Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Michel Gatto

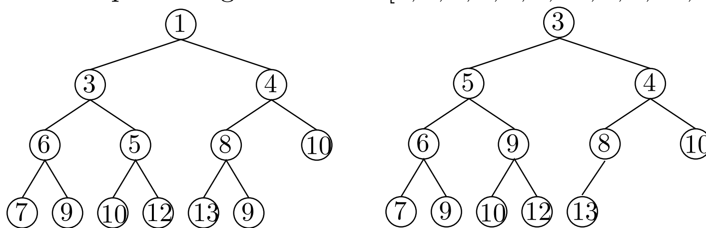
# Beispiellösung
# Datenstrukturen und Algorithmen

7. März 2007

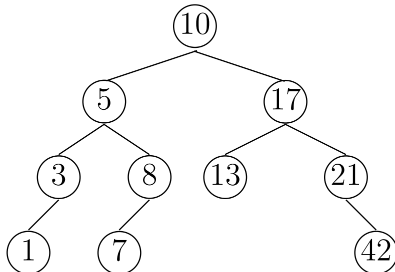
*Hinweis:* Vorsicht, liebe Studierende: diese Musterlösung dient nur dazu, Ihnen eine Kontrolle der eigenen Lösung zu erleichtern. Dementsprechend erläutert sie nur einen Lösungsweg oder eine Lösung (manchmal die einzige, manchmal eine von mehreren), und sie ist manchmal ausführlicher als in der Prüfung verlangt.

**Aufgabe 1:**

a) Der Heap hat folgende Form: $[3, 5, 4, 6, 9, 8, 10, 7, 9, 10, 12, 13]$.



b) Nach dem Balancieren sieht der AVL-Baum folgendermassen aus:



c) Natural mergesort vereinigt jeweils zwei benachbarte längste Sequenzen aufsteigender Zahlen (zwei Runs). Das Array wird deshalb

von:

| 12 | 41 | 29 | 37 | 12 | 22 | 5 | 10 | 12 | 14 | 17 | 47 | 4 | 51 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

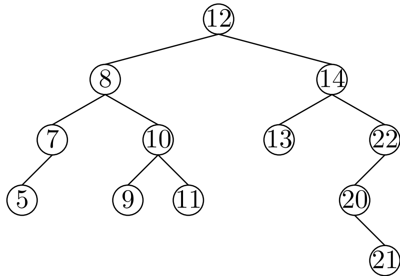
zu :

| 12 | 29 | 37 | 41 | 5 | 10 | 12 | 12 | 14 | 17 | 22 | 47 | 4 | 51 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

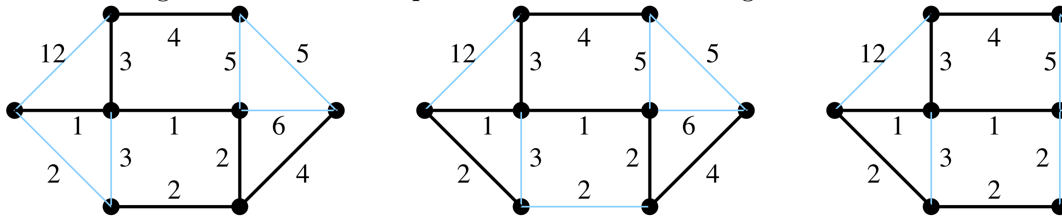
d) Die ausgefüllte Hashtabelle ist:

| Werte: | 41 | 23 | 02 | 03 | 10 | 49 | 20 | 18 | 19 | 31 | 05 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Index : | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

e) Der Baum ist:



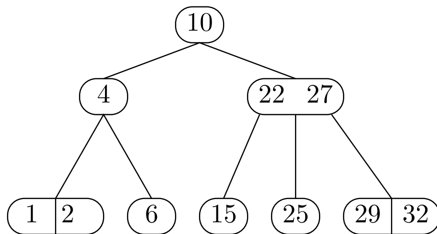

f) Die drei möglichen minimalen Spannbäume sind in der Figur schwarz markiert.



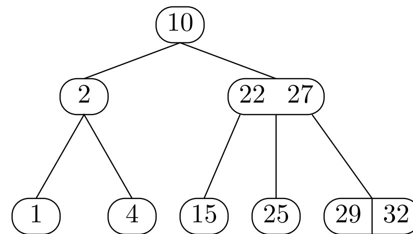

g) Eine mögliche Traveriserung ist: a,c,h,g,f,k,i,e,b,d.

h) Die zwei B-Bäume sehen wie folgt aus:

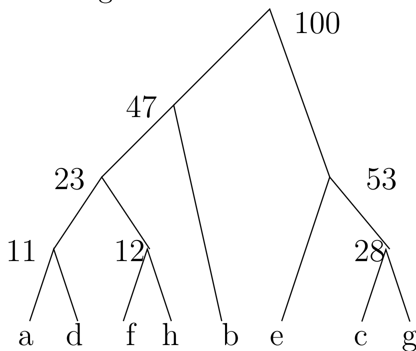
Einfügen von 29




Löschen von 6




i) Ein möglicher Baum ist:



**Aufgabe 2:**

a) Die Reihenfolge ist: $\sum_{i=0}^{n} \frac{1}{2^i}, \frac{n}{\log n}, n \log n, n\sqrt{n}, \sum_{i=0}^{n} i, \Pi_{i=1}^{n} i, n^n$.

b) Die Rekursionsgleichung evaluiert, mittels Teleskopieren auf $T(n) = 10n \log_2 n + 10n - 10$.
Induktionsbeweis: Verankerung: $T(1) = 10 \log_2(1) + 10 \cdot 1 - 10 = 0 + 10 - 10 = 0$.
Schritt:

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + 10n + 10 = 2(10\frac{n}{2} \log_2(\frac{n}{2}) + 10\frac{n}{2} - 10) + 10n + 10 \\ &= 10nlog_2(\frac{n}{2}) + 10n - 20 + 10n + 10 = 10n \log_2(n) - 10nlog_2 2 + 10n + 10n - 10 \\ &= 10n \log_2 n + 10n - 10. \end{aligned}$$

c) $\Theta(n)$.

d) $\Theta(n)$.

e) $\Theta(n \log n)$.

**Aufgabe 3:**

a) Die einfachste Methode ist, ein Array zu benutzen. Die Idee ist, die `resorts` geschickt zu sortieren, damit wir im Array der `resorts` effizient nach $k$ suchen können und anhand des Array-Index bei dem die Suche endet feststellen können, wieviele Elemente grösser als $k$ sind. Zu diesem Zweck sortieren wir die Skigebiete aufsteigend nach Gesamtlänge `laenge` der Pisten.

b) Man implementiert die binäre Suche im sortierten Array so, dass sie diese zwei Eigenschaften erfüllt: eine erfolglose Suche nach $k$ endet bei der nächstgrösseren Zahl $k'$ an Kilometer; bei einer Suche nach einem mehrmals vorkommenden Element endet die Suche beim Element mit kleinstem Index. Sei $i$ der Index bei dem die Suche nach $k$ endet in einem von 1 bis `resorts.count` indizierten Array. Dann wissen wir, dass alle Skigebiete, die einen Array-Index grösser oder gleich $i$ haben, mindestens $k$ Kilometer Pisten haben. Die Anzahl davon ist `resorts.count`$-i+1$ Skigebiete.

```
number_of_resorts( k : INTEGER) : INTEGER is
local
  search_ends: INTEGER
do
search_ends:= binary_search(resorts,resorts.lower, resorts.upper, k)
Result := resorts.count - search_ends +1
end

binary_search(s: ARRAY [ COMPARABLE ], low, high, search): INTEGER  is
local
  m: INTEGER
do
if (low == high ) -- kein Element gefunden
```

```
 if (s[low] >= k) -- erfüllt Anforderungen
   Result := low
 else -- ansonsten macht's der nächstgrößere Nachbar
   Result := low + 1
 end
else
  m := low + (high - low ) //2
  if ( s[m]<k)
     Result := binary_search(s,m+1, high,k)
  else if (s[m] > k )
     Result := binary_search(s,low,m-1,k)
  else  -- s[m]=k, schauen, dass es keine andere mit k gibt
     Result := binary_search(s,low,m,k)
  end
end
```

Die Binäre Suche hat eine Laufzeit von $O(\log n)$, das Array kann in Zeit $O(n \log n)$ sortiert werden (z.B. mit mergesort).

c) Für die Erweiterung kann man einen AVL-Baum mit Zusatzinformation benutzen. Als Schlüssel benutzen wir die Pisten-Kilometer. Zusätzlich speichern wir in jedem Knoten des Baumes die Anzahl Knoten in seinem Teilbaum. Die Anfrage nach der Anzahl Skigebiete mit mehr als $k$ Pisten-Kilometer beantwortet man wie folgt. Man sucht im Baum wie gewohnt nach dem Schlüssel $k$. Jedes Mal, wo man in der Suche zum linken Sohn geht, inkrementiert man einen (am Anfang auf Null initialisierten) Zähler um die Anzahl Knoten des rechten Bruders plus eins (den Vater des linken Sohn, der auch mehr als $k$ Pisten-Kilometer hat). Die Suche endet, wenn man einen NULL-Zeiger erreicht, und man gibt den Wert des Zählers zurück. Beim Einfügen in den AVL-Baum inkrementiert man bei jedem besuchten Knoten dessen Zähler der Knoten im dort gewurzelten Baum um 1, beim Entfernen hingegen (unter der Annahme, dass das zu entfernende Skigebiet auch in der Menge vorkommt) dekrementiert man jeweils die Zähler der besuchten Knoten um 1. Es bleibt noch, die Strukturveränderungen zu beschreiben (Rotationen). Beim Rotieren nach links eines Knotens $b$ mit Vater $a$ ersetzt man die Anzahl Schlüssel im Teilbaum mit Wurzel $b$ (der jetzt $a$ als linken Sohn hat) mit der von $a$, und setzt die von $a$ als 1+(Anzahl Schlüssel im Teilbaum des linken Sohnes)+(Anzahl Schlüssel im Teilbaum des rechten Sohnes). Beim Rotieren nach rechts eines Knotens $b$ mit Vater $a$ erfolgen die gleichen Änderungen.

   Die Laufzeiten sind asymptotisch die, die für die normalen AVL-Operationen benötigt sind: $O(\log n)$ zum Einfügen, Suchen und Löschen.

**Aufgabe 4:**

a) Die eigentliche Frage, die wir beatworten müssen, ist welche Fläche jedes Kübels nach oben nicht von einem anderen Kübel abgedeckt ist, und damit vom Regen getroffen wird. Man verwendet dazy einen Scanline-Ansatz. Die Scanline geht von links nach rechts über die Fassade, und speichert die Kübel die sie schneidet. Zum Abspeichern der geschnittenen Kübel benutzen wir einen AVL-Baum mit der Höhe der Kübel vom Boden als Schlüssel. Die Menge der Punkte, bei der die Scanline hält, sind die Startpunkte `links` und Endpunkte `rechts` der Kübel, die in aufsteigender Reihenfolge durchgegangen werden. Bei jedem Start eines Kübels

wird der Kübel, mit seiner Höhe als Schlüssel, in den AVL-Baum eingefügt. Hat sich das maximale Element in der Scanline geändert, so steht ein neuer Kübel oben. Deshalb muss man die zum Regen exponierte Fläche des vorherigen obersten Kübels, also des Schlüssel-Maximums aktualisieren: man inkrementiert die Fläche Regen, die der Kübel fängt, mit der Länge des Segments ab dem Punkt, bei dem das bisherige Maximum das neue Maximum geworden war (der Wert kann zusätzlich gespeichert werden) bis zum linken Ende des neuen Maximum-Kübels. Der Startpunkt des neuen Maximums wird auf das linke Ende des aktuellen Kübels gesetzt. Endet ein Kübel, so entfernen wir den Kübel aus dem AVL-Baum. Hat sich dabei das Maximum im AVL-Baum geändert (d.h., haben wir soeben das Maximum entfernt), so inkrementiert man die Fläche Regen, die der entfernte Kübel fängt, um die Länge des Segments startend im Startpunkt bei dem dieser Kübel (erneut) das maximale Element geworden ist bis zu seinem rechten Ende. Schlussendlich müssen wir prüfen, ob der entfernte Kübel übergelaufen ist. Dazu multiplizieren wir die Fläche Regen, die der Kübel fängt, mit $r$ und dividieren diesen Wert durch die Länge des Kübel (d.h. mit `rechts-links`). Ist dieser Wert grösser als `seitenwand`, so läuft der Kübel über.

b) Die Laufzeit beträgt $O(n \log n)$: diese stellt sich zusammen aus $O(n \log n)$ zum Sortieren der Endpunkte der Kübel, sowie aus der Zeit, die jeder der $O(n)$ Haltepunkte der Scanline benötigt, um die Scanline zu aktualisieren (d.h. $O(\log n)$ zum Einfügen oder Löschen des Kübels im AVL-Baum, sowie die konstante Arbeit pro Haltepunkt für das Adaptieren der verregneten Fläche).

c) Wir nehmen an, dass `scanline` ein AVL-Baum mit Blattverkettung ist, der zusätzlich das Objekt mit maximalen Schlüssel zurückgeben kann, sowie das nächstkleinere Element im Baum. Hingegen ist `list` eine Priority Queue, die jeweils das Element mit kleinstem Schlüssel zurückgibt. Die Klasse Kübel wird mit einem zusätzlichen Attribut `unbedeckt` erweitert, der die Länge des zum Regen exponierten Teil des Kübel speichert. Es folgt ein sehr detaillierter Pseudocode des Vorgehens.

```
-- Einfügen der Haltepunkte in die Priority queue
from i:= behaelter.lower until i = behaelter.upper do
   list.insert(behaelter[i],behaelter[i].links)
   list.insert(behaelter[i],behaelter[i].rechts)
done
Result := Kein_Ueberlauf
start_max = list.lower.links   --linkeste relevante Punkt als Maximum-anfang
from until list.empty do
  minkey = list.min            --Schlüssel des kleinsten Objekt in der queue
  kuebel = list.remove_min     --nächster relevanten Punkt nehmen
                               --und aus list entfernen
  if (kuebel.left = minkey)    --Haltepunkt ist Startpunkt eines Kübels
  do
     scanline.insert(kuebel,kuebel.hoehe)
     if (scanline.max.getobject == kuebel ) -- neue Kübel ist exponiert
     do -- Füllstand des bisherigen Maximum aktualisieren
        oldmax = scanline.max.previous
        oldmax.unbedeckt = oldmax.unbedeckt + kuebel.links - startmax
        startmax = kuebel.links
     done
  else -- Haltepunkt ist Endpunkt eines Kübels
     if (scanline.max.getobject == kuebel)
```

```
      do -- Kübel ist Maximum, Füllstand aktualisieren
        kuebel.unbedeckt = kuebel.unbedeckt + kuebel.rechts - startmax
        startmax = kuebel.rechts
      done
      if ( (kuebel.unbedeckt * r)/(kuebel.rechts - kuebel.links)
           > kuebel.seitenwand )
      do
        Result := Ueberlauf
      done
      scanline.delete(kuebel.right)
  done
done
```

d) Der Algorithmus hat die selbe Struktur wie der von Teilaufgabe a). Man bestimmt mit dem Scanline-Algorithmus die Fläche $f$ jedes Kübels, die vom Regen getroffen wird. Damit berechnen wir die minimale Menge pro Kubikzentimeter der nötig ist, damit der Kübel überläuft. Das Einfügen eines Kübels in die Scanline bleibt somit genau gleich. Beim Entfernen jedes Kübels, hingegen, führt man (nach dem update der verregneten Fläche) folgende Berechnung durch: Einen Anteil $a = f/($`rechts-links`$)$ des Kübel ist verregnet. Dadurch sind `seitenwand`$/a$ Kubikzentimeter Regen nötig, damit dieser Kübel überläuft. Man vergleicht nun den bisherigen Minimalwert an Regen pro Kubikzentimeter, der irgendeinen Kübel zum Überlaufen bringt mit dem neu berechneten Wert. Falls dieser kleiner ist als bisher, so haben wir temporär ein neues Minimum gefunden, das wir uns zunächst merken, und wir verwerfen die bisherigen Minima. Ist er gleich wie das bisherige Minimum, so speichern wir den Kübel in einer Liste, die ausgegeben werden kann. Am Ende geben wir als Resultat die minimale Regenmenge und den betroffenen (oder die betroffenen) Kübel als Ausgabe. Die Laufzeit beträgt auch in diesem Fall $O(n \log n + k)$: $O(n \log n)$ um die Kübel nach ihren Enden zu sortieren, sowie, für jedes der $O(n)$ Punkte, bei dem die Scanline hält, $O(\log n)$ Zeit zum Einfügen und Entfernen der Kübel von der Scanline. $O(k)$ Zeit braucht man, um die Liste von $k$ betroffenen Kübeln auszugeben.

**Aufgabe 5:**

a) Folgender Algorithmus mit Laufzeit $O(2^n)$ berechnet den Wert der maximalen Schönheit bei einer Auswahl von $n$ Blättern.

```
max_nice(i, L,max)
  if(L < 0)
    return 0
  elseif (i > n)
    return 0
  elseif (L=0)
    return max
  else
    max_mit_i = max_nice(i+1, L-b_i,max+s_i)
    max_ohne_i = max_nice(i+1,L,max)
    return max(max_mit_i, max_ohne_i)
```

b) Man erstellt eine Tabelle $T$ der Grösse $n \times L$, wobei $n$ die Anzahl Blätter der Auswahl ist. Einen Eintrag $T[i,j]$ der Tabelle enthält den Wert der maximalen Schönheit die man durch einer Auswahl der ersten $i$ Blätter erreichen kann bei gesamtbreite $j$ der Ausgewählten Platten. Man füllt der Eintrag der Zelle $(i,j)$ nach folgendem Schemata: $T[i,j] = \max(T[i-1, j-b_i]+s_i, T[i-1,j])$. Der Wert $T[i-1,j-b_i]+s_i$ ist den Wert den man erreicht, wenn man bei Länge $j$ das $i$-te Blatt mit in die Sammlung nimmt. Dieser Wert erreicht man, indem man die maximale Schönheit bei Länge $j - b_i$ unter Betrachtung der ersten $i - 1$ Blätter mit der Schönheit $s_i$ des $i$-ten Objekt inkrementiert. Der Wert $T[i-1, j]$ ist den Wert den man erreicht, wenn man das $i$-te Blatt nicht mit in die Auswahl bei Länge $j$ nimmt, und entsteht deshalb als maximale Schönheit bei Länge $j$ der ersten $i - 1$ Blätter. Wir initialisieren $T[0,0] = 0$. Für die restlichen Einträge mit $i = 0$ nehmen wir für $T[i,j]$ den Wert $-\infty$. Für $j < 0$ nehmen wir immer den Wert $T[i,j] = -\infty$, da $j < 0$ heisst, dass man diesen Wert nicht erreichten kann, wenn man das Objekt $b_j$ in die Auswahl nimmt. Schlussendlich setzen wir $T[i,0] = 0$. Die Tabelle kann aufsteigend zeilenweise oder Spaltenweise aufgefüllt werden. Den Wert der schönsten Auswahl bei Länge $L$ befindet im Eintrag $T[n,L]$. Die Laufzeit beträgt $O(n \cdot L)$, da das Füllen jedes Eintrags konstant viel Aufwand erfordert.

c) Startend im Tabelleneintrag $T[n,L]$, betrachtet man die Einträge der Zellen $T[n-1, L-b_i]$ und $T[n-1,L]$. Falls $T[n-1,L] = T[n,L]$, so wurde das $n$-te Blatt nicht in die Auswahl reingenommen. Falls $T[n-1, L-b_n] = T[n,L] - s_n$, dann wurde das $n$-te Blatt verwendet. Falls beide Fälle möglich sind, so wählt man irgend eine der zwei Zellen. Dann geht man aus dem erreichten Eintrag iterativ weiter, bis man die Zelle $T[0,0]$ erreicht. Die Laufzeit ist $O(n+L)$.

d) Falls man mehrere Wände bedecken will, so macht man eine mehrdimensionale Tabelle, in diesem Fall eine $k+1$-dimensionale Tabelle. Ein Eintrag $T[i, j_1, j_2, \cdots, j_k]$ enthält die maximal erreichbare Schönheit die man mit den ersten $i$ Blätter bei einer Länge von $j_1$ auf der ersten, $j_2$ auf der zweiten, $\cdots$, $j_k$ auf der $k$-ten Wand erreichen kann. Das Füllen der Tabelle erfolgt für jede Dimension analog wie im zweidimensionalen Fall: bei Betracht der $\ell$-ten Wand und des $i$-ten Objekt füllt man das Feld mit:
$T[i, j_1, \cdots, j_\ell, \cdots j_k] = \max\{T[i-1, j_1, \cdots, j_\ell, \cdots j_k], T[i-1, j_1, \cdots, j_\ell - b_i, \cdots j_k]+s_i\}$. Wieder füllt man die Tabelle zeilenweise oder Spaltenweise für jede Dimension, für negative Indizes gelten die gleichen Überlegungen wie für den Zweidimensionalen Fall. Die Laufzeit des Algorithmus ist $O(kn\Pi_{i=1}^{k} L_i)$, für $L_i$ die Länge der $i$-ten Wand, und das Resultat befindet sich im Feld $T[n, L_1, \cdots, L_k]$.