DHBW Lörrach

Duale Hochschule Baden-Württemberg

Algorithmen und Komplexität
TIF 21A/B
Dr. Bruno Becker

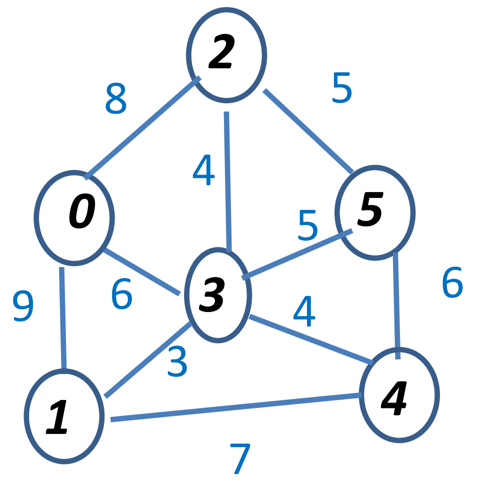
9. Optimierungsprobleme für Graphen
9.1. Minimal Spannende Bäume

Minimal Spannende Bäume

- **Gewichtete Graphen und Bäume**
- Minimale Spannender Baum (MST)
- Algorithmen zum Finden von MST
- Korrektheit der Algorithmen

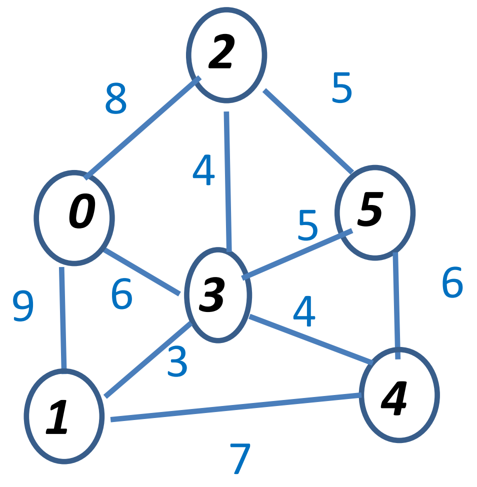
Beispielanwendung

- Wasserleitungsnetz:
  - Leitungen zwischen den Knoten (Orten) mit Kosten
  - Gesucht: Kostenminimimales Netz, das alle Knoten verbindet



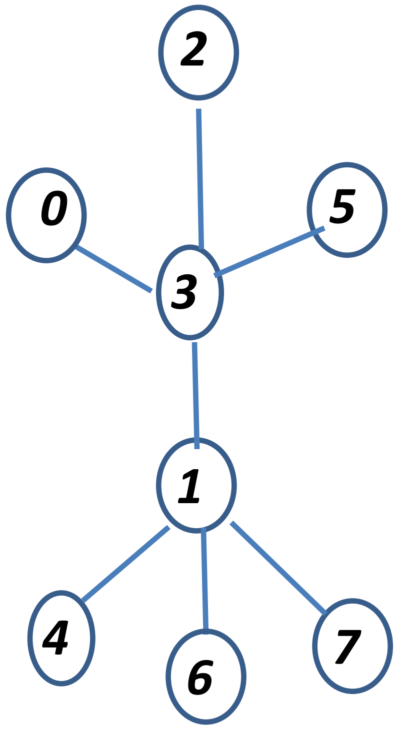
Gewichtete Graphen

- **(Kanten-)Gewichteter Graph**
  - Jeder Kante ist eine reelle Zahl zugeordnet, das **Gewicht** *(Kosten)*
  - Graph $G = (V,E,w)$ mit **Gewichtsfunktion** $w: E \to \mathbb{R}$



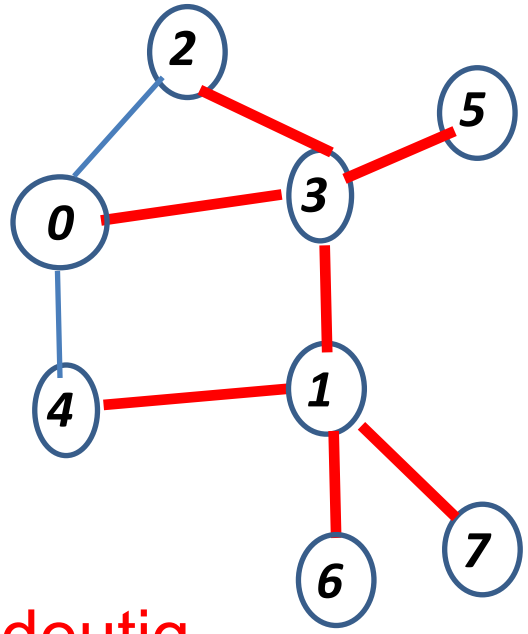
Kantenanzahl von Bäumen

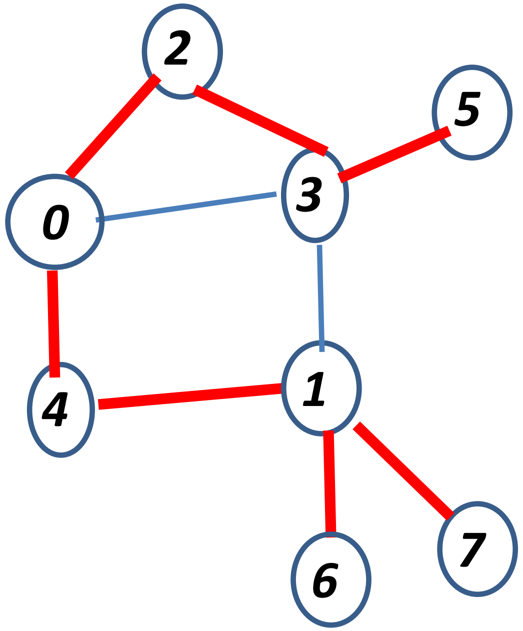
- Bäume mit *n* Knoten haben genau *n-1* Kanten
  - Zusammenhängende Graphen haben mindestens *n-1* Kanten
  - Beweis: Vollständige Induktion



Spannende Bäume von Graphen

- Gegeben sei ein zusammenhängender Graph *G.* Ein **spannender Baum** *(spanning tree) von G* ist ein Teilgraph, der
  - alle Knoten von *G* enthält
  - ein Baum ist, d.h. zyklenfrei ist





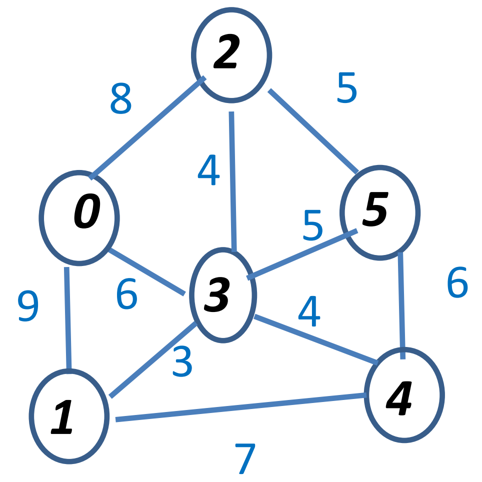
- Nicht eindeutig

Minimal Spannende Bäume

- Gewichtete Graphen und Bäume
- **Minimale Spannender Baum (MST)**
- Algorithmen zum Finden von MST
- Korrektheit der Algorithmen

Minimal Spannender Baum

Ein **minimal spannender Baum** *(minimal spanning tree, MST)* eines gewichteten Graphen *G* ist ein spannender Baum mit minimaler Summe der Kantengewichte



Anwendungen für minimal spannende Bäume (MST)

- Straßennetze mit minimaler Gesamtlänge zwischen Orten
- Ver- und Entsorgungsnetze, das Haushalte anschließt
- Routing in Datennetzwerken
- Bildverarbeitung, Gesichtserkennung
- Näherungslösungen für algorithmisch schwierige Probleme, z.B. Travelling Salesman Problem
- …

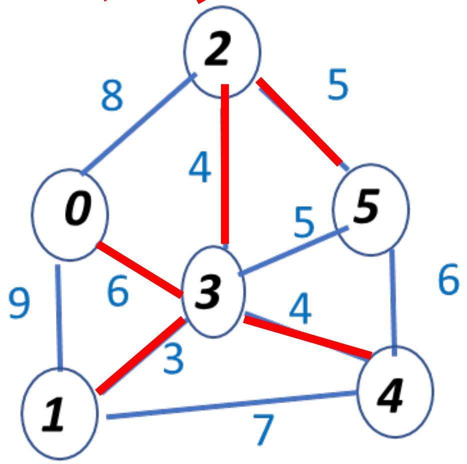
Minimal Spannende Bäume

- Gewichtete Graphen und Bäume
- Minimale Spannender Baum (MST)
- **Algorithmen zum Finden von MST**
- Korrektheit der Algorithmen

Algorithmus von Kruskal

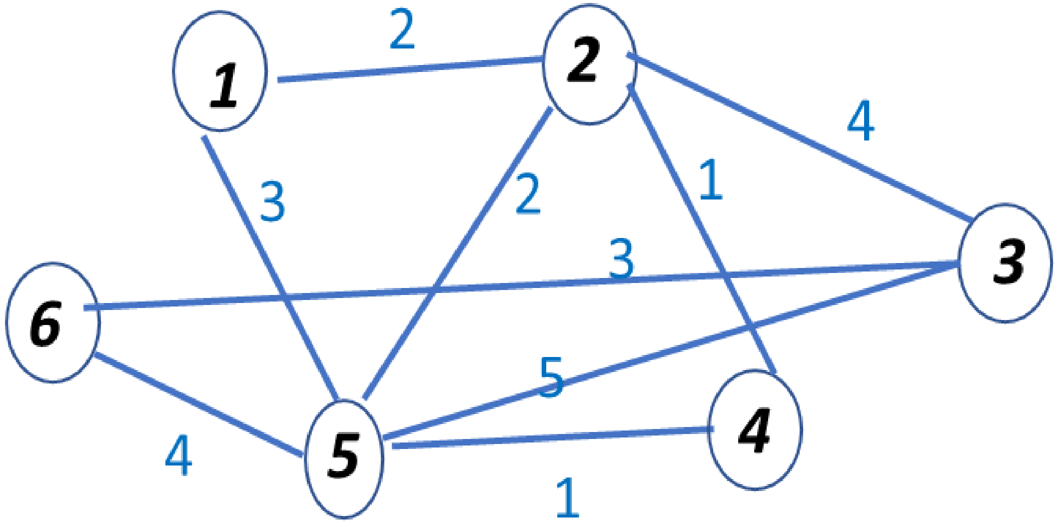
- **Algorithmus für Finden von MST (1956)**
  - Starte mit n Bäumen mit je 1 Knoten
  - Betrachte die Kanten in der Reihenfolge aufsteigender Gewichte
  - Füge jeweils die nächste noch nicht betrachtete Kante dazu, außer wenn hierdurch Zyklus entstehen würde (bis n-1 Kanten)

Sortiere Kanten: 1-3; 2-3;3-4; 2-5; ~~3-5~~; ~~4-5~~; 0-3; 1-4; 0-2; 0-1;



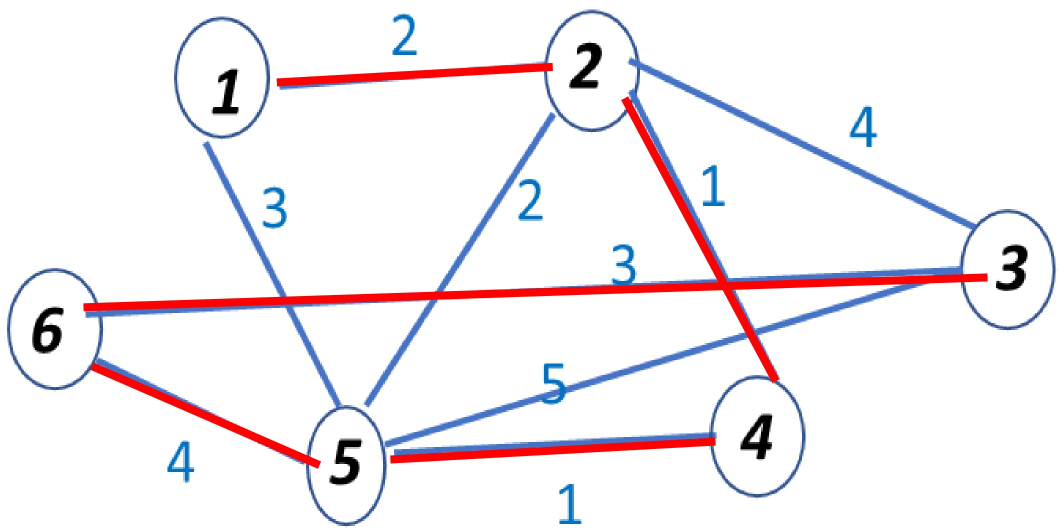
Algorithmus von Kruskal

- Beispiel



Algorithmus von Kruskal

- Beispiel



Sortiere Kanten 2-4; 4-5; ~~2-5~~; 1-2; ~~1-5~~;3-6; 5-6; 2-3; 3-5

Algorithmus von Kruskal

- Union Find (Selbststudium z.B. *Sedgewick*)– unterstützt folgende Operationen:
  - *Find (v) :* Name des Baumes, zu dem Knoten *v* gehört
  - *Union (v,w):* vereinigt Bäume *v* und *w* zu einem Baum mit Namen *v*
  - *Make-set(v):* Erzeugt Baum mit einzigem Knoten *(v)*

```
{
  E` = {};
  sortiere E nach aufsteigender Länge;
   for all v ϵ V { Make-set(v) }; // Erzeuge lauter Bäume mit einem Knoten
   for all (v,w) ϵ E
   { if (Find(v) != Find(w))  // wähle Kante (v,w)
        { Union (Find(v),Find(w); // Die beiden Bäume werden durch (v,w) vereinigt
          E`= E`  + (v,w)
         };
   };
};
```

Algorithmus von Kruskal : Aufwandsanalyse

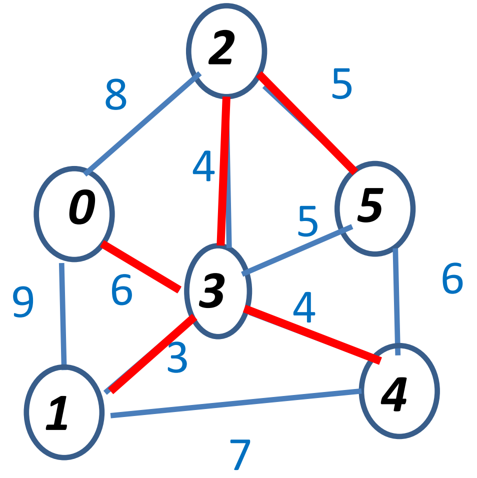
- Union Find-Operationen:
  - *Find (v), Union (v,w): O(log (‖ V ‖), Make-set(v):* O(1).

- *for*-Schleife wird für alle Kanten durchlaufen:

➔ **Gesamtaufwand für *G(V,E)* = *O ((‖ E ‖ log (‖ V ‖)***

Algorithmus von Prim

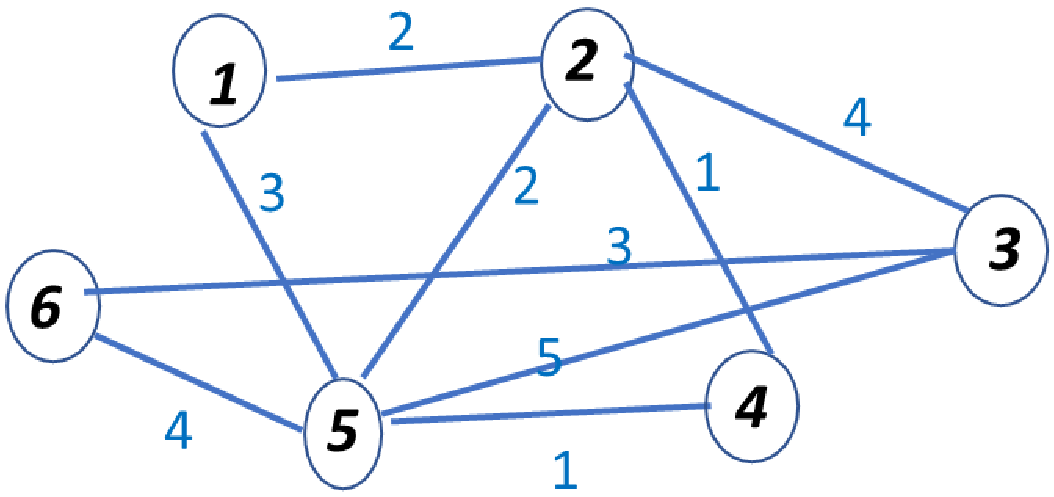
- **Algorithmus für Finden von MST (1959)**
  - Starte mit einem Knoten und baue schrittweise einen Baum *T* auf
  - Füge in jedem Schritt die günstigste Kante dazu, die genau einen Endpunkt in *T* hat (-> Minimum-Priority Queue)



Beispiel: Starte mit Knoten 1

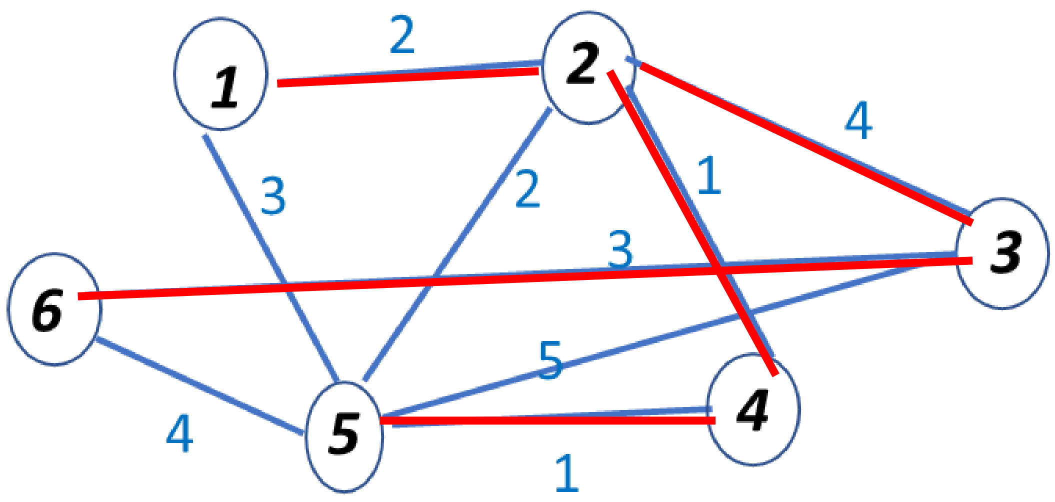
Algorithmus von Prim

- Beispiel (wie vorher) Startknoten 1



Algorithmus von Prim

- Beispiel (wie vorher) Startknoten 1



Algorithmus von Prim : Aufwandsanalyse

- Von der Datenstruktur her etwas einfacher als Kruskal, weil nur ein Baum wächst, statt viele
  - Priority Queue mit Fibonacci-Heap (s. Ottmann/Widmayer 6.1.5) -> O(log N)

➔**Gesamtaufwand für** ***G(V,E)*** **=** ***O*** $((\| E \| + \| V \| \log (\| V \|))$

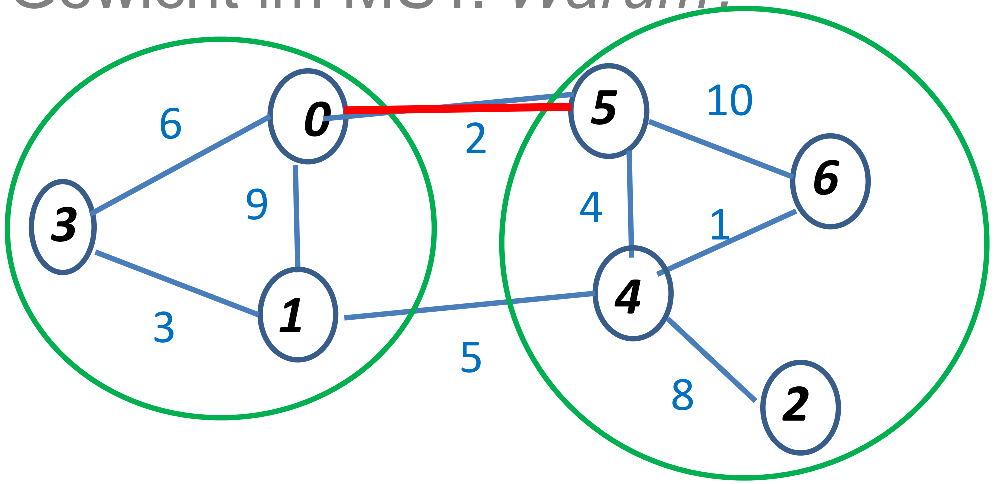
D.h. für $\| E \| > \| V \|$ etwas schneller als Kruskal

Minimal Spannende Bäume

- Gewichtete Graphen und Bäume
- Minimale Spannender Baum (MST)
- Algorithmen zum Finden von MST
- **Korrektheit der Algorithmen**

Schnitte von Graphen

- Ein **Schnitt (*cut*)** in einem Graphen *G(V,E)* ist eine Partition (*Zerlegung)* der Knotenmenge *V* in zwei nichtleere Mengen *S* und *S`*
- Eine **Schnittkante (*crossing edge)*** verbindet einen Knoten in der einen Schnittmenge mit einem Knoten der anderen Schnittmenge
- **Schnitteigenschaft:** Für jeden gegebenen Schnitt liegt die Schnittkante mit dem kleinsten Gewicht im MST. *Warum?*



Korrektheit des Algorithmus von Prim

- Algorithmus *terminiert*
  - Wählt in jedem Schritt eine Kante
  - Wahl wird nichtmehr rückgängig gemacht
    - Kennzeichen eine *greedy (gierigen Algorithmus):* Entscheidungen werden auf der Grundlage der vorhandenen Informationen getroffen und bringen den Rechenprozess näher an Lösung ran (im Gegensatz z.B. *Backtracking*)
- Erzeugt spannenden Baum zum Inputgraphen
- In jedem Teilschritt gilt:
  - Knoten des bisher gefundenen Teilbaums definieren einen Schnitt
  - Neu hinzugenommene Kante ist minimale Schnittkante

Zusammenfassung

- Gewichtete Graphen und Bäume
- Minimale Spannender Baum (MST)
- Zwei Algorithmen zum Finden von MST (Kruskal und Prim)
  - Kruskal baut aus N Einzel-Bäumen schrittweise MST auf durch Wahl der minimalen Kante
  - Prim startet bei einem Knoten und baut schrittweise MST mit minimal angrenzender Kante auf
- Korrektheit der Algorithmen
  - Definition Schnitt
  - Durch Wahl minimaler Kante werden falsche Entscheidungen vermieden
  - Greedy-Algorithmus