Algorithmen und Datenstrukturen

Master

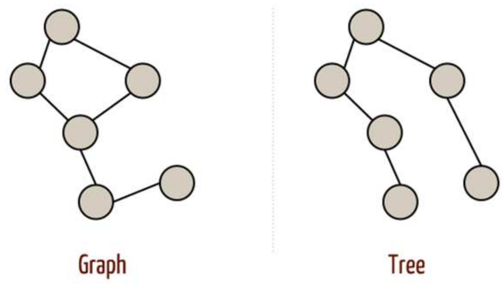
Graphen

Inhalt

- Was ist ein Graph
- Topologisches Sortieren
- Transitive Hülle
- Durchlaufen von Graphen
- Kürzeste Wege
- Minimale spannende Bäume
- Flüsse in Netzwerken

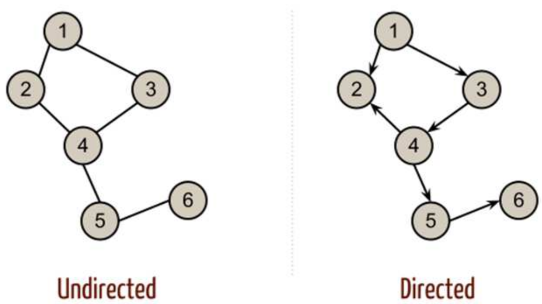
Darstellung von Graphen

- Graph hat Knoten und Kanten
  - Verbindungen zu Knoten auch mehrfach möglich
- Baum ist spezielle (einfache) Art von Graph



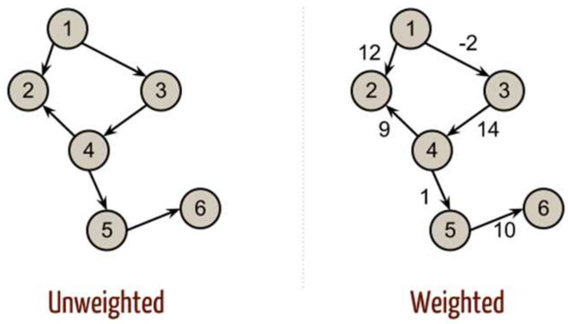
Gerichtete Graphen

- Verbindungskanten sind „Einbahnen“
  - Gerichtete Verbindung von 1->2, aber nicht zurück



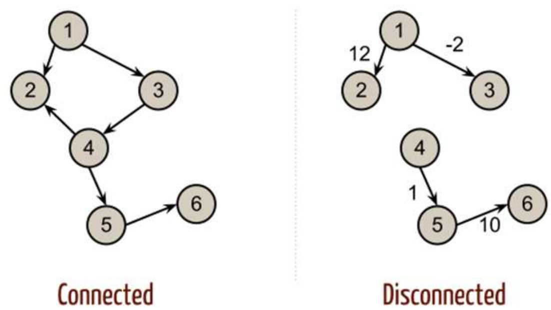
Gewichtete Graphen

- Verbindungen zwischen Knoten können Gewichtung haben
  - Z.B. Weglänge, Übertragungskosten, ...
  - Gewichtung kann auch negativ sein!



Getrennte Graphen

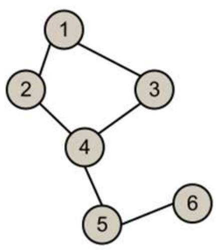
- Graphen können auch aus mehreren nicht verbundenen Teilen bestehen



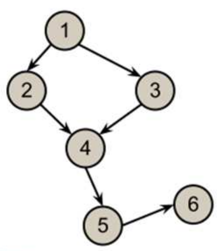
Repräsentation im Programm

- Adjazenz Matrix
  - NxN Matrix, A[i][j] = 1 bei Verbidnung, sonst 0
  - Vorzeichen für gerichtete Graphen
  - Gewichtungsfaktor in Matrix
  -
- Adjazenz Liste
  - Verkettete Liste, die für jeden Knoten alle Verbindungen hält.

Adjazenzmatrix

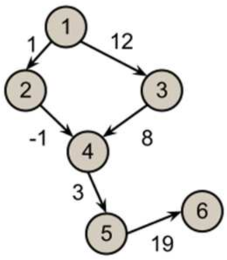


	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	1	1	0	1	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0



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

Adjazenzmatrix mit Wichtung

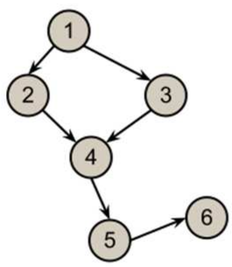


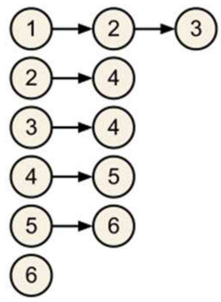
Weighted Directed Graph

	1	2	3	4	5	6
1	0	1	12	0	0	0
2	-1	0	0	-1	0	0
3	-12	0	0	8	0	0
4	0	1	-8	0	3	0
5	0	0	0	-3	0	19
6	0	0	0	0	-19	0

Adjacency Matrix

Adjazenzliste





Operationen bei Graphen

- Einfügen und Löschen von Kanten
- Einfügen und Löschen von Knoten
- Finden einer Verbindung zwischen i und j
- Finden der Nachbarn von i
- Feststellen, ob eine Verbindung zwischen i und j existiert

Komplexität

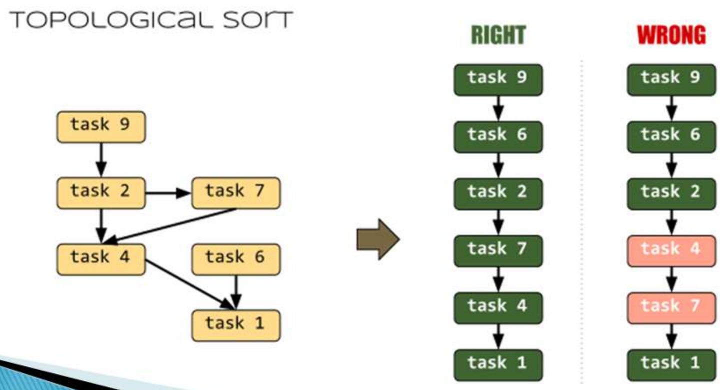
- Einfügen/Löschen von Kanten
  - O(1) O(log N)
- Feststellen ob Verbindung besteht
  - O(1) O(log N)
- Finden der Nachbarn von i
  - O(N) O(k) k..Länge der Liste
- Finden eines Weges von i nach j
  - $O(N^2)$ O(n+m)

Topologisches Sortieren

- Gerichteter Graph repräsentiert Abhängigkeiten (z.B. Projektplan)
- Sortierung nach Abhängigkeiten gewünscht
  - Im einfachen Fall verkettete Liste
  - Allgemein: Gerichteter Graph ohne zyklische Abhängigkeiten (Directed Acyclic Graph)
- Wenn Sortierung passt, können alle Aufgaben ohne Verzögerung der Reihe nach abgearbeitet werden.

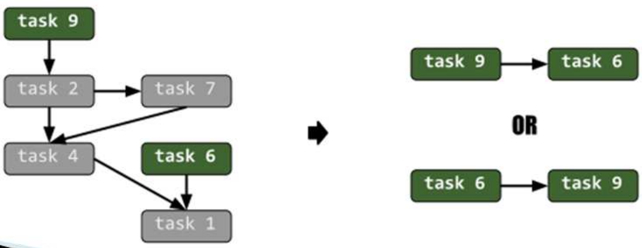
Topologisches Sortieren

- Sortierung soll gewährleisten, dass für jede Kante (i,j) i vor j priorisiert ist.



Topologisches Sortieren

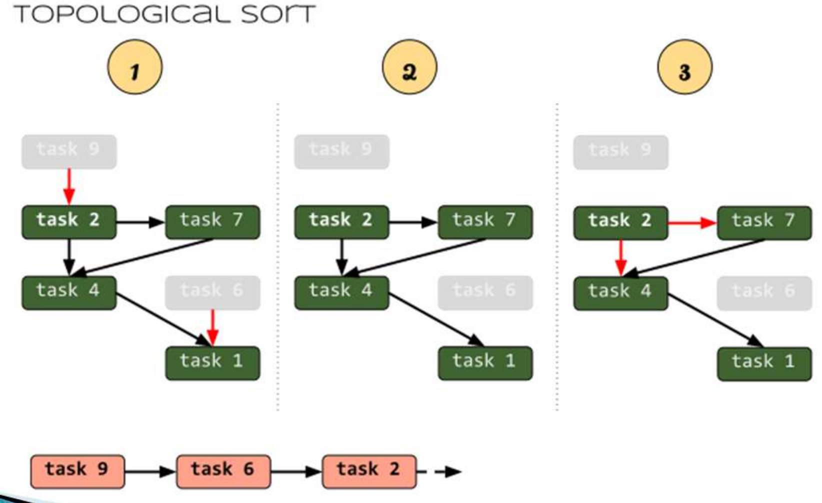
- Als Beginn müssen wir alle Knoten finden, die keinen Vorgänger (also keine Abhängigkeiten) haben
- Mehrere gültige Sortierungen möglich!



Topologisches Sortieren

- Generelles Vorgehen:
  - Suche alle Knoten ohne Vorgänger (Liste O)
  - Solange Einträge in O:
    - Nimm einen Eintrag n und schiebe ihn nach Liste S
    - Für jeden Nachfolger m von n
      - Entferne (n,m)
      - Wenn m keinen Vorgänger hat füge ihn in O dazu

Topologisches Sortieren



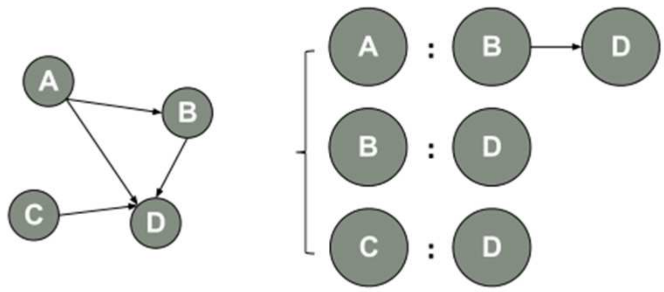
Topologisches Sortieren

- Überlegungen zur Performanz
- Wie finden wir Knoten ohne Vorgänger?

- Matrix: $O(N^2)$ Einträge
  - Zum Finden der Einträge muss ganze Matrix geprüft werden, und das N mal -> $O(N^3)$
- Liste: O(E) Einträge
  - Worst Case wieder $O(N^2)$

Topologisches Sortieren

- Adjazenzliste Beispiel



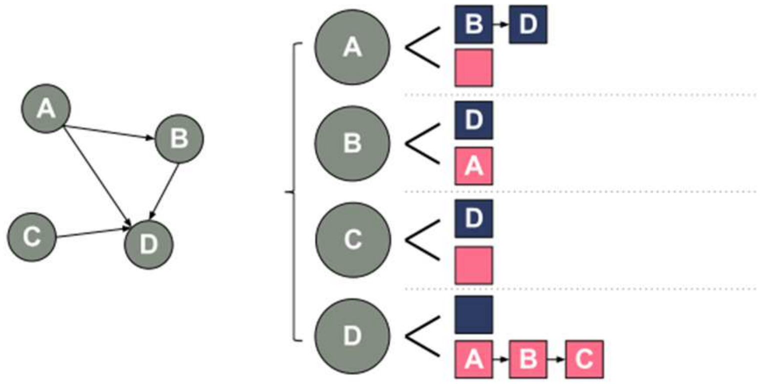
Topologisches Sortieren

- Überlegung zur Liste: Nimm einen zufälligen Knoten und gehe soweit zurück, bis kein Vorgänger mehr existiert.
  - Kann performant sein, oder auch wieder nicht

- Füge pro Knoten eine zweite Liste mit Vorgängern ein
  - Jetzt können alle Knoten ohne Vorgänger leicht gefunden werden
  - Bei Auswahl werden alle Einträge aus weiteren Listen entfernt.

Topologisches Sortieren

- Doppelte Adjazenzliste



Transitive Hülle

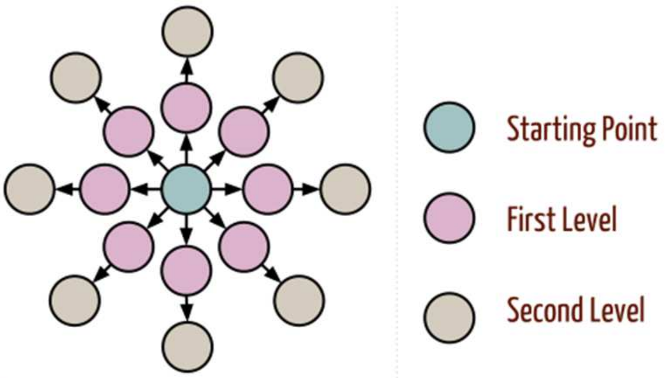
- Die transitive Hülle eines Graphen umfasst alle Knoten und Kanten, die wechselweise miteinander in Verbindung stehen
- Alle von einem Punkt aus erreichbaren Knoten liegen auf dieser Hülle.

Durchlaufen von Graphen

- Ein Baum hat eine Wurzel als Start
- Graphen haben gleichberechtigte Knoten
  - Erste Frage ist die Wahl eines Startpunkts
  - Weg (= Ergebnis) ist abhängig vom Startpunkt
- Breitensuche
  - Durchsuche zunächst alle Nachbarn, dann weiter
- Tiefensuche
  - Gehe entlang einer Kante so weit wie möglich, dann geh die nächste entlang

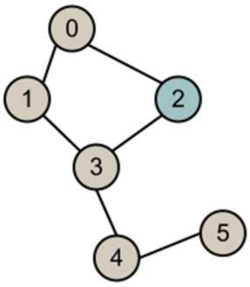
Breitensuche

- Wellenartiges Durchlaufen $O(N^2)$
- Eine „Ebene" nach der anderen



Breitensuche

- Lösung über Queue
  - Markiere zunächst alle Knoten als nicht besucht
  - Nimm einen nach dem anderen aus der Queue

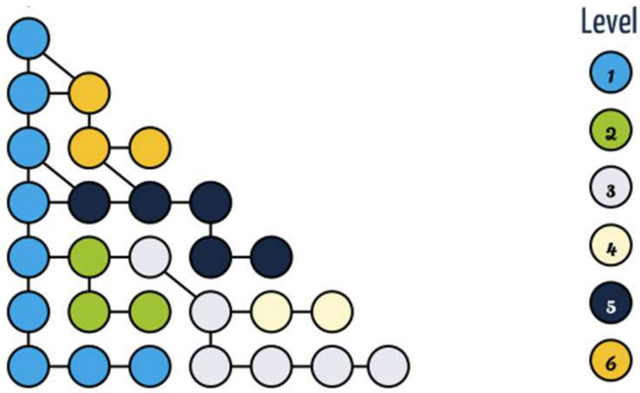



```
1. q = {}
2. q = {2}
3. q = {0, 3}
4. q = {1}
5. q = {4}
6. q = {5}
```

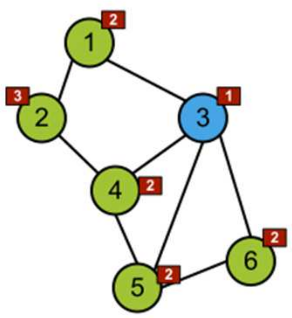
Tiefensuche

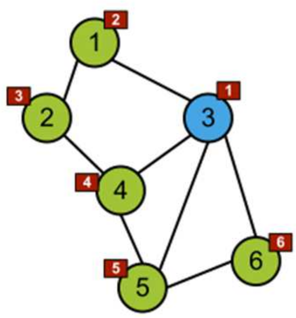
- Bereits bekannt von den Bäumen (pre-order, in-order, post-order)
- Gehe entlang der Kanten bis zum Blatt, dann retour und nächste Kante
- z.B. Finde Pfad von A nach B
  - Wähle Kante, die noch nicht besucht wurde
  - Gehe zum nächsten Nachfolger, der noch nicht besucht wurde
  - Wenn alle Nacholger besucht wurden, gehe zum Parent zurück

Tiefensuche



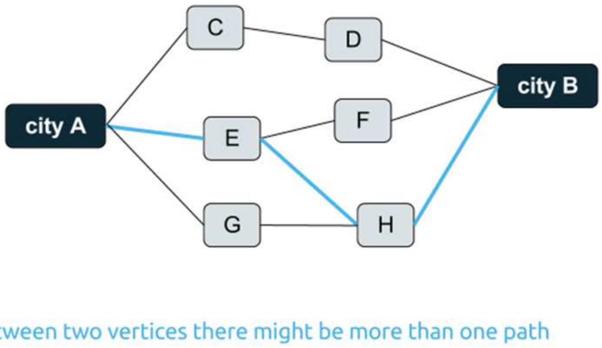
Breitensuche vs. Tiefensuche





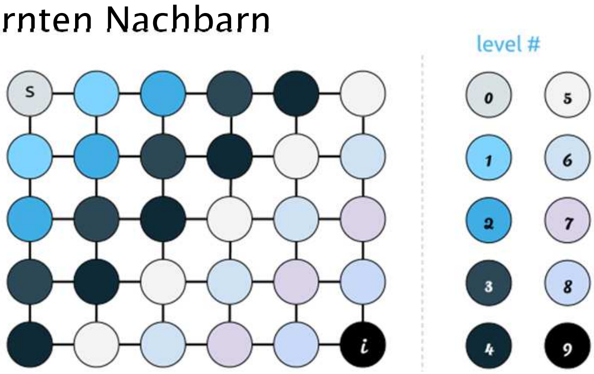
Kürzester Weg von A nach B

- In ungewichteten Graphen zählt nur die Anzahl der Kanten

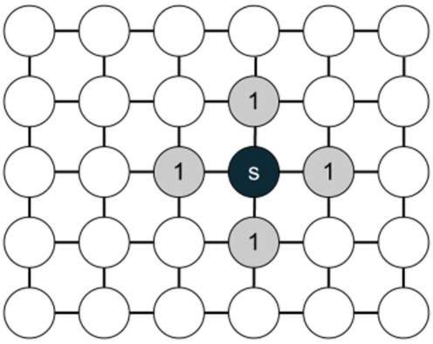


Kürzester Weg – Breitensuche

- Breitensuche liefert den kürzesteh Weg
- Breitensuche liefert auch alle gleich weit entfernten Nachbarn



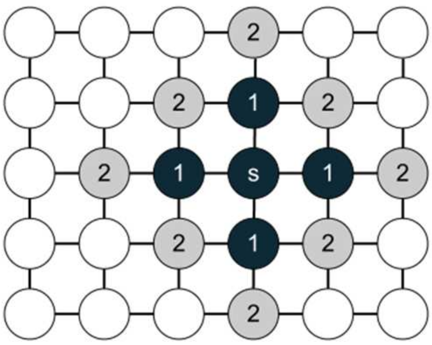
Breitensuche



Colors

The visited vertices are black, while the vertices into the queue are gray. Those of the vertices that aren't visited yet are white!

Breitensuche

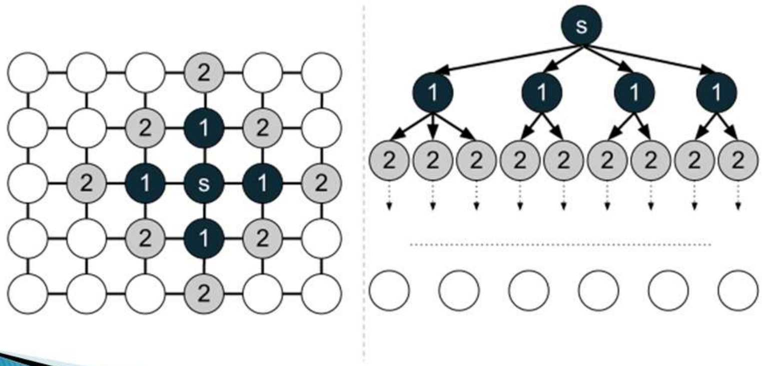


Paths

The distances between the starting vertex and all the other vertices are the shortest possible!

Breitensuche

- Graph wird zum virtuellen Baum
- Die Tiefe entspricht der Entfernung zu S



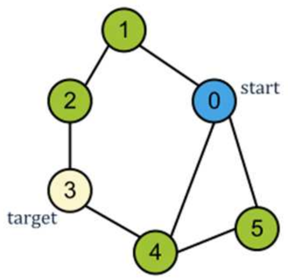
Suche nach dem kürzesten Weg

- In gewichteten Graphen haben Kanten auch Wert (z.B. Straßenkarte, Entfernung)
  x = F(a,b)

- Nicht jeder Weg ist gleich „gut“

- In gerichteten Graphen kommt noch dazu
  F(a,b) != F(b,a)

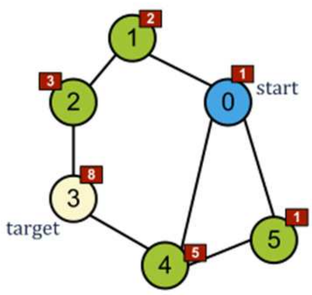
Beispiel ungerichteter Graph




```
Adjacency Matrix
A[0]: [0, 1, 0, 0, 1, 1]
A[1]: [1, 0, 1, 0, 0, 0]
A[2]: [0, 1, 0, 1, 0, 0]
A[3]: [0, 0, 1, 0, 1, 0]
A[4]: [1, 0, 0, 1, 0, 1]
A[5]: [1, 0, 0, 0, 1, 0]
```

Kürzester Weg erfüllt Kriterien

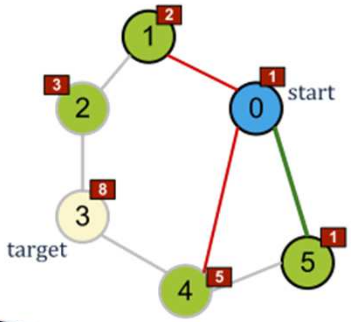
- Wähle bei Tiefensuche nicht einen beliebigen Nachfolger, sondern den “besten“
  - Minimum oder Maximum des Kantengewichts




```
Adjacency Matrix
A[0]: [0, 2, 0, 0, 5, 1]
A[1]: [1, 0, 3, 0, 0, 0]
A[2]: [0, 2, 0, 8, 0, 0]
A[3]: [0, 0, 3, 0, 5, 0]
A[4]: [1, 0, 0, 8, 0, 1]
A[5]: [1, 0, 0, 0, 5, 0]
```

Beispiel: von 1 nach 3

- Kürzester Weg 1 -> 5 (… 4 -> 3)
- Können wir sicher sein, dass das der beste Weg ist?



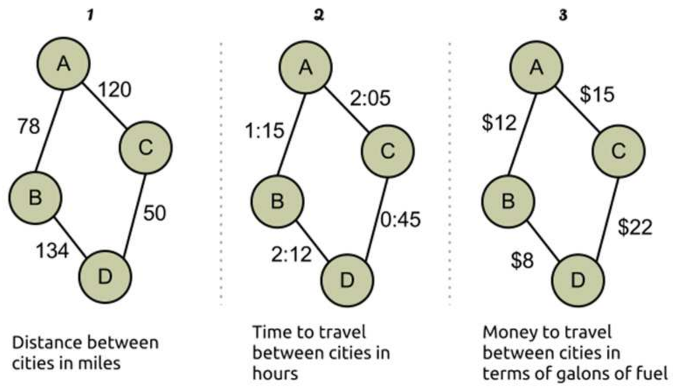

```
Adjacency Matrix
A[0]: [0, 2, 0, 0, 5, 1]
A[1]: [1, 0, 3, 0, 0, 0]
A[2]: [0, 2, 0, 8, 0, 0]
A[3]: [0, 0, 3, 0, 5, 0]
A[4]: [1, 0, 0, 8, 0, 1]
A[5]: [1, 0, 0, 0, 5, 0]
```

Kürzester Weg

- Tiefensuche mit Gewichtung statt einfach nächste Kante zu nehmen
- Wenn wir Weg von A nach B finden, dann ist das der Beste bisher
- Wie können wir sicher sein, dass es der Beste überhaupt ist?
    - Sortiere Nachfolger der Größe nach

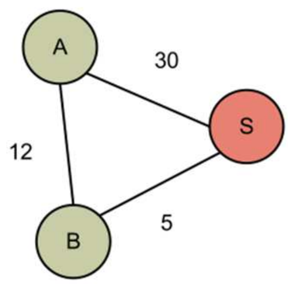
Dijkstra Algorithmus

- Gewichtung kann unterschiedlich sein



Dijkstra Algorithmus (1959)

- Verwendet Priority Queue (Heap)
- Merke mir bisherigen Weg vom Start
- Beispiel: [S,B,A] kürzer als [S,A]



Dijkstra Algorithmus

- Füge Entfernung von S nach A und B in Proirity Queue ein
- Hole Minimum (B)
- Dann teste alle Nachbarn von S auf Nachbarschaft zu B
  - Prüfe, ob B oder andere näher sind
- Animation: http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html

Bellman-Ford Algorithmus 1958

- Dijkstra funktioniert nicht bei negativen Gewichtungen!
- Benutze Breitensuche:
  - Prinzipiell wie Dijkstra, aber
  - Es werden alle Kanten berechnet
- Praktische Anwendung: Routing Information Protocol (RIP)

Minimale spannende Bäume

- Ein minimaler spannender Baum eines Graphen ist jener Baum, der alle Knoten mit der kürzest möglichen Gesamtlänge von Kanten verbindet
  - Finde kürzesten Weg, um eine Reihe von Orten zu besuchen
  - Finde kürzeste Möglichkeit, bei gegebenen Kabeltrassen ein Netzwerkkabel in alle Büros zu legen

Minimale spannende Bäume

- Algorithmus:
  - Initialsierung: alle Knoten sind unbesucht
  - Wähle einen Startknoten und markiere ihn als besucht.
  - Suche jenen Knoten, der dem besuchten am Nächsten ist und markiere ihn als besucht. Gehe weiter.
  - Solange nicht alle Knoten besucht sind wiederhole.
- Animation: http://optlab-server.sce.carleton.ca/POAnimations2007/MinSpanTree.html

Kruskal Algorithmus 1956

- Sortiere Kanten nach Gewicht
- Schleife:
  - Nimm kleinste Kante + Knoten
  - Wenn ein Knoten neu, übernimm Kante in Lösung
  - Wenn beide Knoten bereits in der Lösung, verwirf Kante
  - Wiederhole, bis alle Knoten in der Lösung sind

- Animation: http://students.ceid.upatras.gr/~papagel/project/kruskal.htm

Jarnik Prim Algorithmus 1930

- Wähle beliebigen Anfangsknoten
- Schleife:
  - Wähle aus allen an bereits ausgewählte Knoten anschliessenden Kanten die kürzeste
  - Wenn Kante neuen Knoten dazubringt füge diesen Knoten der Lösung hinzu

- Animation: http://students.ceid.upatras.gr/~papagel/project/prim.htm

Flüsse in Netzwerken

- Optimiere Verkehrsfluss
- Wieviel Wasser kann ich durch ein gegebenes Kanalnetz transportieren?

- Netzwerk besteht aus einem gerichteten Graphen
- Graph enthält nun Kanten, die Kapazität beschreiben
- Fluss muss erhalten bleiben

Flüsse in Netzwerken

- An jedem Knoten gilt die Erhaltung des Flusses
  - Was reinfließt muss auch wieder raus
  - Ausnahme: Quelle und Senke

$$\sum\nolimits_{(v,v')\in E} f((v,v')) = 0 \quad \forall\, v \in V - \{q,s\}$$

- Jede Kante e kann nur soviel Fluss f(e) zulassen wie ihre Kapazität c(e) erlaubt

$$f(e) \le c(e) \quad \forall\, e \in E$$

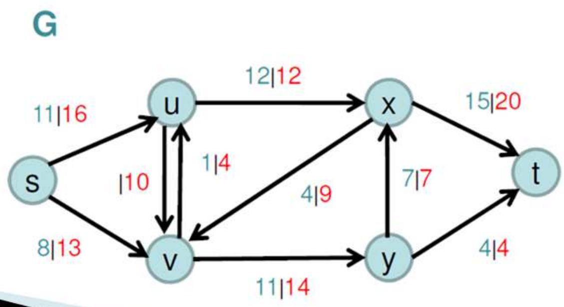
Flüsse in Netzwerken

- Schnitt
  - Schneidet man den Graph zwischen q und s, so gibt die minimale Summe der Kapazitäten der Kanten den maximalen Fluss an
  - (min-cut-max-flow Theorem)
- Residualnetz
  - Ein Residualnetzwerk zeigt neben dem Fluss auch noch zwei weitere Größen:
    - Die nicht ausgelastete Kapazität auf Vorwärtskanten (wieviel darf der Fluss noch erhöht werden)
    - Die Rückkante zeigt um wieviel der Fluss verringert werden darf

Ford-Fulkerson Algorithmus

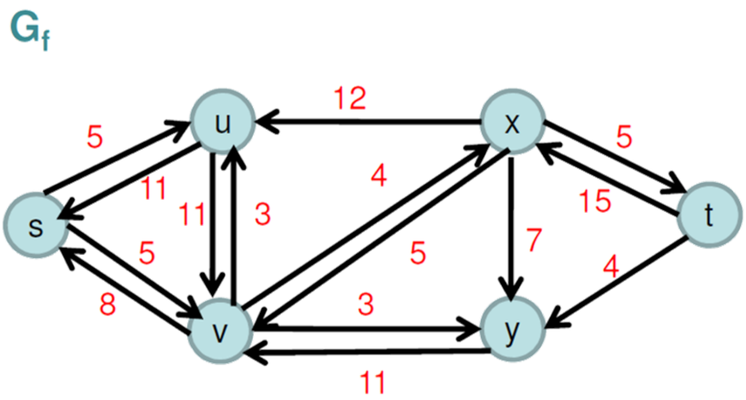
- Suche maximalen Fluss durch das Netzwerk
- Iterativer Algorithmus

Flussnetzwerk:



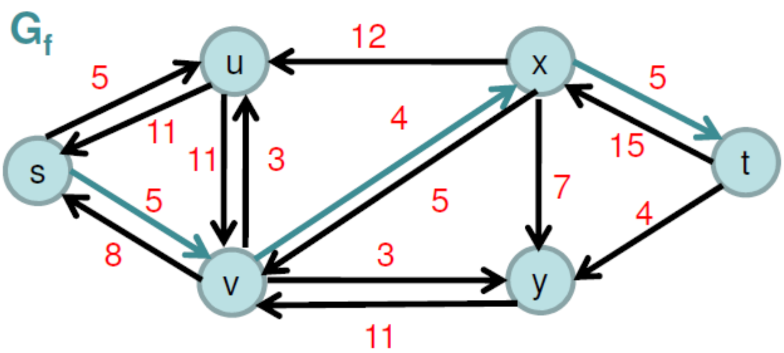
Ford-Fulkerson Algorithmus

- Residualnetzwerk



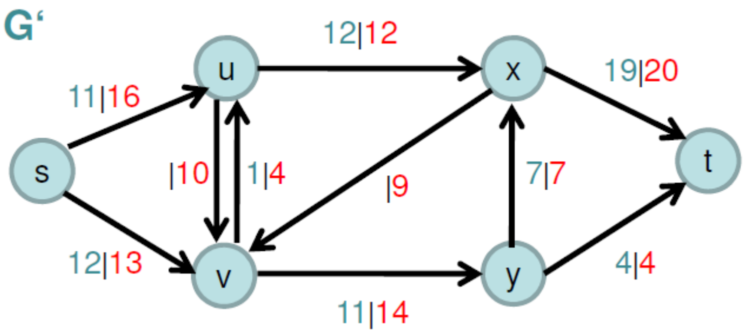
Ford-Fulkerson Algorithmus

- Residualnetzwerk mit augmentiertem Pfad



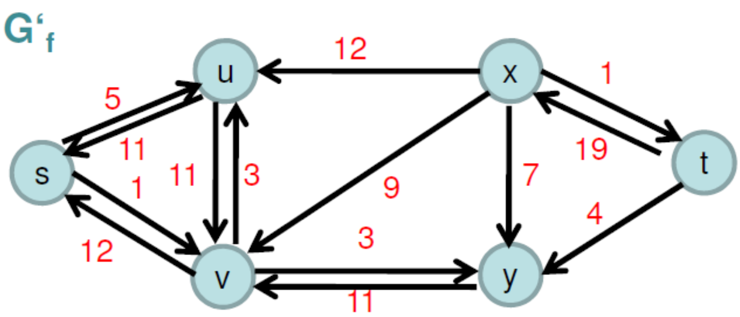
Ford-Fulkerson Algorithmus

- Maximal augmentiertes Netzwerk



Ford-Fulkerson Algorithmus

- Neues Residualnetzwerk



Ford-Fulkerson Algorithmus

- Initialisiere alle Flüsse mit 0
- Schleife
  - Suche möglichen augmentierbaren Pfad
  - Füge maximal möglichen Fluss dazu
    - Reduziere möglicherweise bereits bestehenden Fluss
  - Berechne neues Residualnetz
- Fertig wenn kein weiterer augmentierbarer Pfad von s zu t existiert

Edmonds-Karp Algorithmus

- Problem von Ford-Fulkerson ist zu grosse Auswahl an möglichen augmentierbaren Pfaden
- Daher Heuristiken von Edmonds und Karp:
  - Wähle den augmentierenden Pfad mit dem größten Wert.
  - Wähle den kürzesten augmentierenden Pfad.

Dinitz Algorithmus

- Arbeitet mit gesättigten Kanten
  - Muss nicht unbedingt der Maximalfluss sein
- Der Fluss von s nach t ist blockierend wenn jeder Pfad von s nach t eine gesättigte Kante enthält.

Übung 6

- Suchen Sie auf einer Landkarte 8 Orte aus
- Bestimmen Sie die möglichen Verbindungen (gewichteter Graph)
  - Autobahn 60 FZ/min, Bundesstrasse 40 FZ/min, Nebenstrasse 20 Fz/min, Ort 10 Fz/min
- Schreiben Sie ein Programm, das den kürzesten Weg zwischen zwei dieser Orte berechnet
- Schreiben Sie ein Programm, das den maximalen Verkehrsfluss zwischen 2 Orten bestimmt