgt definiert: Es seien ein Universum $\mathcal{U}$ und eine Hash-Tabelle T gegeben. Eine Familie $\mathcal{H}$ von Hash-Funktionen $h : \mathcal{U} \to T$ heißt *paarweise unabhängig*, falls für alle $x, y \in \mathcal{U}$ mit $x \neq y$ und für alle $a, b \in T$ gilt:

$$\frac{\|\{h \in \mathcal{H} \mid h(x) = a \wedge h(y) = b\}\|}{\|\mathcal{H}\|} = \frac{1}{\|T\|^2}.$$

Die Existenz solcher Familien von Hash-Funktionen ist gesichert (ohne dass wir hier einen Beweis dafür angeben). Wichtige Beipsiele dafür sind Teilfamilien linearer Hash-Funktionen, d.h. Hash-Funktionen h der Form $h(x) = a \cdot x + b \mod p$ für natürliche

Algorithmus: COUNTINGDISTINCTELEMENTS
Eingabe: Datenstrom $s = (s_1, \ldots, S_N)$ über festem Alphabet $\Sigma = \{a_1, \ldots, a_n\}$
Ausgabe: (Schätzung für das) Moment $F_0(s)$

```
Wähle eine zufällige Hash-Funktion h : Σ → {1, ..., n^3} aus einer paarweise
unabhängigen Familie linearer Hash-Funktionen
Initialisiere t geeignet
i := 1
WHILE Element s_i existiert
    a := h(s_i)
    IF a ∉ C
        IF ||C|| ≥ t
            IF a < max C
                Entferne das maximale Element aus C
                Füge a in C ein
        ELSE
            Füge a in C ein
    i := i + 1
Gebe den Wert (t·n^3)/(max C) zurück
```

Abbildung 2.18: Algorithmus zum Schätzen der Anzahl unterschiedlicher Datenstromelemente

Zahlen a, b und p. An dieser Stelle soll uns genügen zu wissen, dass es paarweise unabhängige Familien linearer Hash-Funktionen gibt.

Der Tatsache, dass wir keine idealen Hash-Funktionen verwenden, schulden wir, dass wir nicht den minimalen Hashwert für den Datenstrom zur Schätzung der Anzahl unterschiedlicher Elemente heranziehen, sondern eine gewisse Anzahl kleinster Werte (Ausreißer) ignorieren und das t-te Element in aufsteigender Reihenfolge der Hashwerte in der Schätzung verwenden. Der Algorithmus COUNTINGDISTINCTELEMENTS in Abbildung 2.18 zeigt eine Realisierung.

Wenn wir z.B. AVL-Bäume für die Verwaltung der Menge C verwenden, so kann der Algorithmus mit

- Laufzeit $O((\log t) \cdot (\log n))$ pro Datenstromelement und
- Speicherplatz $O(t \cdot \log n)$

implementiert werden (die Werte a, b und p für die lineare Hash-Funktionen können mit logarithmischen Platz in der Alphabetgröße repräsentiert werden).

Theorem 2.35 *Für alle $1 \geq \varepsilon, \delta > 0$ kann der Parameter t so gewählt werden, dass es eine Implementierung des Algorithmus* CountingDistinctElements *gibt, bei der für einen Datenstrom s der Länge N über einem festen Alphabet Σ der Größe n mit Wahrscheinlichkeit höchstens δ ein Wert Y mit $|Y - F_0(s)| \geq \varepsilon F_0(s)$ ausgegeben und dafür*

$$O\left(\frac{1}{\varepsilon^2} \cdot \log n \cdot \log\left(\frac{1}{\delta}\right)\right)$$

Speicherplatz benötigt wird.

Beweis: Wir setzen $t =_{\text{def}} \lceil \frac{64}{\varepsilon^2} \rceil$. Es sei Y die Ausgabe des Algorithmus CountingDistinctElements für einen Datenstrom s. Wir nehmen $Y \geq (1+\delta)F_0(s)$ für $\delta \leq 1$ an. Damit gibt es also mindestens t Werte in der Menge $\{h(b_1), \dots, h(b_{F_0(s)})\}$, die kleiner oder gleich

$$\frac{t \cdot n^3}{(1+\varepsilon)F_0(s)} \leq \left(1 - \frac{\delta}{2}\right) \cdot \frac{t \cdot n^3}{F_0(s)}$$

sind, wo b_i die im Datenstrom s vorkommenden Symbole sind. Die Abschätzung nach oben folgt wegen $\delta \leq 1$.

Wir betrachten die Indikatorvariablen X_i für diese Ereignisse, d.h. wir definieren:

$$X_i =_{\text{def}} \begin{cases} 1, & \text{falls } h(b_i) \leq \left(1 - \frac{\delta}{2}\right) \cdot \frac{t \cdot n^3}{F_0(s)} \\ 0, & \text{sonst} \end{cases}$$

Folglich erhalten wir

$$\mathbb{E}X_i = \text{Prob}[\ X_i = 1\ ] \leq \left(1 - \frac{\delta}{2}\right) \cdot \frac{t}{F_0(s)} + \frac{1}{n^3} \leq \left(1 - \frac{\delta}{4}\right) \cdot \frac{t}{F_0(s)}.$$

Der Term $\frac{1}{n^3}$ berücksichtigt die Rundungsfehler. Darüber hinaus wissen wir, dass die X_i untereinander unabhängig sind, da die Funktion h aus einer Familie von paarweise unabhängigen Hash-Funktionen gewählt wurde. Dies ist wichtig, wenn wir nun die Zufallsvariable $X =_{\text{def}} \sum_{i=1}^{F_0(s)} X_i$ betrachten. Dann gilt

$$\mathbb{E}X = \sum_{i=1}^{F_0(s)} X_i \leq \left(1 - \frac{\varepsilon}{4}\right) \cdot \frac{t}{F_0(s)} \tag{2.4}$$

und:

$$\text{Var}X \quad = \quad \text{Var} \sum_{i=1}^{F_0(s)} X_i$$

$$
\begin{aligned}
&= \sum_{i=1}^{F_0(s)} \mathrm{Var} X_i && \text{(wegen Unabhängigkeit der } X_i\text{)}\\
&= \sum_{i=1}^{F_0(s)} \left(\mathrm{Prob}[X_i = 1] - \mathrm{Prob}[X_i = 1]^2\right) &&\\
&\leq \sum_{i=1}^{F_0(s)} \mathbb{E} X_i &&\\
&\leq \left(1 - \frac{\varepsilon}{4}\right) t && \text{(wegen Ungleichung (2.4) von oben)}\\
&\leq t && (2.5)
\end{aligned}
$$

Für die Fehlerwahrscheinlichkeit ergibt sich damit:

$$
\begin{aligned}
\mathrm{Prob}[Y \geq (1+\varepsilon)F_0(s)] &= \mathrm{Prob}[X \geq t]\\
&= \mathrm{Prob}[X - \mathbb{E}X \geq t - \mathbb{E}X]\\
&\leq \mathrm{Prob}\left[|X - \mathbb{E}X| \geq \frac{\varepsilon}{4} t\right]\\
&\leq 16 \cdot \frac{\mathrm{Var} X}{\varepsilon^2 t^2}\\
&\leq \frac{16}{\varepsilon^2 t}\\
&\leq \frac{1}{4}
\end{aligned}
$$

Für den Fall $Y \leq (1-\varepsilon)F_0(s)$ erhalten wir eine analoge Abschätzung.

Mittels Bestimmung des Medians der Ausgaben von $O(\log(\frac{1}{\delta}))$ unabhängigen parallelen Ausführungen des Algorithmus erhalten wir eine beliebig genaue Fehlerwahrscheinlichkeit. ■

In der folgende Tabelle fassen wir unsere (und weitere in der Literatur zu findende) Ergebnisse für Platzschranken bei der randomisierten Bestimmung der Häufigkeitsmomente zusammen (für festes ε und δ):

Ordnung	obere Platzschranke	untere Platzschranke
0	$O(\log n)$	$\Omega(\log n)$
1	$O(\log \log N)$	?
2	$O(\sqrt{(}\log N + \log n))$; geht sogar mit $O(\log N + \log n)$	?
k	$O(n^{1-\frac{1}{k}})(\log N + \log n)$	$\Omega(n^{1-\frac{5}{k}})$ für $k \geq 5$
∞	$O(n \log \log N)$	$\Omega(n)$

2.4 Monitore

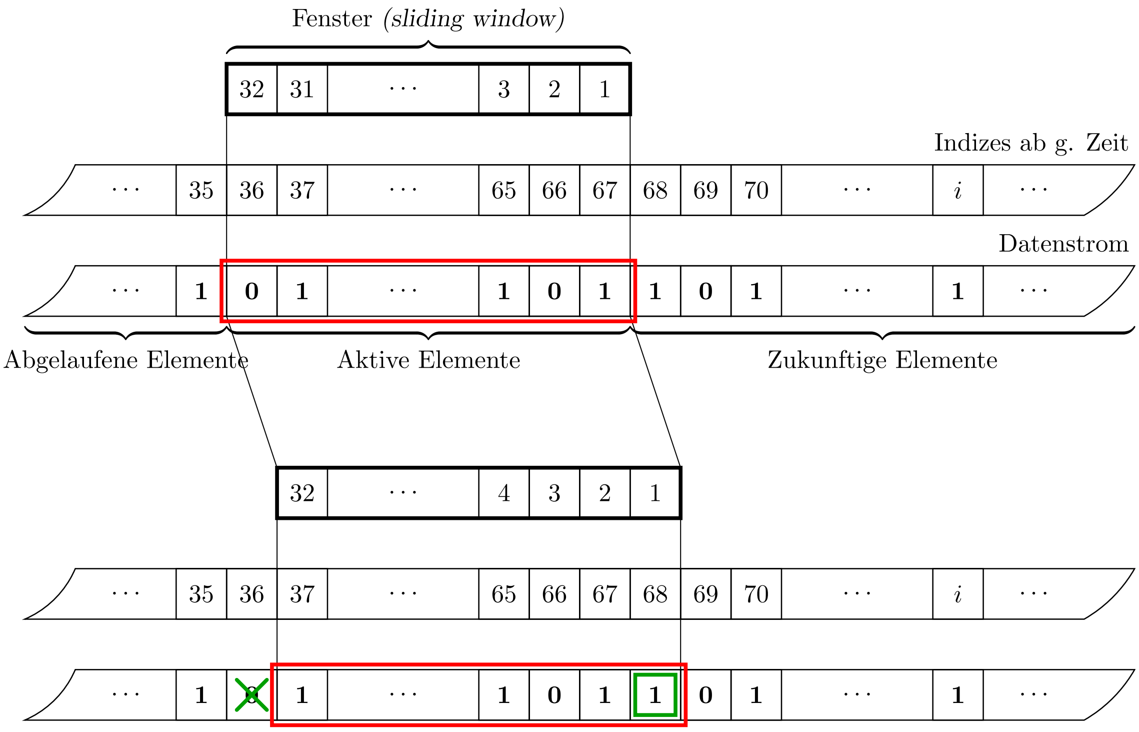
Monitore sind Algorithmen für permanente Anfragen auf (potenziell) unendlich langen Datenströmen.

2.4.1 Verarbeitungsmodelle

Monitore können unterschieden werden auf Basis von:

- Reihenfolge: Es werden nur die N jüngsten angekommenen Datenstromelemente betrachtet.
- Zeitfenster: Es werden nur die in den letzte T Zeiteinheiten angekommenen Datenstromelemente betrachtet

Wir betrachten hier nur Monitore auf Basis der Reihenfolge. Schematisch lässt sich das Verarbeitungsmodells wie folgt darstellen:



2.4.2 Exponenzielle Histogramme

Exponenzielle Histogramme sind eine grundlegende Datenstruktur zur statistischen Auswertung von Datenströmen.

Betrachten dazu das folgende elementare Zählproblem: Ein *Bitstream* s ist ein unendliches Wort über $\{0, 1\}$. Es sei $N \in \mathbb{N}_+$ gegeben. Für einen Bitstream s wollen wir zu jedem Zeitpunkt die Anzahl der Einsen unter den N letzten Datenstromelementen bestimmen; d.h. wir interessieren uns für die Größe $|s_{i+1-N} \dots s_i|_1$ für alle $i \geq N$.

Proposition 2.36

1. *Es gibt einen exakten Algorithmus für das elementare Zählproblem bei Fenstergröße N, der mit $O(N)$ Speicherplatz auskommt.*

2. *Jeder exakte Algorithmus für das elementare Zählproblem bei Fenstergröße N benötigt $\Omega(N)$ Speicherplatz.*

Beweis: Die erste Aussage ist offensichtlich (wenn wir einfach den gesamten Fensterinhalt abspeichern). Die zweite Aussage folgt unmittelbar aus der Behauptung, dass wir alle möglichen 2^N Fensterinhalte unterscheiden müssen, um stets die richtige Antwort geben zu können. Zum Nachweis der Behauptung nehmen wir an, es gibt einen exakten Algorithmus A für das elementare Zählproblem, der aber zwei verschiedene Fensterinhalte $v = v_1 \dots v_N$ und $w = w_1 \dots w_N$ nicht unterscheiden kann. Dabei erfolge die Nummerierung des Fensterinhaltes von links nach rechts. Es sei j die letzte Position, an der sich v und w unterscheiden, d.h. $v_j \neq w_j$ und $v_i = w_i$ für alle $j < i \leq N$. Ohne Beeinträchtigung der Allgemeinheit dürfen wir $v_j = 0$ und $w_j = 1$ annehmen. Es seien nun vom Datenstrom die nächsten $j - 1$ Elemente $u_1, \dots, u_j$ gelesen worden. Da A die Fensterinhalte v und w nicht unterscheiden kann, kann A auch die Fensterinhalte $v_j \dots v_N u_1 \dots u_{j-1}$ und $w_j \dots w_N u_1 \dots u_{j-1}$ nicht unterscheiden und würde für beide Inhalte die gleiche Anzahl von Einsen ausgeben. Es gilt aber

$$
\begin{aligned}
|v_j v_{j+1} \dots v_N u_1 \dots u_{j-1}|_1 &= |v_{j+1} \dots v_N u_1 \dots u_{j-1}|_1 \\
&= |w_{j+1} \dots w_N u_1 \dots u_{j-1}|_1 \\
&< |w_j w_{j+1} \dots w_N u_1 \dots w_{j-1}|_1
\end{aligned}
$$

Dies ist jedoch ein Widerspruch zur Korrektheit des Algorithmus. ∎

Das Ziel im weiteren Verlauf besteht nun darin, das elementare Zählproblem bis auf den Faktor $1 \pm \varepsilon$ in einem Monitor der Größe N mit $o(N)$ Speicherplatz zu approximieren, d.h. zu jedem Zeitpunkt $i \geq N$ können wir eine Schätzung D der Anzahl der Einsen C abgeben mit $(1 - \varepsilon)C \leq D \leq (1 + \varepsilon)C$.

Um dieses Ziel zu erreichen, verwenden wir Histogramme bestehend aus den Klassen $C_1, \dots, C_m$, die die ankommenden Einsen in einem bestimmten Zeitintervall bilanzieren. Dabei haben wir für die zuletzt ankommenden Einsen detailliertere Informationen als über weiter in der Vergangenheit angekommenen Einsen.

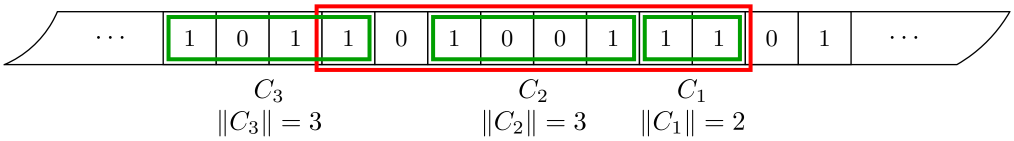
Für eine Histogrammklasse C_i merken wir uns

- eine Zeitmarke t_i (die Fensterposition der jüngsten Eins der Histogrammklasse) und

- die Anzahl n_i der Einsen in C_i, d.h. $n_i = \|C_i\|$.

In unserem einführende Schema repräsentiert die Klasse mit Zeitmarke 2 und Größe 2 die beiden Einsen an den Fensterpositionen 2 und 4.

Falls die Zeitmarke einer Histogrammklasse den Wert $N+1$ erreicht, so verwerfen wir die gesamte Klasse. Dabei gilt natürlich, dass es zu jedem (absoluten) Zeitpunkt höchstens eine Klasse mit abgelaufenen Elementen geben kann. Im folgenden Beispiel wäre dies die Klasse C_3.



Was ist nun eine gute Schätzung für die Anzahl der Einsen in einer Klasse C_i mit abgelaufenen Elementen? Die Antwort ist einleuchtend: $\frac{1}{2}n_i$. Mit dieser Antwort ergibt sich unmittelbar folgende Aussage.

Proposition 2.37 *Es seien $C_1, \ldots, C_m$ Histogrammklassen, die den Fensterinhalt eines Monitors zu einem beliebigen Zeitpunkt repräsentieren. Dabei gelte $t_1 < \cdots < t_m$. Dann ist der absolute Fehler der Schätzung $\frac{1}{2}n_m + \sum_{i=1}^{m-1} n_i$ für die Anzahl der Einsen im Fenster höchstens $\frac{1}{2}n_m$.*

Die Güte der Approximation, d.h. der relative Fehler, wird nun über die Anzahl verwendeter Histogrammklassen und über ihre Größe eingestellt.

Es sei $\varepsilon > 0$. Wir definiere $k =_{\text{def}} \lceil \frac{1}{\varepsilon} \rceil$. Es seien $C_1, \ldots, C_m$ die Histogrammklassen (so nummeriert, dass $t_1 < \cdots < t_m$ gilt). Es sei D die exakte Anzahl der Einsen im Fenster. Wir betrachten nur den Fall $D > 0$ (die Behandlung des Falles $D = 0$ ergibt sicht durch ein einfache Abänderung unseres Algorithmus). Dann gilt $D \geq 1 + \sum_{j=1}^{m-1} n_j$. Mit Hilfe von Proposition 2.37 ergibt sich daraus für den relativen Fehler δ:

$$\delta \leq \frac{n_m}{2 \cdot D} \leq \frac{n_m}{2 \cdot (1 + \sum_{j=1}^{m-1} n_j)}$$

Wenn wir also von unserem Algorithmus zur Aktualisierung der Histogrammklassen verlangen, dass er der Bedingung

Invariante. *Zu jedem Zeitpunkt erfüllen die Histogrammklassen $C_1, \ldots, C_m$ die Eigenschaft*

$$\frac{n_j}{2 \cdot (1 + \sum_{i=1}^{j-1} n_i)} \leq \frac{1}{k} \qquad \textit{für alle } 1 \leq j \leq m.$$

Algorithmus: BasicCounting
Eingabe: Bitstream $s \in \{0,1\}^\infty$
Aufgabe: Gebe $(1 \pm \varepsilon)$-Schätzung für $|s_{i+1-N} \dots s_i|_1$ für alle $i \geq N$ ab

WHILE Datenstromelement s_i existiert
    Inkrementiere Zeitmarken aller Histogrammklassen
    IF Zeitmarke der ältesten Histogrammklasse $\geq N+1$
        Entferne älteste Histogrammklasse C_m
    IF $s_i = 1$
        Initialisiere neue Klasse mit Zeitmarke 1 und Größe 1
    WHILE es gibt $k+2$ Histogrammklasse gleicher Größe
        Verschmelze die ältesten beiden Klassen zu einer neuen Klasse doppelter Größe mit dem Minimum der beiden Zeitmarken als neue Zeitmarke
    IF Anzahl m vorhandener Histogrammklassen höchstens $k+1$
        Gebe m zurück
    ELSE
        Gebe $\frac{1}{2}n_m + \sum_{i=1}^{m-1} n_i$ zurück
    $i := i+1$

Abbildung 2.19: Der Algorithmus BasicCounting

genügt, so bewegt sich der relative Fehler unserer Schätzung gerade im vorgegebenen Gütebereich $\pm\varepsilon$.

Aus gewissen technischen Gründen wird es uns nicht möglich sein, die Invariante wie gefordert für alle $1 \leq j \leq m$ zu erfüllen sondern lediglich für alle $k+2 \leq j \leq m$. Dies ist jedoch behebbar, wenn wir für den Fall $m \leq k+1$ eine präzisere Schätzung verwenden. Für $m \geq k+2$ garantiert uns die Invariante die Korrektheit unserer Approximation.

Der in Abbildung 2.19 angegebene Algorithmus zeigt eine Umsetzung unserer Anforderungen. Dabei wird vorausgesetzt, dass wir nur Klassengrößen der Form 2^r betrachten, was aber durch die verwendeten Verschmelzungsregeln impliziert ist.

Beispiel. Wir wollen uns die Wirkung des Algorithmus BasicCounting verdeutlichen. Dazu nehmen wir die Parameter $\varepsilon = \frac{1}{2}$ und $k = 2$ an.

Schritt	Größen vorhandener Histogrammklassen	ausgeführte Aktion
0	32, 32, 16, 16, 8, 8, 8, 4, 4, 4, 2, 2, 1, 1,	
1	32, 32, 16, 16, 8, 8, 8, 4, 4, 4, 2, 2, 1, 1, **1**	neue 1 angekommen
2	32, 32, 16, 16, 8, 8, 8, 4, 4, 4, 2, 2, 1, 1, 1, **1**	neue 1 angekommen
3	32, 32, 16, 16, 8, 8, 8, 4, 4, 4, 2, 2, **2**, 1, 1,	1-Klassen verschmolzen
4	32, 32, 16, 16, 8, 8, 8, 4, 4, 4, 2, 2, 2, 1, 1, **1**	neue 1 angekommen
5	32, 32, 16, 16, 8, 8, 8, 4, 4, 4, 2, 2, 2, 1, 1, 1, **1**	neue 1 angekommen
6	32, 32, 16, 16, 8, 8, 8, 4, 4, 4, 2, 2, 2, **2**, 1, 1	1-Klassen verschmolzen
7	32, 32, 16, 16, 8, 8, 8, 4, 4, 4, **4**, 2, 2, 1, 1	2-Klassen verschmolzen
8	32, 32, 16, 16, 8, 8, 8, **8**, 4, 4, 2, 2, 1, 1	4-Klassen verschmolzen
9	32, 32, 16, 16, **16**, 8, 8, 4, 4, 2, 2, 1, 1	8-Klassen verschmolzen

Proposition 2.38 *Der Algorithmus* BasicCounting *erfüllt zu jedem Zeitpunkt ab N folgende Eigenschaften, wobei $C_1, \ldots, C_m$ die vorhandenen Histogrammklassen sind mit $t_1 < \cdots < t_m$.*

1. *Die Klassengrößen sind monoton wachsend, d.h. $n_1 \leq \cdots \leq n_m$.*
2. *Für alle Klassen C_j mit $n_j < n_m$ gibt es mindestens k und höchstens $k+1$ Klassen der Größe n_j.*
3. *Für alle $1 \leq j \leq m$ gilt $n_j \in \{1, 2, 4, \ldots, 2^{m'}\}$ mit $m' \leq m$ und $m' \leq \log\left(\frac{N}{k} + 1\right)$.*
4. *Die Invariante ist für alle $j \geq k+2$ erfüllt.*
5. *Ist $m \leq k+1$, so gilt $m = D$.*

Beweis: Wir schauen uns die Aussagen der Reihe nach einzeln an.

1. Die Richtigkeit der Aussage ist offensichtlich.
2. Diese Aussage ist eine einfache Folgerung, die sich aus der iterierten Anwendung der Verschmelzungsregel in Zeile 8 ergibt. Hierbei ist zu beachten, dass Histogrammklassen mit mehr als zwei Elementen ausschließlich durch diese Verschmelzungsregel entstehen können.
3. Offensichtlich gilt $n_j \in \{2^r \mid r \in \mathbb{N}\}$ (wegen Zeile 8). Wie mit Hilfe der zweiten Aussage dieser Proposition leicht einzusehen ist, gilt für C_j die Abschätzung $n_j \leq 2^{\lceil \frac{j}{k} \rceil} \leq 2^j$. Mithin ist $m' \leq \lceil \frac{m}{k} \rceil \leq m$. Außerdem gilt $N \geq \sum_{j=0}^{m'-1} k \cdot 2^j$ und folglich $m' \leq \log\left(\frac{N}{k} + 1\right)$.
4. Für $j \geq k+2$ sei $n_j = 2^r$ die Größe einer Histogrammklasse C_j. Dann gibt es mindestens k Klassen der Größe r' mit $0 \leq r' \leq r-1$ (mit kleineren Zeitmarken als die Zeitmarke von C_j). Damit gilt
$$1 + \sum_{i=1}^{j-1} n_i \geq 1 + k \cdot \sum_{i=0}^{r-1} 2^i = 1 + k \cdot (2^r - 1) = 1 + k \cdot (n_j - 1) \geq \frac{k}{2} \cdot n_j,$$
wobei die letzte Abschätzung wegen $n_j \geq 2$ für $j \geq k+2$ richtig ist. Die Invariante ergibt sich nun durch Umstellung nach den entsprechenden Variablen.
5. Ist $m \leq k+1$, so sind alle Histogrammklassen Einermengen. Mithin ist m die exakte Anzahl der Einsen im Fenster.

Damit ist die Proposition bewiesen. ■

Theorem 2.39 *Es seien* $\varepsilon > 0$ *und* $k = \lceil \frac{1}{\varepsilon} \rceil$. *Der Algorithmus* BasicCounting *gibt zu jedem Zeitpunkt ab* N *für einen Bitstream eine bis auf den Faktor* $1 \pm \varepsilon$ *genaue Schätzung der Anzahl der Einsen unter den* N *zuletzt gesehenen Bitstream-Elementen an und kann mit*

1. *höchstens* $(k+1) \cdot \left(\log\left(\frac{N}{k}+1\right)+1\right)$ *Histogrammklassen,*
2. *höchstens* $\log N + \log\log N$ *Bits pro Histogrammklasse und*
3. $O(1)$ *amortisierter Verarbeitungszeit pro Bitstream-Element*

implementiert werden. Damit ergibt sich ein Speicherplatzbedarf von

$$O\left(\frac{1}{\varepsilon} \cdot \log^2 N\right).$$

Beweis: Die Korrektheit des Algorithmus folgt aus Proposition 2.38(4) und 2.38(5). Die Anzahl der Histogrammklassen ergibt sich aus Proposition 2.38(2) und 2.38(3). Pro Histogrammklasse werden $\log N$ Bits für die Zeitmarke sowie $\log\log N$ Bits für die unterschiedlichen Größen (beachte: Größen sind immer Zweierpotenzen) benötigt. Die amortisierte Zeitkomplexität überträgt sich von der amortisierten Analyse beim Binärzähler. ■

Bemerkungen.

1. Es gibt einen Algorithmus (mit zugehöriger Datenstruktur), mit dem das elementare Zählproblem in $O(\frac{1}{\varepsilon}\log^2 \varepsilon N)$ Platz und $O(1)$ Schritten im schlechtesten Fall pro Bitstream-Element (approximativ) gelöst werden kann (Algorithmus von Gibbons und Tirthapura).
2. Jeder Algorithmus, der das elementare Zählproblem mit Faktor $1 \pm \varepsilon$ approximiert, benötigt $\Omega(\frac{1}{\varepsilon}\log^2 \varepsilon N)$ Platz. Diese Schranke gilt auch für randomisierte Algorithmen.

Wir betrachten nun folgendes Summenproblem: Für einen R-Stream s (d.h. ein Wort $s \in \{0,1,\ldots,R\}^\infty$) gebe zu jedem Zeitpunkt $i \geq N$ eine $(1\pm\varepsilon)$-Schätzung für die Summe der letzten N Elemente an (d.h. für $\sum_{j=i+1-N}^{i} s_i$). Für $R=1$ entspricht dies gerade dem elementaren Zählproblem.

Als Idee für die Behandlung dieses Problemes bietet sich folgende Reduzierung an: Ersetze ein Element $s_i \in \{1,\ldots,R\}$ durch s_i Einsen und wende BasicCounting auf den resultierenden Bitstream an.

Korollar 2.40 *Es sei* $\varepsilon > 0$. *Weiterhin seien* $R \geq 1$ *und* $N \geq 1$ *gegeben mit* $R = O(2^N)$. *Dann gibt es einen Algorithmus, der das Summenproblem für R-Streams mit*

$$O\left(\frac{1}{\varepsilon} \cdot (\log \varepsilon N + \log R) \cdot \log N\right)$$

Speicherplatz löst.

Beweis: Ein Fenster der Größe N für R-Streams entspricht einem Fenster der Größe $R \cdot N$ für die Repräsentierung von s durch einen Bitstream s'. Nach Theorem 2.39 benötigt BasicCounting dafür (mit $k = \lceil \frac{1}{\varepsilon} \rceil$)

- $(k+1) \cdot \left(\log\left(\frac{R \cdot N}{k} + 1\right) + 1\right)$ Histogrammklassen und
- $\log N + \log\log R \cdot N$ Bits pro Klasse.

Zusammen ergibt dies

$$
\begin{aligned}
(k+1) \cdot \left(\log\left(\frac{R \cdot N}{k} + 1\right) + 1\right) \cdot (\log N + \log(\log N + \log R)) \\
= O\left(\frac{1}{\varepsilon} \cdot (\log \varepsilon \cdot N + \log R) \cdot \log N\right)
\end{aligned}
$$

als Platzschranke. ∎

Bemerkung: Jeder Algorithmus (auch randomisiert) für das Summenproblem benötigt $\Omega\left(\frac{1}{\varepsilon} \cdot (\log \varepsilon \cdot N + \log R) \cdot \log N\right)$ Speicherplatz.

2.4.3 Varianzenapproximation

Wir betrachten das folgende Varianzenproblem auf Monitoren: Für einen R-Stream s gebe zu jedem Zeitpunkt $i \geq N$ eine bis auf Faktor $1 \pm \varepsilon$ genaue Schätzung für die Varianz der letzten N Elemente an, d.h. für

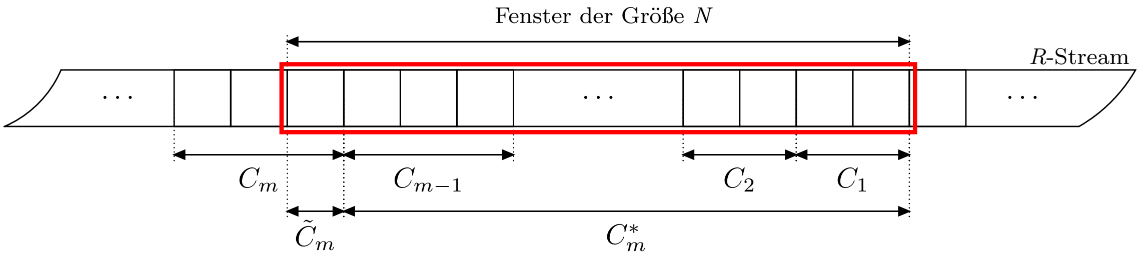
$$\mathrm{Var}_i(s) = \sum_{j=i+1-N}^{i} (s_j - \mu_i)^2,$$

wobei μ_i der Mittelwert der N zum Zeitpunkt i zuletzt gesehenen Elemente ist, also $\mu_i = \frac{1}{N} \sum_{j=i+1-N}^{i} s_j$.

Wir müssen hier beachten, dass die Varianz eigentlich als $\frac{1}{N} \cdot \sum_{j=i+1-N}^{i} (s_j - \mu_i)^2$ definiert ist. Unsere Definition ist aber sowohl aus nummerischer als auch algorithmischer Sicht besser behandelbar. Der relative Fehler bleibt klarerweise erhalten unabhängig davon, welche Definition wir zu Grunde legen.

Lässt der Kontext, d.h. der R-Stream s und der Zeitpunkt i, keine Unklarheiten zu, so lassen wir die Parameter s und i weg, und schreiben nur Var für die Varianz.

Wir werden auch hier wieder exponenzielle Histogramme verwenden, diesmal eine Variante, die sich schematisch wie folgt darstellen lässt:



Die Klassen C_j enthalten alle R-Stream-Elemente mit Zeitindizes aus einem bestimmten Intervall und werden repräsentiert durch:

- Zeitmarke t_j (Zeitindex des jüngsten Elementes in C_j)
- Größe n_j von C_j, d.h. $n_j = \|C_j\|$
- Mittelwert μ_j der Elemente in C_j
- Varianz V_j der Elemente in C_j

Diese Werte werden für alle Klassen immer aktualisiert.

Darüber hinaus verwenden wir auch noch sogenannte *Suffixklassen* C_j^*. Die Suffixklasse C_j^* besteht aus allen Elementen, die nach den Elementen in C_j angekommen sind, d.h. es gilt $C_j^* = \bigcup_{i=1}^{j-1} C_i$. Die Repräsentierungen der Suffixklassen werden bis auf diejenige von C_m^* nicht aktualisiert. (Sie können wohl aber aus den Repräsentierungen aller C_i berechnet werden.)

Im algorithmischen Ablauf werden wir drei Regeln verwenden:

- eine Verschmelzungsregel
- eine Schätzregel
- eine Invariante

Wir wenden uns diesen Regeln der Reihe nach zu.

Verschmelzungsregel. Die Verschmelzungsregel gibt an, wie benachbarte Histogrammklassen zu einer neuen Klasse zusammengefasst werden. Die Aufgabe besteht insbesondere darin, die Repräsentierung für die neue Klasse aus den Repräsentierungen der beiden Klassen zu berechnen. Dazu seien C_i und C_j zwei benachbarte (disjunkte) Histogrammklassen. Dann definieren wir die Repräsentierung für die Klasse $C_{i,j} = C_i \cup C_j$ wie folgt:

- Zeitmarke $t_{i,j} =_{\text{def}} \min(t_i, t_j)$

- Größe $n_{i,j} =_{\text{def}} n_i + n_j$
- Mittelwert $\mu_{i,j} =_{\text{def}} \dfrac{\mu_i \cdot n_i + \mu_j \cdot n_j}{n_i + n_j}$
- Varianz $V_{i,j} =_{\text{def}} V_i + V_j + \dfrac{n_i \cdot n_j}{n_i + n_j} \cdot (\mu_i - \mu_j)^2$

Proposition 2.41 *Die Verschmelzungsregel ist korrekt.*

Beweis: Die Korrektheit der Regeln für Zeitmarke, Größe und Mittelwert sind offensichtlich. Für die Varianz definieren wir Hilfsvariablen $\delta_i =_{\text{def}} \mu_i - \mu_{i,j}$ und $\delta_j =_{\text{def}} \mu_j - \mu_{i,j}$. Damit erhalten wir:

$$
\begin{aligned}
V_{i,j} &= \sum_{x \in C_{i,j}} (x - \mu_{i,j})^2 \\
&= \sum_{x \in C_i} (x - \mu_i + \delta_i)^2 + \sum_{x \in C_j} (x - \mu_j + \delta_j)^2 \\
&= \sum_{x \in C_i} \left((x - \mu_i)^2 + 2\delta_i (x - \mu_i) + \delta_i^2\right) + \sum_{x \in C_j} \left((x - \mu_j)^2 + 2\delta_j (x - \mu_j) + \delta_j^2\right) \\
&= V_i + V_j + \delta_i^2 \cdot n_i + \delta_j^2 \cdot n_j + 2\delta_i \underbrace{\left(\sum_{x \in C_i} x - \sum_{x \in C_i} \mu_i\right)}_{=0} + 2\delta_j \underbrace{\left(\sum_{x \in C_j} x - \sum_{x \in C_j} \mu_j\right)}_{=0} \\
&= V_i + V_j + \delta_i^2 \cdot n_i + \delta_j^2 \cdot n_j \qquad (2.6)
\end{aligned}
$$

Für die Hilfsvariable δ_i ergibt sich

$$
\delta_i = \mu_i - \mu_{i,j} = \mu_i - \frac{\mu_i \cdot n_i + \mu_j \cdot n_j}{n_i + n_j} = \frac{\mu_i \cdot n_i + \mu_i \cdot n_j - \mu_i \cdot n_i - \mu_j \cdot n_j}{n_i + n_j} = n_j \cdot \frac{\mu_i - \mu_j}{n_i + n_j}
$$

und analog $\delta_j = n_i \cdot \dfrac{\mu_j - \mu_i}{n_i + n_j}$. Damit lässt sich die Gleichung (2.6) wie folgt weiter umformen:

$$
\begin{aligned}
V_{i,j} &= V_i + V_j + n_i \cdot \left(n_j \cdot \frac{\mu_i - \mu_j}{n_i + n_j}\right)^2 + n_j \cdot \left(n_i \cdot \frac{\mu_j - \mu_i}{n_i + n_j}\right)^2 \\
&= V_i + V_j + \frac{n_i \cdot n_j^2 + n_j \cdot n_i^2}{(n_i + n_j)^2} \cdot (\mu_i - \mu_j)^2 \\
&= V_i + V_j + \frac{n_i \cdot n_j}{n_i + n_j} \cdot (\mu_i - \mu_j)^2
\end{aligned}
$$

■

Bemerkung: Die Verschmelzungsregel kann auch umgekehrt angewendet werden, d.h. für Histogrammklassen A und B mit $A \subseteq B$ kann aus den Repräsentierungen von A und B auch eine Repräsentierung für $B \setminus A$ korrekt berechnet werden.

Für eine Histogrammklasse A bezeichnen n_A, μ_A und V_A Größe, Mittelwert und Varianz (in obigem Sinne) von A.

Korollar 2.42 *Es seien A und B Histogrammklassen mit $A \subseteq B$. Dann gilt $V_A \leq V_B$.*

Beweis: Wegen $B = A \cup (B \setminus A)$ gilt nach der Verschmelzungsregel

$$V_B = V_A + V_{B\setminus A} + \frac{n_A \cdot n_{B\setminus A}}{n_B} \cdot (\mu_A - \mu_{B\setminus A})^2.$$

Mithin gilt $V_B \geq V_A$. ■

Schätzregel. Es seien $C_1, \ldots, C_m$ die Histogrammklassen (zum Zeitpunkt $i \geq N$) mit $t_1 < \cdots < t_m$. Es sei $\tilde{C}_m$ der Teil von C_m, der die noch nicht abgelaufenen Elemente von C_m enthält. Da eine Repräsentierung von $\tilde{C}_m$ nicht aktuell gehalten wird, müssen wir sie schätzen. Dazu verwenden wir folgende Schätzparameter:

- $\tilde{t}_m^{\text{EST}} =_{\text{def}} t_m$ (beachte: $\tilde{t}_m^{\text{EST}} = \tilde{t}_m$)
- $\tilde{n}_m^{\text{EST}} =_{\text{def}} N + 1 - n_m$ (beachte: $\tilde{n}_m^{\text{EST}} = \tilde{n}_m$)
- $\tilde{\mu}_m^{\text{EST}} =_{\text{def}} \mu_m$
- $\tilde{V}_m^{\text{EST}} =_{\text{def}} \frac{1}{2} \cdot V_m$

Nach Proposition 2.41 erhalten wir den exakten Wert der Varianz durch Verschmelzung der benachbarten Histogramme $\tilde{C}_m$ und C_m^*. Deshalb verwenden wir als Schätzung Var^{EST} für die Varianz:

$$\text{Var}^{\text{EST}} =_{\text{def}} \tilde{V}_m^{\text{EST}} + V_m^* + \frac{\tilde{n}_m^{\text{EST}} \cdot n_m^*}{\tilde{n}_m^{\text{EST}} + n_m^*} \cdot (\tilde{\mu}_m^{\text{EST}} - \mu_m^*)^2$$

Wir müsen uns nun mit der Frage auseinander setzen: Wie gut approximiert Var^{EST} die Varianz?

Lemma 2.43 *Für alle $z \in \mathbb{R}$ und alle endlichen Mengen $B \subseteq \mathbb{R}$ mit dem Mittelwert μ gilt*

$$\sum_{x\in B} (x-z)^2 = \sum_{x \in B} (x-\mu)^2 \quad + \|B\| \cdot (z-\mu)^2.$$

Beweis:

$$\begin{aligned} \sum_{x\in B} (x-\mu)^2 \quad + \|B\| \cdot (z-\mu)^2 &= \sum_{x\in B} \left((x-\mu)^2 + (z-\mu)^2\right) \\ &= \sum_{x\in B} \left(x^2 + 2\mu^2 + z^2 - 2x\mu - 2z\mu\right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{x\in B}\left(x^2 + 2\mu(\mu - x) - 2z\mu + z^2\right) \\
&= \sum_{x\in B}\left(x^2 - 2z\mu + z^2\right) \quad + 2\mu \cdot \underbrace{\sum_{x\in B}(\mu - x)}_{=0} \\
&= \sum_{x\in B}\left(x^2 - 2zx + z^2\right) \\
&= \sum_{x\in B}(x-z)^2
\end{aligned}
$$

■

Lemma 2.44 *Es seien $\varepsilon > 0$ und $C > 0$. Dann gilt*

$$\left|\mathrm{Var} - \mathrm{Var}^{\,\mathrm{EST}}\right| \le \frac{3}{2} \cdot V_m + \max\left(\frac{2C}{3} \cdot V_m, \frac{2\varepsilon}{C} \cdot \mathrm{Var}\right).$$

Beweis: Wir definieren eine Hilfsvariable $\delta =_{\mathrm{def}} \mu_m - \tilde{\mu}_m$. Nach Deifnition von $\tilde{\mu}_m^{\mathrm{EST}}$ gilt $\delta = \tilde{\mu}_m^{\mathrm{EST}} - \tilde{\mu}_m$. Wir rechnen für den absoluten Fehler aus:

$$
\begin{aligned}
\left|\mathrm{Var} - \mathrm{Var}^{\,\mathrm{EST}}\right| &= \left|\left(\tilde{V}_m + V_m^* + \frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot (\tilde{\mu}_m - \mu_m^*)^2\right) - \right. \\
&\qquad \left. - \left(\tilde{V}_m^{\mathrm{EST}} + V_m^* + \frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot (\tilde{\mu}_m^{\mathrm{EST}} - \mu_m^*)^2\right)\right| \\
&= \left|\tilde{V}_m - \tilde{V}_m^{\mathrm{EST}} + \frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot \left((\tilde{\mu}_m - \mu_m^*)^2 - (\underbrace{\tilde{\mu}_m^{\mathrm{EST}} - \mu_m^*}_{=\tilde{\mu}_m - \mu_m^* + \delta})^2\right)\right| \\
&= \left|\tilde{V}_m - \tilde{V}_m^{\mathrm{EST}} + \frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot \left(2\delta(\tilde{\mu}_m - \mu_m^*) + \delta^2\right)\right| \\
&\le \underbrace{\left|\tilde{V}_m - \frac{1}{2} \cdot V_m\right|}_{=:Q_1} + \underbrace{\left|\frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot \delta^2\right|}_{=:Q_2} + \underbrace{\left|\frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot 2\delta(\tilde{\mu}_m - \mu_m^*)\right|}_{=:Q_3}
\end{aligned}
$$

Wir schätzen die Ausdrücke Q_1, Q_2 und Q_3 einzeln ab.

(Q_1) Da $\tilde{C}_m \subseteq C_m$ gilt, folgt aus Korollar 2.42: $\tilde{V}_m \le V_m$. Somit (und wegen $\tilde{V}_m \ge 0$) erhalten wir $Q_1 \le \frac{1}{2} \cdot V_m$.

(Q_2) Wir rechnen aus:

$$Q_2 = \frac{n_m^*}{\tilde{n}_m + n_m^*} \cdot \tilde{n}_m(\mu_m - \tilde{\mu}_m)^2 \qquad \text{(nach Definition von } \delta)$$

$$
\begin{aligned}
&\leq \frac{n_m^*}{\tilde{n}_m + n_m^*} \cdot \sum_{x \in \tilde{C}_m} (x-\mu)^2 && \text{(nach Lemma 2.43)} \\
&\leq \frac{n_m^*}{\tilde{n}_m + n_m^*} \cdot \sum_{x \in C_m} (x-\mu)^2 && \text{(wegen } \tilde{C}_m \subseteq C_m\text{)} \\
&= \underbrace{\frac{n_m^*}{\tilde{n}_m + n_m^*}}_{\leq 1} \cdot V_m \\
&\leq V_m
\end{aligned}
$$

(Q_3) Wir unterscheiden zwei Fälle.

1. Es gelte $|\tilde{\mu}_m - \mu_m^*| \leq \frac{C}{\varepsilon} \cdot \delta$. Dann gilt

$$
Q_3 = \frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot 2\delta |\tilde{\mu}_m - \mu_m^*| \leq \frac{2C}{\varepsilon} \cdot \underbrace{\frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot \delta^2}_{=Q_2} \leq \frac{2C}{\varepsilon} \cdot V_m.
$$

2. Es gelte $|\tilde{\mu}_m - \mu_m^*| > \frac{C}{\varepsilon} \cdot \delta$. An dieser Stelle verwenden wir die Abschätzung $\mathrm{Var} \geq \frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot (\tilde{\mu}_m - \mu_m^*)^2$, dies sich unmittelbar aus der Verschmelzungsregel angewendet auf $\tilde{C}_m$ und C_m^* ergibt. Damit erhalten wir:

$$
\frac{Q_3}{\mathrm{Var}} \leq \frac{\frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot 2\delta |\tilde{\mu}_m - \mu_m^*|}{\frac{\tilde{n}_m \cdot n_m^*}{\tilde{n}_m + n_m^*} \cdot (\tilde{\mu}_m - \mu_m^*)^2} = \frac{2\delta}{|\tilde{\mu}_m - \mu_m^*|} \leq \frac{2\varepsilon}{C}
$$

Insgesamt erhalten wir also

$$
\left|\mathrm{Var} - \mathrm{Var}^{\mathrm{EST}}\right| \leq \frac{1}{2} \cdot V_m + V_m + \max\left(\frac{2C}{\varepsilon} \cdot V_m, \frac{2\varepsilon}{C} \cdot \mathrm{Var}\right).
$$

■

Invariante. Wir müssen nun noch eine Invariante für die Histogrammklassen entwickeln, die uns einen relativen Fehler $\varepsilon > 0$ für die Schätzung garantiert:

Invariante. *Für alle* $1 \leq j \leq m$ *gilt* $\frac{9}{\varepsilon^2} \dot{V}_j \leq V_j^*$.

Proposition 2.45 *Es sei* $\varepsilon > 0$ *und* $C = 3$. *Ist die Invariante erfüllt, so gilt*

$$
(1-\varepsilon) \cdot \mathrm{Var} \leq \mathrm{Var}^{\mathrm{EST}} \leq (1+\varepsilon) \cdot \mathrm{Var}.
$$

Algorithmus: VarianceMonitor
Eingabe: R-Stream $s \in \{0,1\}^\infty$
Aufgabe: Gebe $(1 \pm \varepsilon)$-Schätzung für $\mathrm{Var}_i(s)$ für alle $i \geq N$ ab

1. WHILE R-Stream-Element s_i existiert
2.   Inkrementiere Zeitmarken aller Histogrammklassen
3.   IF $s_i = \mu_1$
4. $t_1 := 1$
5. $n_1 := n_1 + 1$
6.   ELSE
7.   Initialisiere neue Klasse mit Zeitmarke 1, Größe 1, Mittelwert s_i und Varianz 0
8.   IF Zeitmarke der ältesten Klasse $\geq N + 1$
9.   Entferne älteste Klasse C_m und berechne Repräsentierung für C^*_{m-1} aus C^*_m und C_{m-1} mit Hilfe der Verschmelzungsregel
10.   WHILE es gibt Klasse C_j mit $\frac{9}{\varepsilon^2} \cdot V_{j,j-1} \leq V^*_{j-1}$
11.   Für kleinstes j mit dieser Eigenschaft verschmelze C_j und C_{j-1} gemäß der Verschmelzungsregel (beachte: Repräsentierungen von C^*_j berechnen aus C_j und C_{i-1})
12.   Gebe Schätzung gemäß der Schätzregel zurück
13. $i := i + 1$

Abbildung 2.20: Der Algorithmus VarianceMonitor

Beweis: Mit Hilfe von Korollar 2.42 und der Invariante gilt $\frac{9}{\varepsilon^2} \cdot V_m \leq V^*_m \leq \mathrm{Var}$. Unter Verwendung von Lemma 2.44 erhalten wir für den relativen Fehler

$$\frac{\left|\mathrm{Var} - \mathrm{Var}^{\mathrm{EST}}\right|}{\mathrm{Var}} \leq \frac{\frac{3}{2} \cdot V_m}{\mathrm{Var}} + \max\left(\frac{2C}{\varepsilon} \cdot \frac{V_m}{\mathrm{Var}}, \frac{2\varepsilon}{C}\right) \leq \frac{1}{6} \cdot \varepsilon^2 + \max\left(\frac{2C}{9} \cdot \varepsilon, \frac{2}{C} \cdot \varepsilon\right).$$

Für $C = 3$ folgt nun:

$$\frac{\left|\mathrm{Var} - \mathrm{Var}^{\mathrm{EST}}\right|}{\mathrm{Var}} \leq \frac{1}{6} \cdot \varepsilon^2 + \frac{2}{3} \cdot \varepsilon \leq \varepsilon.$$

■

Das Zusammenspiel von Verschmelzungsregel, Schätzregel und Invariante ist im Algorithmus VarianceMonitor wie in Abbildung 2.20 beschrieben niedergelegt.

Theorem 2.46 *Es seien $R > 0$, $N > 0$ und $\varepsilon > 0$. Der Algorithmus* VarianceMonitor *gibt zu jedem Zeitpunkt ab N für einen R-Stream eine bis auf den Faktor $1 \pm \varepsilon$ genaue Schätzung der Varianz der letzten N Stream-Elemente an und benötigt dafür*

1. *höchstens $O(\frac{1}{\varepsilon^2}(\log N + \log R))$ Histogrammklassen und*
2. *höchstens $O(\log N + \log R)$ Bits pro Klasse.*

Dies ergibt den Speicherbedarf

$$O\left(\frac{1}{\varepsilon^2}(\log N + \log R)^2\right).$$

Beweis: Für die Korrektheit des Algorithmus muss gezeigt werden, dass die Invariante erfüllt ist. Es gilt $V_j > 0$ nur, falls V_j durch Verschmelzung von Klassen C_ℓ und $C_\ell - 1$ entstanden ist. Damit gilt:

$$\frac{9}{\varepsilon^2} \cdot V_j = \frac{9}{\varepsilon^2} \cdot V_{\ell,\ell-1} \leq V^*_{\ell-1} \leq V^*_j,$$

wobei die vorletzte Ungleichung aus der Anwendung der Zeilen 10–11 folgt. Die letzte Abschätzung ergibt sich aus der einfachen Beobachtung, dass die Suffixklasse C^*_j im Zeitablauf nur größer werden kann. Mithin ist die Invariante stets erfüllt.

Die Komplexitätsaussagen ergeben sich aus folgenden Überlegungen: Wegen $V_{j,\ell} \geq V_j + V_\ell$ verdoppeln sich im schlechtesten Fall nach $O(\frac{1}{\varepsilon})$ Schritten die Varianzen der Suffixklasse C^*_j. Damit kann es jedoch nur höchstens $O(\frac{1}{\varepsilon^2} \cdot \log \mathrm{Var})$ Klassen geben. Da jedoch für die Varianz die Abschätzung $\mathrm{Var} \leq N \cdot R^2$ gilt, erhalten wir:

1. Es gibt höchstens $O(\frac{1}{\varepsilon^2} \cdot (\log N + \log R))$ Klassen.
2. Jede Klasse benötigt höchstens

   $$O(\underbrace{\log N}_{\text{Zeitmarken}} + \underbrace{\log N}_{\text{Größen}} + \underbrace{\log R}_{\text{Mittelwerte}} + \underbrace{\log N + \log R}_{\text{Varianzen}}) = O(\log N + \log R)$$

   Bits.

Damit ist das Theorem bewiesen. ∎

3 Netzwerkanalyse

Das Ziel der Netzwerkanalyse ist die Entwicklung algorithmischer Techniken zur Exploration großer (weitgehend) unbekannter Netzwerke auf graphentheoretischer Basis.

Hierbei stehen vor allem Inferenzprobleme im Mittelpunkt: Gegeben eine Menge von Beobachtungen, Daten usw. usf., welche Beziehungen bestehen zwischen den Netzwerkelementen, die konsistent mit den Beobachtungen, Daten usw. usf. sind?

Typische Fragen:

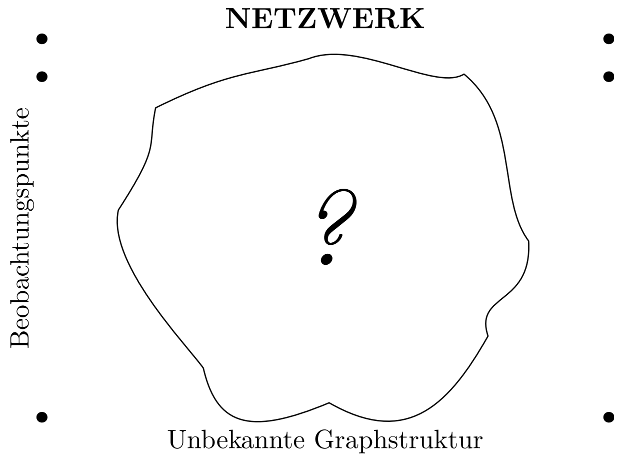
1. Welche Graphen realisieren eine bestimmte Abstandsmatrix?
2. Was sind wichtige Elemente/Gruppen in einem Netzwerk?
3. Gibt es hierarchische Strukturen in einem Netzwerk?

3.1 Rekonstruktion von Netzwerktopologien

Netzwerke sind Graphen mit Funktionen.

Netzwerktopologie = Struktur des dem Netzwerk zu Grunde liegenden Graphen

Wir betrachten folgendes Szenario:



Beobachtungspunkte (Monitore) unterscheidbar in:

- *passiv*: induzieren keine Aktivität (Informationsfluss, Datenverkehr etc.), sondern sammeln nur ankommende Informationen

- *aktiv*: induzieren Aktivität, um gezielt Informationen zu sammeln

Netzwerkrekonstruktionsproblem: Gegeben die Informationen *aller* Beobachtungspunkte, wie sieht die Graphenstruktur im unbekannten Netzwerkbereich aus?

Beachte: Anzahl und Position der Monitore ist i.A. *nicht* exogen vorgegeben

3.1.1 Dimensionierung und Positionierung von Monitoren

Wir führen eine beispielhafte Analyse für den Kantenzusammenhang in Graphen durch.

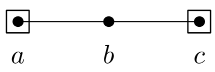
Konkretisierung dazu unser Szenario:

- Monitore sitzen in Knoten aus einer Menge D eines Graphen $G = (V, E)$.
- Monitore kommunizieren in periodischen Abständen untereinander.
- Gibt s Monitore $u, v \in D$, so dass u und v keine gemeinsame Kommunikation mehr aufrecht erhalten können (d.h. es gibt keinen Pfad zwischen u und v), so zeigen u und v ein Netzwerkversagen an.

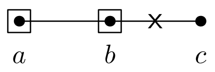
Klar: Gibt es Knoten $u, v \in D$, die ein Versagen anzeigen, so ist G tatsächlich nicht mehr zusammenhängend.

Ziel: Wähle D so aus dem Netzwerk G aus, dass ein Zusammenhangsverlust stets auch von einem Monitorpaar angezeigt wird.

Beispiel.



Monitore in a und c zeigen jeden Zusammenhangsverlust an



Monitore in a und b zeigen nicht Ausfall der Kante (b, c) an

Problem: Wir kennen die Netzwerktopologie nicht!

Definition 3.1 *Es sei $G = (V, E)$ ein ungerichteter Graph, $\varepsilon > 0, k \in \mathbb{N}$.*

1. *Knotenmengen $A, B \subseteq V$ heißen* getrennt *(in G), wenn für alle Knoten $a \in A$ und $b \in B$ kein Pfad von a nach b in G existiert.*

2. *Die Menge $Z \subseteq E$ heißt genau dann* (ε, k)-trennend*, wenn*

   (a) *$\|Z\| \leq k$ und*

   (b) *der Graph $(V, E \setminus Z)$ enthält getrennte Knotenmengen $A, B \subseteq V$ mit $\|A\| \geq \varepsilon \cdot n$ und $\|B\| \geq \varepsilon \cdot n$.*

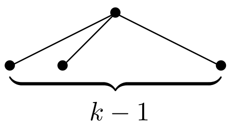
3. *Die Menge $D \subseteq V$ heißt genau dann* (ε, k)-erkennend*, wenn für alle (ε, k)-trennenden Mengen Z Knoten $u, v \in D$ existieren, die in unterschiedlichen Zusammenhangskomponenten von $(V, E \setminus Z)$ liegen.*

Bemerkungen.

1. V ist trivialerweise (ε, k)-erkennend für alle $\varepsilon > 0, k \in \mathbb{N}$.

2. Angenommen wir würden ε in der Definition weglassen, dann sieht die Definition für k-erkennend wie folgt aus: $D' \subseteq V$ heißt genau dann *k-erkennend*, wenn für alle $Z \subseteq E$ mit $\|Z\| \leq k$ gibt es $u, v \in D'$ in unterschiedlichen Zusammenhangskomponenten von $(V, E \setminus Z)$, falls $(V, E \setminus Z)$ nicht zusammenhängend ist. Dann gilt $D' = V$, z.B. für jeden Kreis und $k = 2$.

3. Angenommen wir würden k in der Definition weglassen, dann sieht die Definition für ε-erkennend wie folgt aus: $D'' \subseteq V$ heißt genau dann *ε-erkennend*, wenn es für alle $Z \subseteq E$ Knoten $u, v \in D'$ in unterschiedlichen Zusammenhangskomponenten A und B mit $\|A\|, \|B\| \geq \varepsilon \cdot n$ gibt, falls $(V, E \setminus Z)$ nicht zusammenhängend ist. Dann gilt $\|D'\| \geq (1 - \varepsilon) \cdot n$ z.B. für den Stern $K_{1,n-1}$.



Proposition 3.2 *Es seien $\varepsilon > 0$, $\delta > 0$ und $k \in \mathbb{N}$. Es sei $G = (V, E)$ ein ungerichteter Graph. Ein zufällig ausgewählte Knotenmenge $D \subseteq V$ (unter Gleichverteilung) der Größe*

$$O\left(\frac{k}{\varepsilon} \log \frac{\|E\|}{k} + \frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$$

ist mit Wahrscheinlichkeit mindestens $1 - \delta$ eine (ε, k)-erkennende Menge.

Beweis: Es sei die Menge $D \subseteq V$ mit Kardinalität $\ell =_{\text{def}} \|D\|$ zufällig im Graphen G gewählt. Es seien $Z_1, \ldots, Z_t$ alle möglichen Kantenmengen mit bis zu k Kanten (darunter also auch alle (ε, k)-trennenden Mengen). Es gilt

$$t = \sum_{i=1}^{k} \binom{\|E\|}{i} \leq k \binom{\|E\|}{k}.$$

Für Z_j sei $R_j \subseteq V$ eine Knotenmenge mit $\|R_j\| \geq \varepsilon \cdot n$ und R_j ist getrennt von $V \setminus R_j$ (d.h. es gibt keinen Pfad von R_j nach $V \setminus R_j$). Dann gilt

$$\begin{aligned}
P[D \cap R_j = \emptyset] &\leq \binom{n - \|R_j\|}{\ell} \cdot \binom{n}{\ell}^{-1} = \frac{(n - \|R_j\|)!}{\ell! \cdot (n - \|R_j\| - \ell)!} \cdot \frac{\ell! \cdot (n-\ell)!}{n!} \\
&= \frac{n - \|R_j\|}{n} \cdot \frac{n - \|R_j\| - 1}{n-1} \cdot \ \cdots \ \cdot \frac{n - \|R_j\| - \ell + 1}{n - \ell + 1} \\
&\leq \left(\frac{n - \|R_j\|}{n}\right)^{\ell} \leq \left(\frac{n - \varepsilon n}{n}\right)^{\ell} \leq (1-\varepsilon)^{\ell}
\end{aligned}$$

Wir erhalten folglich

$$\begin{aligned}
&\text{Prob}[\ D \text{ ist nicht } (\varepsilon, k)\text{-erkennend }] \\
&\leq \sum_{j=1}^{t} \text{Prob}[D \cap R_j = \emptyset] \leq k \cdot \binom{\|E\|}{k} \cdot (1-\varepsilon)^{\ell} \\
&\leq k \cdot \left(\frac{e \cdot \|E\|}{k}\right)^{k} \cdot (1-\varepsilon)^{\ell} \\
&\approx k \cdot \left(\frac{e \cdot \|E\|}{k}\right)^{k} \cdot e^{-\varepsilon \ell} \qquad \text{für } \varepsilon > 0 \text{ klein}
\end{aligned}$$

Für die Aussage des Satzes genügt es nun zu zeigen, für welche ℓ der letzte Term höchstens δ ist. Durch Logarithmieren ergibt sich

$$\ln k + k \cdot \ln \frac{e \cdot \|E\|}{k} - \varepsilon \ell \leq \ln \delta.$$

Folglich muss gelten:

$$\ell \geq \underbrace{\frac{1}{\varepsilon} \ln k}_{-\frac{1}{\varepsilon} \ln \frac{1}{k}} + \frac{k}{\varepsilon} \ln \frac{e \cdot \|E\|}{k} \underbrace{- \ \frac{1}{\varepsilon} \ln \delta}_{\frac{1}{\varepsilon} \ln \frac{1}{\delta}}.$$

Damit gibt es ein $\ell = O(\frac{k}{\varepsilon} \ln \frac{e \cdot \|E\|}{k} + \frac{1}{\varepsilon} \ln \frac{1}{\delta})$, das die Eigenschaft erfüllt. ■

Bemerkung. Die Größe von D hängt immer noch von der Graphengröße ab.

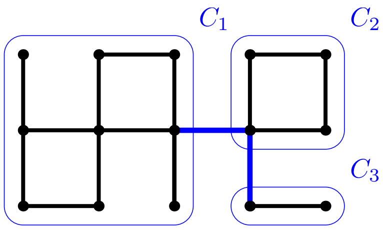
Ziel im weiteren Verlauf: D unabhängig machen von der Graphengröße.

Im Beweis von Proposition 3.2 überschätzen wir die Wahrscheinlichkeit, dass die Menge D getrennt ist von einer Knotenmenge R_j mit $\|R_j\| \geq \varepsilon n$ (insbesondere für große Mengen

R_j). Wir analysieren den Zusammenhang von G bei Entfernung von bis zu k Kanten genauer.

Es seien $G = (V, E)$ ein ungerichteter Graph und $Z \subseteq E$. Mit $C_1^Z, \ldots, C_{\ell_Z}^Z$ bezeichnen wir die Zusammenhangskomponenten von $(V, E \setminus Z)$. Die Menge $S \subseteq V$ heißt *trennbar mit k Kanten* genau dann, wenn es ein $Z \subseteq E$ gibt mit $\|Z\| \leq k$, so dass $I \subseteq \{1, \ldots, \ell_Z\}$ existiert mit $S = \bigcup_{i \in I} C_i^Z$.

Beispiel.



Die Mengen $\emptyset, C_1, C_2, C_3, C_1 \cup C_2, C_1 \cup C_3, C_2 \cup C_3$, und $C_1 \cup C_2 \cup C_3$ sind alle trennbar mit 2 Kanten.

Wir definieren für einen Graphen $G = (V, E)$:

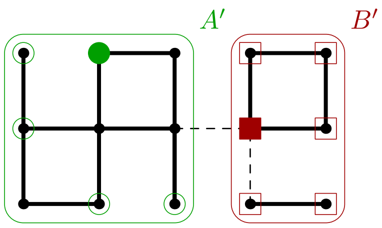
$$\mathcal{S}_k(G) =_{\text{def}} \{S \subseteq V \mid S \text{ ist trennbar mit } k \text{ Kanten}\}.$$

Lemma 3.3 *Es seien $G = (V, E)$ ein ungerichteter Graph, $\varepsilon > 0$ und $k \in \mathbb{N}$. Sei $D \subseteq V$ eine Knotenmenge mit $D \cap S \neq \emptyset$ für alle $S \in \mathcal{S}_k(G)$ mit $\|S\| \geq \varepsilon \|V\|$. Dann ist D eine (ε, k)-erkennende Menge.*

Beweis: Es seien $Z \subseteq E$ eine Kantenmenge mit $\|Z\| \leq k$ und $A, B \subseteq V$ getrennte Knotenmengen in $(V, E \setminus Z)$ mit $\|A\|, \|B\| \geq \varepsilon \cdot n$. Es seien $A_1, \ldots, A_r$ und $B_1, \ldots, B_s$ die Zusammenhangskomponenten von A und B. Wir definieren Mengen A' und B' vermöge

$$A' =_{\text{def}} \bigcup_{A_i \subseteq C_j^Z} C_j^Z \quad \text{und} \quad B' =_{\text{def}} \bigcup_{B_i \subseteq C_j^Z} C_j^Z.$$

Vergleiche dazu auch folgende Abbildung.



Es gilt nun:

- $A' \cap B' = \emptyset$.

- $\|A'\| \geq \|A\| \geq \varepsilon \cdot n$ und $\|B'\| \geq \|B\| \geq \varepsilon \cdot n$.
- $A', B' \in \mathcal{S}_k(G)$.

Damit gibt es in D Knoten $v_1 \in A'$ (der runde Knoten in der Abbildung) und $v_2 \in B'$ (der eckige Knoten in der Abbildung), $v_1 \neq v_2$. Diese Knoten liegen in verschiedenen Zusammenhangskomponenten von $(V, E \setminus Z)$. Damit ist D eine (ε, k)-erkennende Menge. ■

Wir bestimmen die Menge D also nicht bezüglich (ε, k)-trennender Mengen sondern bezüglich des Mengensystems $\mathcal{S}_k(G)$. Insbesondere muss D so gewählt werden, dass aus jeder Menge $S \in \mathcal{S}_k(G)$ ein Knoten in D ist.

Wir kümmern uns zunächst um die generelle Ausdrucksstärke von Mengensystemen.

Definition 3.4 *Es sei Ω eine endliche Grundmenge und es sei $\mathcal{F}$ eine Familie von Teilmengen von Ω.*

1. *Eine Menge $A \subseteq \Omega$ wird von $\mathcal{F}$* zerschmettert *(engl.* shattered*) genau dann, wenn es für alle $B \subseteq A$ ein $X \in \mathcal{F}$ gibt mit $B = A \cap X$.*
2. *Die* Vapnik-Chervonenkis-Dimension *(kurz VC-Dimension) von $(\Omega, \mathcal{F})$ ist die maximale Kardinalität einer Teilmenge von Ω, die von $\mathcal{F}$ zerschmettert wird, d.h.*

$$\text{VC-Dim}(\Omega, \mathcal{F}) =_{\text{def}} \max\{\ell \mid \textit{es gibt } S \subseteq \Omega \textit{ mit } \|S\| = \ell \textit{ und } \mathcal{F} \textit{ zerschmettert } S\ \}.$$

Beispiel. Wir betrachten folgendes Mengensystem $(\Omega, \mathcal{F})$ zusammen mit der Menge $S \subseteq \Omega$

$$\begin{aligned} \Omega &= \{1,2,3,4\}, \\ \mathcal{F} &= \{\{4\},\{1,3\},\{1,4\},\{2,4\},\{3,4\},\{1,2,3\},\{1,2,4\},\{2,3,4\}\} \\ S &= \{1,2,3\} \end{aligned}$$

Dann gilt:

$$\begin{aligned} \emptyset &= \{1,2,3\} \cap \{4\} \\ \{1\} &= \{1,2,3\} \cap \{1,4\} \\ \{2\} &= \{1,2,3\} \cap \{2,4\} \\ \{3\} &= \{1,2,3\} \cap \{3,4\} \\ \{1,2\} &= \{1,2,3\} \cap \{1,2,4\} \\ \{1,3\} &= \{1,2,3\} \cap \{1,3\} \\ \{2,3\} &= \{1,2,3\} \cap \{2,3,4\} \\ \{1,2,3\} &= \{1,2,3\} \cap \{1,2,3\} \end{aligned}$$

Die Menge S wird also von $\mathcal{F}$ zerschmettert, d.h. $\text{VC-Dim}(\Omega, \mathcal{F}) \geq 3$. Auf der anderen Seite wird Ω nicht von $\mathcal{F}$ zerschmettert, da z.B. die Menge $\{1,3,4\}$ nicht darstellbar ist. Damit gilt also $\text{VC-Dim}(\Omega, \mathcal{F}) = 3$.

Proposition 3.5 *Für jedes endliche Mengensystem $(\Omega, \mathcal{F})$ gilt* $\text{VC-Dim}(\Omega, \mathcal{F}) \leq \log \|\mathcal{F}\|$.

Beweis: Es sei $S \subseteq \Omega$ eine Menge, die von $\mathcal{F}$ zerschmettert wird. S enthält $2^{\|S\|}$ Teilmengen. Damit gilt $\|\mathcal{F}\| \geq 2^{\|S\|}$, da $\mathcal{F}$ für jede Teilmenge A von S mindestens eine Menge $X \in \mathcal{F}$ bereit halten muss für $A = S \cap X$. Für die Kardinalität einer größten Menge $S_{\max}$, die von $\mathcal{F}$ zerschmettert wird, gilt mithin

$$\text{VC-Dim}(\Omega, \mathcal{F}) = \|S_{\max}\| \leq \log \|\mathcal{F}\|.$$

■

Wir stellen nun einen Zusammenhang her zwischen VC-Dimension und (ε, k)-erkennenden Mengen. Die Feststellung, dass D für alle $S \in \mathcal{S}_k(G)$ einen Knoten enthalten muss, führt zu folgender allgemeinen Definition.

Definition 3.6 *Es sei $(\Omega, \mathcal{F})$ ein endliches Mengensystem, $\varepsilon > 0$. Eine Menge $N \subseteq \Omega$ heißt ε-*Netz* für $(\Omega, \mathcal{F})$ genau dann, wenn für alle $X \in \mathcal{F}$ mit $\|X\| \geq \varepsilon \|\Omega\|$ gilt $N \cap X \neq \emptyset$.*

Bemerkung. Die Begriffe „VC-Dimension“ und „ε-Netz“ sind wichtige Konzepte im Maschinellen Lernen. Insbesondere kann die VC-Dimension in natürlicher Weise auf unendliche Mengensysteme ausgedehnt werden, womit gezeigt werden kann. Eine Begriffsklasse kann wahrscheinlich annähernd korrekt aus dem Hypothesenraum gelernt werden (PAC-learning) genau dann, wenn die VC-Dimension des Hypothesenraums endlich ist (Satz von Blumer, Ehrenfeucht, Haussler, Warmuth).

Wir verwenden nun folgendes Lemma aus der Algorithmischen Lerntheorie (ohne Beweis).

Lemma 3.7 (Blumer, Ehrenfeucht, Haussler, Warmuth, 1990) *Es seien $(\Omega, \mathcal{F})$ ein endliches Mengensystem mit VC-Dimension d, $\varepsilon > 0$ und $\delta > 0$. Dann gibt es eine Funktion*

$$f(d, \varepsilon, \delta) = O\left(\frac{d}{\varepsilon} \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon} \log \frac{1}{\delta}\right),$$

so dass eine zufällige Teilmenge aus Ω mit Wahrscheinlichkeit mindestens $1 - \delta$ ein ε-Netz für $(\Omega, \mathcal{F})$ ist.

Wir müssen nun noch zeigen, dass die VC-Dimension von $(V, \mathcal{S}_k(G))$ konstant ist.

Die Idee, um dies zu zeigen ist die folgende: Für eine hinreichend große Knotenmenge $A \subseteq V$ betrachte für alle Knoten aus A „benachbarte“ Knoten außerhalb von A, die über paarweise kantendisjunkte Pfade erreichbar sind. Dann kann die Herausnahme von k Kanten nicht alle Pfade zerstören. Dann gibt es stets zwei Knoten in einer Zusammenhangskomponente, von denen einer zu A gehört und der andere nicht. Damit kann A nicht zerschmettert werden.

Lemma 3.8 *Es sei $G = (V, E)$ ein ungerichteter zusammenhängender Graph. Es sei $T \subseteq V$ ein Knotenmenge mit $\|T\| = 2\ell$ für $\ell \in \mathbb{N}$. Dann gibt es paarweise kantendisjunkte Pfade $p_1, \ldots, p_\ell$, so dass alle $v \in T$ Endpunkte für genau einen Pfad sind.*

Beweis: Es genügt, die Aussage für Bäume zu betrachten (ansonsten: ziehe Spannbaum heran). Wir beweisen die Aussage mittels Induktion über die Anzahl der Knoten in Bäumen.

Induktionsanfang: Für $n = 2$ ist die Aussage offensichtlich.

Induktionsschritt: Es sei H ein Baum mit $n \geq 3$ Knoten. Sei T eine Knotenmenge mit den angegebenen Eigenschaften. Wir betrachten zwei Fälle:

1. Es gibt ein Blatt v in H mit $v \notin T$. Dann entferne v aus H und wende die Induktionsvoraussetzung an. Übernehme die Pfade für H.

2. T enthält alle Blätter von H. Es sei v ein Blatt von H. Betrachten Nachbar w von v. Wir unterscheiden wieder zwei Fälle:

   (a) Es gilt $w \in T$. Definiere H' als Baum H ohne Knoten v (und Kante $\{v, w\}$). Setze $T' =_{\text{def}} T \setminus \{v, w\}$. Wir wenden die Induktionsvoraussetzung auf H' und T' an. Es seien $p_1, \ldots, p_r$ die Pfade nach Induktionsvoraussetzung. Dann sind $p_1, \ldots, p_r, \{v, w\}$ die gesuchten Pfade für H.

   (b) Es gilt $w \notin T$. Definiere H' als Baum H ohne Knoten v (und Kante $\{v, w\}$). Setze $T' =_{\text{def}} (T \setminus \{v\}) \cup \{w\}$. Wir wenden die Induktionsvoraussetzung auf H' und T' an. Es seien $p_1, \ldots, p_r$ die Pfade für H' und T' nach Induktionsvoraussetzung. O.B.d.A. ende Pfad $p_r = (u_1, \ldots, u_t, w)$ im Knoten w. Dann sind $p_1, \ldots, p_r, (u_1, \ldots, u_t, w, v)$ die gesuchten Pfade in H.

Damit ist die Aussage bewiesen. ■

Theorem 3.9 *Es seien $G = (V, E)$ ein ungerichteter Graph und $k \in \mathbb{N}$. Dann gilt* $\text{VC-Dim}(V, \mathcal{S}_k(G)) \leq 2k + 1$.

Beweis: Es sei $A \subseteq V$ eine Knotenmenge mit $\|A\| = 2k + 2$. Wir müssen zeigen, dass A nicht von $\mathcal{S}_k(G)$ zerschmettert wird. Nach Lemma 3.8 gibt es kantendisjunkte Pfade $p_1, \ldots, p_{k+1}$, so dass jeder Knoten aus A als Endpunkt für genau einen Pfad vorkommt. O.B.d.A. seien die Elemente von A so nummeriert, dass a_i und a_{i+k+1} die Endpunkte von Pfad p_i sind. Wir betrachten die Menge $B = \{a_1, \ldots, a_{k+1}\}$. Für jede Menge $Z \subseteq E$ mit $\|Z\| \leq k$ gibt es einen Pfad p_i, der auch in $(V, E \setminus Z)$ bestehen bleibt. Dann sind a_i und a_{i+k+1} in der gleichen Zusammenhangskomponente, d.h. für $S \in \mathcal{S}_k(G)$ gilt:

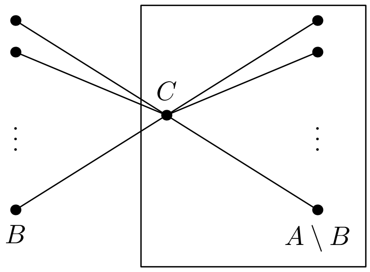
$$a_i \in S \iff a_{i+k+1} \in S$$

D.h. für alle $S \in \mathcal{S}_k(G)$ gilt $B \neq A \cap S$. ■

Proposition 3.10 *Es gibt Graphen $G = (V, E)$ mit* $\text{VC-Dim}(V, \mathcal{S}_k(G)) = 2k + 1$.

Beweis: Es gilt VC-Dim$(V, \mathcal{S}_k(G)) \leq 2k+1$ für alle Graphen G (nach Theorem 3.9). Wir müssen also Graphen finden, so dass eine $(2k+1)$-elementige Teilmenge von V durch $\mathcal{S}_k(G)$ zerschmettert wird. Dazu betrachten wir den Sterngraphen $K_{1,2k+1}$. Es sei A die Menge der Blätter von $K_{1,2k+1}$. Weiterhin sei $B \subseteq A$ eine beliebige Teilmenge. Wir unterscheiden zwei Fälle.

1. Es gelte $\|B\| \leq k$. Dann sei Z die Menge der Kanten von Knoten in B zum Zentrum c, $\|Z\| \leq k$. Dann ist B die Vereinigung von Zusammenhangskomponenten. D.h. $B = A \cap X$ für $X \in \mathcal{S}_k(K_{1,2k+1})$.



2. Es gelte $\|B\| > k$. Dann gilt $\|A \setminus B\| = \|V\| - \|B\| \leq k$. Wir argumentieren nun weiter wie im ersten Fall (ersetze dabei B durch $A \setminus B$).

Damit wird A von $(V, \mathcal{S}_k(K_{1,2k+1})$ zerschmettert. Es gilt $\|A\| \geq 2k+1$. Daraus folgt VC-Dim$(V, \mathcal{S}_k(K_{1,2k+1})) \geq 2k+1$. ∎

Theorem 3.11 *Es seien $\varepsilon > 0$, $\delta > 0$ und $k \in \mathbb{N}$. Es sei $G = (V, E)$ ein ungerichteter Graph. Eine zufällig gewählte Knotenmenge $D \subseteq V$ (unter Gleichverteilung) der Größe*

$$O\left(\frac{k}{\varepsilon}\log\frac{1}{\varepsilon} + \frac{1}{\varepsilon}\log\frac{1}{\delta}\right)$$

ist mit Wahrscheinlichkeit mindestens $1-\delta$ eine (ε, k)-erkennende Menge.

Beweis: Wir betrachten das Mengensystem $(V, \mathcal{S}_k(G))$. Nach Theorem 3.9 gilt

$$\text{VC-Dim}(V, \mathcal{S}_k(G)) \leq 2k+1.$$

Nach Lemma 3.7 existiert ein ε-Netz $D \subseteq V$ für $(V, \mathcal{S}_k(G))$ der Größe

$$f(2k+1, \varepsilon, \delta) = O\left(\frac{k}{\varepsilon}\log\frac{1}{\varepsilon} + \frac{1}{\varepsilon}\log\frac{1}{\delta}\right).$$

Hierbei kann D zufällig gewählt werden. Nach Lemma 3.3 ist jedes ε-Netz für $(V, \mathcal{S}_k(G))$ auch eine (ε, k)-erkennende Menge in G. ∎

Bemerkung. Die im Theorem angegebene Größe der zufällig gewählten Knoten ist asymptotisch optimal.

Im Folgenden betrachten wir zwei Erweiterungen des eben beschriebenen Ansatzes.

1. Erweiterung: Wir teilen die Knotenmenge V von G auf in zwei Kategorien:

- V_0 = Endknoten
- V_1 = interne Knoten

Es gilt $V = V_0 \cup V_1$. Monitore dürfen nur in V_0 platziert werden. Die (ε, k)-trennenden Mengen und (ε, k)-erkennenden Mengen beziehen sich lediglich auf V_0. An Stelle von $\mathcal{S}_k(G)$ betrachten wir die Mengenfamilie:

$$\mathcal{S}_k(G)|_{V_0} =_{\text{def}} \{S \cap V_0 \mid S \in \mathcal{S}_k(G)\}.$$

Proposition 3.12 *Es sei $(\Omega, \mathcal{F})$ ein endliches Mengensystem und $\Omega' \subseteq \Omega$. Es bezeichne $\mathcal{F}|_{\Omega'}$ die Familie $\{X \cap \Omega' \mid X \in \mathcal{F}\}$. Dann gilt*

$$\text{VC-Dim}(\Omega', \mathcal{F}|_{\Omega'}) \leq \text{VC-Dim}(\Omega, \mathcal{F}).$$

Beweis: Für $B \subseteq A \subseteq \Omega' \subseteq \Omega$ und $X' = X \cap \Omega' \in \mathcal{F}|_{\Omega'}$ gilt:

$$B = A \cap (\underbrace{X \cap \Omega'}_{\in \mathcal{F}|_{\Omega'}}) = (\underbrace{A \cap \Omega'}_{=A}) \cap X = A \cap X.$$

■

Korollar 3.13 *Es seien $\varepsilon > 0$, $\delta > 0$ und $k \in \mathbb{N}$. Es sei $G = (V, E)$ ein ungerichteter Graph mit $V_0 \subseteq V$. Eine zufällig gewählte Knotenmenge $D \subseteq V_0$ der Größe*

$$O\left(\frac{k}{\varepsilon} \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$$

ist mit Wahrscheinlichkeit mindestens $1 - \delta$ eine (ε, k)-erkennende Menge für V_0.

2. Erweiterung (nur skizzenhaft). Wir wollen auch Knotenausfälle entdecken. Dafür nehmen wir folgende Anpassungen vor:

- Eine Menge $Z \subseteq V \cup E$ mit $\|Z\| \leq k$ ist (ε, k)-trennend, falls $A, B \subseteq V_0$ mit $\|A\|, \|B\| \geq \varepsilon\|V_0\|$ existieren, die in $G \setminus Z = (V \setminus Z, E \setminus Z)$ getrennend sind.
- Eine Menge $D \subseteq V_0$ ist (ε, k)-erkennend, falls für alle (ε, k)-trennenden Z Knoten $u, v \in D$ in unterschiedlichen Zusammenhangskomponenten von $G \setminus Z$ liegen.

Erste Idee: Eine Menge $S \subseteq V$ trennbar durch k Elemente, falls $Z \subseteq V \cup E$ mit $\|Z\| \leq k$ existiert, so dass S Vereinigung von Zusammenhangskomponenten von $G \setminus Z$ ist. Es ergibt sich folgendes Problem: Betrachte den Stern $K_{1,n-1}$. Jede Teilmenge der Blätter ist trennbar durch einen Knoten (Zentrum von $K_{1,n-1}$). Damit ist VC-Dim$(V, S_1(K_{1,n-1})) \geq n-1$.

Es gibt folgenden Ausweg: Sei V geordnet, d.h. $V = (v_1, v_2, \ldots, v_n)$. Es seien $A, B \subseteq V$ mit $A \cap B$. Definiere: $A \leq B \Longleftrightarrow_{\text{def}} \min A \leq \min B$. Eine Menge $S \subseteq V$ ist ein k-*Segment*, falls eine Menge $Z \subseteq V \cup E$ mit $\|Z\| \leq k$ existiert mit $S = U_p \cup U_{p+1} \cup \cdots \cup U_q$, wobei $U_1, U_2, \ldots, U_s$ die lexikographisch geordneten Zusammenhangskomponenten von $G \setminus Z$ sind.

Proposition 3.14 *Es seien $G = (V, E)$ ein ungerichteter Graph mit $V_0 \subseteq V$, $\varepsilon > 0$ und $k \in \mathbb{N}$, $k < \frac{1}{3}\varepsilon\|V_0\|$. Sei $D \subseteq V_0$ eine Knotenmenge mit $D \cap S = \emptyset$ für alle k-Segmente S mit $\|S \cap V_0\| \geq \frac{1}{3}\varepsilon\|V_0\|$. Dann ist D eine (ε, k)-erkennende Menge für V_0.*

Theorem 3.15 *Es seien $\varepsilon > 0$, $\delta > 0$ und $k \in \mathbb{N}$. Es sei $G = (V, E)$ ein ungerichteter Graph mit $V_0 \subseteq V$. Eine zufällig gewählte Knotenmenge $D \subseteq V_0$ der Größe*

$$O\left(\frac{k^3}{\varepsilon} \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$$

ist mit Wahrscheinlichkeit mindestens $1-\delta$ eine (ε, k)-erkennende Menge für V_0 (bezüglich Kanten- und Knotenausfall).

Literaturverzeichnis

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ron L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 2 edition, 2001.

[GT02] Michael T. Goodrich and Roberto Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples.* John Wiley & Sons, Inc., Chichester, UK, 2002.

[KR03] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison-Wesley, Boston, MA, 2 edition, 2003.