Inhaltsverzeichnis

1 Grundlegende Datenstrukturen

1.1 Datenstrukturen für Graphen

1.1.1 Allgemeines

- Adjazenz:
  Zwei Knoten heißen adjazent, wenn zwischen ihnen eine Kante verläuft.

- Inzidenz:
  Ein Knoten und eine Kante heißen inzident, wenn der Knoten auf der Kante liegt, also Anfangs- oder Endknoten ist.

- Grad $d(v)$ bei ungerichteten Graphen:
  Anzahl der Kanten, mit denen der Knoten v inzident ist.

- Ingrad $\text{ind}(v)$ bei gerichteten Graphen:
  Anzahl der Kanten $(\cdot, v)$, somit Kanten, die am Knoten v enden

- Outgrad $\text{outd}(v)$ bei gerichteten Graphen:
  Anzahl der Kanten $(v, \cdot)$, somit Kanten, die am Knoten v beginnen

1.1.2 Speicherung von Graphen

1.1.2.1 Inzidenzmatrix Es wird eine $n \times m$ Matrix benötigt. Die Einträge können für ungerichtete Graphen die Werte 0 und 1 annehmen, für gerichtete Graphen 0, 1 und 2.

- $(i, j) = 0$, wenn die j-te Kante nicht an Knoten i startet bzw. endet

- $(i, j) = 1$, wenn

  - der i-te Knoten auf der j-ten Kante liegt (bei ungerichteten Graphen)
  - die j-te Kante an Knoten i startet (bei gerichteten Graphen)

- $(i, j) = 2$, wenn die j-te Kante an Knoten i endet

Diese Datenstruktur erfordert recht großen Speicherplatz und wird nur in Spezialfällen benutzt.

1.1.2.2 Adjazenzmatrix Es wird eine $n \times n$ Matrix benötigt. Die Einträge können die Werte 0 und 1 annehmen.

- $(i, j) = 0$, wenn zwischen den Knoten i und j keine Kante existiert

- $(i, j) = 1$, wenn

  - zwischen den Knoten i und j eine Kante existiert (bei ungerichteten Graphen)
  - eine Kante von Knoten i nach Knoten j existiert (bei gerichteten Graphen)

Bei ungerichteten Graphen ist die Matrix symmetrisch, es reicht die Betrachtung der Elemente unterhalb der Hauptdiagonalen.

1.1.2.3 Adjazenzliste Die Knoten werden in einem Array der Länge n abgespeichert. Für jeden Knoten besteht ein Zeiger auf eine Liste, die alle direkten Nachfolger des Knotens enthält.

1.1.3 Tiefensuche auf ungerichteten Graphen

Sei ein ungerichteter Graph $G = (V, E)$ mit n Koten und m Kanten gegeben.

Algorithmus:

1. Initialisierung: $i = 0$, $T = \emptyset$, $B = \emptyset$, für $x \in V$: $\text{num}(x) = 0$
2. Für $x \in V$: IF $\text{num}(x) = 0$ THEN DFS $(x, 0)$[1]

DFS(v, u)**:**

1. $i = i + 1$
2. $\text{num}(v) = i$
3. Für $w \in \text{Adj}(v)$:
   IF $\text{num}(w) = 0$ THEN $T = T \cup \{(v, w)\}$, DFS(w, v)
   ELSE IF $\text{num}(w) < \text{num}(v)$ AND $w \neq v$ THEN $B = B \cup \{(v, w)\}$

Die Laufzeit beträgt $O(n + m)$. Für jeden Knoten wird genau einmal DFS aufgerufen. Weiterhin werden alle Adjazenzlisten einmal durchlaufen, also $2 \cdot m$ Kanten betrachtet.

1.1.4 Tiefensuche auf gerichteten Graphen

Der Algorithmus ist ähnlich, wobei jede Kante nur einmal betrachtet wird.
Es wird ein Hilfsparameter $\alpha(v)$ definiert, der mit 0 initialisiert wird und folgende Bedeutung hat:

- $\alpha(v) = 0$: DFS wurde für den Knoten v noch nicht aufgerufen
- $\alpha(v) = 1$: Der DFS-Aufruf für v ist erfolgt, läuft aber noch.
- $\alpha(v) = 2$: Der DFS-Aufruf für v ist erfolgt und bereits abgearbeitet.

Weiterhin wird die Kanteneinteilung verfeinert, es werde dabei die Kante (v, w) betrachtet:

- Tree-Kante T
  Bedinung: $\text{num}(w) = 0$.
  Dann wird (v, w) T-Kante.
- Forward-Kante F
  Bedingung: $\text{num}(w) \neq 0$, $\text{num}(w) > \text{num}(v)$
  Es gibt einen T-Weg von v zu w. Die F-Kante ist eine "Abkürzung".
- Back-Kante B
  Bedingung: $\text{num}(w) \neq 0$, $\text{num}(w) < \text{num}(v)$, $\alpha(w) = 1$
  Es gibt einen T-Weg von w zu v. Die B-Kante ist im T-Baum rückwärts gerichtet.
- Cross-Kante C
  Bedingung: $\text{num}(w) \neq 0$, $\text{num}(w) < \text{num}(v)$, $\alpha(w) = 2$
  Es gibt keinen Weg von w nach v. Die Kante (v, w) kreuzt entweder von einem Baum zu einem früher konstruierten Baum oder innerhalb eines Baumes zu einem früher konstruierten Teilbaum.

1.2 Datenstrukturen für Intervalle

Für eine Grundmenge $\{1, \cdots, n\}$ sei mit jeder Zahl i der Wert $w(i) \in \mathbb{R}$ assoziiert. Zu jeder Menge $A \subseteq \{1, \cdots, n\}$ gehört der Wert $w(A)$, die Summer aller $w(i)$, $i \in A$.

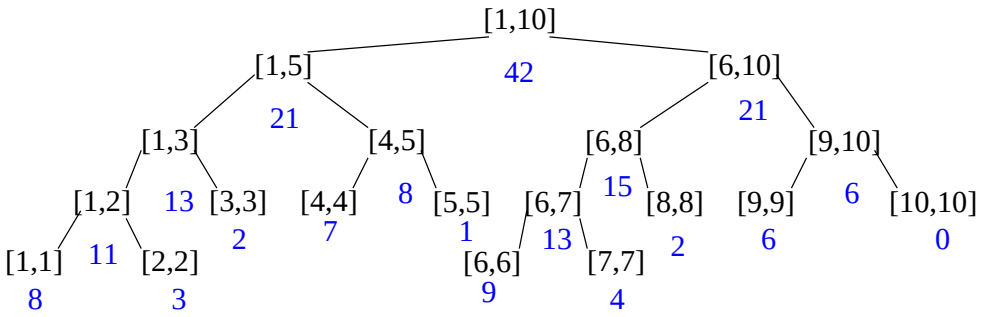
Es soll eine Datenstruktur entworfen werden, die für alle Intervalle $[i, j] = \{i, \cdots, j\}$ die Ermittlung von $w(i, j) = w([i, j])$ ermöglicht.

Dies führt zu sogenannten *segment trees* (Intervallbäumen):

[1] DFS (x, v): Tiefensuche von x aus, wobei x von v aus erreicht wurde
DFS $(x, 0)$: Falls x der erste Knoten einer Zusammenhangskomponente ist, wird $v = 0$ gesetzt.

- Jeder Knoten steht für ein bestimmtes Intervall, an der Wurzel steht das Intervall $[1, n]$.
- In dem Knoten $[l, r]$ steht $w(l, r)$
- Jeder innere Knoten $[l, r]$ hat zwei Kinder $\left[l, \left\lfloor \frac{l+r}{2} \right\rfloor\right]$ bzw. $\left[\left\lfloor \frac{l+r}{2} \right\rfloor + 1, r\right]$
- Ein Knoten $[l, r]$ wird Blatt, wenn $l = r$ ist.
- Die Tiefe beträgt $\lceil \log_2 n \rceil$, wie bei den binären Bäumen.

Beispiel für einen Intervallbaum für $n = 10$:



Berechnung von $w(i, j)$:

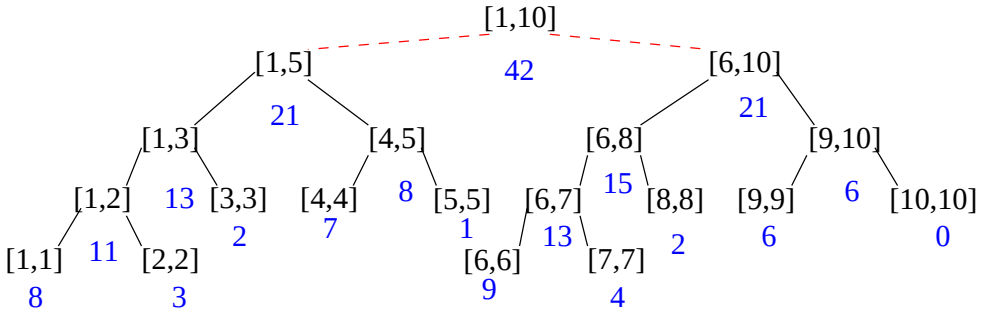
Wenn man im Baum auf den Knoten $[l, r]$ trifft, gibt es vier Möglichkeiten fortzufahren:

1. $i \leq l \leq r \leq j$
   Das Intervall $[l, r]$ des Knotens liegt vollständig in $[i, j]$. Die Information über $[l, r]$ kann genutzt. STOP.
2. $j \leq \left\lfloor \frac{l+r}{2} \right\rfloor$
   Die rechte Grenze j des Intervalls $[i, j]$ liegt "unterhalb" der linken Grenze $\left\lfloor \frac{l+r}{2} \right\rfloor + 1$ des rechten Teilbaums. Es genügt deshalb im linken Teilbaum weiter zu suchen.
3. $i > \left\lfloor \frac{l+r}{2} \right\rfloor$
   Die linke Grenze i des Intervalls $[i, j]$ liegt über der rechten Grenze $\left\lfloor \frac{l+r}{2} \right\rfloor$ des linken Teilbaums. Es genügt deshalb im rechten Teilbaum weiter zu suchen.
4. $i \leq \left\lfloor \frac{l+r}{2} \right\rfloor$, $j > \left\lfloor \frac{l+r}{2} \right\rfloor$ und *nicht* Fall 1.
   Es ist ein Gabelungspunkt erreicht, man muss in beiden Teilbäumen weitersuchen.

Beispiel Gesucht wird im obigen Intervallbaum der Wert $w(2, 8)$, also $i = 2$, $j = 8$:

1. Schritt: $l = 1$ und $r = 10$. Man stellt fest, dass
   - $i = 2 \leq 5 = \left\lfloor \frac{1+10}{2} \right\rfloor = \left\lfloor \frac{l+r}{2} \right\rfloor$ und
   - $j = 8 > 5 = \left\lfloor \frac{1+10}{2} \right\rfloor = \left\lfloor \frac{l+r}{2} \right\rfloor$

   gilt. Es wird also in beiden Teilbäumen weitergesucht:



2. Schritt:

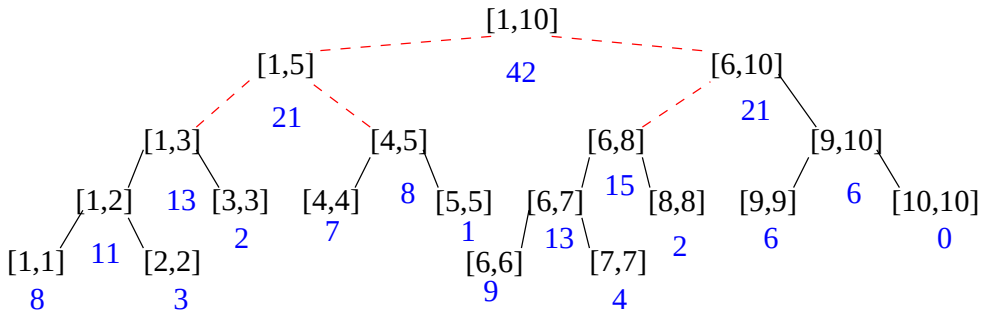
   (a) $l = 1$ und $r = 5$. Man stellt fest, dass
   - $i = 2 \leq 3 = \lfloor \frac{1+5}{2} \rfloor = \lfloor \frac{l+r}{2} \rfloor$ und
   - $j = 8 > 3 = \lfloor \frac{1+5}{2} \rfloor = \lfloor \frac{l+r}{2} \rfloor$

   gilt. Es wird also in beiden Teilbäumen weitergesucht.

   (b) $l = 6$ und $r = 10$. Man stellt fest, dass

$$j = 8 \leq 8 = \left\lfloor \frac{6+10}{2} \right\rfloor = \left\lfloor \frac{l+r}{2} \right\rfloor$$

   gilt. Es reicht, wenn man im linken Teilbaum weitersucht.



3. Schritt:

   (a) $l = 1$ und $r = 3$. Man stellt fest, dass
   - $i = 2 \leq 2 = \lfloor \frac{1+3}{2} \rfloor = \lfloor \frac{l+r}{2} \rfloor$ und
   - $j = 8 > 2 = \lfloor \frac{1+3}{2} \rfloor = \lfloor \frac{l+r}{2} \rfloor$

   gilt. Es wird also in beiden Teilbäumen weitergesucht.

   (b) $l = 4$ und $r = 5$. Man stellt fest, dass

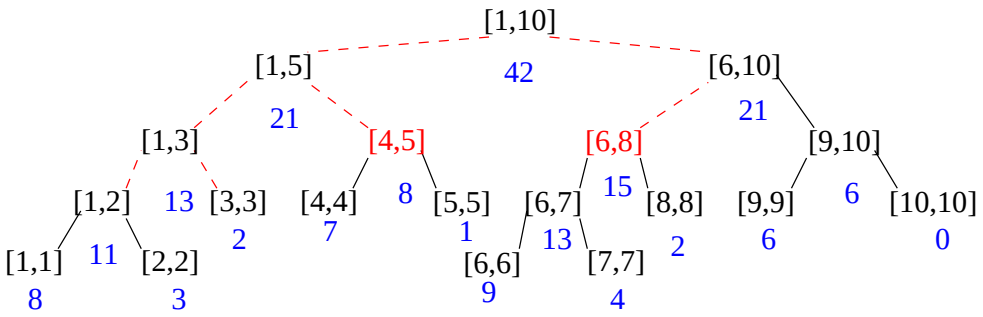
$$i \leq l \leq r \leq j \quad \text{bzw.} \quad 2 \leq 4 \leq 5 \leq 8$$

   gilt. Man kann diese Information verwenden. STOP für diesen Teilbaum.

   (c) $l = 6$ und $r = 8$. Man stellt fest, dass

$$i \leq l \leq r \leq j \quad \text{bzw.} \quad 2 \leq 6 \leq 8 \leq 8$$

   gilt. Man kann diese Information verwenden. STOP für diesen Teilbaum.



4. Schritt:

(a) $l = 1$ und $r = 2$. Man stellt fest, dass

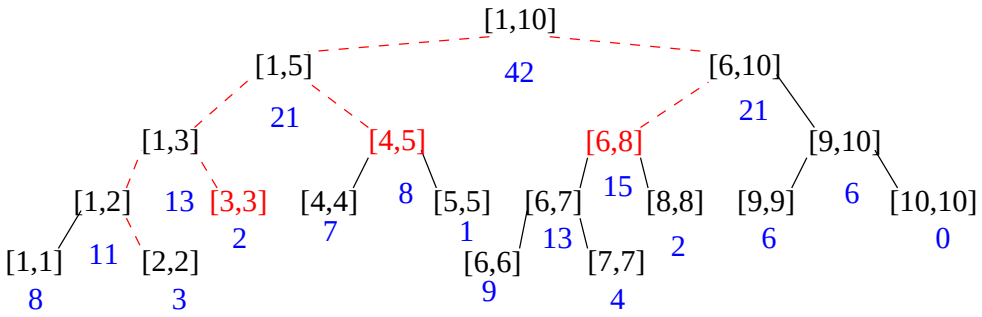
$$i = 2 > 2 = \left\lfloor \frac{1+2}{2} \right\rfloor = \left\lfloor \frac{l+r}{2} \right\rfloor$$

gilt. Es wird im rechten Teilbaum weitergesucht.

(b) $l = 3$ und $r = 3$. Man stellt fest, dass

$$i \leq l \leq r \leq j \quad \text{bzw.} \quad 2 \leq 3 \leq 3 \leq 8$$

gilt. Man kann diese Information verwenden. STOP für diesen Teilbaum.

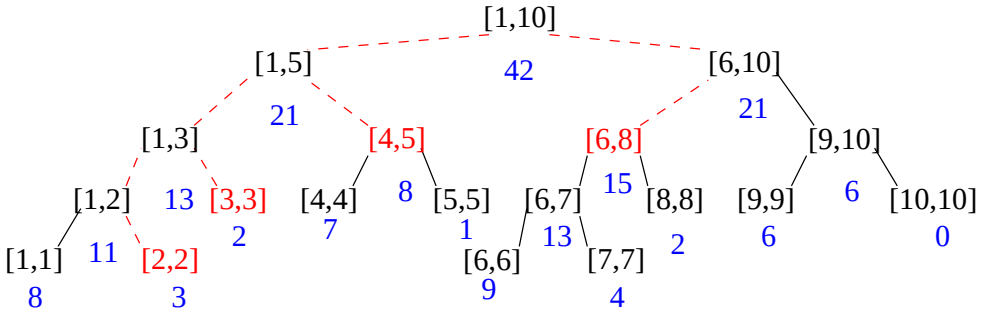


5. Schritt:

$l = 2$ und $r = 2$. Man stellt fest, dass

$$i \leq l \leq r \leq j \quad \text{bzw.} \quad 2 \leq 2 \leq 2 \leq 8$$

gilt. Es wird im rechten Teilbaum weitergesucht.



Nun hat man alle Informationen gesammelt, um $w(2,8)$ zu berechnen.

$$\begin{aligned} w(2,8) &= w(2,2) + w(3,3) + w(4,5) + w(6,8) \\ &= 3 + 2 + 8 + 15 \\ &= 28 \end{aligned}$$

Laufzeit:

Die Laufzeit und die Anzahl der betrachteten Knoten hängt von den Gabelungspunkten ab.

- Gibt es keinen Gabelungspunkt, dann durchlaufen wir höchstens $\lceil \log_2 n \rceil$ Knoten.
- Oder es gibt einen eindeutigen Weg bis zum ersten Gabelungspunkt $[l, r]$.
  Da $[l, r]$ Gabelungspunkt ist, folgt
  - $j > \lfloor \frac{l+r}{2} \rfloor$
  - im linken Teilbaum von $[l, r]$ erreicht man $[l, \lfloor \frac{l+r}{2} \rfloor]$

– Unser Suchintervall $[i,j]$ ragt rechts über den linken Teilbaum $\left[l, \left\lfloor \frac{l+r}{2} \right\rfloor\right]$ heraus

Nehmen wir nun an, dass wir in diesem linken Teilbaum $\left[l, \left\lfloor \frac{l+r}{2} \right\rfloor\right]$ an einem Knoten $[l',r']$ sind. Mit Sicherheit kann man sagen, dass $r' < j$ gilt, da r' maximal den Wert $\left\lfloor \frac{l+r}{2} \right\rfloor$ annehmen kann und nach obigen Bemerkungen $j > \left\lfloor \frac{l+r}{2} \right\rfloor$ gilt.
Nehmen wir weiterhin an, dass $[l',r']$ auch Gabelungspunkt ist, damit muss $i \leq \left\lfloor \frac{l'+r'}{2} \right\rfloor$ gelten. Dies bedeutet

$$i \leq \left\lfloor \frac{l'+r'}{2} \right\rfloor \leq \left\lfloor \frac{l'+r'}{2} \right\rfloor + 1 \leq r' < j$$

, womit der rechte Teilbaum $\left[\left\lfloor \frac{l'+r'}{2} \right\rfloor + 1, r'\right]$ komplett von $[i,j]$ eingeschlossen ist. *Für den rechten Teilbaum gilt also Fall 1,* $[l',r']$ *ist kein echter Gabelungspunkt.*

Insgesamt werden weniger als $4 \cdot \lceil \log_2 n \rceil$ Knoten besucht, von denen weniger also $2 \cdot \lceil \log_2 n \rceil$ zur Lösung beitragen.

2 Sortieralgorithmen

2.1 Eigenschaften von Sortieralgorithmen

- Stabil:
  Ein Sortieralgorithmus heißt stabil, wenn er gleiche Elemente in ihrer Reihenfolge beläßt.
- Adaptiv:
  Adaptive Sortieralgorithmen versuchen, auf vorsortierten Teilfolgen besonders schnell zu seien.
- In Situ:
  Ein Sortieralgorithmus arbeitet in situ[2], wenn er außer dem Eingabearray, das auch Ausgabearray ist, nur $O(\log_2 n)$ Speicherplatz verbraucht.
- Intern:
  Alle Daten sind stets im Hauptspeicher verfügbar.
- Extern:
  Die Daten liegen auf externen Speichermedien, z.B. Bändern, und müssen eingelesen werden, wobei nur wenige Daten im Hauptspeicher sein können.

2.2 InsertionSort

2.2.1 Funktionsweise

Seien die ersten i Daten vorsortiert, d.h. $a_1 \leq a_2 \leq \cdots \leq a_i$. Für das Datum a_{i+1} gibt es $i+1$ mögliche Positionen. Suche mittels binärer Suche die richtige Position. Vergleiche a_{i+1} zunächst mit $a_{\lceil \frac{i+1}{2} \rceil}$, dann genügen $\lceil \log_2 (i+1) \rceil$ um die richtige Position k zu bestimmen. Die Daten $a_k, \cdots, a_n$ rücken eine Position weiter nach rechts und a_{i+1} wird an Position k eingefügt.

2.2.2 Eigenschaften

- arbeitet in situ
- leicht als stabiler Algorithmus implementierbar
- internes Sortierverfahren

2.2.3 Beispiel

Gegeben sei folgendes Array

15	47	33	87	98	17	53	76	83	2	53	27	44

Vorbereitung: Unterteile das Array in einen sortierten und einen unsortierten Bereich. Der sortierte Bereich besteht zuerst aus $a(1)$, der unsortierte aus $a(2) \cdots a(n)$.

15	47	33	87	98	17	53	76	83	2	53	27	44

1. Schritt: Füge durch binäre Suche das erste Element des unsortierten Bereichs $a(2)$ in den sortierten Bereich $a(1)$ ein. Dazu rücke alle Elemente des sortierten Bereichs von der Einfügestelle $i = 2$ an um eins nach rechts und schreibe das zuvor gesicherte Element $a(2)$ an die Stelle $i = 2$.

15	**47**	33	87	98	17	53	76	83	2	53	27	44

2. Schritt: Füge durch binäre Suche das erste Element des unsortierten Bereichs $a(3)$ in den sortierten Bereich $a(1) \cdots a(2)$ ein. Dazu rücke alle Elemente des sortierten Bereichs von der Einfügestelle $i = 2$ an um eins nach rechts und schreibe das zuvor gesicherte Element $a(3)$ an die Stelle $i = 2$.

15	**33**	**47**	87	98	17	53	76	83	2	53	27	44

[2] am Platze

3. Schritt: Füge durch binäre Suche das erste Element des unsortierten Bereichs $a(4)$ in den sortierten Bereich $a(1) \cdots a(3)$ ein. Dazu rücke alle Elemente des sortierten Bereichs von der Einfügestelle $i = 4$ an um eins nach rechts und schreibe das zuvor gesicherte Element $a(4)$ an die Stelle $i = 4$.

15	**33**	**47**	**87**	98	17	53	76	83	2	53	27	44

4. Schritt: Füge durch binäre Suche das erste Element des unsortierten Bereichs $a(5)$ in den sortierten Bereich $a(1) \cdots a(4)$ ein. Schreibe das zuvor gesicherte Element $a(5)$ an die Stelle $i = 5$.

15	**33**	**47**	**87**	**98**	17	53	76	83	2	53	27	44

5. Schritt: Füge durch binäre Suche das erste Element des unsortierten Bereichs $a(6)$ in den sortierten Bereich $a(1) \cdots a(5)$ ein. Dazu rücke alle Elemente des sortierten Bereichs von der Einfügestelle $i = 2$ an um eins nach rechts und schreibe das zuvor gesicherte Element $a(6)$ an die Stelle $i = 2$.

15	**17**	**33**	**47**	**87**	**98**	53	76	83	2	53	27	44

6. Schritt: Füge durch binäre Suche das erste Element des unsortierten Bereichs $a(7)$ in den sortierten Bereich $a(1) \cdots a(6)$ ein. Dazu rücke alle Elemente des sortierten Bereichs von der Einfügestelle $i = 5$ an um eins nach rechts und schreibe das zuvor gesicherte Element $a(7)$ an die Stelle $i = 5$.

15	**17**	**33**	**47**	**53**	**87**	**98**	76	83	2	53	27	44

7. Schritt: Füge durch binäre Suche das erste Element des unsortierten Bereichs $a(8)$ in den sortierten Bereich $a(1) \cdots a(7)$ ein. Dazu rücke alle Elemente des sortierten Bereichs von der Einfügestelle $i = 6$ an um eins nach rechts und schreibe das zuvor gesicherte Element $a(8)$ an die Stelle $i = 6$.

15	**17**	**33**	**47**	**53**	**76**	**87**	**98**	83	2	53	27	44

8. Schritt:

15	**17**	**33**	**47**	**53**	**76**	**83**	**87**	**98**	2	53	27	44

9. Schritt:

2	**15**	**17**	**33**	**47**	**53**	**76**	**83**	**87**	**98**	53	27	44

10. Schritt:

2	**15**	**17**	**33**	**47**	**53**	**53**	**76**	**83**	**87**	**98**	27	44

11. Schritt:

2	**15**	**17**	**27**	**33**	**47**	**53**	**53**	**76**	**83**	**87**	**98**	44

12. Schritt:

2	**15**	**17**	**27**	**33**	**44**	**47**	**53**	**53**	**76**	**83**	**87**	**98**

2.2.4 Komplexitätsanalyse

2.2.4.1 worst case Ansatz für die Anzahl der Vergleiche:

$$\begin{aligned} V(n) = \sum_{i=1}^{n-1} \lceil \log_2(i+1) \rceil = \sum_{i=2}^{n} \lceil \log_2(i) \rceil &< \sum_{i=2}^{n} (\log_2(i) + 1) \\ &< \sum_{i=1}^{n} (\log_2(i) + 1) \\ &< \log_2(n!) + n \end{aligned}$$

$n!$ kann man approximieren mit $\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$. Damit ergibt sich

$$\begin{aligned} V(n) &< \log\left(\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n\right) + n \\ &< \log\left(\sqrt{2\pi n}\right) + n \cdot \log\left(\frac{n}{e}\right) + n \\ &< \log\left(\sqrt{2\pi}\right) + \frac{1}{2} \cdot \log(n) + n \cdot \log(n) - n \cdot \log(e) + n \end{aligned}$$

Ergebnis für die Anzahl der Vergleiche:

$$n \cdot \log_2(n) - 0.4427\, n + O(\log_2(n))$$

Die Anzahl der Vergleiche fällt also recht günstig aus. Jedoch ist die Anzahl an Rechtsverschiebungen (Operationen) hoch. Für *eines* der Elemente stellt man folgende Überlegung an:

Zu Beginn steht das Element a_i an Position i, nach dem Sortieren mit der Wahrscheinlichkeit $\frac{1}{n}$ an Position j. Ist $j > i$, so wurde das Element $j - i$ mal nach rechts gerückt. Im Durchschnitt ergibt sich die Anzahl der Rechtsverschiebungen zu

$$\begin{aligned}
\underbrace{\frac{1}{n}}_{\text{WSK}} \cdot \left(\underbrace{0 + \cdots + 0}_{\text{Position 1 bis } i} + \underbrace{1}_{i+1} + \cdots + \underbrace{(n-i)}_{\text{Position } n} \right) &= \frac{1}{n} \cdot \sum_{j=1}^{n-i} j \\
&= \frac{1}{n} \cdot \frac{(n-i)\cdot(n-i+1)}{2} \\
&= \frac{(n-i)\cdot(n-i+1)}{2n}
\end{aligned}$$

Das war für ein Element, wir fügen aber insgesamt n ein, dies führt zu der Summe

$$\begin{aligned}
\sum_{i=1}^{n} \frac{(n-i)\cdot(n-i+1)}{2n} &= \frac{1}{2n} \cdot \sum_{i=1}^{n} (n-i)\cdot(n-i+1) \\
&= \frac{1}{2n} \cdot \sum_{i=1}^{n} i \cdot (i+1) \\
&= \frac{1}{2n} \cdot \left(\frac{1}{6} \cdot n \cdot (n+1) \cdot (2n+1) - \frac{1}{2} n \cdot (n+1) \right) \\
&= \frac{1}{6} \cdot \left(n^2 - 1\right)
\end{aligned}$$

2.3 Quicksort

2.3.1 Funktionsweise

Wähle ein Pivotelement als Trennpunkt für die beiden Teilprobleme. Gängige Methoden sind:

1. Wähle stets das im Array an vorderster Stelle stehende Datum
2. Wähle mit einem Zufallsgenerator eine Position und damit ein Datum zufällig aus.
3. Wähle aus dem Array die drei Daten, die ganz vorne, ganz hinten und in der Mitte[3] stehen. Berechne das bezgl. der Größe mittlere dieser drei Daten.
4. Wähle mit einem Zufallsgenerator drei Positionen und damit Daten und bestimme den Median, d.h. das bzgl. der Größe mittlere Datum.

Das Pivotelement sei nun als a_i bekannt und das Array habe die Positionen $1, \cdots, n$. Der Algorithmus sorgt dafür, dass links vom Pivotelement nur kleiner Daten und rechts davon nur größere Daten stehen.

1. Laufe von a_1 aus nach rechts bis ein Datum $a_l \geq a_i$ gefunden wird. Spätestens für $l = i$ ist dies der Fall. Dabei wird durch Indexvergleiche sichergestellt, dass a_i nichtmit a_i verglichen wird.
2. Laufe von a_n aus nach links bis ein Datum $a_r \leq a_i$ gefunden wird. Spätestens für $r = i$ ist dies der Fall. Dabei wird durch Indexvergleiche sichergestellt, dass a_i nichtmit a_i verglichen wird.
3. Falls $l = i = r$ STOP.
   Sonst vertausche a_l mit a_r. Suche an den Positionen
   - l falls $r = i$

[3] bei gerader Anzahl das linke der beiden mittleren

- $l + 1$ falls $r \neq i$
- r falls $l = i$
- $r - 1$ falls $l \neq i$

weiter. Es ist zu beachten, dass das Pivotelement an anderer Position stehen kann.

2.3.2 Eigenschaften

- internes Sortierverfahren
- arbeitet in situ
- arbeitet nach dem divide and conquer Prinzip

2.3.3 Beispiel

Gegeben sei folgendes Array $a_1, \cdots, a_n$:

15	47	33	87	98	17	53	76	83	2	53	27	44

1. Schritt:

   Setze $L := 1$ und $R := n$. Der Wert für I wird durch eine der Methoden 1 bis 4 bestimmt, hier gilt z.B. $I := 7$.

L						I						R
15	47	33	87	98	17	53	76	83	2	53	27	44

2. Schritt:

   Suche beginnend bei a_1 nach einem Datum $a_L \geq a_i$.

			L			I						R
15	47	33	87	98	17	53	76	83	2	53	27	44

3. Schritt:

   Suche beginnend bei a_n nach einem Datum $a_R \leq a_i$.

			L			I						R
15	47	33	87	98	17	53	76	83	2	53	27	44

4. Schritt:

   Man stellt fest: $L \neq I \neq R$. Also vertausche a_L mit a_R. Fahre an den Positionen $L + 1$ bzw. $R - 1$ fort, dort findet man Daten, für die $a_L \geq a_i$ bzw. $a_R \leq a_i$ gilt. Vgl. Schritt 1 und Schritt 2.

				L		I					R	
15	47	33	44	98	17	53	76	83	2	53	27	87

5. Schritt:

   Man stellt wiederum fest: $L \neq I \neq R$. Also vertausche a_L mit a_R. Fahre an den Positionen $L + 1$ bzw. $R - 1$ fort.

					L	I				R		
15	47	33	44	27	17	53	76	83	2	53	98	87

6. Schritt:

   Suche weiter nach einem Datum $a_L \geq a_i$ (analog zu Schritt 1). Der Wert für L steigt weiter an, bis er I erreicht und überschreitet diesen nicht.

						L,I				R		
15	47	33	44	27	17	53	76	83	2	53	98	87

7. Schritt:

   Suche weiter nach einem Datum $a_R \leq a_i$ (analog zu Schritt 2).

						L,I				R		
15	47	33	44	27	17	53	76	83	2	53	98	87

8. Schritt:

   $L = I \neq R$. Also vertausche a_L mit a_R. Das Pivotelement wird verschoben, somit auch I. Fahre an den Positionen $L + 1$ bzw. R fort.

							L			R,I		
15	47	33	44	27	17	53	76	83	2	53	98	87

9. Schritt:

   An den neuen Positionen L bzw. R findet man Daten, die dem ersten bzw. zweiten Schritt des Algorithmus genügen.

   $L \neq I = R$. Also vertausche a_L mit a_R. Das Pivotelement wird verschoben, somit auch I. Fahre an den Positionen L bzw. $R - 1$ fort.

							L,I		R			
15	47	33	44	27	17	53	53	83	2	76	98	87

10. Schritt:

    An den neuen Positionen L bzw. R findet man Daten, die dem ersten bzw. zweiten Schritt des Algorithmus genügen.

    $L = I \neq R$. Also vertausche a_L mit a_R. Das Pivotelement wird verschoben, somit auch I. Fahre an den Positionen $L + 1$ bzw. R fort.

								L	R,I			
15	47	33	44	27	17	53	2	83	53	76	98	87

11. Schritt:

    An den neuen Positionen L bzw. R findet man Daten, die dem ersten bzw. zweiten Schritt des Algorithmus genügen.

    $L \neq I = R$. Also vertausche a_L mit a_R. Das Pivotelement wird verschoben, somit auch I. Fahre an den Positionen L bzw. $R - 1$ fort.

								L,I	R			
15	47	33	44	27	17	53	2	53	83	76	98	87

12. Schritt:

    Man stellt fest: $L = I = R$. STOP, das Pivotelement ist richtig eingeordnet.

								L,I,R				
15	47	33	44	27	17	53	2	**53**	83	76	98	87

13. Schritt: Das Pivotelement ist nun richtig eingeordnet. Es werden nun die beiden Teilprobleme gelöst.[4] Zwei neue Pivotelemente wurden bestimmt.

L1			I1				R1		L2	I2		R2
15	47	33	44	27	17	53	2	**53**	83	76	98	87

14. Schritt:

	L1		I1				R1		L2	I2,R2		
15	47	33	44	27	17	53	2	**53**	83	76	98	87

15. Schritt:

		L1	I1			R1			L2,I2,R2			
15	2	33	44	27	17	53	47	**53**	76	83	98	87

[4] Die Probleme werden eigentlich hintereinander gelöst, hier machen wir das ausnahmsweise mal parallel.

16. Schritt:

			L1,I1		R1				L2,I2,R2			
15	2	33	44	27	17	53	47	**53**	**76**	83	98	87

17. Schritt:

				L1	I1,R1					L3	I3	R3
15	2	33	17	27	44	53	47	**53**	**76**	83	98	87

18. Schritt:

					L1,I1,R1						L3,I3	R3
15	2	33	17	27	44	53	47	**53**	**76**	83	98	87

19. Schritt:

					L1,I1,R1							L3,I3,R3
15	2	33	17	27	**44**	53	47	**53**	**76**	83	87	98

20. Schritt:

L1		I1		R1		L2,I2	R2			L3,I3	R3	
15	2	33	17	27	**44**	53	47	**53**	**76**	83	87	**98**

21. Schritt:

		L1,I1		R1			L2,I2,R2			L3,I3,R3		
15	2	33	17	27	**44**	47	53	**53**	**76**	83	87	**98**

22. Schritt:

			L1	I1,R1		L2,I2,R2					L3,I3,R3	
15	2	27	17	33	**44**	47	**53**	**53**	**76**	**83**	87	**98**

23. Schritt:

				L1,I1,R1								
15	2	27	17	33	**44**	**47**	**53**	**53**	**76**	**83**	**87**	**98**

24. Schritt:

L1,I1	R1	L2,I2	R2									
15	2	27	17	**33**	**44**	**47**	**53**	**53**	**76**	**83**	**87**	**98**

25. Schritt:

	L1,I1,R1		L2,I2,R2									
2	15	17	27	**33**	**44**	**47**	**53**	**53**	**76**	**83**	**87**	**98**

26. Schritt:

L1,I1,R1		L2,I2,R2										
2	**15**	17	**27**	**33**	**44**	**47**	**53**	**53**	**76**	**83**	**87**	**98**

27. Schritt:

2	**15**	**17**	**27**	**33**	**44**	**47**	**53**	**53**	**76**	**83**	**87**	**98**

2.3.4 Komplexitätsanalyse

2.3.4.1 worst case

- Varianten 1 und 2

Ansatz für die Anzahl der Vergleiche:

$$\begin{aligned} V(n) &\geq n-1+V(n-1) \\ &\geq n-1+n-2+V(n-2) \\ &\geq n-1+n-2+\cdots+2+1 \\ &\geq \sum_{i=1}^{n-1} i \end{aligned}$$

Ergebnis für die Anzahl der Vergleiche:

$$V(n) \geq \frac{n(n-1)}{2}$$

- Varianten 3 und 4
  Ansatz für die Anzahl der Vergleiche:

$$\begin{aligned} V(n) &\geq n+V(n-2) \\ &\geq n+(n-2)+V(n-4) \\ &\geq n+(n-2)+\cdots+(n-(n+2))+(n-n) \\ &\geq n+n+\cdots+n-2-4-\cdots-(n+2)-n \\ &\geq \frac{1}{2}n^2-\sum_{i=1}^{\frac{1}{2}n} 2i \\ &\geq \frac{1}{2}n^2-2\sum_{i=1}^{\frac{1}{2}n} i \\ &\geq \frac{1}{2}n^2-2\frac{\frac{1}{2}n\cdot\left(\frac{1}{2}n+1\right)}{2} \\ &\geq \frac{1}{2}n^2-\frac{1}{2}n\cdot\left(\frac{1}{2}n+1\right) \\ &\geq \frac{1}{2}n^2-\frac{1}{4}n^2+\frac{1}{2}n \end{aligned}$$

  Ergebnis für die Anzahl der Vergleiche:

$$V(n) \geq \frac{1}{4}n^2-\frac{1}{2}n$$

2.4 HeapSort

Ein Heap ist ein Array, das als binärer Baum interpretiert wird. Jedoch werden keine Zeiger verwaltet. Die Nachfolger (Kinder) des Elements $a[i]$ findet man an den Positionen $a[2i] \leq n$ bzw. $a[2i+1] \leq n$.

- Max-Heap:
  Das Element an Position i ist nicht kleiner als die Nachfolger.
- Min-Heap:
  Das Element an Position i ist nicht größer als die Nachfolger.

HeapSort besteht aus zwei Phasen. In der ersten Phase, der Heap "Creation" Phase, wird aus dem gegebenen Array ein Heap konstruiert. In der zweiten Phase, der Heap "Selection" Phase, wird eine Vertauschung vorgenommen, damit die Heap-Bedingung an der Position $a[1]$ verletzt und dieser Fehler wieder korrigiert.
Insgesamt kann man folgendes Rahmenprogramm erstellen:

- Heap Creation Phase
  FOR $i = \lfloor \frac{n}{2} \rfloor \cdots 1$: *Methodenaufruf*
- Heap Selection Phase
  FOR $m = n \cdots 2$: *Methodenaufruf*

Gegeben sei folgendes Array

15	47	33	87	98	17	53	76	83	2	53	27	44

2.4.1 Reheap-Variante

2.4.1.1 Funktionsweise

Heap Creation Methodenaufruf:
$reheap(i, n)$
Heap Selection Methodenaufruf:
$reheap(1, m-1)$
Reheap:

2.4.1.2 Beispiel

- Heap Creation-Phase:

1. reheap$(6, 13)$:
   Feststellung: $6 < \frac{13}{2}$, $a(6)$ hat zwei Söhne, nämlich $a(12)$ und $a(13)$, wobei $a(12) < a(13)$ und $a(6) < a(12)$. STOP

15	47	33	87	98	17	53	76	83	2	53	27	44

2. reheap$(5, 13)$:
   Feststellung: $5 < \frac{13}{2}$, $a(5)$ hat zwei Söhne, nämlich $a(10)$ und $a(11)$, wobei $a(10) < a(11)$ und $a(5) > a(10)$. Vertausche $a(5)$ und $a(10)$.

15	47	33	87	2	17	53	76	83	98	53	27	44

   Rufe rekursiv reheap$(10, 13)$ auf. Feststellung: $10 > \frac{13}{2}$. STOP

15	47	33	87	2	17	53	76	83	98	53	27	44

3. reheap$(4, 13)$:
   Feststellung: $4 < \frac{13}{2}$, $a(4)$ hat zwei Söhne, nämlich $a(8)$ und $a(9)$, wobei $a(8) < a(9)$ und $a(4) > a(8)$. Vertausche $a(4)$ und $a(8)$.

15	47	33	76	2	17	53	87	83	98	53	27	44

   Rufe rekursiv reheap$(8, 13)$ auf. Feststellung: $8 > \frac{13}{2}$. STOP

15	47	33	87	2	17	53	76	83	98	53	27	44

4. reheap$(3, 13)$:
   Feststellung: $3 < \frac{13}{2}$, $a(3)$ hat zwei Söhne, nämlich $a(6)$ und $a(7)$, wobei $a(6) < a(7)$ und $a(3) > a(6)$. Vertausche $a(3)$ und $a(6)$.

15	47	17	76	2	33	53	87	83	98	53	27	44

   Rufe rekursiv reheap$(6, 13)$ auf. Feststellung: $6 < \frac{13}{2}$. $a(6)$ hat zwei Söhne, nämlich $a(12)$ und $a(13)$, wobei $a(12) < a(13)$ und $a(6) > a(12)$. Vertausche $a(6)$ und $a(12)$.

15	47	17	76	2	27	53	87	83	98	53	33	44

   Rufe rekursiv reheap$(12, 13)$ auf.Feststellung: $12 > \frac{13}{2}$. STOP

15	47	17	76	2	27	53	87	83	98	53	33	44

5. reheap$(2, 13)$:
   Feststellung: $2 < \frac{13}{2}$, $a(2)$ hat zwei Söhne, nämlich $a(4)$ und $a(5)$, wobei $a(5) < a(4)$ und $a(2) > a(5)$. Vertausche $a(2)$ und $a(5)$.

15	2	17	76	47	27	53	87	83	98	53	33	44

   Rufe rekursiv reheap$(5, 13)$ auf. Feststellung: $5 < \frac{13}{2}$, $a(5)$ hat zwei Söhne, nämlich $a(10)$ und $a(11)$, wobei $a(11) < a(10)$ und $a(5) < a(10)$. STOP

15	2	17	76	47	27	53	87	83	98	53	33	44

6. reheap$(1, 13)$:
   Feststellung: $1 < \frac{13}{2}$, $a(1)$ hat zwei Söhne, nämlich $a(2)$ und $a(3)$, wobei $a(2) < a(3)$ und $a(1) > a(2)$. Vertausche $a(1)$ und $a(2)$.

2	15	17	76	47	27	53	87	83	98	53	33	44

   Rufe rekursiv reheap$(2, 13)$ auf. Feststellung: $2 < \frac{13}{2}$, $a(2)$ hat zwei Söhne, nämlich $a(4)$ und $a(5)$, wobei $a(5) < a(4)$ und $a(2) < a(5)$. STOP

- Selection-Phase:

1. Schritt:
   (a) Vertausche $a(1)$ und $a(13)$

44	15	17	76	47	27	53	87	83	98	53	33	**2**

   (b) reheap$(1, 12)$:

15	44	17	76	47	27	53	87	83	98	53	33	**2**

2. Schritt:
   (a) Vertausche $a(1)$ und $a(12)$

33	44	17	76	47	27	53	87	83	98	53	**15**	**2**

   (b) reheap$(1, 11)$:

17	44	27	76	47	33	53	87	83	98	53	**15**	**2**

3. Schritt:
   (a) Vertausche $a(1)$ und $a(11)$

53	44	27	76	47	33	53	87	83	98	**17**	**15**	**2**

   (b) reheap$(1, 10)$:

27	44	33	76	47	53	53	87	83	98	**17**	**15**	**2**

4. Schritt:
   (a) Vertausche $a(1)$ und $a(10)$

98	44	33	76	47	53	53	87	83	**27**	**17**	**15**	**2**

   (b) reheap$(1, 9)$:

33	44	53	76	47	98	53	87	83	**27**	**17**	**15**	**2**

5. Schritt:
   (a) Vertausche $a(1)$ und $a(9)$

83	44	53	76	47	98	53	87	**33**	**27**	**17**	**15**	**2**

   (b) reheap$(1, 8)$:

44	47	53	76	83	98	53	87	**33**	**27**	**17**	**15**	**2**

6. Schritt:
   (a) Vertausche $a(1)$ und $a(8)$

87	47	53	76	83	98	53	**44**	**33**	**27**	**17**	**15**	**2**

   (b) reheap$(1, 7)$:

47	76	53	87	83	98	53	**44**	**33**	**27**	**17**	**15**	**2**

7. Schritt:
   (a) Vertausche $a(1)$ und $a(7)$

53	76	53	87	83	98	**47**	**44**	**33**	**27**	**17**	**15**	**2**

   (b) reheap$(1, 6)$:

53	76	53	87	83	98	**47**	**44**	**33**	**27**	**17**	**15**	**2**

8. Schritt:
   (a) Vertausche $a(1)$ und $a(6)$

98	76	53	87	83	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

   (b) reheap$(1, 5)$:

53	76	98	87	83	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

9. Schritt:
   (a) Vertausche $a(1)$ und $a(5)$

83	76	98	87	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

   (b) reheap$(1, 4)$:

76	83	98	87	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

10. Schritt:
    (a) Vertausche $a(1)$ und $a(4)$

87	83	98	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

(b) reheap(1, 3):

83	87	98	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

11. Schritt:

(a) Vertausche $a(1)$ und $a(3)$

98	87	**83**	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

(b) reheap(1, 2):

87	98	**83**	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

12. Schritt:

(a) Vertausche $a(1)$ und $a(2)$

98	**87**	**83**	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

(b) reheap(1, 1):

98	**87**	**83**	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

2.4.2 Bottom-Up-reheap

2.4.2.1 Beispiel Gegeben sei folgendes Array

15	47	33	87	98	17	53	76	83	2	53	27	44

- Heap Creation-Phase:

1. Schritt:

bottom-up-reheap(6, 13)

(a) leaf-search(6, 13) liefert $j = 12$

(b) bottom-up-search(6, 12) liefert $j = 6$

(c) interchange macht in diesem Fall nichts

15	47	33	87	98	17	53	76	83	2	53	27	44

2. Schritt:

bottom-up-reheap(5, 13)

(a) leaf-search(5, 13) liefert $j = 10$

(b) bottom-up-search(5, 10) liefert $j = 10$

(c) interchange(5, 10) speichert die 98 zwischen, verschiebt die 2 um eine Ebene nach oben, und fügt die 98 an Position 10 ein.

15	47	33	87	2	17	53	76	83	98	53	27	44

3. Schritt:

bottom-up-reheap(4, 13)

(a) leaf-search(4, 13) liefert $j = 8$

(b) bottom-up-search(4, 8) liefert $j = 8$

(c) interchange(4, 8) speichert die 87 zwischen, verschiebt die 76 um eine Ebene nach oben, und fügt die 87 an Position 8 ein.

15	47	33	76	2	17	53	87	83	98	53	27	44

4. Schritt:

bottom-up-reheap(3, 13)

(a) leaf-search(3, 13) liefert $j = 12$

(b) bottom-up-search(3, 12) liefert $j = 12$

(c) interchange(3, 12) speichert die 33 zwischen, verschiebt jeweils die 17 und die 27 um eine Ebene nach oben, und fügt die 33 an Position 12 ein.

15	47	17	76	2	27	53	76	83	98	53	33	44

5. Schritt:

bottom-up-reheap(2, 13)

(a) leaf-search(2, 13) liefert $j = 11$

(b) bottom-up-search$(2, 11)$ liefert $j = 5$

(c) interchange$(2, 5)$ speichert die 47 zwischen, verschiebt die 2 um eine Ebene nach oben, und fügt die 47 an Position 5 ein.

15	2	17	76	47	27	53	87	83	98	53	33	44

6. Schritt:

bottom-up-reheap$(1, 13)$

(a) leaf-search$(1, 13)$ liefert $j = 11$

(b) bottom-up-search$(1, 11)$ liefert $j = 2$

(c) interchange$(1, 2)$ speichert die 15 zwischen, verschiebt die 2 um eine Ebene nach oben, und fügt die 15 an Position 2 ein.

2	15	17	76	47	27	53	87	83	98	53	33	44

- Selection-Phase:

1. Schritt:

(a) Vertausche $a(1)$ und $a(13)$

15	44	17	76	47	27	53	87	83	98	53	33	**2**

(b) bottom-up-reheap

i. leaf-search$(1, 12)$ liefert $j = 11$

ii. bottom-up-search$(1, 11)$ liefert $j = 2$

iii. interchange$(1, 2)$ speichert die 44 zwischen, verschiebt die 15 um eine Ebene nach oben, und fügt die 44 an Position 2 ein.

15	44	17	76	47	27	53	87	83	98	53	33	**2**

2. Schritt:

(a) Vertausche $a(1)$ und $a(12)$

33	44	17	76	47	27	53	87	83	98	53	**15**	**2**

(b) bottom-up-reheap

i. leaf-search$(1, 11)$ liefert $j = 6$

ii. bottom-up-search$(1, 6)$ liefert $j = 6$

iii. interchange$(1, 6)$ speichert die 33 zwischen, verschiebt die 17 und die 27 jeweils um eine Ebene nach oben, und fügt die 33 an Position 6 ein.

17	44	27	76	47	33	53	87	83	98	53	**15**	**2**

3. Schritt:

(a) Vertausche $a(1)$ und $a(11)$

53	44	27	76	47	33	53	87	83	98	**17**	**15**	**2**

(b) bottom-up-reheap

i. leaf-search$(1, 10)$ liefert $j = 6$

ii. bottom-up-search$(1, 6)$ liefert $j = 6$

iii. interchange$(1, 6)$ speichert die 53 zwischen, verschiebt die 27 und die 33 jeweils um eine Ebene nach oben, und fügt die 53 an Position 6 ein.

27	44	33	76	47	53	53	87	83	98	**17**	**15**	**2**

4. Schritt:

(a) Vertausche $a(1)$ und $a(10)$

98	44	33	76	47	53	53	87	83	**27**	**17**	**15**	**2**

(b) bottom-up-reheap

i. leaf-search$(1, 9)$ liefert $j = 7$

ii. bottom-up-search$(1, 7)$ liefert $j = 7$

iii. interchange$(1, 7)$ speichert die 98 zwischen, verschiebt die 33 und die 53 jeweils um eine Ebene nach oben, und fügt die 98 an Position 7 ein.

33	44	53	76	47	53	98	87	83	**27**	**17**	**15**	**2**

5. Schritt:
   (a) Vertausche $a(1)$ und $a(9)$

83	44	53	76	47	53	98	87	**33**	**27**	**17**	**15**	**2**

   (b) bottom-up-reheap
      i. leaf-search$(1, 8)$ liefert $j = 5$
      ii. bottom-up-search$(1, 5)$ liefert $j = 5$
      iii. interchange$(1, 5)$ speichert die 83 zwischen, verschiebt die 44 und die 47 jeweils um eine Ebene nach oben, und fügt die 83 an Position 5 ein.

44	47	53	76	83	53	98	87	**33**	**27**	**17**	**15**	**2**

6. Schritt:
   (a) Vertausche $a(1)$ und $a(8)$

87	47	53	76	83	53	98	**44**	**33**	**27**	**17**	**15**	**2**

   (b) bottom-up-reheap
      i. leaf-search$(1, 7)$ liefert $j = 4$
      ii. bottom-up-search$(1, 4)$ liefert $j = 4$
      iii. interchange$(1, 4)$ speichert die 87 zwischen, verschiebt die 47 und die 76 jeweils um eine Ebene nach oben, und fügt die 87 an Position 4 ein.

47	76	53	87	83	53	98	**44**	**33**	**27**	**17**	**15**	**2**

7. Schritt:
   (a) Vertausche $a(1)$ und $a(7)$

98	76	53	87	83	53	**47**	**44**	**33**	**27**	**17**	**15**	**2**

   (b) bottom-up-reheap
      i. leaf-search$(1, 6)$ liefert $j = 6$
      ii. bottom-up-search$(1, 6)$ liefert $j = 6$
      iii. interchange$(1, 6)$ speichert die 98 zwischen, verschiebt die 53 und die 53 jeweils um eine Ebene nach oben, und fügt die 98 an Position 6 ein.

53	76	53	87	83	98	**47**	**44**	**33**	**27**	**17**	**15**	**2**

8. Schritt:
   (a) Vertausche $a(1)$ und $a(6)$

98	76	53	87	83	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

   (b) bottom-up-reheap
      i. leaf-search$(1, 5)$ liefert $j = 3$
      ii. bottom-up-search$(1, 3)$ liefert $j = 3$
      iii. interchange$(1, 3)$ speichert die 98 zwischen, verschiebt die 53 um eine Ebene nach oben, und fügt die 98 an Position 3 ein.

53	76	98	87	83	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

9. Schritt:
   (a) Vertausche $a(1)$ und $a(5)$

83	76	98	87	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

   (b) bottom-up-reheap
      i. leaf-search$(1, 4)$ liefert $j = 4$
      ii. bottom-up-search$(1, 4)$ liefert $j = 2$
      iii. interchange$(1, 2)$ speichert die 83 zwischen, verschiebt die 76 um eine Ebene nach oben, und fügt die 83 an Position 2 ein.

76	83	98	87	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

10. Schritt:
   (a) Vertausche $a(1)$ und $a(4)$

87	83	98	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

   (b) bottom-up-reheap

i. leaf-search$(1,3)$ liefert $j = 2$

ii. bottom-up-search$(1,2)$ liefert $j = 2$

iii. interchange$(1,2)$ speichert die 87 zwischen, verschiebt die 83 um eine Ebene nach oben, und fügt die 87 an Position 2 ein.

83	87	98	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

11. Schritt:

(a) Vertausche $a(1)$ und $a(3)$

98	87	**83**	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

(b) bottom-up-reheap

i. leaf-search$(1,2)$ liefert $j = 2$

ii. bottom-up-search$(1,2)$ liefert $j = 2$

iii. interchange$(1,2)$ speichert die 98 zwischen, verschiebt die 87 um eine Ebene nach oben, und fügt die 98 an Position 2 ein.

87	98	**83**	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

12. Schritt:

(a) Vertausche $a(1)$ und $a(2)$

98	**87**	**83**	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

(b) bottom-up-reheap

i. leaf-search$(1,1)$ liefert $j = 1$

ii. bottom-up-search$(1,1)$ liefert $j = 1$

iii. interchange$(1,1)$

98	**87**	**83**	**76**	**53**	**53**	**47**	**44**	**33**	**27**	**17**	**15**	**2**

2.5 MergeSort

2.5.1 Funktionsweise

Zwei sortierte Teilfolgen, die auf verschiedenen Bändern stehen, können per "Reißverschlußverfahren" zu einer sortierten Teilfolge verschmolzen (*merge*) werden.

- MERGE$(a_1, \cdots, a_n; b_1, \cdots, b_m)$[5]:
  Vergleiche die vordersten Elemente der beiden Folgen und schreibe das kleinere Datum in die Ausgabe. Für die zugehörige Folge gehe zum nächsten Datum. Wenn für eine Folge das Ende erreicht wird, kann der Rest der anderen Folge in die Ausgabe kopiert werden.
  Es genügen $n + m - 1$ Vergleiche. Für das letzte Element auf dem Band muss kein Vergleich gemacht werden.

- MERGESORT$(a_1, \cdots, a_n)$

  1. Falls $n = 1$ STOP.
  2. Falls $n > 1$,
     - $\left(b_1, \cdots, b_{\lceil \frac{n}{2} \rceil}\right)$:=MERGESORT$\left(a_1, \cdots, a_{\lceil \frac{n}{2} \rceil}\right)$
     - $\left(c_1, \cdots, c_{\lfloor \frac{n}{2} \rfloor}\right)$:=MERGESORT$\left(a_{\lceil \frac{n}{2} \rceil + 1}, \cdots, a_n\right)$
     - $(d_1, \cdots, d_n)$:=MERGE$\left(b_1, \cdots, b_{\lceil \frac{n}{2} \rceil}; c_1, \cdots, c_{\lfloor \frac{n}{2} \rfloor}\right)$

2.5.2 Eigenschaften

- Verfahren zum externen Sortieren
- nicht in situ

[5] $a_1 \leq \cdots \leq a_n$ und $b_1 \leq \cdots \leq b_m$

2.5.3 Beispiele:

Gegeben sei folgendes Array

15	47	33	87	98	17	53	76	83	2	53	27	44

2.5.3.1 Zwei Phasen - Drei Bänder

Funktionsweise: Zu Beginn liegen sortierte Teilfolgen (Blöcke) der Länge 1 vor.
Phase 1: Diese werden abwechselnd auf Band B und C geschrieben.
Phase 2: Anschließend werden der m-te Block auf Band B und der m-te Block auf Band C gemsicht und das Ergebnis auf Band A geschrieben.
Damit ist eine Runde abgeschlossen. In der nächsten Runde wird wieder die gleiche Idee angwandt, nur sind die Blocklängen größer.

A	15	47	33	87	98	17	53	76	83	2	53	27	44
B													
C													

1. Runde:

   - Phase (1): Kopiere sortierte Blöcke der Länge 1 abwechselnd auf das zweite und dritte Band.

A													
B	**15**	33	**98**	53	**83**	53	**44**						
C	**47**	87	**17**	76	**2**	27							

   - Phase (2): Merge die zueinandergehörenden Blöcke von Band B und C und schreibe das Ergebnis auf Band A

A	15	47	33	87	17	98	53	76	2	83	27	53	44

2. Runde:

   - Phase (1): Kopiere sortierte Blöcke der Länge 2 abwechselnd auf das zweite und dritte Band.

A													
B	**15**	**47**	17	98	**2**	**83**	44						
C	**33**	**87**	53	76	**27**	**53**							

   - Phase (2): Merge die zueinandergehörenden Blöcke von Band B und C und schreibe das Ergebnis auf Band A

A	15	33	47	87	17	53	76	98	2	27	53	83	44

3. Runde:

   - Phase (1): Kopiere sortierte Blöcke der Länge 4 abwechselnd auf das zweite und dritte Band.

A													
B	**15**	**33**	**47**	**87**	2	27	53	83					
C	**17**	**53**	**76**	**98**	44								

   - Phase (2): Merge die zueinandergehörenden Blöcke von Band B und C und schreibe das Ergebnis auf Band A

A	15	17	33	47	53	76	87	98	2	27	44	53	83

4. Runde:

   - Phase (1): Kopiere sortierte Blöcke der Länge 8 abwechselnd auf das zweite und dritte Band.

A													
B	**15**	**17**	**33**	**47**	**53**	**76**	**87**	**98**					
C	**2**	**27**	**44**	**53**	**83**								

   - Phase (2): Merge die zueinandergehörenden Blöcke von Band B und C und schreibe das Ergebnis auf Band A

A	2	15	17	27	33	44	47	53	53	76	83	87	98

2.5.3.2 Eine Phase - Drei Bänder

Beispiel 1: Die Anzahl an Elementen wurde so gewählt, dass zu jeder Block ein Gegenstück auf einem zweiten Band hat, sozusagen ein Idealfall.
Suche $j \in \mathbb{N}$ mit Fib (j) $\geq n$. Hier gilt $j = 6$, da Fib (6) = 13. Kopiere Fib (j − 1) = 8 Daten auf Band B.

A	**83**	2	**53**	27	**44**								
B	**15**	47	**33**	87	**98**	17	**53**	76					
C													

1. Durchlauf: Mische Fib (j − 2) = 5 zusammengehörende Blöcke von Band A und B und schreibe das Ergebnis in C.

A													
B	**17**	53	**76**										
C	**15**	**83**	2	47	**33**	**53**	27	87	**44**	**98**			

2. Durchlauf: Mische Fib (j − 3) = 3 zusammengehörende Blöcke von Band B und C und schreibe das Ergebnis in A.

A	**15**	**17**	**83**	2	47	53	**33**	**53**	**76**				
B													
C	**27**	**87**	44	98									

3. Durchlauf: Mische Fib (j − 4) = 2 zusammengehörende Blöcke von Band A und C und schreibe das Ergebnis in B.

A	**33**	**53**	**76**										
B	**15**	**17**	**27**	**83**	**87**	2	44	47	53	98			
C													

4. Durchlauf: Mische Fib (j − 5) = 1 zusammengehörende Blöcke von Band A und B und schreibe das Ergebnis in C.

A													
B	**2**	**44**	**47**	**53**	**98**								
C	**15**	**17**	**27**	**33**	**53**	**76**	**83**	**87**					

5. Durchlauf: Mische Fib (j − 6) = 1 zusammengehörende Blöcke von Band B und C und schreibe das Ergebnis in A.

A	2	15	17	27	33	44	47	53	53	76	83	87	98
B													
C													

Beispiel 2: Die Anzahl der Elemente entspricht mit 12 keiner Fibonacci-Zahl mehr. Für einige Blöcke wird es auf dem anderen Band keine Gegenstücke geben. Dann wird so ein Block mit einem leeren Block gemsicht (siehe 1. Durchlauf).
Suche $j \in \mathbb{N}$ mit Fib (j) $\geq n$. Hier gilt $j = 6$, da Fib (6) = 13. Kopiere Fib (j − 1) = 8 Daten auf Band B.

A	**83**	2	**53**	27								
B	**15**	47	**33**	87	**98**	17	**53**	76				
C												

1. Durchlauf: Mische Fib (j − 2) = 5 zusammengehörende Blöcke von Band A und B und schreibe das Ergebnis in C.

A												
B	**17**	53	**76**									
C	**15**	**83**	2	47	**33**	**53**	27	87	**98**			

2. Durchlauf: Mische Fib (j − 3) = 3 zusammengehörende Blöcke von Band B und C und schreibe das Ergebnis in A.

A	**15**	**17**	**83**	2	47	53	**33**	**53**	**76**			
B												
C	**27**	**87**	98									

3. Durchlauf: Mische Fib (j − 4) = 2 zusammengehörende Blöcke von Band A und C und schreibe das Ergebnis in B.

A	**33**	**53**	**76**									
B	**15**	**17**	**27**	**83**	**87**	2	47	53	98			
C												

4. Durchlauf: Mische Fib (j − 5) = 1 zusammengehörende Blöcke von Band A und B und schreibe das Ergebnis in C.

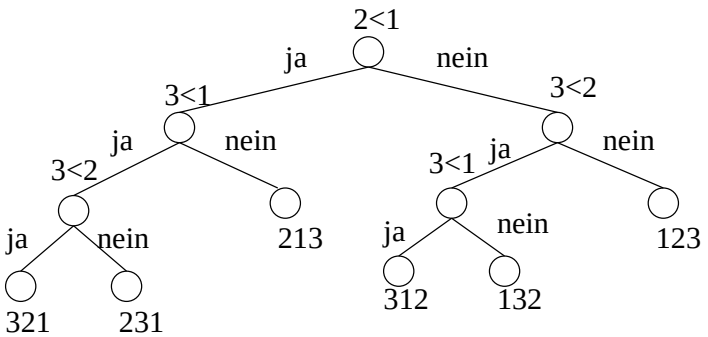
A												
B	**2**	**47**	**53**	**98**								
C	**15**	**17**	**27**	**33**	**53**	**76**	**83**	**87**				

5. Durchlauf: Mische Fib (j − 6) = 1 zusammengehörende Blöcke von Band B und C und schreibe das Ergebnis in A.

A	2	15	17	27	33	47	53	53	76	83	87	98
B												
C												

2.6 Untere Schranke für allgemeine Sortierverfahren

Als Eingabe für ein Sortierverfahren erhalten wir ein unsortiertes Array $a_1, \cdots, a_n$. Eine bestimmte Anordnung dieser n Elemente entspricht der sortierten Ausgabe. Wieviele solcher Anordnungen gibt es? Offensichtlich $n!$ wie man sich am folgenden Beispiel für $n = 3$ klarmachen kann:
$n = 3$: mögliche Anordnungen sind $a_1a_2a_3$, $a_1a_3a_2$, $a_2a_1a_3$, $a_2a_3a_1$, $a_3a_1a_2$, $a_3a_2a_1$
Wendet man ein Sortierverfahren auf das Eingabearray an, so werden nach und nach durch Vergleiche einige der Anordungen ausgeschlossen bis nur noch eine übrig bleibt, dann ist die Eingabe sortiert. Das Ausschliessen von Anordnungen kann an sog. Entscheidungsbäumen visualisiert werden.
An jedem Knoten des Baumes wird eine Entscheidung gemäß des Sortierverfahrens getroffen, so haben verschiedene Verfahren unterschiedliche Bäume auf der gleichen Eingabe. An den Blättern steht die gesuchte Anordnung. Bei $n!$ möglichen Anordungen gibt es also $n!$ Blätter.
Das folgende Beispiel arbeitet mit Insertion-Sort (auf Seite 9), an den Knoten steht $i < j$, was bedeutet, dass in der Ausgabe a_i vor a_j steht.

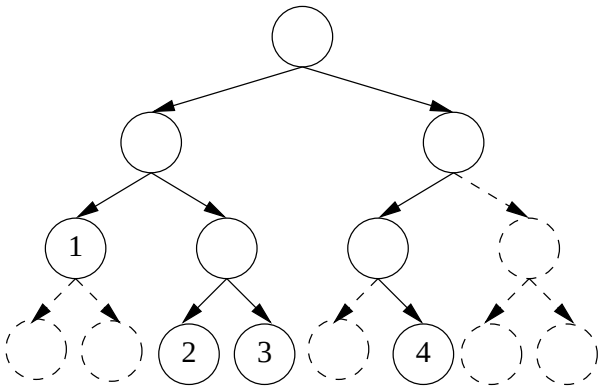


Durch die Darstellung als Baum haben wir einen guten Ausgangspunkt zur Bestimmung der worst- bzw avercase-case Anzahl an Vergleichen, was der entsprechenden worst- bzw avercase-case Tiefe des Baumes entspricht.

2.6.1 Worst case

Die Tiefe jedes binären Baumes mit N Blättern beträgt mindestens $\lceil \log_2 (N) \rceil$.

Beweis: Gegeben sein ein Baum der Tiefe d. Die Anzahl der Blätter eines binären Baumes kann man erhöhen, indem man u.a. einem Blatt auf Tiefe $d' < d$ zwei Kinder gibt.



Ein *vollständiger* binärerer Baum der Tiefe d enthält insgesamt $2^{d+1}-1$ Knoten , wovon 2^d-1 innere Knoten und 2^d Blätter sind.[6] Damit N Blätter existieren muss $2^d \geq N$ gelten, womit die Mindesttiefe festgelegt ist durch $d \geq \lceil \log_2(N) \rceil$.

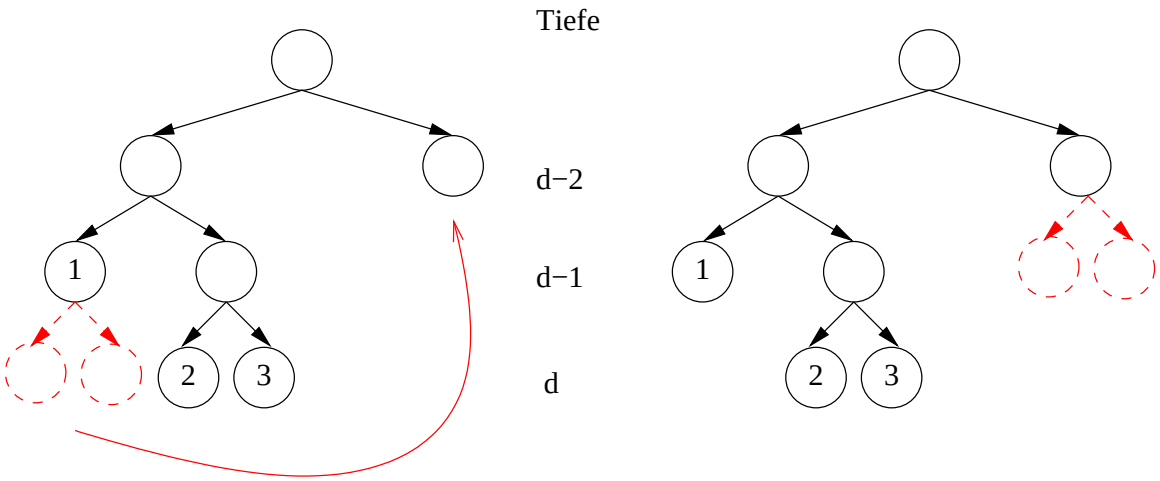
2.6.2 Average Case

Die durchschnittliche Tiefe eines binären Baumes mit N Blätern beträgt mindestens $\lceil \log_2(N) \rceil - \frac{2^{\lceil \log_2(N) \rceil}}{N} + 1$. Den Term kann man nach unten abschätzen durch $\lceil \log_2(N) \rceil - 1$. Es ist also mindestens eine Tiefe von $\lceil \log_2(N) \rceil - 1$ erforderlich.

Beweis:

Ein Entscheidungsbaum kann sehr schwer zu analysieren sein, deshalb transformiert man ihn in einen anderen binären Baum, dessen durchschnittliche Tiefe nicht größer ist. Dazu kann man erstmal folgende Feststellung machen:

Jedes Blatt auf Tiefe d hat ein Geschwist. Hätte es kein Geschwist, so wäre bereits beim Elter keine Entscheidung mehr möglich und dieses Blatt überflüssig. Es könnte somit gelöscht und die Tiefe abgesenkt werden.
Die weitere Vorgehensweise: Wir wollen die Blätter nur auf zwei benachbarten Ebenen d und $d-1$ des Baumes haben. Dazu hängen wir einige der Blätter um.



Falls es ein Blatt auf der Tiefe $d' \leq d-2$ gibt, so kann man die durchschnittliche Tiefe verringern. Dazu betrachten wir die folgenden Knoten:

- ein Blatt B der Tiefe d'
- zwei Geschwister auf Tiefe d

[6] Falls ein unvollständiger binärer Baum der Tiefe d vorliegt, so kann man die Anzahl der Blätter erhöhen, indem jedes Blatt auf Tiefe $d' < d$ zwei Söhne erhält.

Insgesamt haben sie die Tiefen $d' + 2d$.
Wir nehmen nun die beiden Geschwister und hängen sie unter B. Es fällt ein Blatt auf Tiefe d' weg (B), dafür entstehen zwei neue auf Ebene $d' + 1$ sowie ein Blatt auf Ebene $d - 1$.
Insgesamt haben sie die Tiefen $2 \cdot (d' + 1) + d - 1$.
Die durchschnittliche Tiefe nimmt durch unsere Umhängung nicht zu:

$$\begin{aligned} 2 \cdot (d' + 1) + d - 1 = 2d' + d + 1 = d' + d' + d + 1 & \overset{d' \leq d-2}{\leq} & d' + (d - 2) + d + 1 \\ & \leq & d' + 2d - 1 \\ & < & d' + 2d \end{aligned}$$

Nachdem die "Umhängungen" abgeschlossen sind, existieren die Blätter auf den Ebenen d bzw $d - 1$.
Wie viele Blätter liegen auf der Ebene $d - 1$ bzw d?
Beginnen wir mit einem vollständigen binären Baum der Tiefe $d = \lceil \log_2 N \rceil$ mit $2^d = 2^{\lceil \log_2 N \rceil}$ Blättern. Unser Baum hat aber nur N Blätter, also streichen wir um auf N zu kommen $2^{\lceil \log_2 N \rceil} - N$ *Geschwisterpaare* auf Ebene d.

$$\begin{aligned} 2^{\lceil \log_2 N \rceil} - \left(2^{\lceil \log_2 N \rceil} - N\right) &= N \\ \text{Anzahl Blätter im vollständigen binären Baum } - x &= \text{Benötigte Anzahl an Blättern} \end{aligned}$$

Daraus folgt sofort, daß auf der Ebene $d - 1 = \lceil \log_2 N \rceil - 1$ genau $2^{\lceil \log_2 N \rceil} - N$ Blätter liegen. Für die Ebene $d = \lceil \log_2 N \rceil$ ergibt sich die Anzahl der Blätter zu

$$\begin{aligned} x &= N - \left(2^{\lceil \log_2 N \rceil} - N\right) \\ \text{Anzahl Blätter Ebene } d &= \text{Anzahl Blätter im Entscheidungsbaum} - \text{ Anzahl Blätter Ebene } d - 1 \\ &= 2 \cdot N - 2^{\lceil \log_2 N \rceil} \end{aligned}$$

Für die durchschnittliche Tiefe erhält man

$$\begin{aligned} \text{Tiefe } - \frac{\text{Blätter Ebene } d - 1}{\text{Alle Blätter}} & \\ \lceil \log_2 N \rceil - \frac{1}{N} \cdot \left(2^{\lceil \log_2 N \rceil} - N\right) &= \lceil \log_2 N \rceil - \frac{2^{\lceil \log_2 N \rceil}}{N} + 1 \end{aligned}$$

2.6.3 Folgerung

Jedes allgemeine Sortierverfahren benötigt im worst case mindestens $\lceil \log_2 (n!) \rceil \approx n \log_2 n - 1,4427n$ Vergleiche und im average case mindestens $\lceil \log_2 (n!) \rceil - 1$ Vergleiche.

2.7 Batcher-Sort

Gegeben sei folgendes Array $a_1, \cdots, a_n$ mit $n = 2^k$.

15	47	33	87	98	17	53	76

BatcherSort $(15, 47, 33, 87, 98, 17, 53, 76)$

- BatcherSort$(15, 47, 33, 87)$
  - BatcherSort$(15, 47)$
    - BatcherSort(15) $\rightarrow$ (15)
    - BatcherSort(47) $\rightarrow$ (47)
    - BatcherMerge$(15; 47)$ $\rightarrow$ $(15, 47)$
  - $\leftarrow$ $(15, 47)$
  - BatcherSort$(33, 87)$
    - BatcherSort(33) $\rightarrow$ (33)
    - BatcherSort(87) $\rightarrow$ (87)

        * BatcherMerge$(33; 87) \quad \rightarrow \quad (33, 87)$
    - $\leftarrow \quad (33, 87)$
    - BatcherMerge$(15, 47; 33, 87)$
        * BatcherMerge$(15; 33) \quad \rightarrow \quad (15, 33)$
        * BatcherMerge$(47; 87) \quad \rightarrow \quad (47, 87)$
        * Vergleichen $\rightarrow \quad (15, 33, 47, 87)$
    - $\leftarrow \quad (15, 33, 47, 87)$
- $\leftarrow \quad (15, 33, 47, 87)$
- BatcherSort$(98, 17, 53, 76)$
    - BatcherSort$(98, 17)$
        * BatcherSort$(98) \quad \rightarrow \quad (98)$
        * BatcherSort$(17) \quad \rightarrow \quad (17)$
        * BatcherMerge$(98; 17) \quad \rightarrow \quad (17, 98)$
    - $\leftarrow \quad (17, 98)$
    - BatcherSort$(53, 76)$
        * BatcherSort$(53) \quad \rightarrow \quad (53)$
        * BatcherSort$(76) \quad \rightarrow \quad (76)$
        * BatcherMerge$(53; 76) \quad \rightarrow \quad (53, 76)$
    - $\leftarrow \quad (53, 76)$
    - BatcherMerge$(17, 98; 53, 76)$
        * BatcherMerge$(17; 53) \quad \rightarrow \quad (17, 53)$
        * BatcherMerge$(98; 76) \quad \rightarrow \quad (76, 89)$
        * Vergleichen $\rightarrow \quad (17, 53, 76, 89)$
    - $\leftarrow \quad (17, 53, 76, 89)$
- $\leftarrow \quad (17, 53, 76, 89)$
- BatcherMerge$(15, 33, 47, 87; 17, 53, 76, 89)$
    - BatcherMerge$(15, 47; 17, 76)$
        * BatcherMerge$(15; 17) \rightarrow \quad (15, 17)$
        * BatcherMerge$(47; 76) \rightarrow \quad (47, 76)$
        * Vergleichen $\rightarrow \quad (15, 17, 47, 76)$
    - $\leftarrow \quad (15, 17, 47, 76)$
    - BatcherMerge$(33, 87; 53, 89)$
        * BatcherMerge$(33; 53) \rightarrow \quad (33, 53)$
        * BatcherMerge$(87; 89) \rightarrow \quad (87, 89)$
        * Vergleichen $\rightarrow \quad (33, 53, 87, 89)$
    - $\leftarrow \quad (33, 53, 87, 89)$
    - Vergleichen $\rightarrow \quad (15, 17, 33, 47, 53, 76, 87, 89)$
- $\leftarrow \quad (15, 17, 33, 47, 53, 76, 87, 89)$ *Endergebnis*

2.7.0.1 Komplexitätsanalyse:

- BatcherMerge: (Annahme: $n = 2^k$)
  - Ansatz für die Anzahl der benötigten Vergleiche:

$$\begin{aligned} M(n) &= 2 \cdot M\left(\frac{n}{2}\right) + n - 1 \\ &= n \cdot \log_2 n + 1 \end{aligned}$$

  - Beweis:

$$\begin{aligned} M\left(2^k\right) &= 2 \cdot M\left(2^{k-1}\right) + 2^k - 1 \\ &= 2 \cdot \left(2 \cdot M\left(2^{k-2}\right) + 2^{k-1} - 1\right) + 2^k - 1 \\ &= 4 \cdot M\left(2^{k-2}\right) + 2^k - 2 + 2^k - 1 \\ &= 4 \cdot \left(2 \cdot M\left(2^{k-3}\right) + 2^{k-2} - 1\right) + 2^k - 2 + 2^k - 1 \\ &= 8 \cdot M\left(2^{k-3}\right) + 2^k - 4 + 2^k - 2 + 2^k - 1 \\ &\vdots \quad \vdots \\ &= 2^l \cdot M\left(2^{k-l}\right) + \sum_{i=1}^{l} 2^k - 2^{i-1} \end{aligned}$$

    Mit $l = k$ und der speziellen Lösung $M(1) = 1$ ergibt sich:

$$\begin{aligned} M\left(2^k\right) &= 2^k \cdot M\left(2^0\right) + \sum_{i=1}^{k} 2^k - 2^{i-1} \\ &= 2^k \cdot M(1) + \sum_{i=1}^{k} 2^k - \sum_{i=1}^{k} 2^{i-1} \\ &= 2^k + k \cdot 2^k - \sum_{i=0}^{k-1} 2^i \\ &= 2^k + k \cdot 2^k - \left(2^k - 1\right) \\ &= k \cdot 2^k + 1 \end{aligned}$$

    Nun wird für 2^k wieder n bzw für k wieder $\log_2 n$ geschrieben:

$$M(n) = n \cdot \log_2 n + 1$$

  - Ansatz für die Anzahl der benötigte "parallele Zeit":

$$\begin{aligned} PM(n) &= PM\left(\frac{n}{2}\right) + 1 \\ &= \log_2 n + 1 \end{aligned}$$

  - Beweis:

$$\begin{aligned} PM\left(2^k\right) &= PM\left(2^{k-1}\right) + 1 \\ &= \left(PM\left(2^{k-2}\right) + 1\right) + 1 \\ &= PM\left(2^{k-2}\right) + 1 + 1 \\ &= \left(PM\left(2^{k-3}\right) + 1\right) + 1 + 1 \\ &= PM\left(2^{k-3}\right) + 1 + 1 + 1 \\ &\vdots \quad \vdots \\ &= PM\left(2^{k-l}\right) + l \end{aligned}$$

Mit $l = k$ und der speziellen Lösung $PM(1) = 1$ ergibt sich:

$$\begin{aligned} PM\left(2^k\right) &= PM\left(2^0\right) + k \\ &= PM(1) + k \\ &= 1 + k \end{aligned}$$

Nun wird für 2^k wieder n bzw für k wieder $\log_2 n$ geschrieben:

$$PM(n) = \log_2 n + 1$$

- BatcherSort: (Annahme: $n = 2^k$)
  - Ansatz für die Anzahl der benötigten Vergleiche:

$$\begin{aligned} S(n) &= 2 \cdot S\left(\frac{n}{2}\right) + M\left(\frac{n}{2}\right) \\ &= 2 \cdot S\left(\frac{n}{2}\right) + \frac{n}{2} \cdot \log_2 \frac{n}{2} + 1 \end{aligned}$$

  - Ansatz für die Anzahl der benötigte "parallele Zeit":

$$\begin{aligned} PS(n) &= PS\left(\frac{n}{2}\right) + PM\left(\frac{n}{2}\right) \\ &= PS\left(\frac{n}{2}\right) + \log_2 n \end{aligned}$$

2.8 BucketSort

Gegeben sei folgendes Array

fuenf	frei	ferne	fremd	fluss	felgen	floh	folker

- Führe ein BucketSort bzgl. der Wortlängen durch:

4	5	6
frei	fluss	felgen
floh	fremd	folker
	ferne	
	fuenf	

- Initialisierung: Erzeuge ein Array mit den benötigten Queues

d	e	f	g	h	i	k	l	m	n	o	r	s	u

1. Schritt: Sortiere die Wörter mit mindestens sechs Buchstaben nach a_6 ein:

d	e	f	g	h	i	k	l	m	n	o	r	s	u
									felgen		folker		

   Hänge die Queues hintereinander:

felgen	folker

2. Schritt:

   Hänge die Liste der Wörter mit fünf Buchstaben vor das Ergebnis der letzten Runde:

fuenf	ferne	fremd	fluss	felgen	folker

   Sortiere die Wörter mit mindestens fünf Buchstaben nach a_5 ein:

d	e	f	g	h	i	k	l	m	n	o	r	s	u
	folker												
fremd	felgen	fuenf										fluss	
	ferne												

   Hänge die Queues hintereinander:

fremd	ferne	felgen	folker	fuenf	fluss

3. Schritt:

   Hänge die Liste der Wörter mit fünf Buchstaben vor das Ergebnis der letzten Runde:

floh	frei	fremd	ferne	felgen	folker	fuenf	fluss

   Sortiere die Wörter mit mindestens vier Buchstaben nach a_4 ein:

d	e	f	g	h	i	k	l	m	n	o	r	s	u
			felgen	floh	frei	folker		fremd	fuenf			fluss	
									ferne				

   Hänge die Queues hintereinander:

felgen	floh	frei	folker	fremd	ferne	fuenf	fluss

4. Schritt:

   Sortiere die Wörter mit mindestens drei Buchstaben nach a_3 ein:

d	e	f	g	h	i	k	l	m	n	o	r	s	u
	fuenf						folker			floh	ferne		fluss
	fremd						felgen						
	frei												

   Hänge die Queues hintereinander:

frei	fremd	fuenf	felgen	folker	floh	ferne	fluss

5. Schritt:

   Sortiere die Wörter mit mindestens zwei Buchstaben nach a_2 ein:

d	e	f	g	h	i	k	l	m	n	o	r	s	u
	ferne						fluss			folker	fremd		fuenf
	felgen						floh				frei		

   Hänge die Queues hintereinander:

felgen	ferne	floh	fluss	folker	frei	fremd	fuenf

6. Schritt:

   Sortiere die Wörter mit mindestens einem Buchstaben nach a_1 ein, hier fällt alles unter den Buchstaben f:

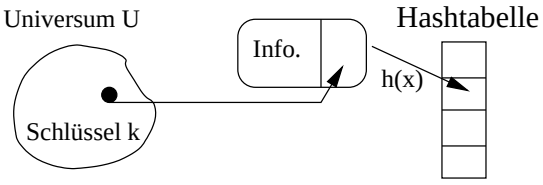
f	felgen	ferne	floh	fluss	folker	frei	fremd	fuenf

3 Dynamische Dateien

3.1 Hashing

3.1.1 Allgemeines

Beim Hashing nehmen wir an, das ein Datum aus der eigentlichen Information und einem Schlüssel k besteht, welcher aus einem Universum U stammt. Zwei Daten mit identischen Schlüsseln schließen wir aus.



Hashfunktion Die Hashfunktion$h$ ist eine Abbildung $h: \quad U \quad \rightarrow \quad \{0, \cdots, M-1\}$. Aus dem Schlüssel wird der Zielort in der Hashtabelle berechnet.

Eigenschaften

- einfache Auswertung, am besten $O(1)$. Die Berechnung des Funktionswertes kostet schließlich Zeit
- gute Streuung
- Surjektivität

3.1.2 Geschlossenes Hashing

Man kann sich vorstellen, dass die Hashtabelle ein Array fester Größe ist. Jedes Feld darin kann nur eine begrenzte Anzahl an Daten aufnehmen (*meistens eins, manchmal auch mehr*). Sollten zwei Daten x und y mit $x \neq y$ und $h(x) = h(y)$ abgespeichert werden, so kommt es zu einer *Kollision*. Um die Abspeicherung zu dennoch zu ermöglichen, gibt es verschiedene Strategien:

3.1.2.1 Lineares Sondieren

Prinzip: Von der ursprünglichen Adresse aus werden alle folgenden Plätze der Reihe nach überprüft.

Dazu verwaltet man für jeden Speicherplatz zwei boolsche Variablen:

- *empty* ist 1 gdw. der Platz frei ist
- *deleted* ist 1 gdw. der Platz bereits einmal belegt war und das Datum gelöscht wurde

Operationen:

- INSERT(x):
  Benutze die (Re-)Hashfunktion:

  $$h(x,j) = (h(x) + j) \mod m \qquad 0 \leq j \leq m-1$$

  bis ein freier Platz gefunden wurde.
- SEARCH(x): Berechne zunächst $h(x)$, falls x dort steht, so war die Suche erfolgreich. Ansonsten durchlaufe die Speicherplätze $(h(x) + j) \mod m$, $0 \leq j \leq m-1$ bis x gefunden und beende erfolgreich die Suche. Erreicht man dabei einen Speicherplatz mit $empty = 1$ und $deleted = 0$ wird die Suche erfolglos beendet.
- DELETE(x):(nach erfolgreicher Suche)
  Setzte die Werte für *empty* und *deleted* des entsprechenden Speicherplatzes auf 1.

Nachteil: Lineares Sondieren neigt zur Klumpenbildung[7]. Das bedeutet, daß lange kontinuierliche Teilstücke wesentlich schneller wachsen als kurze, da ein Schlüssel, sobald er einmal auf ein solches langes Stück trifft, an dessen Ende rutscht und dieses verlängert. Diese langen Stücke machen das Sondieren ineffizient.

[7] Primary clustering

3.1.2.2 Quadratisches Sondieren Quadratisches Sondieren soll die Klumpenbildung des lineares Sondierens vermindern. Benutze dazu folgende Hashfunktion

$$h(x,j) := \left(h(x) + j^2\right) \mod m \text{ für } j = 0, \cdots m-1$$

Man durchläuft eigentlich $h(x)$, $h(x)+1^2$, $h(x)-1^2$, $\cdots$, $h(x)+i^2\, h(x)-i^2$, $\cdots$. Falls m eine Primzahl ist, kann man zeigen, dass $i^2 \mod m$ für $i = 0, 1, \cdots, \lfloor \frac{m}{2} \rfloor$ alle verschieden sind. Weiterhin gilt:

$$\begin{aligned} h(x, 2j-1) &= \left(h(x) + j^2\right) \mod m \\ h(x, 2j) &= \left(h(x) - j^2\right) \mod m \quad 0 \leq j \leq \frac{m-1}{2} \end{aligned}$$

Für das quadratische Sondieren ist m prim mit $m \mod 4 = 3$ eine besonders gute Wahl, was sich über die Zahlentheorie zeigen lässt.

Quadratisches Sondieren kann keine Primärkollisionen $h(x) = h(y)$ verhindern, da $h(x,0) = h(y,0)$ gilt. Ebenso werden Schlüssel mit gleichem Hashwert auch auf die gleiche Sondierungsbahn gebracht[8].

3.1.2.3 Add to hash Benutze für diese Strategie folgende Hashfunktion

$$h(x) := (i \cdot h(x)) \mod m \text{ für } i = 1, \cdots m$$

Von den Eigenschaften her ähnelt es dem quadratischen Hashing. Man kann zeigen, dass innerhalb der ersten m Versuche alle Speicherplätze erreicht werden.

3.1.2.4 Doppeltes Hashing Hierfür werden zwei voneinander unabhängige Hash-Funktionen h und h' gewählt, z.B. $h(x) \equiv x \mod p$ und $h'(x) \equiv x \mod q$ für zwei Primzahlen p, q, die sehr viel größer als M, aber viel kleiner als $|U|$ sind. Die zusammengesetzte Hashfunktion hat folgendes Aussehen

$$h(x,i) = \left(h(x) + h'(x) \cdot i^2\right) \mod m$$

Das Verhalten der Funktion kommt dem idealen Hashing sehr nah.

3.1.3 Offenes Hashing

Das Datum x wird in jedem Fall an der Stelle $h(x)$ abgespeichert. Im Gegensatz zum geschlossenen Hashing enthält nun jedes Feld der Hashtabelle eine dynamisch vergrößerbare Datenstruktur, z.B. eine verkettete Liste, was zu einem Mehr an Speicherplatzverbrauch durch die insgesamt $M+n$ Zeiger (für jedes der M Tabellenfelder einen Zeiger auf das erste Element sowie für n eingefügte Elemente je einen Zeiger auf den Nachfolger) führt. Eine Auslastung mit mehr als M Daten ist möglich.

Operationen

- SEARCH(x):

  Berechne zuerst $h(x)$. Durchlaufe anschließend die Liste L an der Stelle $h(x)$, bis das Element x oder das Listenende erreicht wird. Die Kosten für das erfolglose Suche steigen proportional zur Listenlänge $\rightarrow O(l)$.

- INSERT(x) (nach erfolgloser Suche):

  Füge x vorne in die Liste L an der Stelle $h(x)$ ein. Die Kosten für das Einfügen am Listenkopf sind konstant$\rightarrow O(1)$.

- DELETE(x) (nach erfolgreicher Suche):

  Bei der erfolgreichen Suche merkt man sich den Zeiger auf x und kann dann in $O(1)$ löschen.

Offensichtlich sind die Kosten für die Operationen abhängig von der Listenlänge. Die erwartete Listenlänge für n Daten verteilt auf M Listen ergibt sich zu $\frac{n}{M}$.

- Bei einer erfolglosen Suche innerhalb einer Liste werden alle Zeiger angesprochen. Bei $\frac{n}{M}$ erwarteten Elementen und einem Zeiger auf das erste Element (s.o.) sind das $\frac{n}{M} + 1$ Zeiger.
  Die Suchzeit beträgt $\frac{n}{M} + 1$, da die ganze Liste durchlaufen wird.

[8] Secondary Clustering

- Bei einer erfolgreichen Suche enthält die Liste mit Sicherheit das gesuchte Objekt, es bleiben $\frac{n-1}{M}$Plätze für andere Objekte. Damit werden im Durchschnitt $1 + \frac{n-1}{M}$ Zeiger besucht.
  Bei einer Listenlänge von l ist die Position des gesuchten Objekts mit Wahrscheinlichkeit $\frac{1}{l}$ die Position l, was zum Erwartungswert $\frac{l+1}{2}$ führt, da

$$\begin{aligned} \text{Erwartungswert } \sum_{i=1}^{l} \frac{1}{l} \cdot i &= \frac{1}{l} \cdot \frac{l \cdot (l+1)}{2} \\ &= \frac{l+1}{2} \end{aligned}$$

  gilt.
  Im erfolgreichen Fall ist die Listenlänge $l = 1 + \frac{n-1}{M}$, woraus die erwartete Suchzeit $1 + \frac{n-1}{2M}$ folgt, da

$$\begin{aligned} \frac{l+1}{2} &= \frac{\left(1 + \frac{n-1}{M}\right) + 1}{2} \\ &= \frac{2 + \frac{n-1}{M}}{2} \\ &= 1 + \frac{n-1}{2 \cdot M} \end{aligned}$$

  gilt.

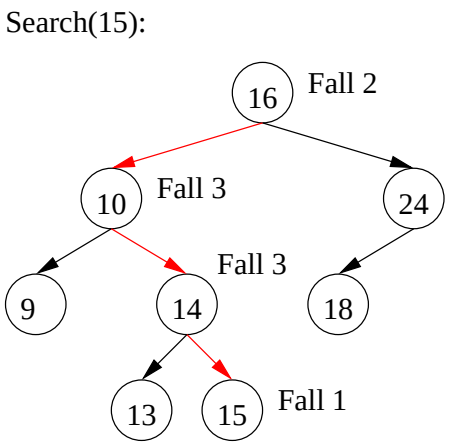
3.2 Binäre Suchbäume

Definition Ein binärer Baum, in dessen Knoten Daten aus einem vollständig geordneten[9] Universum gespeichert sind, heißt Suchbaum, wenn für jeden Knoten v mit seinem Inhalt x gilt

- alle im linken Teilbaum gespeicherten Daten sind kleiner als x
- alle im rechten Teilbaum gespeicherten Daten sind größer als x

Operationen

- SEARCH(x): Starte an der Wurzel. Erreicht man einen Knoten v, so wird dessen Inhalt y mit dem Datum x verglichen.
  1. Fall: $x = y$. Die Suche war erfolgreich.
  2. Fall: $x < y$. Suche im linken Teilbaum von v weiter. Falls dieser nicht existiert, so war die Suche erfolglos.
  3. Fall: $x > y$. Suche im rechten Teilbaum von v weiter. Falls dieser nicht existiert, so war die Suche erfolglos.

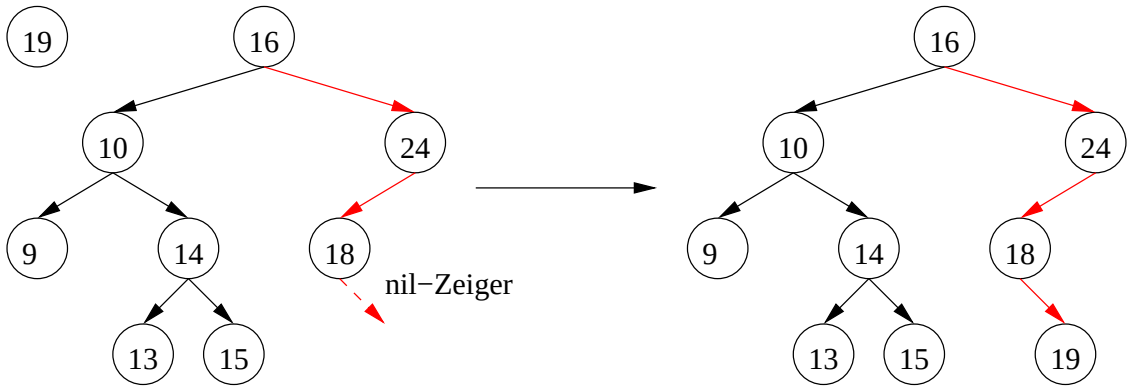


[9] Je zwei Elemente sind miteinander vergleichbar

- INSERT(x): (nach erfolgloser Suche)

  Der letzte Schritt der Suche führte uns zu einem nil-Zeiger eines Blattes v . Dieser Zeiger wird auf den Knoten mit Inhalt x umgebogen.

Insert(19):

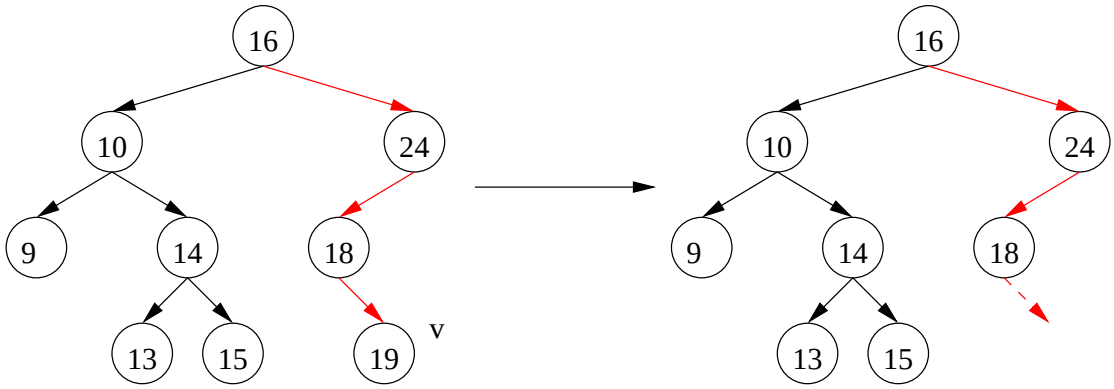


- DELETE(x): (nach erfolgreicher Suche)

  Wir kennen den Zeiger auf den Knoten v, der das zu löschende Datum x enthält.

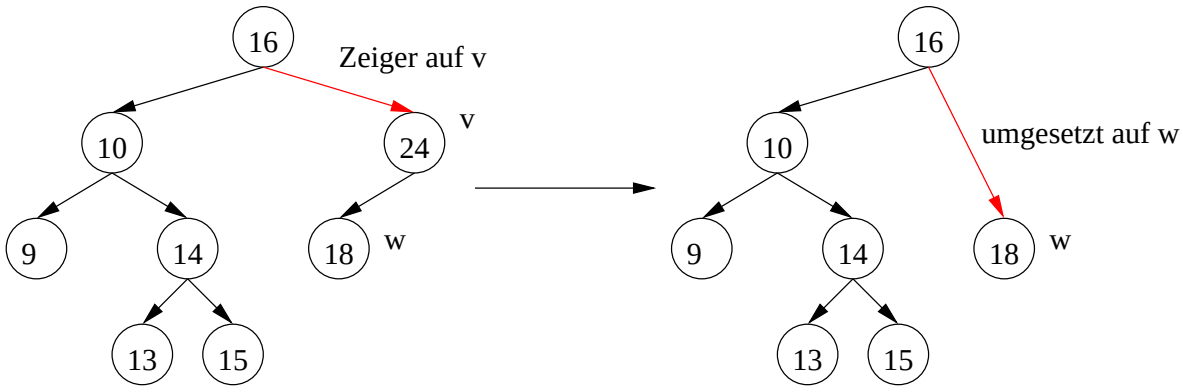
  1. Fall: v ist ein Blatt. Setze den Zeiger auf den Knoten v um auf nil. Gib den Knoten v frei.[10]

Delete(19):



  2. Fall: v hat ein Kind w . Setze den Zeiger auf v auf sein Kind w um. Gib den Knoten v frei.

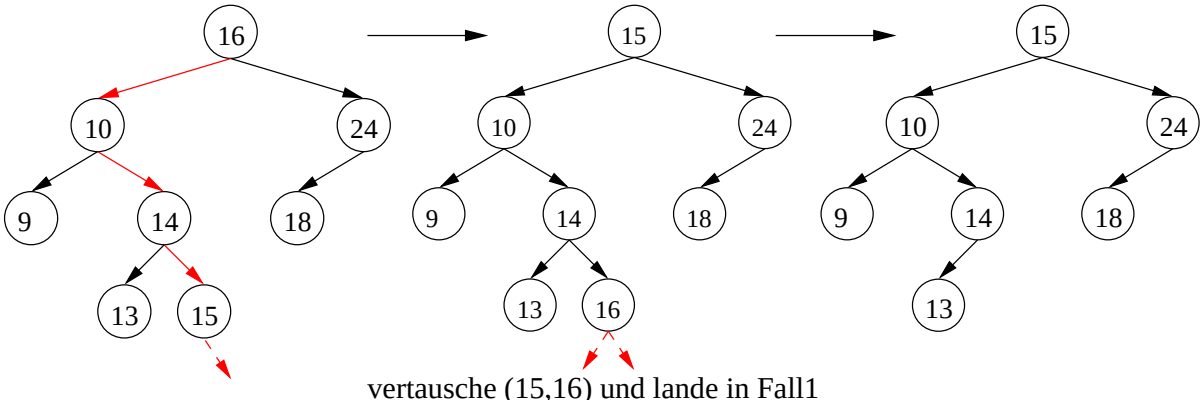
Delete(24):



[10] Falls keine Garbage Collection existiert vgl. Java vs C++

3. Fall: v hat zwei Kinder. Dann finde das größte Datum y kleiner dem Inhalt x von v. Dazu gehe zum linken Kind von v, anschließend in den jeweils rechten Teilbaum, bis ein nil-Zeiger erreicht ist. In dem Knoten von dem dieser nil-Zeiger ausgeht, steht das größte Datum y kleiner x. Vertausche x und y. Der Knoten, in dem sich x nun befindet, hat ein linkes oder kein Kind und man ist in Fall 1 oder 2

Delete(16):



vertausche (15,16) und lande in Fall1

3.3 2-3-Bäume

Definition Ein Baum heißt 2-3-Baum, wenn die folgenden Eigenschaften erfüllt sind:

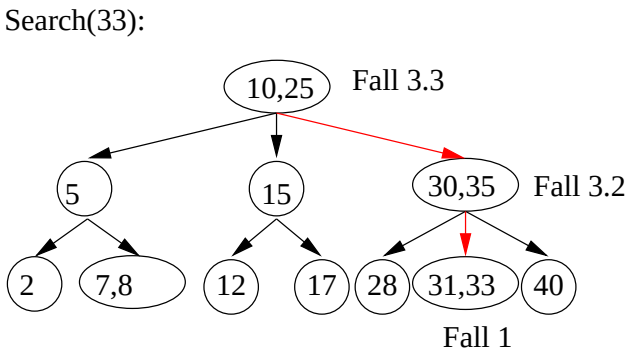
1. Jeder Knoten enthält ein oder zwei Daten.
2. Knoten mit einem Datum x haben zwei Zeiger (auf zwei Teilbäume):
   - alle im linken Teilbaum gespeicherten Daten sind kleiner als x
   - alle im rechten Teilbaum gespeicherten Daten sind größer als x
3. Knoten mit zwei Daten x und y mit $(x < y)$ haben drei Zeiger (auf drei Teilbäume):
   - alle im linken Teilbaum gespeicherten Daten sind kleiner als x
   - für alle im mittleren Teilbaum gespeicherten Daten z gilt: $x < z < y$
   - alle im rechten Teilbaum gespeicherten Daten sind größer als y
4. Die Zeiger, die einen Knoten verlassen, sind entweder alle nil-Zeiger oder alle keine nil-Zeiger. Im ersten Fall heißt der Knoten dann Blatt.
5. Alle Blätter des Baumes haben die gleiche Tiefe.

Operationen

- SEARCH(x):

  Beginne die Suche bei der Wurzel. Betrachte jeweils den erreichten Knoten v mit dem Datum y und dem eventuell vorhandenen zweiten Datum z mit $y < z$.

  1. Fall: Gilt $x = y$ oder $x = z$, so wird die Suche als erfolgreich beendet.
  2. Fall: Gilt $x < y$, so gehe in den ersten (linken) Teilbaum des Knotens v
  3. Fall: Gilt $x > y$ und
     - z existiert nicht, so gehe in den zweiten (rechten) Teilbaum des Knotens v
     - z existiert und $x < z$, also $y < x < z$, so gehe in den zweiten (mittleren) Teilbaum des Knotens v
     - z existiert und $x > z$, so gehe in den dritten (rechten) Teilbaum des Knotens v

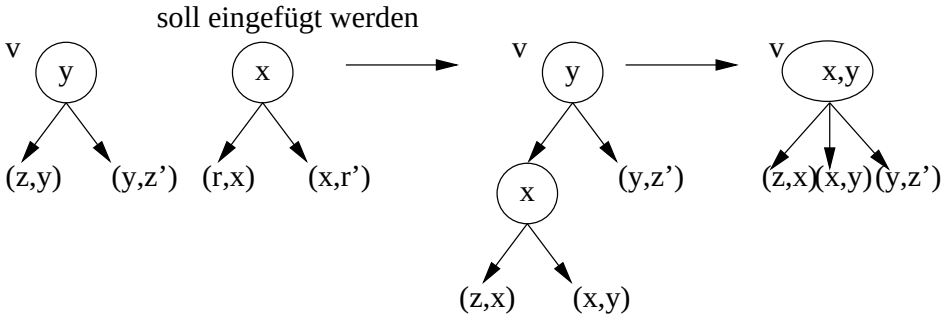


- INSERT(x): (nach erfolgloser Suche)

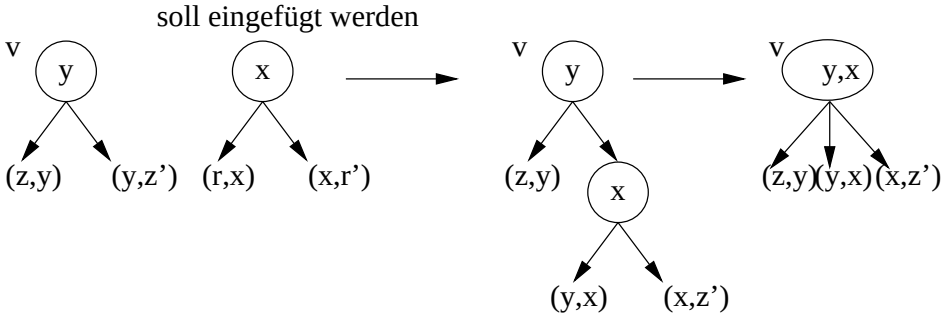
  Wir sind an dem nil-Zeiger eines Blattes gelandet, wo x eigentlich hingehört. Der Knoten, von dem der nil-Zeiger ausgeht, trage den Namen v. Es sind danach folgende Vorgehen möglich:

  – v enthält genau ein Datum y

  1. Fall: $x < y$. Wir stoßen auf den (z, y)-Zeiger. Diesen Suchbereich teilen wir in zwei neue ein, nämlich (z, x) und (x, y). Das Datum x wird vorne im Knoten v eingefügt. Der Knoten hat dann drei Zeiger: (z, x) , (x, y) und (y, z').

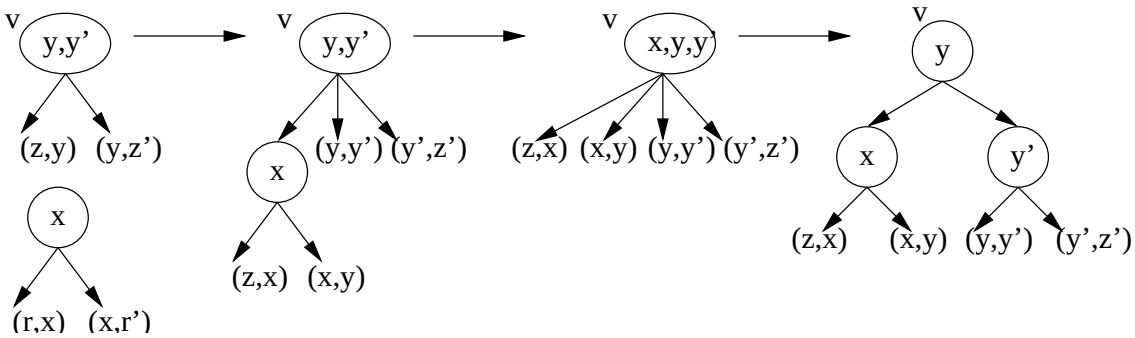


  2. Fall: $x > y$. Wir stoßen auf den (y, z')-Zeiger. Diesen Suchbereich teilen wir in zwei neue ein, nämlich (y, x) und (x, z'). Das Datum x wird hinten im Knoten v eingefügt. Der Knoten hat dann drei Zeiger: (z, y) , (y, x) und (x, z').

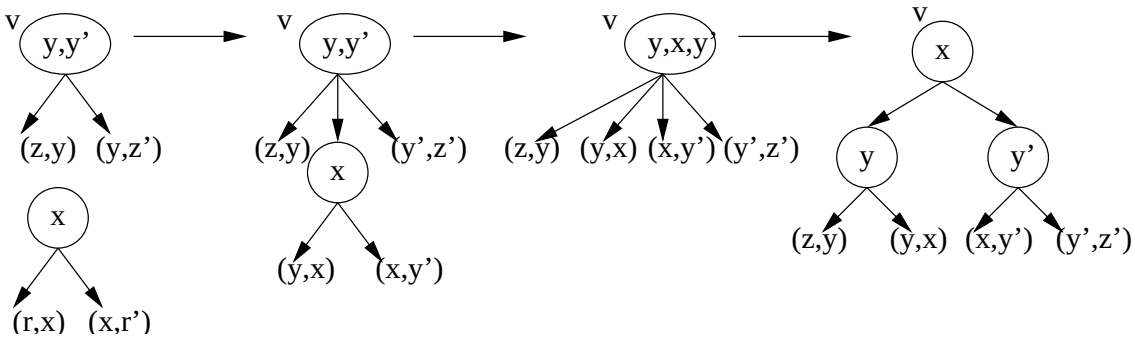


  – v enthält genau zwei Daten y und y' mit $y' > y$.
  Füge gedanklich das Datum x passend in den Knoten v ein, damit ergibt sich ein Knoten mit drei Daten und vier Zeigern. Aus diesem Knoten machen wir dann drei. *In jedem der folgenden Fälle müssen wir die Balancierung nach oben fortsetzen, bis wir in der Wurzel oder einmal im Fall "v enthält genau ein Datum y" landen.*

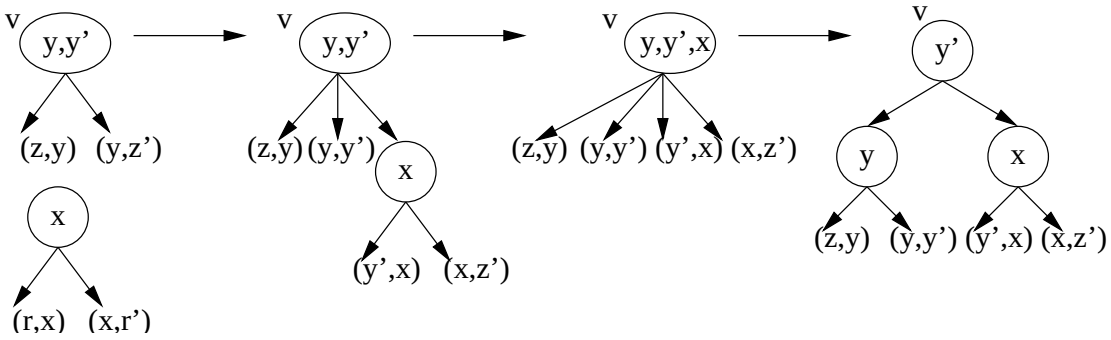
  1. Fall: $x < y$. Der Knoten mit dem mittleren Datum y zeigt auf die beiden anderen Knoten. Der mit dem kleineren Datum x erhält die ersten beiden Zeiger des geplatzten Knotens und der mit dem größeren Datum y' die restlichen beiden Zeiger. Der Zeiger auf den früheren Knoten v zeigt nun auf den Knoten mit Inhalt y.



2. Fall: $y < x < y'$.
   Der Knoten mit dem mittleren Datum x zeigt auf die beiden anderen Knoten . Der mit dem kleineren Datum y erhält die ersten beiden Zeiger des geplatzten Knotens und der mit dem größeren Datum y' die restlichen beiden Zeiger. Der Zeiger auf den früheren Knoten v zeigt nun auf den Knoten mit Inhalt x.



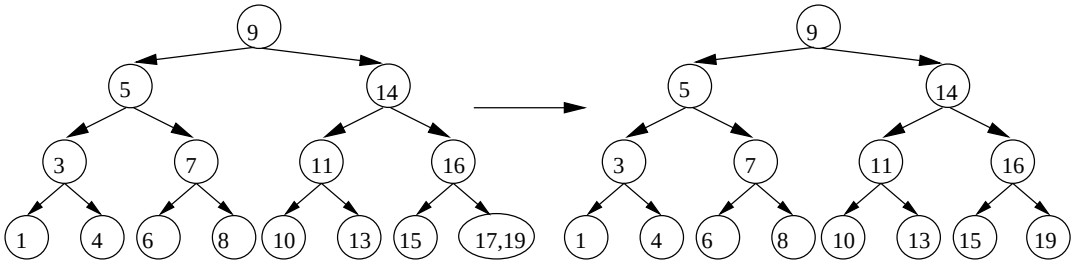
3. Fall: $x > y'$.
   Der Knoten mit dem mittleren Datum y' zeigt auf die beiden anderen Knoten . Der mit dem kleineren Datum y erhält die ersten beiden Zeiger des geplatzten Knotens und der mit dem größeren Datum x die restlichen beiden Zeiger. Der Zeiger auf den früheren Knoten v zeigt nun auf den Knoten mit Inhalt y'.



- DELETE(x): (nach erfolgreicher Suche)

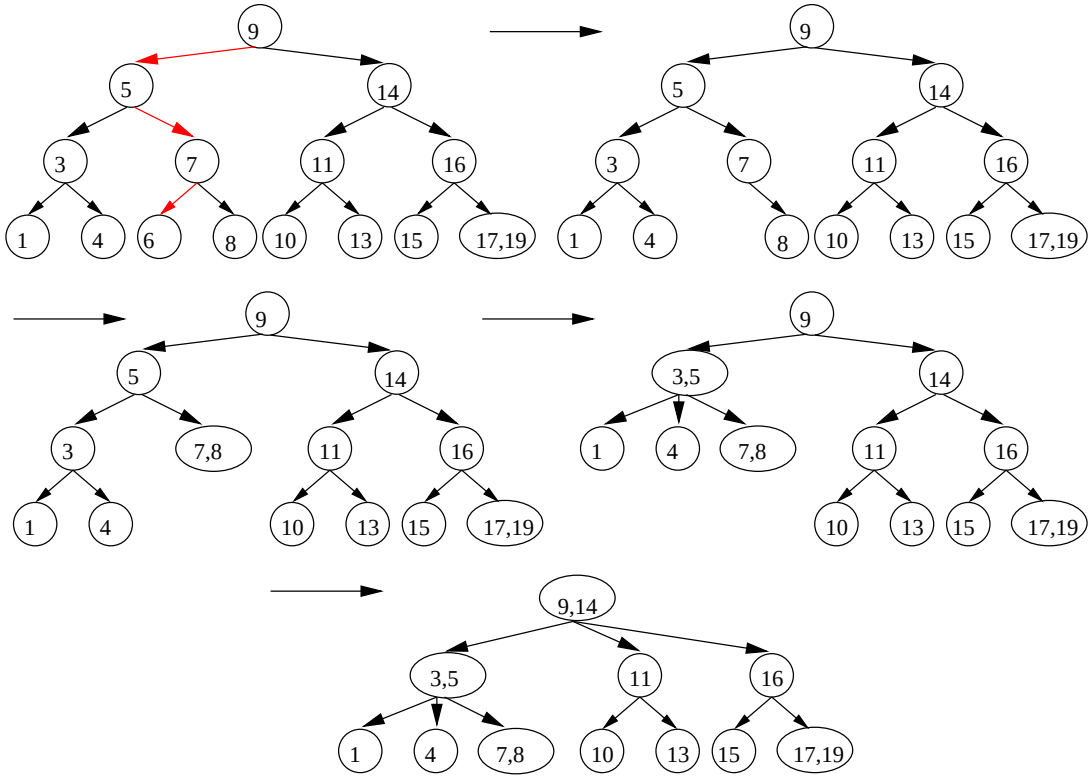
1. Fall: Enthält der Knoten (Blatt) mit dem Datum x noch ein weiteres Datum y, so werden x und ein nil-Zeiger gelöscht.

Delete(17)



2. Fall: War x das einzige Datum im Knoten (Blatt), so setzte den Zeiger auf diesen Knoten auf nil, gib den Knoten frei und mache eine Balancierung nach oben.

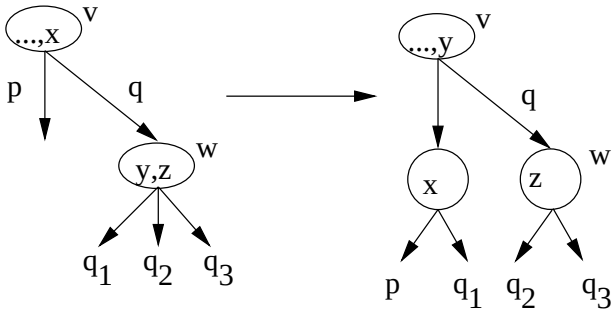
Delete(6)



3. Fall: Wenn x nicht in einem Blatt steht, so suche das größte Datum y mit $y < x$. Falls x einziges Datum im Knoten ist, so wähle den ersten Zeiger, sonst den zweiten. Danach immer den letzten, bis ein nil-Zeiger erreicht wird. Dann ist das Datum y ist das größte Element des zuletzt besuchten Knotens. Vertausche x und y. Nun muss das Datum aus einem Blatt gelöscht werden.

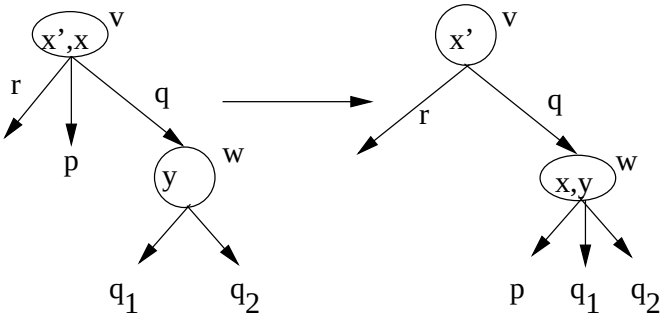
4. *Verallgemeinerung vom ersten und zweiten Fall*: Es gibt einen Zeiger p auf einen Teilbaum, dessen Blätter eine Ebene zu hoch liegen. Wir betrachten den Knoten v von dem p ausgeht. Den Knoten v verlässt mindestens noch ein Zeiger. Wähle einen zu p benachbarten Zeiger, dieser sei hier o.B.d.A rechts von q. Die Zeiger p und q seien durch das Datum x getrennt. Der Zeiger q zeigt auf den Knoten w.

   (a) w enthält zwei Daten y und z mit $y < z$. Mache eine Rotation. Danach ist wieder alles in Balance.

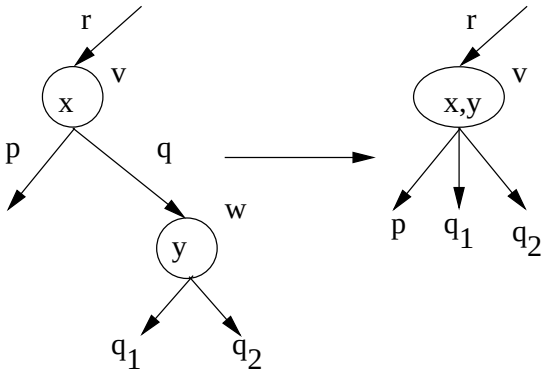


(b) w enthält ein Datum y und daher zwei Zeiger q_1 und q_2

i. Unterfall: v enthält neben x noch ein Datum x', o.B.d.A. $x' < x$. Drücke x und einen Zeiger in w hinein.Danach ist wieder alles in Balance.



ii. Unterfall: v enthält nur x. Ziehe y in v hinein. Balanciere weiter nach oben.

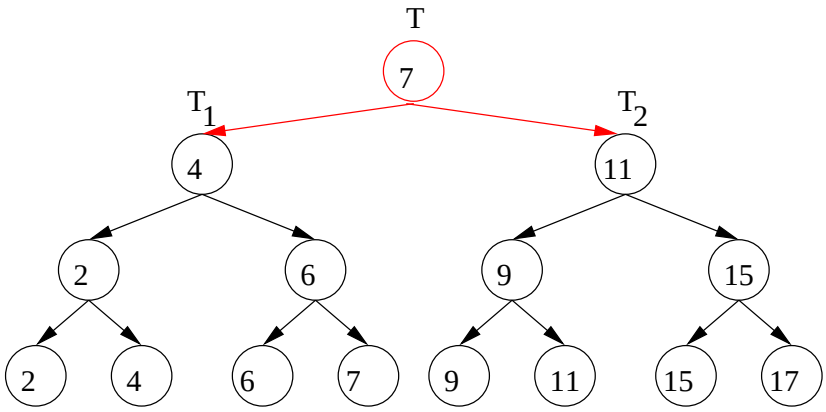


- CONCATENATE(T_1, T_2, T): (von blattorientierten[11] 2-3-Bäumen)

1. Fall: $d(T_1) = d(T_2)$

   - T besteht aus einer Wurzel, deren Datum das größte Datum aus T_1 ist.
   - T_1 und T_2 sind Teilbäume dieser neuen Wurzel.
   - Das größte Datum in T ist gleich dem größten Datum von T_2.

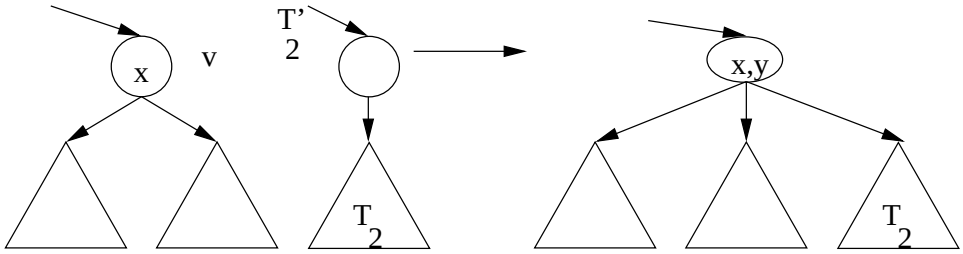
[11] Die Daten sind in den Blättern gespeichert. In den inneren Knoten ist jeweils das größte Datum des linken Teilbaums und falls vorhanden das größte Datum des mittleren Teilbaums abgespeichert.



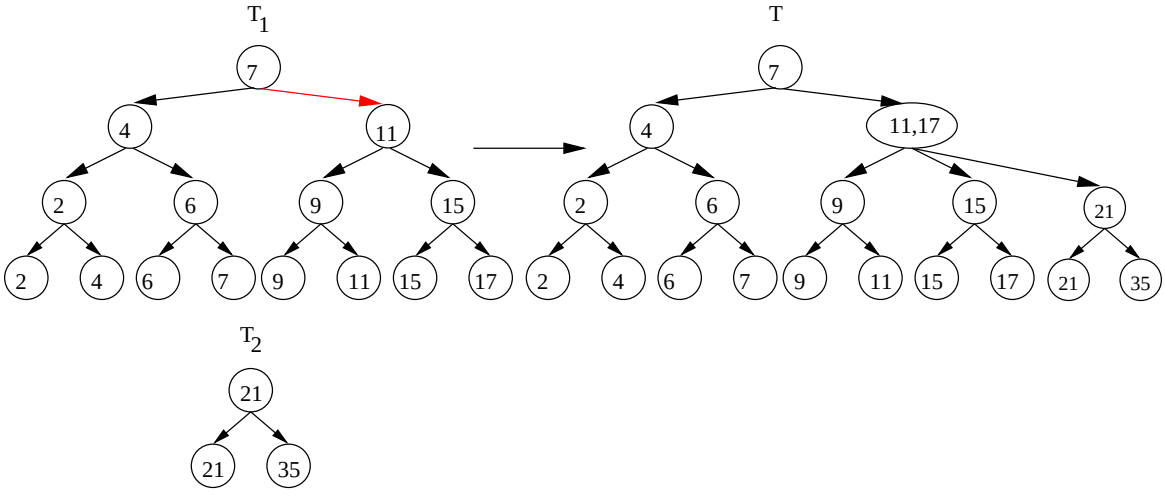
2. Fall: $d(T_1) > d(T_2)$ (der dritte Fall $d(T_1) < d(T_2)$ geht analog)

   Bilde einen Baum (nicht 2-3-Baum) T_2', der als Wurzel einen Knoten ohne Datum hat. Dieser Knoten zeigt auf die Wurzel von T_2. In T_1 gehen wir von der Wurzel aus $d(T_1) - d(T_2) - 1$ Schritte immer den rechtesten Zeiger entlang. Wir erreichen dann einen Knoten v.

   – Der Knoten v enthält nur ein Datum x und hat zwei Zeiger
   Dabei ist x die Information aus v und y das größte Datum aus T_1, welches wir an der Wurzel von T_1 finden[12]. Das größte Datum des Baumes T, dessen Wurzel die alte Wurzel von T_1 ist, ist gleich dem größten Datum aus T_2.

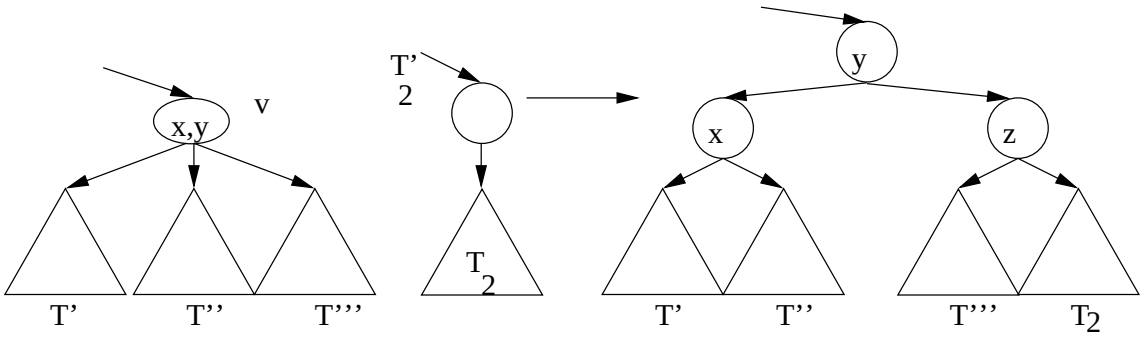


   Beispiel:

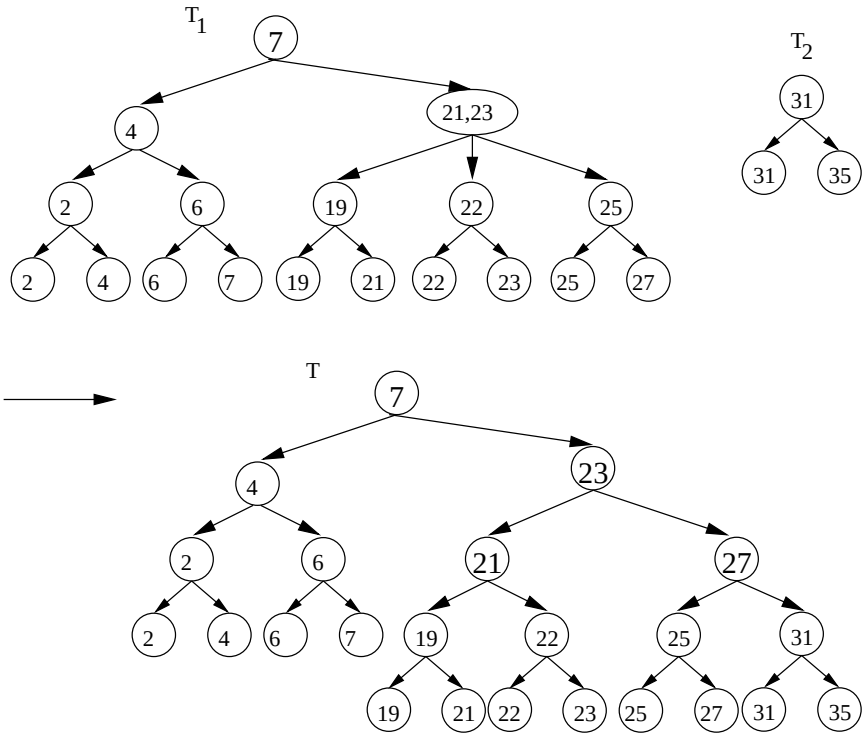


   – Der Knoten v enthält zwei Daten x, y und hat drei Zeiger
   Dabei ist z das größte Datum aus T_1. Der Teilbaum, dessen Wurzel nun y enthält hat die Eigenschaft, daß alle Blätter eine Ebene zu tief liegen. Also muss weiter nach oben balanciert werden (vgl. INSERT(x)).

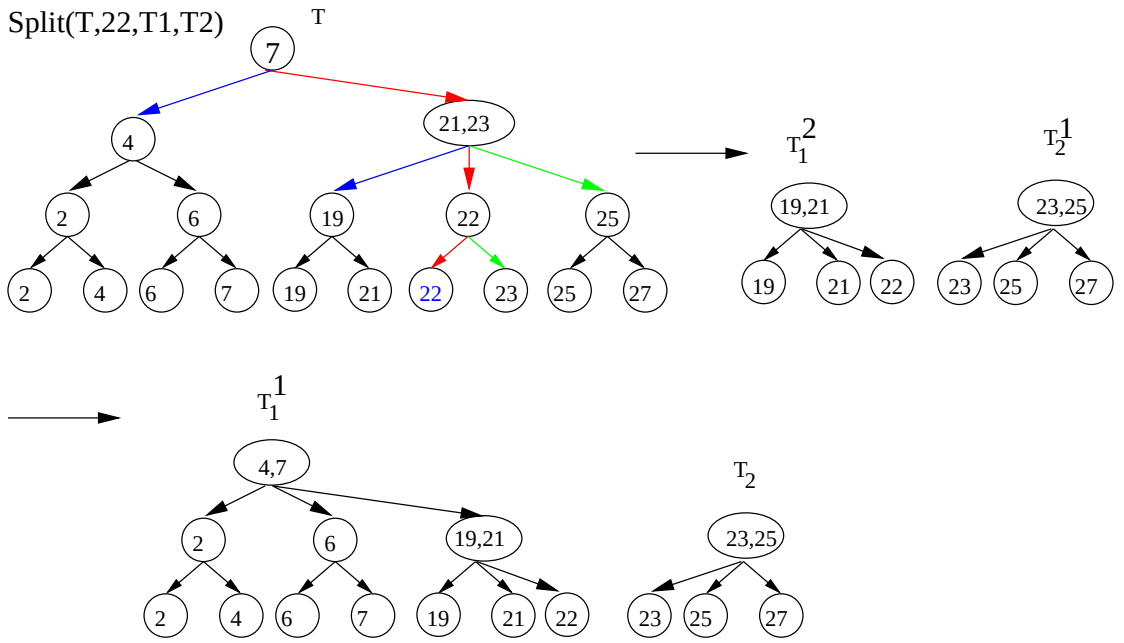
[12] Das größte Element eines Baumes wird extra verwaltet und an dessen Wurzel abgespeichert.



Beispiel:



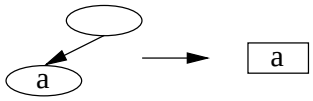
- SPLIT(T, a, T_1, T_2): Führe SEARCH(a) durch und finde a in einem Blatt wieder. Dabei wird gespeichert, was links und rechts vom Suchpfad liegt. Die Daten des Blattes, welches a enthält werden ebenfalls eingeteilt, wobei a als links vom Suchpfad definiert wird.
  Mit $T_1^1, \cdots, T_1^m$ bezeichnen wird die (Anzahl der linken bzw rechten) Teilbäume, aus denen wir T_1 zusammensetzen. Durch die Abspeicherung gilt $d\left(T_1^i\right) \geq d\left(T_1^{i+1}\right)$.
  Für $i = m-1, \cdots, 1$: Concatenate $\left(T_1^i, T_1^{i+1}, T_1^i\right)$. Schließlich gilt $T_1^1 = T_1$.
  Beispiel:



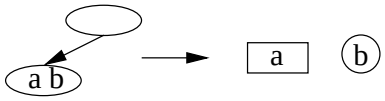
Die Korrektheit des Verfahrens sollte klar seien, nun reden wir über die Rechenzeit und zeigen ersteinmal folgende Aussage per vollständiger Induktion über die Tiefe d':
Die Konkatenation aller Bäume T_1^i mit $d\left(T_1^i\right) \leq d'$ ergibt einen 2-3-Baum , dessen Tiefe kleiner oder gleich $d'+1$ ist.

- Induktionsverankerung:
  $d' = 0$: Wir untersuchen Bäume, deren Tiefe ≤ 0 ist. Man kann dabei folgende Fälle erhalten:

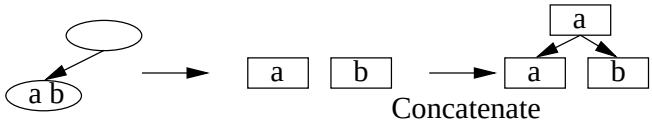
  1. Der Knoten, der das gesuchte Datum enthält, besitzt nur ein Datum a
     Dieser Knoten wird als linker Teilbaum abgespeichert (lauft Definition) und hat dann die Tiefe 0. Er ist der einzige in seiner Tiefe, die Konkatenation[13] mit einem leeren Baum bringt keinen Größenzuwachs. Abschließend beträgt die Tiefe somit $0 \leq 1 = 0 + 1 = d' + 1$.

  2. Der Knoten, der das gesuchte Datum enthält, besitzt zwei Daten a und b, o.B.d.A sei $a < b$.
     (a) Das gesuchte Datum ist a
     Per Definition wird ein neuer Knoten mit a ein linker Teilbaum und ein neuer Knoten, der b enthält, rechter Teilbaum. Im Moment betrachten wir nur die linken Teilbäume, d.h. den Knoten, der a enthält. Wir kommen zum gleichen Schluß wie oben, die Konkatenation bringt keinen Größenzuwachs, die Tiefe bleibt $0 \leq 1 = 0 + 1 = d' + 1$.

     (b) Das gesuchte Datum ist b
     Es werden nun zwei linke Teilbäume definiert, der eine enthält den Knoten mit Datum a, der andere Teilbaum den Knoten mit Datum b. Diese beiden Teilbäume werden durch *Concatenate* zusammengefügt. Es entsteht ein 2-3-Baum der Tiefe $1 \leq 1 = 0 + 1 = d' + 1$.

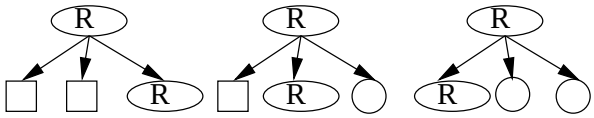
     

[13] wenn sie denn überhaupt ausgeführt werden muss

Die Verankerung ist damit abgeschlossen und gültig.

- Induktionsvoraussetzung:
  Die Induktionsbehauptung gelte für 2-3-Bäume bis zur Tiefe d'. Wir erhalten also aus der Konkatenation von Bäumen, deren Tiefe durch d' beschränkt ist, einen 2-3-Baum, deren Tiefe durch $d'+1$ beschränkt. ist.

- Induktionsschritt:
  $d' \Rightarrow d'+1$: In der T_1^i- Folge kann es nach Konstruktion nur zwei Bäume der Tiefe $d'+1$ geben (siehe Abbildung, R ist die Routing-Information)



  Wir nehmen an, dass drei 2-3-Bäume mit Tiefenbeschränkung $d'+1$ zusammengefügt werden. Zwei von ihnen, seien dies T_1 und T_2, haben im worst case genau die Tiefe $d'+1$, der dritte Baum T_3 ist im worst case minimal kleiner, muss also die Tiefe d' haben.

  - Zuerst fügen wir T_3 mit T_2 zusammen[14]:
    $d(T_3) = d'$ bzw. $d(T_2) = d'+1$. T_3 wird direkt unter der Wurzel von T_2 eingehangen. Dabei ergeben sich folgenden Fälle:

    1. Die Wurzel von T_2 hatte vor dem Einhängen ein Datum und zwei Zeiger
       Nach *Concatenate* besitzt die Wurzel nun zwei Daten und drei Zeiger. Die Größes des zusammengefügten Baumes ist gleich $d(T_2) = d'+1$.
    2. Die Wurzel von T_2 hatte vor dem Einhängen zwei Daten und drei Zeiger
       Nach *Concatenate* enthält die Wurzel kurzfristig drei Daten und vier Zeiger, sie platzt. Es entsteht eine neue Wurzel mit einem Datum und zwei Zeiger, die Tiefe des Baumes ist um 1 gewachsen und beträgt damit $d(T_2)+1 = d'+2$.

  - Danach fügen wir den neu erhaltenen Baum T' mit T_1 zusammen. Dabei betrachten wir die beiden obigen Fälle einzeln:

    1. $d(T') = d'+1 = d(T_1)$. Wir erzeugen eine neue Wurzel und hängen beide Teilbäume darunter ein. Die Größe des entstandenen 2-3-Baumes beträgt $d'+2$.
    2. $d(T') = d'+2$ bzw. $d(T_1) = d'+1$. T_1 wird also unter die Wurzel von T' eingehangen. Wir wissen, dass die Wurzel von T' höchstens ein Datum enthalten kann. Nach dem *Concatenate* enthält die Wurzel des neuen 2-3-Baumes zwei Daten und drei Zeigen, der Baum besitzt aber die gleiche Größe wie T', nämlich $d'+2$.

  Wir wissen jetzt, dass beim Zusammenfügen von drei Teilbäumen, deren Tiefe durch $d'+1$ beschränkt ist, ein neuer 2-3-Baum entsteht, dessen Tiefe durch $d'+2$ beschränkt ist. Weiterhin fügen wir maximal zwei Teilbäume deren, Tiefe durch $d'+1$ beschränkt ist zusammen, d.h. die Schranke $d'+2$ gilt auch hier.

Insgesamt führt man $m-1 \leq 2 \cdot d(T) - 1$[15] Konkatenationen aus.
Jede der Konkatenationen verursacht Kosten von $O((d_{i+1}-d_i)+1)$, wobei d_{i+1} und d_i Tiefen der zu konkatenieren Teilbäume sind.

Die 1 innerhalb des Terms $O((d_{i+1}-d_i)+1)$ wird nach $2 \cdot d(T)-1$ Konkatenationen den Wert $2 \cdot d(T)-1$ angenommen haben, also $\in O(d(T))$ seien. $(d_{i+1}-d_i)$ wird mit $(d_k - d_1)$, einer Konstante angegeben, ist also $\in O(d(T))$.

Satz: Die Prozedur Split kann für einen 2-3-Baum T in Zeit $O(d(T))$ durchgeführt werden.

[14] Die Reihenfolge des Concatenate ergibt sich aus der Nummerierung der linken bzw. rechten Teilbäume. $\text{num}(T_3) > \text{num}(T_2)$
[15] Auf jeder der m Ebenen liegen maximal 2 linke Teilbäume $\Rightarrow 2 \cdot d(T)$

3.4 Bayer-Bäume

Definition Ein Baum heißt B-Baum der Ordnung m, wenn die folgenden Eigenschaften erfüllt sind:

1. Jeder Knoten mit Ausnahme der Wurzel enthält mindestens $\lceil \frac{m}{2} \rceil - 1$ Daten. Die maximale Anzahl an Daten in jedem Knoten ist $m-1$. Die Daten innerhalb des Knotens sind sortiert.
2. Knoten mit k Daten $x_1, \cdots, x_k$ haben $k+1$ Zeiger, die auf die Bereiche $(\cdot, x_1)$, (x_1, x_2), $\cdots$, (x_{k-1}, x_k), $(x_k, \cdot)$ zeigen.
3. Die Zeiger, die einen Knoten verlassen, sind entweder alle nil-Zeiger (der Knoten heißt dann Blatt) oder alle echte Zeiger.
4. Alle Blätter haben die gleiche Tiefe.

Bemerkung Die Tiefe eines Bayer-Baumes der Ordnung m mit n Daten liegt im Intervall $\left[\log_m (n+1) - 1, \log_{\lceil \frac{m}{2} \rceil} \left(\frac{n+1}{2}\right)\right]$.

Beweis:

- untere Schranke:

$$\begin{aligned} m^{d+1} - 1 + &\geq n \\ m^{d+1} &\geq n+1 \\ d+1 &\geq \log_m (n+1) \\ d &\geq \log_m (n+1) - 1 \end{aligned}$$

- obere Schranke:

  In jedem Knoten befindet sich die minimale Anzahl an Daten, d.h. in der Wurzel ein Datum, in jedem anderen Knoten $\lceil \frac{m}{2} \rceil - 1$ Daten. Wie betrachten nun die Anzahl der Knoten auf den ersten Ebenen, um dann die Formel *ausführlich* herzuleiten:

Ebene	Anzahl Knoten
0	1
1	2
2	$2 \lceil \frac{m}{2} \rceil$
3	$2 \lceil \frac{m}{2} \rceil^2$
4	$2 \lceil \frac{m}{2} \rceil^3$

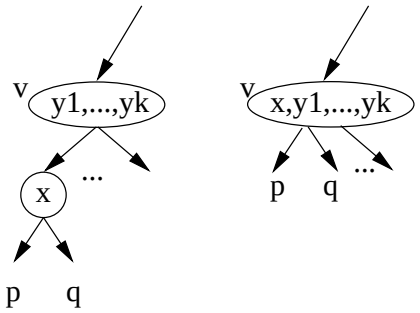
$$\underbrace{1}_{\text{Wurzel}} + \overbrace{\left(\left\lceil \frac{m}{2} \right\rceil - 1\right)}^{\text{Anzahl Daten in einem Knoten}} \cdot \underbrace{\sum_{i=1}^{d} \left(2 \cdot \left\lceil \frac{m}{2} \right\rceil^{i-1}\right)}_{\text{Gesamtanzahl an Knoten}} \leq n$$

$$\begin{aligned} 1 + 2 \cdot \left(\left\lceil \frac{m}{2} \right\rceil - 1\right) \cdot \sum_{i=1}^{d} \left(\left\lceil \frac{m}{2} \right\rceil^{i-1}\right) &\leq n \\ 1 + 2 \cdot \left(\left\lceil \frac{m}{2} \right\rceil - 1\right) \cdot \frac{\lceil \frac{m}{2} \rceil^d - 1}{\lceil \frac{m}{2} \rceil - 1} &\leq n \\ 1 + 2 \cdot \left(\left\lceil \frac{m}{2} \right\rceil^d - 1\right) &\leq n \\ 2 \cdot \left\lceil \frac{m}{2} \right\rceil^d - 1 &\leq n \\ \left\lceil \frac{m}{2} \right\rceil^d &\leq \frac{n+1}{2} \\ d &\leq \log_{\lceil \frac{m}{2} \rceil} \left(\frac{n+1}{2}\right) \end{aligned}$$

Operationen

- INSERT(x): (nach erfolgloser Suche)
  Wir sind nach der Suche an einem nil-Zeiger gelandet, dessen Elter der Knoten v sei. An diesem Zeiger hängen wir unseren neuen Knoten mit Datum x ein. Es werden folgende Fälle unterschieden:

  1. Fall: v enthält weniger als $m-1$ Daten. Ziehe x mitsamt den zugehörigen Zeigern in den Knoten v mit hinein.



  2. Fall: v enthält genau als $m-1$ Daten. Dann ziehe x mitsamt den zwei zugehörigen Zeigern mit in v. Der Knoten platzt, mache aus ihm drei Knoten. Nimm das mittlere der m Elemente $z_{\lceil \frac{m}{2} \rceil}$ als neue Wurzel, im linken Sohn stehen die Daten $z_1, \cdots, z_{\lceil \frac{m}{2} \rceil - 1}$, im rechten Sohn $z_{\lceil \frac{m}{2} \rceil + 1}, \cdots, z_m$. Gehe auf dem Suchpfad einen Schritt zurück und fahre analog fort.

3.5 AVL-Bäume

Diese Bäume sind nach ihren "Entdeckern" Adelson, Velskij und Landis benannt.

Definition Ein AVL-Baum ist ein binärer Suchbaum mit einer Strukturinvarianten. Für jeden Knoten gilt, dass sich die Höhen seiner beiden Teilbäume um höchstens eins unterscheiden.

Dazu definiert man den Balancegrad eines Knotens $b(v)$ durch:

$$\begin{aligned} b(v) = 1 &\Leftrightarrow d(T_L) = d(T_R) + 1 \\ b(v) = 0 &\Leftrightarrow d(T_L) = d(T_L) \\ b(v) = -1 &\Leftrightarrow d(T_L) = d(T_R) - 1 \end{aligned}$$

- Höhe eines AVL-Baumes
  - untere Schranke:
    Ergibt sich analog zu der unteren Schranke für binäre Bäume

$$\begin{aligned} 2^{d+1} - 1 &\geq n \\ 2^{d+1} &\geq n+1 \\ d+1 &\geq \log_2 \lceil n+1 \rceil \\ d &\geq \log_2 \lceil n+1 \rceil - 1 \end{aligned}$$

  - obere Schranke:
    Es wird zuerst ein AVL-Baum rekursiv definiert, dazu führen wir $N(h)$ als Anzahl der Knoten eines AVL-Baumes der Höhe h ein.
    Für die Höhe $h = 0$ besteht der AVL-Baum nur aus der Wurzel und es gilt $N(0) = 1$.
    Für die Höhe $h = 1$ besteht der AVL-Baum nur aus der Wurzel und einem Sohn[16]. Es gilt $N(1) = 2$.
    Für die Höhe $h \geq 2$ kann man sagen, dass die Strukturinvariante auch in den einzelnen Teilbäumen gilt. Somit kann man rekursiv zwei Bäume der Höhe $h-1$ und $h-2$ zu einem neuen, minimal

[16] Für die obere Schranke betrachtet man minimal gefüllte Bäume.

gefüllten Baum der Höhe h kombinieren.
Für die Anzahl der Knoten ergibt sich die Rekursionsgleichung

$$N(h) = 1 + N(h-1) + N(h-2)$$

Eine Ähnlichkeit mit den Fibonacci-Zahlen ist gegeben. Es gilt

$$N(h) = \text{Fib}(h+3) - 1$$

was man über vollständige Induktion zeigt.
Induktionsanfang:
$h = 0$:

$$N(0) = 1 = \text{Fib}(3) - 1$$

$h = 1$:

$$N(1) = 2 = 2 - 1 = \text{Fib}(4) - 1$$

Es gilt nach Induktionsvoraussetzung
Induktionsschritt:
$h \to h + 1$:

$$\begin{aligned} N(h) &= 1 + N(h-1) + N(h-2) \\ &= 1 + \text{Fib}(h+2) - 1 + \text{Fib}(h+1) - 1 \\ &= \text{Fib}(h+3) - 1 \end{aligned}$$

- Sei nun ein AVL-Baum T mit n Daten und Höhe h gegeben. Als Ausgangsgleichung haben wir

$$\text{Fib}(h+3) - 1 \leq n$$

woraus wir die Höhe des AVL-Baumes berechnen wollen. Über die Fibonacci-Zahlen wissen wir

$$\text{Fib(j)} = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^j - \left(\frac{1-\sqrt{5}}{2}\right)^j}{\sqrt{5}}$$

wobei man $\frac{\left(\frac{1-\sqrt{5}}{2}\right)^j}{\sqrt{5}}$ mit $\frac{1}{\sqrt{5}}$ abschätzt. Mit $j = h + 3$ führt dies zu

$$\text{Fib}(h+3) \geq \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}}{\sqrt{5}} - \frac{1}{\sqrt{5}}$$

Nun nutzt man die Transitivität mit $\frac{\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}}{\sqrt{5}} - \frac{1}{\sqrt{5}} - 1 \leq \text{Fib}(h+3) - 1 \leq n$ aus, so ergibt sich

$$\begin{aligned} \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}}{\sqrt{5}} - \frac{1}{\sqrt{5}} - 1 &\leq n \\ \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}}{\sqrt{5}} &\leq n + \underbrace{1 + \frac{1}{\sqrt{5}}}_{\text{Abschätzen mit 2}} \\ \left(\frac{1+\sqrt{5}}{2}\right)^{h+3} &\leq \sqrt{5}(n+2) \\ (h+3) \cdot \log\left(\frac{1+\sqrt{5}}{2}\right) &\leq \log\left(\sqrt{5}(n+2)\right) \\ h+3 &\leq \frac{\log\left(\sqrt{5}(n+2)\right)}{\log\left(\frac{1+\sqrt{5}}{2}\right)} \simeq 1,44 \log n \end{aligned}$$

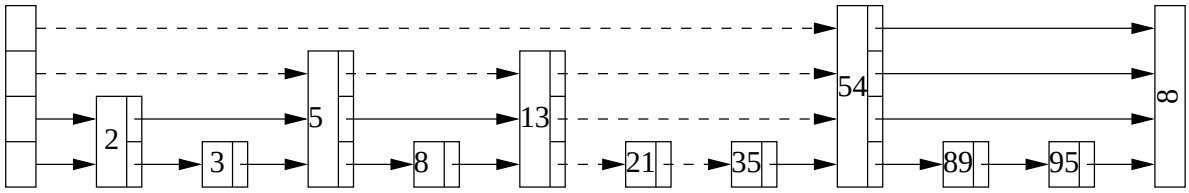
Also ist der AVL-Baum höchstens um 44% höher als ein binärer Suchbaum.

3.6 Skiplisten

3.6.1 Operationen

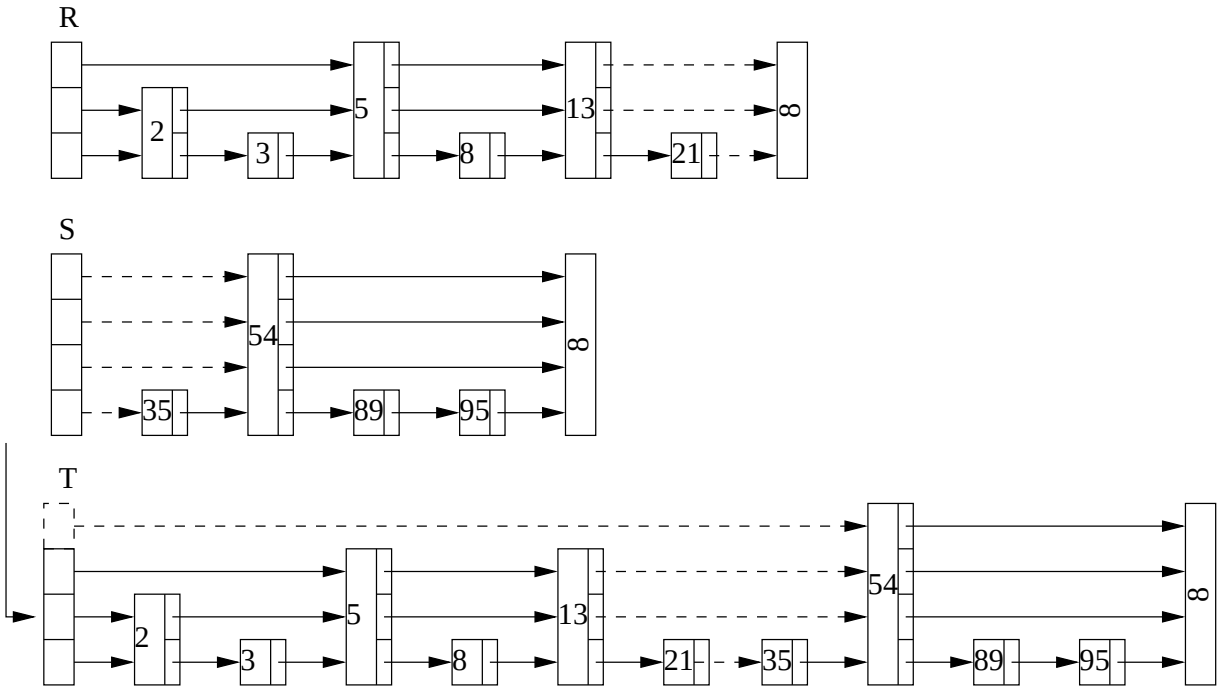
- SEARCH(x): Die Suche beginnt auf der höchsten Ebene des Anfangsobjekts. Stößt man auf einen Zeiger, der auf ein Datum größer als x zeigt, so steigt man in der Listenhierarchie eine Ebene hinab und sucht dort weiter. Ansonsten folgt man dem Zeiger und stellt dort die gleiche Betrachtung wieder an. Nach gewisser Zeit gelangen wir auf Ebene 0, dort finden wir x (erfolgreiche Suche) oder ein zu großes Objekt (erfolglose Suche).

Search(35)



- CONCATENATE(L_1, L_2, L): Als Voraussetzung soll gelten, daß alle Elemente in L_1 kleiner sind als die Elemente von L_2. In der vorderen Skipliste L_1 sucht man die Zeiger, die auf dessen Ende zeigen. In der hinter Skipliste L_2 sucht man die Zeiger, die von dessen Anfangsobjekt ausgehen. Diese markierten Zeiger beider Listen müssen entsprechend der verschiedenen Ebenen verbunden werden. Evtl. muss auch die Höhe des Anfangs- bzw. Endobjekts angepasst werden.

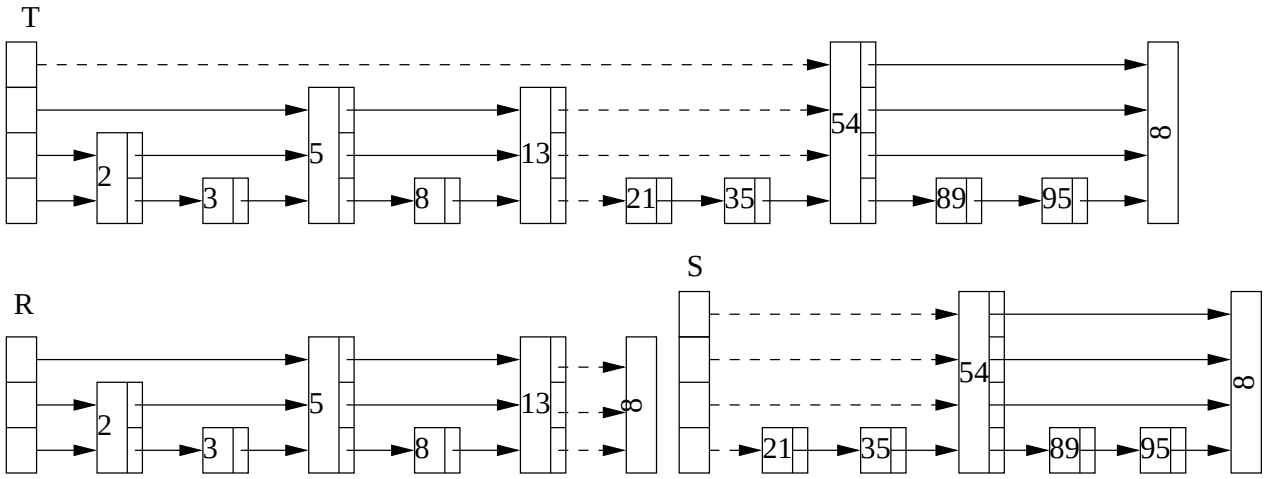
Concatenate(R,S,T)



- SPLIT(L, a, L_1, L_2): (nach erfolgreicher Suche)

  Bei der Suche nach a hat man bereits alle Zeiger gefunden, die umgesetzt werden müssen. Dies sind einerseits die Zeiger, die über a "hinausschießen" und andererseits die Zeiger, die von a ausgehen. Im ersten abgetrennten Teil müssen diese auf das Ende zeigen und im zweiten Teil vom neuen Anfangsobjekt aus starten.

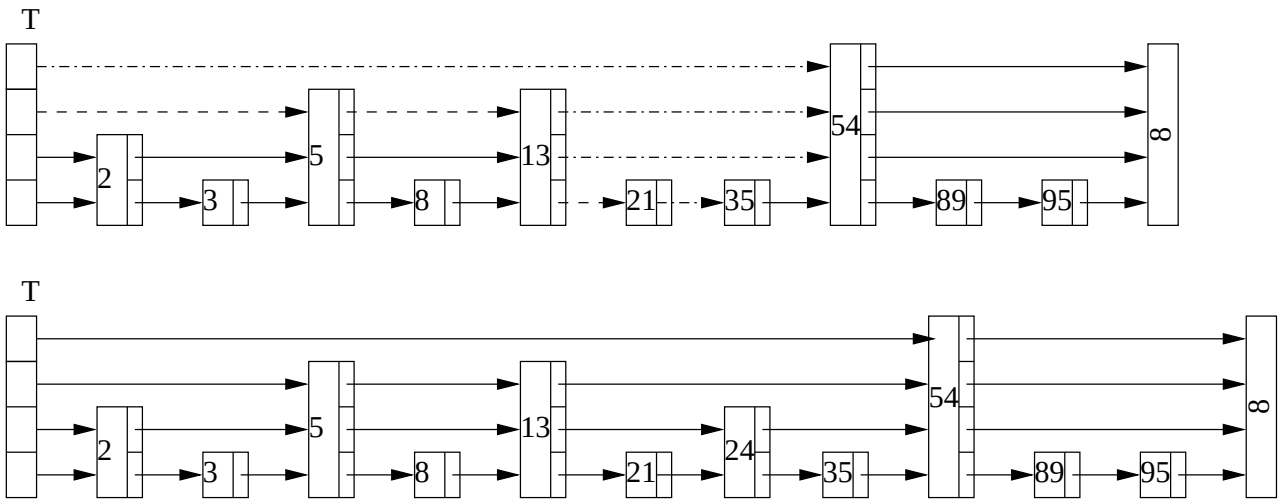
Split(T,13,R,S)



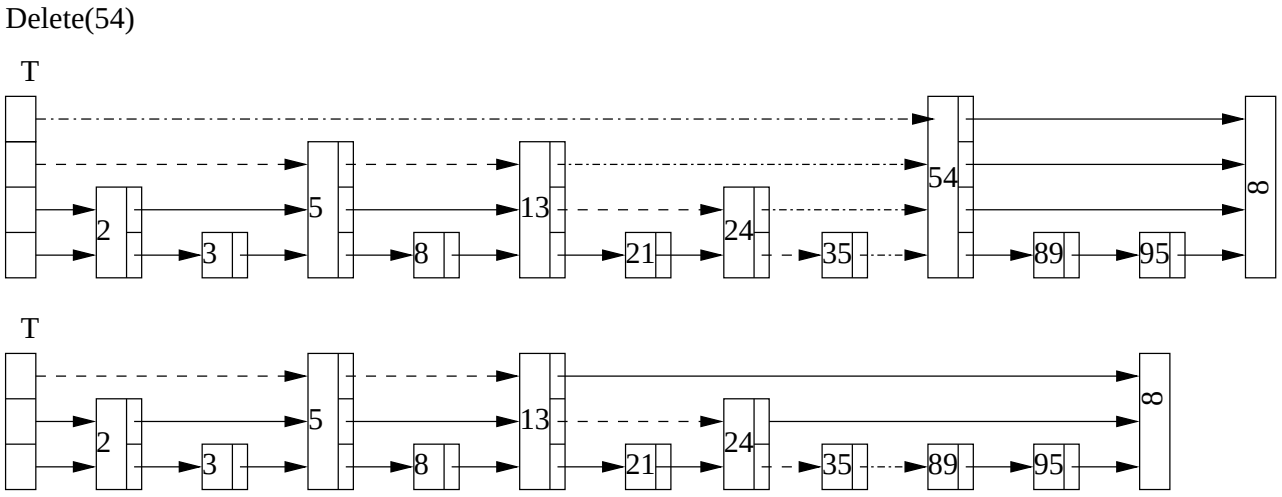
- INSERT(x): Ist das Datum x bereits in der Skipliste vorhanden, so wird es einfach überschrieben. Ansonsten folgt das Einfügen auf eine erfolglose Suche. Für x wird die zugehörige Höhe h ausgewürfelt. Ist h größer als die momentan maximale Höhe, so werden entsprechend viele neuen Listen gebildet, die zwei Zeiger haben. Der erste Zeiger beginnt beim Anfangsobjekt und endet bei x, der zweite Zeiger beginnt bei x und endet beim Endobjekt.
  Im Folgenden betrachten wir die Ebene l und ein einzufügendes Objekt x der Höhe h (vertreten auf den Ebenen $0, \cdots, h-1$)

  - $h - 1 < l$: Hier müssen keine Zeiger umgesetzt werden, da das Objekt x auf l nicht vertreten ist.
  - $h - 1 \geq l$: Durch die erfolglose Suche kennen wir alle Zeiger, die "durch das Objekt x" gehen. Diese werden nun in der Mitte aufgeschnitten und das Objekt x dort eingesetzt.

Insert(24)



- Delete(x): Suche (erfolglos) nach dem größten *nichtexistenten* Objekt y für das gilt: y ist kleiner als x und größer als jedes andere existente Element kleiner x. So erhält man alle Zeiger auf das zu löschende Element x. Diese Zeiger werden nun einfach auf den jeweiligen Nachfolger in den einzelnen Ebenen umgeleitet. Dann kann das Objekt gelöscht werden.



3.6.2 Höhe der Skipliste

Die Wahrscheinlichkeit für die Höhe h beträgt 2^{-h}. Die erwartete Höhe beträgt dann

$$\sum_{h=1}^{\infty} h \cdot 2^{-h} = \frac{1}{2} \sum_{h=1}^{\infty} h \cdot \left(\frac{1}{2}\right)^{h-1}$$

Der Term sieht wie die Ableitung einer geometrischen Reihe $\sum_{h=1}^{\infty} q^h$ aus, die gegen $\frac{1}{1-q}$ konvergiert. Um an unser Ergebnis zu kommen, muss man $\frac{1}{1-q}$ ableiten. Man erhält dann

$$\begin{aligned} \sum_{h=1}^{\infty} h \cdot 2^{-h} &= \frac{1}{2} \cdot \sum_{h=1}^{\infty} h \cdot \left(\frac{1}{2}\right)^{h-1} \\ &= \frac{1}{2} \cdot \left(\frac{1}{1-\frac{1}{2}}\right)^2 \\ &= 2 \end{aligned}$$

3.6.3 Analyse der Skipliste

Definieren wir zuerst zwei Variablen $H(n)$ und $Z(n)$ für die Höhe bzw. die Anzahl der Zeiger der Skipliste.

- $\text{Prob}(H(n) \geq h) \leq \min\left\{1, n \cdot \left(\frac{1}{2}\right)^{h-1}\right\}$
  - Die obere Schranke 1 ist trivial.
  - Die Wahrscheinlichkeit, dass ein Objekt die Höhe h erreicht beträgt $\left(\frac{1}{2}\right)^{h-1}$. Nach bereits $h-1$ erfolglosen Würfen ist sicher, dass die Höhe mindestens h beträgt. Dies ist nämlich genau dann der Fall, wenn der h-te Wurf erfolgreich ist. Wäre er erfolglos, so wird die erwürfelte Höhe sogar größer als h sein.
    Das eines der n Objekte die Höhe h hat, kann man mit $n \cdot \left(\frac{1}{2}\right)^{h-1}$ abschätzen.
- $\text{E}(H(n)) \leq \lfloor \log_2(n) \rfloor + 3$

  $\text{E}(H(n))$ ist definiert durch

$$\text{E}(H(n)) = \sum_{h=1}^{\infty} h \cdot \text{Prob}(H(n) = h)$$

damit ist der Durchschnitt gemeint (Höhe · Wahrscheinlichkeit für diese Höhe).

$$\begin{aligned}
\mathrm{E}(H(n)) &= \sum_{h=1}^{\infty} h \cdot \mathrm{Prob}(H(n)=h) \\
&= \mathrm{Prob}(H(n)=1) + \mathrm{Prob}(H(n)=2) + \mathrm{Prob}(H(n)=3) + \cdots \\
&\quad + \mathrm{Prob}(H(n)=2) + \mathrm{Prob}(H(n)=3) + \cdots \\
&\quad + \mathrm{Prob}(H(n)=3) + \cdots \\
&\quad + \cdots
\end{aligned}$$

Nun wird zeilenweise addiert

$$\begin{aligned}
\mathrm{E}(H(n)) &= \mathrm{Prob}(H(n) \geq 1) \\
&\quad + \mathrm{Prob}(H(n) \geq 2) \\
&\quad + \mathrm{Prob}(H(n)\,3) \\
&\quad + \cdots \\
&= \sum_{h=1}^{\infty} \mathrm{Prob}(H(n) \geq h)
\end{aligned}$$

Für wieviele Summanden liegt eine Wahrscheinlichkeit von 1 vor, d.h. bis zu welcher Höhe kommen bestimmt Elemente vor ?

$$\begin{aligned}
n \cdot \left(\frac{1}{2}\right)^{h-1} &\geq 1 \\
\frac{1}{2^{h-1}} &\geq \frac{1}{n} \\
2^{h-1} &\leq n \\
h-1 &\leq \lceil \log_2(n) \rceil \\
h &\leq \underbrace{\lceil \log_2(n) \rceil}_{\leq \lfloor \log_2(n) \rfloor + 1} + 1 \\
h &\leq \lfloor \log_2(n) \rfloor + 1 + 1 \\
h &\leq \lfloor \log_2(n) \rfloor + 2
\end{aligned}$$

Somit ist eine Höhe von $\lfloor \log_2(n) \rfloor + 2$ garantiert, d.h die ersten $\lfloor \log_2(n) \rfloor + 2$ Summanden kann man mit 1 abschätzen.
Für die restlichen Summanden gilt:
Den $\lfloor \log_2(n) \rfloor + 2 + i$-ten Summanden schätzen wird nach dem ersten Punkt mit $\left(\frac{1}{2}\right)^i$ ab. Damit folgt

$$\begin{aligned}
\sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i &= \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i - \left(\frac{1}{2}\right)^0 \\
&= \frac{1}{1-\frac{1}{2}} - 1 \\
&= 1
\end{aligned}$$

so dass die restlichen Summanden durch 1 abgeschätzt werden können. Insgesamt kann man also $\mathrm{E}(H(n))$ durch $\lfloor \log_2(n) \rfloor + 3$ abschätzen.

- $\mathrm{E}(Z(n)) \leq 2 \cdot n + \lfloor \log_2(n) \rfloor + 3$

  Einerseits hat jedes Objekt die durchschnittliche Höhe 2, also zwei Zeiger. Macht bei n Elementen $2 \cdot n$ Zeiger. Andererseits kommen höchstens noch $H(n)$ Zeiger hinzu, was $\lfloor \log_2(n) \rfloor + 3$ entspricht. Insgesamt folgt, dass $\mathrm{E}(Z(n))$ durch $2 \cdot n + \lfloor \log_2(n) \rfloor + 3$ nach oben abgeschätzt werden kann.

- Der erwartete Platzbedarf für eine Skipliste auf n Daten beträgt $O(n)$.

  Die Anzahl der Elemente ist n, die Anzahl der Zeiger $2 \cdot n + \lfloor \log_2(n) \rfloor + 3$. Insgesamt folgt damit, dass der Platzbedarf $O(n)$ beträgt.

4 Entwurfsmethoden für Algorithmen

4.1 Greedy Algorithmen

Wird bei Optimierungsproblemen angewandt, bei denen die möglichen Löungen aus mehreren Teilen bestehen. Die Lösung wird stückweise konstruiert unter den Gesichtspunkten:

- Wähle das Objekt, das den Wert der Teillösung maximiert
- Schaue bei der Wahl des Objekts nicht in die Zukunft
- Eine einmal getroffene Entscheidung wir nie zurückgenommen

Es sollte klar seien, dass eine im Moment schlechtere Lösung (des Teilproblems) zu einer besseren Lösung des Gesamtproblems führen kann, was von Greedy-Algorithmen nicht beachtet wird.
Bei der Anwendung von Greedy-Algorithmen sind folgende Ergebnisse möglich:

1. Es wird eine optimale Lösung berechnet
2. Eine nicht optimale Lösung, die aber sehr nah an der optimalen liegt wird berechnet
3. Es wird eine sehr schlecht Lösung berechnet

4.1.1 Münzwechselproblem

In einem Land gibt es Münzen mit Werten $n_1, \cdots, n_k$ mit o.B.d.A $n_1 > \cdots > n_k$. Damit man alle Beträge auszahlen kann, wählt man $n_k = 1$.
Problemstellung: Einen gegebenen Betrag N will man dann mit einer minimalen Anzahl an Münzen auszahlen.

4.1.1.1 Greedy-Algorithmus: Teile den Betrag durch den größten Wert n_1 und zahle dementsprechend $x = \left\lfloor \frac{N}{n_1} \right\rfloor$ Münzen aus. Mach mit dem Restbetrag $N - x \cdot n_1$ und dem nächstkleineren Münzwert weiter.
Formal hingeschrieben:

1. $W := N$
2. Für $i = 1, \cdots, k$: Wähle $\left\lfloor \frac{W}{n_i} \right\rfloor$ Münzen mit Wert n_i aus und setze den Restbetrag W auf $W = W - \left\lfloor \frac{W}{n_i} \right\rfloor \cdot n_i$.

Da zu Beginn $n_k = 1$ vereinbart wurde, erhalten wir immer eine zulässige Lösung.

4.1.1.2 Greedy nicht so optimal: In einem fernen Land gibt es folgende Münzwerte:

- $n_1 = 2n_2 + 1$
- $n_2 > 3$ beliebig
- $n_3 = 1$

Wir wollen nun $N = 3n_2$ ausgezahlt haben.
Was Greedy macht:

1. $W := 3n_2$
   Wähle $\left\lfloor \frac{3n_2}{n_1} \right\rfloor = \left\lfloor \frac{3n_2}{2n_2+1} \right\rfloor = 1$ Münze mit Wert n_1 aus und setze den Restbetrag W auf $W = 3n_2 - (2n_2 + 1) = n_2 - 1$.
2. $W := n_2 - 1$
   Wähle $\left\lfloor \frac{n_2-1}{n_2} \right\rfloor = 0$ Münzen mit Wert n_2 aus und setze den Restbetrag W auf $W = n_2 - 1 - 0 \cdot (n_2) = n_2 - 1$.
3. $W := n_2 - 1$
   Wähle $\left\lfloor \frac{n_2-1}{n_3} \right\rfloor = \left\lfloor \frac{n_2-1}{1} \right\rfloor = n_2 - 1$ Münzen mit Wert n_3 aus und setze den Restbetrag W auf $W = n_2 - 1 - 1 \cdot (n_2 - 1) = 0$.

Insgesamt zahlt Greedy somit n_2 Münzen aus, obwohl bereits drei im Wert von n_2 gereicht hätten.

4.1.2 Bin Packing Problem (BPP)

Es sind n Objekte mit Gewichten $a_1, \cdots, a_n \in]0,1]$ gegeben. Diese Objekte sollen auf möglichst wenige Bins (Kisten) verteilt werden, *wobei jede Kiste die Tragfähigkeit 1 hat.*

4.1.2.1 First Fit

1. Seien n leere Bins $B_1, \cdots, B_n$ gegeben.
2. Für $i = 1, \cdots, n$:
   Bestimme das kleinste j, so daß das Objekt i noch in Bin j gepackt werden kann, und packe das Objekt dort hinein.

Suche also nach der erstbesten Kiste, in die das Objekt noch paßt und lege es dort hinein.

4.1.2.2 Best Fit

1. Seien n leere Bins $B_1, \cdots, B_n$ gegeben.
2. Für $i = 1, \cdots, n$:
   Bestimme das j, so daß das Objekt i noch in Bin j gepackt werden kann und der Freiraum minimal unter allen Bins ist, und packe das Objekt dort hinein.

Suche nach einer Kiste, in die das Objekt gerade noch so reinpaßt. Angefangene Bins werden möglichst voll gepackt.

4.1.2.3 Folgerungen Die Eingabe wird mit I, die Anzahl der Bins einer optimalen Lösung mit $OPT(I)$ bezeichnet, mit $FF(I)$ bzw. $BF(I)$ die Anzahl der Bins, die durch FirstFit bzw. BestFit berechnet wurden.

1. Für alle Eingaben I gilt $\frac{FF(I)}{OPT(I)} \leq 1.7$.
   Im Beweis zeigt man $\frac{FF(I)}{OPT(I)} \leq 2$.
2. $\forall \varepsilon > 0: \quad \exists I$ mit $\frac{FF(I)}{OPT(I)} \geq 1.7 - \varepsilon$.
   Im Beweis zeigt man $\forall a > 0: \quad \exists I$ mit $OPT(I) \geq 2$ und $\frac{FF(I)}{OPT(I)} = \frac{5}{3}$.
3. Die Aussagen (1) und (2) gelten auch für BestFit anstelle von FirstFit.

Beweis:

1. Es ist sicherlich $OPT(I) \geq a_1 + \cdots + a_n$, da alle Objekte verstaut werden müssen.
   Weiterhin ist für $FF(I)$ höchstens eine der Kisten halbvoll.Wären zwei "nur" halbvoll, so wäre der Inhalt der zweiten Kiste bereits in die erste gepackt worden.
   Jede (bis auf vielleicht eine) Kiste ist für $FF(I)$ mit über 0,5 ausgelastet. Packt man nocheinmal alle n Objekte ein, so gilt $2(a_1 + \cdots + a_n) > FF(I)$, die Kisten würden nicht genug Platz bieten.
   Man kann nun folgern:

$$\begin{aligned} OPT(I) &\geq a_1 + \cdots + a_n \\ 2(a_1 + \cdots + a_n) &> FF(I) \\ \Rightarrow \quad OPT(I) &\geq a_1 + \cdots + a_n \\ (a_1 + \cdots + a_n) &> \frac{FF(I)}{2} \\ \Rightarrow \quad OPT(I) &\geq \frac{FF(I)}{2} \\ \Rightarrow \quad 2 &\geq \frac{FF(I)}{OPT(I)} \text{ bzw } \frac{FF(I)}{OPT(I)} \leq 2 \end{aligned}$$

4.2 Dynamische Programmierung

Viele Probleme lassen sich in Teilprobleme zerlegen, beispielsweise bei Quicksort. Jedoch ist nicht immer bekannt, wie der Trennpunkt zu setzen ist, so dass eine optimale Gesamtlösung erzielt werden kann. Man muss, um sicher zu gehen, alle möglichen Trennpunkt ausprobieren und das Vorgehen rekursiv fortsetzen. Dies führt zu exponentiell vielen Berechnungen. Viele kleine Teilprobleme werden dabei mehrfach neu berechnet. Bei der dynamischen Programmierung wird ein anderer Ansatz verfolgt. Die kleinen Teilprobleme werden nur einmal berechnet und daraus sukzessiv die größeren konstruiert, bis man das Gesamtproblem erfaßt hat.

4.2.1 Matrizenmultiplikation

Es soll ein Produkt $M_1 \cdot M_2 \cdot \cdots \cdot M_n$ aus mehreren Matrizen berechnet werden, wobei man die Anzahl an Operationen (Multiplikationen) möglichst gering halten will. Dies kann durch eine entsprechende Klammerung erreichen. Löst man jeweils die Ausdrücke in den Klammern optimal, so wird es eine optimale Gesamtlösung geben. Die Bellmansche Optimalitätsgleichung hat für dieses Problem die Form

$$c(i.j) \quad = \quad \min\{c(i,k) + c(k+1,j) + r_{i-1} r_k r_j \mid i < k < j\}$$

Die Kosten für das Produkt $M_i \cdot \cdots \cdot M_j$ ergeben sich also aus den Kosten für $M_i \cdot \cdots \cdot M_k$ bzw. $M_{k+1} \cdot \cdots \cdot M_j$ und der Multiplatiktion von $M_i \cdot \cdots \cdot M_k$ mit $M_{k+1} \cdot \cdots \cdot M_j$.

4.2.1.1 Beispiel 5.3.1 aus dem Skript $n = 4,\ r_0 = 10,\ r_1 = 20,\ r_2 = 50,\ r_3 = 1,\ r_4 = 100$

1. Lösung der trivialen Teilprobleme:

   - Anzahl der Multiplikationen für $M_1 \star M_2$:
     $i = 1: \quad k(1,2) := 1, \quad c(1,2) := r_0 \cdot r_1 \cdot r_2 = 10 \cdot 20 \cdot 50 = 10.000$
   - Anzahl der Multiplikationen für $M_2 \star M_3$:
     $i = 2: \quad k(2,3) := 2, \quad c(2,3) := r_1 \cdot r_2 \cdot r_3 = 20 \cdot 50 \cdot 1 = 1.000$
   - Anzahl der Multiplikationen für $M_3 \star M_4$:
     $i = 3: \quad k(3,4) := 3, \quad c(3,4) := r_2 \cdot r_3 \cdot r_4 = 50 \cdot 1 \cdot 100 = 5.000$

2. Aus den trivialen Teilproblemen werden komplexere zusammengebaut wobei

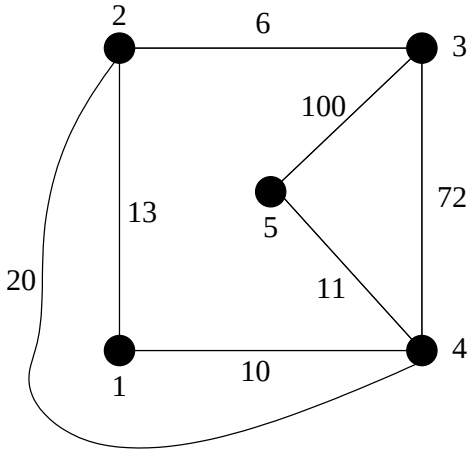
$$c(i,i+l) \quad = \quad \min\{c(i,k) + c(k+1,i+l) + r_{i-1} r_k r_{i+l} \mid i \le k \le i+l-1\}$$

   - $l = 2$:
     - $i = 1$
       $k \in \{1,2\}$
       $c(1,3) = \min \left\{ \begin{array}{c} c(1,1) + c(2,3) + r_0 \cdot r_1 \cdot r_3 = 0 + 1000 + 200 \\ c(1,2) + c(3,3) + r_0 \cdot r_2 \cdot r_3 = 10.000 + 0 + 500 \end{array} \right\} = 1.200$
       $\rightarrow \quad k = 1$
     - $i = 2$
       $k \in \{2,3\}$
       $c(2,4) = \min \left\{ \begin{array}{c} c(2,2) + c(3,4) + r_1 \cdot r_2 \cdot r_4 = 0 + 5000 + 100.000 \\ c(2,3) + c(4,4) + r_1 \cdot r_3 \cdot r_4 = 1000 + 0 + 2000 \end{array} \right\} = 3000$
       $\rightarrow \quad k = 3$
   - $l = 3$:
     - $i = 1$
       $k \in \{1,2,3\}$
       $c(1,4) = \min \left\{ \begin{array}{c} c(1,1) + c(2,4) + r_0 \cdot r_1 \cdot r_4 = 0 + 3000 + 200 \\ c(1,2) + c(3,4) + r_0 \cdot r_2 \cdot r_4 = 10.000 + 5000 + 50000 \\ c(1,3) + c(4,4) + r_0 \cdot r_3 \cdot r_4 = 1200 + 0 + 1000 \end{array} \right\} = 2.200$
       $\rightarrow \quad k = 3$

4.2.2 All-pairs-shortest-paths

In diesem Beispiel benutzen wir einen ungerichteten Graphen und deshalb betrachten wir nur die obere Dreiecksmatrix. Für gerichtete Graphen muss man die ganze Matrix betrachten.

Gegeben sei folgender Graph:



Stelle die Kostenmatrix auf:

1. Für die trivialen Teilprobleme $w_{ij}^{(0)}$, dies sind die direkten Kanten zwischen i und j, gilt

$_i{}^j$	1	2	3	4	5
1		13	∞	10	∞
2			6	20	∞
3				72	100
4					11
5					

2. Nun sei $l = 1$:

$$w_{ij}^{(1)} = \min\left\{w_{ij}^{(0)}, w_{i1}^{(0)} + w_{1j}^{(0)}\right\}$$

Im Einzelnen:

$$\begin{aligned}
w_{12}^{(1)} &= \min\left\{w_{12}^{(0)}, w_{11}^{(0)} + w_{12}^{(0)}\right\} = \min\{13, 0+13\} = 13\\
w_{13}^{(1)} &= \min\left\{w_{13}^{(0)}, w_{11}^{(0)} + w_{13}^{(0)}\right\} = \min\{\infty, 0+\infty\} = \infty\\
w_{14}^{(1)} &= \min\left\{w_{14}^{(0)}, w_{11}^{(0)} + w_{14}^{(0)}\right\} = \min\{10, 0+10\} = 10\\
w_{15}^{(1)} &= \min\left\{w_{15}^{(0)}, w_{11}^{(0)} + w_{15}^{(0)}\right\} = \min\{\infty, 0+\infty\} = \infty\\
w_{23}^{(1)} &= \min\left\{w_{23}^{(0)}, w_{21}^{(0)} + w_{13}^{(0)}\right\} = \min\{6, 13+\infty\} = 6\\
w_{24}^{(1)} &= \min\left\{w_{24}^{(0)}, w_{21}^{(0)} + w_{14}^{(0)}\right\} = \min\{20, 13+10\} = 20\\
w_{25}^{(1)} &= \min\left\{w_{25}^{(0)}, w_{21}^{(0)} + w_{15}^{(0)}\right\} = \min\{\infty, 13+\infty\} = \infty\\
w_{34}^{(1)} &= \min\left\{w_{34}^{(0)}, w_{31}^{(0)} + w_{14}^{(0)}\right\} = \min\{72, \infty+10\} = 72\\
w_{35}^{(1)} &= \min\left\{w_{35}^{(0)}, w_{31}^{(0)} + w_{15}^{(0)}\right\} = \min\{100, \infty+\infty\} = 100\\
w_{45}^{(1)} &= \min\left\{w_{45}^{(0)}, w_{41}^{(0)} + w_{15}^{(0)}\right\} = \min\{11, 10+\infty\} = 11
\end{aligned}$$

Daraus ergibt sich die Matrix:

$_i{}^j$	1	2	3	4	5
1		13	∞	10	∞
2			6	20	∞
3				72	100
4					11
5					

3. Nun sei $l = 2$:

$$w_{ij}^{(2)} = \min\left\{w_{ij}^{(1)}, w_{i2}^{(1)} + w_{2j}^{(1)}\right\}$$

Im Einzelnen:

$$\begin{aligned}
w_{12}^{(2)} &= \min\left\{w_{12}^{(1)}, w_{12}^{(1)} + w_{22}^{(1)}\right\} = \min\{13, 13+0\} = 13\\
w_{13}^{(2)} &= \min\left\{w_{13}^{(1)}, w_{12}^{(1)} + w_{23}^{(1)}\right\} = \min\{\infty, 13+6\} = 19\\
w_{14}^{(2)} &= \min\left\{w_{14}^{(1)}, w_{12}^{(1)} + w_{24}^{(1)}\right\} = \min\{10, 13+20\} = 10\\
w_{15}^{(2)} &= \min\left\{w_{15}^{(1)}, w_{12}^{(1)} + w_{25}^{(1)}\right\} = \min\{\infty, 13+\infty\} = \infty\\
w_{23}^{(2)} &= \min\left\{w_{23}^{(1)}, w_{22}^{(1)} + w_{23}^{(1)}\right\} = \min\{6, 0+6\} = 6\\
w_{24}^{(2)} &= \min\left\{w_{24}^{(1)}, w_{22}^{(1)} + w_{24}^{(1)}\right\} = \min\{20, 0+20\} = 20\\
w_{25}^{(2)} &= \min\left\{w_{25}^{(1)}, w_{22}^{(1)} + w_{25}^{(1)}\right\} = \min\{\infty, 0+\infty\} = \infty\\
w_{34}^{(2)} &= \min\left\{w_{34}^{(1)}, w_{32}^{(1)} + w_{24}^{(1)}\right\} = \min\{72, 6+20\} = 26\\
w_{35}^{(2)} &= \min\left\{w_{35}^{(1)}, w_{32}^{(1)} + w_{25}^{(1)}\right\} = \min\{100, 6+\infty\} = 100\\
w_{45}^{(2)} &= \min\left\{w_{45}^{(1)}, w_{42}^{(1)} + w_{25}^{(1)}\right\} = \min\{11, 20+\infty\} = 11
\end{aligned}$$

Daraus ergibt sich die Matrix:

$_i{}^j$	1	2	3	4	5
1		13	19	10	∞
2			6	20	∞
3				26	100
4					11
5					

4. Nun sei $l = 3$:

$$w_{ij}^{(3)} = \min\left\{w_{ij}^{(2)}, w_{i3}^{(2)} + w_{3j}^{(2)}\right\}$$

Im Einzelnen:

$$
\begin{aligned}
w_{12}^{(3)} &= \min\left\{w_{12}^{(2)}, w_{13}^{(2)} + w_{32}^{(2)}\right\} = \min\{13, 19+6\} = 13\\
w_{13}^{(3)} &= \min\left\{w_{13}^{(2)}, w_{13}^{(2)} + w_{33}^{(2)}\right\} = \min\{19, 19+0\} = 19\\
w_{14}^{(3)} &= \min\left\{w_{14}^{(2)}, w_{13}^{(2)} + w_{34}^{(2)}\right\} = \min\{10, 19+26\} = 10\\
w_{15}^{(3)} &= \min\left\{w_{15}^{(2)}, w_{13}^{(2)} + w_{35}^{(2)}\right\} = \min\{\infty, 19+100\} = 119\\
w_{23}^{(3)} &= \min\left\{w_{23}^{(2)}, w_{23}^{(2)} + w_{33}^{(2)}\right\} = \min\{6, 6+0\} = 6\\
w_{24}^{(3)} &= \min\left\{w_{24}^{(2)}, w_{23}^{(2)} + w_{34}^{(2)}\right\} = \min\{20, 6+26\} = 20\\
w_{25}^{(3)} &= \min\left\{w_{25}^{(2)}, w_{23}^{(2)} + w_{35}^{(2)}\right\} = \min\{\infty, 6+100\} = 106\\
w_{34}^{(3)} &= \min\left\{w_{34}^{(2)}, w_{33}^{(2)} + w_{34}^{(2)}\right\} = \min\{26, 0+72\} = 26\\
w_{35}^{(3)} &= \min\left\{w_{35}^{(2)}, w_{33}^{(2)} + w_{35}^{(2)}\right\} = \min\{100, 0+100\} = 100\\
w_{45}^{(3)} &= \min\left\{w_{45}^{(2)}, w_{43}^{(2)} + w_{35}^{(2)}\right\} = \min\{11, 26+100\} = 11
\end{aligned}
$$

Daraus ergibt sich die Matrix:

$_i{}^j$	1	2	3	4	5
1		13	19	10	119
2			6	20	106
3				26	100
4					11
5					

5. Nun sei $l = 4$:

$$w_{ij}^{(4)} = \min\left\{w_{ij}^{(3)}, w_{i4}^{(3)} + w_{4j}^{(3)}\right\}$$

Im Einzelnen:

$$
\begin{aligned}
w_{12}^{(4)} &= \min\left\{w_{12}^{(3)}, w_{14}^{(3)} + w_{42}^{(3)}\right\} = \min\{13, 10+20\} = 13\\
w_{13}^{(4)} &= \min\left\{w_{13}^{(3)}, w_{14}^{(3)} + w_{43}^{(3)}\right\} = \min\{19, 10+26\} = 19\\
w_{14}^{(4)} &= \min\left\{w_{14}^{(3)}, w_{14}^{(3)} + w_{44}^{(3)}\right\} = \min\{10, 10+0\} = 10\\
w_{15}^{(4)} &= \min\left\{w_{15}^{(3)}, w_{14}^{(3)} + w_{45}^{(3)}\right\} = \min\{119, 10+11\} = 21\\
w_{23}^{(4)} &= \min\left\{w_{23}^{(3)}, w_{24}^{(3)} + w_{43}^{(3)}\right\} = \min\{6, 20+26\} = 6\\
w_{24}^{(4)} &= \min\left\{w_{24}^{(3)}, w_{24}^{(3)} + w_{44}^{(3)}\right\} = \min\{20, 20+0\} = 20\\
w_{25}^{(4)} &= \min\left\{w_{25}^{(3)}, w_{24}^{(3)} + w_{45}^{(3)}\right\} = \min\{106, 20+11\} = 31\\
w_{34}^{(4)} &= \min\left\{w_{34}^{(3)}, w_{34}^{(3)} + w_{44}^{(3)}\right\} = \min\{26, 26+0\} = 26\\
w_{35}^{(4)} &= \min\left\{w_{35}^{(3)}, w_{34}^{(3)} + w_{45}^{(3)}\right\} = \min\{100, 26+11\} = 37\\
w_{45}^{(4)} &= \min\left\{w_{45}^{(3)}, w_{44}^{(3)} + w_{45}^{(3)}\right\} = \min\{11, 0+11\} = 11
\end{aligned}
$$

Daraus ergibt sich die Matrix:

$_i{}^j$	1	2	3	4	5
1		13	19	10	21
2			6	20	31
3				26	37
4					11
5					

6. Nun sei $l = 5$:

$$w_{ij}^{(5)} = \min\left\{w_{ij}^{(4)}, w_{i5}^{(4)} + w_{5j}^{(4)}\right\}$$

Im Einzelnen:

$$\begin{aligned}
w_{12}^{(5)} &= \min\left\{w_{12}^{(4)}, w_{15}^{(4)} + w_{52}^{(4)}\right\} = \min\{13, 21+31\} = 13\\
w_{13}^{(5)} &= \min\left\{w_{13}^{(4)}, w_{15}^{(4)} + w_{53}^{(4)}\right\} = \min\{19, 21+37\} = 19\\
w_{14}^{(5)} &= \min\left\{w_{14}^{(4)}, w_{15}^{(4)} + w_{54}^{(4)}\right\} = \min\{10, 21+11\} = 10\\
w_{15}^{(5)} &= \min\left\{w_{15}^{(4)}, w_{15}^{(4)} + w_{55}^{(4)}\right\} = \min\{21, 21+0\} = 21\\
w_{23}^{(5)} &= \min\left\{w_{23}^{(4)}, w_{25}^{(4)} + w_{53}^{(4)}\right\} = \min\{6, 31+37\} = 6\\
w_{24}^{(5)} &= \min\left\{w_{24}^{(4)}, w_{25}^{(4)} + w_{54}^{(4)}\right\} = \min\{20, 31+11\} = 20\\
w_{25}^{(5)} &= \min\left\{w_{25}^{(4)}, w_{25}^{(4)} + w_{55}^{(4)}\right\} = \min\{31, 31+0\} = 31\\
w_{34}^{(5)} &= \min\left\{w_{34}^{(4)}, w_{35}^{(4)} + w_{54}^{(4)}\right\} = \min\{26, 37+11\} = 26\\
w_{35}^{(5)} &= \min\left\{w_{35}^{(4)}, w_{35}^{(4)} + w_{55}^{(4)}\right\} = \min\{37, 37+0\} = 37\\
w_{45}^{(5)} &= \min\left\{w_{45}^{(4)}, w_{45}^{(4)} + w_{55}^{(4)}\right\} = \min\{11, 11+0\} = 11
\end{aligned}$$

Daraus ergibt sich die Matrix:

$_i{}^j$	1	2	3	4	5
1		13	19	10	21
2			6	20	31
3				26	37
4					11
5					

4.2.3 Eingeschränktes Rucksackproblem

Wir definieren $R(k,g)$ mit $1 \leq k \leq n$ und $0 \leq g \leq G$ als das eingeschränkte Rucksackproblem, bei dem die ersten k Objekte betrachtet werden und das Gewichtslimit g beträgt.

- $F(k,g)$ sei der Nutzen einer optimalen Lösung für $R(k,g)$ mit den Randwerten $F(0,g) = F(k,0) = 0$ für $g \geq 0$ und $F(k,g) = -\infty$ für $g < 0$.
- $M(k,g) \subset \{1, \cdots, k\}$ sei eine optimale Lösung für $R(k,g)$ mit den Randwerten $M(k,g) = \emptyset$ für $k = 0$ oder $g < 0$.

Betrachte $R(k,g)$. Für das k-te Objekt gibt es zwei Möglichkeiten: Einpacken oder Liegenlassen. Für $F(k,g)$ folgt dann

$$F(k,g) = \max\{F(k-1,g), F(k-1,g-g_k) + a_k\}$$

Falls $F(k,g) = F(k-1,g)$ kann $M(k,g) = M(k-1,g)$ gewählt werden, ansonsten $M(k,g) = M(k-1,g-g_k) \cup \{k\}$.

Beispiel:

Objekt	Nutzen	Gewicht
1	3	7
2	8	10
3	12	2
4	4	4
5	1	6
6	8	8
7	12	2

Gewichtsbeschränkung $G = 20$

k G	0	1	2	3	4	5	6	7	8	9	10
0	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}
1	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	3 {1}	3 {1}	3 {1}	3 {1}
2	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	3 {1}	3 {1}	3 {1}	8 {2}
3	0 {0}	0 {0}	12 {3}	12 {3}	12 {3}	12 {3}	12 {3}	12 {3}	12 {3}	15 {1,3}	15 {1,3}
4	0 {0}	0 {0}	12 {3}	12 {3}	12 {3}	12 {3}	16 {3,4}	16 {3,4}	16 {3,4}	16 {3,4}	16 {3,4}
5	0 {0}	0 {0}	12 {3}	12 {3}	12 {3}	12 {3}	16 {3,4}	16 {3,4}	16 {3,4}	16 {3,4}	16 {3,4}
6	0 {0}	0 {0}	12 {3}	12 {3}	12 {3}	12 {3}	16 {3,4}	16 {3,4}	16 {3,4}	16 {3,4}	20 {3,6}
7	0 {0}	0 {0}	12 {3}	12 {3}	24 {3,7}	24 {3,7}	24 {3,7}	24 {3,7}	28 {3,4,7}	28 {3,4,7}	28 {3,4,7}

Fülle eine Tabelle mit n Zeilen und G Spalten spaltenweise von links nach rechts mit Paaren $(F(k,g), M(k,g))$ auf. Die Lösung steht in $(F(n,G), M(n,G))$ (rechte untere Ecke).

$k\,G$	11	12	13	14	15	16	17	18	19	20
0	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}	0 {0}
1	3 {1}	3 {1}	3 {1}	3 {1}	3 {1}	3 {1}	3 {1}	3 {1}	3 {1}	3 {1}
2	8 {2}	8 {2}	8 {2}	8 {2}	8 {2}	8 {2}	11 {1, 2}	11 {1, 2}	11 {1, 2}	11 {1, 2}
3	15 {1, 3}	20 {2, 3}	20 {2, 3}	20 {2, 3}	20 {2, 3}	20 {2, 3}	20 {2, 3}	20 {2, 3}	23 {1, 2, 3}	23 {1, 2, 3}
4	16 {3, 4}	20 {2, 3}	20 {2, 3}	20 {2, 3}	20 {2, 3}	24 {2, 3, 4}	24 {2, 3, 4}	24 {2, 3, 4}	24 {2, 3, 4}	24 {2, 3, 4}
5	16 {3, 4}	20 {2, 3}	20 {2, 3}	20 {2, 3}	20 {2, 3}	24 {2, 3, 4}	24 {2, 3, 4}	24 {2, 3, 4}	24 {2, 3, 4}	24 {2, 3, 4}
6	20 {3, 6}	20 {3, 6}	20 {3, 6}	24 {3, 4, 6}	24 {3, 4, 6}	24 {3, 4, 6}	24 {3, 4, 6}	24 {3, 4, 6}	24 {3, 4, 6}	28 {2, 3, 6}
7	24 {3, 7}	32 {3, 6, 7}	32 {3, 6, 7}	32 {3, 6, 7}	32 {3, 6, 7}	36 {3, 4, 6, 7}	36 {3, 4, 6, 7}	36 {3, 4, 6, 7}	36 {3, 4, 6, 7}	36 {3, 4, 6, 7}

4.2.4 Single source-all paths (Dijkstra)

Berechne für einen Startknoten s und für alle Knoten v kürzeste Wege von s nach v. Die Kantenbewertungen seien nicht negativ, also $c(i,j) \geq 0$. Suche nacheinander die Knoten $v_1, \cdots, v_n$, so daß die für Länge l_i der

kürzesten Wege von s nach v_i gilt: $l_1 \leq l_2 \leq \cdots \leq l_n$. Nehmen wir an, wir kennen $v_1, \cdots, v_i$ und $l_1, \cdots, l_i$. Daraus können wir das Problem zur Berechnung von v_{i+1} und l_{i+1} folgendermaßen einschränken. Für alle Knoten $v \in V_j := V - \{v_1, \cdots, v_j\}$ setzen wir

$$d(v) \quad := \quad \min\{l_i + c(v_i, v) \mid 1 \leq i \leq j\}$$

Dies sind die Kosten des kürzesten Weges von s nach v über die möglichen Zwischenpunkte aus $\{v_1, \cdots, v_j\}$. Setze zu Beginn $s = v_1$ und $l_1 = 0$.
Beispiel: Sei der Graph aus 4.2.2 auf Seite 54 gegeben und unser Startknoten $s = 1$.

- $j = 1$
  Für V_j folgt daraus: $V_j = \{2, 3, 4, 5\}$, da $v_1 = 1 = s$ ist.

$$\begin{aligned} d(2) &= \min\{l_1 + c(v_1, 2)\} = \min\{0 + 13\} = 13 \\ d(3) &= \min\{l_1 + c(v_1, 3)\} = \min\{0 + \infty\} = \infty \\ d(4) &= \min\{l_1 + c(v_1, 4)\} = \min\{0 + 10\} = 10 \\ d(5) &= \min\{l_1 + c(v_1, 5)\} = \min\{0 + \infty\} = \infty \end{aligned}$$

  Wähle als $v_2 = 4$ und als $l_2 = 10$, da diese Entfernung in diesem Durchlauf minimal ist. Die kürzesten Wege von s zu den Knoten ergeben sich im Moment zu

$$d(1) = 0,\ d(2) = 13,\ d(3) = \infty,\ d(4) = 10,\ d(5) = \infty$$

- $j = 2$
  Für V_j folgt daraus: $V_j = \{2, 3, 5\}$, da $v_1 = 1$, $v_2 = 4$.

$$\begin{aligned} d(2) &= \min\left\{ \begin{array}{l} l_1 + c(v_1, 2) \\ l_2 + c(v_2, 2) \end{array} \right\} = \min\left\{ \begin{array}{l} 0 + 13 \\ 10 + \infty \end{array} \right\} = 13 \\ d(3) &= \min\left\{ \begin{array}{l} l_1 + c(v_1, 3) \\ l_2 + c(v_2, 3) \end{array} \right\} = \min\left\{ \begin{array}{l} 0 + \infty \\ 10 + 72 \end{array} \right\} = 82 \\ d(5) &= \min\left\{ \begin{array}{l} l_1 + c(v_1, 5) \\ l_2 + c(v_2, 5) \end{array} \right\} = \min\left\{ \begin{array}{l} 0 + \infty \\ 10 + 11 \end{array} \right\} = 21 \end{aligned}$$

  Wähle als $v_3 = 2$ und als $l_3 = 13$, da diese Entfernung in diesem Durchlauf minimal ist. Die kürzesten Wege von s zu den Knoten ergeben sich im Moment zu

$$d(1) = 0,\ d(2) = 13,\ d(3) = 82,\ d(4) = 10,\ d(5) = 21$$

- $j = 3$
  Für V_j folgt daraus: $V_j = \{3, 5\}$, da $v_1 = 1$, $v_2 = 4$, $v_3 = 2$.

$$\begin{aligned} d(3) &= \min\left\{ \begin{array}{l} l_1 + c(v_1, 3) \\ l_2 + c(v_2, 3) \\ l_3 + c(v_3, 3) \end{array} \right\} = \min\left\{ \begin{array}{l} 0 + \infty \\ 10 + 72 \\ 13 + 6 \end{array} \right\} = 19 \\ d(5) &= \min\left\{ \begin{array}{l} l_1 + c(v_1, 5) \\ l_2 + c(v_2, 5) \\ l_3 + c(v_3, 5) \end{array} \right\} = \min\left\{ \begin{array}{l} 0 + \infty \\ 10 + 11 \\ 13 + \infty \end{array} \right\} = 21 \end{aligned}$$

  Wähle als $v_4 = 3$ und als $l_4 = 19$, da diese Entfernung in diesem Durchlauf minimal ist. Die kürzesten Wege von s zu den Knoten ergeben sich im Moment zu

$$d(1) = 0,\ d(2) = 13,\ d(3) = 19,\ d(4) = 10,\ d(5) = 21$$

- $j = 4$
  Für V_j folgt daraus: $V_j = \{5\}$, da $v_1 = 1$, $v_2 = 4$, $v_3 = 2$, $v_4 = 3$.

$$d(5) \quad = \quad \min\left\{ \begin{array}{l} l_1 + c(v_1, 5) \\ l_2 + c(v_2, 5) \\ l_3 + c(v_3, 5) \\ l_4 + c(v_4, 5) \end{array} \right\} = \min\left\{ \begin{array}{l} 0 + \infty \\ 10 + 11 \\ 13 + \infty \\ 19 + 100 \end{array} \right\} = 21$$

  Wähle als $v_5 = 5$ und als $l_5 = 21$, da diese Entfernung in diesem Durchlauf minimal ist. Die kürzesten Wege von s zu den Knoten ergeben sich insgesamt zu

$$d(1) = 0,\ d(2) = 13,\ d(3) = 19,\ d(4) = 10,\ d(5) = 21$$

4.3 Backtracking

Greedy Algorithmen berechnen eine Lösung Stück für Stück. Dabei gehen sie stur vor. Eine einmal getroffene Entscheidung wir nie zurückgenommen. Eine vollständige Suche ist auf eine andere Weise stur, da alle Möglichkeiten durchprobiert werden.
Beim Backtracking werden alle Lösungsversuche durchprobiert und die einzelnen Lösungen aber Stück für Stück konstruiert. Wenn sich bei einer Teillösung bereist herausstellt, dass sie zu keiner korrekten Lösung fortgesetzt werden kann, wird diese Konstruktion abgebrochen. Mit einer Entscheidung wird also unter Umständen eine Entscheidung über sehr viele Lösungen getroffen.

4.3.1 Anwendungsbeispiele:

- Damenproblem
  Für ein $n \times n$ Schachbrett soll entschieden werden, ob n Damen so platziert werden können, so dass keine eine andere schlagen kann.
- Springerproblem
  Kann ein Springer auf einem $n \times n$ Schachbrett so hüpfen, dass er jedes Feld genau einmal besucht.
- Spiele mit vollständiger Information (siehe 4.3.2)

4.3.2 $\alpha - \beta$-Pruning

4.3.2.1 Vorgeschichte: *Wir beschränken uns auf Zwei-Personen-Spiele.*
Man hat ein Spiel mit vollständiger Information wie Schach oder Dame, aber nicht wie Skat oder Doppelkopf[17].
Zu jedem dieser Spiele gehört ein vollständiger Spielbaum.
Die Wurzel des Baumes entspricht dem Spielbeginn, jeder Knoten einer Spielsituation. Zudem beschreibt jeder Knoten, wer am Zug ist. Für jeden gültigen Zug in der aktuellen Spielsituation s, bekommt der Knoten v_s entsprechend Kinder. Die Blätter gehören zu beendeten Spielen und erhalten eine Bewertung $c \in \mathbb{R}$.
Die Bewertung $c(v)$ bedeutet für

- Spieler 1 eine Strategie, die ihm mindestens einen Gewinn von $c(v)$ sichert.
- Spieler 2 eine Strategie, die den Verlust auf höchstens $c(v)$ begrenzt.

Die Bewertung an den Blättern hat ebenfalls diese Bedeutung.
Sind nun am den Kindern $v_1, \cdots, v_k$ von v die Kosten $c(v_1), \cdots, c(v_k)$ bekannt, dann läßt sich $c(v)$ berechnen durch

$$\begin{aligned} \text{I am Zug}: &\quad c(v) = \max\{c(v_1), \cdots, c(v_k)\} \\ \text{II am Zug}: &\quad c(v) = \min\{c(v_1), \cdots, c(v_k)\} \end{aligned}$$

Mit einem *Postorder-Durchlauf* können alle Bewertungen berechnet werden. Man berechnet also Werte für alle Knoten (bis auf Blätter), so ein Spielbaum kann aber gewaltige Dimensionen annehmen. Man überlege sich z.B. die möglichen Stellungen beim Schachspielen.

4.3.2.2 Was macht nun $\alpha - \beta$-Pruning $\alpha - \beta$-Pruning ist eine spezielle Form des Backtracking, mit der Rechenzeit gespart werden kann[18].
Zuerst geben wir den Knoten Namen und eine vorläufige Bewertung $c^*(v)$

- Knoten, an denen Spieler I zieht $\rightarrow$ Max-Knoten, $c^*(v) = -\infty$
- Knoten, an denen Spieler II zieht $\rightarrow$ Min-Knoten, $c^*(v) = +\infty$

So spart man Rechenzeit:

1. Sei v ein Max-Knoten und v_i ein Kind von v. Falls $c(v_i)$ bekannt ist, so setze $c^*(v) := \max\{c^*(v), c(v_i)\}$. Dann streiche v_i.

[17] man kennt dabei die Kartenverteilung nicht
[18] Z.B. werden Teilbäume, die keinen besseren Gewinn versprechen, nicht berechnet

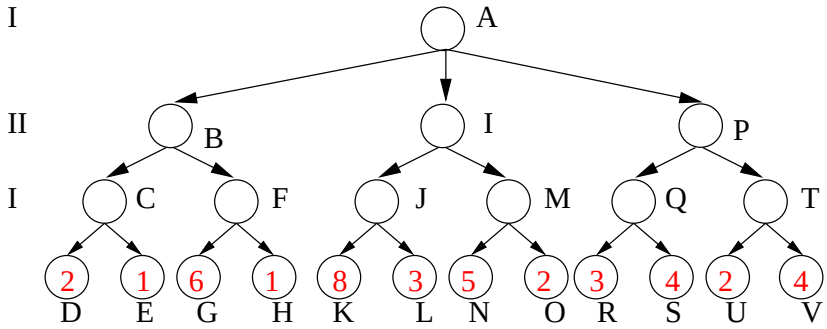
2. Sei v ein Max-Knoten und v_i ein Kind von v und Min-Knoten. Falls $c^*(v_i) \leq c^*(v)$, so streiche den Teilbaum mit Wurzel v_i.
   Erklärung: $c(v_i)$ kann die Bewertung am Knoten v nicht mehr verändern

$$c(v_i) \leq c^*(v_i) \leq c^*(v) \leq c(v)$$

3. Für Min-Knoten gelten analoge Regeln.

Beispiel:

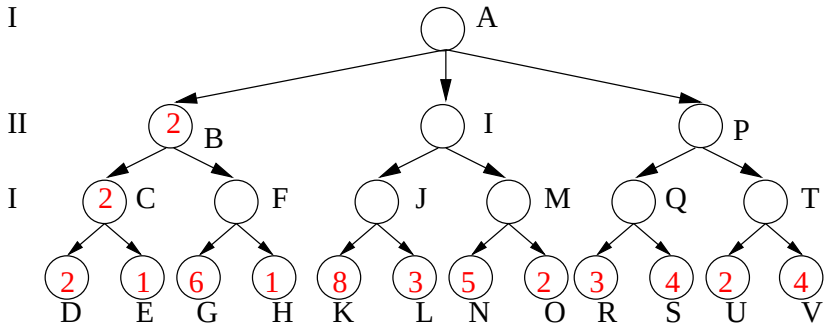
Gegeben sei folgender Spielbaum:



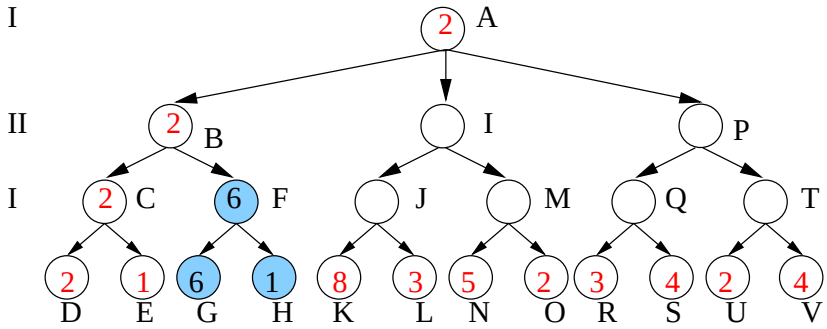
Es wird der Postorder-Durchlauf gestartet. Zunächst wird $c(C) = 2$ berechnet, da

$$\begin{aligned} c^*(C) &= \max\{c^*(C), c(D)\} = \max\{-\infty, 2\} = 2 \\ &\text{und dann} \\ c(C) &= \max\{c^*(C), c(E)\} = \max\{2, 1\} = 2 \end{aligned}$$

Es folgt $c^*(B) = 2$, da $c(C)$ bekannt ist.



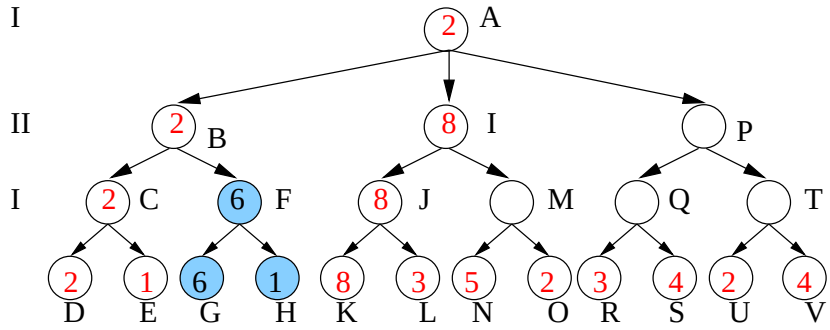
Der nächste betrachtete Knoten ist F, für den $c^*(F) = 6$ folgt. Da $c^*(F) \geq c^*(B)$ gilt, wird der Teilbaum mit Wurzel F nicht weiter betrachtet. Spieler II wird niemals aus Spielsituation B in Spielsituation F wechseln, sondern lieber in B, wo der Verlust kleiner ist. Die abschließenden Kosten für B sind $c(B) = 2$. Es können die vorläufigen Kosten für Knoten A aktualisiert werden zu $c^*(A) = 2$.



Danach wird $c(J) = 8$ berechnet, da

$$\begin{aligned} c^*(J) &= \max\{c^*(J), c(K)\} = \max\{-\infty, 8\} = 8 \\ &\text{und dann} \\ c(J) &= \max\{c^*(J), c(L)\} = \max\{8, 3\} = 8 \end{aligned}$$

Es folgt $c^*(I) = 8$, da $c(J)$ bekannt ist. *Wir aktualisieren aber nicht die vorläufigen von Knoten A, weil nur die vorläufigen Kosten $c^*(I)$ bekannt sind und nicht $c(I)$.*



Nun berechnet man $c(M) = 5$, da

$$\begin{aligned} c^*(M) &= \max\{c^*(M), c(N)\} = \max\{-\infty, 5\} = 5 \\ &\text{und dann} \\ c(M) &= \max\{c^*(M), c(O)\} = \max\{5, 2\} = 5 \end{aligned}$$

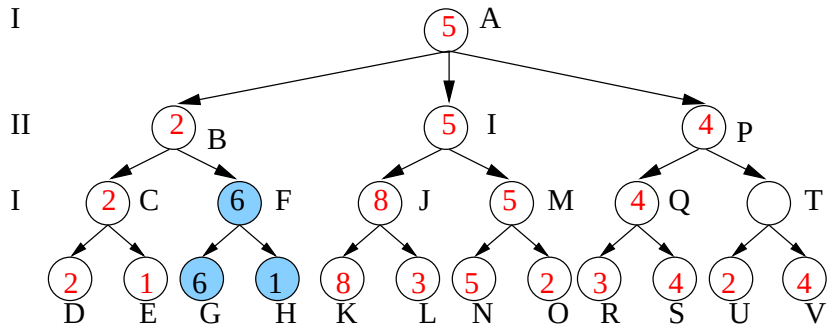
Es folgt unittlerbar $c(I) = 5$, da

$$c^*(I) = \min\{c^*(I), c(M)\} = \min\{8, 5\} = 5$$

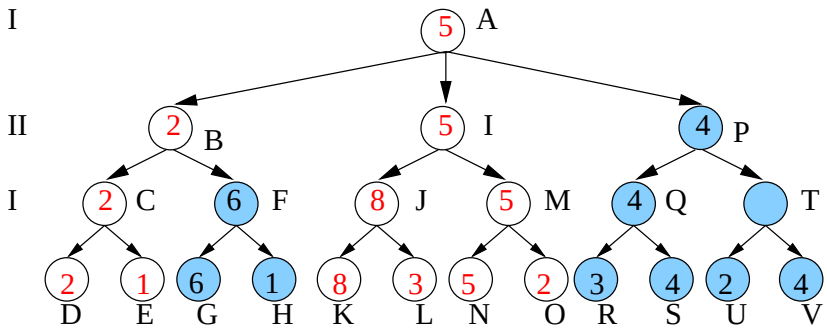
Wir verwenden hier das Minimun, da wir an einem Min-Knoten sind, vorher sind nur Max-Knoten betrachtet worden.
Weiter geht es mit der Berechnung von $c(Q) = 4$ durch

$$\begin{aligned} c^*(Q) &= \max\{c^*(Q), c(R)\} = \max\{-\infty, 3\} = 3 \\ &\text{und dann} \\ c(Q) &= \max\{c^*(Q), c(S)\} = \max\{3, 4\} = 4 \end{aligned}$$

und, da nun $c(Q)$ bekannt ist, mit der Festlegung von $c^*(P) = 4$.



Da $c^*(P) \leq c^*(A)$, wird der Teilbaum mit Wurzel P gestrichen. Dies geschieht, obwohl nur ein Teil davon betrachtet wurde. Hier kann man die Einsparung der Rechenzeit erkennen. Es folgt insgesamt, dass $c(A) = 5$ ist. Die optimale Spielstrategie besteht in der Wahl der Knoten v_i für die $c(v) = c(v_i)$ gilt.



4.4 Branch and Bound Methoden

Besteht im Allgemeinen aus drei Teilen

1. Upper Bound
   Berechne eine (möglichst gute) obere Grenze U für die Lösung des Problems.
2. Lower Bound
   Berechne eine (möglichst gute) untere Schranke L für die Lösung des Problems. Dabei soll eine zulässige Lösung mit dem Wert L berechnet werden.
3. Branching
   Zerlegung des Problems in (möglichst) disjunkte Teilprobleme

Die disjunkten Teilproblem ergeben sich, z.B. beim Rucksackproblem durch Einpacken[19] bzw Liegenlassen[20] des j - ten Objekts. Dabei muss man folgendes beachten

- Liegenlassen
  Streiche das Objekt j aus der Liste
- Einpacken
  - Streiche das Objekt j aus der Liste
  - Setze die Gewichtsschranke um g_j herab
  - Addiere zu jeder Lösung, unteren und oberen Schranke den Nutzen a_j

Als das oben erwähnte j-te Objekt wählt man das Objekt, welches bei der Berechnung der Upper Bound nur zu einem Teil in den Rucksack gepackt wurde.

4.4.1 Allgemeiner Branch-and-Bound-Algorithmus für das Rucksackproblem:

1. Initialisiere den BBB[21] durch einen Baum mit einem Knoten, der das Gesamtproblem P_0 darstellt.
2. Berechne die aktuellen Werte L und U als Maximum aller L_j bzw. U_j an den Blättern des BBB.
3. Ist $L = U$ ist die zulässige Lösung, die zu einer unteren Schranke gehört eine optimale Lösung.
4. Ist $L < U$ wird das Blattproblem P_j an seinem kritischen Objekt in zwei Teilprobleme zerlegt, die im BBB Kinder von P_j werden. Gehe zu Schritt (2).

Die Tiefe des BBB ist durch n beschränkt.

[19] Inklusion
[20] Exklusion
[21] Branch-and-Bound-Baum