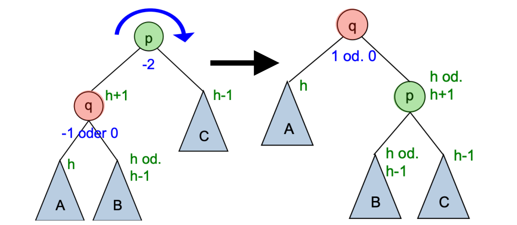
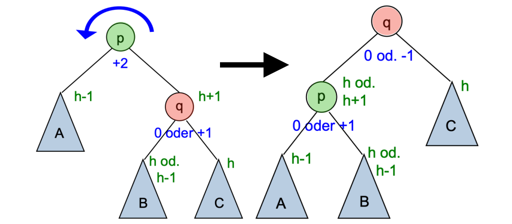
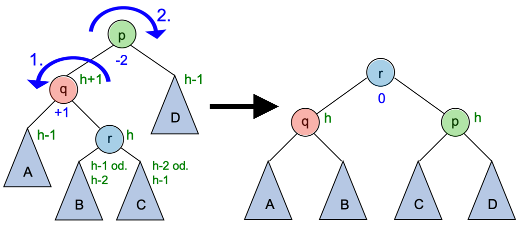
AVL: **H_diff = H(rechts)-H(Links)**
Leerer Teilbaum: H = -1



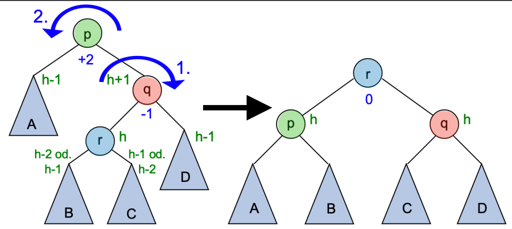
H_diff = -2
Linker Teilbaum = -1 || 0



H_diff = +2
Linker Teilbaum = 0 || +1



H_diff = -2
Linker Teilbaum = +1

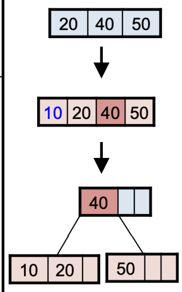


H_diff = +2
Linker Teilbaum = -1

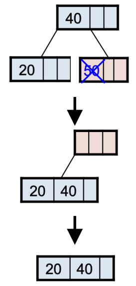
Komplexität:

	Best	Avg	Worst
Insert	log(n)	log(n)	log(n)
Search	O(1)	log(n)	log(n)
Delete	log(n)	log(n)	log(n)

B-Baum: [⌈m/2⌉ -1 , m-1] = [min max]
(m = gegebene Ordnung)



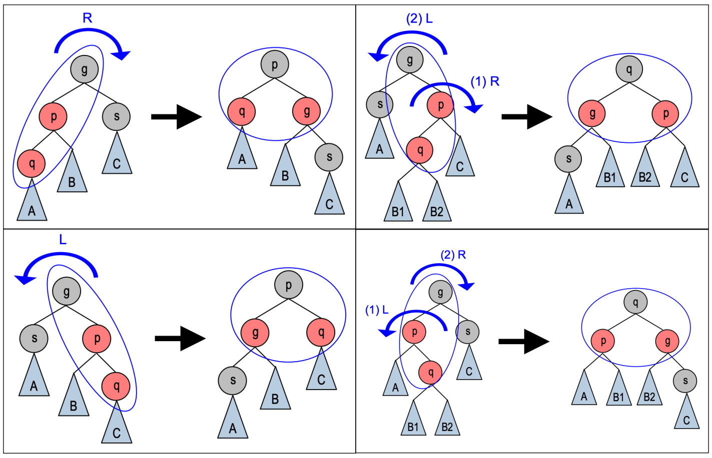
⌊n/2⌋ nach oben schieben



Nächst größeren Knoten suchen

Rot-Schwarz Baum:

- Die Wurzel ist immer schwarz
- Wenn beide kinder rot sind beim einfügen Farbwechsel vollziehen (rot nach oben schieben)
- 2-3-4 Baum zu RS indem die Mitte die root wird dann rote Kinder



Dijkstra Algorithmus:
Problem: alle Kürzesten Wege von Startknoten finden

1. Werte aller direkt erreichbaren Knoten speichern
2. Vorgänger speichern.
3. Aktuellen knoten als besucht markieren
4. Günstigsten Knoten auswählen
5. Alle direkt erreichbaren Knoten besuchen die nicht markiert sind.
6. Wert + Aktuellen Knotenwert speichern
7. Vorgänger speichern
8. j(3.)

	1	2	3	4	5	6
1		5	1	3		5
2					3	
3		3		1		
4		1			5	2
5						
6					1	

b	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	p[1]	p[2]	p[3]	p[4]	p[5]	p[6]
1	0	5	1	3	∞	5	-	1	1	1	-	1
3		4		2				3		3		
4		3			7	4		4			4	4
2					6						2	
6					5						6	
5												

Floyd Algorithmus:

D^{-1}

0	5	∞	4	∞
∞	0	-1	∞	-2
∞	1	0	∞	∞
∞	∞	-2	0	-1
1	∞	∞	∞	0

D^0

0	5	∞	4	∞
∞	0	-1	∞	-2
∞	1	0	∞	∞
∞	∞	-2	0	-1
1	6	∞	5	0

D^1

0	5	4	4	3
∞	0	-1	∞	-2
∞	1	0	∞	-1
∞	∞	-2	0	-1
1	6	5	5	0

D^2

0	5	4	4	3
∞	0	-1	∞	-2
∞	1	0	∞	-1
∞	-1	-2	0	-3
1	6	5	5	0

D^3

0	3	2	4	1
∞	0	-1	∞	-2
∞	1	0	∞	-1
∞	-1	-2	0	-3
1	4	3	5	0

D^4

0	3	2	4	1
-1	0	-1	3	-2
0	1	0	4	-1
-2	-1	-2	0	-3
1	4	3	5	0

P^{-1}

-	0	-	0	-
-	-	1	-	1
-	2	-	-	-
-	-	3	-	3
4	-	-	-	-

P^0

-	0	-	0	-
-	-	1	-	1
-	2	-	-	-
-	-	3	-	3
4	0	-	0	-

P^1

-	0	1	0	1
-	-	1	-	1
-	2	-	-	1
-	-	3	-	3
4	0	1	0	-

P^2

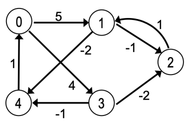
-	0	1	0	1
-	-	1	-	1
-	2	-	-	1
-	2	3	-	1
4	0	1	0	-

P^3

-	2	3	0	1
-	-	1	-	1
-	2	-	-	1
-	2	3	-	1
4	2	3	0	-

P^4

-	2	3	0	1
4	-	1	0	1
4	2	-	0	1
4	2	3	-	1
4	2	3	0	-




```
if ( D[i][j]  > D[i][k] + D[k][j] ) {
    D[i][j]  = D[i][k] + D[k][j];
    P[i][j]  = P[k][j];
}
```

Read: von Zeile nach Spalte
Ersetzen: Zeile + Spalte < Aktueller Wert
Bei Ersetzung, parent Wert aus aktueller Spalte übernehmen

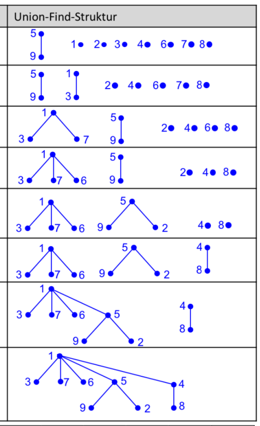
Kruskal Algorithmus:

Günstigste Kante finden und hinzufügen so lange bis alle knoten verbunden sind.

⚠️ Nur Knoten hinzufügen die KEINEN Zyklus erzeugen

	1	2	3	4	5	6	7	8	9
1		10	3			6	6		12
2			12	13	8	11	13	14	7
3				15	11	8	4		
4								9	
5						11			2
6							5	16	
7								14	
8									17
9									

Schritt	Kante	Gewicht	Union-Find-Struktur
1	(5,9)	2	
2	(1,3)	3	
3	(3,7)	4	
4	(6,7)	5	
5	(2,9)	7	
6	(4,8)	9	
7	(1,2)	10	
8	(2,4)	13	

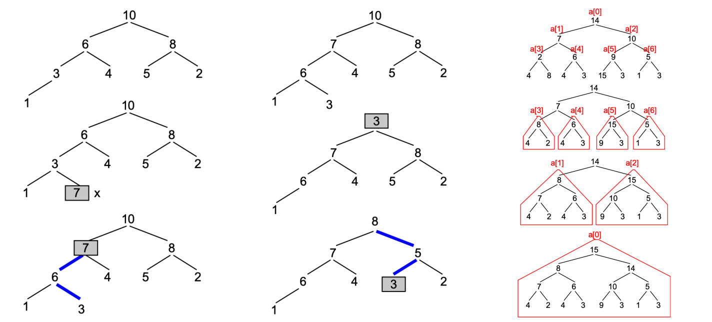


Wert der Wurzel = Höhe des Baums

Knoten	1	2	3	4	5	6	7	8	9
p	-3	5	1	1	1	1	1	4	5

Binäre Heaps:
Jeder Knoten muss >= seiner Kinder sein
Linksbündig einfügen

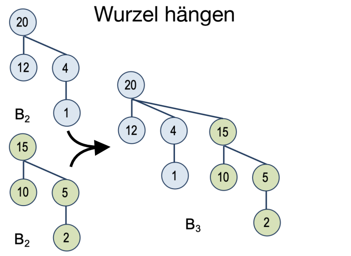
Insert and Upheap   Remove and Downheap   Create binary heap



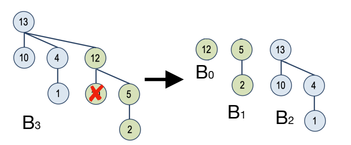
Binomiale Heaps:
Höhe = n   Knotenanzahl = 2^n Knoten pro ebene = $\binom{n}{k}$

⚠️ Jede Baumgröße darf nur ein mal existieren

Merge: Kleinere an Größere Wurzel hängen



Delete: Baum am Knoten in Teilbäume auflösen und ggf. mit anderen Bäumen mergen



O notationen aller ALDA

Algorithmus von Prim:
  Unsortiertes Feld T = O(|V|^2)
  Index-Heap T = O(|E| log|V|)

Algorithmus von Kruskal
  T = O(|E| log|V|)

Union-Find-Struktur:
  Union: T(n) = O(1)
  Find: T(n) = O(log n)

Tiefen/Breitensuche:
  T = O(|V| + |E|)

Ford Fulkerson:
  T = O(|E|2 log|V|)

Dijkstra Algorithmus:
  Dünn besetzt: |E| = O(|V|)
  Dicht besetzt: |E| = O(|V|^2)

Moore-Ford Algorithmus:
  T = O(|E|*|V|).

Floyd Algorithmus:
  T = O(n^3)

Upheap/Downheap:
  O(log n)

Heap Aufbau:
  T(n) = O(n)

Datenstruktur	deleteMax	insert	build	merge
Binäre Heaps	O(log n)	O(log n)	O(n)	O(n)
Binomiale Heaps	O(log n)	O(log n)	O(n)	O(log n)

Datenstruktur	deleteMax()	insert(x)	build(x[ ])	merge(prioList)
verkettete Liste	O(n)	O(1)	O(n)	O(1)
sortierte, verkettete Liste	O(1)	O(n)	O(n log n)	O(n)
Ausgeglichener Suchbaum	O(log n)	O(log n)	O(n log n)	O(n log n)

Datenstruktur	deleteMax()	insert(i, x)	build(i[ ], x[ ])	change(i, x)	remove(i)
Binäre Heaps (Java API PriorityQueue)	O(log n)	O(log n)	O(n)	O(n)	O(n)
Index-Heaps	O(log n)	O(log n)	O(n)	O(log n)	O(log n)

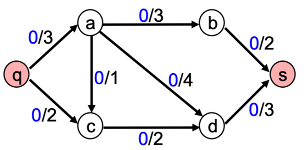
Datenstruktur	change(i, x)	remove(i)
verkettete Liste mit effizienter Suchstruktur für Nummerierung	O(log n)	O(log n)
sortierte, verkettete Liste mit effizienter Suchstruktur für Nummerierung	O(n)	O(log n)
Ausgeglichener Suchbaum mit effizienter Suchstruktur für Nummerierung	O(log n)	O(log n)

(Bellman-)Moore-Ford Algorithmus:
Funktioniert wie Dijkstras Algorithmus allerdings dürfen knoten mehrfach besucht werden. Dies muss passieren sobald sich ein Vorgänger verbessert hat.

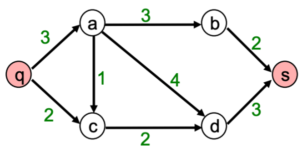
Negative Zyklen sind nicht erlaubt

Um negative Zyklen zu erhalten wird der Algorithmus n-1 mal gelaufen falls sich bei einem weiteren zustand noch werte ändern sind diese teil eines negativen Zyklus

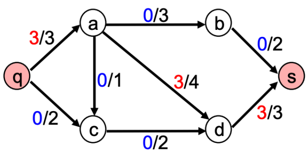
Flüsse (Ford-Fulkerson Algorithmus)
Ein Fluss hat eine quelle und eine Senke
Summe der ausgehenden Flüsse = Wert des Flusses



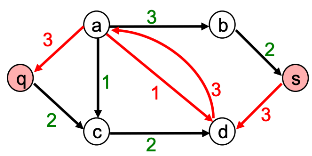
Graph G mit Nullfluss



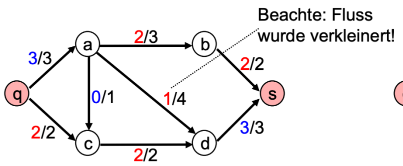
Residualgraph G_r mit Residualflüssen



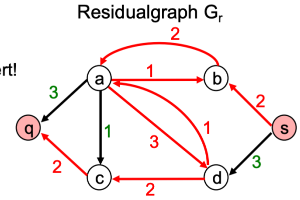
Graph G mit Flusswert 3



Residualgraph G_r

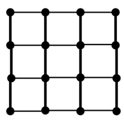


Graph G mit Flusswert 5

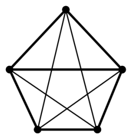


Residualgraph G_r

Dünn/Dicht besetzte Graphen



Dünn besetzt = nur wenige aller möglichen kanten existieren (bsp. Manhatten graph)
Es gilt: |E| ≈ 2*|V| = **O(|V|)**



Dicht besetzt = (fast) alle knoten sind miteinander verbunden. (bsp. ist ein vollständiger graph)
Es gilt: |E| = |V|*(|V|-1)/2 = **O(|V|^2)**

Vorteile von Rot-Schwarz gegenüber B-Baum:
Knoten von Rot-Schwarz-Bäumen sind einfach zu implementieren
nur Rotations- und Umfärbe- Operationen notwendig
Es gibt effiziente, iterative Top-down-Algorithmen.

Begründen Sie kurz, warum auch mit einem AVL-Baum das N-Max-Problem genauso effizient gelöst werden kann.
Bei einem AVL-Baum sind die Operation delMin, getMin und add ebenfalls effizient (d.h. in O(logN)) realisierbar.

Wieviel Knoten hat ein n*m-Manhattan-Graph ganz allgemein
n*m

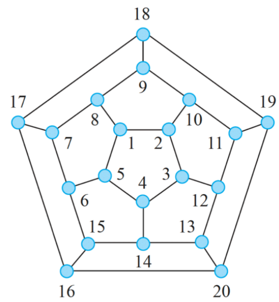
Wieviel Kanten hat ein n*m-Manhattan-Graph ganz allgemein (3 Punkte):
n*(m-1) + m*(n-1)

Welches Problem entsteht bei der rekursiven Tiefensuche bei einem sehr großen n*m-Manhattan- Graph? Zwei kurze Sätze genügen
Maximale Rekursiontiefe = n*m-1. Daher Stack-Overflow bei großem n, m.

Wieviel Kanten hat ein vollständiger Graph mit n Knoten allgemein?
n*(n-1)/2

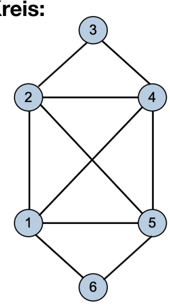
Durch welche Eigenschaft eines Binomialbaums ist sein Namensbestandteil „Binomial" gerechtfertigt?
Ein Binomialbaum der Höhe n hat in der Ebene k (n über k) viele Knoten.

Hammilton Kreis:



Ziel ist es einen Zyklus zu finden bei dem jeder **Knoten** genau ein mal besucht wird
NP-Vollständig
Brute force: T = O(n!)

Euler Kreis:



Ziel ist es einen Zyklus zu finden bei dem jede **Kante** genau ein mal besucht wird
P problem = in polinomieller zeit lösbar

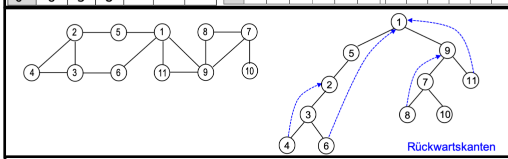
Geben Sie den gefundenen günstigsten Weg von 5 nach 1 an. (2 Punkte)

5 – 4 – 3 – 2 – 1

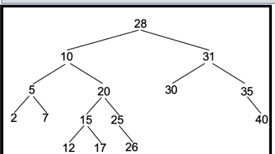
	1	2	3	4	5	6
1						
2	1					
3	3	1				
4	5	3	1			
5	10			2		1
6	8	5	3			

b	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	p[1]	p[2]	p[3]	p[4]	p[5]	p[6]
5	10	∞	∞	2	0	1	5	-	-	5	-	5
6	9	6	4				6	6	6			
4	7	5	3				4	4	4			
3	6	4					3	3				
2	5						2					
1												

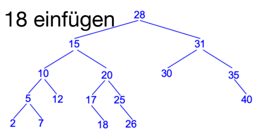
```
for (int k = 0; k < n; k++) {
    // Berechne D^k:
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (D[i][j] > D[i][k] + D[k][j]) {
                D[i][j] = D[i][k] + D[k][j];
                P[i][j] = P[k][j];
                if (i == j && D[i][j] < 0)
                    print("Negativer Zyklus)
            }
}
```



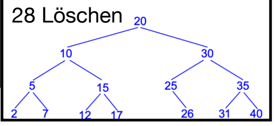

```
// Definition einer Prioritätsliste:
MaxPriorityQueue pq;

while (es kann eine Zahl x vom Datenstrom eingelesen werden) {
    // verarbeite x:
    if (pq.size() < N)
        pq.add(x);
    else if (x < pq.getMax()) {
        pq.delMax();
        pq.add(x);
    }
}

// Gib die N kleinsten Integerzahlen aus:
for (x : pq)
    print(x);
```

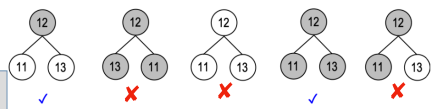


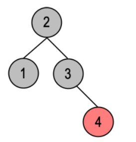
18 einfügen



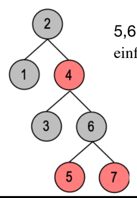
28 Löschen



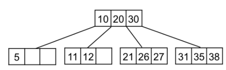




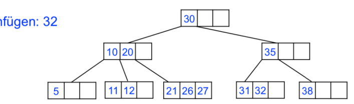
1, 2, 3, 4 einfügen
In leeren baum



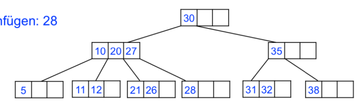
5,6,7
einfügen

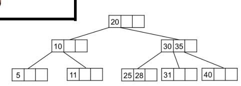


Einfügen: 32

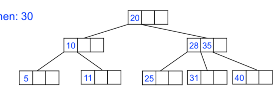


Einfügen: 28

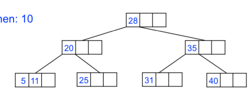




Löschen: 30



Löschen: 10



Löschen: 28