Datenstrukturen & Algorithmen

Peppo Brambilla
Universität Bern
Frühling 2018

Prüfung

- Donnerstag, 07.06.2018, 16:15-18:00 Uhr
- ExWi, A6
- Unterlagen: zwei Blätter (4 Seiten) handschriftliche Notizen
  - Keine weiteren Unterlagen oder Hilfsmittel
- Wichtig: Anmeldung auf KSL zwingend

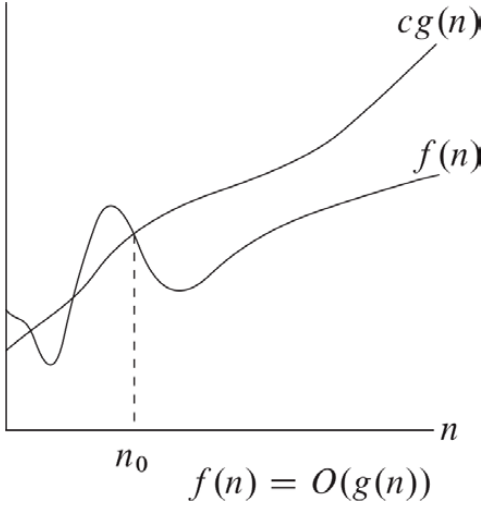
Asymptotische Notationen

$$\boldsymbol{f(n) = O(g(n))}$$

Es gibt positive Konstanten c, n_0, so dass

$$0 \leq f(n) \leq c \cdot g(n) \text{ für alle } n \geq n_0$$

→ g wächst schneller oder gleich wie f



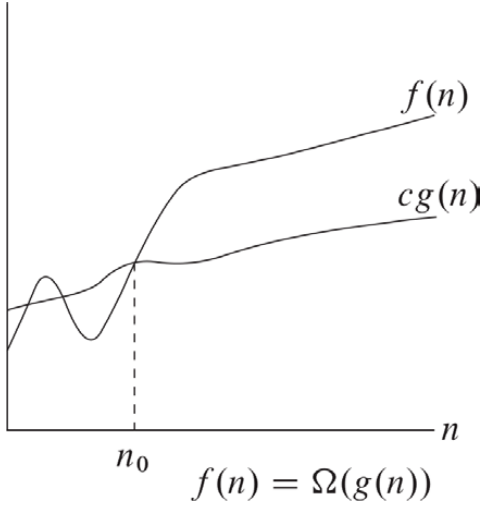
Asymptotische Notation

$$\boldsymbol{f(n) = \Omega(g(n))}$$

Es gibt positive Konstanten c, n_0, so dass

$$0 \leq c \cdot g(n) \leq f(n) \text{ für alle } n \geq n_0$$

→ f wächst schneller oder gleich wie g



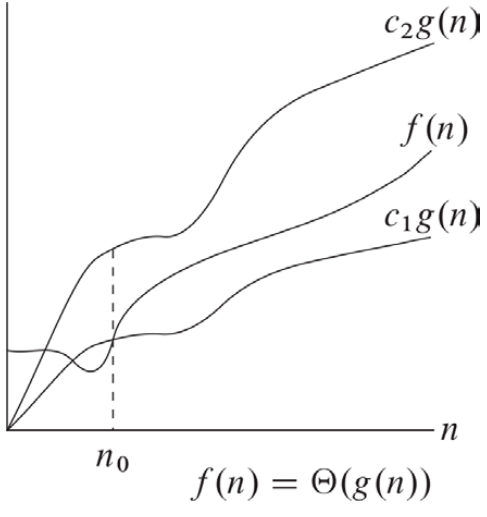
Asymptotische Notationen

$$f(n) = \Theta(g(n))$$

Es gibt positive Konstanten c_1, c_2, n_0, so dass

$0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ für alle $n \geq n_0$

→ g wächst gleich wie f



Beispiele

- $2n^2 = O(n^3)$
- $2n^2 \neq \Theta(n^3)$
- $2n^2 = O(n^2)$
- $2n^2 = \Theta(n^2)$
- $\frac{1}{2}n^2 - 2n = \Theta(n^2)$
- $\log_b n = O(\sqrt[m]{n})$ ($m \in \mathbb{N}$ fest)
- $\log_b n \neq \Theta(\sqrt[m]{n})$ ($m \in \mathbb{N}$ fest)
- $\log_{b_1} n = \Theta(\log_{b_2} n)$

Komplexitätsklassen

- Konstant: $T(n) = \Theta(1)$
- Logarithmisch: $T(n) = \Theta(\lg n)$
- Linear: $T(n) = \Theta(n)$
- Polynomial: $T(n) = \Theta(n^m)$ ($m \in \mathbb{N}$, fest)
- Exponentiell: $T(n) = \Theta(2^n)$
- Faktoriell: $T(n) = \Theta(n!)$
- Weitere siehe http://en.wikipedia.org/wiki/Time_complexity

Beispielaufgabe

- Ordnen nach asymptotischer Wachstumsrate, f links von g heisst $f = O(g)$.

	Vereinfachen:	Reihenfolge:
(a) 2^n	(a) $T(2^n) = \Theta(2^n)$	1. $T(\frac{\log n}{\sqrt{n}}) = T(2^{1024}) = \Theta(1)$
(b) n^3	(b) $T(n^3) = \Theta(n^3)$	2. $T(7n) = T(5^{\log_5 n}) = \Theta(n)$
(c) $5^{\log_5 n}$	(c) $T(5^{\log_5 n}) = \Theta(n)$	3. $T(\sqrt{n^3}) = \Theta(n^{\frac{3}{2}})$
(d) $\prod_{i=1}^{n} i^2$	(d) $T(\prod_{i=1}^{n} i^2) = \Theta((n!)^2)$	4. $T(4n^2 + 100n) = \Theta(n^2)$
(e) $7n$	(e) $T(7n) = \Theta(n)$	5. $T(n^3) = \Theta(n^3)$
(f) $4n^2 + 100n$	(f) $T(4n^2 + 100n) = \Theta(n^2)$	6. $T(2^n) = \Theta(2^n)$
(g) $\frac{\log n}{\sqrt{n}}$ → wächst weniger schnell	(g) $T(\frac{\log n}{\sqrt{n}}) = \Theta(1)$	7. $T(\prod_{i=1}^{n} i^2) = \Theta((n!)^2)$
(h) $\sqrt{n^3}$	(h) $T(\sqrt{n^3}) = \Theta(n^{\frac{3}{2}})$	
(i) 2^{1024}	(i) $T(2^{1024}) = \Theta(1)$	

Rekursionsgleichungen

- Rekursionsgleichung: Gleichung, die Funktion durch eigene Funktionswerte für kleinere Eingaben beschreibt
- Lösen von Rekursionsgleichen zur Analyse der Zeitkomplexität
- 3 Tricks
  - Substitutionsmethode
  - Rekursionsbaum-Methode
  - Mastermethode
- Beweis durch Induktion

Beispielaufgabe

- Löse $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n}{4} + 2,\ T(1) = 4$

a) Mastermethode

b) Rekursionsbaum

c) Zeige durch Induktion, dass $T(n) = O(n^2)$

Mastermethode

- Problem: löse $T(n) = aT(n/b) + f(n)$
- Vergleiche $f(n)$ mit $n^{\log_b a}$

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$

$$T(n) = \Theta(n^{\log_b a})$$

2. Falls $f(n) = \Theta\left(n^{\log_b a}\right)$

$$T(n) = \Theta(n^{\log_b a} \lg n)$$

3. Falls $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für eine Konstante $\varepsilon > 0$ und $f(n)$ erfüllt $af(n/b) \leq cf(n)$ für Konstante $c < 1$

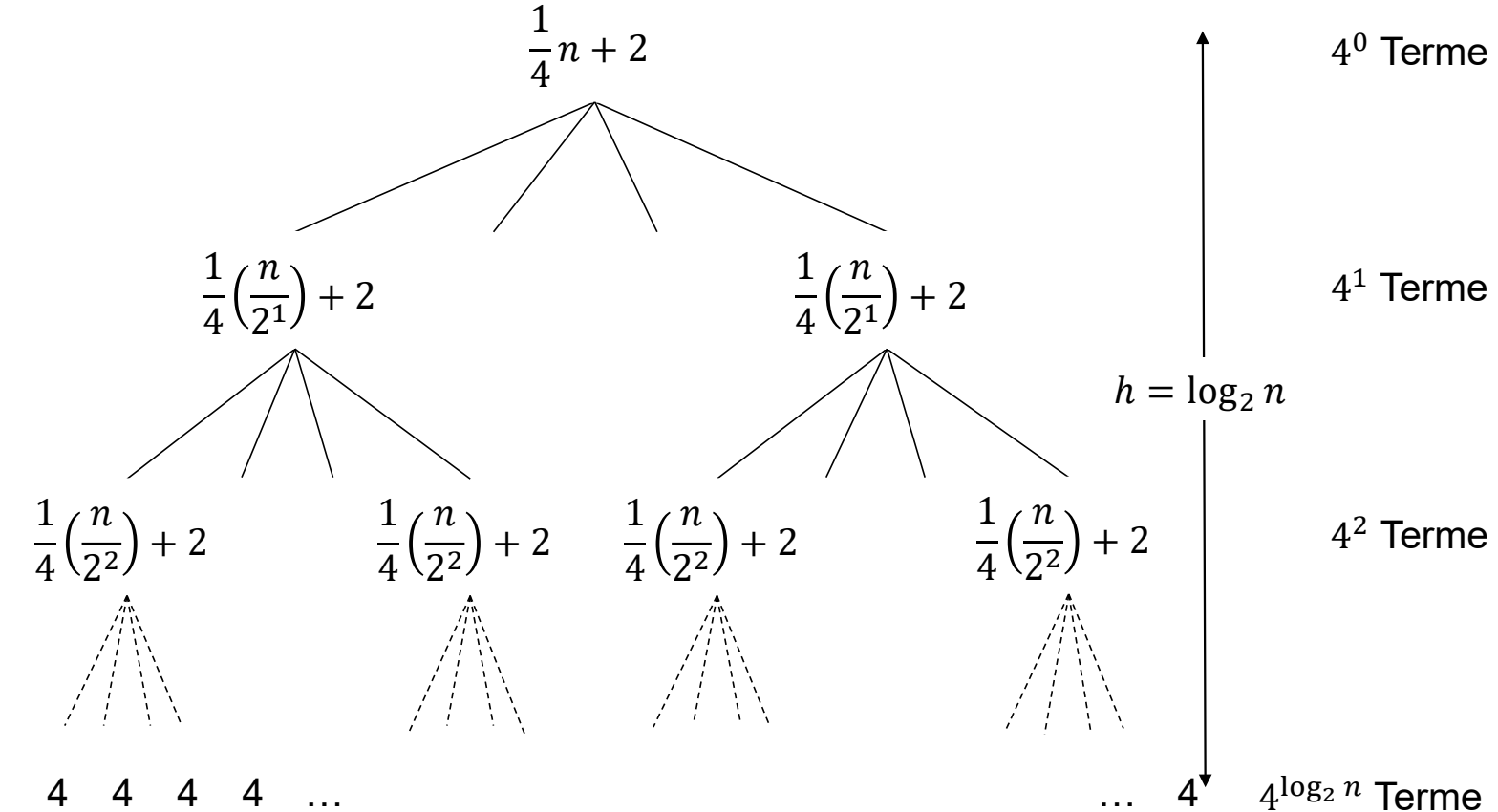
$$T(n) = \Theta(f(n))$$

Beispiel Mastermethode

- $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n}{4} + 2$ (Musterprüfung)
  - $a = 4\ (\geq 1)$
  - $b = 2\ (> 1)$
  - $f(n) = \frac{n}{4} + 2$ (asymptotisch positiv)
  - $n^{\log_b a} = n^{\log_2 4} = n^2$
  - Also Fall 1: $f(n) = \frac{n}{4} + 2 = O(n^{2-\varepsilon})$ für $1 \geq \varepsilon > 0$
  - Lösung: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

Rekursionsbaum

$$T(n) = 4T\left(\frac{n}{2}\right) + \frac{n}{4} + 2,\ T(1) = 4$$



Kosten aufrechnen

- Kosten für alle Knoten auf Tiefe $0 \leq k < \log_2 n$:

$$4^k \left(\frac{1}{4} \left(\frac{n}{2^k} \right) + 2 \right)$$

- Höhe des Baumes $h = \log_2 n$, Kosten für alle Blätter:

$$4^{\log_2 n} \cdot 4 = 4 \cdot 2^{2 \cdot \log_2 n} = 4n^2$$

- Total :

$$\sum_{i=0}^{\log_2 n - 1} 4^i \left(\frac{n}{4 \cdot 2^i} + 2 \right) + 4n^2$$

- Ausgerechnet (mit Summenformel):

$$\frac{59n^2 - 3n - 8}{12}$$

Induktionsbeweis $T(n) = O(n^2)$

- Induktionsannahme: für $k < n$ gelte

$$T(k) \leq c_1 k^2 + c_2 k$$

- Induktionsschritt: $2^{k-1} \to n = 2^k$

$$\begin{aligned} T(n) &= 4T(n/2) + n/4 + 2 \\ &\leq 4(c_1 (n/2)^2 + c_2 n/2) + n/4 + 2 \\ &= c_1 n^2 + 2c_2 n + n/4 + 2 \\ &= c_1 n^2 + c_2 n - (c_2 n - n/4 - 2) \\ &\leq c_1 n^2 + c_2 n, \text{ falls } c_2 n - n/4 - 2 \geq 0 \end{aligned}$$

Für $c_2 \geq 9/4$ gilt obige Ungleichung für $n \geq 1$

- Nebenbedingung: Wähle c_1 gross genug, dass $T(1) = 4 \leq c_1 \cdot 1^2 + c_2 \cdot 1$, z.B. $c_1 = 10$

$c_2 = 3$

Mit Pseudocode

```
1  Foo(n)
2  if n = 1
3        return 1
4  else
5        return Foo(n/4) + Foo(n/4)
```

Aufwand hier = 1

- Rekursion → Rekursionsgleichung

$$T(n) = 2\,T\left(\frac{n}{4}\right) + 1$$

→ Mastertheorem benutzen

Laufzeiten von Schleifen

```
1   i = 1
2   while i < n/4
3         j = 2i
4         while j < n
5               j = j + 1
6         i = i + 1
```

- Innere Schleife:
  $2i$ … $n-1 \rightarrow S_i = \sum_{j=2i}^{n-1} 1 \rightarrow S_i = n - 2i$
- Äussere Schleife: $1 \ldots \left\lfloor \frac{n}{4} \right\rfloor - 1$: $\sum_{i=1}^{\left\lfloor \frac{n}{4} \right\rfloor - 1} S_i$

Laufzeiten von Schleifen

Äussere Schleife: (Annahme $n = 4m$)

$$\sum_{i=1}^{\frac{n}{4}-1} n - 2i = n\left(\frac{n}{4} - 1\right) - 2\sum_{i=1}^{\frac{n}{4}-1} i$$

$$= \frac{n^2}{4} - n - 2\frac{\left(\frac{n}{4} - 1\right)\frac{n}{4}}{2}$$

$$= \frac{3n^2}{16} - \frac{3n}{4} = \Theta(n^2)$$

Quicksort

- Sortiere ein Feld A mit n Elementen

1. Teile: Zerlege Feld in zwei Teilfelder um ein Pivot q, so dass {Elemente links von q} $\leq A[q] \leq$ {Element rechts von q}

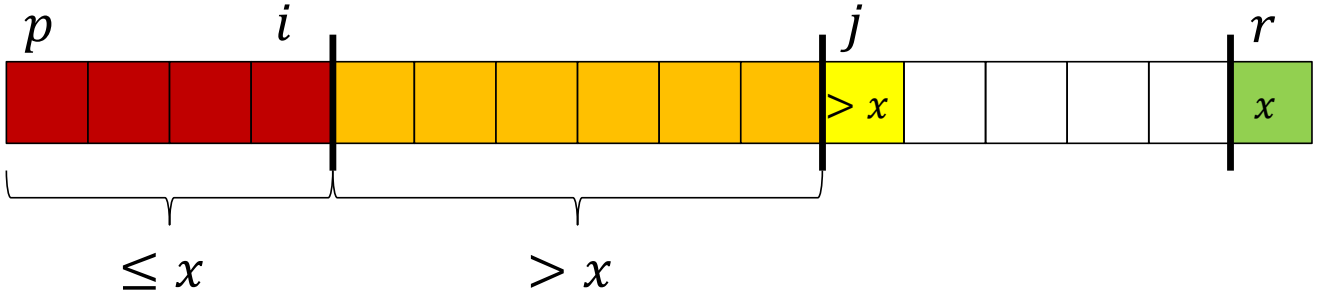
q

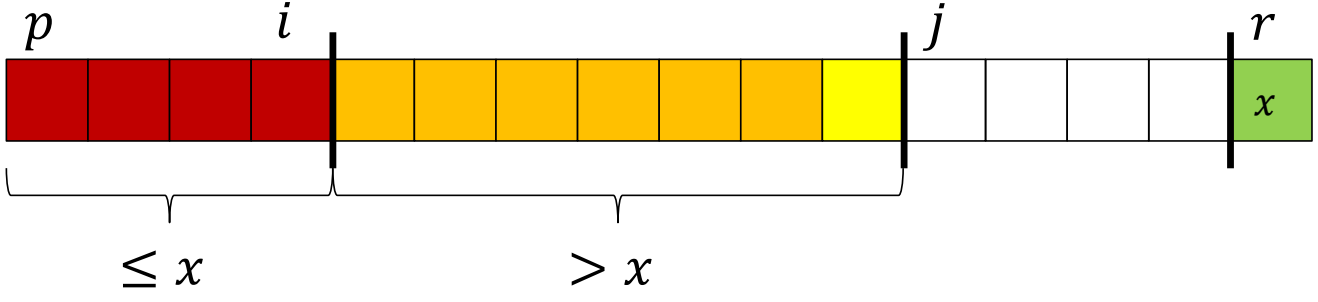
$x = A[q]$

2. Beherrsche: Sortiere Teilfelder rekursiv
3. Verbinde: Teilfelder zusammenfügen, trivial

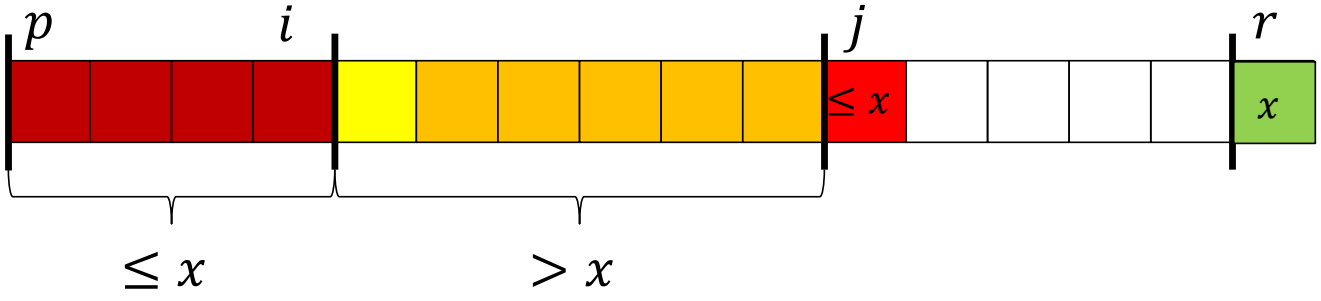
- Schlüssel zum Erfolg: Zerlegung des Felds in linearer Zeit

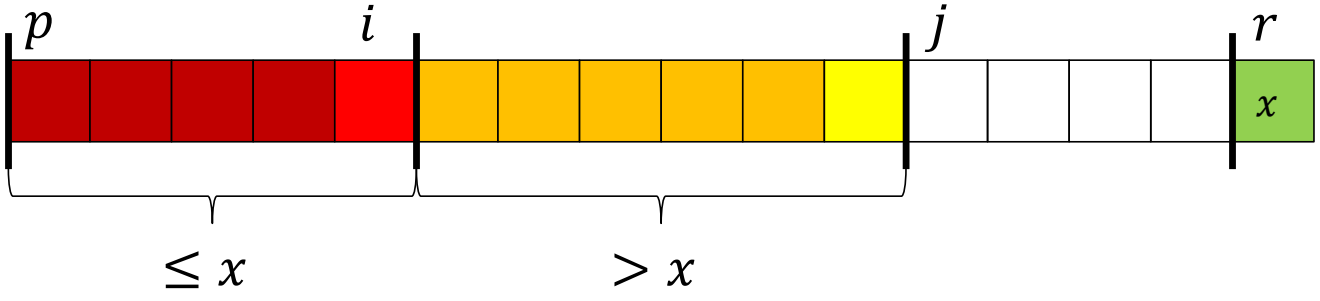
Partitionierung



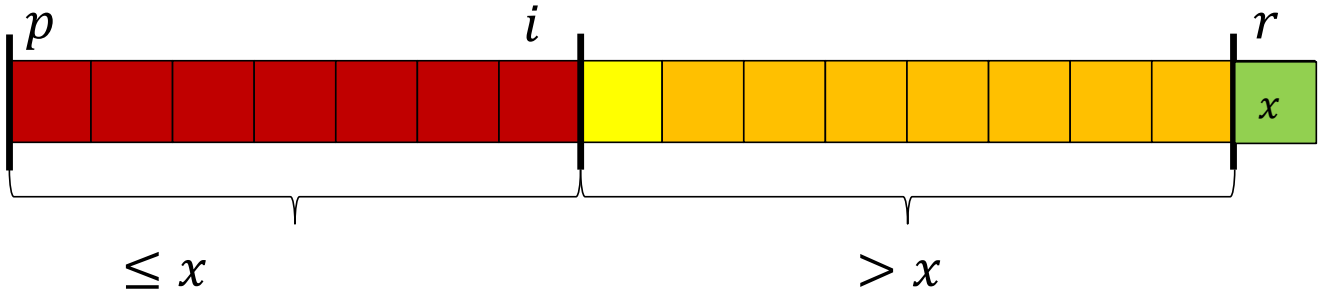


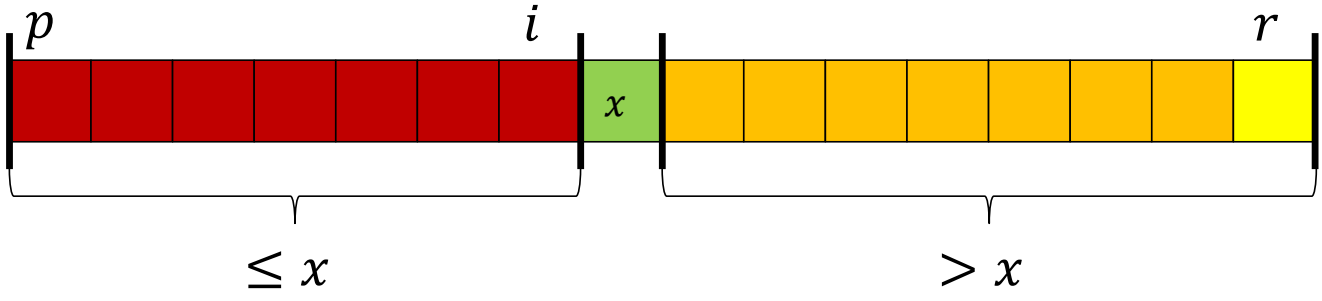
Partitionierung





Partitionierung



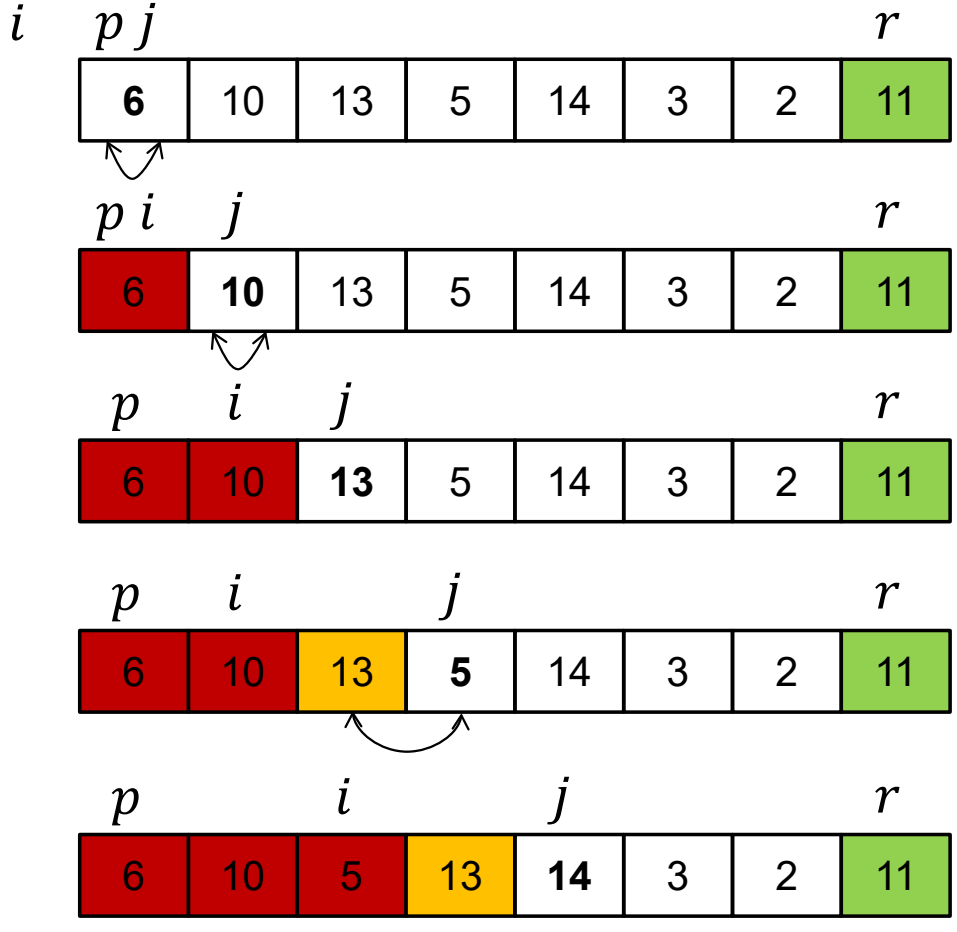


Zerlegung des Felds

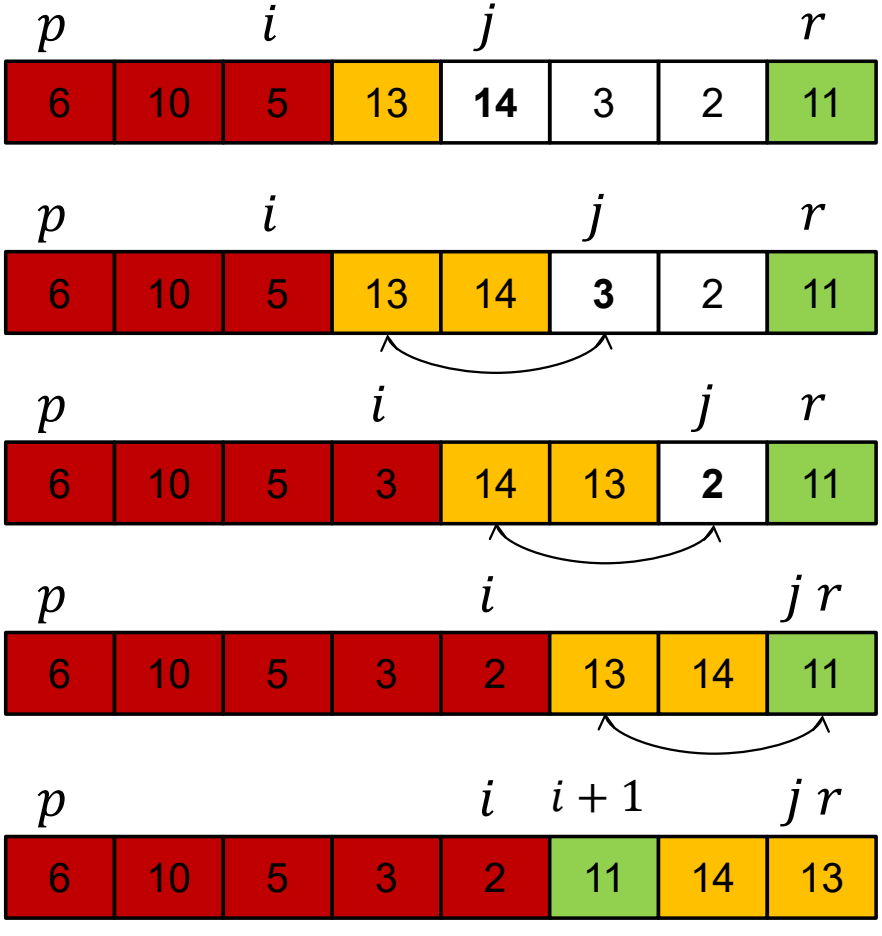
```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4        if A[j] ≤ x
5              i = i + 1
6              exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

```
QUICKSORT(A, p, r)
1  if p < r
2        q = PARTITION(A, p, r)
3        QUICKSORT(A, p, q − 1)
4        QUICKSORT(A, q + 1, r)
```

Beispiel


i p j r
6 10 13 5 14 3 2 11
p i j r
6 10 13 5 14 3 2 11
p i j r
6 10 13 5 14 3 2 11
p i j r
6 10 13 5 14 3 2 11
p i j r
6 10 5 13 14 3 2 11

Beispiel


p i j r
6 10 5 13 14 3 2 11
p i j r
6 10 5 13 14 3 2 11
p i j r
6 10 5 3 14 13 2 11
letzte Ausführung der Schleife
p i j r
6 10 5 3 2 13 14 11
exchange A[i+1] with A[r]
p i i + 1 j r
6 10 5 3 2 11 14 13
nach Verschieben des Pivots A[r]

Quicksort

- Quicksort
  - In-place Verfahren
  - Teile-und-beherrsche Schema
  - Zerlegen des Feldes um Pivot Element
  - Vertauschungen: keine Blöcke werden kopiert (Folge: Sortierverfahren ist instabil)
  - Worst case: $\Theta(n^2)$, average case: $\Theta(n \lg n)$
- Randomisierter Quicksort
  - Pivot-Element zufällig auswählen und mit letztem Element vertauschen

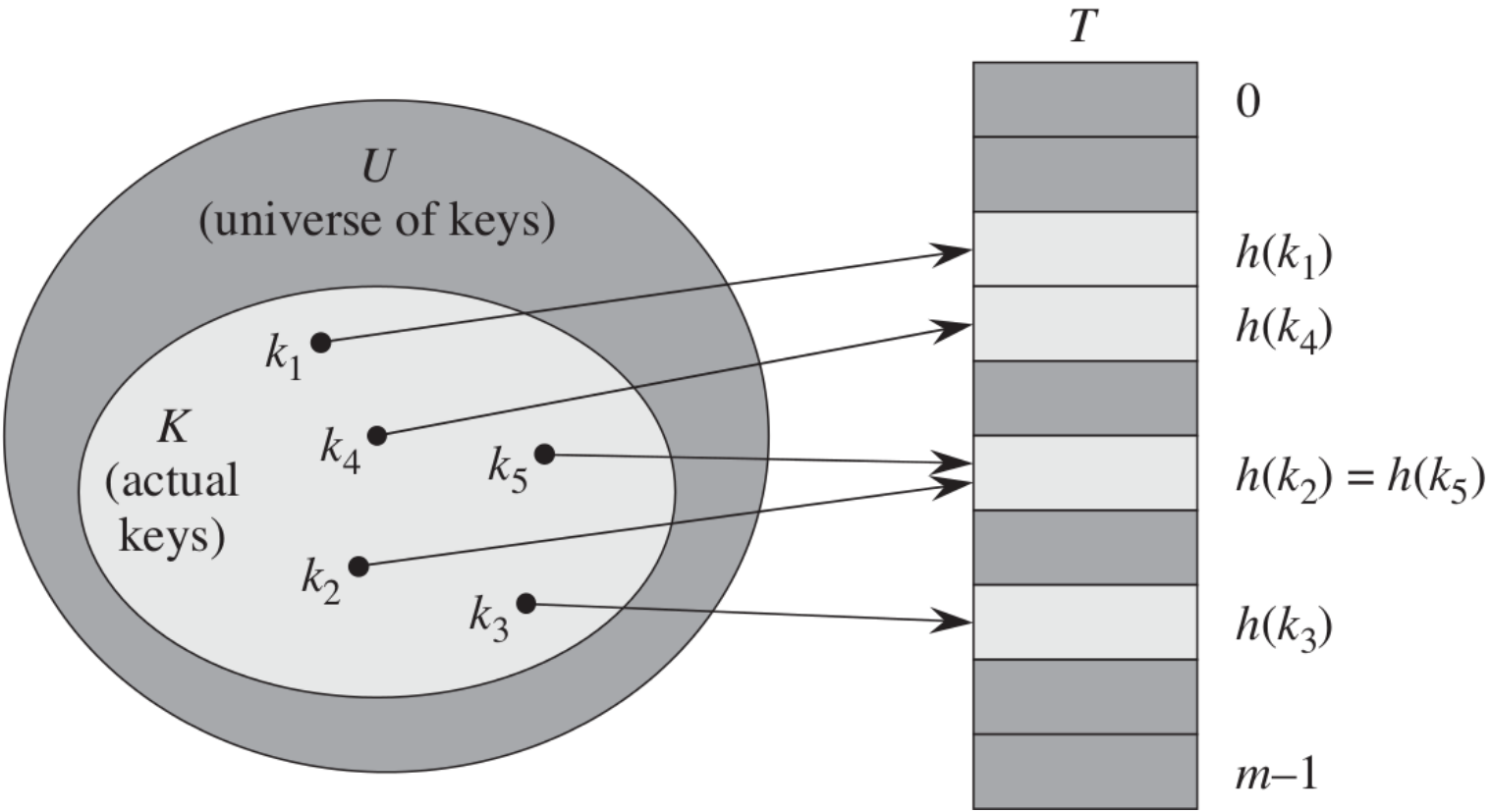
Beispielaufgabe

- Die Methode *Quicksort*(A, p, r) sortiere die Elemente von Index p bis r des Feldes A mittels des Quicksort Algorithmus. Illustrieren Sie den Ablauf von *Quicksort(A,1,8)* anhand des Feldes $A = [3, 7, 9, 4, 8, 10, 11, 6]$, indem Sie den Ablauf unten vervollständigen.
- Geben Sie jeden rekursiven Aufruf von *Quicksort* mit seinen Parametern an, und schreiben Sie jeweils das Feld A auf, nachdem das Feld mit Hilfe des Pivotelements zerlegt wurde. Nehmen Sie an, in jedem Schritt werde das Element an der Stelle r als Pivot-Element verwendet.
- Quicksort(A,1,8)
  Feld A nach Zerlegung mit Pivot A[8]: ...
- Quicksort(A,...,...)
  Feld A nach Zerlegung mit Pivot A[...]: ...
- ...

Lösung

- A: [3, 7, 9, 4, 8, 10, 11, 6]
- Quicksort(A, 1, 8), Pivot A[8]
  → [3, 4, **6**, 7, 8, 10, 11, 9]
  - Quicksort(A, 1, 2), Pivot A[2]
    → [3, **4**, **6**, 7, 8, 10, 11, 9]
    - Quicksort(A, 1, 1), Pivot A[1]
      → [**3**, **4**, **6**, 7, 8, 10, 11, 9]
  - Quicksort(A, 4, 8), Pivot A[8]
    → [**3**, **4**, **6**, 7, 8, **9**, 11, 10]
    - Quicksort(A, 4, 5), Pivot A[5]
      → [**3**, **4**, **6**, 7, **8**, **9**, 11, 10]
      - Quicksort(A, 4, 4), Pivot A[4]
        → [**3**, **4**, **6**, **7**, **8**, **9**, 11, 10]
    - Quicksort(A, 7, 8), Pivot A[8]
      → [**3**, **4**, **6**, **7**, **8**, **9**, **10**, 11]
      - Quicksort(A, 8, 8), Pivot A[8]
        → [**3**, **4**, **6**, **7**, **8**, **9**, **10**, **11**]

Hashtabellen



Kollisionen

- Zwei oder mehr Schlüssel ergeben denselben Hashwert
  - Nicht auszuschliessen da Schlüsseluniversum viel grösser als Hashtabelle, d.h. $|U| > m$
- Zwei Methoden zur Behandlung von Kollisionen
  - Verkettung
  - Offene Adressierung

Offene Adressierung

- Idee: speichere alle Schlüssel in Hashtabelle selbst
  - Jeder Platz enthält einen Schlüssel oder nil
- Hashtabelle kann voll werden
  - Belegungsfaktor immer $\alpha \leq 1$
- Kollisionsbehandlung: berechne Sequenz von Plätzen, die sondiert werden

Hashfunktion

$$h : U \times \underbrace{\{0,1,\ldots,m-1\}}_{\text{Sondierung}} \rightarrow \underbrace{\{0,1,\ldots,m-1\}}_{\text{Platz}}$$

- Sequenz der sondierten Plätze muss Permutation von $(0,1,\ldots,m-1)$ sein
  - Anders formuliert: Sondierungssequenz
    $$\langle h(k,0), h(k,1), \ldots, h(k,m-1)\rangle$$
    muss Permutation von $(0,1,\ldots,m-1)$ sein
  - Jeder Platz muss genau einmal sondiert werden

Lineares Sondieren

- Hilfshashfunktion h'
  - z.B. Multiplikationsmethode
- Lineares Sondieren hat Hashfunktion

$$h(k,i) = (h'(k) + i\ ) \bmod m$$

  - Sondierung i
- Erzeugt Sondierungssequenz

$$T[h'(k)], T[h'(k)+1], \dots, T[m-1], T[0], \dots, T[h'(k)-1]$$

- Nur m verschiedene Sequenzen!

Lineares Sondieren

Nachteil: Primäres Clustering

- Lange Folgen besetzter Plätze
  - Leerer Platz folgend auf i besetzte Plätze wird mit Wahrscheinlichkeit $(i + 1)/m$ als nächstes belegt
  - Lange Folgen besetzter Plätze werden noch länger
  - Suchen wird aufwändiger

Quadratisches Sondieren

- Hashfunktion

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

  - Hilfskonstanten $c_1, c_2 \neq 0$
- Besser als lineares Sondieren
  - Kein primäres Clustering
- Nur m Sondierungssequenzen
  - Alle Schlüssel k mit gleicher ersten Sondierung $h(k,0)$ führen zu gleicher Sequenz
  - Sekundäres Clustering

Doppeltes Hashing

- Zwei Hilfshashfunktionen h_1, h_2
- Hashfunktion

$$h(k,i) = \left(h_1(k) + i\, h_2(k)\right) \bmod m$$

- Bedingung: $h_2(k)$ ist teilerfremd zu m damit Sondierungssequenz eine Permutation von $(0,1,\ldots,m-1)$ erzeugt
  - Keine gemeinsamen Faktoren ausser 1
- Mögliche Lösungen
  - m ist Zweierpotenz und h_2 ist immer ungerade
  - m ist prim und $1 < h_2(k) < m$

Beispielaufgabe

a) Betrachten Sie die folgende Hashtabelle, die bereits fünf Schlüssel enthält. Fügen Sie zusätzlich die Schlüssel 19, 14, 30, 10, 12, 99 in dieser Reihenfolge in die Hashtabelle ein. Verwenden Sie doppeltes Hashing mit den Hilfshashfunktionen $h_1(k) = (k \bmod 11)$ und $h_2(k) = 1 + (k \bmod 9)$. **3 Punkte**

11			3		16				20	0
0	1	2	3	4	5	6	7	8	9	10

b) In Ihrer Anwendung kommen die Operationen Schlüssel einfügen, suchen und entferen etwa gleich häufig vor. Bevorzugen Sie in diesem Fall eine Hashtabelle mit Kollisionsbehandlung durch Verkettung, oder offene Adressierung? Begründen Sie Ihre Antwort mit zwei oder drei Sätzen. **1 Punkt**

Lösung:

a)

11	30	12	3	14	16	99	10	19	20	0
0	1	2	3	4	5	6	7	8	9	10

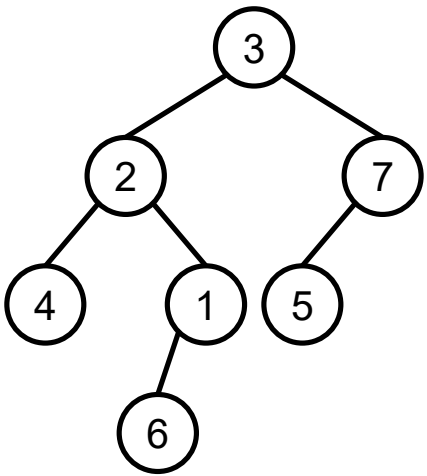
b) Bei Verkettung wird sowohl erfolglose wie erfolgreiche Suche schneller, wenn der Belegungsgrad sinkt weil Elemente wieder gelöscht werden. Bei offener Addressierung funktioniert das Löschen so, dass man einen Platzhalter für "gelöscht" einfügt, um zu markieren, dass die entsprechende Position wieder frei ist. Suchen wird aber dadurch nicht wieder schneller. Aus diesem Grund wird Kollisionsbehandlung durch Verkettung bevorzugt.

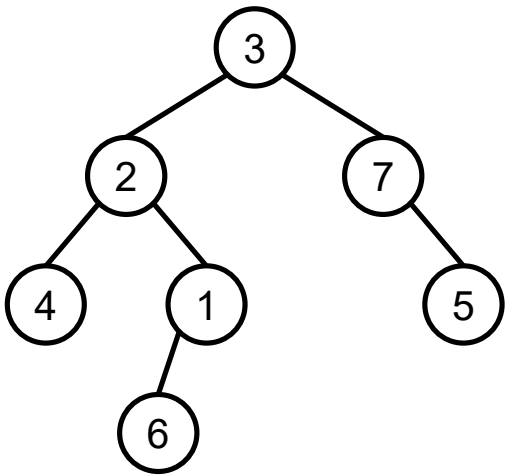
Binärer Baum

- Datenstruktur für dynamische Mengen
  - Unterstützen viele Operationen effizient
- Wörterbuchoperationen
  - Suchen, einfügen, löschen
  - Aufwand proportional zur Höhe des Baumes
  - Höhe zwischen $\Theta(n)$ und $\Theta(\lg n)$
- Zusätzlich
  - Prioritätswarteschlangen (Minimum, Maximum effizient abfragen)
  - Sortierte Auflistung in $\Theta(n)$
  - Vorgänger, Nachfolger effizient abfragen

Beispiel

- Zwei binäre Bäume hier nicht identisch
- Linker und rechter Teilbaum werden unterschieden





Binäre Suchbaum-Eigenschaft

- Falls Knoten y im linken Teilbaum von Knoten x, dann $\boldsymbol{key}[\boldsymbol{y}] \leq \boldsymbol{key}[\boldsymbol{x}]$
- Falls Knoten y im rechten Teilbaum von Knoten x, dann $\boldsymbol{key}[\boldsymbol{y}] \geq \boldsymbol{key}[\boldsymbol{x}]$

Traversierungen

- Inorder-Traversierung: traversiere linken Teilbaum, gib Knoten aus, traversiere rechten Teilbaum
- Preorder-Traversierung: gibt Knoten aus, traversiere linken Teilbaum, traversiere rechten Teilbaum
- Postorder-Traversierung: traversiere linken Teilbaum, traversiere rechten Teilbaum, gib Knoten aus
- Nur Inorder-Traversierung gibt für einen binären Suchbaum die Schlüssel sortiert aus

Traversierungen

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2        INORDER-TREE-WALK(x.left)
3        print x.key
4        INORDER-TREE-WALK(x.right)
```

```
POSTORDER-TREE-WALK(x)
1  if x ≠ NIL
2        POSTORDER-TREE-WALK(x.left)
3        POSTORDER-TREE-WALK(x.right)
4        print x.key
```

Beispielaufgabe

- Zeichnen Sie den binären Suchbaum, dessen Postorder-Traversierung die Folge $1, 4, 5, 3, 2, 6, 9, 11, 10, 8, 7, 13, 12$ ergibt.
  Hinweis: Überlegen Sie sich zuerst, welches die Wurzel des Baumes ist.

Lösung

- Postorder:
  linker Teilbaum, rechter Teilbaum, Wurzel
  → Wurzel zuletzt ausgegeben
- Knoten der Teilbäume können durch Vergleich mit Wurzel bestimmt werden
  - $<$ Wurzel → liegt im linken Teilbaum
  - $>$ Wurzel → liegt im rechten Teilbaum
- Rekursiv auf Output der Teilbäume fortfahren

Suchen

- Überprüfte Knoten bilden Pfad von Wurzel nach unten
- Laufzeit in $O(h)$, h Höhe des Baumes
- Iterative Variante, ähnlich wie Traversierung einer Liste

```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2        if k < x.key
3              x = x.left
4        else x = x.right
5  return x
```

Einfügen und Löschen

- Einfügen und Löschen ändern die Struktur des Suchbaumes
- Müssen sicherstellen, das Suchbaum Eigenschaft wieder hergestellt wird
- Einfügen einfacher als Löschen

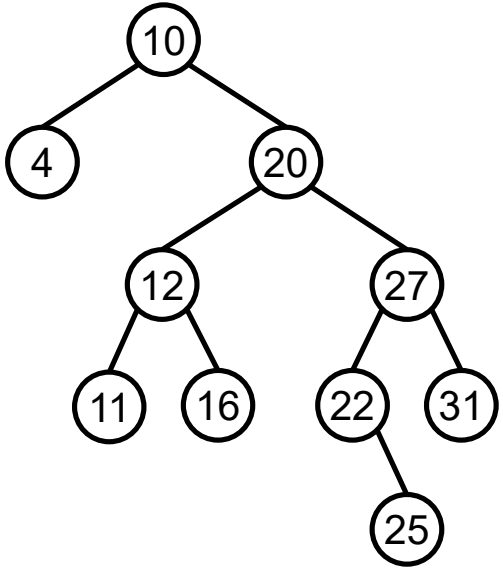
Löschen

```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2       TRANSPLANT(T, z, z.right)
 3  elseif z.right == NIL
 4       TRANSPLANT(T, z, z.left)
 5  else y = TREE-MINIMUM(z.right)
 6       if y.p ≠ z
 7            TRANSPLANT(T, y, y.right)
 8            y.right = z.right
 9            y.right.p = y
10       TRANSPLANT(T, z, y)
11       y.left = z.left
12       y.left.p = y
```

```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2       T.root = v
3  elseif u == u.p.left
4       u.p.left = v
5  else
6       u.p.right = v
7  if v ≠ NIL
8       v.p = u.p
```

Beispielaufgabe

- Lösche Knoten mit Schlüssel 20 aus dem Baum unten
- Markiere die Knoten z und y zugewiesen wurden
- Gibt die Zeilen der Funktion Tree-Delete an, die ausgeführt wurden
- Zeichne den Baum nach dem Löschen

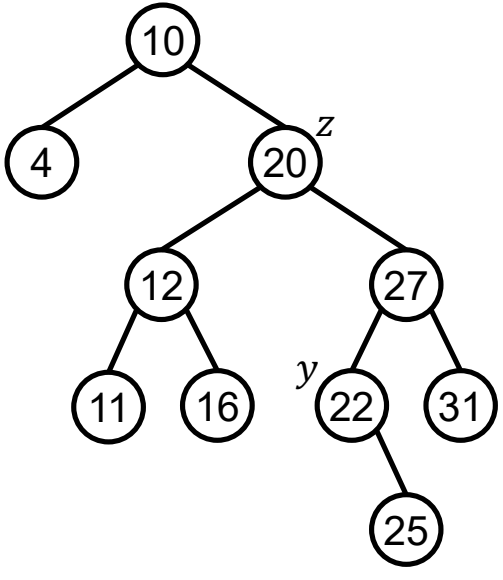


Beispielaufgabe

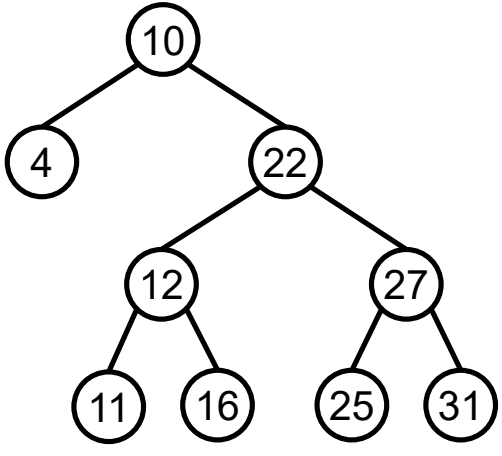
Lösung

- Zeilen: 1,3,5,6,7,8,9,10,11,12

Vor dem Löschen



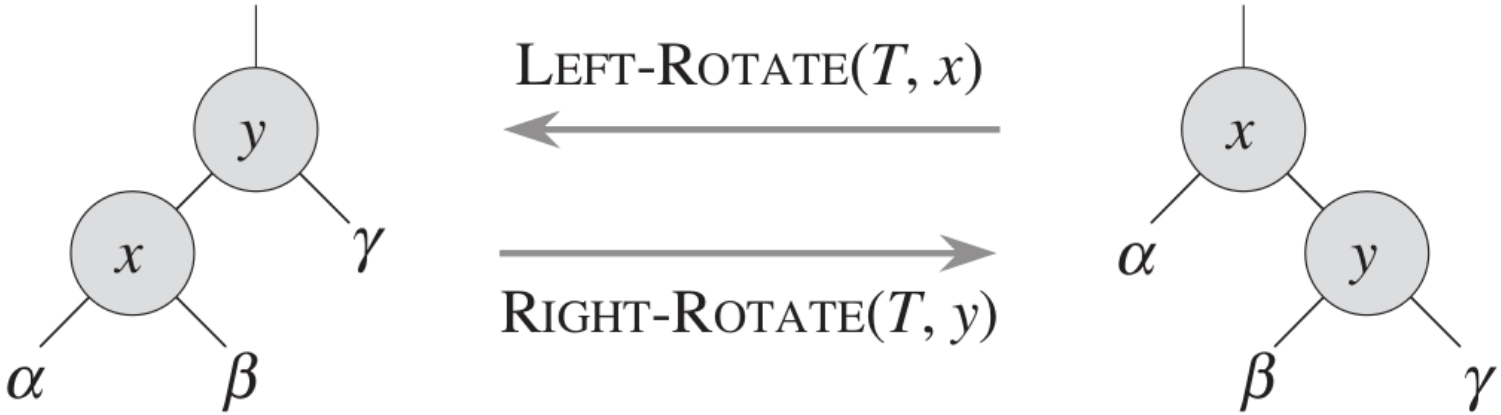
Nach dem Löschen



Rotationen

- Benötigen Rotationen, um Baum zu balancieren
  - Ändern Baumstruktur
  - Bewahren binäre Suchbaum-Eigenschaft
- Es gibt Links- und Rechtsrotationen
  - Invers zueinander
  - Implementiert durch Umhängen von Zeigern

Rotationen



Minimale Spannbäume

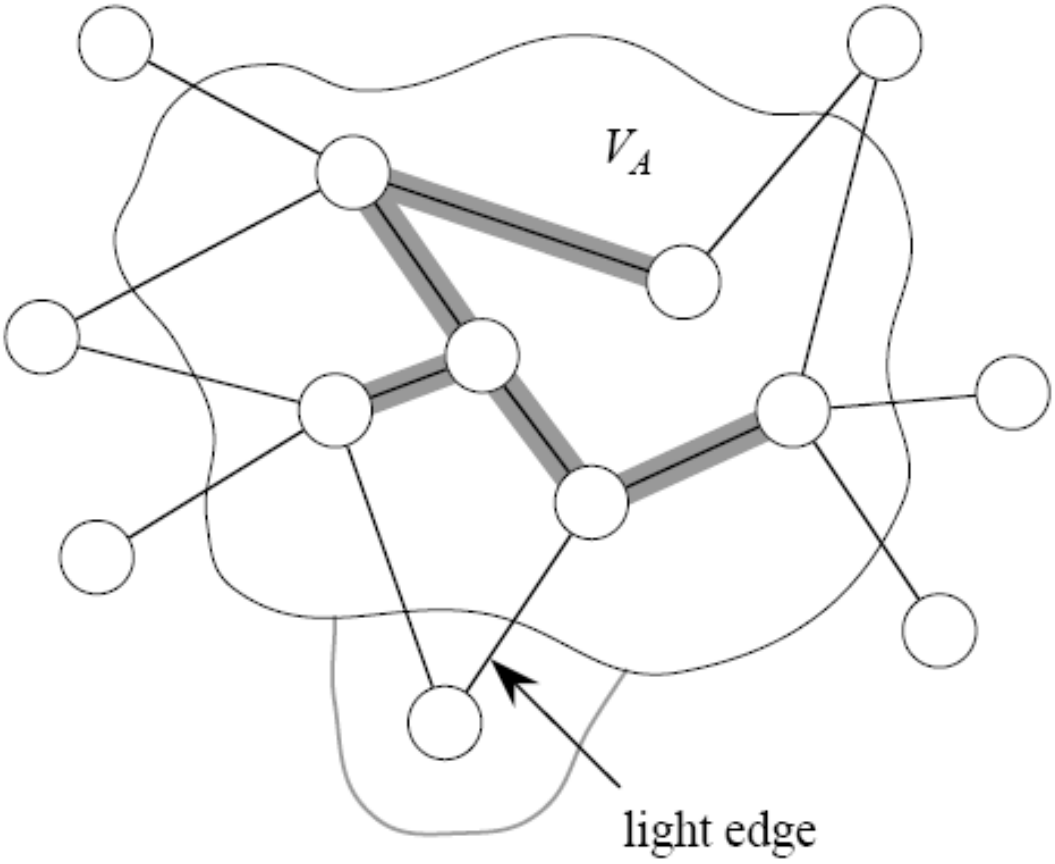
Modellierung als Graph

- Zusammenhängender, ungerichteter Graph $G = (V, E)$
- Kantengewichte $w(u, v)$ für Kanten $(u, v) \in E$
- Finde Teilmenge T der Kanten so dass
  - T alle Knoten verbindet (T ist ein Spannbaum)
  - Kosten $w(T) = \sum_{(u,v)\in T} w(u, v)$ minimiert werden
- Spannbaum mit minimalen Kosten über alle Spannbäume ist minimaler Spannbaum (minimal spanning tree, MST)

Algorithmus von Prim

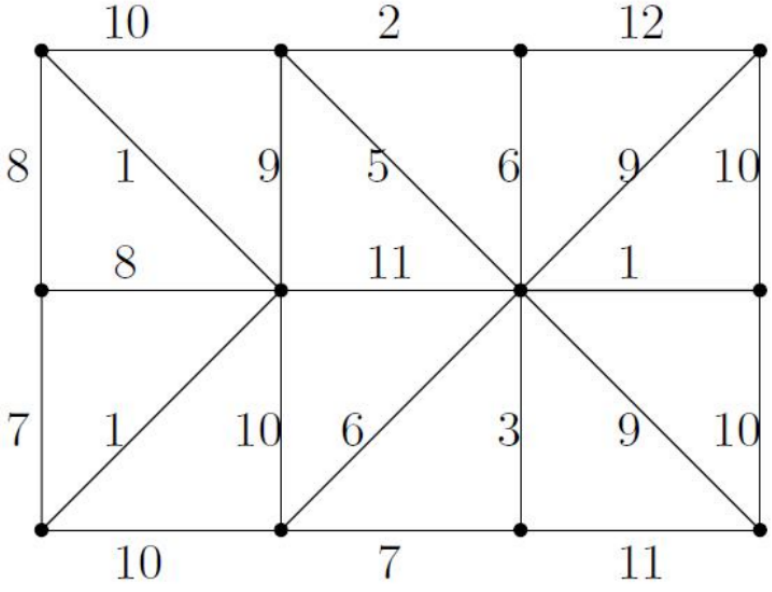
- Teilmenge von Kanten A des MST ist immer ein einziger Baum
- In jedem Schritt, finde leichte Kante, die Baum mit einem neuem Knoten verbindet
  - Sei V_A Menge der Knoten, die in A erreichbar
  - Finde leichte Kante über Schnitt $(V_A, V - V_A)$

Algorithmus von Prim



Beispielaufgabe

Markieren Sie im untenstehenden gewichteten Graphen die Kanten eines minimalen Spannbaums. Geben Sie den Namen des Algorithmus an, den Sie verwenden, und schreiben Sie die Reihenfolge auf, in der Sie die Kanten zum Spannbaum hinzufügen.
2 Punkte



Flussnetzwerke

- Gerichteter Graph $G = (V, E)$
  - $(u, v) \in E \rightarrow (v, u) \notin E$
    (keine antiparallelen Kanten)
  - für alle $u \in V$: $(u, u) \notin E$ (keine Schlingen)
- Jede Kante (u, v) hat Kapazität $c(u, v) \geq 0$
  - Konvention: $(u, v) \notin E \rightarrow c(u, v) = 0$
- 2 Ausgezeichnete Knoten
  - Quelle $s \in V$
  - Senke $t \in V$
- Annahme: für jeden Knoten v gibt es einen Pfad $s \rightsquigarrow v \rightsquigarrow t$ von s über v nach t

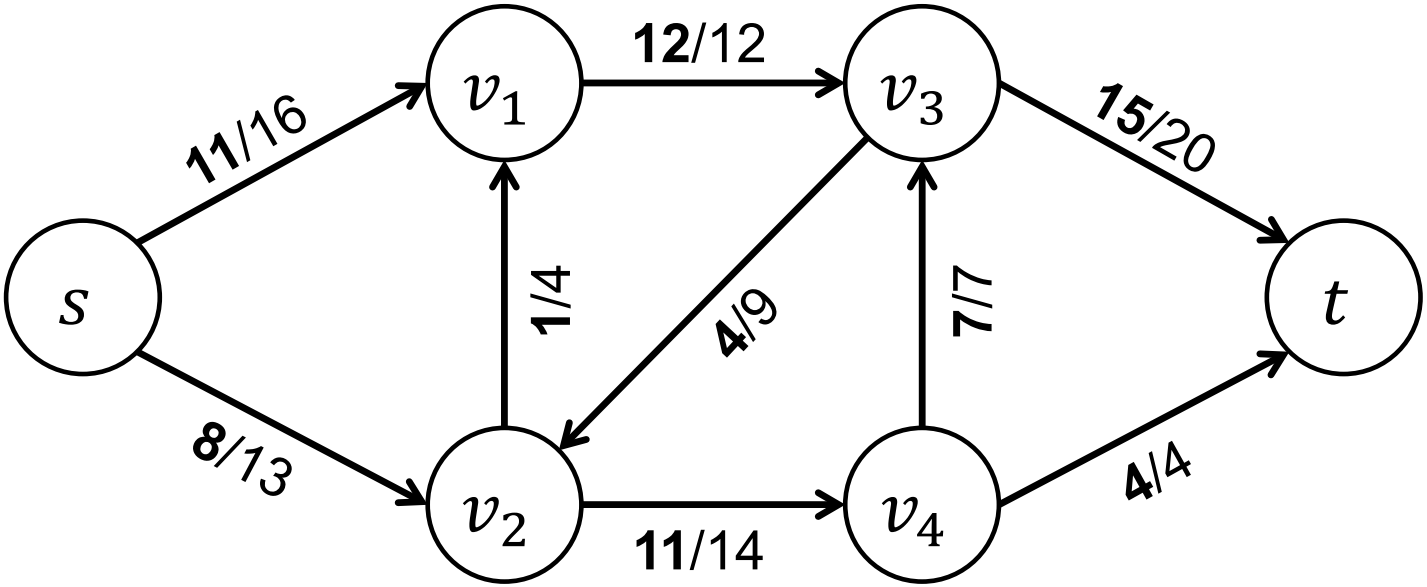
Fluss in Flussnetzwerk

- Fluss in Flussnetzwerk $G = (V, E)$: Funktion $f: V \times V \to \mathbb{R}$, die folgende Bedingungen erfüllt
  - Kapazitätsbedingung
    Für alle $(u, v) \in V \times V$: $0 \leq f(u, v) \leq c(u, v)$
    Wenn $(u, v) \notin E$, dann $f(u, v) = 0$
  - Flusserhaltung
    Für alle $u \in V - \{s, t\}$ gilt:

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

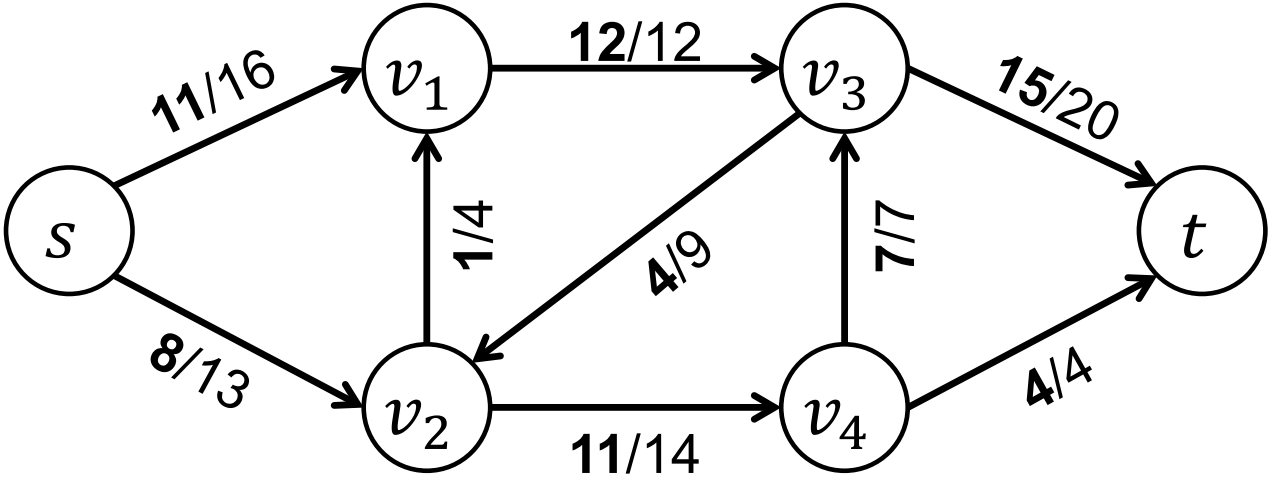
«Eingehender Fluss» = «Ausgehender Fluss»

Fluss in Flussnetzwerk

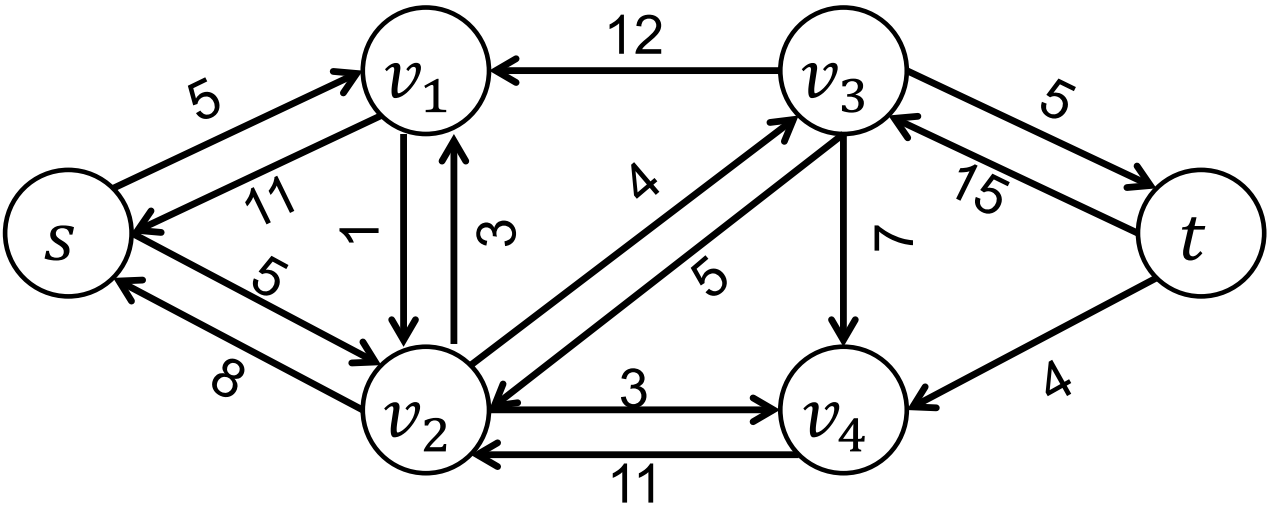


Jede Kante ist annotiert mit
Fluss/Kapazität, d.h. $\boldsymbol{f(u,v)}/c(u,v)$

Restnetzwerke

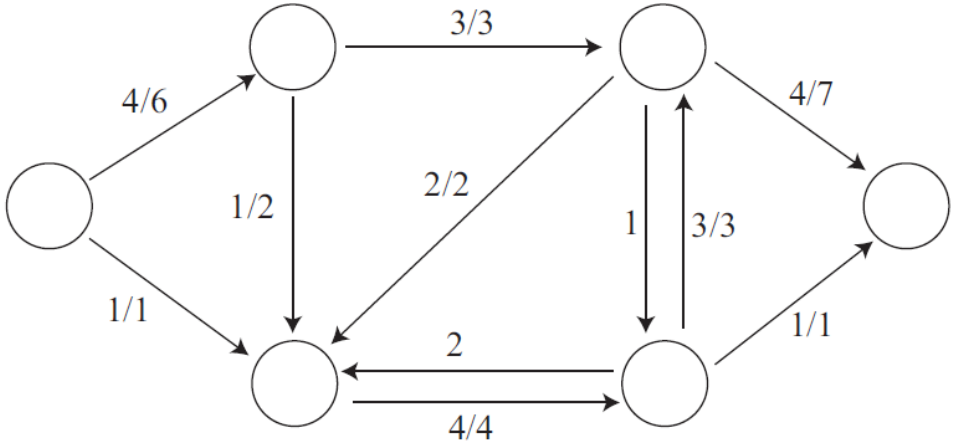


G_f



Beispielaufgabe

Betrachten Sie folgendes Flussnetzwerk mit einem gegebenen Fluss. Jede Kante ist mit ihrem Fluss und Ihrer Kapazität gemäss (*Fluss/Kapazität*) bezeichnet. Da der Fluss asymmetrisch definiert ist, ist jeweils nur der positive Wert in eine Richtung explizit bezeichnet.



a) Geben Sie den Wert des Flusses an. **1 Punkt**

b) Geben Sie die Kanten und die Kapazität eines minimalen Schnittes an. **2 Punkte**

c) Skizzieren Sie das Restnetzwerk und finden Sie einen erweiternden Pfad. Bezeichnen Sie den Fluss entlang des erweiternden Pfads, dessen Wert der Restkapazität des Pfads entspricht. **2 Punkte**

d) Addieren Sie den Fluss entlang des erweiternden Pfads zum ursprünglichen Fluss. Was ist der Wert des neuen Flusses? Ist dies ein maximaler Fluss? Begründen sie. **2 Punkte**

Prüfung

- Donnerstag, 7. Juni, 16-18 Uhr, Hörsaal A6

Viel Erfolg!