AUD Klausurblatt SoSe 2020 | Janson/Fischlin

Definitionen/Wissen

Sortier-Algorithmen in einem Satz

- *InsertionSort*: Sortierthalten der linken Teilfolge, neuen Wert an richtige Position einfügen
- *BubbleSort*: Vergleiche Paare von benachbarten Schlüsselwerten
- *SelectionSort*: Wähle kleinstes Element und tausche es nach vorne
- *MergeSort*: Teilen, sortiertes Zurückschreiben in Array
- *QuickSort*: Vergleich der Werte mithilfe PivotElement, rekursiver Aufruf auf Teilarray

Asymptotik

- $\frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$
- $\frac{f(n)}{g(n)} : konvergent \Rightarrow f(n) = O(g(n))$
- $o(n) \in O(n) \Rightarrow$ schließt alle anderen aus
- $f(n) = g(n) \Rightarrow f(n) = \Theta(g(n))$

Master-Theorem

- *Anwendbar?*
  - $a \geq 1$ und konstant?
  - $b > 1$ und konstant?
  - $f(n)$ positiv?
- *Vorgehen*
  - Berechnung von $log_b(a)$
  - Vergleich mit $f(n)$
    - $f(n)$ **polynomial kleiner** als $n^{log_b(a)}$ $(f(n) = O(n^{log_b(a-\epsilon)}),\ \epsilon > 0)$ $\Rightarrow T(n) = \Theta(n^{log_b(a)})$
    - $f(n)$ und $n^{log_b(a)}$ **gleiche Größe** $(f(n) = \Theta(n^{log_b(a)})$ $\Rightarrow T(n) = \Theta(n^{log_b(a)} lg(n))$
    - $f(n)$ **polynomial größer** als $n^{log_b(a)}$ und $af(\frac{n}{b}) \leq c\ f(n),\ (\epsilon < 1)$ $(f(n) = \Omega(n^{log_b(a+\epsilon)}),\ \epsilon > 0)$ $\Rightarrow T(n) = \Theta(f(n))$

String-Matching

Allgemein

- durchsuchender Text: Array `T` der Länge `lenTxt`
- Textmuster: Array `P` der Länge `lenPat` $\leq$ `lenTxt`
- Gesucht: alle gültigen Verschiebungen mit denen `P` in `T` auftaucht
- Rückgabe: alle `sft` $\in \mathbb{N}$, sodass `T[sft,...,sft+lenPat-1] = P` gilt

NaiveStringMatching

Pseudocode:

NaiveStringMatching(T,P)

```
1  lenTxt = length(T)
2  lenPat = length(P)
3  L = empty
4  FOR sft = 0 TO lenTxt - lenPat DO
5      isValid = TRUE
6      FOR j = 0 TO lenPat - 1 DO
7          IF P[j] ≠ T[sft+j] THEN
8              isValid = FALSE
9      IF isValid THEN
10         L = append(L, sft)
11 RETURN L
```

Beispiel: `T=[h,e,h,e,h,h,h,e,y,h]`, `P=[h,e,h]`

sft	$T[sft, \ldots, sft + lenPat - 1] \stackrel{?}{=} P$	L
0	**true**	[0]
1	**false**	[0]
2	**true**	[0, 2]
3	**false**	[0, 2]
4	**false**	[0, 2]
5	**false**	[0, 2]
6	**false**	[0, 2]
7	**false**	[0, 2]

Rabin-Karp-Algorithmus

Pseudocode:

RobinKarpMatch(T,P,q)

```
1  n = T.length
2  m = P.length
3  h = 10^(m−1) (mod q)
4  p = 0, t_0 = 0, L = empty
5  FOR i = 0 TO m-1 DO
6      p = (10p + P[i]) (mod q)
7      t_0 = (10t_0 + T[i]) (mod q)
8  FOR s = 0 TO n - m DO
9      IF p == t_s THEN
10         b = TRUE
11         FOR j = 0 TO m - 1 DO
12             IF P[j] ≠ T[s+j] THEN
13                 b = FALSE
14                 BREAK
15         IF b THEN
16             L.add(s)
17     IF s < n - m THEN
18     t_{s+1} = (10(t_s - T[s]h) + T[s+m]) (mod q)
19 RETURN L
```

Beispiel: `P = [7,3,4]`, `T = [6,9,1,7,3,4,5,0,9,4,6,2,4,8,7,3,4]`, `q = 13`, Ergebnis `L = [3,14]`

s	$t_s \pmod q$	$t_s == p \pmod q$	$T[s, \ldots, s+m-1] == P$	Treffer?
0	2	false		
1	7	false		
2	4	false		
3	6	**true**	**true**	Ja
4	7	false		
5	8	false		
6	2	false		
7	3	false		
8	10	false		
9	7	false		
10	0	false		
11	1	false		
12	6	**true**	false	Unecht
13	2	false		
14	6	**true**	**true**	Ja

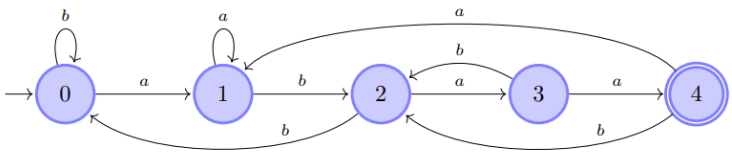
FiniteAutomationMatching

Pseudocode:

FiniteAutomationMatching(T, δ, lenPat)

```
1 lenTxt = length(T)
2 L = empty
3 st = 0
4 FOR sft = 0 TO lenTxt - 1 DO
5     st = δ(st, T[sft])
6     IF st = lenPat THEN
7         L = append(L, sft - lenPat + 1)
8 RETURN L
```

Beispiel: Σ = {a,b}, P = [a,b,a,a], T = [a,a,b,a,b,a,a,b,a,a]



sft	$T[sft]$	st	L
0	a	1	[]
1	a	1	[]
2	b	2	[]
3	a	3	[]
4	b	2	[]
5	a	3	[]
6	a	4	[3]
7	b	2	[3]
8	a	3	[3]
9	a	4	[3, 6]

Queue mithilfe von zwei Stacks

enqueue pusht auf den ersten Stack.
dequeue wird als *pop*(S_2) definiert.
Falls der zweite Stack leer ist werden alle Elemente aus S_1 geholt und in S_2 überführt.
Dann wird das erste Element von S_2 ausgegeben.

new(Q)

```
11: S_1 = new(S_1)
12: S_2 = new(S_2)
13: Q = [S_1, S_2]
14: return Q
```

isEmpty(Q)

```
21: parse Q = [S_1, S_2]
22: b_1 = isEmpty(S_1)
23: b_2 = isEmpty(S_2)
24: return b_1 ∧ b_2
```

enqueue(Q, x)

```
31: parse Q = [S_1, S_2]
32: push(S_1, x)
```

dequeue(Q)

```
41: parse Q = [S_1, S_2]
42: if isEmpty(Q) then
43:     return Error
44: if isEmpty(S_2) then
45:     while ¬isEmpty(S_1) do
46:         push(S_2, pop(S_1))
47: return pop(S_2)
```

Stack mithilfe von zwei Queues

Eine der beiden Queues bleibt immer leer (anfangs Q_2)
push fügt Wert immer der leeren Queue hinzu.
pop holt alle Elemente bis auf das letzte aus der Queue zurück und fügt sie in die leere ein.
Das letzte Element wird dann ausgegeben.

new(S)

```
11: Q_1 = new(Q_1)
12: Q_2 = new(Q_2)
13: S = [Q_1, Q_2]
14: return S
```

isEmpty(S)

```
21: parse S = [Q_1, Q_2]
22: b_1 = isEmpty(Q_1)
23: b_2 = isEmpty(Q_2)
24: return b_1 ∧ b_2
```

push(S, x)

```
31: parse S = [Q_1, Q_2]
32: if isEmpty(Q_1) then
33:     enqueue(Q_2, x)
34: else
35:     enqueue(Q_1, x)
```

pop(S)

```
41: parse S = [Q_1, Q_2]
42: if isEmpty(S) then
43:     return Error
44: if isEmpty(Q_2) then
45:     t = dequeue(Q_1)
46:     while ¬isEmpty(Q_1) do
47:         enqueue(Q_2, t)
48:         t = dequeue(Q_1)
49: else
50:     t = dequeue(Q_2)
51:     while ¬isEmpty(Q_2) do
52:         enqueue(Q_1, t)
53:         t = dequeue(Q_2)
54: return t
```

Graphen - Adjazenzmatrix/-liste

Umformung Liste → Matrix

listToMatrix(L)

```
1 nNodes = length(L)
2 M = newMatrix(nNodes, nNodes, 0)
3 FOR i = 0 to nNodes - 1 DO
4     node = L[i]
5     WHILE node ≠ nil DO
6         j = node.key
7         M[i,j] = 1
8         node = node.next
9 RETURN M
```

Umformung Matrix → Liste

matrixToList(M)

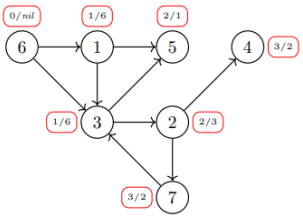
```
1 nNodes = rows(M)
2 L = newArray(nNodes)
3 FOR i = 0 to nNodes - 1 DO
4     L[i] = newList()
5     FOR j = 0 TO nNodes - 1 DO
6         IF M[i,j] = 1 THEN
7             insert(L[i],j)
8 RETURN L
```

0.1 Graphenalgorithmen

0.1.1 Breadth-First-Search

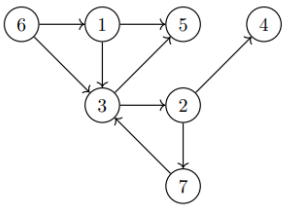
- Besuche zuerst alle unmittelbaren Nachbarn, dann deren Nachbarn, usw.
- Funktionsweise:
  - setzt alle Knoten auf Weiß und Distanz auf $+\infty$
  - Startknoten grau, Distanz grau, Vorgänger nil
  - Fügt alle Nachbarn einer Queue hinzu die weiß sind und passt deren Distanz an (dist+1)
  - Setzt Knoten auf Schwarz, wenn alle Nachbarn hinzugefügt und holt sich neuen Knoten aus Queue
  -

Iteration	u	v	Q
0	–	□	[6]
1	6	1, 3	[1, 3]
2	1	5	[3, 5]
3	3	2	[5, 2]
4	5	□	[2]
5	2	4, 7	[4, 7]
6	4	□	[7]
7	7	□	[]



0.1.2 Depth-First-Search

- Besuche zuerst alle noch nicht besuchten Nachfolgeknoten
- "Laufe so weit wie möglich weg vom aktuellen Knoten"
- Funktionsweise DFS (Initialaufruf):
  - Setzt alle Knoten auf Weiß, Vorgänger auf nil, time auf 0
  - Ruft innerhalb einer for-Schleife auf jedem Knoten DFS-VISIT auf
- Funktionsweise DFS-Visit:
  - Wird min. einmal auf jedem Knoten aufgerufen
  - Erhöht zuallererst time um 1
  - setzt discovery time der Node auf time und Farbe auf grau
  - Ruft innerhalb For-Schleife für jeden weißen Nachbarn DFS-Visit auf
  - Falls die rekursive For-Schleife abgeschlossen ist -> Node Schwarz / time + 1
  - Abspeichern der finish-Zeit (u.finish = time)

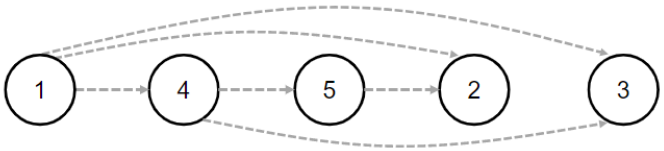


Knoten	Entdeckungszeit	Abschlusszeit	Vorgängerknoten
1	1	12	*nil*
2	3	8	3
3	2	11	1
4	4	5	2
5	9	10	3
6	13	14	*nil*
7	6	7	2

- Anwendungen DFS: `Topo-Sort`
  - nur für directed acyclic graphs
  - Kanten zeigen immer nur nach rechts

TOPOLOGICAL-SORT(G)

```
1 newLinkedList(L)
2 run DFS(G) but, each time a node is finished, insert in front of L
3 RETURN L.head
```

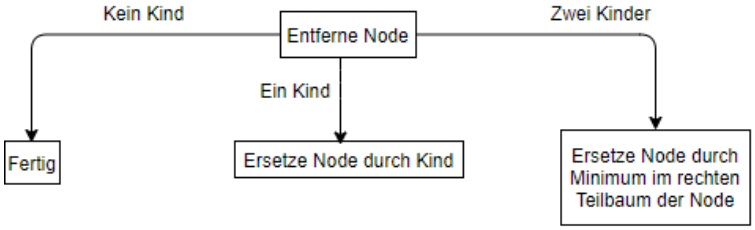


- Starke Zusammenhangskomponenten
  - SCCs sind nur in eine Richtung verbunden und verschiedene sind disjunkt
  - Algorithmus lässt einmal DFS laufen und dann nochmal auf dem transponierten Graphen
  - DFS auf transponierten Graphen aber in abnehmender finish-time des ersten Graphen (Transponierter Graph: Umdrehen der Kantenrichtungen)
  - Ausgabe jedes DFS-Baumes als ein SCC (Verwendung des Outputs transponierten Graphens)

Baumoperationen

BST

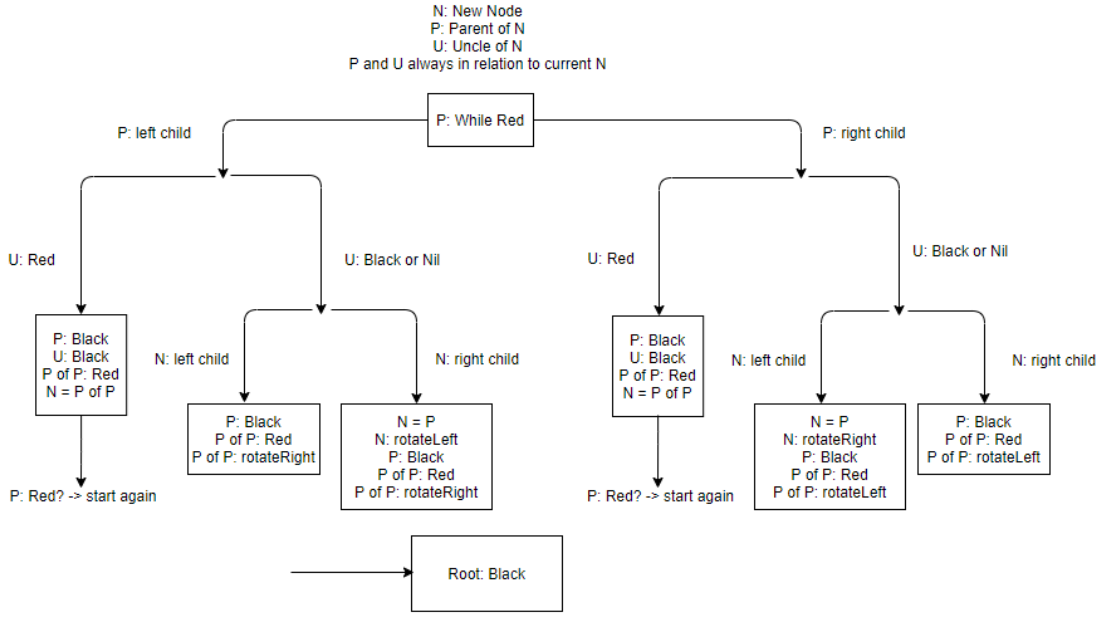
Löschen



RBT

Insert

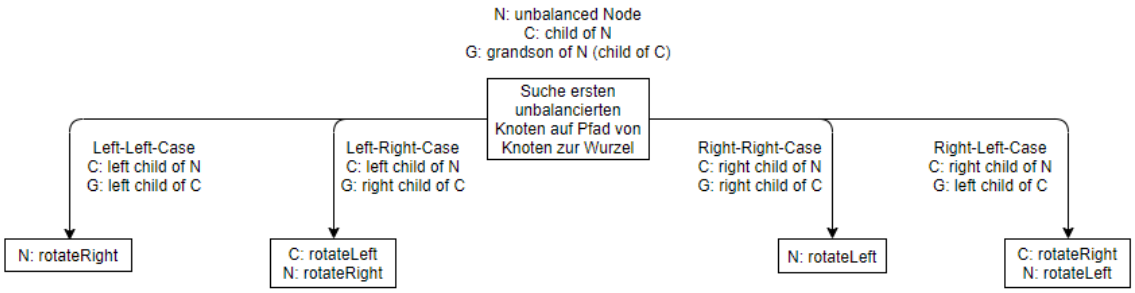
Wie im BST, neuen Knoten rot färben, danach FixUp:



AVL-Bäume

Einfügen/Löschen

Einfügen und Löschen wie beim BST, danach jeweils Rebalancieren:



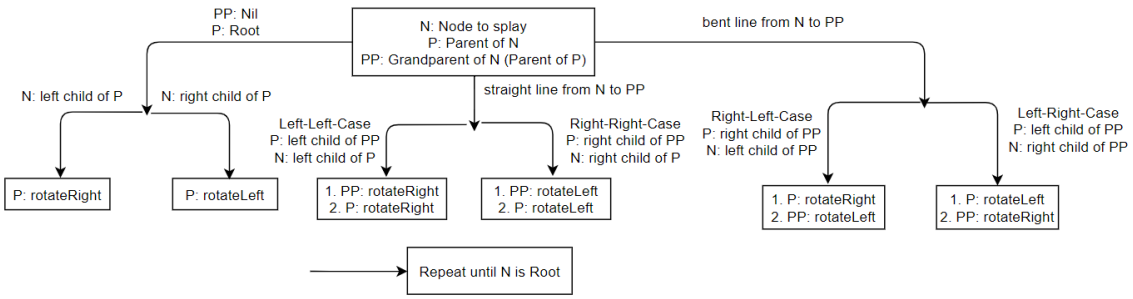
Beachte beim Löschen:

- Wahl von C: größter Teilbaum von N (eindeutig, da N unbalanciert)
- Wahl von G: größter Teilbaum von C (nicht eindeutig, Wahl Rechts-Rechts/Links-Links)
- **Potenziell** müssen mehrere Knoten bearbeitet werden ⇒ alle Knoten bis Wurzel prüfen

Splay-Bäume

- Suchen: Spüle gesuchten Knoten an die Wurzel (alternativ zuletzt besuchten Knoten)
- Einfügen: Einfügen nach BST-Regeln und danach Hochspülen des Knotens
- Löschen:
  1. zu löschenden Knoten hochspülen
  2. Knoten löschen
     - Falls nur ein Kind: Dieses Kind neue Wurzel und fertig
     - Falls zwei Kinder: Spüle größten Knoten im linken Teilbaum hoch
       Hänge danach beide Teilbäume an diesen Knoten

Spülen

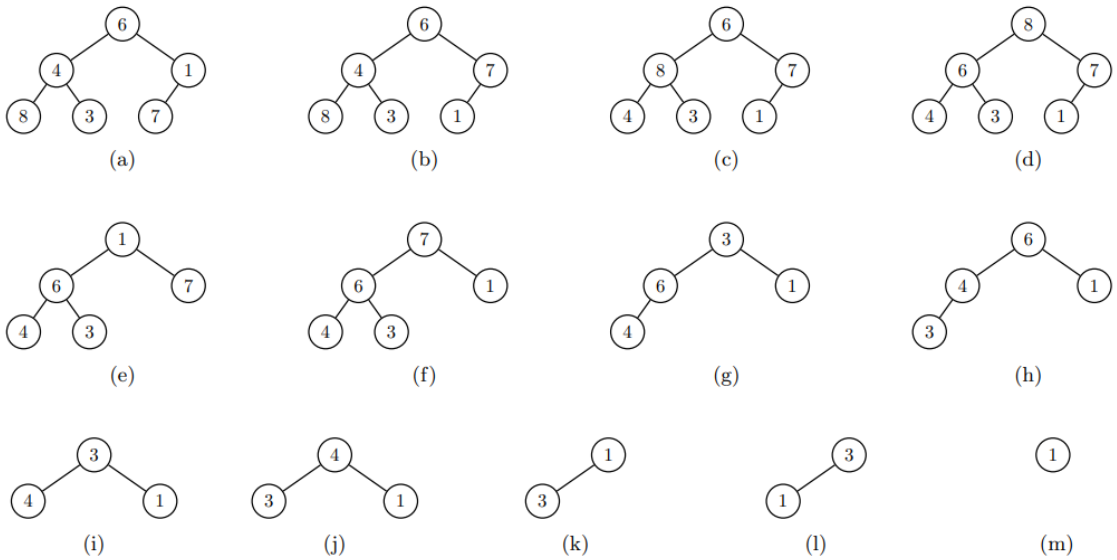


Heaps

Heap-Sort

1. Array wird als Heap aufgefasst
2. Heapeigenschaft wird wiederhergestellt (Heapify)
3. Extrahieren der Wurzel (Maximum) und Ersetzen durch "letztes" Blatt
4. Wieder Heapify um Wert an die richtige Stelle zu rücken
5. Falls der Baum noch nicht leer ist, gehe zu Schritt 3

Heapify: beginnend bei `ceil((H.length-1)/2) - 1` bis 0: vertausche nach unten, falls Parent kleiner als Child



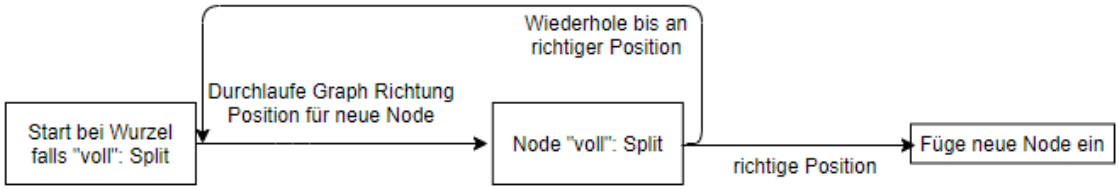
B-Bäume vom Grad `t`

Knoten zwischen `[t-1, ...,2t-1]` Werte (Wurzel: `[1, ...,2t-1]`)

Einfügen

`Split:`

- Aufbrechen der vollen (`2t-1`) Node
- Hinzufügen der mittleren Node zur Elternnode
- Aus den anderen Nodes entstehen nun jeweils einzelne Kinder
- Splitten an der Wurzel erzeugt neue Wurzel und erhöht Baumhöhe um 1



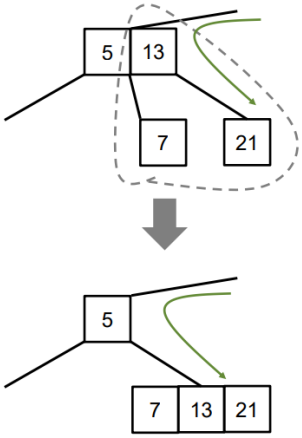
Löschen

1. Start bei Wurzel

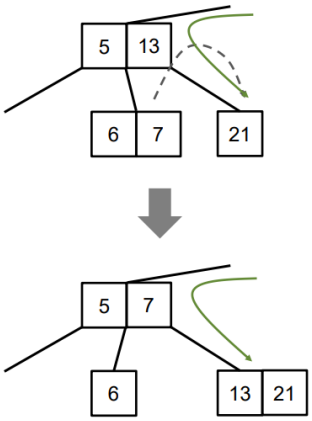
   Wurzel: `1` Wert | Kinder der Wurzel: `t-1` Werte → Verschmelze Kinder und Wurzel

2. Durchlaufe Graph von Node bis zum löschenden Knoten
3. Überprüfe bei jeder Node:

Verschmelzen
Kind + Geschwister `t-1` Werte
$t = 2$

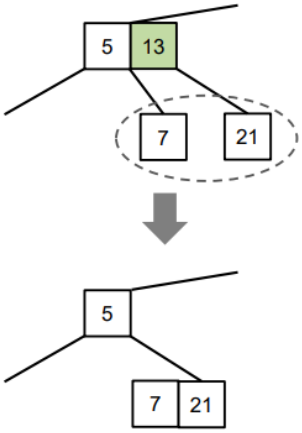


Rotation
Kind nur `t-1` Werte
Geschwister jedoch mehr als `t-1` Werte
$t = 2$

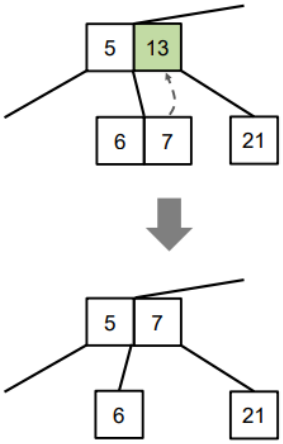


4. Knoten gefunden:
   - **Löschen im Blatt:** Einfach entfernen, fertig
   - **Löschen im inneren Knoten:**

Verschmelzen
Beide Kinder `t-1` Werte
→ Kindknoten verschmelzen
$t = 2$



Verschieben
Eines der Kinder mehr als `t-1` Werte
Größten Wert vom linken Kind nach oben kopieren oder
Kleinsten Wert vom rechten Kind nach oben kopieren
Potentiell rekursiv nach unten suchen
$t = 2$



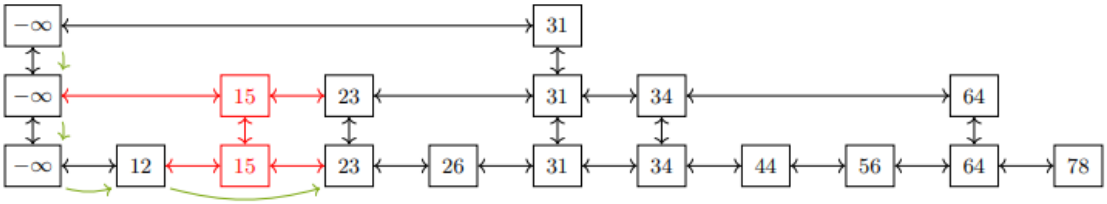
Randomized Data Structures

Skip-Lists

Einfügen

- Suche richtige Position in unterster Liste
- Füge Wert ein
- Füge ihn auch in drüberliegende Skip-Lists ein, falls die zufällige Zahl $< p$ ist

$p = \frac{1}{2}$, Zufallswerte: $0.33, 0.82 \rightarrow 15$ wird in eine höhere Skip-List eingefügt ($0.33 < \frac{1}{2}$, $0.82 > \frac{1}{2}$)



Löschen

- Wert und alle Vorkommen in Expresslisten entfernen

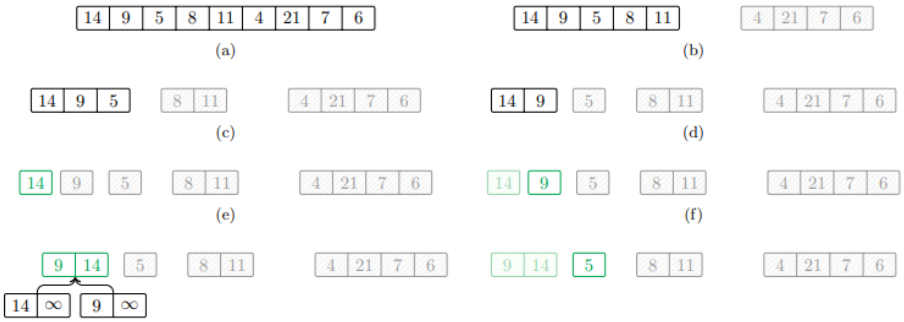
Anwendungsbeispiel

Sorting

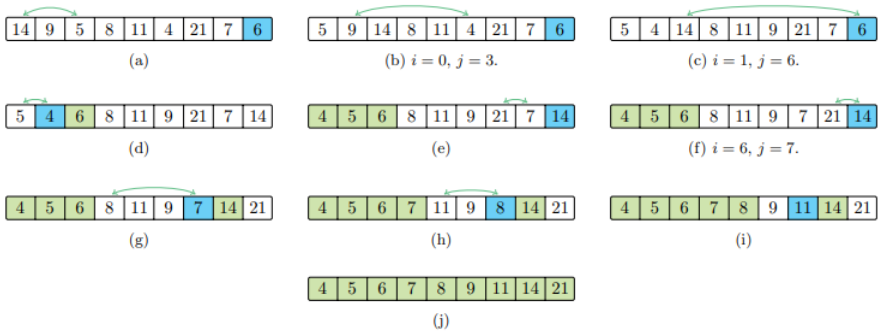
Insertion Sort

$j = 1$:	auf	Baum	Daten	Landesbibliothek	Haus	sortieren
$j = 2$:	Baum	auf	Daten	Landesbibliothek	Haus	sortieren
$j = 3$:	Daten	Baum	auf	Landesbibliothek	Haus	sortieren
$j = 4$:	Landesbibliothek	Daten	Baum	auf	Haus	sortieren
$j = 5$:	Landesbibliothek	Daten	Baum	Haus	auf	sortieren
$j = 6$:	Landesbibliothek	sortieren	Daten	Baum	Haus	auf

Merge Sort

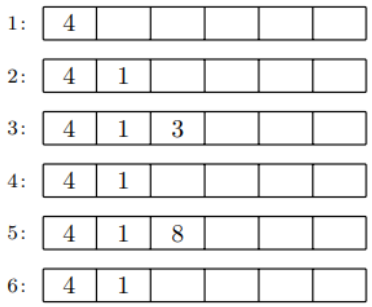


Quick Sort

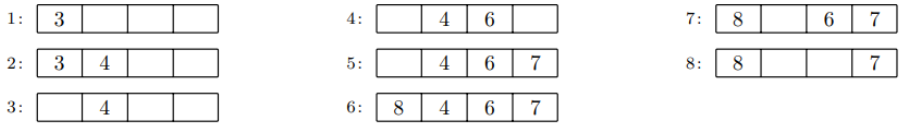


Basic Data Structures

Stacks



Queues



Pseudocode

Sorting

Insertion Sort(A)

```
FOR j = 1 TO A.length - 1
  key = A[j]
  // Füge A[j] in die sortierte Sequenz A[0...j-1] ein
  i = j - 1
  WHILE i >= 0 and A[i] > key
      A[i + 1] = A[i]
      i = i - 1
  A[i + 1] = key
```

BubbleSort(A)

```
FOR i = 0 TO A.length - 2
    FOR j = A.length - 1 DOWNTO i + 1
        IF A[j] < A[j-1]
            SWAP(A[j], A[j-1])
```

SelectionSort(A)

```
FOR i = 0 TO A.length - 2
    k = i
    FOR j = i + 1 TO A.length - 1
        IF A[j] < A[k]
            k = j
    SWAP(A[i], A[k])
```

Merge Sort

MergeSort(A, p, r)

```
IF p < r
    q = ⌊(p+r)/2⌋ // Teilen in 2 Teilfolgen
    MERGE-SORT(A,p,q) // Sortieren der beiden Teilfolgen
    MERGE-SORT(A,q+1,r)
    MERGE(A,p,q,r) // Vereinigung der beiden sortierten Teilfolgen
```

MERGE(A,p,q,r)

```
// Geteiltes Array an Stelle q
n_1 = q - p + 1
n_2 = r - q
Let L[0...n_1] and R[0...n_2] be new arrays
FOR i = 0 TO n_1 - 1 // Auffüllen der neu erstellten Arrays
    L[i] = A[p + i]
FOR j = 0 TO n_2 - 1
    R[j] = A[q + j + 1]
L[n_1] = ∞ // Einfügen des Sentinel-Wertes
R[n_2] = ∞
i = 0
j = 0
FOR k = p TO r  // Eintragweiser Vergleich der Elemente
    IF L[i] ≤ R[j]
        A[k] = L[i] // Sortiertes Zurückschreiben in Original-Array
        i = i + 1
    ELSE
        A[k] = R[j]
        j = j + 1
```

Quicksort

QUICKSORT(A,p,r)

```
IF p < r    // Überprüfung, ob Teilarray leer ist
    q = PARTITION(A,p,r)
    QUICKSORT(A,p,q-1)
    QUICKSORT(A,q+1,r)
```

PARTITION(A,p,r)

```
x = A[r]    // Wahl des Pivotelements
i = p - 1   // Index i setzen
FOR j = p TO r - 1 // Auffüllen des Teilarrays mit Elementen
    IF A[j] ≤ x
        i = i + 1
        SWAP(A[i], A[j]) /
SWAP(A[i+1], A[r]) // Tausch des Pivotelements
RETURN i + 1 // Neuer Index des Pivotelements
```

0.2 Graphenalgorithmen

0.2.1 Breadth-First Search

BFS(G,s)

```
FOREACH u in V-{s} DO
    u.color = WHITE;          // Weiß = noch nicht besucht
    u.dist = +∞               // Setzen der Distanzen auf Unendlich
    u.pred = nil;             // Setzen der Vorgänger auf nil
s.color = GRAY;               // Anfang bei Startnode
s.dist = 0;
s.pred = nil;
newQueue(Q);
enqueue(Q,s);
WHILE !isEmpty(Q) DO
    u = dequeue(Q);
    FOREACH v in adj(G,u) DO
        IF v.color == WHITE THEN
            v.color == GRAY;
            v.dist = u.dist+1;
            v.pred = u;
            enqueue(Q,v);
    u.color = BLACK;              // Knoten abgearbeitet
```

0.2.2

DFS(G)

```
FOREACH u in V DO
    u.color = WHITE;
    u.pred = nil;
time = 0;                 // time hier als globale Variable
FOREACH u in v DO
    IF u.color == WHITE THEN
        DFS-VISIT(G,u)  // Start eines rekursiven Aufrufs
```

DFS-VISIT(G,u)

```
time = time + 1;
u.disc = time;            // discovery time
u.color = GRAY;
FOREACH v in adj(G,u) DO
    IF v.color == WHITE THEN
        v.pred = u;
        DFS-VISIT(G,v);
u.color = BLACK;
time = time + 1;
u.finish = time;          // finish time
```

MasterTheorem - Beispiele

Begründen Sie für jede der folgenden Rekursionsgleichungen $T(n)$, ob Sie das Mastertheorem anwenden können oder nicht. Benutzen Sie gegebenenfalls das Mastertheorem, um eine asymptotische Schranke für $T(n)$ zu bestimmen. Die entsprechenden Anfangsbedingungenen (also die Werte $T(1)$ in allen Beispielen und, in (f), (h) und (i), zusätzlich $T(2)$) sind dabei vorgegebene Konstanten.

(a) $T(n) = 3T\left(\frac{n}{2}\right) + n^2$ (für $n > 1$);
(b) $T(n) = 4T\left(\frac{n}{2}\right) + n^2$ (für $n > 1$);
(c) $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n$ (für $n > 1$);
(d) $T(n) = \frac{1}{2}T\left(\frac{n}{2}\right) + 1/n$ (für $n > 1$);
(e) $T(n) = \sqrt{2}T\left(\frac{n}{2}\right) + \log(n)$ (für $n > 1$);
(f) $T(n) = 2T\left(\frac{n}{\log(n)}\right) + n^2$ (für $n > 2$);
(g) $T(n) = 64T\left(\frac{n}{8}\right) - n^2\log(n)$ (für $n > 1$);
(h) $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n}{\log(n)}$ (für $n > 2$);
(i) $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log(n)}$ (für $n > 2$);
(j) $T(n) = 6T\left(\frac{n}{3}\right) + n^2\log(n)$ (für $n > 1$);
(k) $T(n) = 2T\left(\frac{4n}{3}\right) + n$ (für $n > 1$);
(l) $T(n) = T\left(\frac{n}{2}\right) + 2T\left(\frac{n}{4}\right) + n$ (für $n > 1$).

Lösung. (a) Hier können wir das Mastertheorem anwenden: Es sind $a = 3 \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = n^2 \geq 0$ für alle $n \in \mathbb{N}$. Daraus folgt $\log_b(a) < 1{,}59 < 2$. Wir sind also im Fall 3 des Mastertheorems, denn $f(n) \in \Omega\big(n^{1{,}59+\varepsilon}\big)$ für ein $\varepsilon > 0$ (z. B. mit $\varepsilon = 1/10$), vorausgesetzt, dass die Regularitätsbedingung erfüllt ist. Diese gilt aber für $c = \frac{3}{4} < 1$ und alle $n \in \mathbb{N}$ (da $3\left(\frac{n}{2}\right)^2 = \frac{3}{4}n^2$), und wir erhalten $T(n) \in \Theta\big(n^2\big)$.

(b) Hier können wir das Mastertheorem anwenden: Es sind $a = 4 \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = n^2 \geq 0$ für alle $n \in \mathbb{N}$. Daraus folgt $\log_b(a) = 2$. Wir sind also im Fall 2 des Mastertheorems, denn natürlich ist $f(n) \in \Theta\big(n^2\big)$, und wir erhalten $T(n) \in \Theta\big(n^2\log(n)\big)$.

(c) Hier können wir das Mastertheorem nicht anwenden, weil $a = 2^n$ nicht konstant ist.

(d) Auch hier können wir das Mastertheorem nicht anwenden, weil $a = \frac{1}{2}$ zwar konstant ist, aber $a < 1$.

(e) Hier können wir das Mastertheorem wieder anwenden: Es sind $a = \sqrt{2} \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = \log(n) \geq 0$ für alle $n \in \mathbb{N}$. Daraus folgt $\log_b(a) = \frac{1}{2}$. Wir sind also im Fall 1 des Mastertheorems, denn $f(n) \in O\big(n^{1/2-\varepsilon}\big)$ für ein $\varepsilon > 0$ (z. B. mit $\varepsilon = 1/10$), und wir erhalten $T(n) \in \Theta(\sqrt{n})$.

(f) Hier können wir das Mastertheorem wieder nicht anwenden, da $b = \log(n)$ nicht konstant ist.

(g) Hier können wir das Mastertheorem auch nicht anwenden, weil $f(n) = -n^2\log(n)$ nicht positiv ist.

(h) Hier können wir das Mastertheorem wieder anwenden: Es sind $a = 4 \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = \frac{n}{\log(n)} \geq 0$ für alle $n \in \mathbb{N}_{\geq 2}$. Daraus folgt $\log_b(a) = 2$. Wir sind also im Fall 1 des Mastertheorems, denn $f(n) \in O\big(n^{2-\varepsilon}\big)$ für ein $\varepsilon > 0$ (z. B. mit $\varepsilon = 1/2$), und wir erhalten $T(n) \in \Theta\big(n^2\big)$.

(i) Hier können wir das Mastertheorem wieder nicht anwenden, weil keine der drei Bedingungen erfüllt ist. Die Anfangsbedingungen passen zwar, aber keiner der drei Fälle trifft zu. In der Tat, es sind $a = 2 \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = \frac{n}{\log(n)} \geq 0$ für alle $n \in \mathbb{N}_{\geq 2}$. Daraus folgt $\log_b(a) = 1$. Wir argumentieren jetzt, dass keiner der drei Fälle des Mastertheorems zutrifft:

- Wäre $f(n) \in O\big(n^{1-\varepsilon}\big)$ für ein $\varepsilon > 0$, dann gäbe es $C > 0$ und $N_0 \in \mathbb{N}$ sodass, für alle $n \geq N_0$, $\frac{n}{\log(n)} \leq Cn^{1-\varepsilon}$. Daraus würde $\frac{n^\varepsilon}{\log(n)} \leq C$ für alle $n \geq N_0$ folgen, was unmöglich ist, da $\lim_{n\to+\infty} \frac{n^\varepsilon}{\log(n)} = +\infty$. Somit gilt, für jedes $\varepsilon > 0$, $f(n) \notin O\big(n^{1-\varepsilon}\big)$, und Fall 1 des Mastertheorems trifft nicht zu.
- Wäre $f(n) \in \Theta(n)$, dann gäbe es $C > 0$ und $N_0 \in \mathbb{N}$ sodass, für alle $n \geq N_0$, $Cn \leq \frac{n}{\log(n)}$, also $C\log(n) \leq 1$ für alle $n \geq N_0$. Das ist auch unmöglich, denn $\lim_{n\to+\infty} C\log(n) = +\infty$. Somit gilt $f(n) \notin \Theta(n)$, und Fall 2 des Mastertheorem trifft auch nicht zu.
- Wäre $f(n) \in \Omega\big(n^{1+\varepsilon}\big)$ für ein $\varepsilon > 0$, dann gäbe es $C > 0$ und $N_0 \in \mathbb{N}$ sodass, für alle $n \geq N_0$, $Cn^{1+\varepsilon} \leq \frac{n}{\log(n)}$, also $Cn^\varepsilon\log(n) \leq 1$ für alle $n \geq N_0$. Das kann auch nicht sein, da wieder $\lim_{n\to+\infty} Cn^\varepsilon\log(n) = +\infty$. Somit gilt, für jedes $\varepsilon > 0$, $f(n) \notin \Omega\big(n^{1+\varepsilon}\big)$, und Fall 3 des Mastertheorems trifft ebenfalls nicht zu.

(j) Hier können wir das Mastertheorem wieder anwenden: Es sind $a = 6 \geq 1$ und $b = 3 > 1$ konstant, und es gilt $f(n) = n^2\log(n) \geq 0$ für alle $n \in \mathbb{N}$. Daraus folgt $\log_b(a) < 1{,}64 < 2$. Wir sind also im Fall 3 des Mastertheorems, denn $f(n) \in \Omega\big(n^{1{,}64+\varepsilon}\big)$ für ein $\varepsilon > 0$ (z. B. mit $\varepsilon = 1/10$), vorausgesetzt, dass die Regularitätsbedingung erfüllt ist. Diese gilt aber für $c = \frac{2}{3} < 1$ und alle $n \in \mathbb{N}$ (da $6\left(\frac{n}{3}\right)^2\log\left(\frac{n}{3}\right) \leq \frac{2}{3}n^2\log(n)$), und wir erhalten $T(n) \in \Theta\big(n^2\log(n)\big)$.

(k) Hier können wir das Mastertheorem wieder nicht anwenden, da $b = \frac{3}{4} < 1$.

(l) Auch hier können wir das Mastertheorem nicht anwenden, da die Rekursionsgleichung nicht die Form hat, die vom Mastertheorem abgedeckt wird. □