INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR EFFIZIENTE ALGORITHMEN

Skriptum

zur Vorlesung

Internet-Algorithmik

gehalten im Wintersemester 2006/2007

von

Sven Kosub

16. Februar 2007

Version 0.6

Vorwort

Dieses Skriptum bietet die Grundlage für die Vorlesung „Internet-Algorithmik“, wie sie im Wintersemester 2006/2007 an der Technischen Universität München stattfand. Die Vorlesung hat einen Umfang von vier Semesterwochenstunden. Eine zugehörige Übung wurde nicht angeboten.

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.

Mittlerweile gibt es auch ein Lehrbuch, das für einen speziellen Teilbereich des Themengebietes der Vorlesung algorithmische Fragestellungen behandelt:

- [Var05] George Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices.* Morgan Kaufmann Publishers, San Francisco, CA, 2005.

Inhaltsverzeichnis

Einleitung 1

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

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

3. 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.

Wir werden alle drei Aspekte in unsere Betrachtungen einbeziehen.

Grundlegend für alle Aspekte ist die Verwendung von Protokollen.

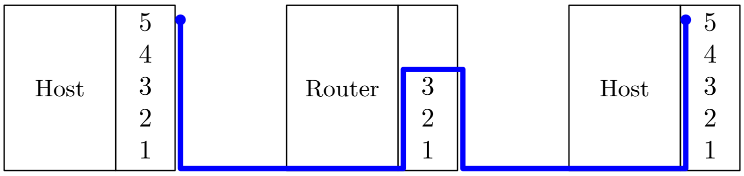
Was ist ein Protokoll?

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 Ereignisses 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

Die Vorlesung in diesem Semester behandelt primär die Analytik des Internets.

Datenanalyse 2

Ziel der Datenanalyse ist der Entwurf guter Algorithmen zur Analyse großer und dichter Datenmengen. Unter einem guten Algorithmus wollen wir in diesem Zusammenhang einen Algorithmus mit linearer Laufzeit abhängig von der Eingabegröße verstehen. Unter Umständen bedingt dies, dass die entsprechenden Probleme nicht exakt oder nur mit Hilfe des Zufalls gelöst werden können.

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 $s = s_0 \dots s_{n-1}$ gilt $|s| = n$.
- Das leere Wort wird mit ε bezeichnet, d.h. es gilt $|\varepsilon| = 0$.

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

1. *Ein Wort $s' \in \Sigma^*$ der Länge m heißt* Teilwort *von s, falls ein $i \in \{0, 1, \dots, n-1\}$ existiert mit $s_i s_{i+1} \dots s_{i+m-1} = s'$.*
2. *Ein Wort $s' \in \Sigma^*$ der Länge m heißt* Präfix *von s, falls $s' = s_0 s_1 \dots s_{m-1}$ 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 $s' = s_{n-m} s_{n-m+1} \dots s_{n-1}$ 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 aus
    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 $s = s_0 s_1 \dots s_{m-1}$ in einem Wort $t = t_0 t_1 \dots t_{n-1}$. 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 ergibt, dass der Algorithmus maximal $m(n - m + 1)$ Vergleiche benötigt, um ein Vorkommen von s in t zu finden. Die maximale Anzahl wird z.B. für $s = a^{m-1}b$ und $t = a^{n-1}b$ angenommen. Wir halten fest:

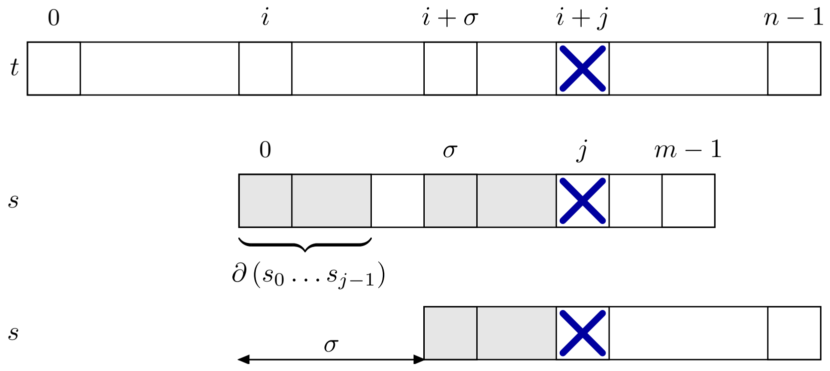
Proposition 2.2 *Der naive Algorithmus benötigt sowohl zum Finden eines Vorkommens als auch zum Finden aller Vorkommen eines Wortes der Länge m in einem Wort der Länge n im schlechtesten Fall $O(n \cdot m)$ Vergleiche.*

Im Folgenden 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 Donald E. Knuth, James H. Morris Jr. und Vaughan R. 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:

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

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

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

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 $i + j - |\partial(s_0 \ldots s_{j-1})|$.*

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:

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

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

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 $B[j] = |\partial(s_0 \ldots s_{j-1})|$. 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 $j < m$ AND $s_j = t_{i+j}$
3. $j := j + 1$
4.   IF $j = m$
5.   Gebe i aus
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. ■

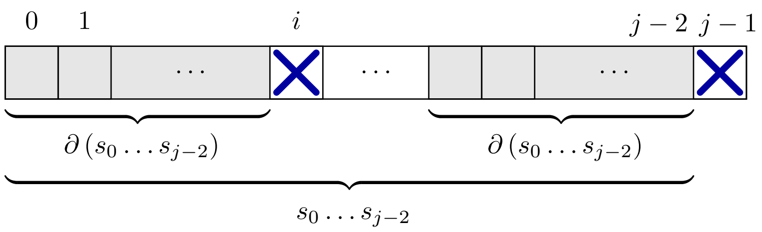
Wir müssen uns nun noch damit beschäftigen, wie wir das Feld B möglichst effizient berechnen können. Dazu führen wir weitere Vorüberlegungen durch. Angenommen wir hätten die Feldelemente $B[0], \ldots, B[j-1]$ bereits berechnet und wollen nun das Element

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 ≥ 0 AND s_k ≠ 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. ■

Wie finden wir alle Positionen? Unser Algorithmus ist bereits so formuliert, dass er alle Vorkommen des Suchwortes in einem Text ausgibt. Wieso ist dies richtig? Dazu überlegen wir uns folgendes: Es seien \$ und # Buchstaben, die nicht zum Alphabet Σ gehören. Für $s, t \in \Sigma^*$ mit $|s| = m$ und $|t| = n$ suchen wir nach dem Wort $s' = s\#$ in dem Wort $t' = t\$$. Wann immer eine Unverträglichkeit durch # verursacht wird, geben wir i aus. Die Korrektheit folgt, da $s\#$ nicht in $t\$$ vorkommt und, falls # die Unverträglichkeit (i, m) verursacht, gilt: $s_0 \ldots s_{m-1} = t_i \ldots t_{i+m-1}$ für $i \leq (n+1) - (m+1) = n - m$. Setzen wir die neuen Parameter formal in Satz 2.6 ein, so erhalten wir, dass alle Vorkommen von s in t mit höchstens $2(n+1) + (m+1) = 2n + m + 3$ Vergleichen gefunden werden. Natürlich können wir die gleichen Analyse wie zum Beweis von Satz 2.6 durchführen und erhalten die gleichen obere Schranke für die Anzahl der Vergleiche.

Korollar 2.7 *Der Algorithmus von Knuth, Morris und Pratt kann so modifiziert werden, dass $O(n + m)$ Vergleiche benötigt werden, um alle Anfangspositionen eines Wortes der Länge m in einem Wort der Länge n zu finden.*

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

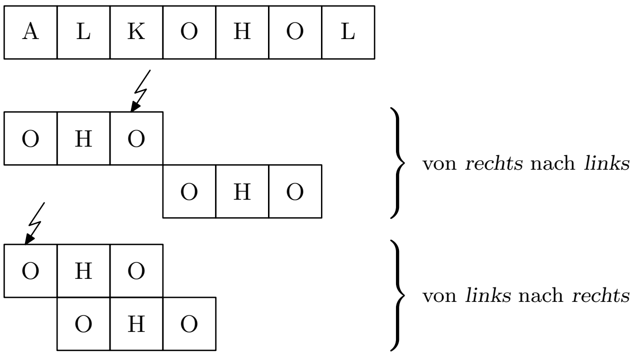
```
WHILE i ≤ n − m
    j := m − 1
    WHILE s_j = t_{i+j}
        j := j − 1
        IF j < 0
            Gebe i zurück
    i := i + 1
```

Abbildung 2.4: Einfacher Algorithmus mit Rechts-Links-Test

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.

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 in folgendem Beispiel zu sehen ist.



In diesem Fall können wir bei der Verschiebung i gleich auf $i + j + 1$ 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

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
WHILE $i \leq n - m$
    $j := m - 1$
    WHILE $s_j = t_{i+j}$
        $j := j - 1$
        IF $j < 0$
            Gebe i zurück
    $i := i + S[j]$

Abbildung 2.5: Algorithmus von Boyer und Moore

einen Linearzeit-Algorithmus zu erhalten. Ohne an dieser Stelle bereits näher auf die Vorberechnung der Verschiebungstabelle 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 $s_0 \dots s_{m-1}$ mit $t_i \dots t_{i+m-1}$. Es sei eine Unverträglichkeit bei (i, j) aufgetreten, d.h. es gilt

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

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 folgender Abbildung illustriert:

Bild zur Berechnung gemäß *good suffix rule*

Wir geben das Resultat der Laufzeitanalyse ohne Beweis an.

Theorem 2.8 (Cole 1994) *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 $\sigma =_{\text{def}} j - r$.

Algorithmus:	ComputeShifts
Eingabe:	Wort s mit $\|s\| = m$
Gesucht:	Tabelle 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

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

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

2.1.5 Galil's Erweiterung des Algorithmus von Boyer und Moore

Bisher haben wir den Algorithmus von Boyer und Moore nur für das einfache Suchproblem entwickelt, d.h. für die Beantwortung der Frage, ob eine Wort s in einem Wort t überhaupt vorkommt. Nunmehr wollen wir uns damit beschäftigen, wie alle Vorkommen berichtet werden können. Dies ist beim Algorithmus von Boyer und Moore nicht so einfach

```
Algorithmus:  BOYER-MOORE'
Eingabe:      Wörter s und t mit |s| = m und |t| = n
Gesucht:      alle Positionen, ab denen s in t vorkommt

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

Abbildung 2.7: Modifizierter Algorithmus von Boyer und Moore

zu erreichen, wie beim Algorithmus von Knuth, Morris und Pratt. Die hier präsentierte Erweiterung geht auf Zvi Galil zurück. Mit dieser Modifikation werden alle Positionen eines Suchwortes der Länge m in einem Wort der Länge n in Zeit $O(n+m)$ ausgegeben.

Ausgangspunkt für die nachfolgenden Betrachtungen ist die in Abbildung 2.7 angegebene Standardmodifikation des Algorithmus von Boyer und Moore zur Ausgabe aller Positionen.

Theorem 2.9 *Der modifizierte Algorithmus von Boyer und Moore benötigt max. $3n + 4rm - 6r$ Vergleiche, um alle r Positionen, ab denen ein Wort der Länge m in einem Wort der Länge n vorkommt, zu finden.*

Beweis: Es seien Wörter s und t mit $|s| = m$ und $|t| = n$ gegeben. Weiterhin sei r die Anzahl der Vorkommen von s in t. Es bezeichne $V(n,m,r)$ die Anzahl der Vergleiche, die der modifizierte Algorithmus von Boyer und Moore benötigt, um s in t genau r-mal zu finden. Es seien $n_1 - 1, n_2 - 1, \ldots, n_r - 1$ die Endpositionen der Vorkommen in t. Damit können wir eine rekursive Ungleichung für $V(n,m,r)$ aufstellen:

$$\begin{aligned} V(n,m,0) &\leq 3n && \text{(nach Satz 2.8 für erfolglose Suche)} \\ V(n,m,r) &\leq \underbrace{3(n_1-1)}_{\text{(I)}} + \underbrace{m}_{\text{(II)}} + \underbrace{V(n-n_1+m-1,m,r-1)}_{\text{(III)}} \end{aligned}$$

Der Term (I) ergibt sich als obere Schranke gemäß Satz 2.8 für die Suche von s im Präfix $t_0 \ldots t_{n_1-2}$, die erfolglos ausgeht, da erst $n_1 - 1$ die Endposition des ersten Vorkommens von s in t ist. Term (II) steht für die Anzahl der erfolgreichen Vergleiche zur Feststellung des ersten Vorkommens von s in t. Term (III) erklärt sich wie folgt: Wenn wir s bei Position $n_1 - 1 - m$ das erste Mal in t gefunden haben, so kommt s ab Position $n_1 - m$ in t noch genau $(r-1)$-mal vor. Dies entspricht der Suche von s in einem Wort der Länge $n - n_1 + m - 1$.

Diese Argumentation können wir nun auch für die anderen Vorkommen durchführen und erhalten unter der Vereinbarung $n_0 = m - 1$:

$$\begin{aligned} V(n,m,r) &\leq \left(\sum_{j=1}^{r} 3(n_j - n_{j-1} + m - 1)\right) + rm + V(n - n_r + m - 1, m, 0) \\ &\leq 3\left(\sum_{j=1}^{r}(n_j - n_{j-1})\right) + 3rm - 6r + rm + 3(n - n_r + m - 1) \\ &= 3n_r - 3n_0 + 4rm - 6r + 3n - 3n_r + 3(m-1) \\ &= 3n + 4rm - 6r \end{aligned}$$

Dies beweist die Aussage des Satzes. ■

Die Einsicht, die nun zu Galil's Erweiterung des Algorithmus führt, ist die, dass ein Suchwort nur dann sehr häufig in t vorkommt, wenn die Vorkommen sich häufig überlappen. Mithin muss das Wort s in einem bestimmten Sinne periodisch sein. Wir formalisieren dies im Folgenden.

Definition 2.10 *Es sei w ein Wort über Σ.*

1. *Ein Präfix u von w heißt genau dann* Periode *von w, wenn w ein Präfix von u^k für $k \geq 1$ ist.*
2. *Das Wort w heißt genau dann* periodisch*, wenn w ein Periode u mit $|u| \leq \frac{1}{2}|w|$ besitzt. Anderenfalls heißt das Wort w* aperiodisch*.*

Um Beispiele für die Begriffsbildungen zu geben, betrachten wir das Wort *abaabaabaa*. Dieses Wort hat die Perioden *aba*, *abaaba*, *abaabaaba* und natürlich *abaabaabaa*. Die Periode *aba* belegt, dass *abaabaabaa* ein periodisches Wort ist.

Eine alternative Charakterisierung für Perioden eines Wortes enthält die folgende Proposition.

Proposition 2.11 *Es seien u und w Wörter über Σ, wobei u ein Präfix von w ist. Dann ist u genau dann eine Periode von w, wenn es ein v gibt, so dass $w = uv$ gilt und v wiederum ein Präfix von w ist.*

Beweis: Übungaufgabe! ■

Um Aussagen darüber zu treffen, welche Periodenlängen in einem Wort vorkommen müssen, zitieren wir ein klassisches Resultat von Lyndon und Schützenberger ohne Beweis.

Lemma 2.12 (Lyndon & Schützenberger 1962) *Besitzt ein Wort $w \in \Sigma^*$ Perioden der Längen p und q und gilt $|w| \geq p + q$, so besitzt w eine Periode der Länge $\mathrm{ggT}(p, q)$.*

Im Folgenden werden aperiodische und periodische Suchwörter getrennt behandelt. Zunächst werden wir zeigen, dass aperiodische Suchwörter nicht sehr häufig in einem Text vorkommen können. Dazu beweisen wir folgendes Lemma.

Lemma 2.13 *Es seien s und t Wörter über Σ mit $|s| = m$ und $|t| = n$. Weiterhin sei $r = r(s, t)$ die Anzahl der Vorkommen von s in t. Ist $r > \frac{2n}{m}$, so ist das Wort s periodisch.*

Beweis: Angenommen es gilt $r > \frac{2n}{m}$. Nach dem Schubfachschlussprinzip muss es zwei Positionen p und $p+k$ in t geben, ab denen s in t vorkommt, wobei $k \leq \frac{m}{2}$ und $p \leq n-m-1$ gilt. (Anderenfalls würde $(r-1)m < p_r - p_1 \leq n - m - 1$ gelten, wobei p_1 die Position des ersten und p_r die Position des letzten Vorkommens in t ist. Dies wäre jedoch ein Widerspruch zu $r > \frac{2n}{m}$.) Damit ist $s = uv$ für $|u| = k$ und v ein Präfix von s. Nach Proposition 2.11 ist u eine Periode von s. Somit ist s periodsch. ■

Eine Folgerung aus diesem Lemma ist, dass der modifizierte Algorithmus von Boyer und Moore auf aperiodischen Suchwörtern bereits in Linearzeit läuft.

Korollar 2.14 *Der modifizierte Algorithmus von Boyer und Moore benötigt maximal $11n$ Vergleiche, um alle Vorkommen eines aperiodischen Suchwortes s in einem Wort der Länge n auszugeben.*

Beweis: Es sei s ein aperiodisches Wort der Länge m. Somit gilt nach Lemma 2.13 für die Anzahl $r = r(s, t)$ der Vorkommen von s in t die Ungleichung $r \leq \frac{2n}{m}$. Mithin ergibt sich unter Verwendung von Satz 2.9 für die Anzahl $V(n, m, r)$ der Vergleiche, die der modifizierte Algorithmus benötigt, um alle r Vorkommen zu finden:

$$V(n, m, r) \leq 3n + 4rm - 6r \leq 11n.$$

Dies beweist die Aussage des Korollars. ■

Wenden wir uns nunmehr periodische Suchwörtern zu. Dazu müssen wir unser Vokabular etwas erweitern.

Es sei $s = s_0 \ldots s_{m-1}$ ein periodisches Wort über Σ. Weiterhin sei u die kürzeste Periode von s. Es bezeichne k die Länge von u. Für k gilt also $k \leq \frac{m}{2}$. Es sei $t = t_0 \ldots t_{n-1}$ ebenfalls ein Wort über Σ. Wir definieren

$$P(s, t) =_{\mathrm{def}} \{\ p \mid s \text{ kommt in } t \text{ ab Position } p \text{ vor }\}.$$

Weiter definieren wir: Zwei Positionen $p, q \in P(s,t)$ heißen

- *nah*, falls $|p-q| \le \frac{m}{2}$ gilt,
- *benachbart*, falls es $p_0, p_1, \ldots, p_\ell \in P(s,t)$ gibt mit $p_0 = p$, $p_\ell = q$ und p_i und p_{i+1} sind nah für alle $0 \le i < \ell$.

Die Eigenschaft eines Positionspaares benachbart zu sein ist also der reflexive und transitive Abschluss der Eigenschaft nah zu sein. Da letztere Relation symmetrisch ist, definiert Benachbartheit also eine Äquivalenzrelation.

Ein *Cluster* ist eine maximale Teilmenge benachbarter Positionen aus $P(s,t)$. Jedes Cluster ist mithin eine Äquivalenzklasse. Es seien $C_1, \ldots, C_{r'}$ alle Cluster von $P(s,t)$ geordnet in aufsteigender Reihenfolge nach ihren maximalen Positionen, d.h. für alle $1 \le i \le r'-1$ gilt $\max C_i < \max C_{i+1}$. Dann können wir folgende Aussagen festhalten:

1. $\min C_{i+1} > \max C_i$ für alle $1 \le i \le r'-1$
2. $r' \le \frac{2n}{m}$

Die Positionen innerhalb eines Cluster haben eine sehr gleichmäßige Struktur.

Lemma 2.15 *Es sei $C \subseteq P(s,t)$ ein Cluster mit mindestens zwei Positionen. Die Positionen von C seien $p_1 < p_2 < \ldots < p_\ell$ für $\ell > 1$. Dann gilt $p_i - p_{i-1} = k$ für alle $1 < i \le \ell$.*

Beweis: Wir betrachten zwei Positionen $p_i, p_{i-1} \in C$. Dann gilt $k' =_{\text{def}} |p_i - p_{i-1}| \le \frac{m}{2}$. Nach Proposition 2.11 hat s somit eine Periode der Länge $k' \le k$. Wegen $k' + k \le m = |s|$ folgt nach Lemma 2.12, dass s auch eine Periode der Länge $\mathrm{ggT}(k', k)$ besitzt. Aus der Minimalität von k folgt somit, dass k' von k geteilt wird, d.h. $k' = d \cdot k$ für eine geeignete natürliche Zahl $d \ge 1$. Ist $d \ge 2$, so gibt es eine Position $p_{i-1} + k < p_i$ in $P(s,t)$ und folglich auch in C. Dies ist jedoch ein Widerspruch zur Reihenfolge der Elemente in C. Damit ergibt sich $d = 1$ und $k' = k$. ∎

Die Idee in Galil's Modifikation des Algorithmus von Boyer und Moore ist nun, Cluster als Ganzes zu überprüfen und nicht alle Einzelvorkommen in einem Cluster. Dazu müssen wir in der Lage sein, die kleinste Länge einer Periode von s zu bestimmen. Diese Information haben wir jedoch in der Standardmodifikation bereits mitberechnet. Denn es gilt ganz offensichtlich: $k = m - |\partial(s)|$. (Zur Erinnerung: Im Algorithmus ComputeShifts bestimmen wir auch die Ränderlängen alle Präfix von s einschließlich der Länge des eigentlichen Randes von s.)

Befinden wir uns innerhalb eines Clusters, so haben wir nach Lemma 2.15 nur kurze Verschiebungen um k. Wegen der Periodenüberlappungen brauchen wir lediglich die k neuen Buchstaben in t mit den letzten k Buchstaben von s vergleichen. Vergleiche für das

Algorithmus: Boyer-Galil-Moore
Eingabe: Wörter s und t mit $|s| = m$ und $|t| = n$
Gesucht: alle Positionen, ab denen s in t vorkommt

Berechne Verschiebungstabelle S für das Wort s
$S[-1] := m - B[m]$ /* B wird im ersten Schritte bereits mitberechnet */
$k_0 := B[m] + 1$
WHILE $i \leq n - m$
  $j := m - 1$
  WHILE $j \geq k$ AND $s_j = t_{i+j}$
    $j := j - 1$
  IF $j < k$
    Gebe i aus
    $k := k_0$
    $j := -1$
  ELSE
    $k := 0$
  $i := i + S[j]$

Abbildung 2.8: Galil's Modifikation des Algorithmus von Boyer und Moore

ganze Wort s müssen nur für das erste Vorkommen in einem Cluster getätigt werden. Eine präzise algorithmische Formulierung ist in Abbildung 2.8 zu sehen.

Theorem 2.16 *Galil's Erweiterung des Algorithmus von Boyer und Moore benötigt maximal $11n$ Vergleiche, um alle Vorkommen eines Wortes der Länge m in einem Wort der Länge n zu finden.*

Beweis: Es seien $s = s_0 \ldots s_{m-1}$ und $t = t_0 \ldots t_{n-1}$ gegeben. Wir unterscheiden zwei Fälle.

1. Ist s aperiodisch, dann folgt die Aussage aus Korollar 2.14, da für aperiodische Wörter s alle Cluster in $P(s,t)$ einelementig sind und somit der Algorithmenablauf der Standardmodifikation entspricht.

2. Es sei s periodisch. Wir bezeichnen mit $V'(n,m,r)$ die Anzahl der Vergleiche des Algorithmus Boyer-Galil-Moore, um alle Vorkommen von s in t in r Clustern auszugeben. Es seien $n_1 - 1, \ldots, n_r - 1$ die Endpositionen der ersten Vorkommen in den r Clustern und $n'_1 - 1, \ldots, n' - r - 1$ die Endpositionen der letzten Vorkommen in den r Clustern. Analog zum Beweis von Satz 2.9 erhalten wir wieder eine Rekursionsungleichung:

$$\begin{aligned} V'(n,m,r) &\leq 3n \\ V'(n,m,r) &\leq 3(n_1 - 1) + m + (n'_1 - n_1) + V'(n - n'_1 + m - 1, m, r - 1) \end{aligned}$$

Der einzige Unterschied zum Beweis von Satz 2.9 ist der dritte Term der rechten Seite der Abschätzung von $V'(n,m,r)$. Dieser ist jedoch leicht einzusehen, da innerhalb eines Clusters jede Position von t nur ein einziges Mal verglichen wird. Das ergibt gerade $n'_1 - n_1$ Vergleiche. Unter Abschwächung der Ungleichung erhalten wir eine Rekursionsungleichung, die bis auf die Interpretation von r identisch mit der im Beweis von Satz 2.9 ist. Wir schätzen wie folgt ab:

$$V'(n,m,r) \leq 3(n'_1 - 1) + V'(n - n'_1 + m - 1, m, r-1) \leq 3n - 4rm - 6r.$$

Außerdem wissen wir bereits, dass es für periodische Suchwörter maximal $\frac{2n}{m}$ Cluster gibt. Damit erhalten wir

$$V'(n,m,r) \leq 3n + 4rm \leq 3n + 8n = 11n.$$

Damit ist der Satz bewiesen. ■

2.1.6 Der Algorithmus von Aho und Corasick

Es ist unter Umständen wünschenswert, einen gegebenen Text nach den Vorkommen verschiedener Suchwörter abzusuchen. Ein Algorithmus, der in einem Text t das Vorkommen eines der Suchwörter $s_1, \ldots, s_k$ in Zeit $O(|t| + \sum_{i=1}^{k} |s_i|)$ finden kann, ist der Algorithmus von Alfred V. Aho und Margaret J. Corasick. Dieser Algorithmus basiert auf der alten Idee, das Suchproblem automatentheoretisch aufzufassen.

Wir präsentieren hier nur die Ideen für den Algorithmus.

Es sei s ein Suchwort. Wir definieren die Sprache L_s wie folgt:

$$L_s =_{\text{def}} \{\ t \in \Sigma^* \mid s \text{ kommt in } t \text{ vor }\}$$

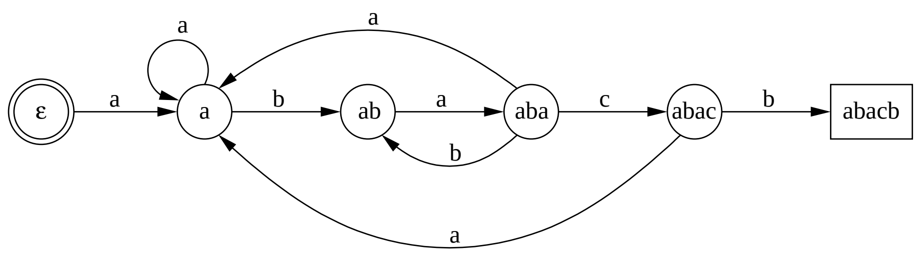
Zwei Probleme treten auf:

1. Ist L_s regulär? Wenn ja, dann gibt es einen deterministischen endlichen Automaten M_s, der L_s akzeptiert, d.h. gegeben M_s kann die Suche in t für $|t| = n$ in Zeit $O(n)$ durch geführt werden.

2. Kann ein solcher endliche Automat M_s in Zeit $O(m)$ mit $m = |s|$ berechnet werden?

Aus den Lösungen für beide Probleme ergibt sich ein Algorithmus mit Laufzeit $O(n+m)$.

Wir betrachten zunächst das erste Problem. Ein Beispiel für einen Automaten zur Akzeptierung einer Sprache L_s ist der folgende:



Hierbei ist $s = abacb$ ein Wort über dem Alphabet $\Sigma = \{a, b, c\}$. Die nicht gezeichneten Übergänge führen immer in den Zustand ε. Dies wollen wir als generelle Vereinbarung beibehalten.

Allgemein konstruieren wir den endlichen Automaten wie folgt: Es seien Σ ein festes endliches Alphabet und $s \in \Sigma^*$. Wir definieren eine Abbildung $\sigma_s : \Sigma^* \to \mathbb{N}$ als

$$\sigma_s(x) =_{\text{def}} \max\{\ j \mid s_0 \dots s_{j-1} \text{ ist ein Suffix von } x\ \}.$$

Besteht keine Verwechselungsgefahr bezüglich des Wortes s, so lassen wir den Index weg. Nun betrachten wir den folgenden Automaten $M_s = (Q, \delta, q_0, F_+)$:

- $Q =_{\text{def}} \{\ s' \mid s' \text{ ist ein Präfix von } s\ \}$
- $\delta(s', a) =_{\text{def}} s_0 \dots s_{\sigma(s'a)-1}$ für $s' \in Q$ und $a \in \Sigma$
- $q_0 =_{\text{def}} \varepsilon$
- $F_+ =_{\text{def}} \{s\}$

Die Korrektheit der Konstruktion gilt wegen:

Proposition 2.17 *Es sei M_s der deterministische endliche Automat für ein Wort $s \in \Sigma^*$. Für alle $t \in \Sigma^*$ und für alle $0 \leq i \leq |t|$ gilt $\hat{\delta}(t_0 \dots t_{i-1}) = s_0 \dots s_{\sigma(t_0 \dots t_{i-1})-1}$.*

Beweis: Wir verwenden Induktion über i.

1. Zum Induktionsanfang sei $i = 0$. Dann gilt die Aussage trivialerweise ($\varepsilon = \varepsilon$).
2. Für den Induktionsschritt sei $i > 0$ (und $i \leq |t|$). Es sei $t' =_{\text{def}} t_0 \dots t_{i-2}$ und $a = t_{i-1} \in \Sigma$. Dann gilt für $t_0 \dots t_{i-1} = t'a$:

$$\begin{aligned} \hat{\delta}(t'a) &= \delta(\hat{\delta}(t'), a) && \text{(nach Definition von } \hat{\delta}) \\ &= \delta(s_0 \dots s_{\sigma(t')-1}, a) && \text{(nach Induktionsvoraussetzung)} \\ &= s_0 \dots s_{\sigma(t'a)-1} && \text{(nach Definition von } \delta) \end{aligned}$$

Damit ist die Proposition bewiesen. ■

Wenden wir uns nun dem zweiten, eingangs erwähnten Problem zu: Wie berechnen wir M_s, d.h. insbesondere die Übergangsfunktion δ? Halten wir zunächst fest: $\|Q\| = 1 + m$. Es gibt also keine trivialen Gründen, warum δ nicht berechnet werden können sollte. Weiterhin gilt nach Definition $\sigma_s(s') = |s'|$ für alle Präfixe s' von s. Insgesamt ergibt sich somit für $0 \leq i \leq m - 1$:

$$\sigma_s(s_0 \dots s_{i-1}a) = \begin{cases} i + 1 & \text{falls } s_i = a \text{ gilt} \\ |\partial(s_0 \dots s_{i-1}a)| & \text{sonst} \end{cases}$$

Algorithmus:	ComputeTransitions
Eingabe:	Wort $s = s_0 \dots s_{m-1}$
Gesucht:	eine Menge von Arrays S_a für $a \in \Sigma$, wobei $S_a[j] = \sigma_s(s_0 \dots s_{j-1}a)$ gilt

FOR $a \in \Sigma$
  $B_a[0] := -1$
  $B_a[1] := 0$
  $\ell_a := 0$
FOR $j := 2$ TO m
  $\ell := \ell_{s_{j-2}}$
  FOR $a \in \Sigma$
    $\ell_a := \ell$
    WHILE $\ell_a \geq 0$ AND $s_{\ell_a} \neq a$
      $\ell_a := B_{s_{l_a}}[\ell_a]$
    $B_a[j] := \ell_a + 1$
    $\ell_a := \ell_a + 1$
FOR $i := 0$ TO $m-1$
  IF $a = s_i$
    $S_a[i] := i + 1$
  ELSE
    $S_a[i] := B_a[i+1]$

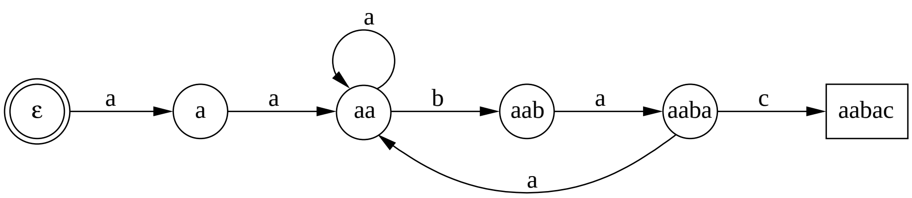
Abbildung 2.9: Algorithmus zur Berechnung der Übergangsfunktion

Zur Berechnung kann deshalb einfach der Algorithmus ComputeBoundaries zur Bestimmung der Ränderlängen beim Algorithmus von Knuth, Morris und Pratt adaptiert werden. Wie dies aussehen kann, ist in Abbildung 2.9 angegeben.

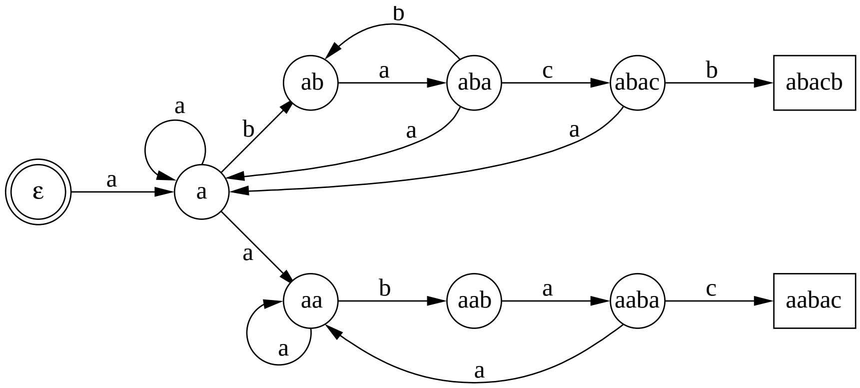
Proposition 2.18 *Für ein Alphabet Σ und ein Wort $s \in \Sigma^*$ der Länge m kann die Übergangstabelle für den deterministischen endlichen Automaten M_s in Zeit $O(\|\Sigma\| \cdot m)$ berechnet werden.*

Beweis: Analog zur Analyse von ComputeBoundaries (Lemma 2.5). ■

Im Folgenden geben wir noch eine Idee für die Verallgemeinerung des Ansatzes auf Suchwörter $s_1, \dots, s_k$. Exemplarisch seien dazu wiederum das Suchwort $s_1 = abacb$ mit dem zugehörigen Automaten M_{s_1} und das Suchwort $s_2 = aabac$ mit dem zugehörgin Automaten M_{s_2}



betrachtet. Um nach einem der beiden Wörter in einem Text zu suchen, müssen wir gleichzeitig die Eingabe in M_{s_1} und M_{s_2} verarbeiten. Dies führt standardmäßig zu einem nichtdeterministischen Automaten $\hat{M}_{s_1,s_2}$



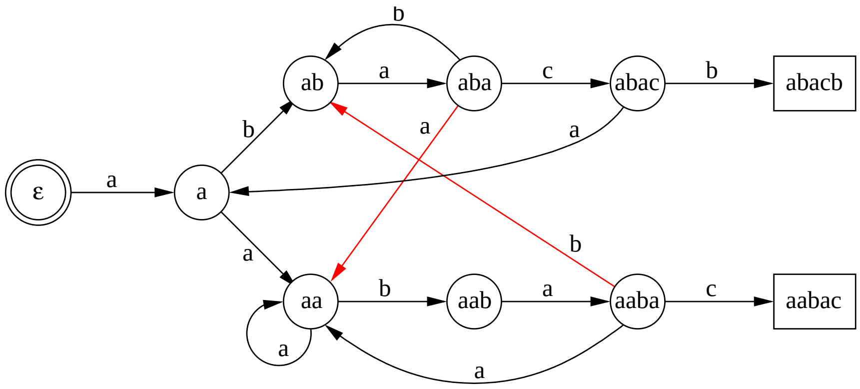
Es seien $M_{s_1}, \ldots, M_{s_k}$ die endlichen Automaten für die Suchwörter $s_1, \ldots, s_k$. Mit δ_{s_i} bezeichnen wir die Übergangsfunktion des Automaten M_{s_i}. Die allgemeine Konstruktionsvorschrift für den nichtdeterministischen endlichen Automaten $\hat{M}_{s_1,\ldots,s_k} = (Q, \delta, q_0, F_+)$ lautet nun wie folgt:

- $Q =_{\text{def}} \{ s' \mid s' \text{ ist Präfix für irgendein } s_i \}$
- $\delta(s', a) =_{\text{def}} \{\delta_{s_i}(s', a) \mid s' \text{ ist Präfix von } s_i \}$
- $q_0 =_{\text{def}} \varepsilon$
- $F_+ =_{\text{def}} \{s_1, \ldots, s_k\}$

Die Korrektheit der Konstruktion ist offenkundig eine Folgerung aus der Korrektheit der Automaten für die einzelnen Suchwörter.

Problematisch ist jedoch noch die Umwandlung des nichtdeterministischen in einen determinischen Automaten, da im Allgemeinen eine kombinatorische Explosion der Zustandsmenge auftreten kann. In unserem Fall ist dies jedoch nicht der Fall. Ein äquivalenter deterministischer endlicher Automat $M_{s_1,\ldots,s_k}$ besitzt die gleiche Zustandsmenge Q wie

$\hat{M}_{s_1,\ldots,s_k}$, für die sowieso $\|Q\| \leq 1 + \sum_{i=1}^{k} |s_i|$ gilt. In unserem Beispiel hat der resultierende deterministische Automat $M_{abacb,aabac}$ das folgende Aussehen:



Der Algorithmus von Aho und Corasick produziert darüber hinaus die Übergangsfunktion stets in linearer Zeit. Ohne tiefer in die Details des Algorithmus zu gehen und ohne Beweis fassen wir zusammen:

Theorem 2.19 *Der Algorithmus von Aho und Corasick benötigt $O(n + \|\Sigma\| \cdot \sum_{i=1}^{k} |s_i|)$ Schritte, um in einem Wort aus Σ^* der Länge n festzustellen, ob eines der Wörter $s_1, \ldots, s_k \in \Sigma^*$ vorkommt.*

Mit einer leichten Modifikation der Automatensimulation ist auch erreichbar, dass alle Vorkommen der Suchwörter berichtet werden. Die Laufzeit bleibt dabei asymptotisch unverändert.

2.2 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 (z.B. Erkennung von Adressraumexploration)
- Router-Statistiken („Wem gehören die meisten Pakete?“ usw. usf.)

2.2.1 Verarbeitungsmodelle und Komplexitätsmaße

Es sei $\Sigma = \{a_1, \dots, a_n\}$ ein Alphabet der Größe n. (Wir betrachten aber auch unbeschränkte Alphabete.)

Ein *Datenstrom* s ist eine endliche Folge $s = (s_1, \dots, s_i, \dots, s_N)$ 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 (sequenzieller Zugriff).

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?“
- 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. Für feste Suchwörter sind sowohl der Algorithmus von Knuth, Morris und Pratt also der Algorithmus von Aho und Corasick gute Datenstromalgorithmen (ein Pass, konstanter Speicherplatz, lineare Verarbeitungszeit). Der Algorithmus von Boyer und Moore benötigt $\Omega(N)$ Pässe im schlechtesten Fall.

2.2.2 Ein-Pass-Algorithmen zur Hot-List-Analyse

Unter der *Hot-List-Analyse* verstehen wir das folgende Auswertungsproblem: Gegeben sei ein Datenstrom $s = (s_1, \dots, s_N)$ über einem Alphabet Σ mit $\|\Sigma\| = n$. Bestimme die Elemente in s, die mit Häufigkeit ϑ vorkommen, d.h. gebe 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.

Die Menge $H(s, \vartheta)$ heißt *Hot-List* von s bezüglich ϑ.

Proposition 2.20 *Es gibt einen Ein-Pass-Algorithmus, der die Hot-List $H(s, \vartheta)$ für einen Datenstrom s der Länge N mit $O(n \cdot \log N)$ Speicherplatz berechnet.*

Beweis: Für jedes $a \in \Sigma$ zähle Vorkommen in s in einem Binärzähler. ∎

Im Folgenden wollen wir zeigen: Mit einem Pass geht es nicht wesentlich besser als in Proposition 2.20 angegeben.

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 Ein-Pass-Algorithmus, der die Hot-List $H(s, \vartheta)$ bestimmt. Wir betrachten den Zustand von A in dem Moment, wenn das Element s_M mit $M = \lfloor \frac{N}{2} \rfloor$ 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

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

Es sei $(k_1, \ldots, k_r) \in K_N$. Für dieses Tupel können wir jedoch für jedes $i \in \{1, \ldots, n-1\}$ die zugehörigen Grenzen zu den Blöcken $i-1$ bzw. $i+1$ unabhängig um bis zu $\lfloor \frac{N}{4n} \rfloor \leq \frac{1}{2} \lfloor \frac{N}{2n} \rfloor$ nach links und rechts verschieben und es gilt

$$\left\lceil \frac{N}{2n} \right\rceil + \left\lfloor \frac{N}{4n} \right\rfloor \leq \frac{3N}{4n} + 1 \leq \frac{3}{16} \cdot \vartheta \cdot N + 1 < \vartheta \cdot N.$$

Für die letzte Abschätzung beachten wir, dass $N > 16$ ist. 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. ∎

2.2.3 Hot-List-Analyse mit zwei Pässen

Wir gehen zur Betrachtung von Zwei-Pass-Algorithmen über. Der hier vorgestellte Algorithmus basiert auf einer Idee von Richard M. Karp, Christos H. Papadimitriou und Scott J. Shenker.

Zunächst halten wir folgende einfache Abschätzung der Kardinalität von Hot-List-Mengen fest.

Proposition 2.22 *Für jedes $s \in \Sigma^*$ und $\vartheta \in (0,1)$ gilt $\|H(s,\vartheta)\| < \frac{1}{\vartheta}$.*

Beweis: Es gilt für alle $s \in \Sigma^*$, $\vartheta \in (0,1)$:

$$N = |s| \geq \sum_{a \in H(s,\vartheta)} |s|_a > \vartheta \cdot N \cdot \|H(s,\vartheta)\|.$$

Damit folgt sofort nach Umstellung: $\|H(s,\vartheta)\| < \frac{1}{\vartheta}$. ∎

Wir verwenden folgende Idee:

> *Wenn alle Elemente im Datenstrom so angeordnet werden 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 2.10 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 2.10: Berechnung der Kandidatenmenge für $\vartheta \in (0, 1)$

Beweis: Wir müssen zeigen, dass alle $a \in H(s, \vartheta)$ mit K ausgegeben werden. Es sei $a \in \Sigma$ ein Element, dass von ComputeCandidates nicht ausgegeben wird, d.h. $a \notin K$. Jedes Vorkommen von a in s wird zusammen mit $\lceil \frac{1}{\vartheta} \rceil - 1 \geq \frac{1}{\vartheta} - 1$ Vorkommen anderer Symbole aus Σ gelöscht (Zeilen 8–12). Insgesamt werden also 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)

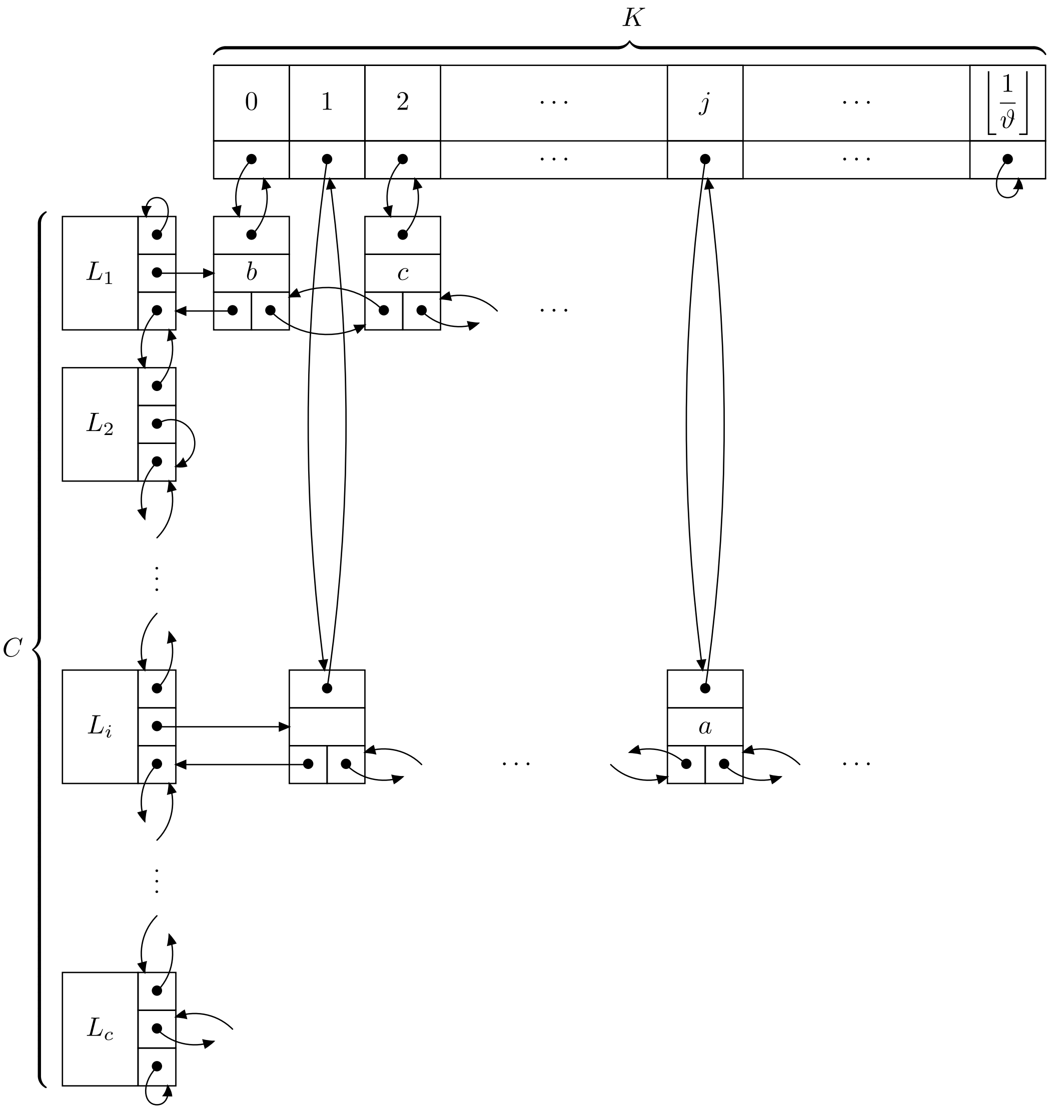
Verfeinerte Datenstrukturen für K und C:

1. Operationen für Objekt K:
   - insertItem(a): Füge neues Element a in K ein
   - incCounter(a): Inkrementiere Zähler von Element $a \in K$
   - decAllCounters(): Dekrementiere alle Zähler für Elemente $a \in K$ (und entferne Elemente mit Zählerstand 0)
   - size(), isElement(a) usw. usf.

2. Implementierungen:

   - K als Hash-Tabelle
   - C als Listenstruktur $(L_1, \ldots, L_c)$ mit folgender Interpretation:
     - L_i ist Liste $(a_{i_1}, \ldots, a_{i_r})$ mit $a_{i_j} \in K$ und $C[a_{i_j}] = i$
     - L_c enthält Elemente mit höchstem aktuell vorkommenden Zählerstand
   - insertItem(a): Füge a in K und L_1 ein
   - incCounter(a): Entferne a aus L_i und füge a in L_{i+1} ein (wenn $i = c$, so erzeuge neue Liste L_{i+1})
   - decAllCounters(): 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.

2.2.4 Exakte Algorithmen zur Momentenanalyse

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 (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* Moment $F_\infty(s)$ 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 oder um Bearbeitungspläne für Datenbankanfragen zu optimieren).

- Mit Momenten lassen sich Erwartungswert, Varianz, Schiefe usw. bestimmen.

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

Proposition 2.26 *Für $k \in \mathbb{N}$ kann $F_k(s)$ für einen Datenstrom s der Länge N über einem Alphabet der Größe n lassen sich exakt mit $O(n \log N)$ Platz berechnet werden.*

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 Ein-Pass-Algorithmen (unter Verwendung der sogenannten Spurenmethode aus Theorem 2.21)

2.2.5 Der Schätzalgorithmus von Alon, Matias und Szegedy

Dieser 1999 von Noga Alon, Yossi Matias und Mario Szegedy publizierte Algorthmus, der zusammen mit noch weiteren, sehr raffinierten Resultaten in der selben Arbeit 2005 mit dem renommierten Gödel-Preis ausgezeichnet worden ist, basiert auf der einfachen Idee der Stichprobe (englisch *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 2.11 zu sehen. Zu beachten ist allerdings, dass diese Form keinem Ein-Pass-Algorithmus entspricht. Dies lässt sich aber leicht einrichten.

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.

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

1. Parameter m_1 und m_2 geeignet initialisieren
2. FOR $i := 1$ TO m_2
3.   FOR $j := 1$ TO m_1
4.   RANDOM p IN $\{1, \ldots, N\}$
5. $r := 0$
6. $a := s_p$
7.   FOR $q := p$ TO N
8.   IF $s_q = a$
9. $r := r + 1$
10. $X[i,j] := N \cdot (r^k - (r-1)^k)$
11.   FOR $j := 1$ TO m_1
12. $Y[i] := Y[i] + \frac{1}{m_1} \cdot X[i,j]$
13. Gebe den Median von $\{Y[1], \ldots, Y[m_2]\}$ aus

Abbildung 2.11: Der Algorithmus SampleCount

Wie sind nun aber m_1 und m_2 zu wählen, damit sowohl Platzbedarf als auch Fehlerwahrscheinlichkeit klein gehalten werden können?

Proposition 2.27 *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:

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

Dies beweist die Aussagen der Proposition. ■

Lemma 2.28 *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

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

Dies beweist die Aussage des Lemma. ∎

Lemma 2.29 *Für alle Zahlen $x_1, \ldots, x_n \in \mathbb{R}_+$ gilt*

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

(Für $x_1 = n^{1/k}, x_2 = \ldots = x_n = 1$ gilt Gleichheit bis auf konstanten Faktor.)

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

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

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

Bei der Abschätzung in der vorletzten Zeile haben wir 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 2.30 *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 2.27 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

$$\begin{aligned}
\mathrm{Var}Y[i] &= \mathrm{Var}\left(\frac{1}{m_1}\sum_{j=1}^{m_1} X[i,j]\right) \\
&= \frac{1}{m_1} \cdot \mathrm{Var}X && \text{(mit Hilfe von Proposition 2.27)} \\
&\leq \frac{1}{m_1} \cdot \mathbb{E}X^2 \\
&\leq \frac{1}{m_1} \cdot k \cdot F_1(s) \cdot F_{2k-1}(s) && \text{(Lemma 2.28)} \\
&\leq \frac{k \cdot n^{1-\frac{1}{k}}}{m_1} \cdot F_k(s)^2 && \text{(Lemma 2.29)}
\end{aligned}$$

Dies beweist die Aussage des Korollars. ■

Unser Ziel ist es, die Fehlerwahrscheinlichkeit von SampleCount klein zu halten. Für diese gilt mit Korollar 2.30 und der Chebyshev'schen Ungleichung

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

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 2.31 *Werden für $k \geq 1$, $\delta > 0$ und $\varepsilon > 0$ die Parameter m_1 und m_2 im Algorithm* SampleCount *(siehe Abbildung 2.11) 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). ■

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.
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 2.12 beschrieben Algorithmus von Alon, Matias und Szegedy gibt eine Lösung für diese Probleme an.

Korollar 2.32 *Der Algorithmus von Alon, Matias und Szegedy berechnet das Moment $F_k(s)$ 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 $\frac{1}{p}$ (dies geschieht nur im Falle $L = p$).

Algorithmus: Alon-Matias-Szegedy
Eingabe: Datenstrom $s = (s_1, \dots, s_N)$ über $\Sigma = \{a_1, \dots, a_n\}$
Ausgabe: (Schätzung für das) Moment $F_k(s)$

```
Parameter m_1 und m_2 geeignet initialisieren /* siehe Theorem 2.31 */
FOR i := 1 TO m_2
   FOR j := 1 TO m_1
      A[i,j] := s_1
      R[i,j] := 1
L := 2
WHILE Element s_L existiert
   FOR i := 1 TO m_2
      FOR j := 1 TO m_1
         RANDOM p IN {1, ..., L}
         IF p = L
            A[i,j] := s_p
            R[i,j] := 1
         ELSE
            IF s_L = A[i,j]
               R[i,j] := R[i,j] + 1
         X[i,j] := L · (R[i,j]^k − (R[i,j] − 1)^k)
   L := L + 1
Berechne Y[i] als Durchschnitt über X[i,1], ..., X[i,m_1]
Gebe den Median von {Y[1], ..., Y[m_2]} aus
```

Abbildung 2.12: Der Algorithmus von Alon, Matias und Szegedy

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:

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

Dies beweist die Aussage des Korollars. ■

Bemerkungen.

1. Für $k = 2$ gibt es einen Algorithmus, der mit $O(\log N + \log N)$ Platz auskommt an Stelle von $O(\sqrt{n}(\log N + \log n)$, wie aus Korollar 2.32 folgt.

2. Für $k \geq 5$ ist $\Omega(n^{1-5/k})$ eine untere Platzschranke.

3. Für $k = 0$ gibt es einen $O(\log n)$-Platz-Algorithmus (Algorithmus von Flajolet und Martin); der Algorithmus von Alon, Matias und Szegedy funktioniert für $k = 0$ nicht.

2.3 Monitore

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

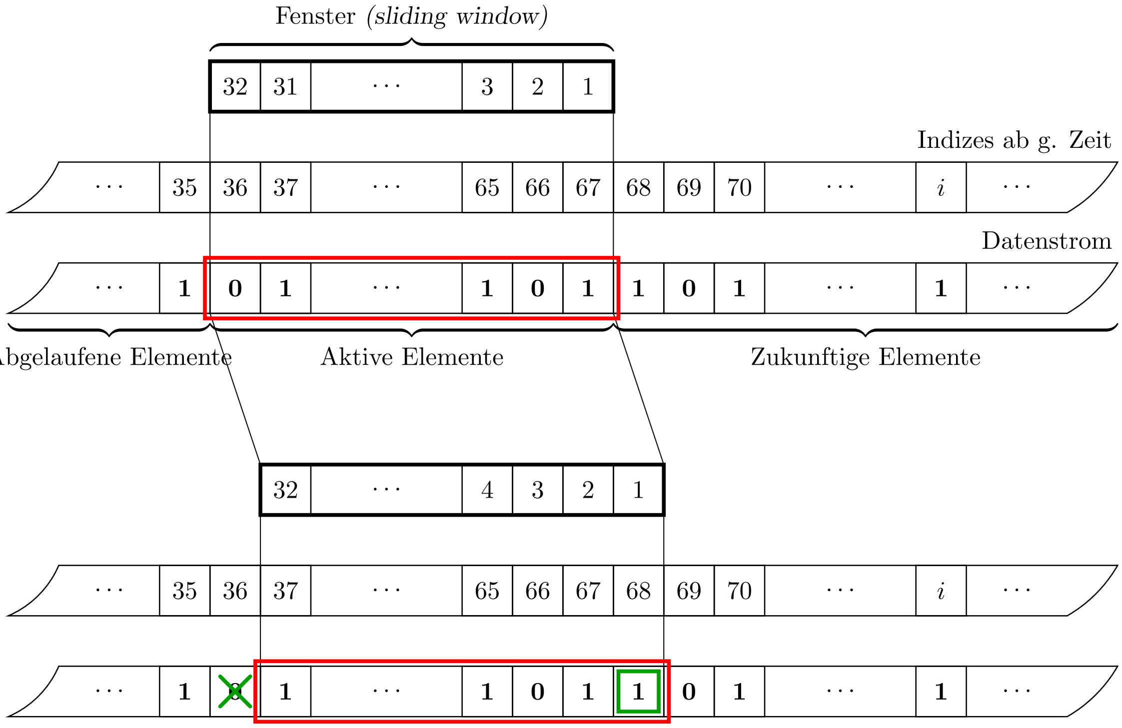
2.3.1 Verarbeitungsmodelle

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:



2.3.2 Exponenzielle Histogramme

Exponenzielle Histogramme sind eine grundlegende Datenstruktur zur statistischen Auswertung von Datenströmen. Die Datenstruktur geht auf Entwicklungen von Mayur Datar, Aristides Gionis, Piotr Indyk und Rajeev Motwani zurück.

Wir 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 2.33

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 $(1-\varepsilon)C \leq D \leq (1+\varepsilon)C$.

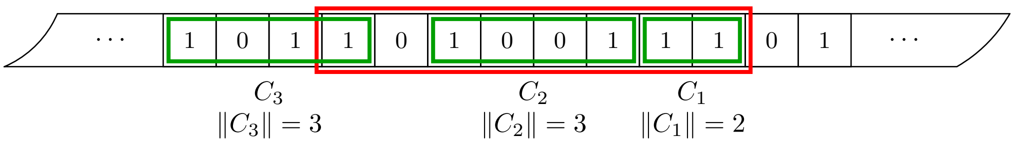
Um dieses Ziel zu erreichen, verwenden wir Histogramme bestehend aus den Klassen $C_1, \dots, C_m$, die die ankommenden Einsen in einem bestimmten Zeitintervall bilanzieren. Dabei haben wir für die zuletzt ankommenden Einsen detailliertere Informationen als über weiter in der Vergangenheit angekommenen 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 einführenden 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 2.34 *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 definieren $k =_{\text{def}} \lceil \frac{1}{\varepsilon} \rceil$. 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 2.34 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*
>
> $$\frac{n_j}{2 \cdot (1 + \sum_{i=1}^{j-1} n_i)} \leq \frac{1}{k} \qquad \textit{für alle } 1 \leq j \leq m.$$

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 2.13 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.

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

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

Abbildung 2.13: Der Algorithmus BasicCounting

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 2.35 *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 2.36 *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 2.35(4) und 2.35(5). Die Anzahl der Histogrammklassen ergibt sich aus Proposition 2.35(2) und 2.35(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. ■

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 2.37 *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 2.36 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

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

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.

2.3.3 Waves

Waves sind als Datenstruktur für Monitoralgorithmen von Phillip B. Gibbons und Srikanta N. Tirthapura eingeführt worden.

Wir betrachten wieder unser elementares Zählproblem für Bitstreams.

Bild zum Bitstreambeispiel

Der *Rang* einer Position im Bitstream ist die Anzahl der Einsen von Beginn des Bitstream bis zur aktuellen Position. Außerdem sei N die Größe des Fensters, ε die Approximationsgarantie und $k = \lceil \frac{1}{\varepsilon} \rceil$.

In einem abstrakten Sinn besteht eine *Wave* aus verschiedenen Leveln $i \in \{0, \ldots, L-1\}$ mit:

- $L =_{\text{def}} \lceil \log(2\varepsilon N) \rceil$,
- Level i enthält die $k+1$ jüngsten Positionen von Einsen in s, deren Rang ein Vielfaches von 2^i ist,
- P is die aktuelle Position,

- R ist Rang der jüngsten Eins.

Sollten irgendwelche Level weniger als $k + 1$ Positionen enthalten, so wird die kleinste Position auf 0 gesetzt.

Bild für Wave zum Bitstreambeispiel

Wir verwenden folgende Schätzregel:

- Ist $N \geq P$, so gebe den Rang R der maximalen Positionen der Wave aus.
- Sei $N < P$. Es sei $t =_{\text{def}} P - N + 1$, d.h. wir betrachten die Anzahl der Einsen in $s_t \dots s_P$. Weiterhin seien

$$\begin{aligned} p_1 &=_{\text{def}} \text{maximale Position} < t \\ p_2 &=_{\text{def}} \text{minimale Position} \geq t \\ r_1 &=_{\text{def}} \text{Rang von } p_1 \\ r_2 &=_{\text{def}} \text{Rang von } p_2 \end{aligned}$$

Als Schätzung η verwenden wir entsprechend einer Fallunterscheidung:

$$\eta =_{\text{def}} \begin{cases} 0, & \text{falls } p_2 \text{ nicht existiert} \\ R + 1 - r_2, & \text{falls } t = p_2 \text{ gilt} \\ R + 1 - \frac{r_1 + r_2}{2}, & \text{sonst} \end{cases}$$

Beispiel. In unserem Beispiel sei $P = 99$, $N = 24$ und $t = 76$. Damit gilt $p_1 = 72, r_1 = 36, p_2 = 77$ und $r_2 = 40$. Unsere Schätzung ist demnach $\eta = 51 - 38 = 13$. Als exakter Wert ergibt sich 12. Damit liegt unsere Schätzung klarerweise im gültigen Bereich: $\frac{2}{3} \cdot 12 = 0 \leq 13 \leq \frac{4}{3} \cdot 12 = 16$.

Proposition 2.38 *Für jeden Bitstream s liefert die Schätzregel für Waves eine bis auf den Faktor $1 \pm \varepsilon$ genauer Schätzung der Anzahl der Einsen unter den N zuletzt gelesenen Bitstream-Elementen.*

Beweis: Im Beweis folgen wir der Fallunterscheidung der Schätzregel:

- Ist $N \geq P$, dann ist Schätzwert η korrekt, da der gelesene Anteil des Bitstreams sich komplett im Fenster befindet.
- Ist $N < p$ und p_2 existiert nicht, d.h. $p_2 = 0$, dann gibt es keine Einsen im Fenster. Damit ist auch in diesem Fall die Schätzung korrekt.
- Ist $N < P$ und gilt $t = p_2$, so ist die Schätzung nach Definition korrekt.
- Es seien nun $N < P$ und $t < p_2$. Wir zeigen zunächst, dass p_1 existiert. Für Level i gilt: Der kleinste Rang r^i ist entweder 0 oder $r^i \leq R - k \cdot 2^i$. Wir betrachten den Level $L-1$. Enthält Level $L-1$ die Position 0, dann existiert p_1. Anderenfalls sei p^{L-1} diejenige Position, deren Rang r^{L-1} ist. Dann gilt:

  $$P - p^{L-1} \geq R - r^{L-1} \geq k \cdot 2^{L-1} \geq \frac{k}{2} \cdot 2\varepsilon N \geq n.$$

  D.h. $p^{L-1} \leq t$.

  Es sei j der kleinste Level, der p_1 enthält. Zur Fehlerbaschätzung halten wir fest, dass die exakte Anzahl D der Einsen im Fenster durch $R - r_2 + 1 \leq D \leq R - r_1$ eingegrenzt werden kann. Damit ergibt sich:

  $$\left| D - \left(R + 1 - \frac{r_1 + r_2}{2} \right) \right| \leq \frac{r_2 - r_1}{2} \leq 2^{j-1}$$

  Zur Beachtung sei erwähnt, dass $j > 0$ wegen $t < p_2$ gilt. Es sei r_3 kleinster Rang im Level $j-1$. Somit gilt $r_3 > r_1$ nach Definition von j. Außerdem gilt $r_3 \leq R - k \cdot 2^{j-1}$ sowie $r_3 \geq r_2$. Insgesamt erhalten wir für den relativen Fehler:

  $$\frac{\left| D - \left(R + 1 - \frac{r_1 + r_2}{2} \right) \right|}{D} \leq \frac{2^{j-1}}{R - r_2 + 1} \leq \frac{2^{j-1}}{R - r_3 + 1} \leq \frac{2^{j-1}}{k \cdot 2^{j-1} + 1} < \frac{1}{k} \leq \varepsilon$$

Damit ist die Proposition bewiesen. ■

Implementierung von Waves:

- Wir verwenden Modulo-Arithmetik für die Zähler; dabei sei N' die minimale Zweierpotenz $\geq 2N$.
- Positionen P' in der Vergangenheit mit $P' \leq P-$ werden entfernt; wir halten nur den größten Rang einer entfernten Position aufrecht, dieser entspricht r_1.
- Wir aktualisieren stets r_2.
- Für jedes Level verwenden wir eine Queue. Ist die Queue voll, dann entfernen wir für jedes eingefügte Paar (p, r) (d.h. bei $\mathsf{enQueue}(p, r)$) das letzte Element (d.h. wir führen $\mathsf{deQueue}()$ aus).

Algorithmus: WaveMonitor
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
  $P := (P+1)$ /* Rechnungen modulo N' */
  Falls $p \leq P - N$ für erstes Element (p, r) der Liste L, entferne (p, r) aus L und aus zugehöriger Queue und setze $r_1 := r$
  IF $s_i = 1$
    $R := (R+1)$
    $j := \max\{i \mid R \not\equiv 0 \bmod 2^i\}$
    IF Q_j.size() $= k+1$
      $(p, r) := Q_j$.deQueue()
      Entferne (p, r) aus Liste L
    Q_j.enQueue(P, r)
    Hänge (P, r) am Ende von L an
Gebe Schätzung gemäß Schätzregel ab

Abbildung 2.14: Der Algorithmus WaveMonitor

- Alle Positionen werden in einer Liste verwaltet mit Zeigern in Qeues.
- Jedes Paar (p, r) wird nur in der Queue für das maximale Level verwaltet.

Bild für Waveimplementierung zum Bitstreambeispiel

Komplexität von WaveMonitor:

- $O(\log \varepsilon N)$ Queues mit jeweils höchstens $k+1$ Elementen
- eine Liste mit insgesamt $O(k \log \varepsilon N)$ Elementen
- jedes Element (p, r) benötigt $\log N + \log\log N$ Bits; Positionen können als Differenzen gespeichert werden, dann $O(\log \varepsilon N)$ Bits (da Abstand höchsten bis $O(\varepsilon N)$ überprüft werden muss)

- $O(1)$ Bearbeitungszeit pro Bitstreamelement im schlechtesten Fall

Theorem 2.39 *Der Algorithmus* WaveMonitor *kann so implementiert werden, dass für einen Bitstream s zu jedem Zeitpunkt eine bis auf den Faktor $1 \pm \varepsilon$ genauer Schätzung der Einsen unter den N zuletzt gelesenen Elementen ausgegeben wird und*

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

Speicherplatz benötigt wird.

Bemerkung. 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.

2.4 Fallstudie: Klassifikation von IP-Paketen

Wir betrachten folgendes Szenario:

- Host 1 sendet Daten zu einem anderen Host 2
- Die Vermittlungsschicht von Host 1 bekommt ein (Daten-)Segment von der Transportschicht; das Segment wird verkapselt in ein IP-Datagramm, die Adresse des Zielhosts und weitere Felder werden in das Datagramm eingetragen; das Datagramm wird an den ersten Router auf dem Pfad in Richtung Zielhost gesendet
- Ein Host hat normalerweise nur eine Verbindungsleitung zum Netzwerk, d.h. der Host sendet das IP-Datagramm über diese Leitung
- Ein Router ist an mindestens zwei Verbindungsleitungen angeschlossen
- Die Grenzen zwischen Geräten und Leitung heißen Schnittstellen.
- Jede Schnittstelle hat eine eigene IP-Adresse (32 Bits in IPv4, 128 Bits in IPv6).

Bild zum kleinen Internet

2.4.1 Spezifischste Adresspräfixe

Eine wichtige Technik beim Netzwerkmanagement ist die *Routen-Aggregation*. Wir treffen eine Unterscheidung zwischen zwei Arten.

1. Einfache Routen-Aggregation:

- Wir koppeln Schnittstellen von Hosts/Routern ab
- Wir bestimmen für alle entstehenden Teilnetzwerke gemeinsame Adresspräfixe
- Bei einfacher Routen-Aggregation sind alle Adresspräfixe für ein Netzwerk unvergleichbar bezüglich der Präfixrelation
- Im Beispiel muss der Router R bei der Weiterleitung nur auf die ersten 24 Bits statt auf 32 Bit achten.

Bild zur einfachen Routen-Aggregation

2. Hierarchische Routen-Aggregation:

- Wir wenden einfache Routen-Aggregation rekursiv in Teilnetzwerken an
- Im Beispiel könnte jedes Teilnetzwerk über 223.1.0.0/16 erreicht werden; aber wie reagiert Router für Adressen mit 223.1.3.0/24?

Beispiel. Wir wollen uns einen exemplarischen aber typischen vereinfachten Inhalt eine Routingtabelle ansehen.

Zieladresse	Schnittstelle	Ausgang
223.1.1.0/24	223.1.1.4	A
223.1.2.0/24	223.1.2.9	B
223.1.4.0/24	223.1.4.27	C
223.1.0.0/16	223.1.3.13	D
199.2.1.0/24	223.1.3.13	D
0.0.0.0/0	223.1.255.1	E

Für ein IP-Datagramm mit der Zieladresse 223.1.4.2 wird der Ausgang C benutzt, obwohl auch Ausgang D und Ausgang E in Frage kommen würden. Ausgang C gehört jedoch zum längsten Präfix von 223.1.4.2, das in der Tabelle enthalten ist.

Wir betrachten folgende allgemeine Formulierung.

Definition 2.40 *Es seien $P \subseteq \{0,1\}^*$ mit $\varepsilon \in P$ und s ein Wort über $\Sigma = \{0,1\}$.*

1. *Ein Wort $p \in P$, das Präfix von s ist, heißt* spezifisches Präfix *von s in P.*
2. *Das spezifische Präfix von s in P mit maximaler Länge (unter allen spezifischen Präfixen) heißt* spezifischstes Präfix *von s in P.*

Damit ergibt sich als wesentliches Problem beim Routing: Für eine feste Menge $P \subseteq \{0,1\}^*$ bestimme für $s \in \{0,1\}^*$ das spezifischste Präfix von s in P (englisch: *longest prefix matching*).

2.4.2 Unibit-Tries

Unibit-Tries sind die einfachste Datenstruktur für das Longest-Prefix-Matching-Problem basierend auf binären Wörtern s.

Ein *Unitbit-Trie* ist ein Baum, bei dem jeder Knoten eine feste Anzahl von Datenfeldern sowie einen 0-Zeiger (Zeiger auf linkes Kind) und einen 1-Zeiger (Zeiger auf rechtes Kind) enthält.

Es sei T ein Unibit-Trie. Für $x \in T$ sei $\text{path}_T(x)$ der Weg von der Wurzel r zu x in T, repräsentiert durch eine Wort aus $\{0,1\}^*$, wobei 0 für linkes Kind und 1 für rechtes Kind steht.

Wir verlangen folgende Invariante für den Unibit-Trie T von P: Für alle Knoten x enthält der Teilbaum T_x von T mit x als Wurzel alle Wörter $p \in P$, so dass $\text{path}_T(x)$ ein Präfix von p ist.

Example. Die Wortmenge sei $P = \{101, 111, 11001, 1, 0, 100000, 100, 110\}$.

Bild zum Unitbit-Trie für Beispielmenge

Algorithmus: UNIBITFINDLMP
Eingabe: Unibit-Trie T, Wort $s = s_0 \dots s_n \in \{0,1\}^*$
Aufgabe: spezifischstes Präfix von s in T

```
v := T.root()
u := nil; p := ε; i := 0
WHILE ((u ≠ v) AND (i < n))
    u := v
    IF s_i = 0
        v := T.leftChild(v)
    ELSE
        v := T.rightChild(v)
    IF ((u ≠ v) AND (v.prefix() ≠ nil))
        p := v.prefix()
    i := (i + 1)
RETURN p
```

Abbildung 2.15: Der Algorithmus UNIBITFINDLMP

Die Zeitkomplexität des Algorithmus UNIBITFINDLMP ist offensichtlich $O(|s|)$.

Der Unibit-Trie T für eine Menge $P = \{p_1, \dots, p_n\}$ kann wie folgt aufgebaut werden: Für $i = 1, \dots, n$ bestimme den Knoten mit spezifischstem Präfix in $\{p_1, \dots, p_{i-1}\}$ (bereits in T enthalten) und füge p_i in den Teilbaum ein, der p_i enthalten müsste.

Proposition 2.41 *Es gibt einen Algorithmus, der einen Unibit-Trie zu gegebenem $P \subseteq \{0,1\}^*$ in $O(\sum_{p \in P} |p|)$ Schritten konstruiert.*

Beweis: Die Aussage ist klar. ■

Problematisch bei der bisherigen Formulierung der Standard-Unibit-Tries ist das Folgende: Die Anzahl der Knoten von T für $P = \{p_1, \dots, p_n\}$ ist im schlechtesten Fall $\Omega(n \cdot w)$, wobei

$w = \max_{1 \leq i \leq n} |p_i|$ ist (Übungsaufgabe). Als Ausweg bietet sich die Komprimierung von Pfaden an.

Bild zum Unitbit-Trie mit Pfadkomprimierung

Proposition 2.42 *Ein Unibit-Trie mit Pfadkomprimierung zu einer gegebenen Menge* $P = \{p_1, \ldots, p_n\} \subseteq \{0,1\}^*$ *hat maximal* $2n$ *Knoten.*

Beweis: Das leere Wort erzeugt einen Knoten. Jedes andere Wort erzeugt beim Einfügen maximal einen Knoten für die Pfadabkürzung und einen Knoten für das Präfix. ∎

2.4.3 Multibit-Tries

Multibit-Tries sind eine Erweiterung von Unibit-Tries nach Venkatachary Srinivasan und George Varghese.

Bei Unibit-Tries hat jeder Knoten höchstens $2^1 = 2$ Kinder. Wir relaxieren diese Bedingungen dahin gehend, dass ein Knoten $v \in T$ der *Schrittweite* b_v genau 2^{b_v} Kinder hat.

Wir unterscheiden zwischen zwei Arten von Multibit-Tries:

- *Multibit-Tries mit fester Schrittweite* k: Für alle inneren Knoten $v \in T$ gilt $b_v = k$.
- *Multibit-Tries mit variabler Schrittweite*: Es bestehen keine Einschränkungen an b_v für $v \in T$.

Es entsteht das Problem der *kontrollierten Präfixexpansion*. Im Konfliktfall wird die Regel angewendet, dass Expansion längerer Wörter vorgeht.

Beispiel. Wir betrachten einen Multibit-Tries der Schrittweite 3. Die Präfixmenge sei $P = \{101, 111, 11001, 1, 0, 100000, 100, 1100\}$. Kontrollierte Präfixexpansion ergibt folgende Transformation der Präfixmenge.

alter Präfix	neuer Präfix	
P1 = 101	101	
P2 = 111	111	
P3 = 11001	110010	
	110011	
P4 = 1	100	entspricht P7-Expansion
	101	entspricht P1-Expansion
	110	
	111	entspricht P2-Expansion
P5 = 0	000	
	001	
	010	
	011	
P6 = 100000	100000	
P7 = 100	100	
P8 = 1100	110000	
	110001	
	110010	entspricht P3-Expansion
	110011	entspricht P3-Expansion

Die Knoten von Multibit-Tries bestehen aus Arrays, deren Größe der Schrittweite des entsprechenden Knotens enstpricht.

Bild zum Multibit-Trie für Beispielmenge

Um in einem Multibit-Trie das spezifischste Präfix zu finden, gehen wir wie bei UNIBIT-FINTLMP vor, wobei in jedem Schritt immer b-Bit-Blöcke verglichen werden. Dies ergibt eine $O(2^b \cdot |s|)$ Laufzeit für MULTIBITFINDLMP in Multibit-Tries der Schrittweite b und für ein gegebenes Wort s.

Problematisch bei der Verwendung von Multibit-Tries mit fester Schrittweite ist, dass viele Arrays nur wenige Informationen enthalten.

Es sei T ein Multibit-Trie. Die *Größe* $|T|$ ist definiert als

$$|T| =_{\text{def}} \sum_{v \in T} 2^{b_v}.$$

Beispiel. In unserem Beispiel ergibt sich eine Größe von 24.

Optimierung der Multibit-Trie-Größe bei gegebener Höhe:

Es sei T ein Unibit-Trie zur Repräsentierung einer Menge P.

Es sei $v \in T$. Wir wollen v so expandieren, dass alle Wörter, die in T_v enthalten sind, in einem Array für v gefunden werden. Dann ist die Größe des Arrays für v gerade $2^{h(T_v)}$.

Nun verfahren wir rekursiv. Es ergibt sich folgende Rekursionsgleichung für die Größe $T^*(v,k)$ des minimalen Multibit-Tries T^* der Höhe k:

$$\begin{aligned}
T^*(v,0) &=_{\text{def}} 2^{h(T_v)} \\
T^*(v,k) &=_{\text{def}} \min_{b \in \{1,\dots,h(T_v)\}} \left(2^b + \sum_{u \in D_T^b(v)} T^*(u,k-1) \right)
\end{aligned}$$

Hierbei ist $D_T^b(v)$ die Menge der Nachfolger von v in T mit Abstand b zu v. Die Kinder von v haben den Abstand 1.

Bild zum optimalen Multibit-Trie für Beispielmenge

Algorithmus: MinimumMultibitTrie
Eingabe: Menge $P \subseteq \{0,1\}^*$, Parameter k
Aufgabe: Größe eines minimalen Multibit-Tries der Höhe k

```
Berechne Unibit-Trie T von P
Bestimme für alle Knoten v ∈ T die Höhen h(T_v) sowie ihre DFS-Eintrittsnummer
FOR i := 0 TO (k − 1)
    FOR v ∈ T in aufsteigender DFS-Reihenfolge
        S[v, i] := 2^{h(T_v)}
        IF i > 0
            FOR j := 1 TO h(T_v)
                S[v, i] := min(S[v, i], 2^j + H[v, i, j])
        u := T.parent(v)
        j := 1
        WHILE (u ≠ nil)
            H[u, i + 1, j] := (H[u, i + 1, j] + S[v, i])
            j := (j + 1)
            u := T.parent(u)
r := T.root()
S[r, k] := 2^{h(T_r)}
IF k > 0
    FOR j := 1 TO h(T_r)
        S[r, k] := min(S[r, k], 2^j + H[r, k, j])
RETURN S[r, k]
```

Abbildung 2.16: Der Algorithmus MinimumMultibitTrie

Proposition 2.43 *Der Algorithmus* MinimumMultibitTrie *kann so implementiert werden, dass ein minimaler Multibit-Trie für* $P \subseteq \{0,1\}^*$ *der Höhe* k *in* $O(k \cdot n \cdot w^2)$ *Schritten berechnet wird.*

2.4.4 Präfix-Automaten

Wir gehen über zu einer etwas detaillierteren Betrachtung der Problemstellung für das Longest-Prefix-Matching-Problem: Gegeben sind eine Mengen $P \subseteq \{0,1\}^*$ und eine Funktion $f : P \to H$, wobei H endlich ist. Für $s \in \{0,1\}^*$suchen wir $f(p)$ für das längste Präfix $p \in P$.

Beispiel. Es seien $P = \{00, 1, 101, 000\}$ und $H = \{2,3\}$. Die Funktion f ist gegeben durch:

$$\begin{array}{rrcl} f: & 00 & \mapsto & 2 \\ & 1 & \mapsto & 3 \\ & 101 & \mapsto & 2 \\ & 100 & \mapsto & 3 \end{array}$$

Bild zum Kompaktheit von Unibit-Tries

Allgemein versuchen wir deshalb eine Automatenrepräsentierung für P und f.

Ein Tupel $(A_1, \ldots, A_k)$ heißt genau dann *k-Partition* von Ω, wenn gilt:

- $A_i \subseteq \Omega$ für alle $i \in \{1, \ldots, k\}$
- $A_i \cap A_j = \emptyset$ falls $i \neq j$
- $\bigcup_{i=1}^{k} A_i = \Omega$

Die *charakteristische Funktion* $\chi_A : \Sigma^* \to \{1, \ldots, k\}$ einer k-Partition $A = (A_1, \ldots, A_k)$ ist definiert als

$$\chi_A(x) = i \iff_{\text{def}} x \in A_i.$$

Ein Tupel $(\Sigma, Q, \delta, q_0, P)$ heißt *Partitionsautomat* der Ordnung k, symbolisch DP$_k$A, falls gilt:

- Σ ist ein endliche Alphabet
- Q ist eine endliche Zustandsmenge
- $\delta : Q \times \Sigma \to Q$ ist die Übergangsfunktion
- $q_0 \in Q$ ist der Startzustand
- $P = (P_1, \ldots, P_k)$ ist eine k-Partition von Q

Es sei M ein DP$_k$A. Die von M erkannte Partition $L(M) = (L_1, \ldots, L_k)$ ist definiert als

$$L_i =_{\text{def}} \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in P_i\}.$$

Äquivalent ist auch $L_i = \{w \in \Sigma^* \mid \chi_P(\hat{\delta}(q_0, w)) = i\}$.

Wir verwenden Partitionsautomaten wie folgt: Für $P \subseteq \Sigma^*$ und $f : P \to H$ nehmen wir an, dass $f(w) \geq 2$ für $w \in P$ ist; d.h. 1 ist reserviert für „unbekannte“ Wörter.

Ein *Präfixautomat* für $P \subseteq \Sigma^*$ und $f : P \to H$ ist ein DP$_k$A $M = (\Sigma, Q, \delta, q_0, F)$, so dass $k = \|H\|$ und für $L(M) = (L_1, \ldots, L_k)$ und alle $i \geq 2$

$$L_i = \{w \in P \mid f(w) = i\}$$

gilt.

Beispiel. P und f seien wie gehabt.

Bild für Präfixautomat zur Beispielmenge

Der generische Algorithmus findLMP benötigt unabhängig vom gewählten Präfixautomaten $O(\|\Sigma\| \cdot n)$ Schritte zur Bestimmung von $f_F(p)$.

Wir haben damit die Freiheit einen Präfixautomaten minimaler Größe zu gegebenen P und f zu bestimmen. Dazu benötigen wir weitere Definitionen.

Algorithmus: findLMP
Eingabe: $\mathrm{DP}_k\mathrm{A}\ M = (\Sigma, Q, \delta, q_0, F)$, Wort $s = s_0 \ldots, s_{n-1} \in \Sigma^*$
Aufgabe: $f_F(p)$ für spezifischstes Präfix $p \in P$ (repräsentiert durch M)

```
q := q_0
h := χ_F(q)
FOR i := 1 TO (n − 1)
    q := δ(q, s_i)
    IF χ_F(q) ≠ 1
        h := χ_F(q)
RETURN h
```

Abbildung 2.17: Der Algorithmus findLMP

Es sei $M = (\Sigma, Q, \delta, q_0, F)$ ein $\mathrm{DP}_k\mathrm{A}$. Ein Zustand $q \in Q$ ist von $p \in Q$ genau dann *erreichbar*, wenn es ein Wort $w \in \Sigma^+ (= \Sigma^* \setminus \{\varepsilon\})$ mit $\hat{\delta}(p, w) = q$ gibt. Ein Zustand $q \in Q$ heißt *unerreichbar*, falls q von q_0 nicht erreichbar ist. Wir definieren eine Relation $\equiv_M \subseteq Q \times Q$ wie folgt:

$$q \equiv p \Longleftrightarrow_{\mathrm{def}} \text{ für alle } w \in \Sigma^* \text{ gilt } \chi_F(\hat{\delta}(q, w)) = \chi_F(\hat{\delta}(p, w))$$

Lemma 2.44 *Es sei $M = (\Sigma, Q, \delta, q_0, F)$ ein $\mathrm{DP}_k\mathrm{A}$ ohne unerreichbaren Zustände.*

1. *Die Relation $\equiv_M$ ist eine Äquivalenzrelation.*
2. *Ist $p \equiv_M q$, so gilt $\delta(p, a) \equiv_M \delta(q, a)$ für alle $a \in \Sigma$.*

Beweis: Wir beweisen die Aussagen einzeln.

1. Reflexivität und Symmetrie von $\equiv_M$ sind offensichtlich. Zum Nachweis der Transitivität seien $p \equiv_M q$ und $q \equiv_M r$ für $p, q, r \in Q$. Wir müssen $p \equiv_M r$ zeigen. Dies folgt jedoch sofort, da für $w \in \Sigma^*$

$$\begin{aligned} \chi_F(\hat{\delta}(p, w)) &= \chi_F(\hat{\delta}(q, w)) && (\text{wegen } p \equiv_M q) \\ &= \chi_F(\hat{\delta}(r, w)) && (\text{wegen } q \equiv_M r) \end{aligned}$$

gilt.

2. Es sei $w \in \Sigma^*$ und $a \in \Sigma$. Dann gilt:

$$\begin{aligned} \chi_F(\hat{\delta}(\delta(p, a), w)) &= \chi_F(\hat{\delta}(p, aw)) && (\text{nach Definition von } \hat{\delta}) \\ &= \chi_F(\hat{\delta}(q, aw)) && (\text{wegen } p \equiv_M q) \\ &= \chi_F(\hat{\delta}(\delta(q, a), w)) && (\text{nach Definition von } \hat{\delta}) \end{aligned}$$

Mithin ist $\delta(p, a) \equiv_M \delta(q, a)$.

Damit ist das Lemma bewiesen. ■

Ein klassisches Resultat der Automatentheorie ist der Satz von Myhill und Nerode. Mit Hilfe dieses Satzes kann ein Verfahren angegeben werden, um zu einem gegebenen deterministischen endlichen Automaten einen minimalen deterministischen Automaten zu bestimmen, der bis auf Umbenennung der Zustände eindeutig bestimmt ist. Benjamin Hummel übertrug den Satz auf den Partitionsfall. Wir geben das Resultat ohne Beweis an.

Theorem 2.45 (Myhill & Nerode 1958, Hummel 2006) *Es sei $M = (\Sigma, Q, \delta, q_0, F)$ ein DP$_k$A ohne unerreichbaren Zustände. Dann ist der DP$_k$A*

$$M' = (\Sigma, \{[q]_{\equiv_m} | q \in Q\}, \delta', [q_0]_{\equiv_M}, F)$$

mit $\delta'([q]_{\equiv_M}, a) = [\delta(q,a)]_{\equiv_M}$ der bis auf Isomorphie eindeutig bestimmte minimale DP$_k$A mit $L(M) = L(M')$.

In unserem Kontext tritt eine Besonderheit für die Automaten auf: Die Komponenten L_i mit $i \geq 2$ sind alle endlich. Dies werden wir im Folgenden ausnutzen.

Es sei $M = (\Sigma, Q, \delta, q_0, F)$ ein DP$_k$A. Für $q \in Q$ definieren wir

$$\begin{aligned} W(q) &=_{\text{def}} \{\ w \in \Sigma^* \mid \chi_F(\hat{\delta}(q,w)) \neq 1\ \} \\ h(q) &=_{\text{def}} \sup\{\ 1 + |w| \mid w \in W(q)\} \end{aligned}$$

Es sollte beachtet werden, dass $h(q) = 0$ gilt, falls $W(q)$ leer ist, und dass $h(q) = \infty$ gilt, falls $\|W(q)\| = \infty$ ist.

Proposition 2.46 *Es sei $M = (\Sigma, Q, \delta, q_0, F)$ ein DP$_k$A. Es seien $p, q \in Q$, $p \neq q$, Zustände, so dass q von p aus erreichbar ist. Dann gilt $h(q) \leq h(p)$. Die Gleichheit gilt nur dann, falls $h(q) = \infty$ oder $h(p) = 0$ ist.*

Beweis: Es sei $w \in \Sigma^*$ so, dass $\hat{\delta}(p, w) = q$. Dann gilt für $v \in W(q)$ stets $wv \in W(p)$. Damit gilt $h(p) \geq h(q)$. Ist $\|W(q)\| = \infty$, so ist $\|W(p)\| = \infty$, d.h. $h(q) = \infty$ und $h(p) = \infty$. Ist andererseits $\|W(p)\| = 0$, so ist $\|W(q)\| = 0$, da q von p aus erreichbar ist. Somit gilt $h(p) = 0$ und $h(q) = 0$. ■

Es sei weiterhin $M = (\Sigma, Q, \delta, q_0, F)$ ein DP$_k$A. M heißt genau dann *azyklisch*, wenn es ein $\ell \in \mathbb{N}$ gibt, so dass für $L(M) = (L_1, \ldots, L_k)$ gilt: $L_i \cap \Sigma^{\geq \ell} = \emptyset$ für alle $i \in \{2, \ldots, k\}$. Äquivalent können wir auch azyklische Automaten M mittels der Bedingung $h(q_0) < \infty$ definieren.

Beispiel. Für unser Beispiel P und f ist der Automat azyklisch. Die Höhen sind:

$$\begin{array}{llll} h(q_0) = 4 & h(q_1) = 2 & h(q_2) = 3 & h(q_3) = 1 \\ h(q_4) = 2 & h(q_5) = 0 & h(q_6) = 0 & \end{array}$$

Algorithmus: ComputeHeight
Eingabe: azyklischer DP$_k$A $M = (\Sigma, Q, \delta, q_0, F)$, Zustand $q \in Q$
Aufgabe: $h(q)$ (gespeichert in globalem Array)

```
visited(q) := true
IF q ∈ F_1
    h(q) := 0
ELSE
    h(q) := 1
FOR a ∈ Σ
    IF NOT (visited(δ(q,a)))
        h(δ(q,a)) := ComputeHeight(M, δ(q,a))
    h(q) := max(h(q), 1 + h(δ(q,a)))
```

Abbildung 2.18: Der Algorithmus ComputeHeight

Lemma 2.47 *Es sei $M = (\Sigma, Q, \delta, q_0, F)$ ein DP$_k$A ohne unerreichbaren Zustände. M is genau dann azyklisch, wenn $h(q) = 0$ für alle nicht-azyklischen Zustände $q \in Q$ gilt.*

Beweis: Wir zeigen beide Richtungen der Aussage separat.

($\Rightarrow$) Sei M azyklisch. Es sei $q \in Q$ ein Zustand mit $\hat{\delta}(q, w) = q$ für ein geeignetes Wort $w \in \Sigma^+$. Angenommen $\|W(q)\| > 0$. Somit gibt es ein Wort $v \in W(q)$. Dann gilt $W(q) \supseteq \{\, w^i v \mid i \in \mathbb{N} \,\}$. Mithin ist $\|W(q)\| = \infty$. Da q jedoch von q_0 aus erreichbar ist, folgt $h(q_0) = \infty$ nach Proposition 2.46. Dies ist ein Widerspruch zur Azyklizität. Also ist $\|W(q)\| = 0$ und folglich ist $h(q) = 0$.

($\Leftarrow$) Der Beweis der Aussage geht über die Kontraposition. Sei M also nicht azyklisch, d.h. $\|W(q_0\| = \infty$. Dann gibt es ein $q \in Q \setminus F_1$ mit $\|\{w \in \Sigma^* | \hat{\delta}(q_0, w) = q\}\| = \infty$. Sei $w \in \Sigma^*$ ein Wort mit $|w| > \|Q\|$. Somit gibt es Wörter $x, y, z \in \Sigma^*$ mit $|y| > 0$, so dass $xyz = w$ und $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, xy)$. Es folgt $\hat{\delta}(\hat{\delta}(q_0, x), y) = \hat{\delta}(q_0, xy) = \hat{\delta}(q_0, x)$. Damit ist der Zustand $\hat{\delta}(q_0, x)$ azyklisch und q ist erreichbar von $\hat{\delta}(q_0, x)$. Außerdem gilt $\|W(\hat{\delta}(q_0, x))\| = \infty$. Folglich ist $h(\hat{\delta}(q_0, x)) > 0$.

Damit ist das Lemma bewiesen. ∎

Der Algorithmus ComputeHeight (siehe Abbildung 2.18) bestimmt die Höhen in einem azyklischen Automaten in der Zeit $O(\|\Sigma\| \cdot \|Q\|)$.

Proposition 2.48 *Es sei $M = (\Sigma, Q, \delta, q_0, F)$ ein azyklischer DP$_k$A. Es seien $p, q \in Q$.*

1. *Ist $h(p) = h(q) = 0$, so gilt $p \equiv_M q$.*
2. *Ist $h(p) \neq h(q)$, so gilt $p \not\equiv_M q$.*

Algorithmus: MinimalAcyclicDP$_k$A
Eingabe: azyklischer DP$_k$A $M = (\Sigma, Q, \delta, q_0, F)$
Aufgabe: minimaler DP$_k$A M' mit $L(M') = L(M)$

```
1.   Eliminiere alle unerreichbaren Zustände
2.   Berechne die Höhen der Zustände mittels ComputeHeight
3.   Q:=Zustandsmenge nach Verschmelzung aller Zustände in { q ∈ Q | h(q) = 0 }
4.   FOR i := 1 TO h(δ(q_0))
5.       H[i] := { q ∈ Q | h(q) = i }
6.       FOR q ∈ H[i]
7.           I(q) := (χ_F(q), δ(q, a_1), ..., δ(q, a_m))
8.       Q:=Zustandsmenge nach Verschmelzung aller p, q ∈ H[i] mit I(p) = I(q)
9.   RETURN (Σ, Q, δ', q'_0, F), wobei δ', q'_0 und F' durch die berechnete Zustandsmenge
10.                Q induziert werden
```

Abbildung 2.19: Der Algorithmus MinimalAcyclicDP$_k$A

Beweis: Offensichtlich. ■

Für die Komplexität des Algorithmus MinimalAcyclicDP$_k$A ergibt sich folgendes Bild:

- Zeile 1 benötigt $O(\|\Sigma\| \cdot \|Q\|)$ Schritte
- Zeile 2 benötigt $O(\|\Sigma\| \cdot \|Q\|)$ Schritte
- Zeile 3 benötigt $O(\|Q\|)$ Schritte
- Zeilen 4–8 benötigen, falls Zeile 8 in der Laufzeit $O(\|\Sigma\| \cdot \|H[i]\|)$ realisiert werden kann, $O\left(\sum_{i=1}^{h(q_0)} \|\Sigma\| \cdot \|H[i]\|\right) = O(\|\Sigma\| \cdot \|Q\|)$

Insgesamt ergibt sich also eine Laufzeit von $O(\|\Sigma\| \cdot \|Q\|)$ unter der Voraussetzung der effizienten Realisierbarkeit von Zeile 8. Dies wird von dem Algorithmus SortState (siehe Abbildung 2.20) gewährleistet, der BucketSort als Sortierverfahren verwendet.

Die Korrektheit von SortStates folgt aus der Stabilität von BucketSort. Die Analyse der Komplexität ergibt eine Laufzeit von $O(k + \|H[i]\| + \|\Sigma\| \cdot \|H[i]\|) = O(k + \|\Sigma\| \cdot \|H[i]\|)$. Ist k fest (und das ist natürlich bei gegebenem Router stets der Fall), so entspricht die Laufzeit genau der geforderten.

Theorem 2.49 *Der Algorithmus* MinimalAcyclicDP$_k$A *kann so implementiert werden, dass ein minimaler äquivalenter azyklischer DP$_k$A zu gegebenem DP$_k$A über einem Alphabet der Größe m und mit n Zuständen in der Zeit $O(n \cdot m)$ bestimmt wird.*

Beweis: Die Aussage folgt aus der angegebenen Herleitung und Analyse. ■

Algorithmus:	SortStates
Eingabe:	azyklischer DP$_k$A $M = (\Sigma, Q, \delta, q_0, F)$, Zustandsmenge $H[i]$ zusammen mit den Tupeln $I(q)$ wie in Zeile (8) von MinimalAcyclicDP$_k$A
Aufgabe:	Sortierung von $H[i]$, so dass p und q mit $I(p) = I(q)$ benachbart sind

1. Sortiere $H[i]$ aufsteigend nach $\chi_F(q)$ (mittels BucketSort)
2. FOR $a \in \Sigma$
3. $j := 1$
4.   FOR $q \in H[i]$ (in sortierter Reihenfolge)
5. $L[\delta(q, a)] := j$
6. $j := (j + 1)$
7.   Sortiere $H[i]$ nach $L[\delta(q, a)]$ (mittels BucketSort)

Abbildung 2.20: Der Algorithmus SortStates

Netzwerkanalyse 3

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

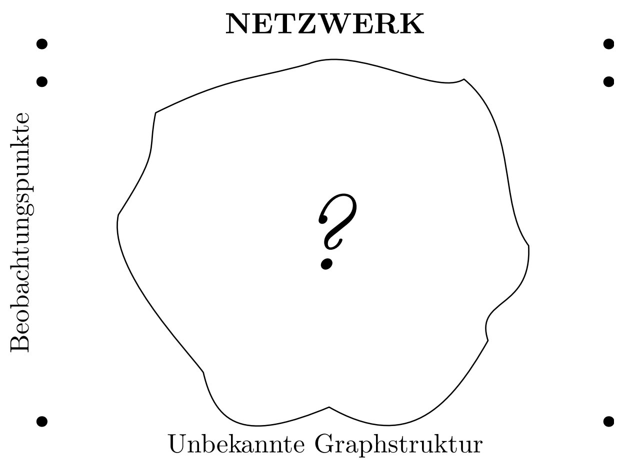
Hierbei stehen vor allem Inferenzprobleme im Mittelpunkt: Gegeben eine Menge von Beobachtungen, Daten usw. usf., welche Beziehungen bestehen zwischen den Netzwerkelementen, die konsistent mit den Beobachtungen, Daten usw. usf. sind?

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 Dimensionierung und Positionierung von Monitorsystemen

Eine wichtige erste Frage ist, an welchen Stellen im Netzwerk Daten, Beobachtungen etc. genommen werden sollten. 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

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

3.1.1 Trennende und erkennende Mengen

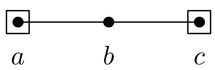
Wir führen eine beispielhafte Analyse für den Kantenzusammenhang in Graphen durch.

Konkretisierung dazu unser Szenario:

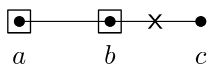
- Monitore sitzen in Knoten aus einer Menge D eines Graphen $G = (V, E)$
- Monitore kommunizieren in periodischen Abständen untereinander
- Gibt es Monitore $u, v \in D$, so dass u und v keine gemeinsame Kommunikation mehr aufrecht erhalten können (d.h. es gibt keinen Pfad zwischen u und v), so zeigen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

Nach diesem Szenario ist klar: Gibt es Knoten $u, v \in D$, die ein Versagen anzeigen, so ist G tatsächlich nicht mehr zusammenhängend. Das Ziel ist somit: 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

Im Folgenden sei erschwerend angenommen, dass wir die Netzwerktopologie nicht kennen!

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 $\|Z\| \leq k$ Knoten $u, v \in D'$ in unterschiedlichen Zusammenhangskomponenten von $(V, E \setminus Z)$ existieren, 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 $\|D'\| \geq (1 - \varepsilon) \cdot n$ beispielsweise für den vollständigen Graphen K^n.

3.1.2 Eine einfache Analyse für den Kantenzusammenhang

Wir analysieren die Strategie, die Menge D zufällig zu wählen.

Proposition 3.2 *Es seien $\varepsilon > 0$, $\delta > 0$ und $k \in \mathbb{N}$. Es sei $G = (V, E)$ ein ungerichteter Graph. Eine 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 $D \subseteq V$ der Kardinalität $\ell =_{\text{def}} \|D\|$ zufällig im Graphen G gewählt. Es seien $Z_1, \dots, Z_t$ 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, so dass $\|R_j\| \geq \varepsilon \cdot n$ gilt und R_j von $V \setminus R_j$ getrennt ist (d.h. es gibt keinen Pfad von R_j nach $V \setminus R_j$). Dann gilt

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

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.

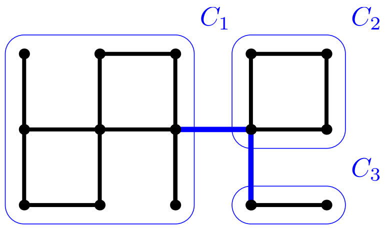
3.1.3 Kleinberg's Technik für den Kantenzusammenhang

Das Ziel für den weiteren Verlauf ist vorgegeben: Wir wollen 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 $C_1^Z, \dots, C_{\ell_Z}^Z$ bezeichnen wir die Zusammenhangskomponenten von $(V, E \setminus Z)$. Die Menge $S \subseteq V$ heißt genau dann *trennbar mit k Kanten*, wenn es ein $Z \subseteq E$ gibt mit $\|Z\| \leq k$, so dass eine Menge $I \subseteq \{1, \dots, \ell_Z\}$ mit $S = \bigcup_{i \in I} C_i^Z$ existiert.

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 $G = (V, E)$:

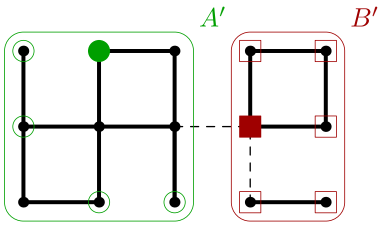
$$\mathcal{S}_k(G) =_{\text{def}} \{S \subseteq V \mid S \text{ ist trennbar mit } k \text{ Kanten}\}.$$

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 $A_1, \dots, A_r$ und $B_1, \dots, B_s$ die Zusammenhangskomponenten von A und B. Wir definieren Mengen A' und B' vermöge

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

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:

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

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

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

Damit ist die Proposition bewiesen. ■

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 genau dann* ε-Netz *für* $(\Omega, \mathcal{F})$, *wenn für alle* $X \in \mathcal{F}$ *mit* $\|X\| \geq \varepsilon\|\Omega\|$ *gilt* $N \cap X \neq \emptyset$.

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 *der Größe* $f(d, \varepsilon, \delta)$ *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, wie dies zu zeigen wäre, ist die folgende: Für eine hinreichend große Knotenmenge $A \subseteq V$ betrachten wir 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. Es gibt also 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 $p_1, \dots, p_\ell$, 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 $p_1, \dots, p_r$ die Pfade nach Induktionsvoraussetzung. Dann sind $p_1, \dots, p_r, \{v, w\}$ 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 $p_1, \dots, p_r$ die Pfade für H' und T' nach Induktionsvoraussetzung. O.B.d.A. ende Pfad $p_r = (u_1, \dots, u_t, w)$ im Knoten w. Dann sind $p_1, \dots, p_r, (u_1, \dots, u_t, w, v)$ 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:

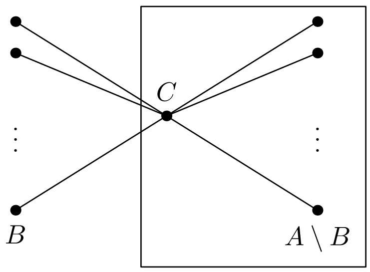
$$a_i \in S \Longleftrightarrow a_{i+k+1} \in S$$

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 $\text{VC-Dim}(V, \mathcal{S}_k(G)) \leq 2k+1$ 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 (wobei wir B durch $A \setminus B$ ersetzen).

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

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.

3.1.4 Erweiterungen von Kleinberg's Technik

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:

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

Damit ist die Proposition bewiesen. ■

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 $G \setminus Z = (V \setminus Z, E \setminus Z)$ getrennt 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 $S \subseteq V$ ist trennbar durch k Elemente, falls $Z \subseteq V \cup E$ mit $\|Z\| \leq k$ existiert, so dass S die 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).

3.2 Rekonstruktion der Netzwerktopologie

Wir gehen zur Untersuchung von Netzwerkrekonstruktionsproblemen über. Diese Probleme haben die folgende Form: Gegeben die Informationen *aller* Beobachtungspunkte, wie sieht die Graphenstruktur im unbekannten Netzwerkbereich aus?

3.2.1 Graphenrealisierung von Gradfolgen

Zunächst besteht unser Ziel darin, dass wir aus beobachteten Gradfolgen (Definition später) Rückschlüsse darauf ziehen wollen, wie der gegebene Graph aussehen könnte.

Notationen. Es sei $G = (V, E)$ ein (einfacher) gerichteter Graph. Für $v \in V$ definieren wir folgende Größen:

- $N^-(v) =_{\text{def}} \{\ u \in V \mid (u, v) \in E\ \}$ (Vorgänger)
- $N^+(v) =_{\text{def}} \{\ u \in V \mid (v, u) \in E\ \}$ (Nachfolger)
- $N(v) =_{\text{def}} N^-(v) \cup N^+(v)$
- $\deg^-(v) =_{\text{def}} \|N^-(v)\|$ (Eingangsgrad)
- $\deg^+(v) =_{\text{def}} \|N^+(v)\|$ (Ausgangsgrad)
- $\deg(v) =_{\text{def}} \deg^-(v) + \deg^+(v)$ (Grad; beachte: i.A. $\deg(v) \neq \|N(v)\|$)

Ist G ungerichtet, so definieren wir für $v \in V$:

- $N(v) =_{\text{def}} \{\ u \in V \mid \{u, v\} \in E\ \}$
- $\deg(v) =_{\text{def}} \|N(v)\|$

Wir verwenden weiterhin stets für $G = (V, E)$:

- $n =_{\text{def}} \|V\|$
- $m =_{\text{def}} \|E\|$

Proposition 3.16

1. *Jeder ungerichtete Graph $G = (V, E)$ ohne Schleifen mit mindestens zwei Knoten enthält zwei Knoten $u, v \in V$ mit $\deg(u) = \deg(v)$.*
2. *Für alle (gerichteten und ungerichteten) Graphen $G = (V, E)$ gilt*

$$\sum_{v \in V} \deg(v) = 2\|E\|.$$

Beweis: Wir beweisen die Aussagen einzeln.

1. Angenommen in G gibt es keinen Knoten v mit $\deg(v) = 0$. Dann gilt für alle $v \in V$: $1 \leq \deg(v) \leq n-1$. Damit gibt es unter n Knoten zwei Knoten mit gleichem Grad (Schubfachschluss). Angenommen es gibt genau einen Knoten v mit $\deg(v) = 0$. Somit gilt für die restlichen $n-1$ Knoten $u \in V \setminus \{v\}$: $1 \leq \deg(u) \leq n-2$. Der Rest ist nun analog zum ersten Fall.

2. Für gerichtete Graphen $G = (V,E)$ gilt $\sum_{v \in V} \deg^-(v) = \|E\| = \sum_{v \in V} \deg^+(v)$. Damit ergibt sich $\sum_{v \in V} \deg(v) = \sum_{v \in V} \deg^-(v) + \deg^+(v) = 2\|E\|$. Für ungerichtete Graphen ist jede eingehende auch eine ausgehende Kante; ersetze $\{u,v\} \in E$ durch (u,v) und (v,u) und wende Aussage für gerichtete Graphen an.

Damit ist die Proposition bewiesen. ■

Definition 3.17 *Es sei $G = (V,E)$ ein gerichteter oder ungerichteter Graph mit $V = \{v_1 \ldots, v_n\}$.*

1. *Ist G gerichtet, so heißt die Folge*

$$D(G) =_{\text{def}} ((\deg^-(v_1), \deg^+(v_1)), \ldots, (\deg^-(v_n), \deg^+(v_n)))$$

   Gradfolge *von G.*

2. *Ist G ungerichtet, so heißt die Folge*

$$D(G) =_{\text{def}} (\deg(v_1), \ldots, \deg(v_n))$$

   Gradfolge *von G.*

Definition 3.18 *Es sei $D = (d_1, \ldots, d_n)$ eine Folge natürlicher Zahlen. Die* konjugierte *Zahlenfolge $D^* = (d_1^*, \ldots, d_n^*)$ ist definiert als*

$$d_i^* =_{\text{def}} \|\{j \in \{1, \ldots, n\} \mid d_j \geq i\}\|.$$

Beispiel. Wir betrachten die Zahlenfolge $D = (4,3,3,2,2,2,2)$. Das zugehörige Ferrers-Diagramm sind dann wie folgt aus:

	d_1^*	d_2^*	d_3^*	d_4^*	$\sum$
d_1	•	•	•	•	4
d_2	•	•	•		3
d_3	•	•	•		3
d_4	•	•			2
d_5	•	•			2
d_6	•	•			2
d_7	•	•			2
$\sum$	7	7	3	1	18

Als Beobachtung halten wir fest: Es gilt stets $\sum_{i=1}^{n} d_i = \sum_{i \geq 1} d_i^*$.

Beispiele für natürliche Gradfolgen.

- Power-Law-Folgen (Zipf-Verteilungen) sind Gradfolgen D mit $d_i^* \simeq i^{-\alpha}$ und $\alpha > 0$. Wichtige Eigenschaften: Skalenfreiheit (Durchschnittsgrad für $\alpha > 2$ konstant, d.h. unabhängig von Anzahl der Knoten), kleiner Durchmesser ($\log n$) im Erwartungswert. Power-Law-Folgen können beobachtet werden in der Internetkonnektivität, WWW-Hyperlinkstruktur, Einwohnerverteilungen.
- Log-Normal-Verteilungen
- doppelte Pareto-Verteilungen

Wir untersuchen die Frage: Welche Folgen natürlicher Zahlen sind Gradfolgen von Graphen?

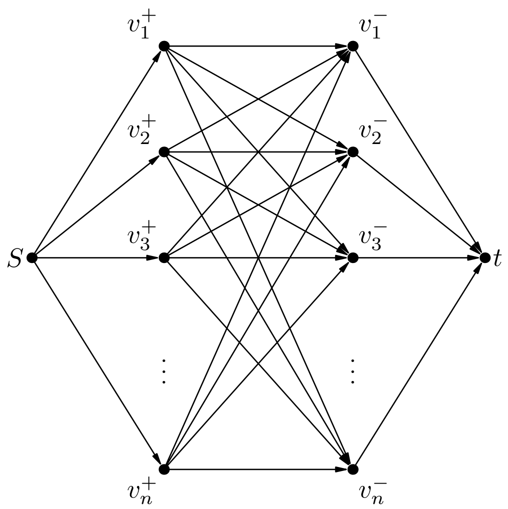
3.2.2 Der Satz von Gale und Ryser

Für gerichtete Graphen gibt der folgende Satz eine vollständige Charakterisierung der realisierbaren Gradfolgen.

Theorem 3.19 (Gale & Ryser 1957) *Eine Folge* $((a_1, b_1), \ldots, (a_n, b_n))$ *von Paaren natürlicher Zahlen mit* $a_1 \geq \cdots \geq a_n$ *und* $a_j, b_j \leq n$ *ist die Gradfolge eines gerichteten Graphen* $G = (V, E)$ *(mit Schleifen) genau dann, wenn folgende Bedingungen erfüllt sind:*

$$\begin{aligned} \sum_{i=1}^{n} a_i &= \sum_{j \geq 1} b_j^* \quad \left(= \sum_{j=1}^{n} b_j \right) \\ \sum_{i=1}^{k} a_i &\leq \sum_{j=1}^{k} b_j^* \quad \left(= \sum_{j=1}^{n} \min(b_j, k) \right) \qquad \textit{für alle } k \in \{1, \ldots, n-1\} \end{aligned}$$

Beweis: (Idee) Wir betrachten das folgende Flussnetzwerk:



Für einen gerichteten Graphen $G = (V, E)$ mit Kantengewichten $c : E \to \mathbb{R}_+$ ist ein Fluss f definiert als eine Abbildung $f : E \to \mathbb{R}$ mit $0 \leq f(e) \leq c(e)$ für alle $e \in E$ und $\sum_{u \in N^-(v)} f(u, v) = \sum_{u \in N^+(v)} f(v, u)$. Der Wert eines Flusses f in unserem Flussnetzwerk ist definiert als $|f| = \sum_{u \in N^+(s)} f(s, u)$. Der maximale Fluss f, der alle zu s und t Kanten saturiert, hat den Wert

$$|f| = \sum_{j=1}^{n} b_j = \sum_{j=1}^{n} a_j.$$

Der eigentliche Beweis, wann solch ein maximaler Fluss existiert, ergibt dann die Charakterisierung in der Aussage des Satzes. ■

3.2.3 Der Algorithmus von Havel und Hakimi

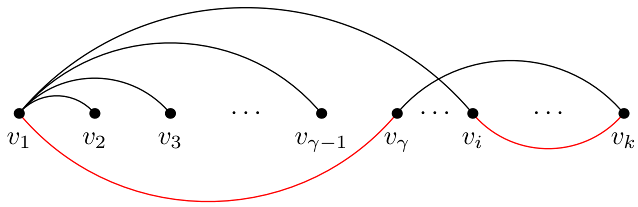
Wir betrachten nunmehr Konstruktionen für ungerichteten Graphen.

Lemma 3.20 *Es sei $D = (d_1, \ldots, d_n)$ eine Folge natürlicher Zahlen mit $d_1 \geq \cdots \geq d_n$ und $d_1 \leq n - 1$. Es bezeichne $\mathcal{G}_D$ die Menge aller ungerichteten Graphen $G = (V, E)$ mit $V = (v_1, \ldots, v_n)$ und $\deg(v_i) = d_i$. Ist $\mathcal{G}_D \neq \emptyset$, so existiert ein Graph $G = (V, E) \in \mathcal{G}_D$ mit $N(v_1) = \{v_2, \ldots, v_{d_1+1}\}$.*

Beweis: Ist $d_1 = n-1$, so erfüllt jeder Graph in $\mathcal{G}_D$ die Eigenschaft. Es sei also $d_1 < n-1$. Wir definieren für alle $G \in \mathcal{G}_D$:

$$\gamma(G) =_{\text{def}} \min\{\ j \mid (v_1, j) \notin E\ \}.$$

Es sei $G^* \in \mathcal{G}_D$ ein Graph mit $\gamma(G^*) \geq \gamma(G)$ für alle $G \in \mathcal{G}_D$. Wir müssen zeigen, dass $\gamma(G^*) = d_1 + 2$ gilt. Angenommen $\gamma = \gamma(G^*) < d_1 + 2$.



Dann gilt $\|N(v_1) \cap \{v_1, \ldots, v_\gamma\}\| \leq d_1 - 1$. Damit gibt es $i > \gamma$ mit $\{v_1, v_i\} \in E^*$. Nach Voraussetzung gilt $d_\gamma \geq d_i$. Da $v_1 \in N(v_i) \setminus N(v_\gamma)$ gilt, existiert ein $v_k \in N(v_\gamma) \setminus N(v_i)$. Wir betrachten nun den Graphen $G' = (V, E')$ mit (siehe auch die Abbildung)

$$E' =_{\text{def}} (E \setminus \{\ \{v_1, v_i\}, \{v_\gamma, v_k\}\ \}) \quad \cup \quad \{\ \{v_1, v_\gamma\}, \{v_i, v_k\}\ \}.$$

Dann gilt $G' \in \mathcal{G}_D$ und $\gamma(G') > \gamma(G^*)$. Dies ist ein Widerspruch zur Maximalität von $\gamma(G^*)$. ■

Algorithmus:	HAVEL-HAKIMI
Eingabe:	Array $D[1 \dots n]$ mit natürlichen Zahlen
Aufgabe:	Adjazenzmatrix A_G für Graphen $G = (V, E)$ mit Gradfolge D

/* Verwenden Maximum Priority Queues Q_1 und Q_2 mit Ordnung bezüglich $D[i]$ */

```
FOR i := 1 TO n
    Q1.insertItem((i, D[i]))
WHILE Q1 ist nicht leer
    (v, d_v) := Q1.extractMax
    IF ||Q1|| ≥ d_v
        FOR j := 1 TO d_v
            (u, d_u) := Q1.extractMax
            A_G[v, u] := 1
            A_G[u, v] := 1
            IF d_u ≥ 2
                Q2.insertItem((u, d_u − 1))
        Q1 :=merge(Q1, Q2)
    ELSE
        Gebe „Konstruktion nicht möglich“ zurück
Gebe A_G zurück
```

Abbildung 3.1: Der Algorithmus von Havel und Hakimi

Korollar 3.21 (Havel 1955, Hakimi 1962) *Eine Folge $(d_1, \dots, d_n)$ mit $d_1 \geq \cdots \geq d_n$ und $d_1 \leq n-1$ ist genau dann die Gradfolge eines ungerichteten Graphen (ohne Schleifen), wenn $(d_2 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n)$ Gradfolge eines solchen Graphen ist.*

Das Korollar kann direkt in einen Algorithmus übertragen werden (siehe Algorithmus 3.1).

Beispiel. Wir betrachten wieder die Folge $D = (4, 3, 3, 2, 2, 2, 2)$-

Erste Runde:

Knoten	1	2	3	4	5	6	7
Grad (alt)	4	3	3	2	2	2	2
Grad (neu)	-	2	2	1	1	2	2

Zweite Runde:

Knoten	2	3	6	7	4	5
Grad (alt)	2	2	2	2	1	1
Grad (neu)	-	1	1	2	1	1

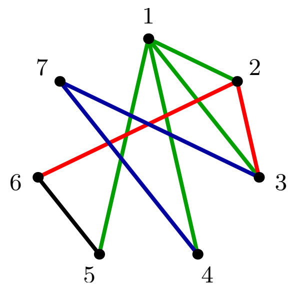
Dritte Runde:

Knoten	7	3	4	5	6
Grad (alt)	2	1	1	1	1
Grad (neu)	-	-	-	1	1

Vierte Runde:

Knoten	5	6
Grad (alt)	1	1
Grad (neu)	-	-

Folgender Graph ergibt sich:



Theorem 3.22 *Der Algorithmus von Havel und Hakimi kann so implementiert werden, dass er für eine Folge $(d_1, \ldots, d_n)$ mit $d_1 \geq \cdots \geq d_n$ und $d_1 \leq n-1$ in der Zeit (ohne Ausgabe)*

$$O\left((n+m)\log n\right)$$

einen Graph $G = (V, E)$ mit $\|V\| = n$ und $\|E\| = m = \frac{1}{2}\sum_{i=1}^{n} d_i$ konstruiert.

Beweis: Wir verwenden Binomial Queues als Implementierung der Priority Queues. Dann erhalten wir als Laufzeiten

- Zeilen 1 und 2: $O(n)$
- Zeilen 3–14:

$$\begin{aligned}
&\sum_{i=1}^{n}\Big(\underbrace{T_{\mathsf{extractMax}}}_{O(\log n)} + \sum_{j=1}^{\deg(v_i)} \big(\underbrace{T_{\mathsf{extractMax}}}_{O(\log n)} + \underbrace{T_{\mathsf{insertItem}}}_{O(1)}\big) + \underbrace{T_{\mathsf{merge}}}_{O(\log n)} \Big) \\
&= O(n\log n + \sum_{i=1}^{n} \deg(v_i)(1+\log n)) \\
&= O((n+m)\log n)
\end{aligned}$$

Damit erhalten wir insgesamt eine Laufzeit $O((n+m)\log n)$. ■

3.2.4 Graphenrealisierung aus Abstandsmatrizen

Es sei $G = (V, E, w)$ ein ungerichteter Graph mit Kantengewichten. Für einen Weg $p = (v_1, \ldots, v_r)$ in G ist

$$w(p) =_{\text{def}} \sum_{i=1}^{r} w(v_i, v_{i+1})$$

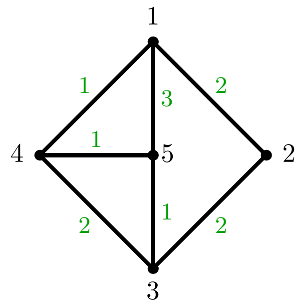
das Gewicht von p. Für Knoten $u, v \in V$ ist der Abstand definiert als

$$d_G(u,v) =_{\text{def}} \inf\{w(p) \mid p = (u, t_1, \ldots, t_r, v) \text{ ist ein Weg in } G\}.$$

Wir wissen bereits, dass *all-paris shortest-paths* in Zeit $O(n^3)$ mit dem Algorithmus von Floyd und Warshall gelöst werden kann.

Es sei $D_G = (d_G(u,v))_{u,v \in V}$.

Beispiel. Wir betrachten folgenden ungerichteten Graphen G mit Kantengewichten:



Dazu korrespondiert folgende Abstandsmatrix D_G:

$$D_G = \begin{pmatrix} 0 & 2 & 3 & 1 & 2 \\ 2 & 0 & 2 & 3 & 3 \\ 3 & 2 & 0 & 2 & 1 \\ 1 & 3 & 2 & 0 & 1 \\ 2 & 3 & 1 & 1 & 0 \end{pmatrix}$$

Wir interessieren uns für die umgekehrte Frage: Wann existiert für Matrix ein passender Graph?

Definition 3.23 *Es sei $A \in \mathbb{N}^{n \times n}$ eine symmetrische Matrix.*

1. *Ein (ungerichteter) gewichteter Graph $G = (V, E, w)$* realisiert A exakt *genau dann, wenn $V = \{1, \ldots, n\}$ und $D_G = A$, d.h. $a_{ij} = d_G(i,j)$ für alle $i, j \in \{1, \ldots, n\}$.*

2. *Ein (ungerichteter) gewichteter Graph $G = (V, E, w)$* realisiert A *genau dann, wenn $\{1, \ldots, n\} \subseteq V$ und $a_{ij} = d_G(i,j)$ für alle $i, j \in \{1, \ldots, n\}$.*

3.2.5 Charakterisierung graphischer Matrizen

Matrizen, die eine exakte Realisierung durch Graphen zulassen, nennen wir auch *graphische Matrizen*. Diese Matrizen lassen sich einfach charakterisieren und der zugehörige Graph auch einfach konstruieren.

Theorem 3.24 *Eine symmetrische Matrix $A \in \mathbb{N}^{n \times n}$ ist genau dann graphisch, wenn folgende Bedingungen erfüllt sind:*

1. *Für alle $i \in \{1, \ldots, n\}$ gilt $a_{ii} = 0$.*

2. *Für alle $i, j, k \in \{1, \ldots, n\}$ gilt $a_{ij} + a_{jk} \geq a_{ik}$.*

Beweis: Wir müssen zwei Richtungen zeigen.

($\Rightarrow$) Es sei $G = (V, E, w)$ ein Graph mit $D_G = A$. Bedingung 1 ist trivial. Angenommen Bedingung 2 gilt nicht. D.h. es gibt $i, j, k \in V$ mit $d_G(i, j) + d_G(j, k) < d_G(i, k)$. Dann ist der kürzeste Weg p_{ij} von i nach j verlängert um den kürzesten Weg p_{jk} von j nach k ein kürzerer Weg von i nach k. Dies ist ein Widerspruch zur Definition von $d_G(i, k)$.

($\Leftarrow$) Wir konstruieren zur Matrix A einen Graphen $G = (V, E, w)$ wie folgt:

- $V =_{\text{def}} \{1, \ldots, n\}$
- $E =_{\text{def}} \{ \{i, j\} \mid i, j \in V \text{ und } i \neq j \}$
- $w(i, j) = w(j, i) =_{\text{def}} a_{ij}$

Klarerweise gilt $d_G(i, j) \leq a_{ij}$ für alle $i, j \in V$. Wir zeigen mittels Induktion über die Länge ℓ kürzester Wege, dass die Gleichheit gilt:

Induktionsanfang. Es gilt $\ell = 1$. Es sei (i, j) ein kürzester Weg zwischen i und j. Dann gilt $d_G(i, j) = a_{ij}$.

Induktionsschritt. Es sei $\ell > 1$. Es sei $(i, t_1, \ldots, t_{\ell-1}, j)$ ein kürzester Weg zwischen i und j. Dann ist $(i, t_1, \ldots, t_{\ell-1})$ ein kürzester Weg zwischen i und $t_{\ell-1}$ über $\ell - 1$ Kanten. Mit der Induktionsvoraussetzung erhalten wir

$$d_G(i, j) = d_G(i, t_{\ell-1}) + d_G(t_{\ell-1}, j) = a_{i,t_{\ell-1}} + a_{t_{\ell-1},j} \geq a_{ij}.$$

Damit gilt $d_G(i, j) = a_{ij}$.

Damit ist das Theorem bewiesen. ■

Bemerkungen.

1. Eine Matrix ist durch ungerichteten Graphen exakt realisierbar genau dann, wenn die Matrix eine Metrik beschreibt.
2. Theorem 3.24 lässt sich mit den gleichen Bedingungen auch für gerichteter Graphen und nicht-symmetrische Matritzen formulieren.
3. Es lassen sich auch Charakterisierungen für exakte Realisierbarkeit durch ungewichtete Graphen angeben.

3.2.6 Der Algorithmus von Culberson und Rudnicki

Die Konstruktion des Graphen im Beweis von Theorem 3.24 enthält nur wenig Information über die Graphenstruktur einer exakten Realisierung. Im Folgenden wollen Realisierungen finden, die sinnvoll hinsichtlich weitere Nebenbedingungen wie z.B. der Graphgrößen sind.

Beispiel. Wir betrachten die symmetrische Matrix

$$A = \begin{pmatrix} 0 & 2 & 3 \\ 2 & 0 & 2 \\ 3 & 2 & 0 \end{pmatrix}$$

Bild für Freiheitsgrade der Realisierungen

Bemerkung. Zu entscheiden, ob es für eine Matrix A ein Realisierung mit Gesamtkantengewicht höchstens k gibt, ist **NP**-vollständig.

Deshalb betrachten wir optimale Realisierbarkeit mit Bäumen.

Definition 3.25 *Es sei $A \in \mathbb{N}^{n \times n}$ ein realisierbare Matrix. Ein Baum $T = (V, E, w)$ heißt genau dann* optimale Baumrealisierung *von A, wenn T die Matrix A realisiert und es keine Baumrealisierung $T' = (V', E', w')$ für A gibt mit $\|V'\| < \|V\|$.*

Um eine optimale Baumrealisierung für eine Matrix zu bestimmen, verwenden wir folgende Beobachtung:

> *Für jeweils drei verschiedene Knoten eines Baumes gibt es genau einen Knoten, der auf allen Pfaden zwischen den einzelnen Knoten liegt.*

Den eindeutig bestimmten Knoten r für gegebene Knoten a, b, c in T werden wir *Hub* nennen, symbolisch $r = \mathrm{Hub}(a, b, c)$.

Bild für Hub in Baum

Damit ergibt sich folgende Zerlegung von Bäumen. Wir betrachten den Pfad zwischen zwei beliebigen Knoten $a, b \in V$. Alle Knoten sind somit eindeutig einem Knoten des Pfades zuordenbar – je nach Hub.

Bild für Hubzerlegung

Wir definieren

$$\mathrm{hub}_T(a, b, c) =_{\mathrm{def}} d_T(a, \mathrm{Hub}(a, b, c)).$$

Klarerweise gilt nun die Beziehung für alle Knoten $a, b, c \in T$

$$\mathrm{hub}_T(a, b, c) = \frac{1}{2}(d_T(a, b) + d_T(a, c) - d_T(b, c))$$

In unserem Fall ist der Baum T jedoch zunächst unbekannt. Deshalb werden wir die rechte Seite und Abstände benutzen, um die Entfernungen der Knoten zu ihren Hubs zu bestimmen. Wir definieren also

$$\mathrm{hub}(a, b, c) =_{\mathrm{def}} \frac{1}{2}(d(a, b) + d(a, c) - d(b, c)),$$

Algorithmus:	CULBERSON-RUDNICKI
Eingabe:	Knoten r; (Terminal)Knotenmenge $V \subseteq \{1,\dots,n\}$, $r \notin V$; Abstandsfunktion $d_r : V \to \mathbb{R}_+$ (als Array); als globale Variable die Matrix A
Aufgabe:	gewichteter Baum $T = (\hat{V}, E, w)$

/* Im Rahmen des Algorithmus sei $d(x,y) = d_A(x,y)$, falls $x, y \in \{1,\dots,n\}$ Terminalknoten sind, und $d(r,x) = d_r(x)$ für Terminalknoten $x \in \{1,\dots,n\}$ */

```
IF ||V|| > 0
    b := Knoten in V mit d_r(b) maximal
    Zerlege V ∪ {r} in Mengen U_1, ..., U_ℓ, so dass gilt:
                                  u, v ∈ U_i ⟺ hub(r, b, u) = hub(r, b, v)
    FOR i := 0 TO ℓ
        IF es gibt v ∈ U_i mit hub(v, r, b) = 0
            r_i := v                          /* r_i ist Terminalknoten für i > 0 */
        ELSE
            r_i := neuer Knoten               /* r_i ist innerer Knoten */
        Z_i := U_i \ {r_i}
        FOR u ∈ Z_i
            d_{r_i}(u) := hub(u, r, b)
        IF i = 0
            h_0 := 0
        ELSE
            h_i := hub(r, b, v) für ein v ∈ U_i
    Sortiere die r_i's so, dass h_i < h_{i+1} gilt
    FOR i := 1 TO ℓ
        Füge neuen Knoten r_i in V̂ ein
        Füge neue Kante {r_{i-1}, r_i} in E ein
        w(r_{i-1}, r_i) := (h_i − h_{i-1})
    FOR i := 0 TO ℓ
        CULBERSON-RUDNICKI(r_i, Z_i, d_{r_i})
```

Abbildung 3.2: Der Algorithmus von Culberson und Rudnicki

wobei $d(\cdot,\cdot)$ eine Abstandsfunktion ist (die wesentlich mit A übereinstimmt). An dieser Stelle ist zu beachten, dass sowohl hub und hub_T im Allgemeinen nicht symmetrisch sind!

Der Algorithmus von Joseph C. Culberson und Piotr Rudnicki benutzt die Funktion hub, um wie in Abbildung 3.2 beschrieben rekursiv jeweils die Hubs und Kantengewichte zu bestimmen. Der Erstaufruf erfolgt als CULBERSON-RUDNICKI$(1, \{2,\dots,n\}, d_A(1,\cdot))$. Hierbei steht $d_A(1,\cdot)$ für die erste Zeile der Abstandsmatrix A. Der Baum ist anfangs als $T = (\{1\}, \emptyset, \emptyset)$ initialisiert.

Beispiel. Wir wollen die Arbeitsweise von Culberson-Rudnicki an einer 6×6-Matrix verdeutlichen. Die Menge V der Terminalknoten ist $V = \{1, 2, \ldots, 6\}$. Es sei A die folgende symmetrische Matrix:

$$A = \begin{pmatrix} 0 & 3 & 4 & 7 & 7 & 5 \\ 3 & 0 & 3 & 6 & 6 & 4 \\ 4 & 3 & 0 & 5 & 5 & 3 \\ 7 & 6 & 5 & 0 & 4 & 6 \\ 7 & 6 & 5 & 4 & 0 & 6 \\ 5 & 4 & 3 & 6 & 6 & 0 \end{pmatrix}$$

Der erste Aufruf ist Culberson-Rudnicki$(1, \{2, \ldots, 6\}, [3, 4, 7, 7, 5])$. Es ergeben sich folgende Werte bei Wahl von $b = 4$:

$$\begin{aligned} \mathrm{hub}(1,4,2) &= \tfrac{1}{2}(7+3-6) = 2 \\ \mathrm{hub}(1,4,3) &= \tfrac{1}{2}(7+4-5) = 3 \\ \mathrm{hub}(1,4,5) &= \tfrac{1}{2}(7+7-4) = 5 \\ \mathrm{hub}(1,4,6) &= \tfrac{1}{2}(7+5-6) = 3 \end{aligned}$$

Die Anordnung der Mengen $U_0, U_1, \ldots, U_4$ lässt sich wie folgt veranschaulichen.

Bild für ersten Aufruf von CR

Die Wurzeln r_i für den rekursiven Aufruf berechnen sich folgendermaßen:

- U_0: Klarerweise ist $r_0 = 1$.
- U_1: Wegen $\mathrm{hub}(2,1,4) = \frac{1}{2}(3+6-7) = 1$ ist $r_1 = 7$. Damit ist r_1 also ein innerer Knoten.
- U_2: Wegen $\mathrm{hub}(3,1,4) = \frac{1}{2}(4+5-7) = 1$ und $\mathrm{hub}(6,1,4) = \frac{1}{2}(5+6-7) = 2$ ist $r_2 = 8$. Auch hier ist r_2 ein innerer Knoten.
- U_3: Wegen $\mathrm{hub}(5,1,4) = \frac{1}{2}(7+4-7) = 2$ ist $r_3 = 9$. Der Knoten r_3 ist also ebenfalls ein innerer Knoten.
- U_4: Klarerweise ist $r_4 = 4$.

Damit ergeben sich die rekursiven Aufrufe:

Culberson-Rudnicki$(1, \emptyset, \emptyset)$
Culberson-Rudnicki$(7, \{2\}, [1])$
Culberson-Rudnicki$(8, \{3, 6\}, [1, 2])$
Culberson-Rudnicki$(9, \{5\}, [2])$
Culberson-Rudnicki$(4, \emptyset, \emptyset)$

Diese Aufrufe führen zu folgender Konstellation nach Ablauf der zweiten Iteration. Die Veranschaulichung erfolgt mittels levelweiser Traversierung des Rekursionsbaums. Für den Knoten 3 rechnen wir aus: $\text{hub}(8,6,3) = \frac{1}{2}(2+1-3) = 0$ und $\text{hub}(3,8,6) = \frac{1}{2}(1+3-2) = 1$.

Bild für zweite Iteration von CR

Der letzte nicht-triviale Aufruf ist Culberson-Rudnicki$(8, \{3\}, [1])$. Damit haben wir folgenden Baum mit 6 Terminal- und 3 inneren Knoten konstruiert:

Bild für Baumrealisierung der Beispielmatrix

Wir gehen zum Beweis der Korrektheit des Algorithmus über.

Es seien $G = (V, E, w)$ und $G' = (V', E', w')$ gewichtete ungerichtete Graphen. Dann heißt G zu G' *isomorph*, symbolisch $G \simeq G'$, falls eine bijektive Abbildung $\varphi : V \to V'$ existiert, so dass $\{u, v\} \in E \Leftrightarrow \{\varphi(u), \varphi(v)\} \in E'$ für alle $u, v \in V$ sowie $w(u, v) = w'(\varphi(u), \varphi(v))$ für alle $\{u, v\} \in E$ gilt.

Lemma 3.26 *Es sei $A \in \mathbb{N}^{n\times n}$ eine symmetrische Matrix, die sich durch Bäume realisieren lässt. Dann konstruiert der Algorithmus von Culberson und Rudnicki einen zur optimalen Realisierung isomorphen Baum.*

Beweis: Der Algorithmus Culberson-Rudnicki wird stets mit einer Eingabe der Form (v, V, d), wobei v ein Knoten, V eine Menge von Terminalknoten und d eine Abstandsfunktion sind, aufgerufen. Es sei $A[U]$ eine Matrix über einer Indexmenge U. Es bezeichne $T^*_{A[U]}$

eine optimale Baumrealisierung von $A[U]$. Die Ordnung p eines Aufrufes von CULBERSON-RUDNICKI sei definiert als $p =_{\text{def}} 1 + \|U\|$. Wir zeigen mittels Induktion über p, dass $T_{A[U]} \simeq T^*_{A[U]}$. Hierbei ist $T_{A[U]}$ der von CULBERSON-RUDNICKI knostruierte Baum. (Ist die Matrix im Kontext klar, so lassen wir die Indizes weg.)

Induktionsanfang: Es sei $p = 1$. Somit ist $V = \emptyset$ bei Aufruf von CULBERSON-RUDNICKI. Wegen der Initialisierung bzw. Zeile 18 des Algorithmus ist $v \in T$. Folglich gilt $T \simeq T^*$.

Induktionsschritt: Es sei $p > 1$. Weiterhin sei $b \in U$ ein Terminalknoten mit $d_v(b)$ maximal. Wir zeigen, dass nach Einfügen aller Knoten und Kanten in den Zeilen 17–20 der Pfad von v nach b in T isomorph zu Pfad von v nach b in T^* ist. Dafür sei $(w_0, w_1, \ldots, w_k)$ mit $w_0 = v, w_k = b$ der eindeutige (!) Pfad in T^* von v nach b. Dann zerfällt T^* in $k+1$ Teilbäume mit w_j als Wurzel und, für $j \geq 1$, mit mindestens einem Terminalknoten in T^*_j. Wir betrachten den Knoten w_j für $j \geq 1$. Dann gibt es offenbar ein $u \in V$ mit

$$d_{T^*}(v, w_j) = \text{hub}(v, b, u) = h_{\pi(j)}.$$

Damit gilt: $d_{T^*}(w_{j-1}, w_j) = h_{\pi(j)} - h_{\pi(j-1)}$. Gibt es nun $u \in V$ mit $\text{hub}(u, v, b) = 0$ und $\text{hub}(v, b, u) = d_{T^*}(v, w_j)$, so ist $u = w_j$, d.h. w_j ist ein Terminalknoten. Anderenfalls ist w_j ein innerer Knoten. D.h. der Pfad in T ist isomorph zu Pfad in T^* (bis auf Nummerierung der inneren Knoten). Nun sind aber alle Knoten in T^*_j über w_j mit dem Pfad von v nach b verbunden. In den Zeilen 3 und 9 wird V so aufgeteilt, dass $Z_j \cup \{w_j\}$ alle Terminalknoten von T^*_j enthält. Es sei A_j die Teilmatrix von $A[V]$ mit Spalten und Zeilen für Terminalknoten in T^*_j. Ist $A[V]$ durch Baum realisierbar, so auch A_j für alle $0 \leq j \leq k$. Weiterhin gilt $\|Z_j\| < \|V\|$ für alle $0 \leq j \leq k$, da insbesondere b in keinem Z_j enthalten ist. Nach Induktionsvoraussetzung konstruiert CULBERSON-RUDNICKI einen Baum $T_j \simeq T^*_j$. Somit folgt die Isomorphie $T \simeq T^*$. ■

Korollar 3.27 (Hakimi & Yau 1964) *Es sei $A \in \mathbb{N}^{n \times n}$ eine symmetrische Matrix, die durch Bäume realisierbar ist. Die optimale Baumrealisierung ist bis auf Isomorphie eindeutig bestimmt.*

Bemerkung. Der Algorithmus von Culberson und Rudnicki kann leicht erweitert werden, um zu entscheiden, ob eine Matrix überhaupt durch Bäume realisierbar ist (Übungsaufgabe!).

Wir wenden uns nun der Laufzeitanalyse des Algorithmus von Culberson und Rudnicki zu.

Proposition 3.28 *Es sei $A \in \mathbb{N}^{n \times n}$ eine symmetrische Matrix mit optimaler Baumrealisierung T.*

1. *Für alle inneren Knoten v gilt $\deg_T(v) \geq 3$.*
2. *Es gibt maximal $n - 2$ innere Knoten.*

Beweis: Die erste Aussage ist offensichtlich. Für den Nachweis der zweiten sei $V = V_0 \cup V_1$, wobei V_0 die Menge der Terminalknoten und V_1 die Menge der inneren Knoten ist. Definiere $k =_{\text{def}} V_1$. Dann gilt für den optimalen Baum T:

$$\begin{aligned} n &\leq \sum_{v \in V_0} \deg_T(v) \\ &= \sum_{v \in V} \deg_T(v) \quad - \sum_{v \in V_1} \deg_T(v) \\ &\leq 2(n+k-1) - 3k && (\deg_T(v) \geq 3 \text{ für alle } v \in V_1 \text{ nach 1.}) \\ &= 2n - k - 2 \end{aligned}$$

Durch Umstellung folgt $k \leq n-2$. ■

Theorem 3.29 *Der Algorithmus von Culberson und Rudnicki kann so implementiert werden, dass er zu einer mit Bäumen realisierbaren symmetrischen Matrix $A \in \mathbb{N}^{n \times n}$ eine optimale Baumrealisierung in $O(n^2)$ Schritten konstruiert.*

Beweis: Die Korrektheit folgt aus Lemma 3.26. Die Komplexitätsaussage ergibt sich wie folgt: In jedem rekursiven Aufruf (mit $U = \emptyset$) wird mindestens eine Kante in den Baum eingefügt. Nach Proposition 3.28 hat der optimale Baum max. $2n-2$ Knoten und $2n-3$ Kanten. Damit gibt es maximal $2n-3$ Aufrufe. Als amortisierte Laufzeit pro eingefügter Kante ergibt sich:

- Zeile 2 benötigt $O(n)$ Schritte
- Zeilen 4–11 benötigen $O(n)$ Schritte
- Zeilen 17–22 benötigen $O(n)$ Schritte
- Zeilen 3, 12–15 und 16 sind implementierbar mit $O(n \log \ell)$ Schritten bei Verwendung von z.B AVL-Bäumen zur Speicherung der Werte $\text{hub}(r, b, v)$

Ingesamt sind dies $O(n) + O(n \log \ell)$ Schritte für $\ell \geq 2$. Dabei werden jedoch ℓ Kanten eingefügt. Folglich ist der Aufwand pro Kante:

$$\frac{O(n) + O(n \log \ell)}{\ell} = O(n)$$

Damit benötigt der Algorithmus insgesamt $O(n^2)$ Schritte. ■

Bemerkungen.

1. Algorithmus von Culberson und Rudnicki ist bezogen auf die Matrixgröße linear.

2. Jeder Algorithmus zur Konstruktion optimaler Baumrealisierungen benötigt $\Omega(n^2)$ Schritte (für die Zugriffe auf die Matrix).

3. Bessere obere Schranken ergeben sich für spezielle Baumklassen:

   - $O(n \log n)$ Schritte für gewichtete Pfadrealisierungen
   - $O(n)$ Schritte für ungewichtete Pfaderealisierungen
   - $O(kn \log_k n) = O(\frac{k}{\log k} \cdot n \log n)$ Schritte für Realisierungen durch ungewichtete Bäume mit beschränktem Grad $k \geq 2$

3.3 Fallstudie: Inter-Domain Routing mit BGP

In dieser Fallstudie ist unser Ziel, den Einfluss individueller Präferenzen (z.B. Geschäftsbeziehungen) auf die Routing-Qualität (insbesondere Stabilität) im Internet zu analysieren und die Präferenzen zu bestimmen.

3.3.1 Ein abstraktes Modell für BGP

BGP (ausführlich *border gateway protocol*) regelt den Verkehr (sowohl für Routing als auch für Routenpropagation) beim Inter-Domain Routing.

Dazu betrachten wir folgendes einfaches Szenario.

Bild für kleines Internet

Hierbei ist ein AS (*autonomes System*) eine zusammenhängende Gruppe von einem oder mehreren IP-Präfixen (IP-Adressen), die von einem oder mehreren Netzwerkoperatoren mit einer *einzigen* und *klar definierten* Routingstrategie betrieben werden (RFC 1930). Jeder Router hält (BGP-)Routingtabelle, die angibt, welcher AS-Pfad zu einer IP-Adresse führt. Zu beachten ist, dass die Routingtabelle verschieden zur IP-Forwarding-Tabelle ist.

Beispiel. Die Routingtabelle für einen Router von AS4 könnte folgendes Aussehen haben:

```
>  default    1
   default    2  1
>  164.0.0.0
>  165.0.0.0  2  5
   165.0.0.0  1  2  5
   166.0.0.0  1  3  6
>  166.0.0.0  2  3  6
```

Die mit > gekennzeichneten Routen sind aktiv.

Wie kommen die Routen die Routentabellen?

Bild für Schema der Routenpropagation

Die Propagation läuft wie folgt:

- AS i beabsichtigt die Routenmenge R_i anzukündigen
- Der Exportfilter von AS i bezüglich AS j lässt eine Routenmenge

$$\mathrm{Exp}_{ij}(R_i) \subseteq \{i\} \times R_i$$

  passieren
- Importfilter von AS j bezüglich AS i lässt Routenmenge

$$\mathrm{Imp}_{ji}(\mathrm{Exp}_{ij}(R_i)) \subseteq \mathrm{Exp}_{ij}(R_i)$$

  passieren
- AS j wählt die besten Routen zu allen Präfixen aus der Menge R_j (ohne die mit i beginnenden Routen) und $\mathrm{Imp}_{ji}(\mathrm{Exp}_{ij}(R_i))$ aus und schickt diese entsprechen der Exportfilter weiter; die Auswahl erfolgt über eine lokale Präferenzfunktion

Damit hängen die vefügbaren Routen maßgeblich von den individuellen Filtern (Policies) sowie den Präferenzen ab. Die Geschäftsbeziehungen zwischen den autonomen Systemen spielen dabei eine zentrale Rolle.

Wichtige Arten von Geschäftbeziehungen zwischen AS sind:

- Kunde-Anbieter (*customer-to-provider*): Anbieter verkaufen Routen an Kunden
- Geschwister (*sibling-to-sibling*): Beide AS gehören zur gleichen Verwaltungseinheit (z.B. ISP)
- Partner (*peer-to-peer*): Partner bieten ihren Kunden gegenseitig Routen an, aber keinen generellen Transit

Wir gehen zu einer graphentheoretischen Betrachtungsweise über.

Es sei V eine Menge vergebener AS-Nummern (im Moment $V \subseteq \{1, \ldots, 65536\}$).

Für $v \in V$ sei $N(v)$ die Menge der Knoten (AS), mit denen v eine direkte Routerverbindung hält. Damit setzen wir $E =_{\text{def}} \{ \{u, v\} \mid v \in N(v) \}$.

Der Graph $G = (V, E)$ heißt *Konnektivitätsgraph* (auf dem AS-Level).

Wir zerlegen $N(v)$ für jedes $v \in V$ gemäß der Beziehungstypen in:

- $\text{Cust}(v)$ sind die Kunden von v
- $\text{Prov}(v)$ sind die Anbieter von v
- $\text{Sib}(v)$ sind die Geschwister von v
- $\text{Peer}(v)$ sind die Partner von v

Für $r \in V$ sei $R(v)$ die Menge aller *aktiven* Routen in der Routingtabelle von v. Insbesondere gilt: Alle Routen sind kreisfrei und v gehört nicht zur Route.

Definition 3.30 *Es seien $G = (V, E)$ ein Konnektivitätsgraph und $v \in V$. Es sei $p = (u_1, \ldots, u_k) \in R(v)$ eine aktive Route.*

1. *Die Route r heißt* Kundenroute*, falls $u_i \in \text{Cust}(v)$ für erstes $u_i \notin \text{Sib}(v)$.*
2. *Die Route r heißt* Anbieterroute*, falls $u_i \in \text{Prov}(v)$ für erstes $u_i \notin \text{Sib}(v)$.*
3. *Die Route r heißt* Partnerroute*, falls $u_i \in \text{Peer}(v)$ für erstes $u_i \notin \text{Sib}(v)$.*
4. *Die Route r heißt* Eigenroute*, falls $u_i \in \text{Sib}(v)$ für alle $i \in \{1, \ldots, k\}$.*

Typische (und vernünftige) Exportfilter respektieren die *selective export rule*:

u exportiert zu	Anbieter	Kunde	Partner	Geschwister
Eigenroute	Ja	Ja	Ja	Ja
Kundenroute	Ja	Ja	Ja	Ja
Anbieterroute	Nein	Ja	Nein	Ja
Partnerroute	Nein	Ja	Nein	Ja

Definition 3.31 *Es sei $G = (V, E)$ ein Konnektivitätsgraph. Ein Pfad $(v_1, \ldots, v_k)$ in G heißt* senkenfrei*, falls für alle $1 \leq i < j < n$ gilt: Ist $v_{i+1} \in \mathrm{Cust}(v_i) \cup \mathrm{Peer}(v_i)$, so gilt $v_{j+1} \in \mathrm{Cust}(v_j) \cup \mathrm{Sib}(v_j)$.*

Theorem 3.32 *Es sei $G = (V, E)$ ein Konnektivitätsgraph. Es sei $P \subseteq \bigcup_{v \in V} R(V)$ eine Routenmenge in G. Respektieren alle AS als Exportfilter die Selective-Export-Rule, so sind alle Pfade in P senkenfrei.*

Beweis: Angenommen es gibt ein $v \in V$ mit $(u_1, \ldots, u_k) \in P \cap R(v)$, aber $(u_1, \ldots, u_k)$ ist nicht senkenfrei. D.h. es gibt $1 \leq i < j < r$ mit:

1. $u_{i+1} \in \mathrm{Cust}(u_i)$ und $u_{j+1} \in \mathrm{Prov}(u_j) \cup \mathrm{Peer}(u_j)$
2. $u_{i+1} \in \mathrm{Peer}(u_i)$ und $u_{j+1} \in \mathrm{Prov}(u_j) \cup \mathrm{Peer}(u_j)$

Beide Fälle können analog behandelt werden. Wir betrachten nur den ersten Fall. Es sei also $u_{i+1} \in \mathrm{Cust}(u_i)$. Es sei j kleinster Index mit $u_{j+1} \in \mathrm{Prov}(u_j) \cup \mathrm{Peer}(u_j)$. Dann gilt $u_{\ell+1} \in \mathrm{Cust}(u_\ell) \cup \mathrm{Sib}(u_\ell)$ für alle $1 \leq \ell < j$. Wir definieren $\ell^* =_{\mathrm{def}} \{\ell | \ell < j$ und $u_{\ell+1} \in \mathrm{Cust}(u_\ell)\}$, d.h. für alle $\ell^* < \ell < j$ gilt $u_{\ell+1} \in \mathrm{Sib}(u_\ell)$. Damit ist $(u_{\ell^*+1}, \ldots, u_k)$ eine Partner- oder Anbieterroute für u_{ℓ^*}. Wegen $u_{\ell^*+1} \in \mathrm{Cust}(u_\ell)$ bzw. $u_{\ell^*} \in \mathrm{Prov}(u_{\ell^*+1})$ wurde $(u_{\ell^*+1}, \ldots, u_k)$ nicht an u_ℓ exportiert. Dies ist ein Widerspruch. Damit ist die Annahme falsch oder wir sind im zweiten Fall. Da dieser ebenso zum Widerpsruch geführt werden kann, ist der Satz bewiesen. ∎

3.3.2 Kombinatorische Inferenz von Beziehungstypen

Es sei $G = (V, E)$ ein ungerichteter Graph. (Im Folgenden fassen wir dabei E als Paarmenge auf, d.h. es gilt stets $(u, v) \in E \Leftrightarrow (v, u) \in E$.)

Es sei T eine Menge von Kantentypen. Eine Abbildung $\varphi : E \to T$ heißt (gemischte) *Orientierung* von G bezüglich T.

Beispiel. Es sei $G = (V, E)$ ein gerichteter Graph. Definiere

$$\varphi(u, v) =_{\mathrm{def}} \begin{cases} \rightarrow, & \text{falls } (u, v) \in E \\ \leftarrow, & \text{falls } (v, u) \in E \end{cases}$$

D.h. G ist der durch φ orientierte Graph bezüglich $T = \{\leftarrow, \rightarrow\}$.

Wir bezeichnen mit $\varphi(G)$ den durch φ orientierten Graphen von $G = (V, E)$, d.h. wir vereinbaren $\varphi(G) =_{\text{def}} (V, E, \varphi)$.

Es sei $(v_0, v_1, \ldots, v_k)$ ein Weg in G. Die Fortsetzung von φ auf Wege ist definiert als:

$$\varphi(v_0, v_1, \ldots, v_k) =_{\text{def}} \varphi(v_0, v_1)\varphi(v_1, v_2)\cdots\varphi(v_{k-1}, v_k),$$

d.h. $\varphi(v_0, v_1, \ldots, v_k) \in T^*$ (ein Wort über T).

Bild für Graph- und Wegorientierung

Die für uns interessanten Kantentypen sind in der Menge H zusammengefasst:

$$H =_{\text{def}} \{\leftarrow, \rightarrow, -, \leftrightarrow\}.$$

Die Interpretation dieser Kantentypen ist wie folgt: Es sei $G = (V, E)$ ein Konnektivitätsgraph. Es seien $u, v \in V$ mit $(u, v) \in E$. Wir definieren eine *konforme* Orientierung als

$$\begin{aligned}
\varphi(u, v) = \rightarrow \quad &\Longleftrightarrow_{\text{def}} \quad v \in \text{Prov}(u)\\
\varphi(u, v) = \leftarrow \quad &\Longleftrightarrow_{\text{def}} \quad v \in \text{Cust}(u)\\
\varphi(u, v) = - \quad &\Longleftrightarrow_{\text{def}} \quad v \in \text{Peer}(u)\\
\varphi(u, v) = \leftrightarrow \quad &\Longleftrightarrow_{\text{def}} \quad v \in \text{Sib}(u)
\end{aligned}$$

Was bedeutet also Senkenfreiheit?

Proposition 3.33 *Es sei $G = (V, E)$ ein Konnektivitätsgraph und $\varphi(G)$ eine konforme Orientierung von G. Es sei $(v_1, \ldots, v_k)$ ein Weg in $\varphi(G)$. Dann gilt: $(v_1, \ldots, v_k)$ ist genau dann senkenfrei, wenn*

$$\varphi(v_1, \ldots, v_k) \in \{\rightarrow, \leftarrow\}^*\{\leftarrow, \rightarrow\}^* \;\cup\; \{\rightarrow, \leftrightarrow\}^* - \{\leftarrow, \leftrightarrow\}^*$$

gilt.

Eine weitere vernünftige Struktur ist Kreisfreiheit des orientierten Graphen. Ein Kreis könnte z.B. bedeuten, dass AS 1 Kunde von AS 2 ist, AS 2 Kunde von AS 3 und schließlich wiederum AS 3 Kunde von AS 1.

Definition 3.34 *Es sei $G = (V, E)$ ein Konnektivitätsgraph und $\varphi(G)$ eine Orientierung von G bezüglich H. Ein minimaler Kreis $(v_1, \ldots, v_k, v_1)$ in G heißt* orientierter *Kreis in $\varphi(G)$, falls*

$$\varphi(v_1, \ldots, v_k, v_1) \in \{\text{—}, \leftrightarrow\}^* \rightarrow \{\rightarrow, \text{—}, \leftrightarrow\}^* \;\cup\; \{\text{—}, \leftrightarrow\}^* \leftarrow \{\leftarrow, \text{—}, \leftrightarrow\}^* \;\cup\; \leftrightarrow^* \text{—} \leftrightarrow^*$$

gilt.

Definition 3.35 *Es sei $G = (V, E)$ ein Konnektivitätsgraph. Eine (Internet-)* Hierarchisierung *von G ist eine Orientierung von G bezüglich H, die keine orientierten Kreise enthält und bei der für alle $v \in V$ die Pfade in $R(v)$ senkenfrei sind.*

Bild für Beispiele von Hierarchisierungen

Damit entsteht folgendes algorithmisches Problem: Gegeben eine Pfadmenge (Routenmenge) P gibt es eine Hierarchisierung auf dem durch P induzierten Konnektivitätsgraph G_P?

Falls der Typ $\leftrightarrow \in H$ zugelassen wird, dann gibt es immer eine triviale (und informationslose) Lösung. Deshalb lassen wir im Folgenden $\leftrightarrow$ nicht zu. Die Partnerbeziehung — ist darüber hinaus auch nicht essentiell für die Existenz von Hierarchisierungen. Wir geben den folgenden Satz ohne Beweis an (für einen Beweis siehe [KMT06]).

Theorem 3.36 *Es sei P eine Pfadmenge. Dann gilt: Es gibt genau dann eine Hierarchisierung von G_P bezüglich $\{\rightarrow, \leftarrow, \text{—}\}$, wenn es eine Hierarchisierung von G_P bezüglich $\{\rightarrow, \leftarrow\}$ gibt.*

Algorithmus:	TopologicalSort
Eingabe:	Pfadmenge P
Aufgabe:	Hierarchisierung für G_P bezüglich $\{\leftarrow, \rightarrow\}$, falls eine existiert

Bestimme für alle $v \in V(P)$, wie oft v in der Mitte von Pfaden vorkommt; speichere die Anzahl in $A[v]$
FOR $v \in V$
    IF $A[V] = 0$
        $Q.\mathsf{push}(v)$
$U := \emptyset$
WHILE NOT $Q.\mathsf{isEmpty}()$
    $u := Q.\mathsf{pop}()$
    FOR $v \in N(u) \cap (V \setminus U)$
        $\varphi(u, v) :=\rightarrow$; $\varphi(v, u) :=\leftarrow$
        FOR $p \in P$, so dass u und v in p Nachbarn sind
            IF es gibt einen Nachbarn w von v in p auf anderer Seite als u
                $A[v] := (A[v] - 1)$
            IF $A[v] = 0$
                $Q.\mathsf{push}(v)$
    $U := U \cup \{u\}$
    IF $U \neq V$
        RETURN „Keine Hierarchisierung möglich“

Abbildung 3.3: Der Algorithmus TopologicalSort

Damit suchen wir nur noch Hierarchisierungen für $\{\leftarrow, \rightarrow\}$.

Bild für Pfadmenge ohne Hierarchisierungen

Proposition 3.37 *Es sei $P = (v_1, \ldots, v_n)$ ein Pfad. Für jede senkenfreie Orientierung φ von $G_{\{p\}}$ bezüglich $\{\leftarrow, \rightarrow\}$ gilt $\deg^-_{\varphi(G_{\{p\}})}(v_i) \geq 1$ für alle $1 < i < n$.*

Proposition 3.38 *Ein gerichteter Graph $G = (V, E)$ ist genau dann azyklisch, wenn jeder induzierte Teilgraph $G' \subseteq G$ einen Knoten $v \in V(G')$ mit $\deg^-_{G'} = 0$ enthält.*

Theorem 3.39 *Der Algorithmus* TopologicalSort *kann so implementiert werden, dass er für eine Pfadmenge P der Größe $N = |P| = \sum_{p \in P} |p|$ (wobei $|(v_1, \ldots, v_k)| = k$ ist) eine Hierarchisierung für G_P bezüglich $\{\leftarrow, \rightarrow\}$ in $O(N)$ Schritte bestimmt, falls eine existiert.*

Beweis: Wir zeigen lediglich die Korrektheit: P erlaubt ein Hierarchisierung genau dann, wenn TopologicalSort ein totales φ ausgibt. Zwei Richtungen sind zu zeigen:

($\Leftarrow$) Es gilt $U = V$, falls TopologicalSort ein totales φ ausgibt. D.h. jedes $u \in V$ wird irgendwann in Zeile 7 aus Queue ausgelesen. Zu diesem Zeitpunkt gilt $u \in V \setminus U$. Nach Zeile 9 gilt $\deg^-_{G_P[V \setminus U]}(u) = 0$. Damit ist $\varphi(G_P)$ azyklisch nach Proposition 3.38. Die Senkenfreiheit ist offensichtlich.

($\Rightarrow$) Wir verwenden Induktion über $|P| = \sum_{p \in P} |p|$.

Induktionsanfang: Der Fall $|P| = 1$ ist offensichtlich.

Induktionsschritt: Es sei $|P| > 1$. Es gebe eine Hierarchisierung für P. Dann gibt es ein $u \in V(P)$, so dass u auf keinen $p \in P$ in der Mitte liegt (nach Proposition 3.37 und Proposition 3.38). Entferne v aus allen Pfaden (wie in Zeile 15). Dies führt zu einer neuen Pfadmenge P' mit folgenden Eigenschaften:

1. $|P'| < |P|$.
2. P' erlaubt eine Hierarchisierung.
3. P' entspricht der Konstellation von TopologicalSort (in Zeilen 8–14), d.h. $V \setminus U = V(P')$ und für alle $v \in V(P')$ ist $A[v]$ gerade die Anzahl der Pfade von P' mit v in der Mitte

Nach Induktionsvoraussetzung bestimmt TopologicalSort eine Hierarchisierung $\varphi(G_{P'})$. Fügen wir u nun wieder zu den entsprechenden Pfaden hinzu mit den Orientierung $\varphi(u, v) =\rightarrow$ and $\varphi(v, u) =\leftarrow$ (wie in Zeile 8), so ist $\varphi(G_P)$ offenbar eine totale, azyklische (nach Proposition 3.38) und senkenfreie (nach Proposition 3.37) Orientierung.

Damit ist die Korrektheit bewiesen. ■

Bemerkungen.

1. Es gibt heuristische Ansätze basierend auf Gradverteilungen in P und G_P (aber sehr sensibel gegenüber der Pfadmenge).
2. Zu entscheiden, ob es eine nichttriviale Hierarchisierung bezüglich $\{\leftarrow, \rightarrow, \leftrightarrow\}$ gibt, ist NP-vollständig.
3. Der Algorithmus TopologicalSort kann so erweitert werden, dass er beliebiges widerspruchsfreies Vorwissen respektiert.

Literaturverzeichnis

[AC75] Alfred V. Aho und Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[AMS99] Noga Alon, Yossi Matias und Mario Szegedy. The space complexity of approximating frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler und Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis-Dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[BM77] Robert S. Boyer und J. Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

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

[Col94] Richard Cole. Tight bounds on the complexity of the Boyer-Moore string matching algorithm. *SIAM Journal on Computing*, 23(5):1075–1091, 1994.

[CR89] Joseph C. Culberson und Piotr Rudnicki. A fast algorithm for constructing trees from distance matrices. *Information Processing Letters*, 30(4):215–220, 1989.

[DGIM02] Mayur Datar, Aristides Gionis, Piotr Indyk und Rajeev Motwani. Maintainin stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[FM85] Philippe Flajolet und G. Nigel Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and Systems Sciences*, 31(2):182–209, 1985.

[Gal57] David A. Gale. A theorem on flows in networks. *Pacific Journal of Mathematics*, 7:1073–1082, 1957.

[Gal79] Zvi Galil. On improving the worst case running time of the Boyer-Moore string matching algorithm. *Communications of the ACM*, 22(9):505–508, 1979.

[Gao01] Lixin Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6)733–745, 2001.

[GT04] Phillip B. Gibbons und Srikanta N. Tirthapura. Distributed Stream Algorithms for Sliding Windows. *Theory of Computing Systems*, 37(3):457–478, 2004.

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

[Hak62] S. Louis Hakimi. On the realizability of a set of integers as degrees of the vertices of a linear graph. I. *Journal of the SIAM*, 10(3):496–506, 1962.

[HY65] S. Louis Hakimi und S. S. Yau. Distance matrix of a graph and its realizability. *Quarterly of Applied Mathematics*, 22:305–317, 1965.

[Hav55] Václav Havel. Eine Bemerkung über die Existenz der endlichen Graphen. In Tschechisch. *Casopis pro Pestováni Matematiky*, 80:477–480, 1955.

[Hum06] Benjamin Frank Hummel. *Automata-based IP Packet Classification*. Diplomarbeit, Fakultät für Informatik, Technische Universität München, 2006.

[Kle03] Jon M. Kleinberg. Detecting a Network Failure. *Internet Mathematics*, 1(1):37–56, 2003.

[KMP77] Donald E. Knuth, James H. Morris, Jr. und Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[KMT06] Sven Kosub, Moritz G. Maaß und Hanjo Täubig. Acyclic Type-of-Relationship Problems on the Internet. In *Proceedings of the 3rd Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN'2006)*, Band 4235 der *Lecture Notes in Computer Science*, S. 98–111. Springer-Verlag, Berlin, 2006.

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

[KSP03] Richard M. Karp, Scott J. Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*, 28(1):51–55, 2003.

[LS62] Roger C. Lyndon und Marcel Paul Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Mathematical Journal*, 9:289–298, 1962.

[Myh57] John R. Myhill. Finite automata and the representations of events. WADC TR-57-624, Wright Air Development Center, Air Research and Development Command, United States Air Force, Wright-Patterson Air Force Base, Ohio, 1957.

[Ner58] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9:541–544, 1958.

[Rys57] Herbert J. Ryser. Combinatorial properties of matrices of zeroes and ones. *Canadian Journal of Mathematics*, 9:371–377, 1957.

[SV99] Venkatachary Srinivasan und George Varghese. Fast Address Lookups Using Controlled Prefix Expansion. *ACM Transactions on Computer Systems*, 17(1):1–40, 1999.

[Var05] George Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann Publishers, San Francisco, CA, 2005.