Algorithmen

Prof. M. Kaufmann[1]

Sommersemester 2010

Stand: 14. Juli 2010, 21:41

[1]Mitschrift: Volodymyr Piven, Alexander Peltzer, Korrektur: Demen Güler, Sebastian Nagel, Michael Ruckaberle

Inhaltsverzeichnis

Vorwort

Beim vorliegenden Text handelt es sich um eine inoffizielle Mitschrift. Als solche kann sie selbstverständlich Fehler enthalten und ist daher für Übungsblätter und Klausuren nicht zitierfähig.

Korrekturen und Verbesserungsvorschläge sind jederzeit willkommen und können an wpiven@gmail.com oder alex.peltzer@gmail.com gerichtet werden.

Der Text wurde mit LaTeX2e in Verbindung mit dem $\mathcal{AMS}$-TeX-Paket für die mathematischen Formeln gesetzt.

Kapitel 0

Einleitung

Dozent:

- Professor Dr. Michael Kaufmann
  Tel.: +49-7071-2977404
  Email: mk@informatik.uni-tuebingen.de

Vorlesungszeiten:

- Dienstag 14-16 Uhr Hörsaalzentrum N9
- Donnerstag 10-12 Uhr Hörsaalzentrum N3

Übungsleitung;

- Christoph Behle
  behle@informatik.uni-tuebingen.de
- https://cis.informatik.uni-tuebingen.de/algo-ss-10/

Literatur:

- Algorithmen - Kurz gefasst, Schöning, Springer

Kapitel 1

Vorlesung

§1 Komplexität

O-Notation: $f(n) = O(g(n))$ (obere Schranke)
wenn $\exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
nützliches:
Logarithmen:

- $\log(a \cdot b) = \log a + \log b$
- $\log_a b = \frac{\log b}{log a}$
- $\log a^b = b \cdot \log a$
- $b^{\log_b a} = a$
- $n! = \sqrt{2\pi n} \cdot (\frac{n}{e})^n (1 + 0(\frac{1}{n}))$ (Stirling-Approximation)
  Eine einfachere Abschätzung ist:
  $(\frac{n}{2})^{\frac{n}{2}} \leq n! \leq n^n$
- $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$ (Gauß'sche Summenformel)
- $\sum_{k=0}^{n} q^k = \frac{q^{n+1}-1}{q-1}$ (geometrische Reihe)
- $\sum_{k=0}^{\infty} q^k = \frac{1}{1-q}$ für $q < 1$
  für $q = \frac{1}{2}$: $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots = 2$
- $\sum_{k=0}^{\infty} k \cdot q^{k-1} = \frac{1}{(1-q)^2}$ (Ableitung!)
- n-te harmonische Zahl
  $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} \approx \ln n + O(1)$

§2 Rekursionen

- Merge-Sort: $T(n) = 2 \cdot T(\frac{n}{2}) + n$
- Schaltkreise: $C(n) \leq C(\frac{2}{3}n) + \frac{n}{3}$
- parallele Algorithmen: $T(n) = T(\frac{9}{10}) + O(\log n)$
- Selection: $T(n) \leq \frac{2}{n} \cdot \sum_{k=\frac{n}{2}}^{n-1} T(k) + O(n)$

3 Methoden

Schätzen und Beweisen

- $T(n) = 2 \cdot T(\frac{n}{2}) + n$
  Schätzen: $T(n) = O(n \log n)$
  Beweisen: Behauptung $T(n) \leq c \cdot n \log n + 1$ für c groß genug.
  Induktion:
  Induktionsanfang:
  $T(1) = 1$
  Induktionsschritt:
  $T(n) \leq 2 \cdot c \cdot \frac{n}{2} \cdot \log(\frac{n}{2}) + n$
  $= c \cdot n \cdot ((\log n) - 1) + n \leq c \cdot n \log n$ (für $c \geq 1\sqrt$
- $C(n) = c(\frac{2}{3}n) + \frac{n}{3}$
  Schätzen: $C(n) = O(n)$
  Beweis:
  $C(n) \leq d \cdot n$ für konstante d
  $C(1) = 1 \leq d$ für $d \geq 1$
  $C(n) \leq d \cdot \frac{2}{3}n + \frac{n}{3} \leq d(\frac{2}{3}n + \frac{n}{3})$ für $d \geq 1$
  $= d \cdot n \ \sqrt$
- $A(n) = 2 \cdot A(\frac{n}{2}) + \sqrt{n}$
  Schätzen:
  Beweis:
  $A(n) = O(n)$
  $A(n) \leq c \cdot n$ für c groß.
  $A(1) = 1$
  $A(n) = 2 \cdot c \cdot \frac{n}{2} + \sqrt{n}! \leq c \cdot n$ so nicht!

Ausrechnen

$A(n) = 2 \cdot A(\frac{n}{2}) + \sqrt{n}$
$= 2(2 \cdot A(\frac{n}{4}) + \sqrt{\frac{n}{2}} + \sqrt{n}$
$= 4 \cdot A(\frac{n}{4}) + 2 \cdot \sqrt{\frac{n}{2}} + \sqrt{n}$
$= 8 \cdot A(\frac{n}{8}) + 4\sqrt{\frac{n}{4}} + 2\sqrt{\frac{n}{2}} + \sqrt{n}$
...
$= 2^i \cdot A(\frac{n}{2^i}) + \sum_{j=0}^{i-1} 2^j \cdot \sqrt{\frac{n}{2^j}}$ i-ter Schritt
$= 2^i \cdot A(\frac{n}{2^i}) + \sqrt{n} \cdot \sum_{j=0}^{i-1} 2^{\frac{j}{2}} \sqrt{2}^j$ (Schritt ist durch Induktion zu zeigen!)
i= $\log n$:
$2^{\log n} \cdot A(1) + \sqrt{n} \sum_{j=0}^{\log n - 1} \sqrt{2}^j$
$= n + \sqrt{n} \cdot c \cdot \sqrt{n}$
$= O(n)$

Master Theorem

Löse $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ $(a, b \geq 1, f(n) > 0)$
Beispielsweise für Merge-Sort: $a = b = 2, f(n) = n$

2.1 Satz
Sei $a \geq 1, b > 1$ konstant, $f(n), T(n)$ nicht negativ mit $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ wobei $\frac{n}{b}$ für $\lceil \frac{n}{b} \rceil$ oder $\lceil \frac{n}{b} \rceil$ steht.

- *Für $f(n) = O(n^{\log_b a - \epsilon})$, $\epsilon > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$*
- *Für $f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \cdot \log_b n)$*
- *Für $f(n) = \Omega(n^{\log_b a + \epsilon})$, $\epsilon > 0$ und $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ für $c < 1$ und n genügend groß, so ist: $T(n) = \Theta(f(n))$*

Beispiel:

- $T(n) = 9 \cdot T(\frac{n}{3}) + n, a = 9, b = 3, f(n) = n$
  $\log_3 9 = 2 \Rightarrow$Fall 1 $\Rightarrow T(n) = \Theta(n^2)$
- $T(n) = T(\frac{n}{2}) + 1, a = 1, b = 2, f(n) = 1$
  $\log_b a = \log_2 1 = 0 \Rightarrow$Fall 2 $\Rightarrow T(n) = \Theta(\log n)$
- $T(n) = 2 \cdot T(\frac{n}{2}) + n \log n, a = 2, b = 2, f(n) = n \log n$
  $log_2 2 = 1$
  $n \log n < n^{1+\epsilon} \Rightarrow$ nicht anwendbar! (da "zwischen" Fall 2 und 3!)

$T(n) = a \cdot T(\frac{n}{b}) + f(n), a, b > 1, f(n) > 0$

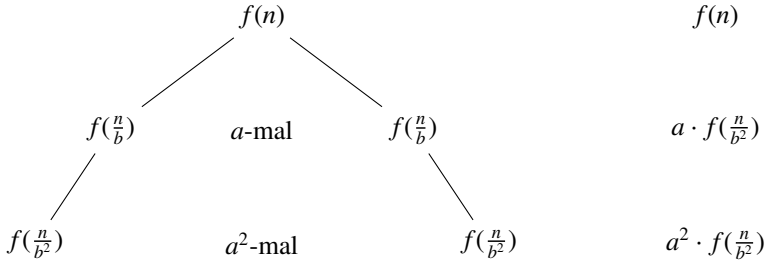
Beweis:
Annahme: $n = b^i, i \in \mathbb{N}$
das heißt, Teilprobleme haben Größe $b^i, b^{i-1}, \ldots, b, 1$
Lemma: Sei $a, b > 1, f(n) > 0, n = b^i, i \in \mathbb{N}$
mit

$$T(n) = \begin{cases} \Theta(1) & \text{für } n = 1 \\ a \cdot T(\frac{n}{b}) + f(n) & \text{für } n = b^i \end{cases}$$

Dann gilt $T(n) = \Theta(n^{\log_b a}) + \sum_{j=1}^{(\log_b n)-1} a^j f(\frac{n}{b^j})$

Beweis:

$$\begin{aligned} T(n) &= f(n) + a \cdot T(\frac{n}{b}) \\ &= f(n) + a(f(\frac{n}{b}) + a \cdot T(\frac{n}{b^2})) \\ &= f(n) + a \cdot f(\frac{n}{b}) + a^2 \cdot T(\frac{n}{b^2}) \\ &= f(n) + a \cdot f(\frac{n}{b}) + a^2 \cdot T(\frac{n}{b^2}) + a^3 \cdot T(\frac{n}{b^3}) \\ &= \vdots \\ &= f(n) + a \cdot f(\frac{n}{b}) + a^2 \cdot f(\frac{n}{b^2}) + \ldots + a^i T(\frac{n}{b^i}) \\ &= f(n) + a \cdot f(\frac{n}{b}) + \ldots + \underbrace{a^{\log_b n} \cdot T(1)}_{=\Theta(n^{\log_b a})} \\ &= \Theta(n^{\log_b a}) + \sum_{j=0}^{(\log_b n)-1} a^j f(\frac{n}{b^j}) \end{aligned}$$



3 Fälle, je nachdem wie groß $f(n)$ ist.
Lemma:
Sei $a, b \geq 1$, $f(n)$ definiert auf Potenzen von b und sei $g(n) = \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})$

1. Ist $f(n) = O(n^{\log_b a-\epsilon})$, $\epsilon \geq 0$, so ist $g(n) = O(n^{\log_b a})$
   Beweis: Es gilt $f(\frac{n}{b^j}) = O((\frac{n}{b^j})^{\log_b a-\epsilon})$
   $\Rightarrow g(n) = O\left(\sum_{j=0}^{\log_b n-1} a^j \cdot (\frac{n}{b^j})^{\log_b a-\epsilon}\right)$
   $= O\left(n^{\log_b a-\epsilon} \cdot \sum_{j=0}^{\log_b n-1} (\frac{a \cdot b^\epsilon}{b^{\log_b a}})^j\right)$

$$= O\left(n^{\log_b a-\epsilon} \cdot \sum_{j=0}^{\log_b n-1} (b^\epsilon)^j\right) \text{ (geom. Reihe)}$$
$$= O\left(n^{\log_b a-\epsilon} \cdot \frac{(b^\epsilon)^{\log_b n}-1}{b^\epsilon-1}\right) < n^\epsilon$$
$$= O\left(n^{\log_b a-\epsilon} \cdot n^\epsilon\right) = O\left(n^{\log_b a}\right)$$

§3 Suchen in geordneten Mengen (Telefonbuch)

Universum U, mit linearer Ordnung $(U, <)$.
Sei $S \subseteq U$ die Menge zu verwalt. Schlüssel.

1. lineare Suche
   S als Array, geordnet
   Starte mit next $\leftarrow 0$
   Schleife: next $\leftarrow$ next + 1
   Abfrage: Ist $a < S[\text{next}]$?
   Laufzeit:
   best case: $O(1)$
   worst case: $O(n)$, n Iterat., $|S| = n$
   average case: $\frac{n}{2}$ Iterat. $= O(n)$

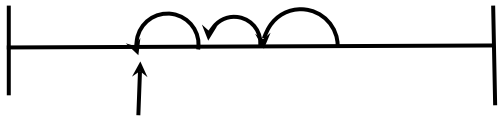
2. binäre Suche: Suchen (a, S)
   S als Array
   2 Grenzen oben, unten als Indizes
   Schleife: next $\leftarrow \lceil \frac{\text{oben - unten}}{2} \rceil$ + unten

$$\begin{aligned}
\text{Laufzeit: } T(n) &= 1 + T\left(\frac{n}{2}\right) \\
&= 1 + 1 + T\left(\frac{n}{4}\right) \\
&= 1 + 1 + 1 + T\left(\frac{n}{8}\right) \\
\text{i-ter Schritt} &= i + T\left(\frac{n}{2^i}\right) \\
&= \log n + T(1) = O(\log n)
\end{aligned}$$

Interpolationssuche

next $\leftarrow \left\lfloor \frac{a - S[\text{unten}]}{S[\text{oben}] - S[\text{unten}]} \cdot (\text{oben} - \text{unten}) \right\rfloor$ + unten

Laufzeit: worst case: $O(n)$
average case: $O(\log \log n)$



unten oben

Interpolation

Interpolieren und Springen sukzessive in Sprüngen der Länge $\sqrt{\text{oben - unten}}$, bis das richtige Suchintervall gefunden ist. Suchintervall hat Größe $\sqrt{\text{oben - unten}}$.

Suchen in geordneten Mengen

$|S| = n$

a.) Lineare Suche $O(n)$

b.) Binäre Suche $O(\log n)$

c.) Interpolationssuche $O(\log \log n)$ im Mittel.

Variante von Reingold

S als Array $S[1], \ldots, S[n]$ füge künstliche Elemente ein $S[0], \ldots S[n+1]$
next $\leftarrow$ (unten -1) + $\lceil \frac{a - S(\text{unten}-1)}{S(\text{oben}+1) - S(\text{unten}-1)} \cdot (\text{oben - unten} + 1) \rceil$
Starte mit $n \leftarrow$ oben $-$ unten $+1$

(1) Interpoliere: $a : S[next]$

(2) falls "=" fertig
falls >: lineare Suche in $S[next + \sqrt{n}], S[next + 2 \cdot \sqrt{n}] \cdots$ bis $a < S[next + (i - 1)\sqrt{n}]$
Es gilt : Nach i Vergleichen gilt: $S[next + (i-2)\sqrt{n}) \leq a < S[next + (i-1)\sqrt{n}]$

(3) rekursiv im Suchintervall der Größe $\sqrt{n}$ weitersuchen.

Analyse

Sei C die mittlere Anzahl von Vergleichen um auf Teilproblem der Größe $\sqrt{n}$ zu kommen, dann:

$$T(n) \leq C + T(\sqrt{n})$$
$$T(1) = 1$$
$$T(n) = O(\log \log n) \text{ falls } C \text{ konstant.}$$

Wahrscheinlichkeitsannahme

Elemente $a_1 \dots a_n$ und auch a sind zufällige Elemente aus (a_0, a_{n+1}) gezogen nach Gleichverteilung.
Sei p_i Wahrscheinlichkeit, dass $\geq i$ Vergleiche nötig sind.
$\Rightarrow C_i = \sum_i i \cdot \underbrace{(p_i - p_{i+1})}_{\text{Prob(genau i Vergleiche.)}} = 1(p_1 - p_2) + 2(p_2 - p_3) + 3(p_3 - p_4) = \sum_{i \geq 1} p_i$

$p_1 = 1 = p_2$
Was ist jetzt p_i ?
Falls i Vergleiche nötig sind, weicht der tatsächliche Rang von a in der Folge $a_1, \dots, a_n$ um mindestens $(i-2) \cdot \sqrt{n}$ von next ab.
(Rang von a: Anzahl von a_i mit $a_i < a$)
das heißt: $p_i \leq$ prob $((Rang(a) - next) \geq (i-2)\sqrt{n})$
bedeutet prob $((y - \mu(y)) \geq (i-2) \cdot \sqrt{n})$ mit unten = 1, oben = n, next = $p \cdot n$
$p = \frac{a - S[unten-1]}{S[oben+1] - S[unten-1]}$

next ist die erwartete Anzahl der Elemente $\leq a$
also der erwartete Rang von a.
Sei y Zufallsvariable, die Rang von a in $a_1, \dots, a_n$ angibt, sowie $\mu(y)$ Erwartungswert.
Tschebyscheff'sche Ungleichung:

$$\text{prob } ((y - \mu(y)) > t) \leq \frac{Var(y)}{t^2}$$

$Var(x) = E((x - E(x))^2)$

Es gilt: Erwartungswert: $p \cdot n$
Varianz: $(1-p) \cdot p \cdot n$

Erwartungswert μ
prob(genau j Elemente < a) = $\binom{n}{j} p^j (1-p)^{n-j}$
Deshalb:

$$\begin{aligned}
\mu &= \sum_{j=1}^{n} j \cdot \binom{n}{j} p^j (1-p)^{n-j} \\
&= n \cdot p \cdot \sum_{i=1}^{n} \binom{n-1}{j-1} p^{j-1} (1-p)^{n-j} \\
&= n \cdot p \cdot \sum_{j=0}^{n-1} \binom{n-1}{j} p^j (1-p)^{n-(j+1)} \\
&= n \cdot p \cdot 1 \\
&= n \cdot p
\end{aligned}$$

$$p_i \leq \text{prob} \left(\underbrace{Rang}_{y} - \underbrace{next}_{\mu} \geq \underbrace{(i-2)\sqrt{n}}_{t} \right)$$
$\leq \frac{Var(y)}{t^2} = \frac{p(1-p) \cdot \not{n}}{(i-2)^2 \cdot \not{n}} \leq \frac{1}{4(i-2)^2}$

$C \leq \sum_i p_i \leq 2 + \sum_{i \geq 3} \frac{1}{4(i-2)^2} \leq 2 + \frac{\pi^2}{24} \approx 2.4$
$\Rightarrow$ Im Mittel brauchen wir $O(\log \log n)$ Versuche.
worst case:

$$T_{\text{worst}}(n) = T_{\text{worst}}(\sqrt{n}) + (\sqrt{n} + 1) = O\left(\sqrt{n} + \sqrt{\sqrt{n}} + \sqrt{\sqrt{\sqrt{n}}} + \ldots\right) = O(\sqrt{n})$$

aber: Annahme der Gleichverteilung kritisch.

Sortieren: $A : 11, 3, 4, 5, 4, 12 \quad \rightarrow \quad B : 3, 6, 6, 7, 11, 12$

Verfahren: 1.Minimumsuche + Austausch mit 1. Element usw. $O(n^2)$
2.Insertionsort $O(n^2)$
3.Bubble Sort $O(n^2)$
4.Quicksort, Heapsort, MergeSort, BucketSort

§4 Quicksort(Divide and Conquer)

Eingabe $S \quad a_1, \ldots, a_n$
1. Wähle Pivotelement a_1
Teile $S \backslash \{a_1\}$ auf in $A = \{a_i \mid a_i \leq a_1 \text{ für } i \geq 2\}$, $B = \{a_i \mid a_i > a_1 \text{ für } i \geq 2\}$
Speichere A in $S[1], \ldots, S[i]$ a_1 in $S[j+1]$, B in $S[j+2], \ldots S[n]$
2. Sortiere A und B rekursiv.
Analyse: Schlechtester Fall: wenn a_1 minimal ($A = \emptyset$) oder a_1 maximal ($B = \emptyset$).
$\Rightarrow$ # Vergleiche: $(n-1) + (n-2) + (n-3) + \ldots + 1 = \frac{n(n-1)}{2} = O(n^2)$
Mittlere Analyse: Modelliere Elemente paarweise verschieden jedes der $n!$ Möglichen Permutationen der Eingabe ist gleich wahrscheinlich (oBdA)
$S = \{1, 2, \ldots, n\}$ $S[1]$ mit Wahrscheinlichkeit $\frac{1}{n}$ für $k = 1, 2, \ldots, n$
Teilprobleme der Größe $k-1$ bzw. $n \cdot k$ erüllen wieder die W' annahmen. Sei $\overline{QS}(n)$ die mittlere Anzahl von Vergleichen an Feld der Größe n.
$\overline{QS}(0) = \overline{QS}(1) = 0$
für $n \geq l \cdot \overline{QS}(n) = n + E(QS(A) + QS(B)) =$
$n + \frac{1}{n} \cdot \sum_{k-1}^{n} \overline{QS}(k-1) + \overline{QS}(n-k) =$
$n + \frac{2}{n} \cdot \sum_{k=1}^{n-1} \overline{QS}(k)$

$$n \cdot \overline{QS}(n) = n^2 + 2 \sum_{k-1}^{n-1} \overline{QS}(k) \text{ (i)}$$
$$(n+1) \cdot \overline{QS}(n+1) = (n+1)^2 + 2 \cdot \sum_{k-1}^{n} (\overline{QS}(k)) \text{ (ii)}$$
$$(ii) - (i) = (n+1)\overline{QS}(n+1) \cdot n\overline{QS}(n) = (n+1)^2 - n^2 + 2\overline{QS}(n)$$
$$n + 1\overline{QS}(n+1) = 2n + 1 + (n+2)\overline{QS}(n)$$
$$\overline{QS}(n+1) = 2 + \tfrac{n+2}{n+1}\overline{QS}(n)$$
$$= 2 + \tfrac{n+2}{n+1}(2 + \tfrac{n+1}{n}\overline{QS}(n-1))$$
$$= 2 + (n+2) \cdot (\tfrac{2}{n+1} + \tfrac{2}{n} + \tfrac{2}{n-1} + \ldots + \tfrac{2}{2} + \tfrac{2}{1})$$
$$\overline{QS}(n) = 2 + 2 \cdot (n+1)(\tfrac{1}{n} + \tfrac{1}{n-1} + \tfrac{1}{n-2} + \ldots + 1)$$
$$< 2 + 2(n+1) \cdot \ln(n+1)$$
$$O(n \cdot \log(n))$$

§5 Heap Sort

Beispiel für Datenstruktur "Heap"
Sortieren durch Min-Suche.
Heap: Binärer Baum: Der ist entweder leer, oder hat Wurzelknoten mit linkem und rechtem Teilbaum .
Baum B und u Knoten in T . Tiefe $(u) = \begin{cases} O & \text{, falls u Wurzel von T} \\ x & \text{, Tiefe (parent) + 1, sonst} \end{cases}$
"Tiefe = Abstand zur Wurzel"
Höhe$(u) = \begin{cases} 0 & \text{, falls u Blatt} \\ max(Hhe \mid v) & \text{, } fallsuBlattoderKnotenu(sonst) \end{cases}$
Beispiel Höhe$(v) = 1$
Beispiel Höhe$(u) = 3$

Allgemein gilt für Baum T:
Tiefe(T) = max{Tiefe (v) | $v \in T$}
Höhe(T) = Höhe { Wurzel (T)}
Heap wird durch binären Baum dargestellt, wobei jedes Element aus S einem Knoten aus h zugeordnet wird.
Heapeigenschaft: für Knoten mit u, v mit parent(v) = u gilt $S[u] \leq S[v]$
Beobachtung:
Beim Heap stet das Minimum in der Wurzel. Sortieren durch Min-Suche.
1. Suche: Min → Wurzel O(1)
2. Lösche Min aus Suchmenge und repariere Heap.

Ablauf:

- Nimm Blatt und füge es zu Wurzel hinzuu
- Lass es "nach unten sinken" (Vergl. vertausche mit kleinstem Kind)
- # Iterationen = Höhe(T)

Ausgewogene Bäume: Alle Blätter haben Tiefe k oder $k+1$
Blätter auf Tiefe $k+1$ stehen ganz links.
"Kanonische Form": Es gilt auf Tiefe $1, 2, \ldots, k$ liegen $2^1, \ldots, 2^k$ Knoten.

§6 Quicksort(Divide and Conquer)

Eingabe $S \quad a_1, \ldots, a_n$
1. Wähle Pivotelement a_1
Teile $S \backslash \{a_1\}$ auf in $A = \{a_i \mid a_i \leq a_1$ für $i \geq 2\}$, $B = \{a_i \mid a_i > a_1$ für $i \geq 2\}$
Speichere A in $S[1], \ldots, S[i]$ a_1 in $S[j+1]$, B in $S[j+2], \ldots S[n]$
2. Sortiere A und B rekursiv.
Analyse: Schlechtester Fall: wenn a_1 minimal ($A = \varnothing$) oder a_1 maximal ($B = \varnothing$).
$\Rightarrow$ # Vergleiche: $(n-1) + (n-2) + (n-3) + \ldots + 1 = \frac{n(n-1)}{2} = O(n^2)$
Mittlere Analyse: Modelliere Elemente paarweise verschieden jedes der $n!$ Möglichen Permutationen der Eingabe ist gleich wahrscheinlich (oBdA)
$S = \{1, 2, \ldots, n\}$ $S[1]$ mit Wahrscheinlichkeit $\frac{1}{n}$ für $k = 1, 2, \ldots, n$
Teilprobleme der Größe $k-1$ bzw. $n \cdot k$ erüllen wieder die W' annahmen. Sei $\overline{QS}(n)$ die mittlere Anzahl von Vergleichen an Feld der Größe n.
$\overline{QS}(0) = \overline{QS}(1) = 0$
für $n \geq l \cdot \overline{QS}(n) = n + E(QS(A) + QS(B)) =$
$n + \frac{1}{n} \cdot \sum_{k-1}^{n} \overline{QS}(k-1) + \overline{QS}(n-k) =$
$n + \frac{2}{n} \cdot \sum_{k=1}^{n-1} \overline{QS}(k)$

$$n \cdot \overline{QS}(n) = n^2 + 2 \sum_{k-1}^{n-1} \overline{QS}(k) \text{ (i)}$$

$$(n+1) \cdot \overline{QS}(n+1) = (n+1)^2 + 2 \cdot \sum_{k-1}^{n} (\overline{QS}(k)) \text{ (ii)}$$

$$
\begin{gathered}
(ii) - (i) = (n+1)\overline{QS}(n+1) \cdot n\overline{QS}(n) = (n+1)^2 - n^2 + 2\overline{QS}(n) \\
n+1\overline{QS}(n+1) = 2n+1+(n+2)\overline{QS}(n) \\
\overline{QS}(n+1) = 2 + \tfrac{n+2}{n+1}\overline{QS}(n) \\
= 2 + \tfrac{n+2}{n+1}(2 + \tfrac{n+1}{n}\overline{QS}(n-1)) \\
= 2 + (n+2) \cdot (\tfrac{2}{n+1} + \tfrac{2}{n} + \tfrac{2}{n-1} + \ldots + \tfrac{2}{2} + \tfrac{2}{1}) \\
\overline{QS}(n) = 2 + 2 \cdot (n+1)(\tfrac{1}{n} + \tfrac{1}{n-1} + \tfrac{1}{n-2} + \ldots + 1) \\
< 2 + 2(n+1) \cdot \ln(n+1) \\
O(n \cdot \log(n))
\end{gathered}
$$

§7 Heap Sort

Beispiel für Datenstruktur “Heap”
Sortieren durch Min-Suche.
Heap: Binärer Baum: Der ist entweder leer, oder hat Wurzelknoten mit linkem und rechtem Teilbaum .

Baum B und u Knoten in T . Tiefe $(u) = \begin{cases} O & \text{, falls u Wurzel von T} \\ x & \text{, Tiefe (parent) + 1, sonst} \end{cases}$

“Tiefe = Abstand zur Wurzel”

Höhe$(u) = \begin{cases} 0 & \text{, falls u Blatt} \\ max(Hhe \mid v) & \text{, } fallsuBlattoderKnotenu(sonst) \end{cases}$

Beispiel Höhe$(v) = 1$
Beispiel Höhe$(u) = 3$
Allgemein gilt für Baum T:
Tiefe$(T) = max\{\text{Tiefe } (v) \mid v \in T\}$
Höhe(T) = Höhe { Wurzel (T)}
Heap wird durch binären Baum dargestellt, wobei jedes Element aus S einem Knoten aus h zugeordnet wird.
Heapeigenschaft: für Knoten mit u, v mit parent$(v) = u$ gilt $S[u] \leq S[v]$
Beobachtung:
Beim Heap stet das Minimum in der Wurzel. Sortieren durch Min-Suche.
1. Suche: Min → Wurzel O(1)
2. Lösche Min aus Suchmenge und repariere Heap.

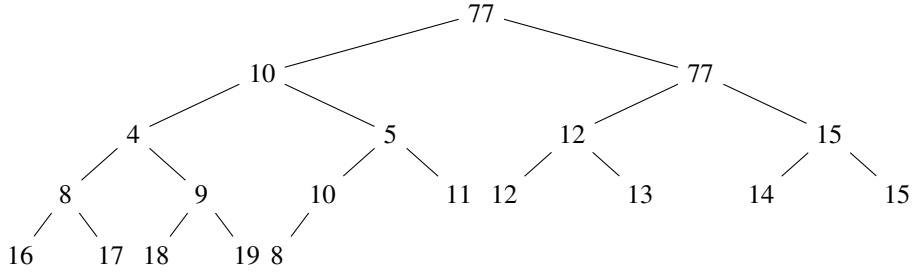
Ablauf:

- Nimm Blatt und füge es zu Wurzel hinzuu
- Lass es “nach unten sinken” (Vergl. vertausche mit kleinstem Kind)
- # Iterationen = Höhe(T)

Ausgewogene Bäume: Alle Blätter haben Tiefe k oder $k+1$
Blätter auf Tiefe $k+1$ stehen ganz links.
“Kanonische Form”: Es gilt auf Tiefe $1, 2, \ldots, k$ liegen $2^1, \ldots, 2^k$ Knoten.

Wiederholung
Heapsort, ausgewogene Bäume
Heapeigenschaft, Elemente sinken nach unten $O(h)$, h Tiefe.

Numeriere levelweise von links nach rechts.



Es gilt für Knoten mit Nummer i: *parent*(i) hat die Nummer $\lfloor \frac{i}{2} \rfloor$, Kinder haben $2i$ und $2i+1$.
Heap wird mit der Nummerierung als Array dargestellt , nur konzeptuell als Baum.
Initialisieren den Heap H mit :

```
for all a  ∈  S do Insert (a, H)
Insert(a,H): Sei n die Größe von H
n <---  n + 1
A(n) <-- a; i <---j <---n   (Lasse das neue Element aufsteigen)
fertig <----(i=1)
while not fertig do j <---  ⌊ i/2 ⌋
   if A(j) > A(i) then Tausche (A(j) , A(i)), i <--- j
     else fertig <--- true out
```

Laufzeit:
Initialisierung: $O(n \cdot \text{Höhe}(H))$
eigentliche Sortierung: $O \cdot \text{Höhe}(H)$
Höhe (H) ist $O(\log n)$, da Baum ausgewogen ist. (Beweis: Übung)

7.1 Satz
HeapSort läuft in Zeit $O(n \log n)$, auch im Worst-Case.

Beachte $\overline{QS} \leq 2 \cdot n \log n$
Wie ist HeapSort-Konstante.

§8 MergeSort

Prinzip: mische 2 sortierte Folen zusammen zu einer.
(7, 4, 2, 1) + (8,6,5) ⟩ 8,7,6,5,4,2,1
Es wirden immer die aktuell minimmalen Elemente verglichen ($O(1)$) und dann das kleinste gestrichen.
Idee:
Starte mit Teilfolgen der Länge 1. (n Teilfolgen)
Mache daraus Teilfolgen der Länge 2 ($\frac{n}{2}$ Teilfolgen)
usw. bis aus 2 Teilfolgen eine entsteht.

Laufzeit:
Teilfolgen haben nach der i-ten Iteration die Länge $\leq 2^i$. Ist $2^i \geq n \Rightarrow i \geq \log n$ Iterationen
Pro Iteration mache $O(n)$ Vergleiche. $\Rightarrow$ Insgesamt $O(n \log n)$ Vergleiche.
Allgemeiner:
"m-Wege-Mischen": $O(\log_m n)$ Iterationen

§9 Bucket Sort

(Sortieren durch Fachverteilung)
Gegeben: Wörter über Alphabet $\sum$. Sortiere lexikographisch.

Ordnung: $a < B \qquad |\sum| = m$
$a < aa < aba$

Fall 1: Alle wörter haben Länge 1. Stelle m Fächer bereit (für $a, b, c, \ldots, z$). Wirf jedes Wort in entspr. Fach. Konkateniete die Inhalte der Fächer.
$\Rightarrow$ Laufzeit: $O(n) + O(m)$
Implemetiere einzelne Fächer als lin. Listen.
Fall 2: Alle Wörter haben Länge k. Sei Wort $a^i = a_1^i a_2^i \cdots a_k^i$ das i-te Wort, $1 \leq i \leq n$
Idee: Sortiere zuerst nach dem letzten Zeichen, dann dem vorletzten, usw. Damit sind Elemente, die zum Schluss ins gleiche Fächer fallen, richtig geordnet.
Laufzeit: $O((n + m)k)$

Beispiel 124, 223, 324, 321, 123

Fächer	1	2	3	4
1.Iter.	321		223 123	124
2.Iter.		321 223 123 124 224		
3.Iter.	123 124	223 224	321	

Problem: Wollen leere Listen überspringen beim Aufsammeln der Listen. Also Ziel: statt $O((n + m)k)$ besser $O(n \cdot k + m)$.
Erzeugen Paare $\left(j, a_j^i\right)$, $1 \leq i \leq n$, $1 \leq j \leq k$.
Sortiere nach 2. Komp., dann nach der ersten, dann liegen im j-ten Fach die Buchstaben sortiert, die an der j-ten Stelle vorkommen.
$\Rightarrow$ leere Fächer werden übersprungen.
$\Rightarrow$ Laufzeit: $O(n \cdot k + m)$

Allgemein: Wort a^i habe Länge l_i, $L = \sum_{i=1}^{n} l_i$
$R_{max} = \max\{l_i\}$
Idee: Sortiere Wörter der Länge nach und beziehnen zuerst die langen Wörter ein.

Algorithmus: 1. Erzeuge Paare $\left(l_i, \text{ Verweis auf } a^i\right)$
2. Sortiere Paare durch Buckertsort nach 1.Komp
3. Erzeuge L Paare $\left(j, x_j^i\right), 1 \leq i \leq n, 1 \leq j \leq l_i$ und sortiere zuerst nach 2. Komp., dann nach 1. Komp. Diese liefert lineare Listen. Nichtleer $(j), 1 \leq j \leq r_{max}$
4. Sortierenun a^is durch Bucketsort wie oben unter Berücksicht von $L(k)$

9.1 Satz
Buckertsort sortiert n Wörter der Gesamtlänge L über Alphabet $\sum$, $|\sum| = m$ *in Zeit* $O(m + L)$

§10 Hybridsort

Jetzt reelle Zahlen aus $]0, 1]$:
Hybridsort: Sei $k = c \cdot n$ mit $c \in \mathbb{R}$ fest.

1.) Wirf a_i in Fach $\lceil k \cdot a_i \rceil$ für $1 \leq c \leq n$

2.) Wende HeapSort auf jedes Fach an und konkateniere die Fächer.

Laufzeit:
Worst case: $O(n \log n)$
average case: Sind a_i unabhängig und gleichverteilt , so hat Hybrid Sort Laufzeit $O(n)$.
Auswahlproblem: Finde den Median (mittleres Element)
formal: Gegeben Menge S von n Zahlen und Zahl i, $1 \leq i \leq n$. Finde x mit $|\{y|y > x\}| = i$

1.) Idee: Sortiere und ziehe dann das i-te Element $\Rightarrow O(n \log n)$

2.) Idee: Wie Quicksort
Random. Partiion (A, q, r) zerlegt Array A zwischen Indizes p und r an zufälliges Element q in zwei Mengen $A[p, q-1]$ und $A[q+1, r]$

Random-Select (A, p, r, i)

```
if p=r then return A[p]
q <-- Random Partition(A,p,r)  (in O(r-p))
k <-- q-p+1
if i = k then A[q]
 else if i < k then return Random.Select (A,p,q-1, i)
else return Random.Select (A,q+1, r,i)
```

worst case: $O(n^2)$
ideal case: $n + \frac{n}{2} + \frac{n}{4} + \ldots + 2 + 1 = O(2n) \in O(n)$

average case: Sei $T(n)$ Zufallsvar. für die Laufzeit von Random Selection.
Random Partition zerlegt die Menge der Größe n mit Wahrscheinlichkeit $\frac{1}{n}$ in Teilmengen der Größen k und $n-k$.
Also definiere für $k = 1, 2, \ldots, n$

$$x_k = \begin{cases} 1 & \text{falls } A[p,q]k \text{ Elemente hat} \\ 0 & \text{sonst} \end{cases} \Rightarrow E(X_k) = \frac{1}{n} \text{ Beobachtung:}$$

- $T(n)$ ist monoton.
- Laufzeit ist größer, falls i in größerer Teilmenge.

$$\Rightarrow$$

$$\begin{aligned} T(n) &\leq \sum_{k=1}^{n} X_k \cdot (T(\max(k-1, n-1)) + O(n)) \\ &= \sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n) \end{aligned}$$

und damit :

$$\begin{aligned} E(T(n)) &= E(\sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n)) \\ &= E(\sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n) \\ &= \sum E(X_k) \cdot E(T_{\max}(k-1, n-k)) + O(n) \\ &= \sum_{k=1}^{n} \frac{1}{n} \cdot E(T_{\max}(k-1, n-k)) + O(n) \\ &\leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E(T(k)) + O(n) \end{aligned}$$

Da X_k und $T_{\max}$ unabhängig sind.

Behauptung: $E(T(n)) \leq c \cdot n$ für konstantes c.
$T(n) = O(1)$ falls n konstant $\surd$. Induktion:

$$
\begin{aligned}
E(T(n)) &\leq \frac{2}{n}\sum_{k=\lfloor \frac{n}{2} \rfloor}^{n} c\cdot k + a\cdot n \text{ (a ist geeignete Konstante.)} \\
&= \frac{2c}{n}(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor -1} k) + a\cdot n \\
&\leq \frac{2c}{n}\cdot(\frac{n^2}{2} - \frac{(\frac{n}{2})^2}{2}) + a\cdot n \\
&\leq \frac{2n}{n}\cdot(\frac{3}{8}n^2) + a\cdot n \\
&= \frac{3}{4}\cdot c\cdot n + a\cdot n \leq c\cdot n \text{ für } a \leq \frac{c}{4}
\end{aligned}
$$

Deterministisch

1. Teile n Elemente in $\frac{n}{5}$ Gruppen der Größe 5, sortiere sie und bestimme Median jeder Gruppe.
2. Suche rekursiv den Median x der Medianе.
3. Teile Menge S bezüglich x in 2 Mengen S_1, S_2. Sei $|S_1 \cup x| = k$ und $|S_2| = n-k$
4. `If i = k then return x`
   `else if i < k then Auswahl` $(S\_1,\ i)$
   `else Auswahl` $(S\_2,\ i-k)$

Wieso Medianе?
$\frac{1}{2}\cdot\frac{n}{5}$ der Medianе sind größer als x. Die Gruppen dieser Medianе liefern jeweils 3 Elemente, die größer sind als x; also sind mindestens $\frac{3}{10}n$ Elemente größer als x und genauso $\frac{3n}{10}$ kleiner.
$\Rightarrow |S_1|, |S_2| \leq \frac{7n}{10}$ Um $\frac{7}{10}$ schrumpft die Menge rekursiv mindestens.
Laufzeit:

$$
T(n) = \begin{cases} O(1) & \text{falls n konstant ist} \\ T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n) & \text{sonst} \end{cases}
$$

Behauptung: $T(n) = c\cdot n$ für c konstant.
Induktion:

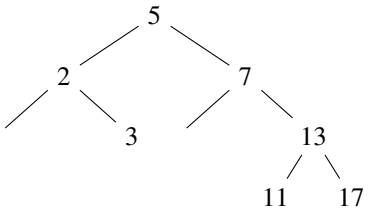
$$
\begin{aligned}
T(n) &\leq c\cdot\frac{n}{5} + c\cdot\frac{7n}{10} + a\cdot n \\
&= \frac{9}{10}c\cdot n + a\cdot n \leq c\cdot n \text{ für } a \leq \frac{c}{10}
\end{aligned}
$$

√

10.1 Satz
In zeit $O(n)$ können wir das i-te größte element aus einer ungeordneten Menge bestimmen.

§11 Dynamische Mengen

Operationen: Einfügen, Streichen, Vereinigen, Spalten, etc.
Suchbäume: Beispiel: $\{2, 3, 5, 7, 11, 13, 17\}$



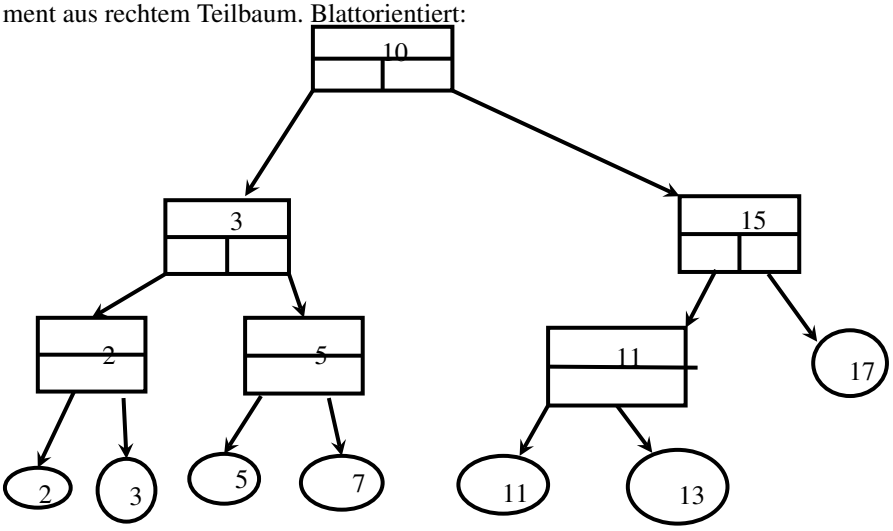
Knoten haben:

- Info-Element
- Verweise zu Kindern (lson, rson)
- oft Verweis zu parent

Knotenorientierte Speicherung:

- 1 Element pro Knoten
- Alle Elemente im linken Teilbaum von Knoten v sind kleiner als alle im rechten Teilbaum

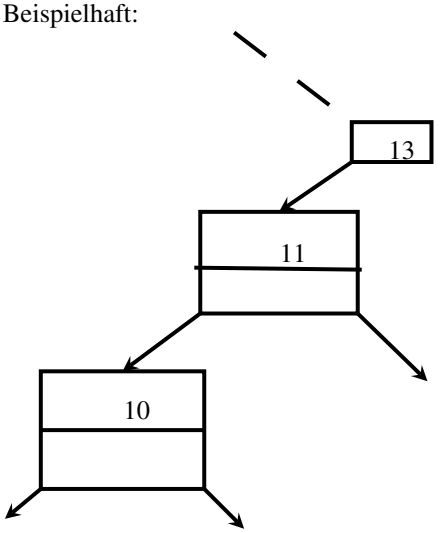
Blattorientierte Speicherung:
1 Element pro Blatt, Elemente aus linkem Teilbaum ≤ (Hilfs-) Schlüssel an v ≤ Element aus rechtem Teilbaum. Blattorientiert:



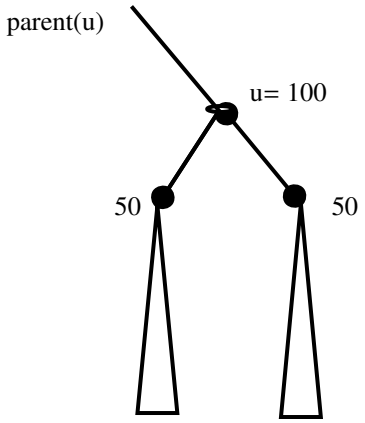

```
   Suchen(x): u ← Wurzel; found ⇐ false;
while (u existiert) and not found
```

```
do if info(u) = x then found  ⇐  true
  else if info(u)  >  x then u  ⇐  lson(u)
    else u  ⇐  rson(u)
    fi
  fi
od
```

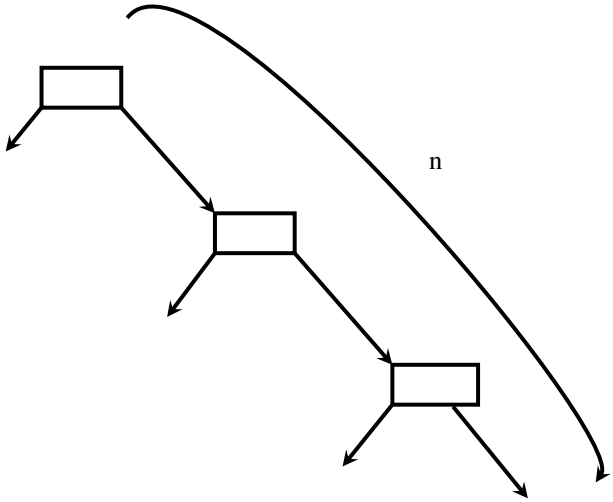
Einfügen (x) : zuerstSuchen(x)
Sei u der zuletzt besuchte Knoten u hat ≤ 1 Kind (oder $x \in S$).
Falls $x <$ info(u), erzeuge neues Kind von u mit info(v) = x als lson(u), andernfalls als rson (u).
Beispielhaft:




```
 Streichen (x); zuerst Suchen(x),
    Suchen ende in Knoten  u,  x  ∈  S
1. u ist Blatt
    streiche u, rson, lson, Verweis von parent(v) auf undef.
2. u hat 1 Kind
    Streiche u, setze w als Kind von parent(u) an die Stelle von u.
3. u hat 2 Kinder
    Suche v mit größtem Info-Element im linken Teilbaum von u, (einmal nach links, dann
     ⇒  v hat keinen rson)
    Ersetze u durch v & Streiche v unten (wie Fall 1 oder 2)
```



Laufzeit $O(h + 1)$, wobei h Höhe des Suchbaums ist.



Kann natürlich sehr schlecht sein. Beispiel: .....

Idee:

1 Hoffe, dass es nicht vorkommt.

2 Baue Baum von Zeit zu Zeit neu auf.

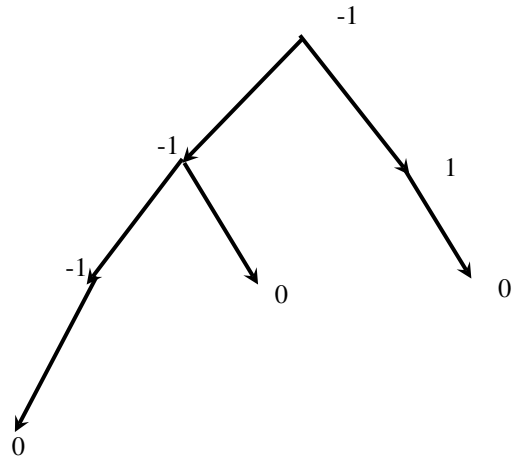
3 Balancierte Bäume.

Balancierte Bäume

Sei u Knoten im Suchbaum. Die Balance von u sei definiert als Bal(u) = Höhe des Rechten Teilbaums (TB) - Höhe des linken TB. Setze Höhe des undefinierten TBs = -1

Bal(u) = 0 -1 = -1
Bal(u) = 0-2 = -2

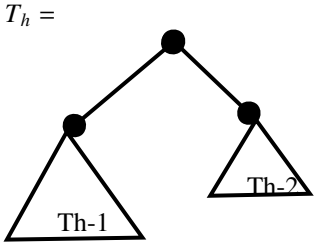
11.1 Definition Ein binärer Baum heißt AVL-Baum, falls Bal(u) $\in \{-1, 0, 1\} \forall u$



11.2 Definition Fibonacci-Bäume

$$T_0, T_1, T-2, \ldots$$

sind definiert durch T_0 = leer, $T_1 = \bullet$,
$T_h =$

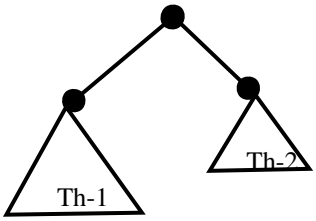


Fibonacci-Zahlen:

$$F_0 = 0, F_1 = 1, F_h = F_{h-1} + F_{h-2}$$

Beweis Behauptung: T_h enthält genau F_h Blätter (für alle $h \geq 0$) Beweis per Induktion.
Behauptung: AVL-Bäume der Höhe h haben $\geq F_h$ Blätter.
Beweis:

- $h = 0$ $\bullet\surd$
- $h = 1$ / oder \ oder /\
- $h \geq 2$: Erhalten den blattminimalen AVl-Baum der Höhe h durch Kombination von blattmin. AVL-Bäumen der Höhe $h - 1$ und $h - 2$.



$\Rightarrow \geq F_{h-1} + F_{h-2} = F_h$ Blätter.
$F_h = h$-te Fibonacci -Zahl.

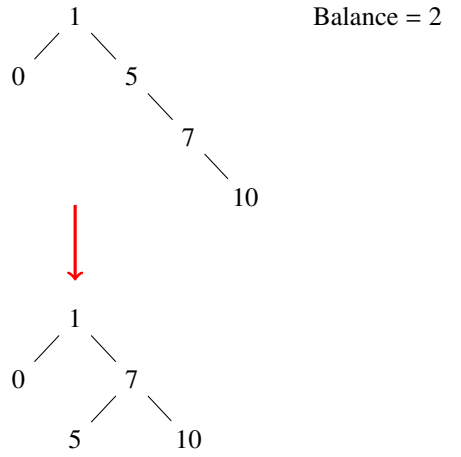
$$F_h = \frac{\alpha^h - \beta^h}{\sqrt{5}}$$

mit $\alpha = \frac{1+\sqrt{5}}{2}, \beta = \frac{1-\sqrt{5}}{2}$.

11.3 Lemma
AVL-Bäume mit n Knoten haben Höhe $O(\log n)$

Beweis Baum hat $\leq n$ Blätter, also $F_h \leq n$, damit $\frac{\alpha^h - \beta^h}{\sqrt{5}} \leq n$
Da $|\beta| < 1$: $\frac{\alpha^h - \beta^h}{\sqrt{5}} \geq \frac{\alpha^h}{2\sqrt{5}}$
$\Rightarrow \alpha^h \leq 2\sqrt{5} \cdot h$.
$h \cdot \log \alpha < \log(2\sqrt{5}) + \log n$
$h \leq \frac{\log 2\sqrt{5} + \log n}{\log \alpha} = O(\log n)$ □

Beim Einfügen kann die Balance gestört werden, z.B. auf -2 oder 2.
⇒ Rebalancierungsoperation!



Suchbäume

Bal(u) = Höhe(TB_R) - Höhe(TB_L)
Fordern: Bal(u) $\in \{-1, 0, 1\}$ (AVL)
Fibonacci-Bäume: $O(\log n)$ Höhe
Einfügen/Streichen in AVL-Bäumen: Rebalancieren.
Einfügen (w)

Können Bal. $\pm z$ entstehen
Sei u der tiefste dieser „unbal." Knoten.
O.B.d.A. sei Bal(u)=z
Sei w der neue Knoten. Der wurde rechts eingefügt. Sei v rechte Kind von u

Fall 1) Bal(v) = 1

Es gillt: 1) Links-Rechts Ordnung wird aufrechterhalten:
$T_L \leq u \leq T_A \leq v \leq T_B$
2) Nach Rotation haben u und v Balance 0.
Knoten in T_A, T_B, T_L behalten ihre alte Balance.
3) Höhe(v) nach Rotation = Höhe(u) vor Einfügen(w).

Fall 2) Bal(v) = -1
Nach Einfügen von w folge dem Pfad von w zur Wurzel.
Berechne alle Balancen neu.
Tritt Bal. -2/2 auf, führe Rotation/Doppelrotation aus.
Laufzeit: $O(\log n)$ (Rot./Doppelrot. in $O(1)$)

Fall 3) Bal(v) = 0
Kann nicht vorkommen. AVL-Bedingung vorher schon verletzt.

Streichen (w)
Annahme: es gibt Knoten mit Bal ± 2.
Sei u der tiefste solche Knoten.
O.B.d.A. sei Bal(u) = 2.
Sei v das rechte Kind von u

Fall 1) Bal(v) = 0
Höhe insgesamt hat sich nicht geändert gegenüber vor Streichen. Können hier abbrechen!

Fall 2) Bal(v) = 1
Beobachte: Der Teilbaum hat jetzt kleinere Höhe als zuvor. Balancen oben drüber ändern sich!
$\Rightarrow$ Iteriere Rebal.-Prozess weiter oben.

11.4 Satz
Baloncierte Bäume (AVL) erlauben Suchen/Einfügen/Streichen in Zeit $O(\log n)$, wobei n Zahl der Knoten.

Anwendung: Schnitt von achsenparallelen Liniensegmenten.
Wollen: # Schnittpunkte.
Annahme: allgemeine Lage.
Paarweiser Vergleich geht in $O(n^2)$
PlaneSweep
x-Struktur: geordnete Liste von Endpunkten nach x-Koor. statisch, durch Sortieren erzeugt.
y-Struktur: repräsentiert einen Zustand der Sweepline L (dynamisch). Speichern in L horiz. Segmente, die im Moment von L gekreuzt werdem.

→ AVL-Baum unterstützt Einf./Streichen in Zeit $O(\log n)$
Vertikale Segmente: (x, y_u, y_o)
Wollen berechnen # horiz. Segmente mit y-Koord. zwischen y_u und y_o
→ berechne Rang(y_u), Rang(y_o). (# Elemente, die kleiner sind)
Rang(y_o) - Rang(y_u) = # Schnittpunkt auf vert. Segment.
zu tun: Bestimme Rang(x) in AVL-Baum.
Merke in jedem Knoten die Zahl der Knoten im linken Teilbaum (l count).
Suchen(x): Beim Rechtsabbiegen erhöhe Rangzähler um (l count + 1) → Rang(x) in $O(\log n)$
⇒ Gesamtlaufzeit: $O(\log n)$

B-Bäume

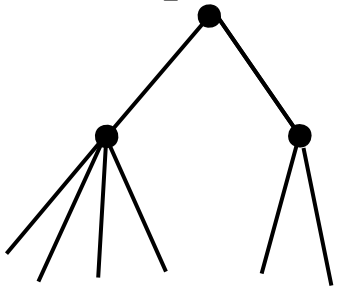
Idee: Mehr Daten in einen Knoten

11.5 Definition B-Baum der Ordnung k. $k \geq 2$ ist ein Baum dessen:

1 Blätter alle gleiche Tiefe haben

2 Wurzel ≥ 2 Kinder und dessen andere inneren Knoten $\geq k$ Kinder Haben.

3 innere Knoten $\leq 2k - 1$ Kinder haben.



Höhe eines B-Baums:

11.6 Lemma
Sei T ein Baum der Ordnung k mit Höhe h und n Blättern.
Dann gilt:

$$2 \cdot k^{k-1} \leq n \leq (2k-1)^k$$

Beweis Zahl der Blätter ist minimal, wenn jeweils die Minimal-Anzahl von Kindern vorkommen und maximal, wenn jeweils die Maximalzahl vorkommt.

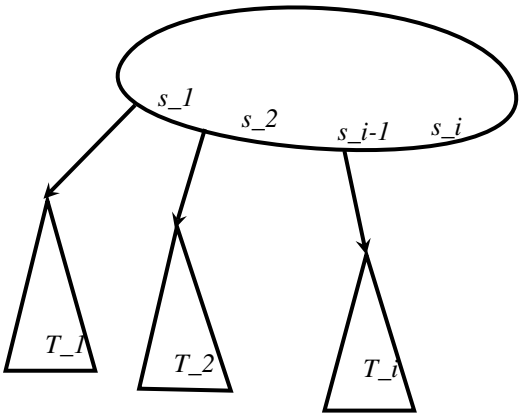
$$2 \cdot \underbrace{k \cdot k \ldots \cdot k}_{h-1} \text{ mal} = n_{\min} \leq n \leq n_{\max} = (2k-1)^h$$

Es gilt somit $\log_{(2k-1)} n \leq h \leq 1 + \log_k(\frac{n}{2})$ *(logarithmieren!)*
Operationen: Suchen(x), Einfügen(x), Streichen(x)
Sei u Knoten mit k Kindern und Schlüsseln

$$s_1 < s_2 < \ldots < s_{l-1}$$

mit Unterbäumen

$$T_1, \ldots, T_l$$



Es gilt für alle $v \in T_i$ *und Schlüssel s in v:*

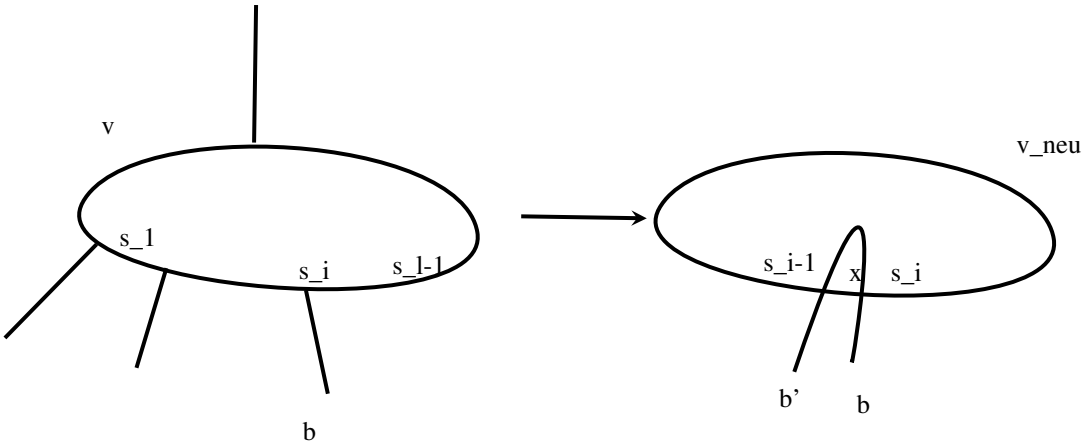
$$\begin{gathered} s \leq s_i \textit{ falls } i = 1 \\ s_{i-1} < s \leq s_i \textit{ falls } 1 < i < l \\ s_{i-1} < s \textit{ falls } i = l \end{gathered}$$

Algorithmus:

```
Suchen(x): Starte in w= Wurzel

suche kleinstes  s_i  in w mit  x  ≤  s_i

falls  s_i  existiert, dann
  if  x =  s_i  then gefunden
  else w  ←  i-tes Kind von w , suche dort weiter
falls  s_i  nicht existiert, dann
  w  ←  rechtestes Kind von w, suche dort weiter.
```

Laufzeit: $O(\log_k n \cdot k)$ suche in einem Knoten.

Einfügen(x)

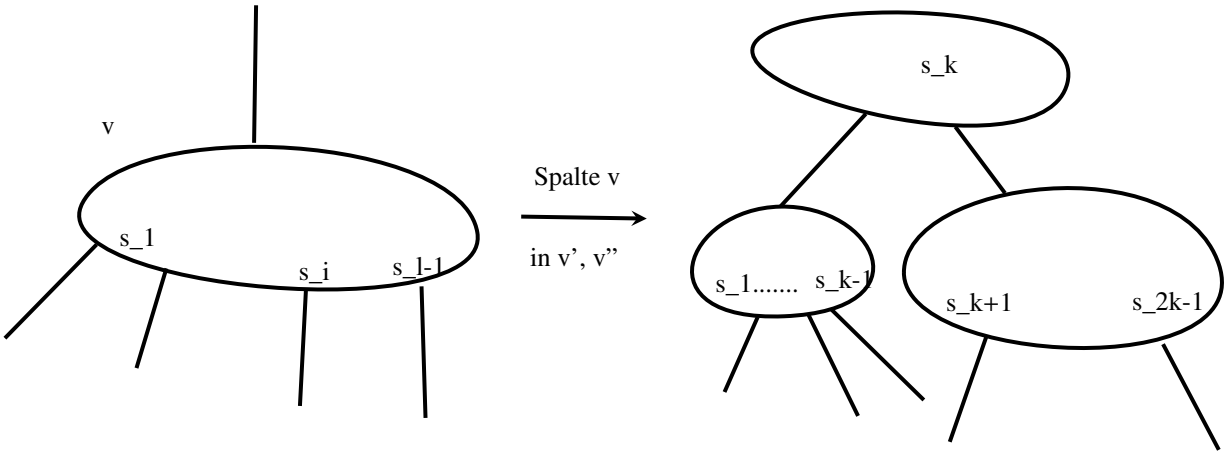


Suchen(x) endet in Blatt b mit parent v, der l Kinder hat.
Sei s_i der kleinste Schlüssel in v mit $x < s_i$, falls existiert.
Füge neuen Schlüssel x in v sowie neues Blatt b' mit:

- x links von s_i , falls $i = 1$
- x zwischen s_{i-1} und s_i falls $1 < i < l$
- x rechts von s_{l-1} falls s_i nicht existiert

Beachte: zahl der Schlüssel in v wächst.
ok, falls $l < 2k - 1$ war.

Fall $l = 2k - 1$



v', v'' haben k Kinder, also $k - 1$ Schlüssel jeweils.
Schlüssel s_k wird obne beim *parent*(v) eingefügt. *parent*(v) erhalt also ein Kind mehr, wird also eventuell auch aufgespalten. Prozeß setzt sich fort, eventuell bis zur Wurzel.
$\Rightarrow O(\log_k n \cdot k)$

11.7 Bemerkung Wird die Wurzel aufgespalten, so ergibt sich eine neue Wurzel mit 2 Kindern und Tiefe wächst um 1.

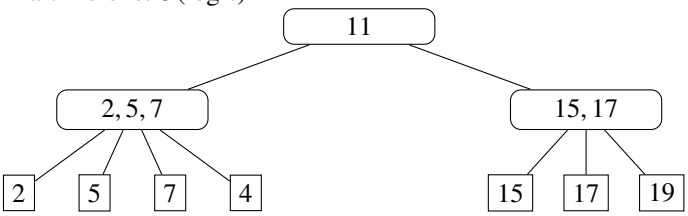
Beispiel: 2.4 Bäume: blattorientiert

1. Schlüssel in Blättern.
2. Innere Knoten v mit d Kindern hat Elemente $K_1(v), \ldots, K_{d-1}(v)$
   $k_i(v)$ = Inhalt des rechtesten Blattes im i-ten Unterbaum.

Beispiel

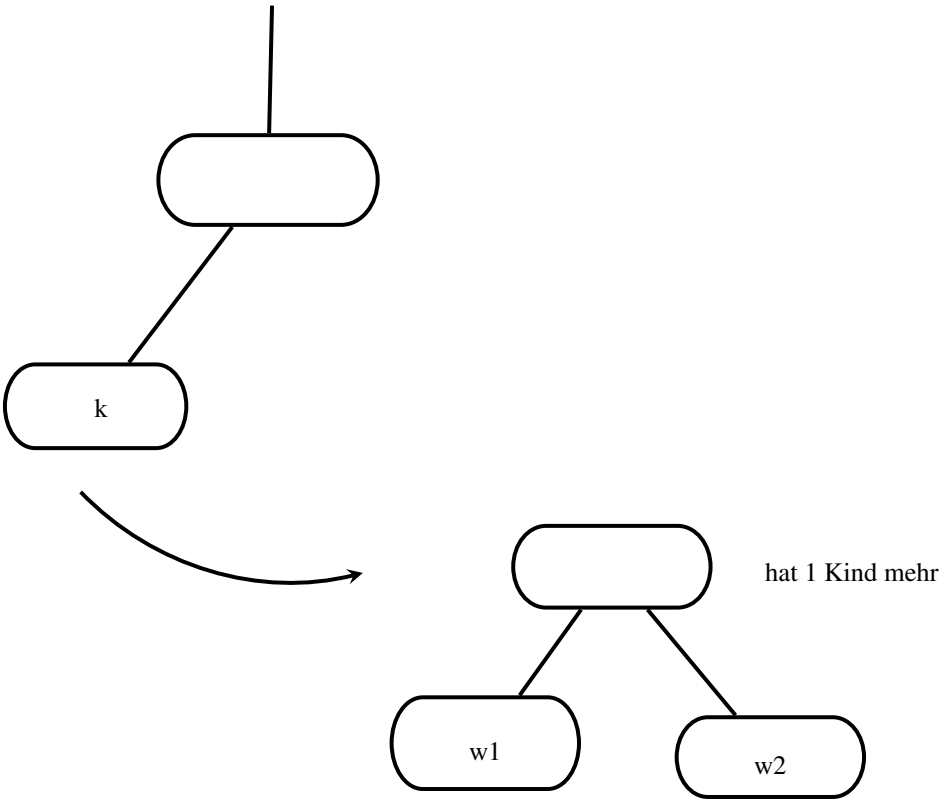
$$S = \{2, 5, 7, 11, 15, 17, 19\}$$

Klar: Tiefe ist $O(\log n)$



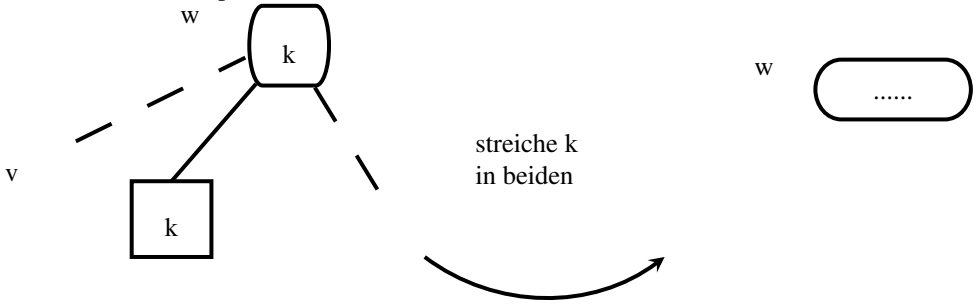
Suchen(k) $\surd$ Suche nach k liefert Blatt $k' = \min\{x \in S \mid k \leq x\}$ *Einfügen*(k): Zuerst *Suchen*(k) liefert Blatt v_i mit *Schlüssel* $(v_i) < k <$ *Schlüssel*(v_i)

```
Sei  w  der parent von  v  :

 while  w  hat 5 kinder do Spalte( w )
  w   ⇐  parent( w )
```

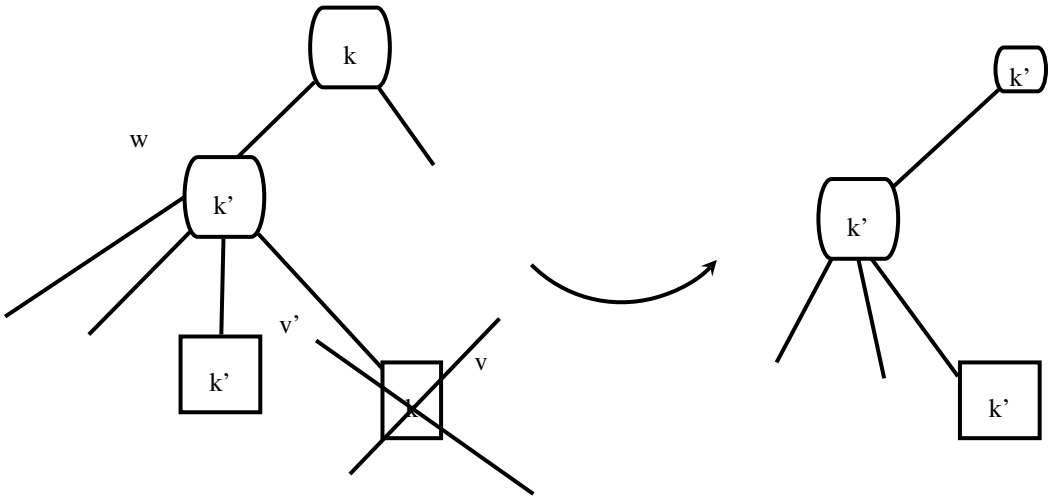


Laufzeit: $O(1 + \#\text{ Spaltungen})$
Streichen(k) : Zuerst Suchen(k) $\rightarrow$ Endet in Blatt v mit Schlüssel k.

1. Fall: K steht auch in parent(v) = w



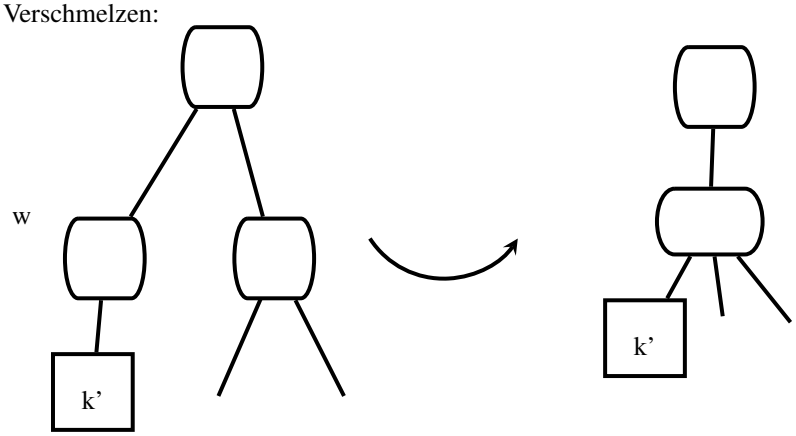
2. v rechtestes Kind w sei Vorgänger von v und v' der linke Nachbar mit Schlüssel k'.



streiche v
ersetze Vor
von k durch

```
while w hat ein Kind
  do Verschmelze oder Stehlen od
```

Verschmelzen:

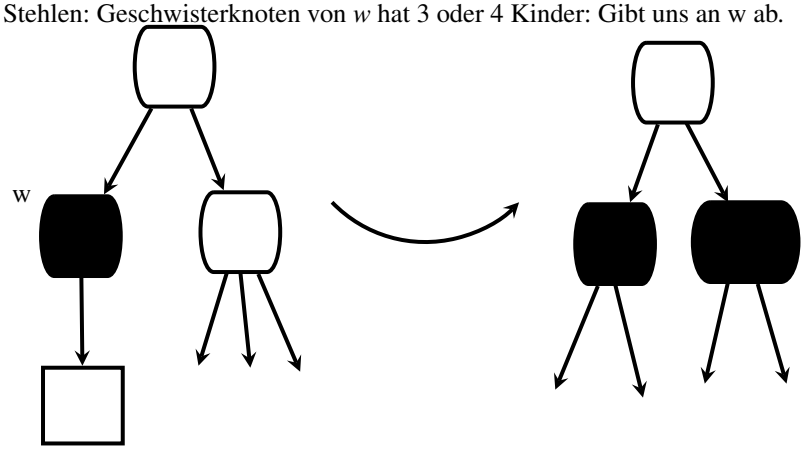


Falls Geschwister knoten grad 2 hat.

verschmelze Geschw. knoten zu einem Knoten Grad 3.

Setze fort!

Stehlen: Geschwisterknoten von *w* hat 3 oder 4 Kinder: Gibt uns an w ab.



keine Fortsetzung!

Laufzeitanalyse:
Amortisierte Analyse:

11.8 Lemma
In einem (2.4) Baum sind die amortisierten Kosten der Operationen Einfügen/Streichen $O(1)$.

Beweis Beschreibung des Zustands des Baums T:
$pot(T)$

$$\begin{aligned} &= 2 \cdot \#\text{ Knoten von Grad 1} \\ &= 1 \cdot \#\text{ Knoten von Grad 2} \\ &= 0 \cdot \#\text{ Knoten von Grad 3} \\ &= 2 \cdot \#\text{ Knoten von Grad 4} \\ &= 4 \cdot \#\text{ Knoten von Grad 5} \end{aligned}$$

Invariante:

- $pot(T) \geq 0$
- bei Spalten/Verschmelzen/Stehlen ist nur ein Knoten nicht auf Grad 2,3,4
- Vor Einfügen /Streichen haben alle Knoten den Grad 2,3,4.

Einzeloperationen haben Kosten von $O(1)$.
Behauptung: Spalten / Verschmelzen verringern Potential. Stehlen erhöht es nicht.
Beweis:
Stehlen. Knoten w trug 2 Einheiten zum Potential bei und sein Nachbarknoten p. Danach trägt w 1 Einheit bei, Nachbar $\leq p + 1$. □

Spalten: w trägt 4 bei zum Potential, der parent trägt p bei. Danach haben die beiden neuen Knoten 0 und 1, der parent hat $\leq p + 2$ √.

Verschmelzen: w trägt 2 bei, Geschwister von u trägt 1 bei, der parent trägt p bei
$\Rightarrow$ Danach 0 und $\leq p + 1$.

Amortisierte Laufzeit für Einfügen: Tatsächliche Kosten + Potentialerhöhung.
Tatsächliche Kosten 1 + Potentialerhöhung ≤ 2
Folge on f Spaltungen: tatsächlcihe Kosten f, Potentialerhöhung $\leq f$.
$\Rightarrow$ Amortisierte Kosten von Einfügen: ≤ 3

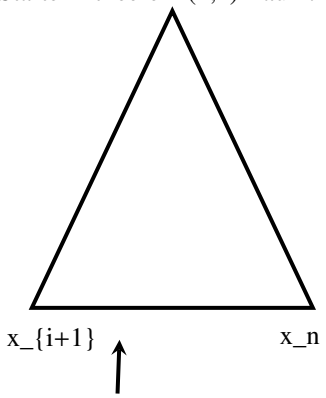
Sortieren:
durch Einfügen: $O(n \log n)$ vorsortierter Folgen: Mache Suchen billiger!
Sei $x_1, \ldots, x_2$ Folge reeller Zahlen.

$$F = \left|\{(i, j) \mid i < j \text{ und } x_i > x_j\}\right|$$

(Zahl der Inversionen). Es gilt $O \leq F \leq \frac{n^2}{2}$

11.9 Satz
$x_1, \ldots x_n$ kann in Zeit $O(n \log \frac{F}{n})$ sortiert werden.

Beweis Sei $f_i = \lceil \{j \mid i < j, x_i > x_j\} \rceil$ Es gilt $F = \sum_i f_i$
Starte mit leerem (2,4) Baum. Füge Folge in umgekehrter Reihenfolge ein.



Füge $x_n, \ldots, x_1$ ein. Einfügen (x_i): Starte am linkesten Blatt; laufe hoch und drehe an der richtigen Stelle um, laufe runter und füge ein. Ist x_i klein, so laufe ich nicht sehr hoch.
Es gilt: Kosten für Einfügen = $O(1 + \log f_i)$
Suchen(x_i) = $O(\log f_i)$
Einfügen: = $O(1)$.

Laufe hoch bis Höhe h:

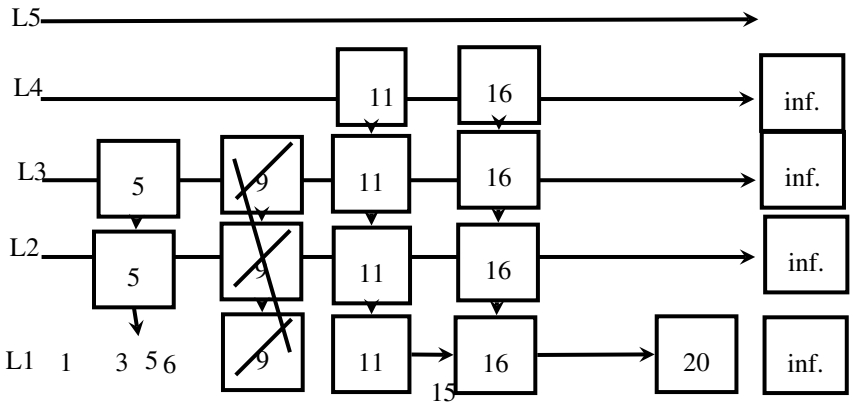
$$2^{n-2} \leq f_i \Rightarrow h \leq 2 + \log f_i$$

$\Rightarrow$ Gesamtkosten: $\sum_i O(1 + \log f_i) = O(n + \sum_{i=1}^{n} \log f_i)$
$\leq O(n + n \log \frac{F}{n})$ □

Randomisierte Suchbäume

Treaps(Aragorn/Seidel)

Skiplist Beispiel



Skipbits (Pugh) Gesucht werden: $S \subseteq U$, S Schlüssel, $|S| = n$
$S = \{x_1, \ldots, x_n\}$, $p = (\frac{1}{2})$
Levelzuweisung $L : S \to \mathbb{N}$
$S = L_1 \supseteq L_2 \supseteq \ldots \supseteq L_V$ mit letztem nichtleerem Level.
Levelzuweisung:

```
Starte mit  L_1  =  S

Gehe zu  L_2 . Mache Münzwurf für jedes Element aus dem Level darunter
und kopiere es hoch, mit Wahrscheinlichkeit  p .

Level  t  sei das höchste mit  x . Die Wahrscheinlichkeit dass Schlüssel  x

auf höchstem Level  k  ist, ist
$$p^k-1 (1-p) $$
```

Erwartungswert:

$$\sum_{k=1}^{\infty} k \cdot p^{k-1}(1-p) = (1-p)\sum_{k=1}^{\infty} k \cdot p^{k-1} = (1-p) \cdot \frac{1}{(1-p)^2} = \frac{1}{1-p}$$

höchster Level:(erwartet)
Sei $v = \max\{L(t) \mid t \in S\}$
$L(x)$ hat den Erwartungswert $\frac{1}{1-p}$
Wahrscheinlichkeit, dass $L(t) > t$ ist, ist $< p^t$
Levelzuweisung der Elemente ist unabhängig, d.h. die Wahrscheinlichkeit dass ein Element (aus n vielen) Level $2t$ hat, ist $< n \cdot p^t$
Setze $t = c \cdot \log n$, $p = \frac{1}{2}$

$\Rightarrow prob(v > c \cdot \log n) \leq \frac{n}{2^{c \log n}} < \frac{1}{n}$ für $c > 1$
Höhe des Skipbits ist demnach $O(\log n)$

Suchen(x)

Starte im Level v mit Header. Laufe nach rechts in L_v, bis Element $> x$ gefunden wird. Im Element zuvor gehe nach unten. Iteriere.
Suche durchläuft Skipbits von rechts oder hin zu x. Strategie: Gehe möglichst nach rechtst, wenns nicht mehr geht, gehe nach unten.

Laufzeit

Anzahl Level: $O(\log n)$, Anzahl Schritte nach unten.
Der Suchpfad von x zurück zu Level v wird beschrieben durch:
Gehe nach oben, wenn möglich; wenn nicht gehe nach links.
Schritt nach links gibt es mit Wahrscheinlichkeit $\frac{1}{2}$ jeweils $\Rightarrow$ Erwartete Anzahl der Schritte nach links ist genauso groß, wie erwartete Anzahl der Schritte nach oben. $\Rightarrow$ Laufzeit $O(\log n)$

Einfügen(x)

Suche zuerst nach x. Füge es in L_1 ein und hänge zwei Zeiger um.
Laufe hoch und mache ZUfallsexperiment, füge nacheinander eventuell Kopien ein. Erwartete Zahl von Kopie ist $2 \Rightarrow O(1)$ erwartet.

Streichen(x)

Während dem Suchlauf nach x lösche von oben nach unten alle Vorkommen von x durch Zeigerumhängen. Laufzeit: $O(\log n)$.

§12 Hashing

Wörterbücher
Werte V
Keys $U = [0, \ldots, N-1]$
$S \subseteq U$
S sehr klein im Vergleich zu U .
Operation Zugriff s / Einfügen(s, v) / Streichen s (möglichst in $O(1)$)

- Hashtable $T[0, \ldots, m-1]$
- Hashfunktion h: $U \to [0, \ldots, m-1]$

$s \to T[h(s)]$
Problem: Kollission

$$\begin{aligned} h(x) &= h(y) \\ x &\neq y \\ x, y &\in U \end{aligned}$$

Hashing mit Verkettung

Idee:
Jeder Tafeleintrag ist eine Liste. Die i-te Liste enthält alle $x \in S$ mit $h(x) = i$.
underlineLaufzeit:
worstcase: $O(|S|) = O(n)$, im Mittel aber viel besser.
Folgende Annahmen:

(1) $h(x)$ kann in $O(1)$ ausgerechnet werden

(2) $\left|h^{-1}(i)\right| = \frac{|U|}{m}$
Gleichmäßige Verteilung der Elemente mit gleichem Schlüssel)

(3) Für eine Folge von Operationen gilt: Wahrscheinlichkeit, daß das j-te Element in der Folge ein festes $x \in U$ ist, ist $\frac{1}{N} \to$ Operationen sind unabhängig voneinander.
x_k Arguement der k-ten Operation ist $prob(h(x_k) = i) = \frac{1}{N}$

12.1 Definition $\delta_n(x, y) = \begin{cases} 1 & x \neq y \;\; h(x) = h(y) \\ 0 & \text{sonst} \end{cases}$
$\delta_n(x, S) = \sum_{y \in S} \delta_n(x, y)$
Kosten von Operation $XYZ(x) = 1 + \delta_n(x, S)$

12.2 Satz
Die Mittleren Kosten von $XYZ(x)$ *sind* $1 + \beta = 1 + \frac{n}{m}$

Beweis Sei $h(x) = i$ und p_{ik} sei die Wahrscheinlichkeit, daß Liste i genau k Element enthält. Dann ist:

$$p_{ik} = \binom{n}{k} (\frac{1}{m}^{k} (1 - \frac{1}{m})^{n-k}$$

Die mittleren Kosten sind:

$$\begin{aligned}
&\sum_{k=0}^{n} p_{ik}(1+k) \\
&\qquad = \sum_{k=0}^{n} p_{ik} + \sum_{k=0}^{n} k\binom{n}{k}(\frac{1}{n})^k(1-\frac{1}{m}^{n-k} \\
\text{Nebenrechnung: } k\binom{n}{k} &= n\binom{n-1}{k-1} \\
&= 1 + \frac{n}{m}\sum_{k=1}^{n}\binom{n-1}{k-1}(\frac{1}{m}^{k-1}(1-\frac{1}{m})^{n-k} \\
&= 1 + \frac{n}{m}(1+(1-\frac{1}{m}))^{n-1} \\
&= 1 + \frac{n}{m} \\
&= 1+\beta
\end{aligned}$$

□

Perfektes Hashing

$U, S \subseteq U$, $|S| = n$
T der Größe m.

$$h : U \to \{0, \ldots, m-1\}$$

$S \to V$
Grundidee:

$$m = \vartheta(n^2) \Rightarrow \text{ Viele Injektive Hashfunktionen}$$

2- Stufen-Verfahren:

1. Stufe: hashing auf Tabelle der Größe m mit Chaining Erzeugt "kurze Listen".

2. Stufe: Für jede Liste Hashing mit injektiver Hashfunktion.

Wahl einer injektiven Hashfunktion:
$U = \{0, \ldots, N_1\}$; $h_k : \{0, \ldots, N-1\} \to \{0, \ldots, m-1\}$
$h_k(x) = (k \cdot x \mod N) \mod m$
$S \subseteq U$
$|S| = n$ und S bekannt .
Messung der "Injektivität" von h_k:

$$b_{ik} = |\{x \in S : h_k(x) = i\}| ; \ 1 \le k \le N-1, 0 \le i \le m-1$$

Dann ist

$$b_{ik} \cdot (b_{ik}-1) = \left|\{(x,y) \in S^2, x \neq y; \ h_k(x) = h_k(y)\}\right|$$

die Anzahl der Paare in S die die "Injektivität" an der Stelle i verletzen.
Setze

$$B_k = \sum_{i=0}^{m-1} b_{ik}(b_{ik}-1)$$

(Zahl der Paare in S^2, die die Injektivität verletzen)

12.3 Lemma
Es gilt:

1.) $B_k < 2$*, dann ist* h_k *injektiv auf* S.

2.) Falls $b_{ik} = \frac{n}{m}$ $\forall i \in \{0, \ldots, m-1\}$*, dann ist*

$$B_k = \sum_{i=0}^{m-1} \frac{n}{m}(\frac{n}{m} - 1) = n(\frac{n}{m} - 1)$$

Beweis 2. ist Offensichtlich
1. $\Rightarrow$:

$$\begin{aligned} B_k &< 2 \\ \sum b_{ik}(b_{ik} - 1) &\in \{0, 1\} \\ &\Rightarrow \forall i\ b_{ik} \in \{0, 1\} \\ &\Rightarrow h_k I_s \text{ ist injektiv} \\ &\Leftarrow h_k I_s \text{ injektiv} \\ &\Leftarrow \forall i\ b_{ik} = 0 \\ &\Rightarrow B_k = 0 \end{aligned}$$

□

12.4 Lemma
Sei N Primzahl, dann gilt:

$$\sum_{k=1}^{N-1} \sum_{i=0}^{m-1} b_{ik}(b_{ik} - 1) \leq 2 \cdot \frac{n(n-1)}{m}(N - 1)$$

12.5 Bemerkung $\Rightarrow$ mittlere Anzahl von Kollissionen $\leq \frac{n(n-1)}{m}$, da $B_k \approx 2\cdot$ Anzahl der Kollissionen)
$\Rightarrow$ Mit $m > n(n-1)$ ist die mittlere Anzahl < 1 , das heisst es gibt ein h_k , dass auf S injektiv ist.

Beweis

$$\begin{aligned} &\sum_{k=1}^{N-1} \sum_{i=0}^{m-1} b_{ik}(b_{ik} - 1) \\ &\qquad = \sum_{k=1}^{N-1} \sum_{i=0}^{m-1} \left|\{(x, y) \in S^2 \middle| x \neq y,\ i = h_k(x) = h_k(y)\}\right| \\ &\qquad = \sum_{(x,y) \in S^2, x \neq y} |\{k \mid h_k(x) = h_k(y)\}| \end{aligned}$$

□

Seien $x, y \in S$ fest. Wie viele Funktionen h_k mit $h_k(x) = h_k(y)$ gibt es ?

$$\begin{aligned} h_k(x) = h_k(y) &\Leftrightarrow (k \cdot x \mod N) \mod m = (k \cdot y \mod N) \mod m \\ &\Leftrightarrow \underbrace{((k \cdot x \mod N) - (k \cdot y \mod N))}_{g(k)} \mod m = 0 \end{aligned}$$

Es gilt: $-N + 1 \leq g(k) \leq N - 1$
$g(k) = m \cdot i$
$i \in \{\lfloor \frac{-N+1}{m} \rfloor, \ldots, \lfloor \frac{N-1}{m} \rfloor\}$

Beweis Behauptung: Für jedes i gibt es höchstens ein k mit $g(k) = i \cdot m$.
Falls diese Behauptung wahr ist:

$$|\{k|h_k(x) = h_k(y)\}| \leq 2 \cdot \left\lfloor \frac{N-1}{m} \right\rfloor$$

$$\Rightarrow \sum_{i=1}^{N-1} \sum_{i=0}^{m-1} b_{ik}(b_{ik} - 1) \leq 2 \cdot \frac{N-1}{m} \cdot n(n-1)$$

Beweis der Behauptung
Seien $k_1, k_2 \in [0, \ldots, N-1]$ mit $g(k_1) = g(k_2)$

$$\begin{aligned} &\Rightarrow k_1 x \mod N - k_1 y \mod N = k_2 x \mod N - k_2 y \mod N \\ &\Rightarrow (k_1 x - k_1 y) \mod N = (k_2 x - k_2 y) \mod N \\ &\Rightarrow (k_1 x - k_1 y) - (k_2 x - k_2 y) \mod N = 0 \\ &\Rightarrow (k_1 - k_2)(x - y) \mod N = 0 \end{aligned}$$

Da $\mathbb{Z}/N\mathbb{Z}$ Körper, folgt entweder $k_1 - k_2 \mod N = 0$ oder $x - y \mod N = 0$
$\Rightarrow k_1 = k_2$ □

12.6 Korollar

1. $\exists k \in [1, \ldots, N-1] : B_k \leq 2 \cdot \frac{n(n-1)}{m}$
2. *Mindestens* $\frac{N-1}{2}$ *viele* h_k*s haben* $B_k \leq \frac{4n(n-1)}{m}$

Beweis

1. $2 \cdot \frac{n(n-1)}{m}$ ist Mittelwert
   $\Rightarrow \exists h_k$ mit $B_k \leq 2\frac{n(n-1)}{m}$
2. $A = \{k|B_k > \frac{4n(n-1)}{m}\}$
   Angenommen: $|A| > \frac{N-1}{2}$ dann

$$\sum_{k=1}^{N-1} B_k \geq \sum_{k \in A} B_k > \frac{N-1}{2} : 4\frac{n(n-1)}{m} = 2 \cdot \frac{n(n-1)}{m} \cdot (N-1)$$

   Wiederspruch zum Lemma!
   Also für quadratisch große Hashtabeln gibt es „viele“ injektive Hashfkt. □

12.7 Korollar

1. *Ein $k \in [1, \ldots, N-1$ mit $B_k \leq 2\frac{n(n-1)}{m}$ ist in Zeit $O(n + N \cdot n)$ bestimmbar.*
2. *Sei $m = n(n-1) - 1$. Dann gibt es k mit $h_k|s$ injektiv. k ist in Zeit $O(n^2 + Nm)$ zu finden.*
3. *Sei $m = 2n(n-1) + 1$ Dann ist mindestens die Hälfte der h_k auf s injektiv. Bestimmungszeit $O(n^2)$*

Beweis

1. Teste alle k durch. Es gibt $N-1$ viele und n viele $x \in S$. x in Fächer werfen $O(n \cdot N)$. Teste, ob Fächer in voll sind $O(m)$
2. Einsetzen bei 1 $B_k < 2 \Rightarrow$ injektiv
3. Mindestens die Hälfte der h_k haben B_k mit $B_k < 2$
   $\Rightarrow$ mindestens die Hälfte sind injektiv. □

12.8 Korollar

a) $k \in \{1, \ldots, N-1\}$ mit $B_k \leq 4\frac{n(n-1)}{m}$ randomisiert in Zeit $O(m+1)$ bestimmbar.

b) Sei $m = n, k$ mit $B_k \leq 2(n-1)$ in Zeit $O(n \cdot N)$ zu finden.

c) Sei $m = n, k$ mit $B_k \leq 4(n-1)$ randomisiert in Zeit $O(n)$ zu finden.

Realisierung von perf. Hashing

Stufe 1 $m = n$
Es gibt Hashfkt. h_k, so dass die Listenmengen alle $O(\sqrt{n})$
$\sum_{i=0}^{n+1} b_{ik}(b_{ik}-1) = B_k \leq 2\frac{n(n-1)}{m} \overset{n=m}{=} 2(n-1)$
$\Rightarrow \; b_{ik} \in O(\sqrt{n})$

1. $|\sigma| = n = m$
   Wähle $B_k \leq k(k-1)$
   $h_k = (kx \mod N) \mod n$
2. $w_i = \{x \in S | h_k(x) = i\}$ $b_i = |w_i|$
   $m_i = 2b_i(b_i - 1) + 1$
   Wähle k_i mit $h_{k_i}(v) = (k_i x \mod N) mod m_i$
   h_{k_i} injektiv auf w_i
3. $s_i = \sum_{j<i} m_j$
   $s \in S$ in $T[s_i + j]$
   $i = (kx \mod N) \mod m$
   $j = (k_i x \mod N) \mod m$
   Platzbedarf
   $\sum_{i=0}^{m} m_i = \sum_{i=0}^{m-1} b_{ik}(b_{ik}-1) + 1 = n + ZB_{ik} \leq n + 8(n-1) = (9n-8) \in O(n)$

Laufzeit 1.Schritt: rand. $O(n)$
2.Schritt: $O(n)$ rand. $b_{ik} \in O(\sqrt{n})$
3.Schritt: $O(m_i + b_i)$ d.h. insgesamt $O(n)$
$\Rightarrow\ O(n+m) = O(n)$ rand.

12.9 Satz
Es kann für S eine perfekte Hashtafel der Größe $O(n)$ und Zugriffszeit $O(N)$ deterministisch in Zeit $O(n \cdot N)$ und randomisiert in $O(n)$ erwarteter Zeit aufgebaut werden.

Tafelaufbau Wähle $k \in [1, \ldots, N-1]$

```
for i=0 to n-1 do
  Initialisierung
  Wähle k_{i} zufällig
od
m <-- n, B <-- 0
for l = 0 to n-1 do
  Insert(x_{l})
od
```

Insert(x)

```
i <-- (kx mod N) mod n
j <-- w(x)
B <-- B + 2b_{i}
if B > 4(n-1) then
  Tafel aufbauen
else if T[s_{i} + j] frei then
       speichern x in T[s_{i} + j]
     else if m_{i} < 2b_{i}(b_{i}-1)+1 then
            vergößere Bereich für w_{i}
          fi
          Versuche Wieder:
            wähle k_{i} zufällig
            lösche Bereich T[s_{i}, ..., s_{i} + m_{i} - 1]
          for all x \in w_{i} do
            j <-- (( k_{i}x mod N) mod m_{i})
            if T[s_{i} + j] frei then
              Speichere x
            else goto versuche wieder T[s_{i}+j]
            fi
          od
     fi
fi
```

Universelles Hashing
$\overline{(ax + b_i \mod n) \mod m}$
$\frac{|H|}{m}$
Bloomfilter

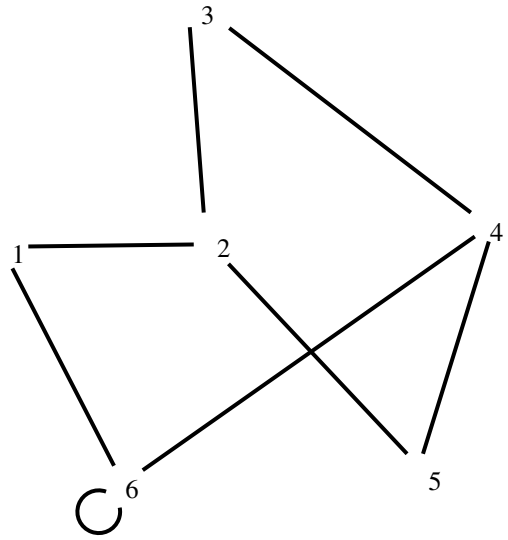
$S \subseteq U$
add
Query
T Vektor von $\underline{m}$ Bits, initial auf O
K Hashfkt. h_i

$add(x)$	$query(x)$
$l_i \leftarrow h_i(x)$	$\wedge T[h_i(x)] = 1$
$T[l_i] = 1$	

§13 Graphen

$G = (V, E)$
V beschreibt die Menge der Knoten, E die Menge der Kanten.
$E \subseteq V \times V$
$e = (v, w) \in E$ heißt Kante von v nach w.
w ist der Nachfolgeknoten von v (Adjazent)
Pfad in G ist Folge $(v_o, \ldots, v_k)$ von Knoten mit $k \geq 0$ und $(v_i, v_{i+1}) \in E \forall 0 \leq i < k$ (Pfad von v_0 nach v_k).
falls $v_0 = v_k$ und $k \geq 1$ heißt Pfad Kreis/Zykel
falls $v_i \neq v_j$ für $i \neq j$ so ist der Pfad einfach
Pfad von v nach w $v \vec{*}$
Graph heißt zyklisch, falls er mindestens einen Kreis enthält und sonst azyklisch

Darstellung:
Annahme: $V = \{1, \ldots, n\}$
1. Adjazenzmatrix A:
$(a_{ij}) = \begin{cases} 1 & \text{falls } (i, j) \in E \\ 0 & \text{sonst} \end{cases}$ Bei ungerichteten Graphen:



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ . & 0 & 0 & 0 & 1 & 1 \\ . & . & . & 0 & 0 & 0 \\ . & . & . & 0 & 0 & 1 \end{pmatrix}$$
symmetrisch da $a_{ij} = a_{ji}$

Platz: $O(n^2)$, ist okay falls $m = |E| \approx n^2$
meistens aber nur $m \approx O(n) \rightarrow$ "dünne" Graphen.

Dünne Graphen

- Bäume mit n Knoten haben $n - 1$ Kanten
- *planare Graphen*: Graphen, die kreuzungsfrei erreichbar sind. ($m \leq 3n - 6$ Eulerformel)

$\Rightarrow$ verbotene Konfigurationen

2. Adjazenzlistendarstellung: speichern für jeden Knoten v die Nachbarknoten von v:

- $\texttt{Inadj}(v) = \{w \in V \mid (w, v) \in E\}$
- $\texttt{Outadj}(v) = \{w \in V \mid (v, w) \in E\}$
- Wenn G ungerichtet ist auch:
  $\texttt{Adj}(v) = \{w \in V \mid \{v, w\} \in E\}$

Platzbedarf: $O(n + m)$
Nachteil: Zugriff auf die Liste geht in $O(\texttt{Outdegree}(v_i))$
Wobei $\texttt{Outdegree}(v)$ = Anzahl der von v ausgehenden Kanten.
Grad(v) = Degree(v) = # Nachbarn von v.

a. V hat genau ein v_0 mit $\texttt{indegree}(v) = 0$

b. $\forall v \in V \backslash \{v_o\} : \texttt{indegree}(v) = 1$

c. G ist azyklisch

Billigste Wege
Netzwerk (V, E, c)
gerichteter Graph $G = (V, E)$; Kostenfunktion $c : E \rightarrow \mathbb{R}$
Kosten eines Pfades $v_0 \ldots v_n = p$

$$c(p) = \sum_{i=1}^{c} (v_{i-1}, v_i)$$

Probleme:

1 single pair shortest path

2 single source shortest path

3 all pairs shortest path

13.1 Bemerkung Es ist nicht bekannt ob 1.) leichter als 2.). Wir betrachten zunächst Problem 2.

Für ein $n \in V$ sei $P(s,n)$ die Menge aller Pfade von v nach u

$$\sigma(n) = \begin{cases} \infty & fallsP(s,b) = \emptyset \\ inf(c(p) \mid p \in P(s,n)) & \text{sonst} \end{cases}$$

13.2 Bemerkung $-\infty$ Pfade können existieren.

negativer Zyklus: zyklischer Pfad p mit $c(p) < 0$

13.3 Lemma
Sei $u \in V$, dann gilt:

i $\sigma(u) = -\infty \Leftrightarrow$ *u erreichbar von negativem Zyklus, da von s aus erreichbar ist.*

ii $\sigma(u) \in \mathbb{R} \Rightarrow \exists$ *billigster Weg von s nach u mit* $\sigma(u)$

Beweis 1 "$\Leftarrow$" nach Definition.
"$\Rightarrow$" Sei $C = \sum |c(e)|$ und sei p ein kürzester Weg von s nach u mit $c(p) < -c$
$\Rightarrow$ p muß Zyklus enthalten, Zyklus q (p nicht einfach).
wäre $c(q) \geq 0$, könnten wir q aus p entfernen, und so p verkürzen $\Rightarrow c(q) < 0$

2 $\sigma(u) < \infty \Rightarrow$ es gibt einen Pfad von s nach u. Behauptung: $\sigma(u) = \min\{c(p) \mid p \in P(s,u) \cup p \text{ einfach}\} \Rightarrow u$
Beweis: Sei p^* ein billigster, einfacher Weg von s nach u. Ist die Behauptung falsch, dann gibt es ein nicht nicht einfachen Weg q von s nach u der $c(q) < c(p)$. Durch Entfernen des Zyklus aus q entsteht ein kürzerer Weg q' und da der Zyklus nichtnegativ (nach 1) ist

$$c(q') < c(q) < c(p^*) \rightarrow \text{Widerspruch}$$

Beispiel 1 Beispiel: Graph ist azyklisch.
allgemeine Annahme: Knoten topologisch sortiert., also $V = \{1, 2, \ldots, n\}$

```
d(s)  ←  0,

Pfad(s)  ←  s;
```

```
 for  v ∈ V \ { s } do
d (v)  ← ∞
    od,
 for v  ←  s+1 to n do
    d(v)  ←   min { d(u) + c(u,v) | (u,v) ∈ E }  =  u*
    Pfad(v)  ←  Pfad(  u*  Ilv. (Konkatenation)
```

13.4 Lemma
Nach Ausführung gilt:

1. $d(v) = \sigma(v), v \in V$
2. $d(v) < \infty \Rightarrow$ *Pfad*(v) *billigster Weg von s nach v.*

Beweis Induktion über v:

- v < s: $\sigma(v) \to \infty$, da v nicht erreichbar $d(u) \to \infty$, da nicht verändert.
- v = s: $\sigma(v) = d(u) = 0$ Pfad(s) = $\{s\}$
- v > s: Sei In(v) die Menge der Knoten mit $In(v) = \{u \in V \mid (u, v) \in E\}$
  Für $u \in In(v)$ gilt $\sigma(u) = d(u)$ (IV) und Pfad (U) schon berechnet.
  Ist $In(v) = \emptyset$, dann $\sigma(v) = d(u) \to \infty$, sonst wählen u^* mit $d(u^*) + c(u^*, v)$ minimal.
  Ist $d(u^*) \to \infty$ so auch $d(v) = \sigma(v) \to \infty$. Sonst ergibt der billigste Weg nach v durch Konkatenation und

$$\sigma(v) = \sigma(u^*) + c(u^*, u) = d(v)$$

13.5 Bemerkung Laufzeit:

$$\sum_v |\text{In}(v)| = \sum_v indig(v) = m$$

Konkatenation in $O(m)$

2. Situation: $c(e) \geq 0 \forall e \in E$, Zyklen erlaubt.
Dijkstra's Algorithmus. S die Menge der Knoten mit berechnetem $d(u)$ S' Menge der Knoten aus $V \backslash S$, da Nachbarn in S haben.

Algorithmus
$S \leftarrow \{s\}; d(s) \leftarrow 0$,
$S' \leftarrow out(S)$,
$\forall u \in S' : d'(u) = c(s, u)$ $\forall u \in V \backslash \{S, S'\} : d'(u) \to \infty$

```
while S ≠ V do
  wähle w ∈ S' geeignet (das mit dem kleinsten d'-Wert)
  d(w) <-- d'(w)
  S <-- S ∪ {w}, S' <-- S'\{w}
  for all n ∈ out(w) do
     if n ∉ S then
       S' <-- S' ∪ {n}
       d'(n) <-- min  {d'(n), d(w) + c(w,n)}
     fi
  od
od
```

Beispiel Dijkstra

S	S'	d/d'
s	2,1,5	d(s) = 0 d'(1) = 1, d'(2) = 2 d'(5) = 5
s,1	2,5 +4	d(1) = 1 d'(4) = 4 d'(5) =3
s,1,5	2,4	d(5) = 3
s,1,5,2	3,4	d(2) = 2 d'(3) = 3
s,1,5,2,3	4	d(3) = 3 d'(4) = 4
s,1,5,2,3,4	$\emptyset$	d(4) = 4

Laufzeit:

Implemetierung: S, S' als Bitvektor
d, d' als Array

- $\forall n \in out(n) \quad \sum_{w} |out(n)| = O(m+n)$
- n-Mal minimal über $S' \rightarrow O(n) \Rightarrow O(n^2)$

$\Rightarrow$ $O(n^2 + m)$ (dichte Graph $m \approx n^2$, ok)

alternativ: S' in Heap (balanciert, nach d' geordnet)
$O((n+m)\log n)$ gut für dimere Graphen $m << n^2$
Verbesserung: Fibonacci-Heap als Datenstruktur

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

13.6 Lemma
Sei $w \in S'$ so gewählt, dass $d'(w)$ minimal, dann ist $d'(w) = \delta(w)$

Beweis Sei p billigster Weg von s nach w mit allen Knoten (bis auf w) in S
Annahme Es gäbe einen billigeren Weg q von s nach w.
q muß einer ersten V\ S haben und nach Wahl von w und ersten Kanten u:

$$d'(u) \geq d(w)$$

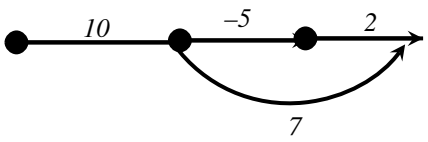
Da alle Kanten nicht negativ sind, gillt:

$$c(q) \geq d'(u) \geq d(w) = c(p) \lightning$$ □

13.7 Lemma
$G = (V, E, c)$ $c : E \to \mathbb{R}$, SSSP, APSP.

1 G azyklisch

2 G gerichtet, $c : E\mathbb{R}^+$, Dijkstra's Algorithmus, in $O((n+m)\log n)$



3

Beispiel (Bellman-Ford Algorithmus) erlaube negative Kantenkosten, aber keine negativen Zyklen.

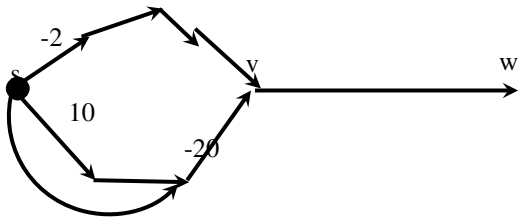
```
Relax(v,w): d(w)  ←   min  { d(w), d(v) + c(v,w) }
```

13.8 Lemma
Relaxieren einer Kante erhöht keinen d-Wert.

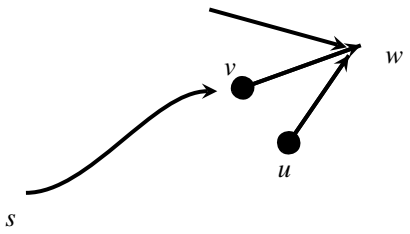
13.9 Lemma
Ist $d(u) \geq \delta(u) \forall u \in V$ vor dem relaxieren, dann auch danach.

Beweis Jeder Weg von s nach $v + (v, w)$ ist Weg von s nach w. Deshalb $\delta(v) + c(v, w) \geq \delta(w)$
$\Rightarrow d(v) + c(v, w) \leq d(w)$
$\Rightarrow d(w) \geq \delta(w)$ □

Beispiel Algorithmus: starte mit $d(v) = \begin{cases} \infty & \text{sonst} \\ 0 & \text{für } v = s \end{cases}$ relaxiere nacheinander alle Kanten. $d(u)$ wird immer kleiner, unterschreitet $\delta(u)$ nicht. Reihenfolge!



13.10 Lemma
Sei $w \in V$ und $\delta(w) < \infty$ und sei (v, w) die letzte Kante auf dem billigsten Weg von s nach w. Dann gilt: Falls (v, w) relaxiert wird, nachdem $d(v) = \delta(v)$ gefunden ist, so ist $d(w) = \delta(w)$.



$d(w) = \delta(w)$
$d(w) = \min\{d(w), d(v) + c(v, w) = \delta(v) + c(v, w) = \delta(w)$
Algorithmus:

```
d(s) ← 0,
d(v) ← ∞ ∀ v ∈ V \ { s }
 for i ← 1 to n-1 do
     for all (v,w) ∈ E do Relax(v,w) od
 od
```

13.11 Lemma
Für $i = 0, \ldots, n-1$ gilt : Nach Phase i ist:
$d(w) = \delta(w)$ für alle $w \in V$, für die es einen billigsten Pfad der Länge i von s nach w gibt.

Beweis Induktion über i.

$i = 0 : d(s) = \delta(s)$
$i \to i + 1$: Sei w Knoten mit kürzestem Weg der Länge $i + 1$.

Sei (v, w) die letzte Kante auf kürzestem Weg zu w. Also gibt es billigsten Weg von s nach v der Länge i. Nach induktionsannahme gilt:

$$d(v) = \delta(v)$$

Und in Phase $(i + 1)$ wurde (v, w) relaxiert.
$\Rightarrow d(w) = d(v) + c(v, w) = \delta(v) + c(v, w) = \delta(w)$ □

13.12 Lemma
Nach Phase $n - 1$ ist $S(v)$ für alle $v \in V$ berechnet.

Beweis Alle kürzesten Wege haben Länge $\leq n - 1$ □

Laufzeit: $O((n - 1)m) = O(n \cdot m)$

4. Situationn: 3. + negative Zyklen:

1. zuerst $n - 1$ Phasen von Bellman- Forrd, nenne diese Werte d_1 - Werte.
2. führe nochmals n Phasen aus $\rightarrow d_2$ Werte.

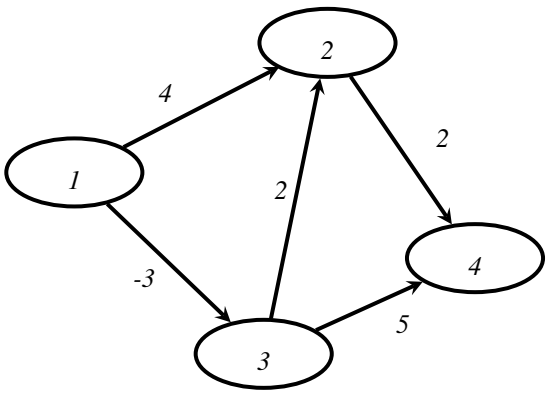
13.13 Lemma
Sei $w \in V$:

$$\text{Falls: } \begin{cases} d_2(w) = d_1(w) & \Rightarrow \delta(w) = d_1(w) \\ d_2(w) < d_1(w) \Rightarrow \delta(w) = -\infty \end{cases}$$

All pairs shortest paths (APSP)

1 $n\times$ Dijkstra / B.-F. $O(n(n \log n + m))$ / $O(n \cdot (n \cdot m))$

2 Floyd/Warshall:
Annahme keine negativen Zyklen. $V = \{1, 2, \ldots, n\}$
Definiere $\delta_(i, j)$: Kosten des billigsten Weges von i nach j mit inneren Knoten $\leq k$.

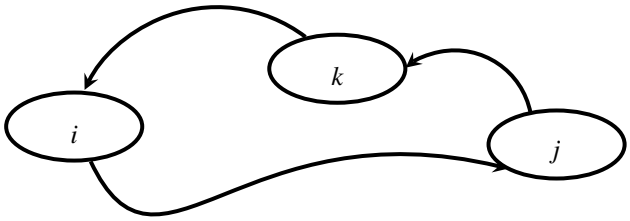


- $\delta_1(1, 4) = \infty$

- $\delta_2(1,4) = 6$
- $\delta_3(1,4) = 2$
- $\delta_4(1,4) = 2$

$$\delta_0(i,j) = \begin{cases} c(i,j) & \textit{falls } (i,j) \in E \\ 0 & \textit{fallsi} = j \\ \infty & \textit{sonst} \end{cases} \qquad \textit{Es gilt: } \delta_n(i,j) = \delta(i,j).$$

Kosten des billigsten Weges von i nach j.



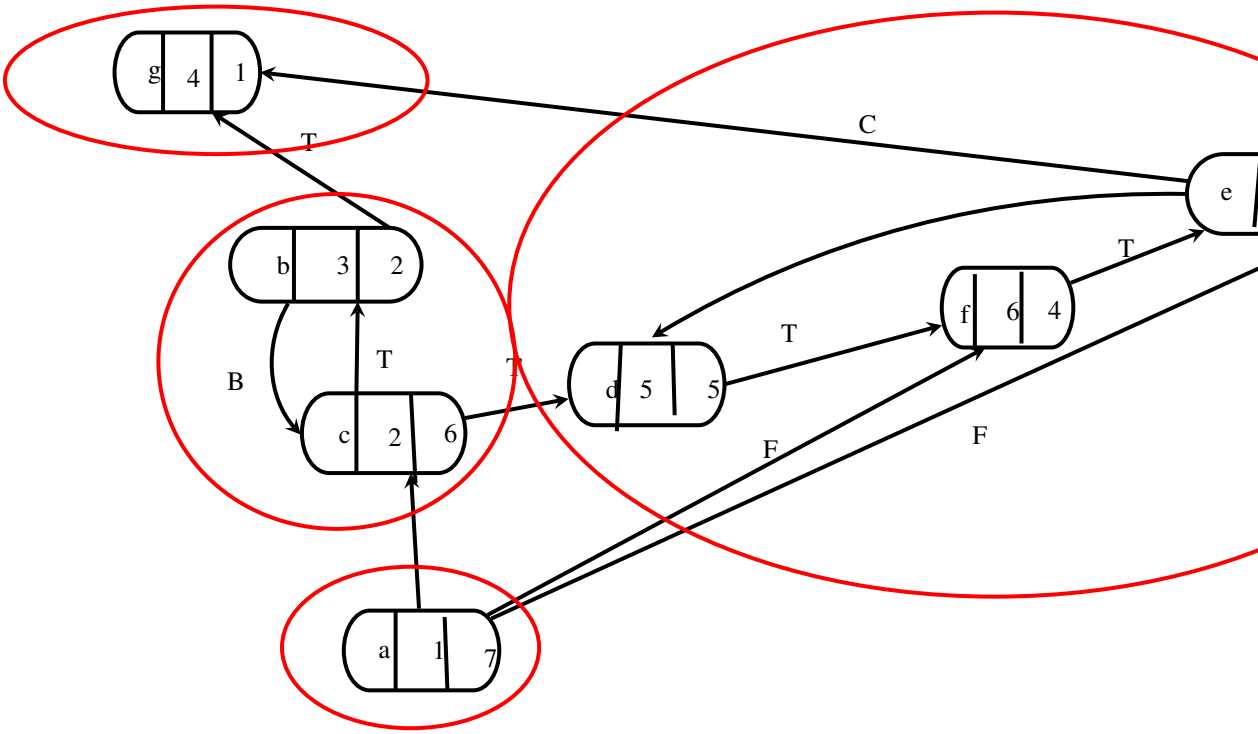
$\delta_k(i,j) = \min\{\delta_{k-1}(i,j), \delta_{k-1}(i,k) + \delta_{k-1}(k,j)\}\ \forall i,j$
Laufzeit: $O(n^3)$

Starke Zusammenhangskomp. (SZK) von gerichteten Graphen:

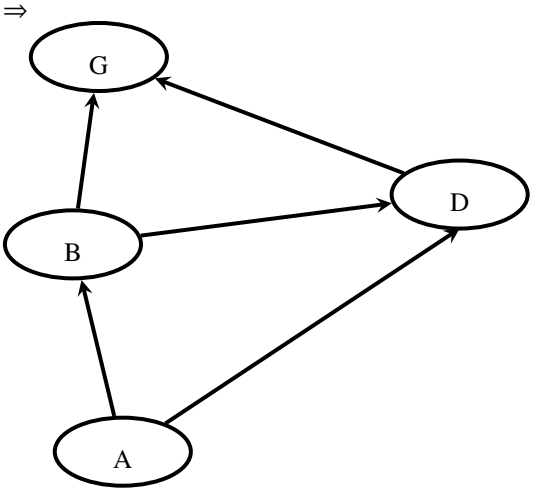
Digraph: $G = (V,E)$ heißt stark zusammenhängend $\Leftrightarrow$:

$$\forall v, w \in V : v \overset{*}{\rightarrow} w$$

Eine SZK von G ist ein maximaler stark zusammenhängender Teilgraph.



⇒



Sei (V', E') eine SZK. Knoten $v \in V'$ heißt Wurzel der Komponenten, wenn `dfssum`(v) minimal ist.

Idee:
Sei $G_{akt} = (V_{akt}, E_{akt})$ der Graph, der von den schon besuchten Knoten aufgespannt wird. Wir verwalten die SZKs von G_{akt}.

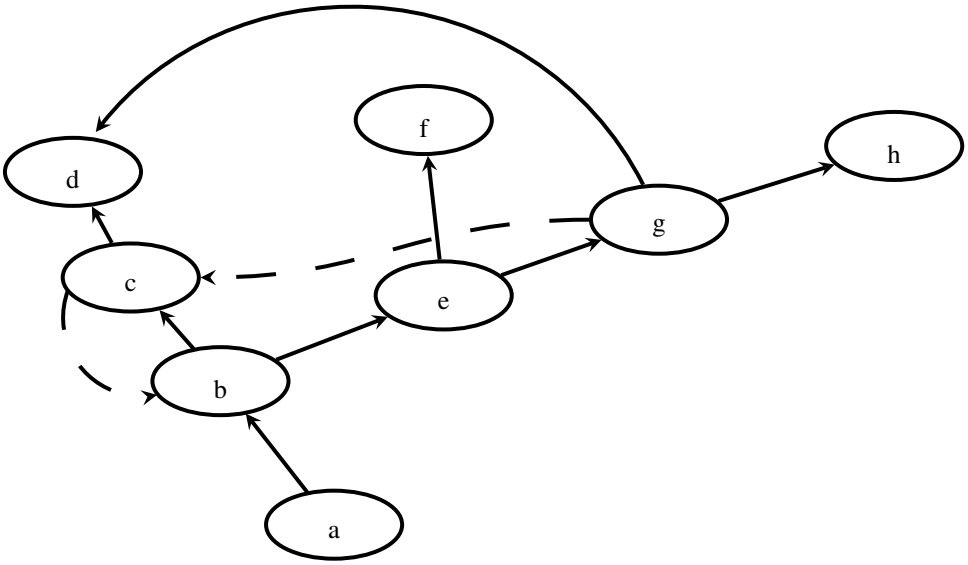
Anfang: $V_{akt} = \{1\}, E_{akt} = \emptyset$

betrachte Kante (v, w) .

1. $(v, w) \in T$: w kommt zu V_{akt} hinzu, bildet eigene SZK.
2. $(v, w) \notin T$: Mische eventuell mehrere SZKs zu einer. Abgeschlossene SZK k ist eine SZK, für die `dfssum`$(v) \forall v \in k$ abgeschlossen ist.

2 Mengen:

- Wurzeln: Folge der Wurzel von nicht abgeschlossenen Komponenente in aufsteigender Reihenfolge der `dfs`.
- unfertig: Folge den Knoten v , für die `dfssum`(v) aufgerufen wurde, aber die zugehörige SZK noch nicht abgeschlossen wurde: Aufsteigend nach `dfssum`.



- unfertig: a ~~b~~ ~~c~~ ~~d~~ ~~e~~ ~~f~~ ~~g~~ ~~h~~
- Wurzeln: a b ~~c~~ ~~e~~ ~~f~~ ~~h~~

$\Rightarrow$ d h b a

Kanten Ausgehend von g

1. (g, d) nix gemacht
2. (g, c): vereinungen 3 SZKs mit Wurzeln b e g. Dadurch sind e und g keine Wurzel mehr.
3. (g, h): h ist nur Knoten, $(g, h) \in T$.
   h ist eine SZK, füge h zu unfertig ind Wurzeln.

Pseudocode:

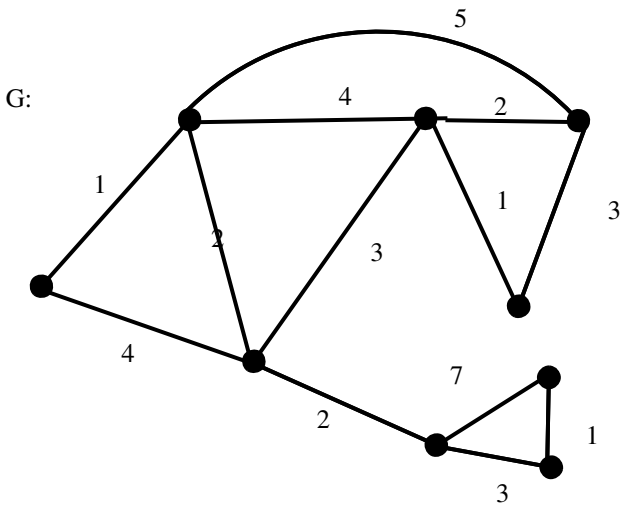
```
Anfang: unfertig <-- Wurzeln <-- leerer Keller
  for all v do inunfertig(v) <-- false od
dfs(v): push(v, unferig); inunfertig(v) <-- true
        z1++, dfsnum(v) ,__ z1; S <-- S \cap {v}
        push(v, wurzeln)
        for all (v,w) \in E do if w \notin S then dfs(w)
            else if inunfertig(w) then
                    while dfsnum(w) < dfsnum(top, Wurzel) do
                        pop(Wurzel)
                    od
        od
        z2++;  compnum(v) <-- z2
        if v = top(Wurzel) then
  repeat w <-- top(unfertig); inunfertig(w) <-- false
          until w = v
          pop(Wurzel)
        fi
```

Laufzeit: $O(n+m)$
Ähnliches geht für 2 ZK(2 fach Zusammenhangs Komponenten)
Unger. $G' = (V', E')$ heißt zweifachzusammenhängend $\Leftrightarrow$ $\forall v, w$: Es gibt 2 Knotendisjunkte Pfade von v nach w.
Auch 2 ZK's können in $O(n+m)$ mit DFS berechnet werden.

Minimale aufspannende Bäume

[MSTs]

Sei G ein zusammenhängender ungerichteter Graph. Sei $c : E \to \mathbb{R}^+$ eine Kostenfunktion.



Bestimme $E_T \subseteq E$, so daß $G_T = (V, E_T)$ zusammenhängend ist und $c(E_T) = \sum_{e \in E_T} c(e)$ minimal ist.

13.14 Lemma
E_T ist azyklisch.

Beweis Wenn E_T einen Kreis enthalte, führt Löschen einer beliebigen Kante aus dem Kreis zu einer Kostenreduktion $\Rightarrow$ `. □

13.15 Lemma
Algorithmus (Greedy) (Kruskal)

- *ordne Kanten $e_1, \ldots, e_m$ so daß $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$*
- $E_T = \emptyset$

```
for i <-- 1 to m
  do if (V, E_T∪ { e_i }) azyklisch
       then E_T <--- E_T∪ { e_i }
       fi
  od
```

13.16 Lemma (Korrektheit)
Kantenmenge $E' \subseteq E$ heißt "gut", wenn sie zu einem MST erweiterbar ist.
Behauptung:
Sei $E' \subseteq E$ gut und sei $e \in E \backslash E'$ die billigste Kante, so daß $(V, E' \cup \{e\})$ azyklisch ist. Dann ist auch $E' \cup \{e\}$ gut.

Beweis Sei $T_1 = (V, E_1)$ ein MST mit $E' \subseteq E_1$. T_1 existiert, da E' gut ist.

- Ist $e \in E_1$, fertig.
- Ist $e \notin E_1$ betrachte Graph $H = (V, E_1 \cup \{e\})$. H enthält einen Zykel, auf dem e liegt. □

Da $(V, E' \cup \{e\})$ azyklisch ist, enthält der Zykel eine Kante aus $E_1 \backslash \{E' \cup \{e\}\}$. Sei e_1 so eine Kante. Betrachten Baum $T_2 = (V, (E_1 \backslash \{e_1\}) \cup \{e\})$. T_2 ist aufspannend und $c(T_2) = c(T_1) + c(e) - c(e_1)$. e ist billigste Kante, also $c(e) \leq c(e_1) \Rightarrow e(T) \leq c(T_1)$
Da T_1 MST , ist $c(T_1) \leq c(T_2)$, damit ist T_2 auch MST, damit war $E' \cup \{e\}$ gut.

Beispiel (Implementierung) Halten partition von $V = \{1, 2, \ldots, n\}$
$V = V_1 \cup \ldots \cup V_k$ mit $V_i \cap V_j = \emptyset$
$V_i \neq \emptyset \forall i$
Operationen:

- Starten mit $\{1\}, \{2\}, \ldots, \{n\}$

- `find`(x): gibt Namen der Menge zu der x gehört
- `Union`(A, B): Vereinige Mengen A und B.

```
for  i <------- to n
  do sei e_i = (u,v); A <--- Find(u); B <--- Find (v);
    if A ≠ B then E_T <--- E_T∪ { e_i }, Union (A,B)
   fi
  od
```

Wir machen $2m \times$ `Find`
Sowie $n-1 \times$ `Union`
Realisieren Union-Find-Datenstruktur:

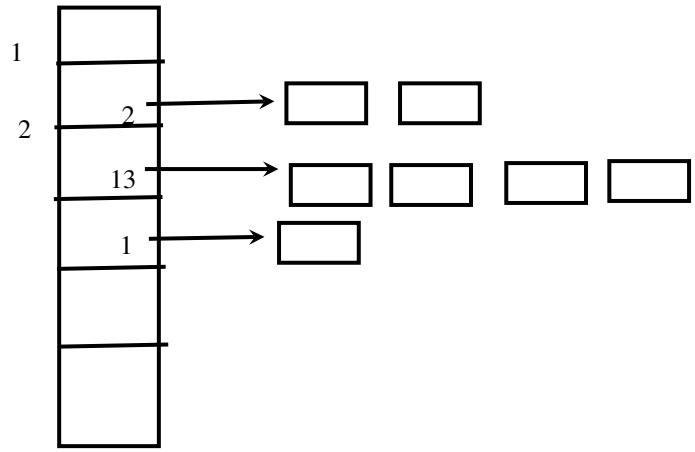
1. Feld mit Namen $R[1 \ldots n] \to [1 \ldots n]$
   $R(x)$: Name der Menge, die x enthält.

```
Find  (x) return R[x]

Union  (A,B):

  for i  ←  1 to n do

     if R[i] = A

      then R[i]  ←  B

     fi

   od
```

   Laufzeit: $O(m \log m + m + n^2)$

Beispiel (Verbesserung)

- `Union` soll nicht alle Elemente betrachten, sondern nur die Elemente die betroffen sind
- Verwalte Elemente der Menge separat
- Behalte bei `Union` immer den Namen der größeren Menge.



Initial:

```
for x ← 1 to n do R[x] ← x
 Elem[x] ← x
 size[x] ← 1
                                od
   Find (x) : return R[X]
Union  (A,B) if size(A) < size(B) then A ⇒ B
                           for all x ∈ Elem(B) do
                               R[x] ← A
      insert(x,Elem(A))
                            od
```

- worst-case: `Union` immer noch in $O(n)$.

Behauptung:
$n -$ `Unions` gehen in Zeit $O(n \log n)$.
`Union`(A,B) mit n_A, n_B Elementen und $n_b \leq n_a$. Kostet: $O(1 + n_b)$. Sei n_i beim i-ten `Union` die Größe der kleineren Menge: Zeit für alle Union $O(\sum_{i=1}^{n-1}(n_i + 1)) = O(n + \sum n_i)$

Immer wenn Element den Namen seiner Menge wechselt, trägt es "1" zu $\sum_{i=1}^{n-1} n_i$ bei.

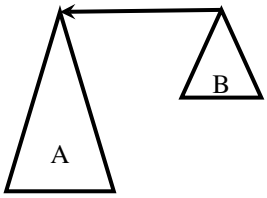
Es gilt: $\sum_{i=1}^{n-1} n_i = \sum_{j=1}^{n} r_j$, r_j gibt an, wie oft Element j seine Menge wechselt.
Behauptung:

$r_j \leq \log n$ für $j = 1, 2, \ldots, n$
Es gilt: Wechselt j die Menge einmal, so steigt die Größe seiner neuen Menge um den Faktor 2. $\Rightarrow$ Nach dem k-ten Wechsel, ist die Menge von j, mindestens 2^k groß, sie kann aber höchstens nur n Elemente umfassen, also gibt es höchstens $\leq \log n$ Wechsel.
$\sum_{i=1}^{n-1} n_i = \sum r_j \leq \sum_{j=1}^{n} \log n = O(n \log n)$
$\Rightarrow$ Laufzeit: $O(m \log m + m + n \log n)$

MST: Kruskal
Sortieren der Kanten nach Kosten. Anschließend :
Iteriere: Füge Kanten in sortierter Reihenfolge ein, wenn sie keinen Zykel schließen.
$\rightarrow$ Union-Find

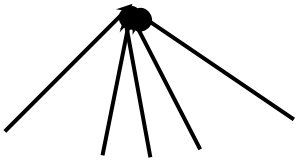
Weighted-Union-Rule: $O(n \log n + m)$

Andere Ideen zu Union-Find. Halte Mengen als Bäume: Knoten hat Zeiger auf parent,

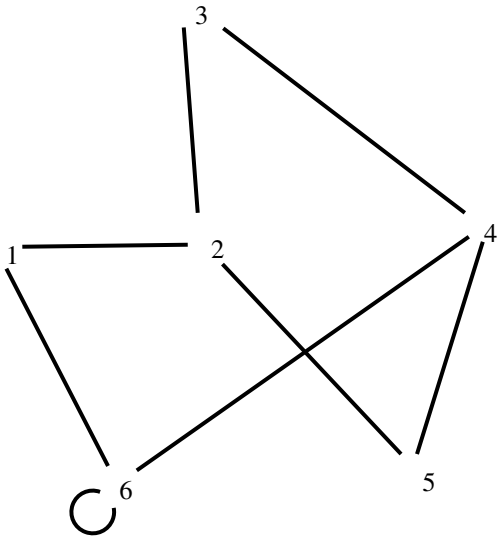


Union(A,B)
in O(1)

Wurzel jedes Baums enthält Namen der Menge.



Find ist eventuell teuer. Ideal:
Weighted Union Rule: Hänge kleineren Baum an größeren an.
Path Compression(Pfadkomprimierung):



Bei Find laufe PFad hoch zur Wurzel und hänge alle Zeiger um direkt zur Wurzel.

13.17 Satz
Mit gewichteter Vereinigungsregel und Pfadkomprimierung gehen n Unions und m Finds in Zeit von $O(n + m \cdot \alpha(m + n, n))$, wobei α invers zur Ackermannfunktion. ($\alpha \leq 5$ für alle realen Werte)

13.18 Lemma
MST (Prim)

```
T ← { v };

solange T ≠ V

   do sei e=(u,w) ∈ E mit u ∈ T, w ∉ T mit c(e) minimal

   E_T ← E_T ∪ {e }

   T ← T ∪ { w }

   od
```

Korrektheit "ähnlich" wie vorher.
Implementierung: Speichern in Warteschlange PQ:

$\{c(w) \mid w \notin T$ mit $c(w) = \min \{c(u,w) \mid u \in T\}\}$

entspricht Deletemin bei Dijkstra und benötigt $O(n)$ Operationen.

Update: wird w nach T eingefügt, so:

$\forall (w,x) \in E$ mit $x \notin T$

```
   do c(x) ← min { c(x) , c(w,x) }
   od
```

Was "Decreasekey" bei Dijkstra entspricht und in $O(m)$ Operationen erfolgt. Realisiere PQ als Heap $O(m \cdot \log n)$ Operationen.

SZK gerichteter Graph.
ZZK:

- ungerichteter Graph G ist 2fach zusammehängend, wenn $G - \{v\}$ zusammenhängend ist für alle Knoten $v \in V$.
- ZZK eines ungerichteten Grpaphen ist ein maximal 2fach zusammenhängender Teilgraph.
- Knoten $v \in V$ heißt Artikulationspunkt, wenn $G - \{v\}$ nicht zusammenhängt.

ZZK's: $\{2,3,4\}, \{4,5,6\}, \{1,2\}, \{6,7\}$
DFS auf ungerichtetem Graphen:

Im ungerichteten Fall gibt es keine Vorwärts- und Querkanten sondern nur Baum und Rückwärtskanten.

Beispiel Idee: v ist kein Artikulationspunkt, falls ein Baumnachfolger von v eine Rückwärtskante vor v hat.
afsnum ist kleiner als afsnum(v).

Beispiel

- Gegenbeispiel zur Idee: v Artikulationspunkt, da es einen Zweig aus v, von dem aus es keine Rückwärtskante "vor" v gibt,
- Ist v Wurzel des Baumes, so ist v Artikulationspunkt , falls es mehr als 1 Zweig hat.

13.19 Definition low(u) $\leftarrow \min\{afsnum(u), \min\{afsnum(v) \text{ mit } \exists u \to Z \to v\}\}$

```
for all (v,w) ∈ E

 do if v unbesucht

  then dfsnum(w) ← z1, low(w) ← dfsnum (w) , z1++;

    dfs(w);

(*)   if low(w) < low(v) then low (v) ← low(w)

(**)  if low(w) ≥ dfsnum(v) then Artikulationspunkt(v) ← true;
(***) else if dfsnum(w) < low(v) then low(v) ← dfsnum(w)
 od
```

- In (*) wird low Wert von w an V übergeben, falls er kleiner ist.
- In (**) wird der Fall "Rückwärtskante" behandelt
- In (***) wird erkannt, ob der Zweig, der mit (v,w) startet, keine Rückwärtskante enthält, die "vor" v endet.

Spezialfall: v ist Wurzel (dfsnum 1):

```
if (dfsnum(v) = 1 und ∃ w_1 ≠ w_2 mit (v, w_1 ), (v, w_2 ) ∈ T

   then Artikulationspkt. (v) ← true;
```

$\Rightarrow$ Bestimmung der Artikulationspunkte in $O(n + m)$ (da Verfeinerung von dfs)
Übung: Bestimme ZZK für gegebene Artikulationspunkte.

§14 Patternmatching im Strings

T String über Alphabet Σ
$T[i, j]$ Teilstring von stelle i bis Stelle j

```
Gegeben: String T[1,n], Pattern P{1...m]
         Suche P im T
         Naiv: FOR s <-- 1 TO n-m-1 DO
                  Test ob T[s,...,s+m-1] = P
               OD
               Laufzeit: O(n * m)
         1.Algorithmus: Knuth/Moris/Pratt (KMP)
            Idee: Verschiebe P nicht nur um 1, sondern um
                  T: babaabaa        Haben P[1...q] = T[s+1...s+q], g<m
                  P:  abab           Suche das kleinste s'>, so dass
                      s              P[1...k] = T[s'+1...s'+k] mit s'+k = s+q
                                     Im Beispiel: Verschiebe um 2.
                                     Vergleiche dazu P mit sich selbst
                  Prafixfunktion pi: {1,...,n} --> {0,...,m-1}definiert durch:
                                     pi(q) = max{k|P[1...q]=P:P[1...k] fur ein Prafix
                                     p' \in Sigma^(q-k)}
                                     also P[1...k] Suffix von P[1...q]
            Beispiel: P[1...q] = abab fur q=4
                      P ababaa
                  Index 123456
                  pi(i) 001231
  Prefixfunktion(P) pi(1) = 0; k <-- 0
                  FOR q=2 TO m DO
                      WHILE k>0 AND P[k+1] \neq P[q] DO
                            k <-- pi(k)
                      OD
                      IF P[k+1] = P[q] THEN
                         k <-- k+1
                      FI
                      pi(q) <--k
                  OD
```

Laufzeit: while-Schleife nur, wenn $k > 0$. In der while-Schleife wird k ernidrigt, da $\pi(k) < k$.
1. Durchlauf erhöht (das FOR-Schleife) k maximal um 1. d.h. es kann höhstens so viele Ernidrigungeb geben, wie Erhöhungen.
$\Rightarrow O(m)$

```
Algorithmus (KMP) Sei T,P,n,m,pi gegeben.
  q <-- 0
  FOR i <-- 1 TO n DO
      WHILE q>0 AND P[q+1] \neq P[i] DO
            q <-- pi[q]
      OD
      IF P[q+1]=T[i] THEN
         q <-- q+1
      FI
      IF q = m THEN
```

```
        'P kommt am Stelle i-m+1 vor'
        q <-- pi(q)
    FI
OD
```

Laufzeit: Analogs Algorithmusargument wie bei Präfixfunktion(P)
while-Schleife ernidrigt das q
FOR-Schleife erhöht q maximal um 1.
$\Rightarrow O(n)$

```
2. Algorithmus von Bayer.Moore (BM)
  Naiv: Gegeben n,m,T,P
        lambda <-- letztes Vorkommen(P,m,Sigma)
        gamma <-- gutes Suffix (P,m)
        s <-- 0
        WHILE s <= n-m DO
              j <-- m
              WHILE j > 0 AND P[j]=T[s+j] DO
                    j <-- j-1
              OD
              IF j = 0 THEN
                 'P kommt ab Stelle s+1 vor'
                 s <-- s+gamma(0)
              ELSE
                 s <-- s+max{gamma[j], j-gamma[T[s+j]]}
        OD
```

Wesentlich: -Durchlaufe Pattern von rechts nach links
-Heurictiken 'schlechter Buchstabe'
'guter Suffix'

'schlechter Buchstabe'

```
T: a b c a b c e a b c b a
P:       a b c a b c
               j = 4
```

Sei $P[j \neq T[s+j]$ ein gefundes Mismatch.
Dann sei $k = \begin{cases} \max i | T[s+j] = P[i] \text{ falls } T[s+j] \in P \\ 0 \end{cases}$

14.1 Lemma
s kann um $j-k$ verschoben werden

Beweis 1) $k = 0 : T[s+j]$ kommt in P nicht vor $\rightarrow$ schlechter Buchstabe. Verschiebe um j, bis hinter diesen Buchstaben.

2) $k < j : T[s+j]$ kommt in P links von der Stelle j als rechtestes Vorkommen vor, dann $j-k > 0$. Verschiebe um $j-k$

Beispiel

```
T: abcabceabcba
P:  eaebcabc
       eaebcabc  => Verschiebe um 3.
    k=3  j=6
```

3) $k > j$: bedeutet Shift von P nach links, denn $j - k < 0$. Aber das wird verhindert durch 'gutes Suffix'. □

Brauchen für jedes Zeichen sein rechtestes Vorkommen in P, also für $a \in \Sigma$ berechne $\lambda[a]$ mit $a = P[\lambda[a]]$
Berechne λ:

```
FOR ALL a \in Sigma DO
        lambda[a] <-- 0
OD
FOR j <-- 1 TO m DO
    lambda[P[j]] <-- j
OD
```

Laufzeit: $O(|\Sigma| + m)$
d.h. es kann $\leq$ so viele Erniedrigungen geben, wie Erhöhungen.
$\Rightarrow$ $O(m)$ Laufzeit.

Guter Suffix: Q Suffix zu R oder R Suffix von Q: $Q \sim R$

```
Idee: Falls P[j] \neq T[s+j], verschiebe P um
      gamma[j] <-- m - max{k|P[j+1...m] ~ P[1...k]}
      1. gamma[j] < m - pi[m]
      2. Ist P[1...k] Suffix von P[j+1...m], dann auch Suffix von P[1...m]
 T: a b r a l a l a b r a
 P: <-> a l b a l
    s=2 <---> \ /
         j=3   guttes Suffix
```

$\gamma[j] = 5 - 2 = 3$ Also verschiebe P um 3
$\Rightarrow$ $O((n - m) \cdot m)$, aber sehr praktikabel.

Patternmatching $S = S_1, \ldots, S_n$
$P = Y_1, \ldots, y_m$
naiv: $O(m \cdot (n - m))$
Karp/Rabin:

Beispiel (Idee) Fingerabdruckfunktion
Sei $\Sigma = \{0, \ldots, 9\}$ $S, P \in \mathbb{N}$

$$F_p : \mathbb{Z} \to \mathbb{Z}_p \text{ mit } F_p(z) = z \mod p$$

wobei p eine Primzahl ist.

Beispiel

s	7	5	3	2	8	9
p		3	7	5		

Algorithmus: Sei p zufällige Primzahl zwischen 2 und $mn^2 \log(mn^2)$.

```
Match ← false; i ← 1;
 while not Match and 1 ≤ i ≤ n − m + 1
   if F_p (S_i ... S_i + m − 1 = F_p(P)
    then Match ← true;
   else i ← i + 1
        berechne F_p (S_i + 1 ... S_i + m
         i ← i + 1
   fi
od
```

Berechne nacheinander $F_p(s_i \dots s_{i+m-1}) \forall i$:

Beispiel 2 3 7 5 8 6 1 3 2 5
$m = 4$:

$$F_p(S_{i+1} \dots S_{i+1}) = \left(\underbrace{10}_{\text{Shift nach links}} \cdot \left(F_p(S_i \dots S_{i+m-1}) - 10^{m-1} \cdot s_i \right) + \underbrace{s_{i+m}}_{\text{rechteste Stelle}} \mod p \right)$$

Geht in konstanter Zeit, falls $10^{m-1} \mod p$ vorberechnet.

14.2 Satz
Algorithmus von Karp/Rabin läuft in $O(n+m)$ und hat Fehlerwahrscheinlichkeit von $O(\frac{1}{m})$.

Beweis Laufzeit klar. Sei $a = P$, $b = s_i \dots s_{i+m-1}$.
Wir berechnen $a \mod p \equiv b \mod p \Leftrightarrow p$ teilt $|a - b|$
Es gilt:
$|a - b| \leq 10^m < 2^{4m}$
$\Rightarrow |a - b|$ hat $\leq 4m$ verschiedene Primfaktoren.
Wir haben p aus dem Bereich $[2, mn^2 \log(mn^2)]$ gewählt. In diesem Bereich gibt es höchstens $O\left(\frac{m \cdot n^2 \log(mn^2)}{\log mn^2 \cancel{\log mn^2}}\right) = O(mn^2)$ viele Primzahlen.
Für festes i gilt also:

$$pro(F_p(P)) = F_p(s_i \dots s_{i+m-1}) \, |P \neq s_i \dots s_{i+m-1}| = O(\frac{4m}{mn^2}) = \frac{1}{m^2}$$

Gesamfehlerwahrscheinlichkeit:

$$O(\underbrace{(n-m+1)}_{\#\text{ Versuche}} \cdot \underbrace{\frac{1}{n^2}}_{\text{Einzel Fehlerwahrscheinlichkeit}}) = O(\tfrac{1}{n})$$

Das bedeutet: Wir haben Algorithmus, der in Linearzeit läuft, aber mit kleiner Wahrscheinlichkeit ein falsches Ergebnis liefert. (Monte-Carlo-Algorithmus)
Las Vegas Algorithmus : Immer richtiges Ergebnis, eventuell aber längere Laufzeit.

□

14.3 Definition (Las-Vegas-Algorithmus) 1. 1. Falls $F_p(s_i \dots s_{i+m-1} = F_p(P)$ feste in $O(m)$ da $s_i \dots s_{i+m-1} = P$
Falls nicht, starte naiven Algorithmus.

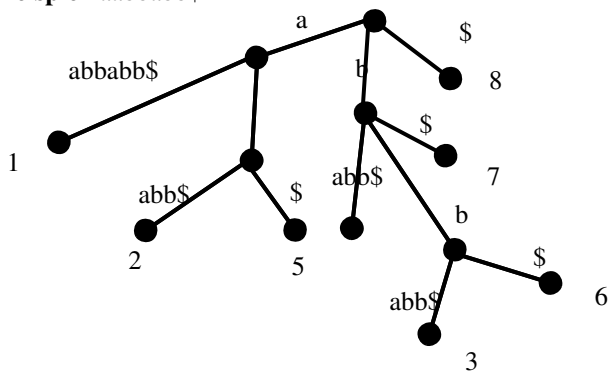
Laufzeit:
$O((n+m)(1-\frac{1}{n}) + (n \cdot m)(\frac{1}{n}) = O(n+m)$

Suffixbäume

Baum T mit n Blättern mit Nummern 1 bis n.

- Jeder innere Knoten hat ≤ 2 Kinder
- Jede Kante ist mit Teilstring aus S markiert.
- Kanten au demselben Knoten tragen Markierung mit verschidenen ersten Zeichen
- Vom Wurzel zum Blatt i ergeben die Kantenmarkierungen zusammen genau Suffix $S[i \dots n]$

Beispiel aabbabb$



Geg. Pattern z.B. *abb*, kommt bei 2 und 5 vor.
Starte an Wurzel. Laufe *abb* ab. Stoppe.
Lies alle Indizes, die unterhalb dem Stopknoten stehen.
Laufzeit: $O(m+k)$, fass es k vorkommen von P gibt. Algorithmus:
Starte mit 1 Kante für Suffix $S[1 \dots n]$

- Iterativ sei T_i oder Baum der Suffixe $S[1 \dots n], \dots S[i \dots n]$
  Starte im Wurzel, suche längsten Pfad für Suffix $S[i+1 \dots n]$ im T_i, der mit enem Präfix von $S[i+1 \dots n]$ übereinstimmt.

1. Pfad endet auf einer Kante. Dort endet Übereinstimmung.
   Teil (u, v) in (u, w) und (w, v) und teile Markierungen entsprechend, so dass Übereinstimmung in w endet.
   Füge neue Kante $(w, Blatt_{i+1})$ mit Markierung rest. Suffix von $S[i+1 \ldots n]$
2. Pfad endet an einem Knoten w.
   Füge neue Kante $(w, Blatt_{i+1})$ ein mit Markierung rest. Suffix von $S[i+1 \ldots n]$
   Laufzeit: $\sum_{i=n}^{i} i = O(n^2)$

Konstruktion geht auch in $O(n)$ (Ukkonen)
Suffixbaum ist geeignet, wenn auf Sequenz nach verschiedenen Mustern gesucht wird.
Verallgemeinerung mehrere Strings $S_1, \ldots, S_r$ endet mit $\$_1, \ldots \$_r$.
Konkateniere sie alle und baue großen Suffixbaum.
Beachte: Es entstehen künstliche Suffixe über Stringgrenzen hinweg
Also: Endzeichen $\$_i$ müssen in Blättern stehen
Problem: Finde größten gemeinsamen Teilstring on S_1 und S_2

1. Konstruire verallgem. Suffixbaum für S_1 und S_2
2. Innere Knoten werden mit $1/2$ markiert, wenn sie zu Blatt mit Suffix aus S_1/S_2 führen
3. Pfadbeschriftungen zu doppelt markierten Knoten entsprechen gemeinsamen Teilstrings
4. Tiefensuche von Wurzel aus über doppelt markierte Knoten liefert besten Teilstring. Laufzeit $O(n)$

Index