Algorithmen und Datenstrukturen

Große Übung #4

Christian Scheffer   Jan-Marc Reinhardt

17.12.2015

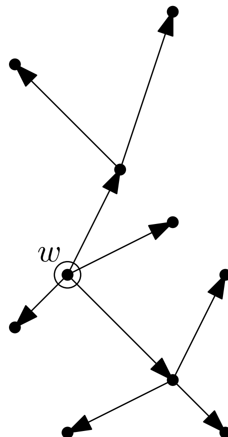
Programm für heute

1. Binäre Suchbäume, insbesondere AVL-Bäume
2. Dynamische Arrays

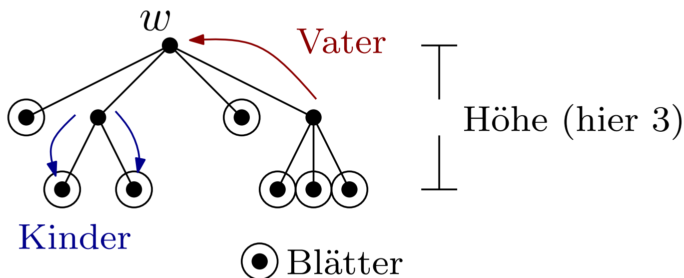
Binäre Suchbäume

Definition: Gewurzelter Baum

- Gerichteter Baum $T = (V, A)$ mit
- ausgezeichnete Wurzel w mit Eingangsgrad 0 und
- Eingangsgrad 1 für alle $v \in V \setminus \{w\}$



Darstellung & Bezeichnungen



Eigenschaften 1

- Jeder gerichtete Baum hat einen Knoten mit Eingangsgrad 0 (d.h. eine potentielle Wurzel).

Beweis

Annahme: Es existiert ein gerichteter Baum $T = (V, A)$ ohne Knoten mit Eingangsgrad 0. Wähle $v \in V$. Betrachte die Sequenz $v = v_0, v_1, v_2, \ldots, v_n$, wobei v_i der Vater von v_{i-1} ist (für alle $1 \leq i \leq n$). Mindestens ein Knoten muss in dieser Sequenz zweimal auftreten. Sei u der Knoten, der zuerst zweimal auftritt. Betrachte die Teilsequenz vom ersten Auftreten von u zum zweiten Auftreten in umgekehrter Reihenfolge. Dies ist ein Kreis in T – ein Widerspruch dazu, dass T ein Baum ist. □

Eigenschaften 2

- In einem gewurzelten Baum T gibt es einen eindeutigen Weg von der Wurzel zu jedem anderen Knoten.

(Teil-)Beweis

Annahme: Für einen Knoten v gibt es zwei unterschiedliche Wege $A = (w = a_0, a_1, \ldots, a_n = v)$ und $B = (w = b_0, b_1, \ldots, b_m = v)$ von der Wurzel w zu v. Betrachte den vom Ende gezählt letzten Knoten u, der in beiden Sequenzen gleich ist (d.h. $a_i = b_j = u$, $i, j > 0$). Dann hat u zwei unterschiedliche Väter, a_{i-1} und b_{j-1}, im Widerspruch dazu, dass T ein gewurzelter Baum ist. □

Selbst überlegen: Warum gibt es überhaupt für jeden Knoten v einen Weg von w zu v?

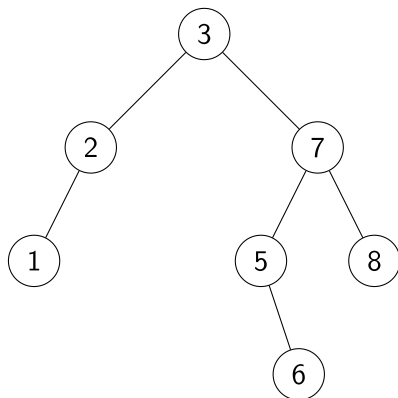
Binäre Suche

Suche nach 10!

2	5	7	11	15	16	23	34	35

- Voraussetzung: Elemente sortiert (Totalordnung)
- Resultat: Suche hat $\mathcal{O}(\log n)$ Laufzeit
- Aber: Arrays erlauben kein (schnelles) Einfügen

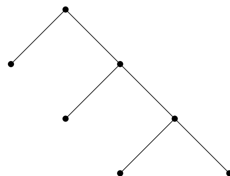
Binäre Suchbäume

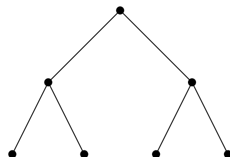


- Gewurzelter Baum mit
- höchstens zwei Kindern pro Knoten (linkes/rechtes Kind),
- Schlüsseln, im linken Teilbaum $\leq$, im rechten $>$.

Definitionen

- In einem *vollen* binären Baum hat jeder Knoten entweder 0 oder 2 Kinder.
- Ein binärer Baum ist *vollständig*, wenn er voll ist und alle Blätter dieselbe Entfernung zur Wurzel haben.





Satz

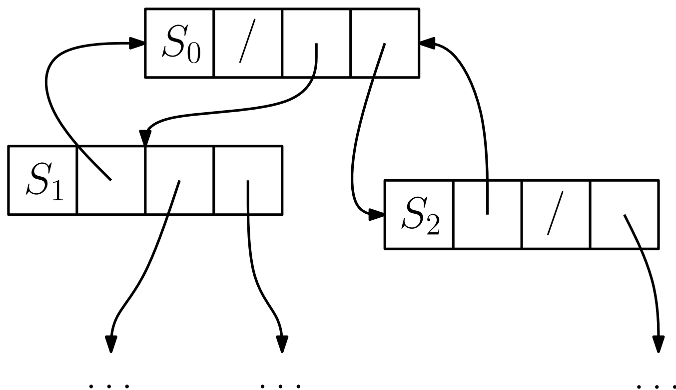
- Ein vollständiger binärer Baum der Höhe h hat 2^{h-1} Blätter.

Beweis per Induktion über h

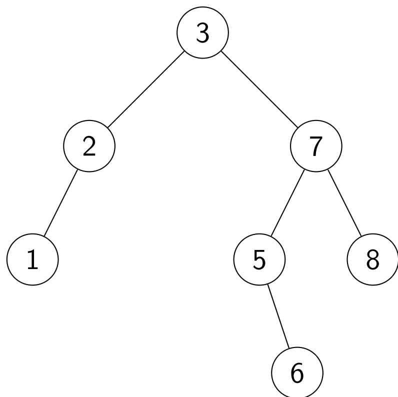
Für $h = 1$ hat ein binärer Baum nur einen Knoten, der gleichzeitig ein Blatt ist. Angenommen, jeder vollständige binäre Baum der Höhe $h - 1$ hat 2^{h-2} Blätter. Einen vollständigen binären Baum der Höhe h erhält man aus einem der Höhe $h - 1$, indem man jedem Blatt 2 Kinder gibt. Dies ergibt $2 \cdot 2^{h-2} = 2^{h-1}$ Blätter. □

Selbst überlegen: Ein vollständiger binärer Baum der Höhe h hat $2^h - 1$ Knoten.

Speicherung im Rechner

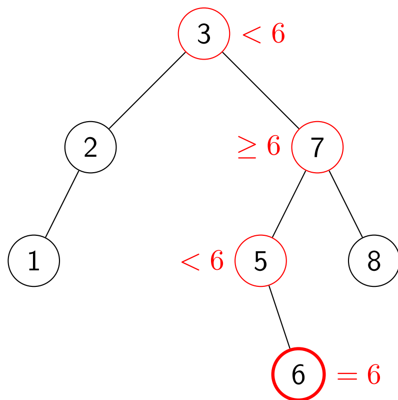


Traversierung

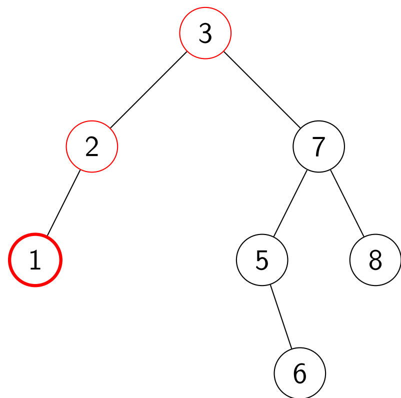


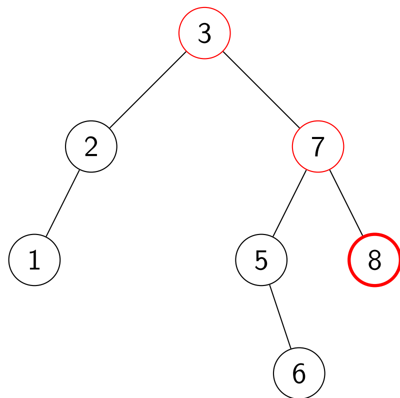
- in-order: 1, 2, 3, 5, 6, 7, 8
- pre-order: 3, 2, 1, 7, 5, 6, 8
- post-order: 1, 2, 6, 5, 8, 7, 3

Operationen: `Search(6)`

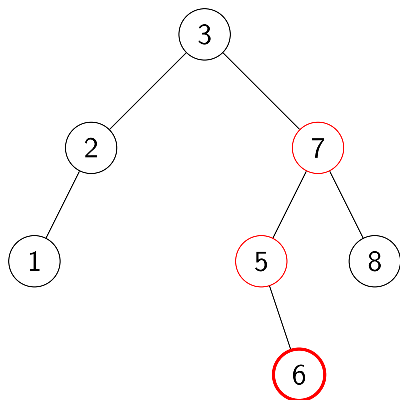


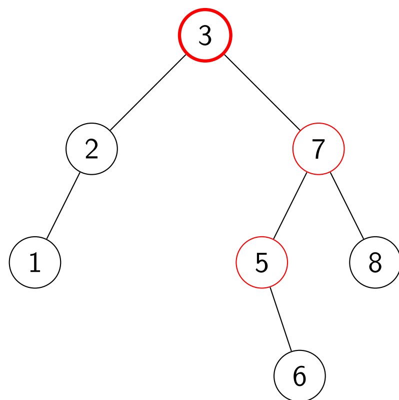
Operationen: `Minimum`, `Maximum`

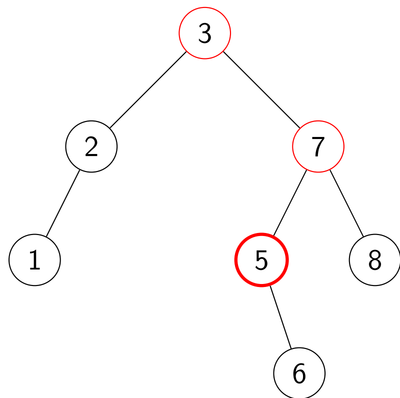


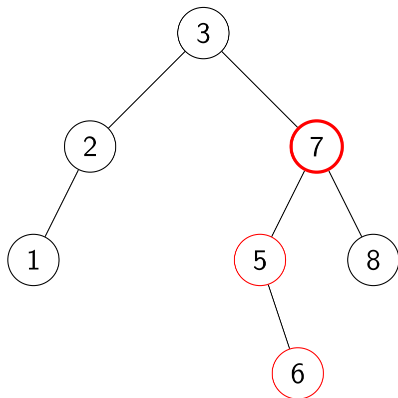


Operationen: `Predecessor(7)`, `Predecessor(5)`

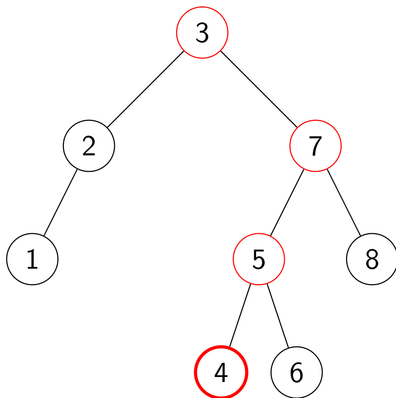




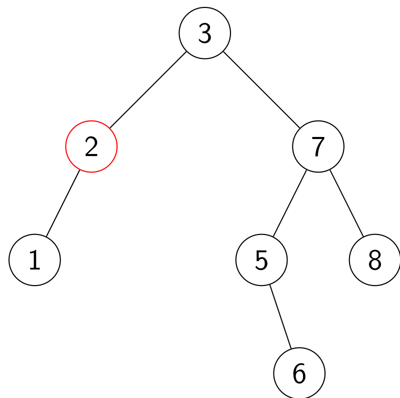

3
2
7
1
5
8
6

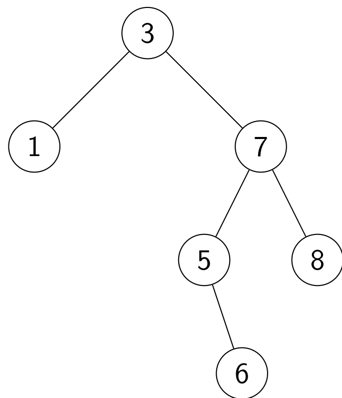

3
2
7
1
5
8
6

Operationen: `Insert(4)`

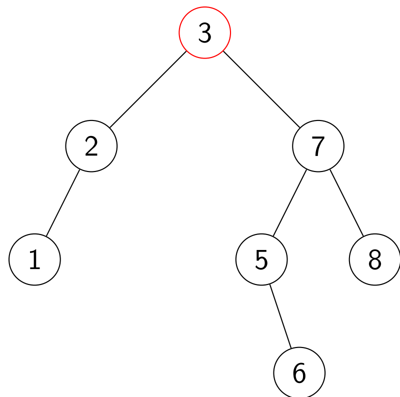


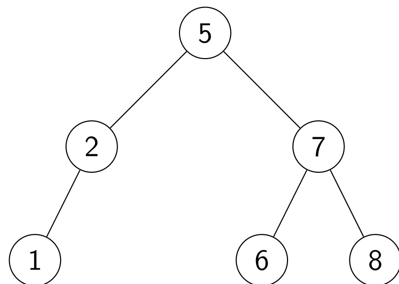
Operationen: `Delete(2)`





Operationen: `Delete(3)`

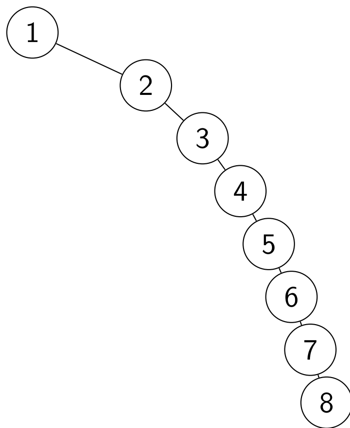




Selbst überlegen: Warum funktioniert das immer?

Laufzeit

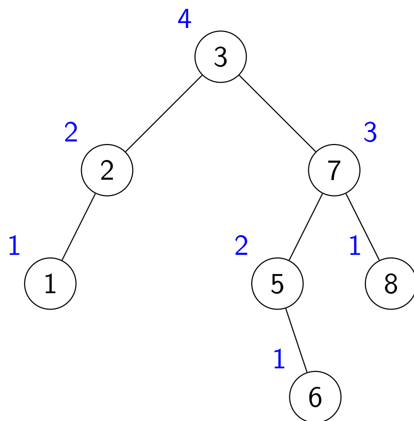
- Jeweils $\mathcal{O}(h)$, wobei h die Höhe des Baumes ist
- Ist das gut?



AVL-Bäume

Definition

Ein binärer Suchbaum heißt höhenbalanciert (oder AVL-Baum), wenn sich für jeden Knoten die Höhe der Kinder um höchstens 1 unterscheidet.



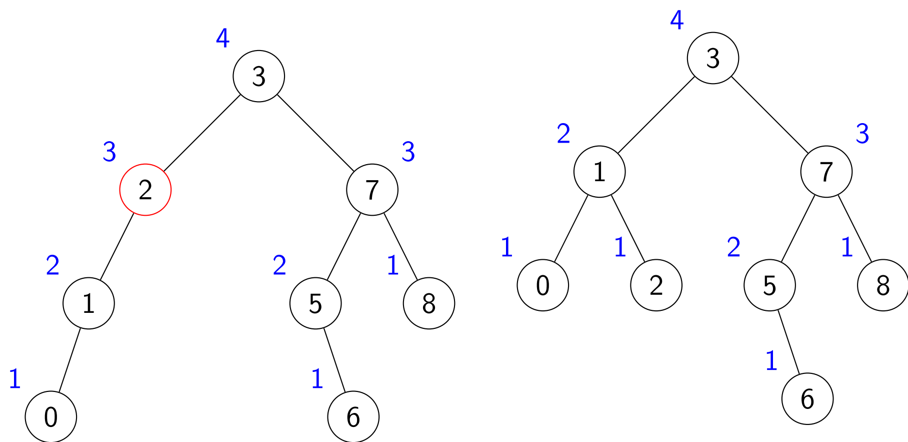
AVL-Bäume cont'd

Satz aus der Vorlesung

Die Höhe eines AVL-Baumes mit n Knoten ist in $\mathcal{O}(\log n)$.

- Wie erhalten wir die AVL-Eigenschaft beim Einfügen und Löschen?
- Lokal restrukturieren!

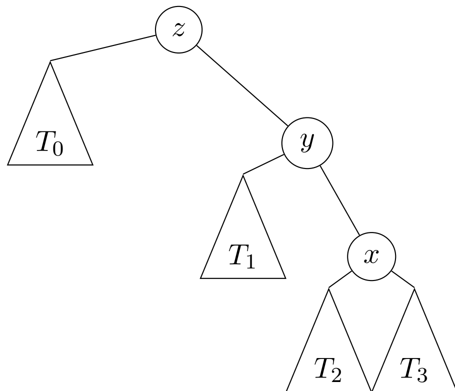
Insert(0)

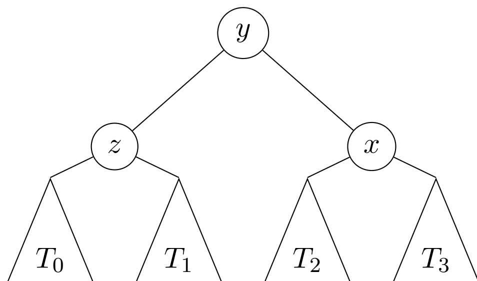


Restructure

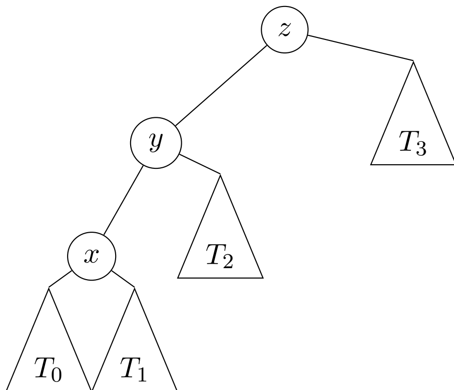
- Sei v der eingefügte/entfernte Knoten.
- z ist der erste unbalancierte Knoten auf dem Weg von v zur Wurzel.
- y ist das höhere Kind von z.
- x ist das höhere Kind von y (falls gleich hoch: beliebig).
- Sei $a \leq b \leq c$ die Größensortierung von x, y, z und T_0, T_1, T_2, T_3 die Größensortierung der Teilbäume an den übrigen Kindern von x, y, z.
- Dann: Ersetze z durch b mit linkem Kind a, rechtem Kind c und hänge die Teilbäume T_0 und T_1 als linkes/rechtes Kind an a und T_2 und T_3 als linkes/rechtes Kind an c.

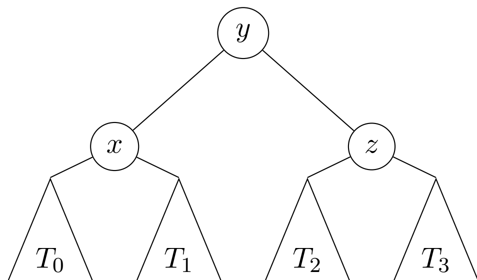
Fall 1



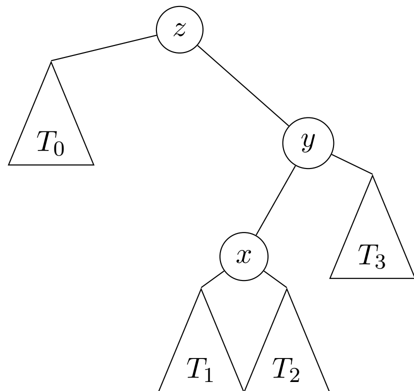


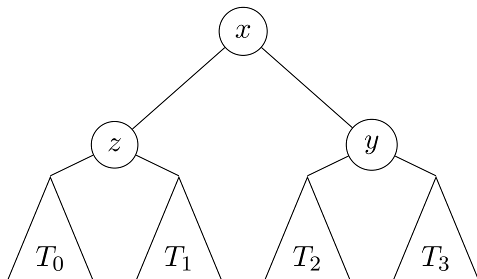
Fall 2



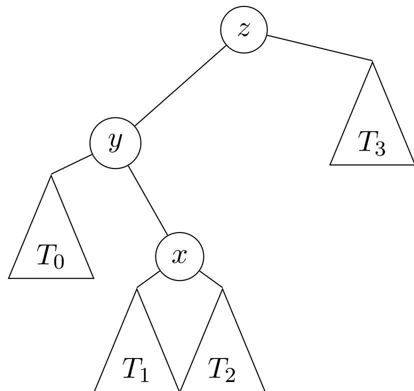


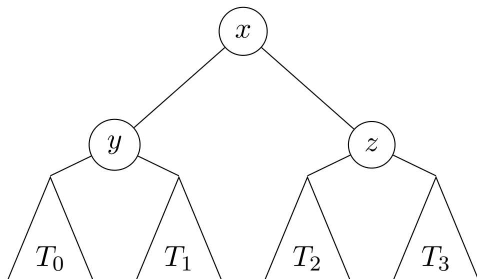
Fall 3



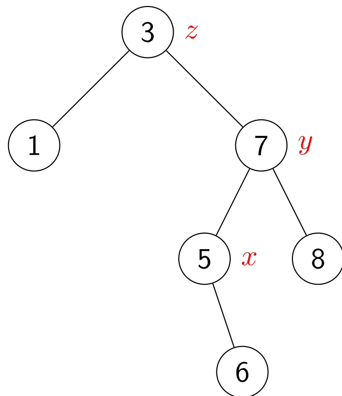


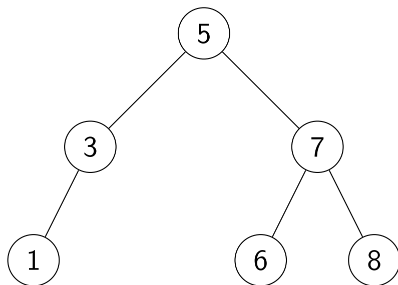
Fall 4





Delete(2)

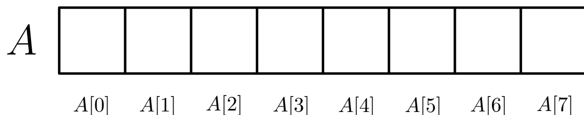


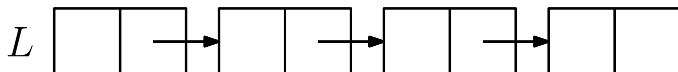


- Achtung: Nach dem Löschen können durch `Restructure` weitere Knoten unbalanciert werden.
- Weiter restrukturieren! (Insgesamt maximal $\mathcal{O}(\log n)$ mal)

Dynamische Arrays

Arrays vs. verkettete Listen

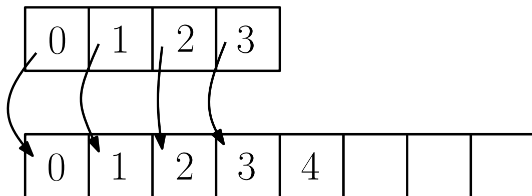




- Arrays: schneller wahlfreier Zugriff ($\mathcal{O}(1)$), feste Größe
- Listen: dynamische Größe, langsamer Zugriff ($\mathcal{O}(n)$)

Idee: dynamische Arrays

- Bei vollem Array einfach in ein größeres verschieben!



- Viele unterschiedliche Bezeichnungen: `Vector` (C++), `ArrayList` (Java), `list` (Python), ...

Laufzeit für eine Einfügeoperation

- Im schlimmsten Fall $\Omega(n)$
- Aber: oft nur $\mathcal{O}(1)$
- Spannende Frage: Wie oft?
- Wenn man die Größe des Arrays immer verdoppelt, müssen $n/2$ Objekte eingefügt werden bevor n Objekte verschoben werden.
- Im Mittel müssen für jedes eingefügte Objekt zwei verschoben werden.
- *Amortisierte* Zeit $\mathcal{O}(1)$

Vor- und Nachteile

Vorteile:

- wahlfreier Zugriff in $\mathcal{O}(1)$
- Einfügen in $\mathcal{O}(1)$ amortisierter Zeit
- Räumliche Lokalität

Nachteile:

- Worst Case: $\Omega(n)$ Zeit für eine Einfügeoperation
- ggf. viel verschwendeter Platz

Fragen?

Schöne Weihnachtsferien!