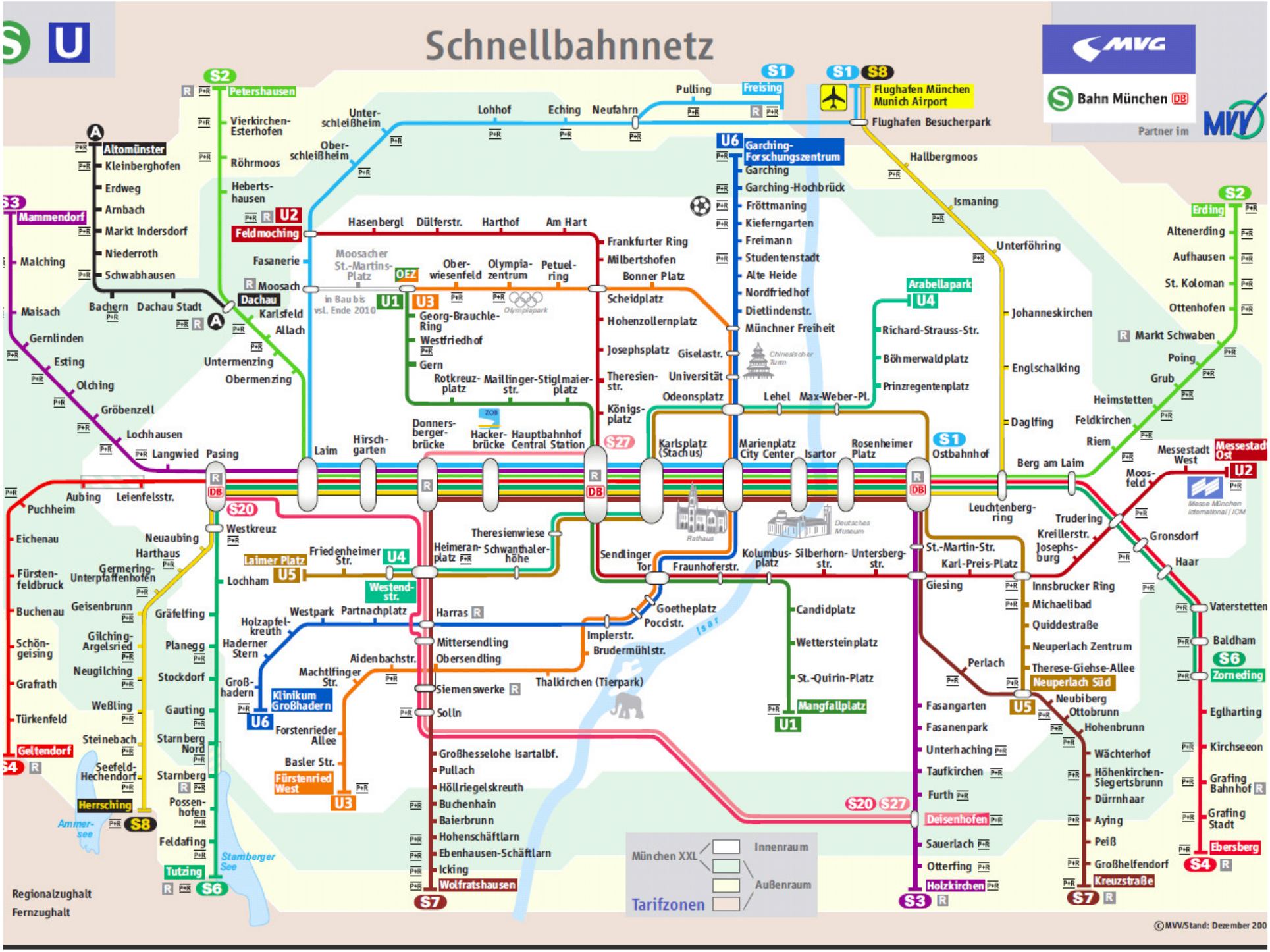
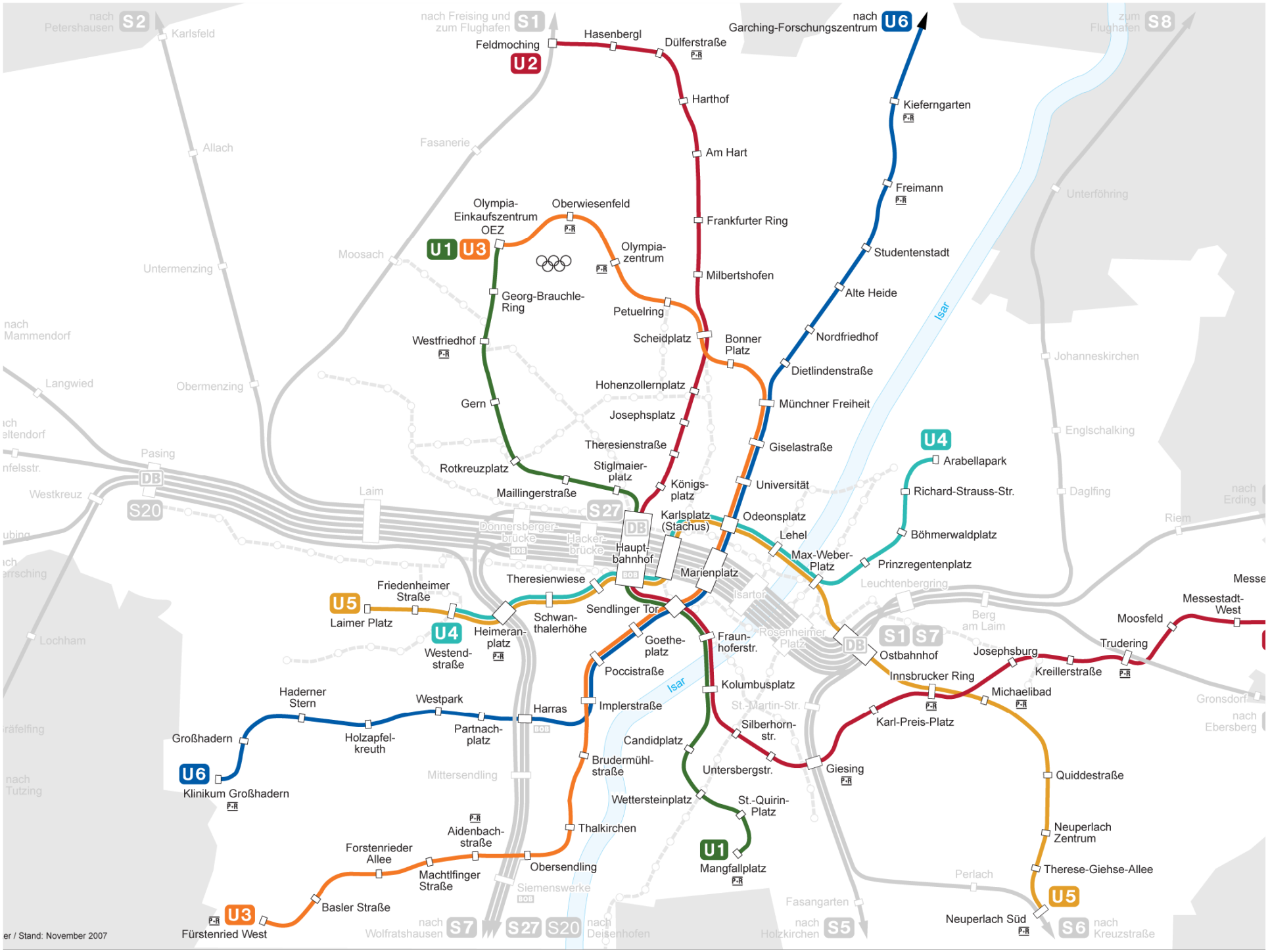
Graphalgorithmen II

Korbinian Breu


Schnellbahnnetz
MVG
Bahn München DB
Partner im MVV
S1 Freising
S1 S8 Flughafen München Munich Airport
Flughafen Besucherpark
Pulling
Lohhof
Eching
Neufahrn
Unterschleißheim
Oberschleißheim
S2 Petershausen
Vierkirchen-Esterhofen
Röhrmoos
Hebertshausen
U2 Feldmoching
Altomünster
Kleinberghofen
Erdweg
Arnbach
Markt Indersdorf
Niederroth
Schwabhausen
Bachern
Dachau Stadt
Dachau
Karlsfeld
Allach
Untermenzing
Obermenzing
S3 Mammendorf
Malching
Maisach
Gernlinden
Esting
Olching
Gröbenzell
Lochhausen
Langwied
Pasing
Aubing
Leienfelsstr.
Puchheim
Eichenau
Fürstenfeldbruck
Buchenau
Schöngeising
Grafrath
Türkenfeld
Geltendorf
S4
Germering-Unterpfaffenhofen
Geisenbrunn
Gilching-Argelsried
Neugilching
Weßling
Steinebach
Seefeld-Hechendorf
Herrsching
Ammersee
S8
Neuaubing
Harthaus
Gräfelfing
Planegg
Stockdorf
Gauting
Starnberg Nord
Starnberg
Possenhofen
Feldafing
Tutzing
Stamberger See
S6
Moosacher St.-Martins-Platz
OEZ
in Bau bis vsl. Ende 2010
U1
Hasenbergl
Dülferstr.
Harthof
Am Hart
Frankfurter Ring
Milbertshofen
Bonner Platz
Fasanerie
Moosach
Ober-wiesenfeld
Olympia-zentrum
Petuel-ring
U3
Georg-Brauchle-Ring
Westfriedhof
Gern
Rotkreuz-platz
Maillinger-str.
Stiglmaier-platz
Scheidplatz
Hohenzollernplatz
Josephsplatz
Theresien-str.
Königs-platz
Giselastr.
Universität
Odeonsplatz
Lehel
Max-Weber-Pl.
Laim
Hirsch-garten
Donners-berger-brücke
Hacker-brücke
Hauptbahnhof Central Station
S27
Karlsplatz (Stachus)
Marienplatz City Center
Isartor
Rosenheimer Platz
S1 Ostbahnhof
Rathaus
Deutsches Museum
Chinesischer Turm
Olympiapark
ZOO
U6 Garching-Forschungszentrum
Garching
Garching-Hochbrück
Fröttmaning
Kieferngarten
Freimann
Studentenstadt
Alte Heide
Nordfriedhof
Dietlindenstr.
Münchner Freiheit
Arabellapark
U4
Richard-Strauss-Str.
Böhmerwaldplatz
Prinzregentenplatz
Hallbergmoos
Ismaning
Unterföhring
Johanneskirchen
Englschalking
Daglfing
Berg am Laim
Leuchtenbergring
Trudering
Kreillerstr.
Josephsburg
Riem
Messestadt West
Messestadt Ost
U2
Messe München International / ICM
Moosfeld
Gronsdorf
Haar
S2 Erding
Altenerding
Aufhausen
St. Koloman
Ottenhofen
Markt Schwaben
Poing
Grub
Heimstetten
Feldkirchen
Vaterstetten
Baldham
S6 Zorneding
Eglharting
Kirchseeon
Grafing Bahnhof
Grafing Stadt
Ebersberg
S4
S20
Westkreuz
Laimer Platz
Friedenheimer Str.
U4
U5
Lochham
Westendstr.
Westpark
Partnachplatz
Holzapfelkreuth
Haderner Stern
Großhadern
Klinikum Großhadern
U6
Forstenrieder Allee
Basler Str.
Fürstenried West
U3
Aidenbachstr.
Machtlfinger Str.
Theresienwiese
Heimeranplatz
Schwanthalerhöhe
Harras
Mittersendling
Obersendling
Siemenswerke
Solln
Großhesselohe Isartalbf.
Pullach
Höllriegelskreuth
Buchenhain
Baierbrunn
Hohenschäftlarn
Ebenhausen-Schäftlarn
Icking
Wolfratshausen
S7
Sendlinger Tor
Fraunhoferstr.
Goetheplatz
Poccistr.
Implerstr.
Brudermühlstr.
Thalkirchen (Tierpark)
Isar
Kolumbus-platz
Silberhorn-str.
Untersberg-str.
Candidplatz
Wettersteinplatz
St.-Quirin-Platz
Mangfallplatz
U1
St.-Martin-Str.
Karl-Preis-Platz
Giesing
Perlach
Neuperlach Süd
U5
Fasangarten
Fasanenpark
Unterhaching
Taufkirchen
Furth
Deisenhofen
Sauerlach
Otterfing
Holzkirchen
S3
S20 S27
Innsbrucker Ring
Michaelibad
Quiddestraße
Neuperlach Zentrum
Therese-Giehse-Allee
Neubiberg
Ottobrunn
Hohenbrunn
Wächterhof
Höhenkirchen-Siegertsbrunn
Dürrnhaar
Aying
Peiß
Großhelfendorf
Kreuzstraße
S7
München XXL
Innenraum
Außenraum
Tarifzonen
Regionalzughalt
Fernzughalt
©MVV/Stand: Dezember 200


nach Petershausen S2
Karlsfeld
nach Freising und zum Flughafen S1
Feldmoching
U2
Hasenbergl
Dülferstraße
nach Garching-Forschungszentrum U6
zum Flughafen S8
Harthof
Kieferngarten
Allach
Fasanerie
Am Hart
Freimann
Unterföhring
Olympia-Einkaufszentrum OEZ
Oberwiesenfeld
Frankfurter Ring
U1 U3
Olympia-zentrum
Moosach
Studentenstadt
Untermenzing
Milbertshofen
Alte Heide
Georg-Brauchle-Ring
Petuelring
Isar
nach Mammendorf
Westfriedhof
Scheidplatz
Bonner Platz
Nordfriedhof
Johanneskirchen
Langwied
Obermenzing
Dietlindenstraße
Hohenzollernplatz
Münchner Freiheit
Gern
Josephsplatz
Englschalking
U4
Theresienstraße
Giselastraße
Arabellapark
Pasing
Rotkreuzplatz
Stiglmaier-platz
Königs-platz
Universität
Richard-Strauss-Str.
Daglfing
Westkreuz
DB
S20
Laim
Maillingerstraße
S27
Karlsplatz (Stachus)
Odeonsplatz
Böhmerwaldplatz
Donnersberger-brücke
Hacker-brücke
DB
Haupt-bahnhof
Lehel
Max-Weber-Platz
Prinzregentenplatz
Riem
Marienplatz
Leuchtenbergring
Messe
Friedenheimer Straße
Theresienwiese
Isartor
Messestadt-West
U5
Laimer Platz
Schwan-thalerhöhe
Sendlinger Tor
Berg am Laim
Moosfeld
U4
Heimeran-platz
Rosenheimer Platz
S1 S7
Trudering
Westend-straße
Goethe-platz
Fraun-hoferstr.
DB
Ostbahnhof
Josephsburg
Lochham
Poccistraße
Innsbrucker Ring
Kreillerstraße
Haderner Stern
Westpark
Implerstraße
Harras
Kolumbusplatz
Michaelibad
Gronsdorf
Großhadern
Holzapfel-kreuth
Partnach-platz
St.-Martin-Str.
Karl-Preis-Platz
nach Ebersberg
U6
Klinikum Großhadern
Candidplatz
Silberhorn-str.
Brudermühl-straße
Mittersendling
Untersbergstr.
Giesing
Quiddestraße
Wettersteinplatz
St.-Quirin-Platz
Thalkirchen
Aidenbach-straße
Neuperlach Zentrum
Forstenrieder Allee
Obersendling
U1
Mangfallplatz
Therese-Giehse-Allee
Machtlfinger Straße
Perlach
Basler Straße
U5
U3
Siemenswerke
Fasangarten
Neuperlach Süd
Fürstenried West
nach Wolfratshausen S7 S27 S20 nach Deisenhofen
nach Holzkirchen S5
S6 nach Kreuzstraße
er / Stand: November 2007

Motivation

- Sehr breites Anwendungsspektrum
  - Planungssysteme
  - Optimierungsaufgaben
  - Uvm.
- Für uns: Shortest Path & Minimum Spanning Tree Probleme sehr häufig im ICPC

Ziel des Vortrags

- Funktionsweise von wichtigen Graphalgorithmen verstehen
- Unter Zeitdruck schnell programmieren
- Graphprobleme erkennen
- Algorithmen bei Bedarf an das Problem anpassen

Graphalgorithmen II

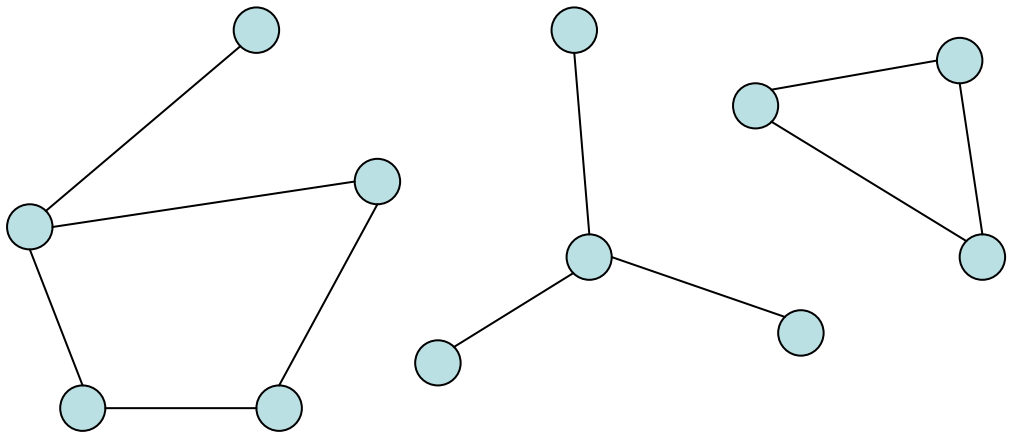
- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

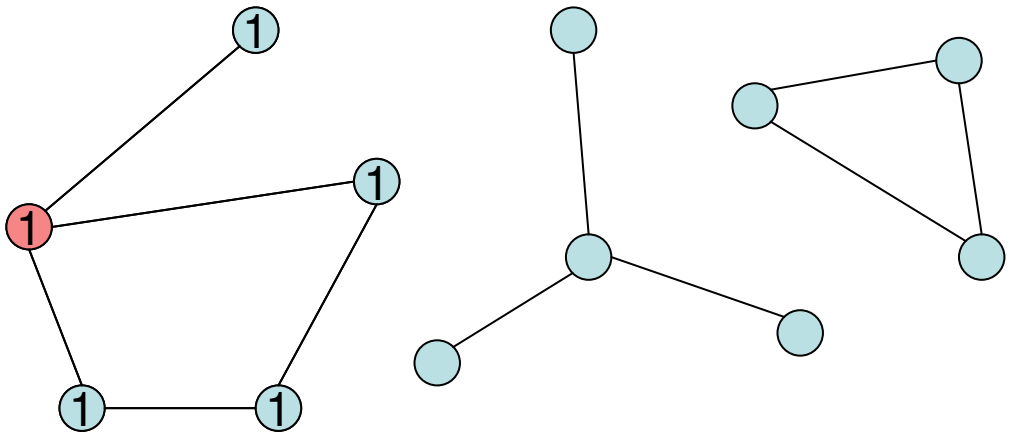
Union/Find

- Gegeben: Graph G(V, E)
- Problem: Finde die Zusammenhangskomponenten!



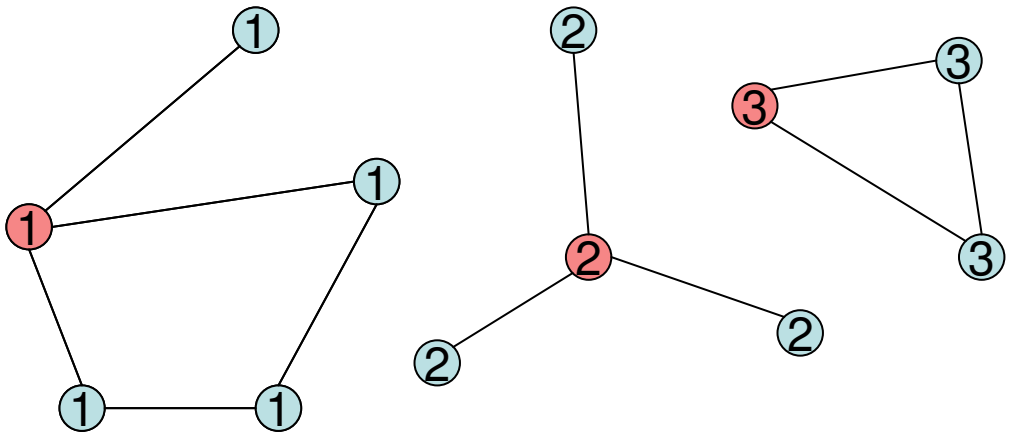
1. Idee

- Breitensuche von jedem Knoten aus
- Zahl zuweisen



1. Idee

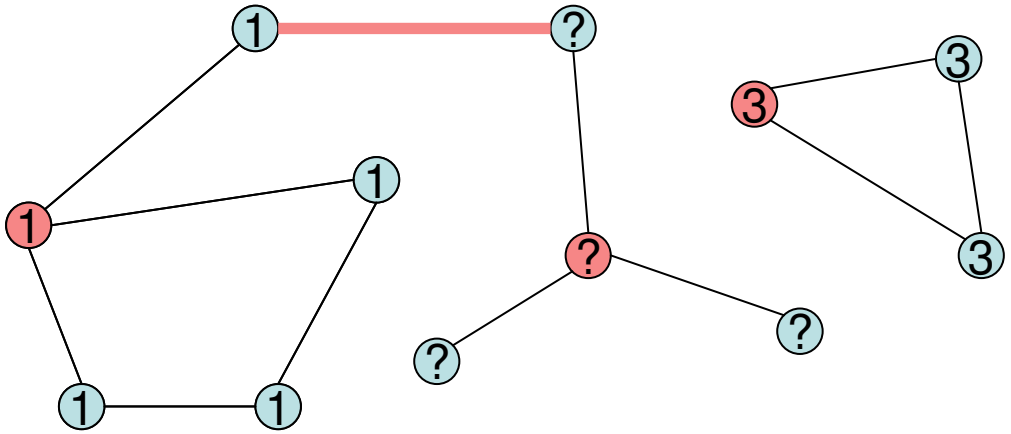
- Breitensuche von jedem Knoten aus
- Zahl zuweisen



1. Idee

- Breitensuche von jedem Knoten aus
- Zahl zuweisen

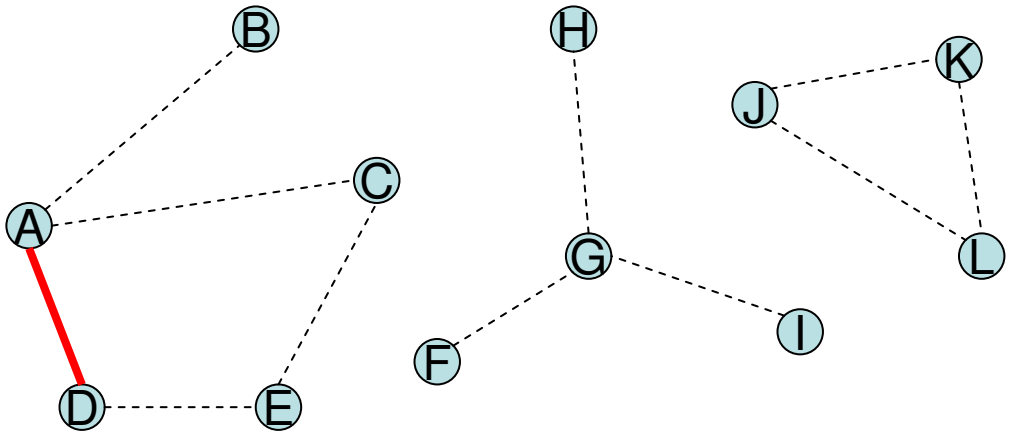
- Funktioniert bei *statischen* Graphen, aber:

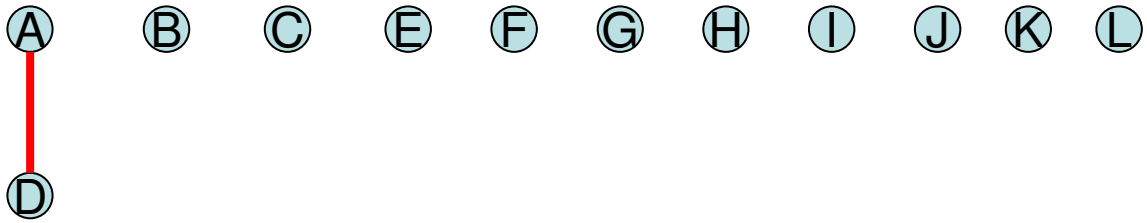


Besser:

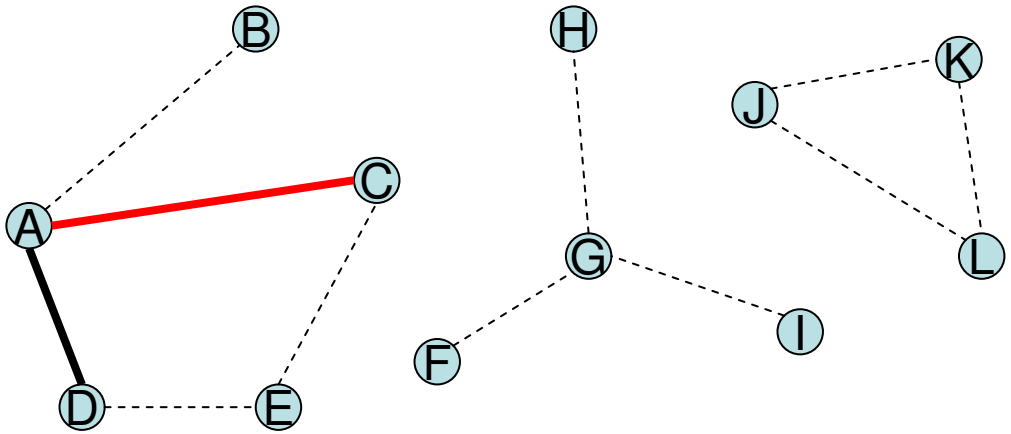
- Datenstruktur, die uns die Arbeit abnimmt
- Verwalten der Knoten in Bäumen
- Operationen auf Knoten x,y:
  - Union (neue Kante zwischen x und y)
  - Find (sind x und y in der gleichen Komponente?)

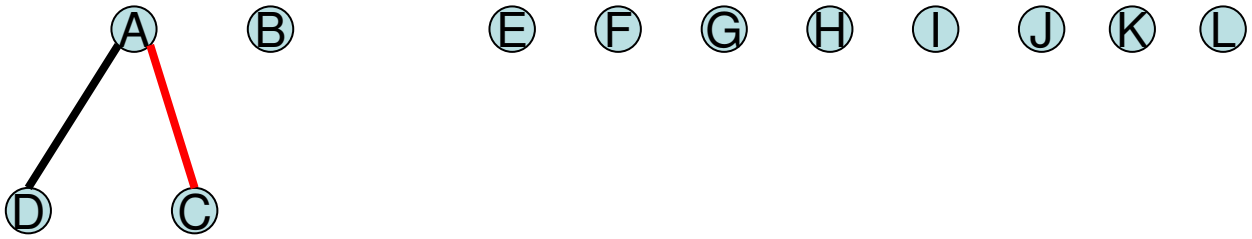
Beispiel



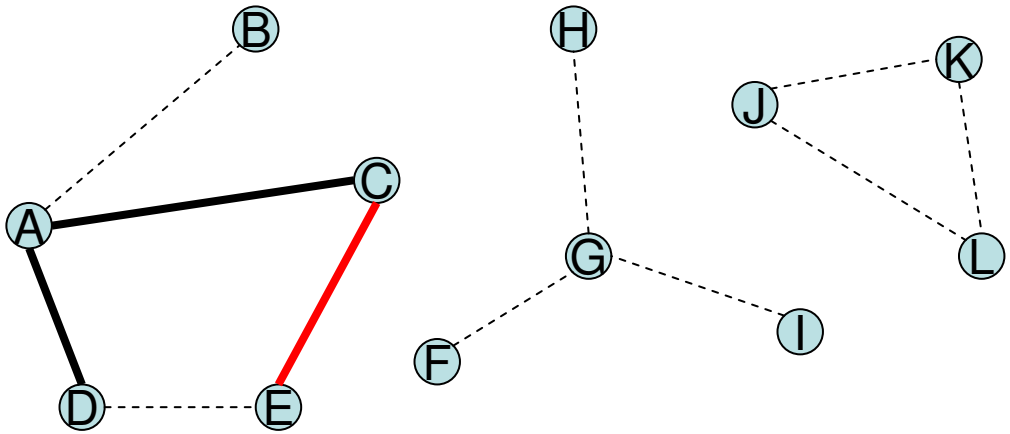


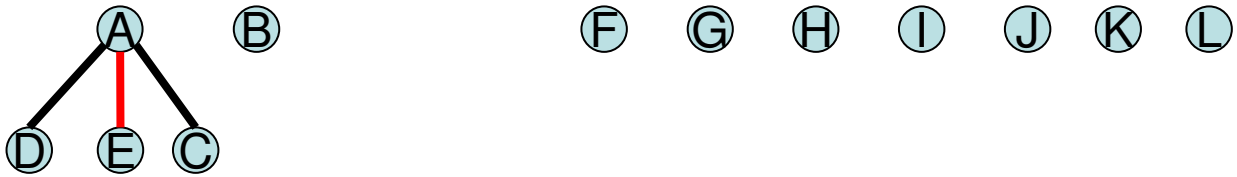
Beispiel



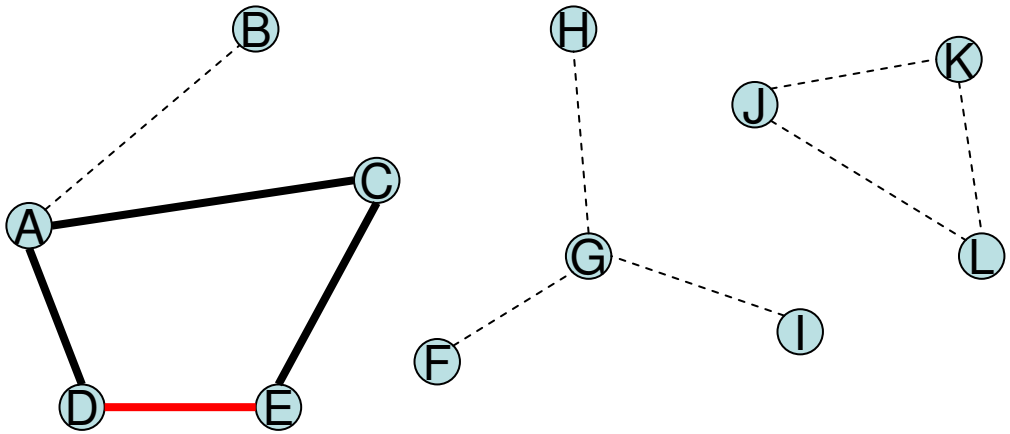


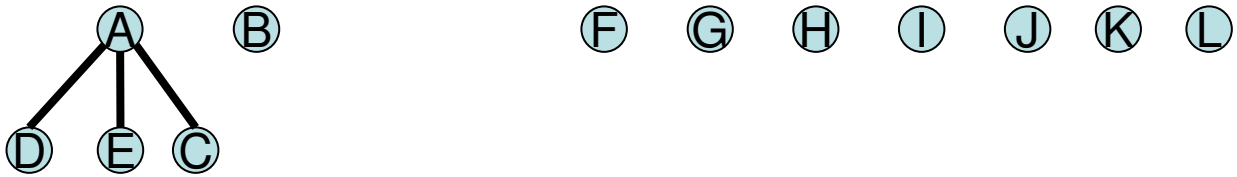
Beispiel



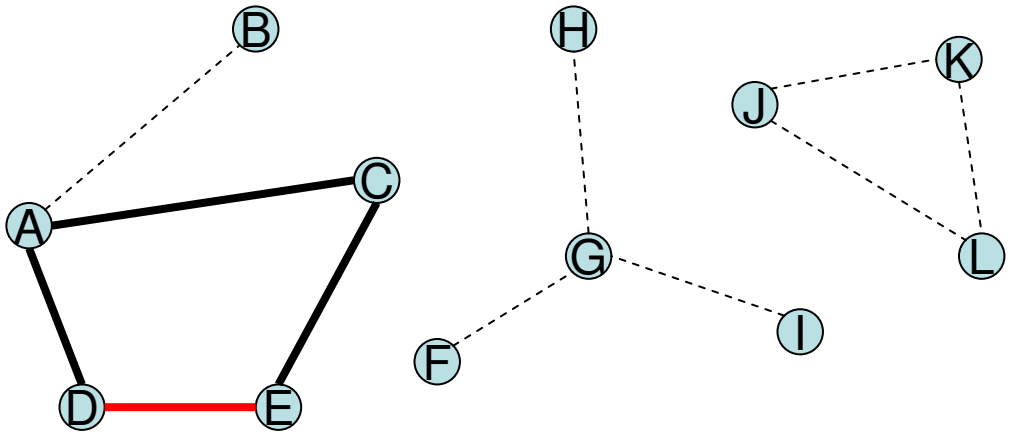


Beispiel

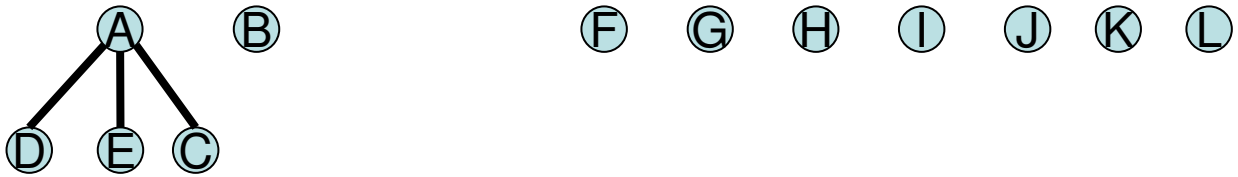




Beispiel

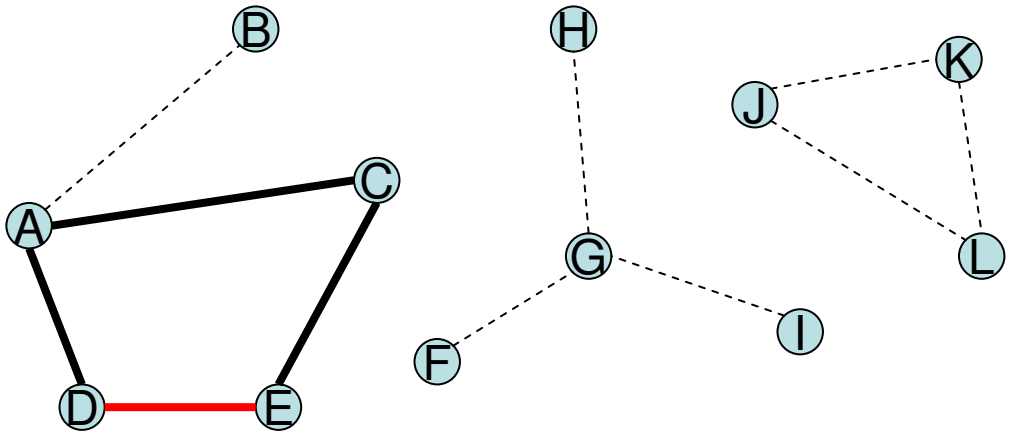


Nichts passiert!

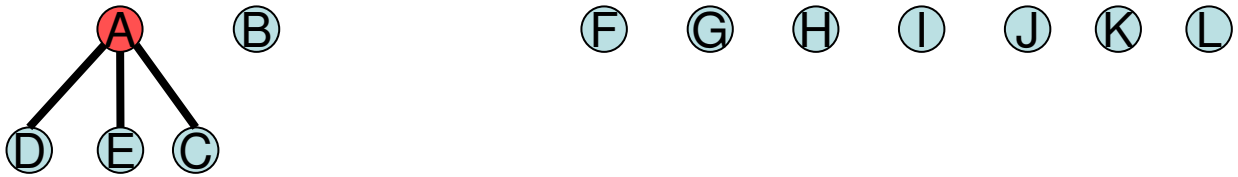


Wie kann man diesen Fall erkennen?

Beispiel

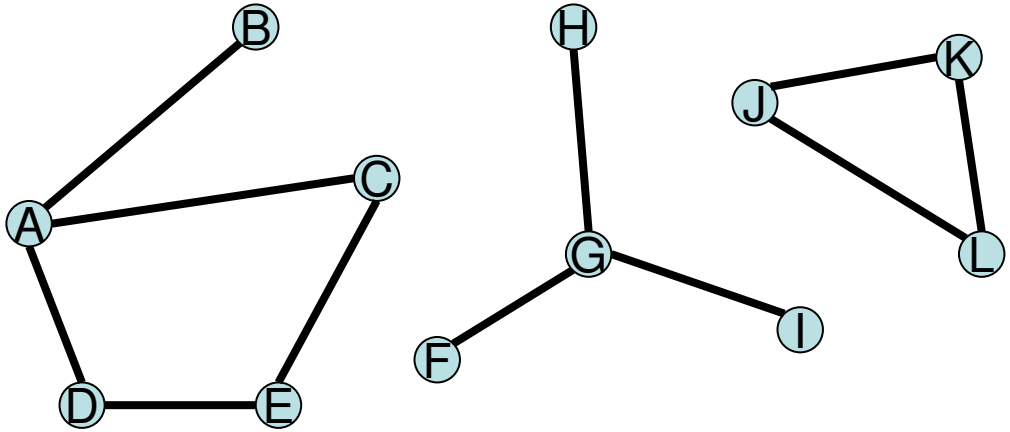


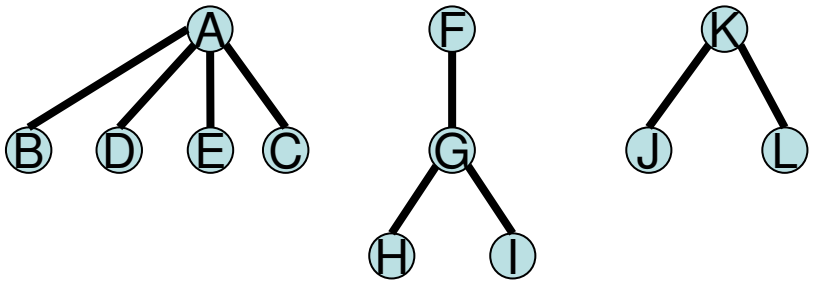
Nichts passiert!



Wie kann man diesen Fall erkennen?

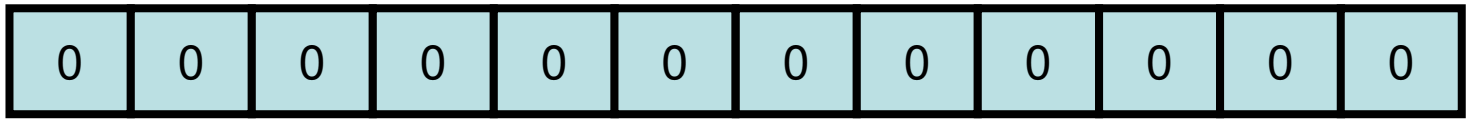
Beispiel





Union/Find

- Eine Komponente wird also durch die Wurzel des Baumes repräsentiert
- Wir müssen nur nach oben iterieren
- Verwalten der Bäume in einem Array:




```
const int maxvertices = 100000;
int dad[maxvertices];
memset(dad, 0, maxvertices * sizeof(int));
```

Union/Find

```
int find(int x, int y, bool union) {
     int i = x, j = y;
     while (dad[i] > 0) i = dad[i];
     while (dad[j] > 0) j = dad[j];
     if(union && (i != j)) dad[j] = i;
     return (i != j);
}
```

Union/Find

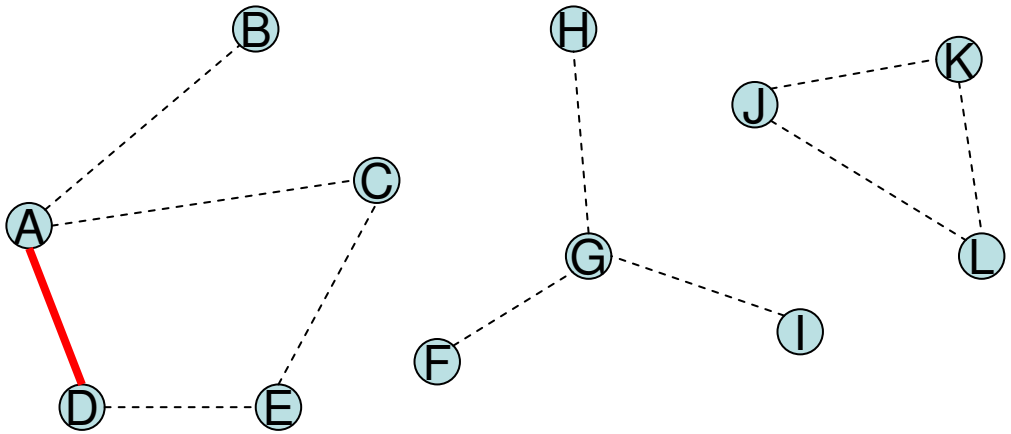
```
int find(int x, int y, bool union) {
    int i = x, j = y;
    while (dad[i] > 0) i = dad[i];
    while (dad[j] > 0) j = dad[j];
    if(union && (i != j)) dad[j] = i;
    return (i != j);
}
```

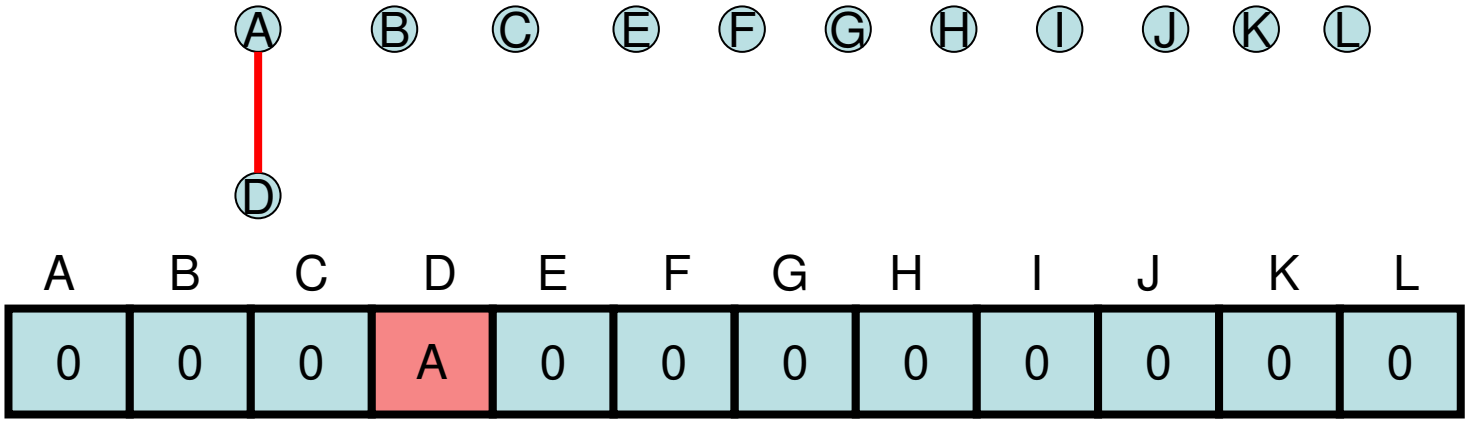
1. Wurzel

2. Wurzel

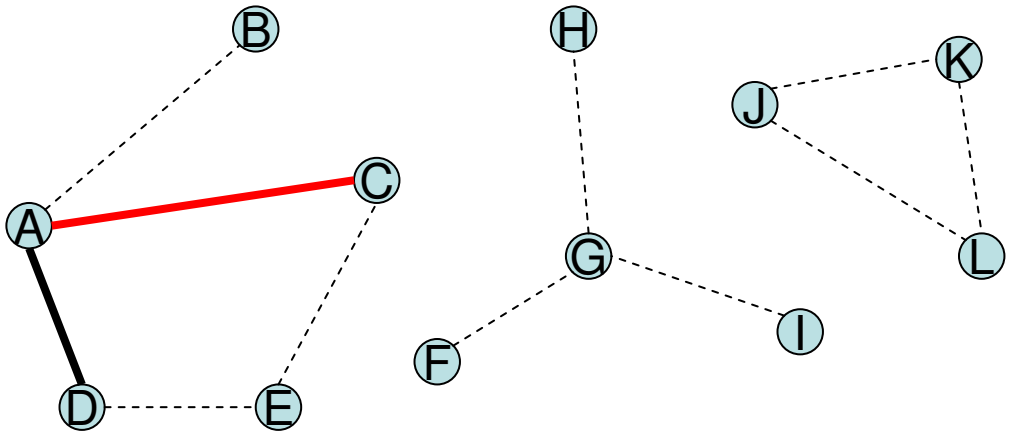
Union

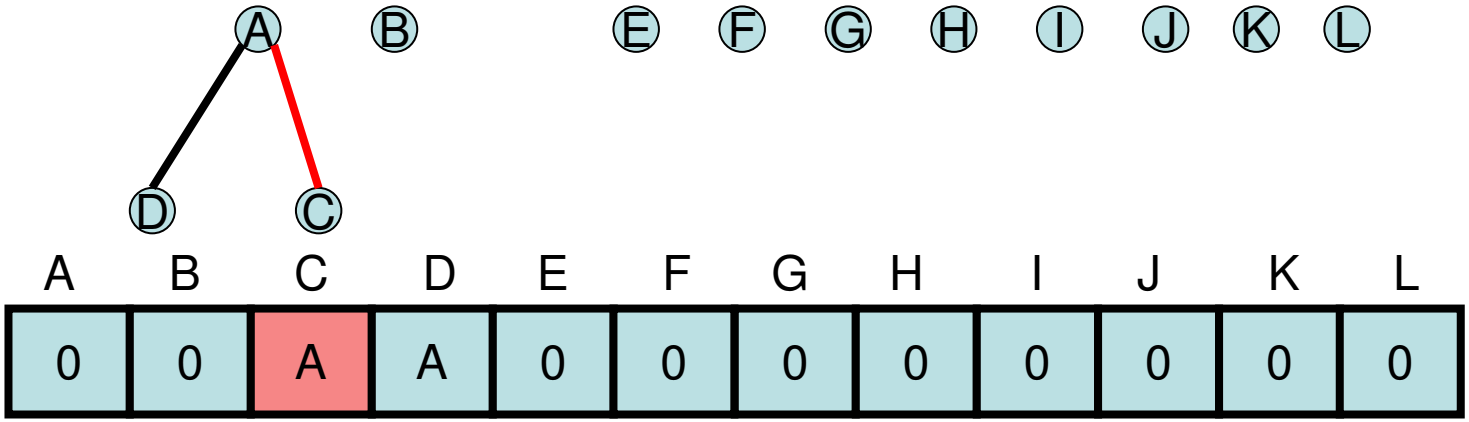
Beispiel



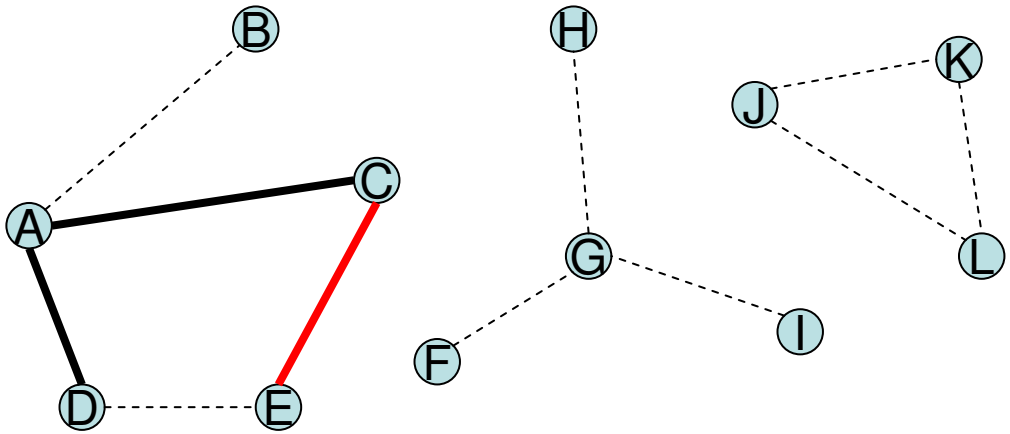


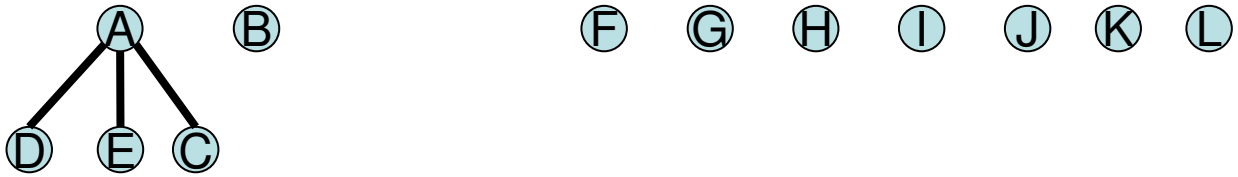
Beispiel





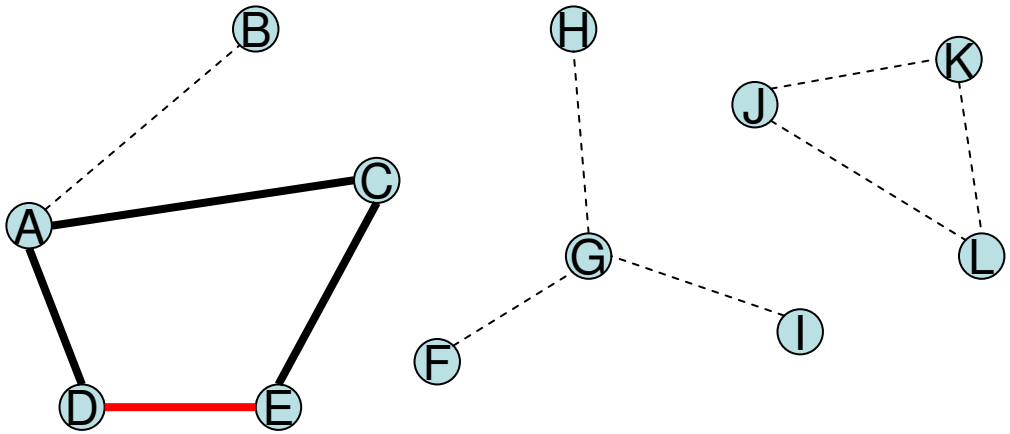
Beispiel

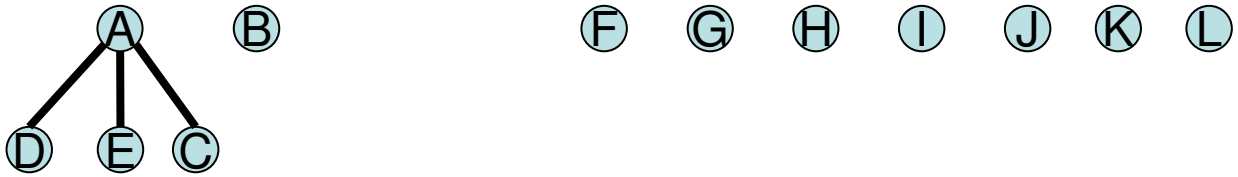




A	B	C	D	E	F	G	H	I	J	K	L
0	0	A	A	A	0	0	0	0	0	0	0

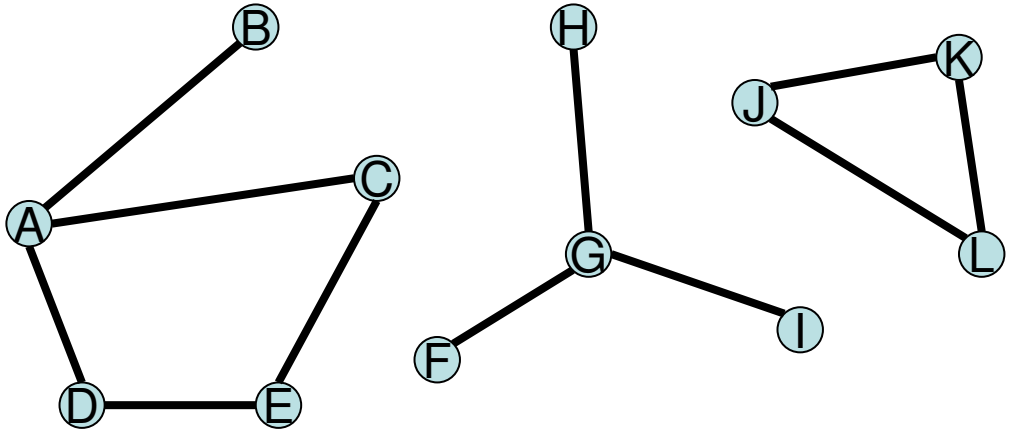
Beispiel

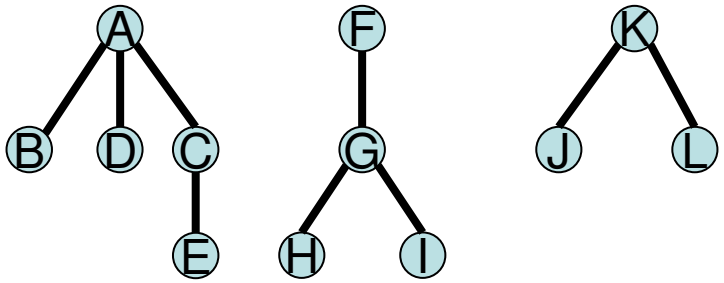




A	B	C	D	E	F	G	H	I	J	K	L
0	0	A	A	A	0	0	0	0	0	0	0

Beispiel

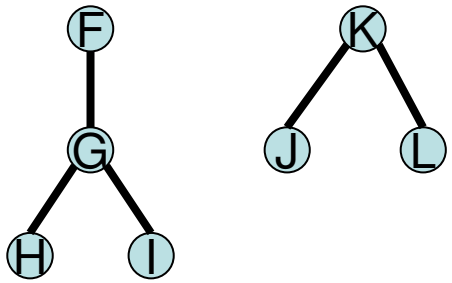




A	B	C	D	E	F	G	H	I	J	K	L
0	A	A	A	A	0	F	G	G	K	0	K

Union/Find

- Verallgemeinerung:
  - Die Relation „ist verbunden mit“ ist eine Äquivalenzrelation (refl., trans., symm.)
  - Allgemein: Die Union/Find-Struktur verwaltet eine Äquivalenzrelation auf Mengen
  - {F,G,H,I}, {K,J,L}



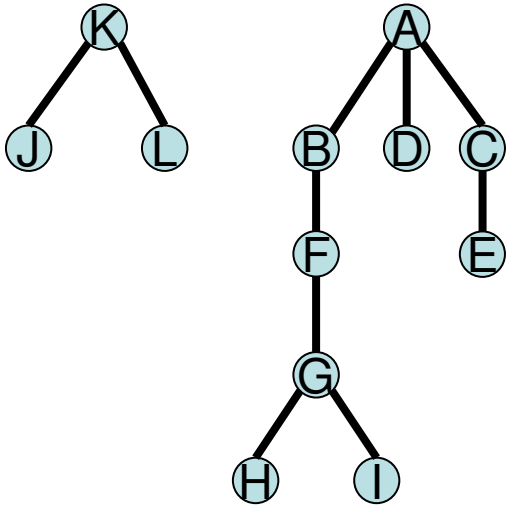
Union/Find (Komplexität)

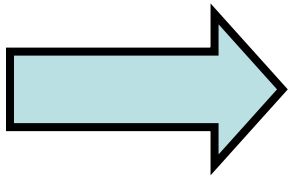
- Das Verfahren kann im Worst-Case entarten => O(|E|)

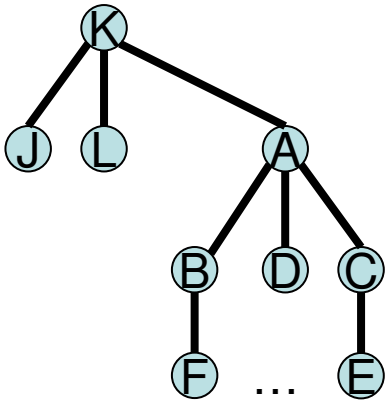
- Heuristiken:
  - Gewichtausgleichen (Balancing)
  - Pfadverdichtung (Path compression)

Balancing

- Problem:



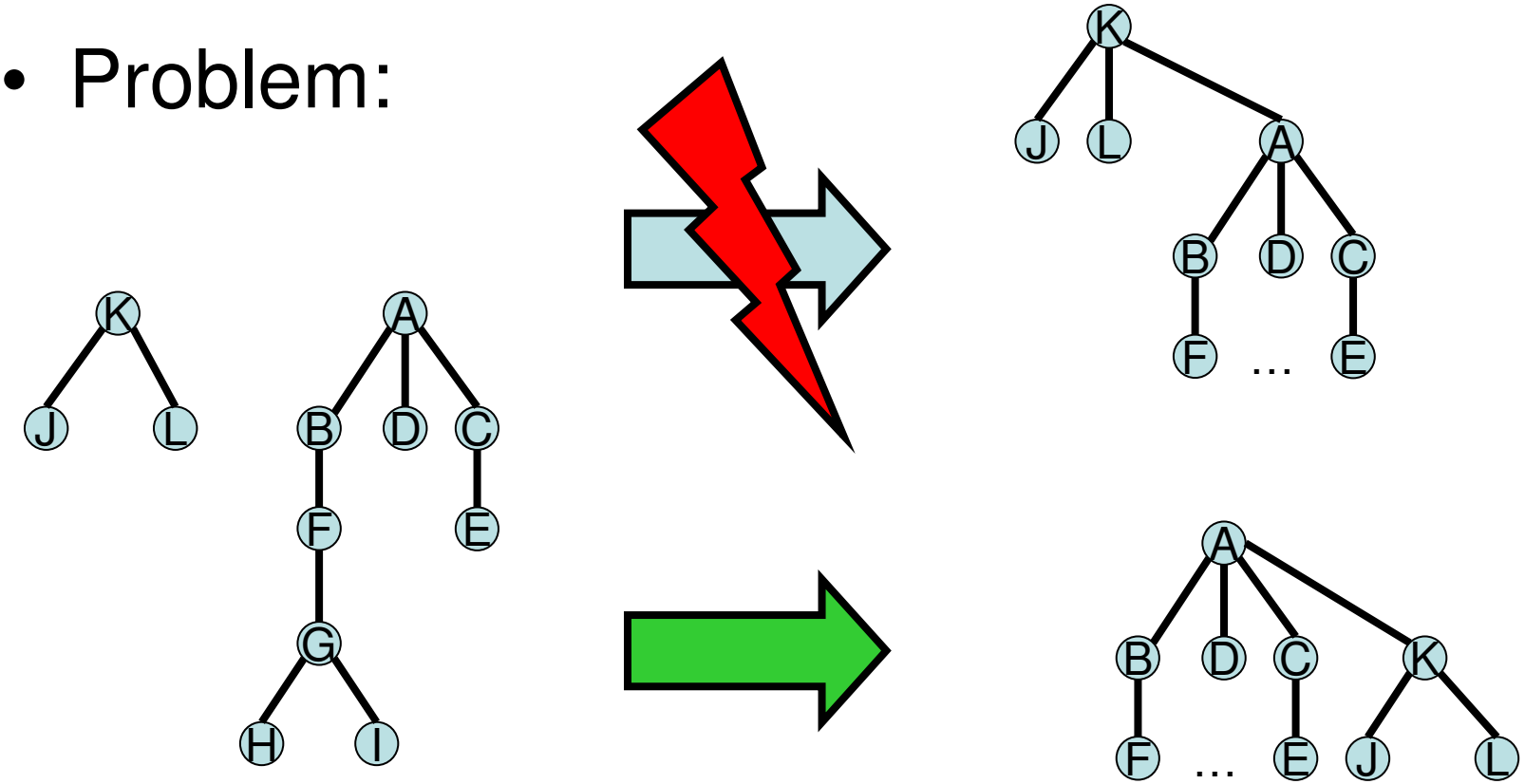




find(L, G, true)!

Balancing

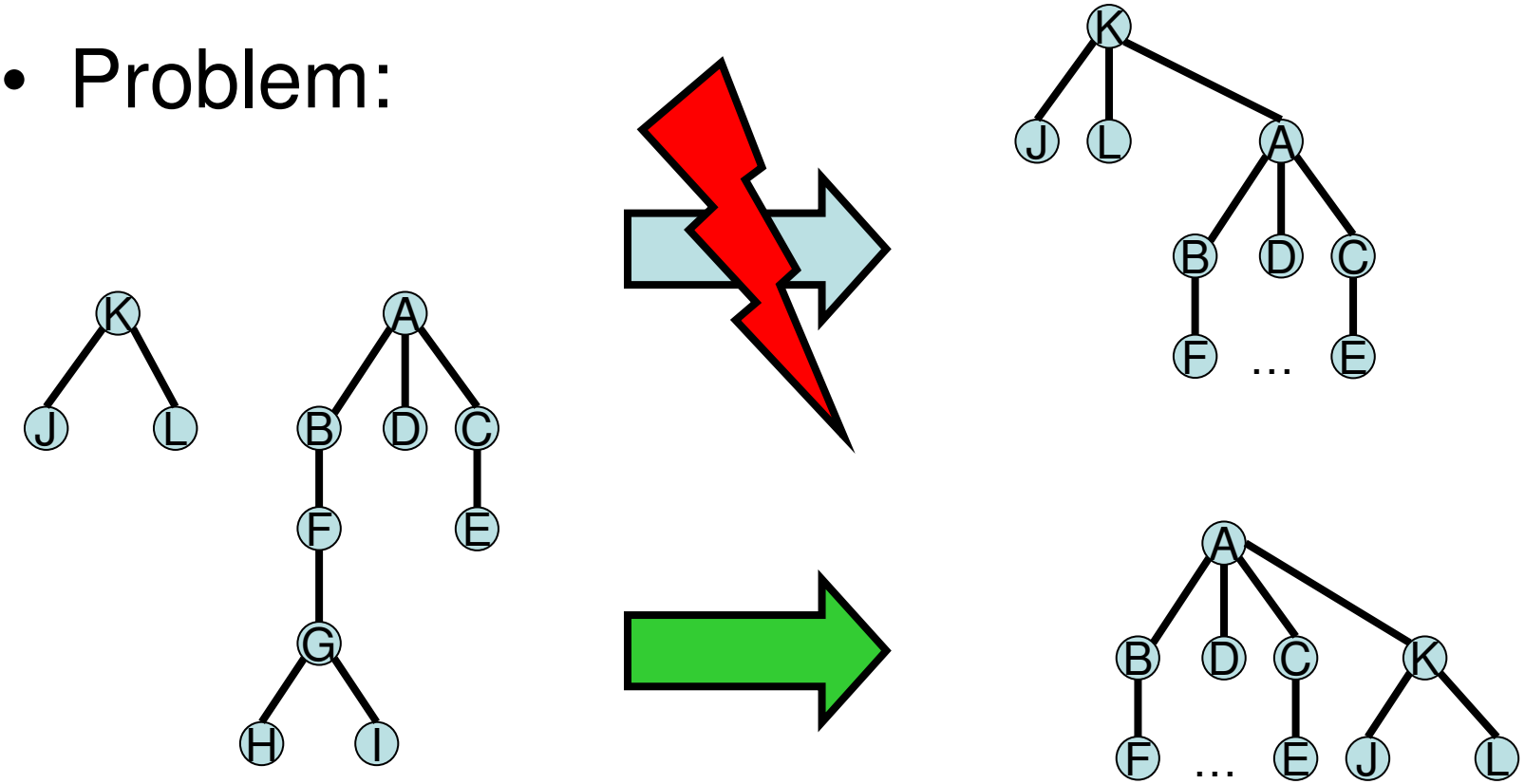
- Problem:



find(L, G, true)!

Balancing

- Problem:

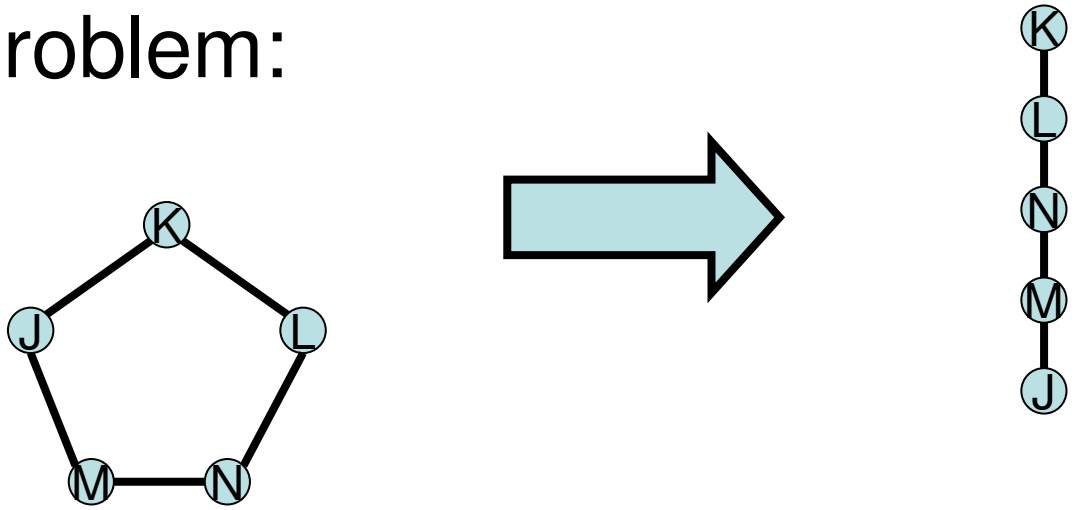


find(L, G, true)!

Idee: Speichere in dad[i] der Wurzel die Anzahl der Nachfolger (negativ)!

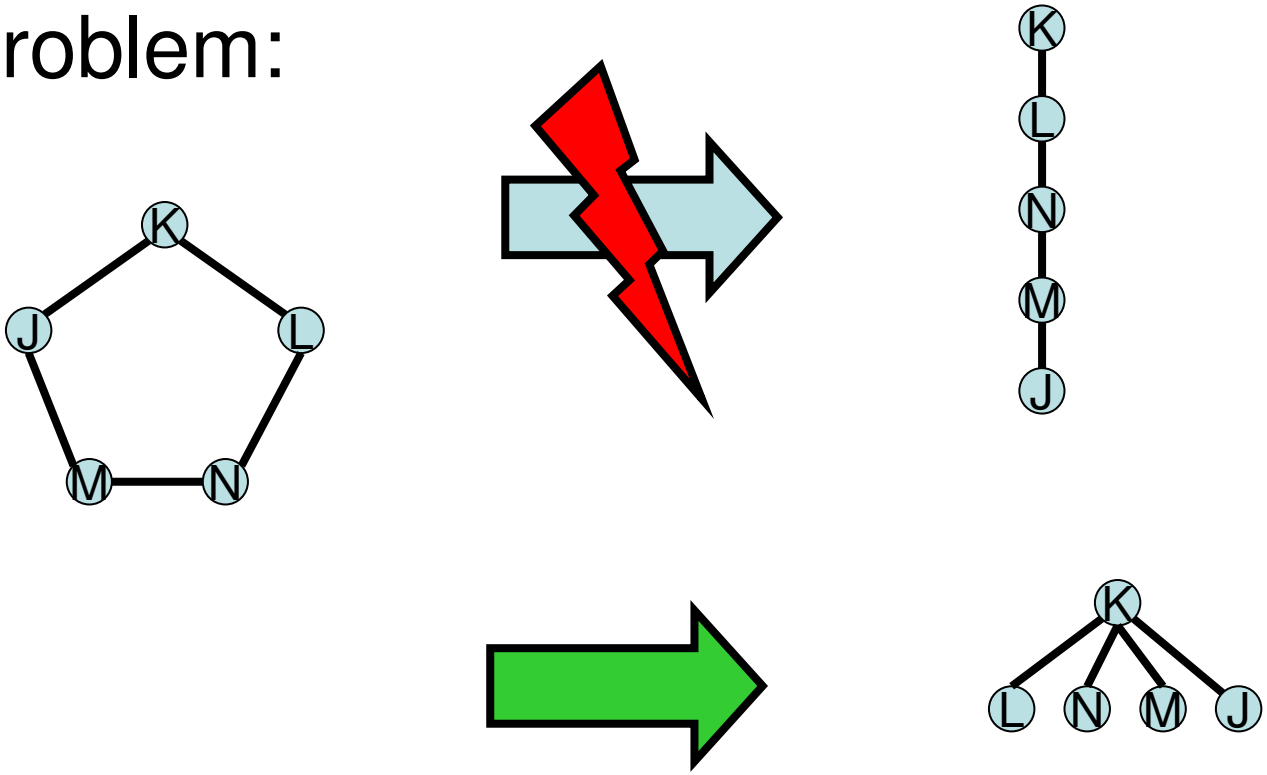
Path Compression

- Problem:



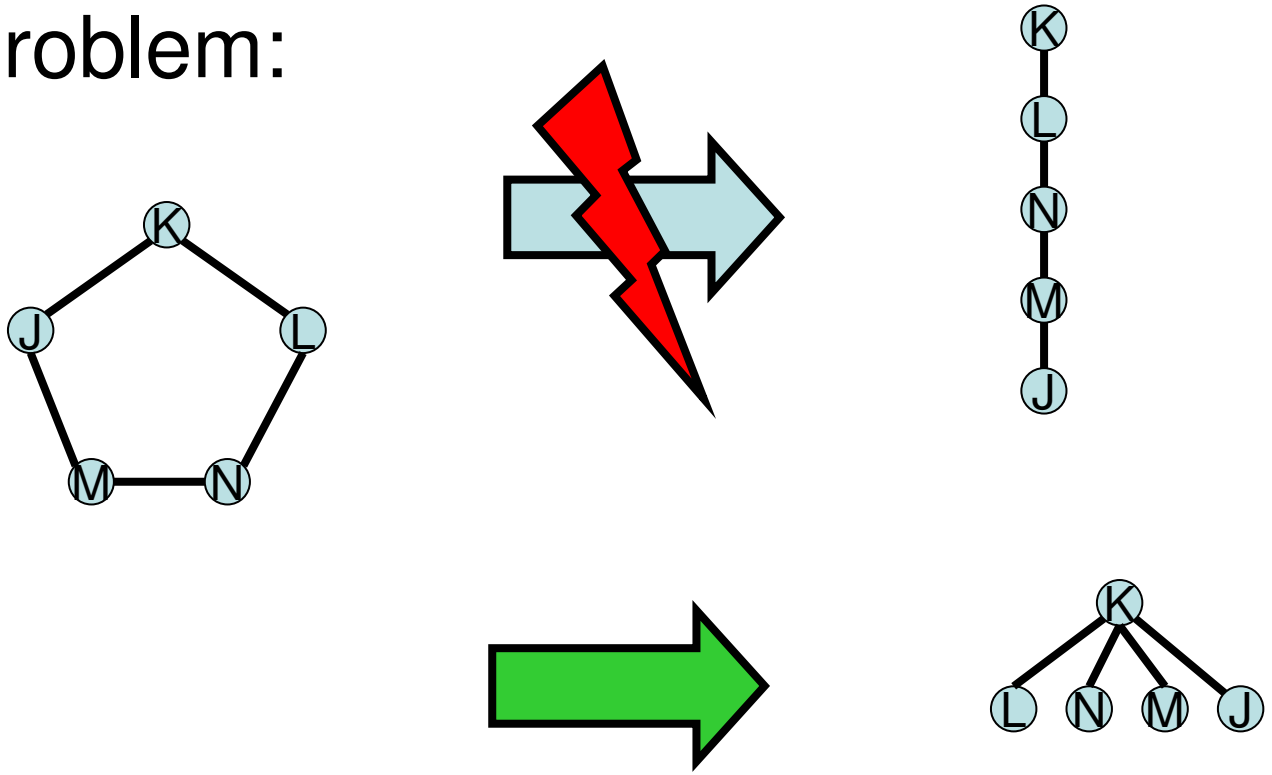
Path Compression

- Problem:



Path Compression

- Problem:



Idee: Alle Knoten, die wir prüfen, auf die Wurzel zeigen lassen!

Union/Find

- Mit Balancing/Path Compression können wir die Laufzeit auf O(α) verbessern
  - O(α) ≈ O(1)

- Code: Übungen!

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

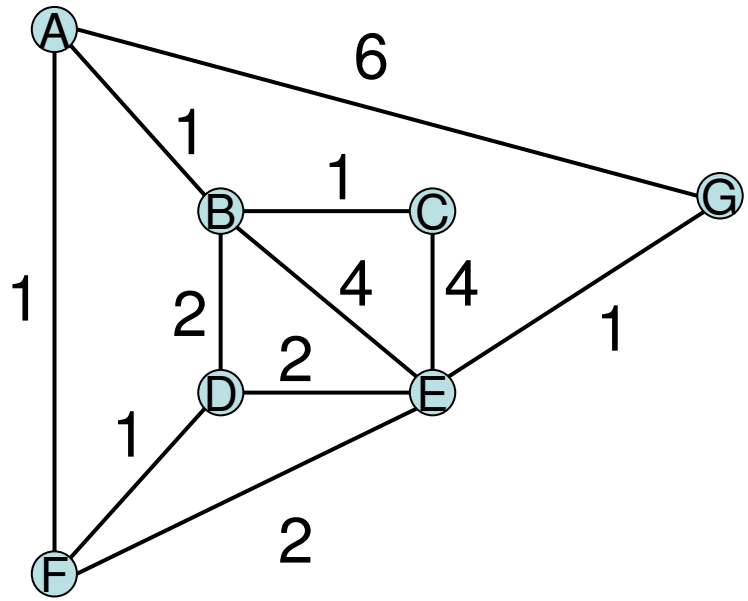
Minimale Spannbäume

- G(V, E) ungerichteter Graph
- Spannbaum: Baum mit Kanten aus E, der alle Knoten des Graphs umspannt.

- Minimaler Spannbaum (MST):
  Spannbaum, dessen Summe der Kantengewichte unter allen Spannbäumen minimal ist.

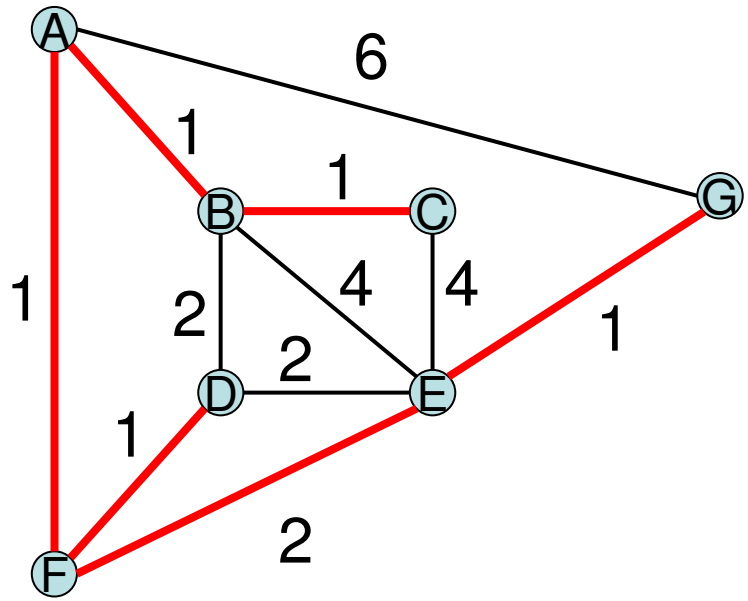
Minimale Spannbäume

- Eingabe: gew. Graph G(V,E,δ: E → R)
- Problem: Finde einen minimalen Spannbaum (MST) von G!



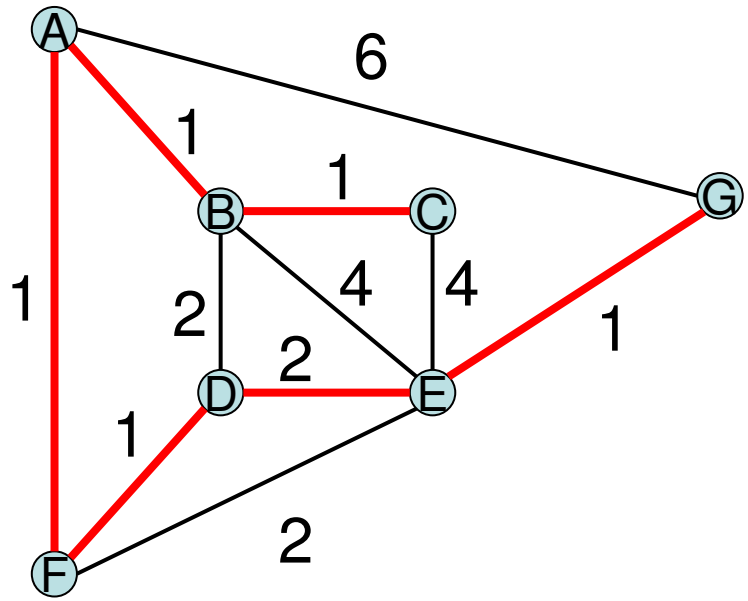
Minimale Spannbäume

- Eingabe: gew. Graph G(V,E,δ: E → R)
- Problem: Finde einen minimalen Spannbaum (MST) von G!



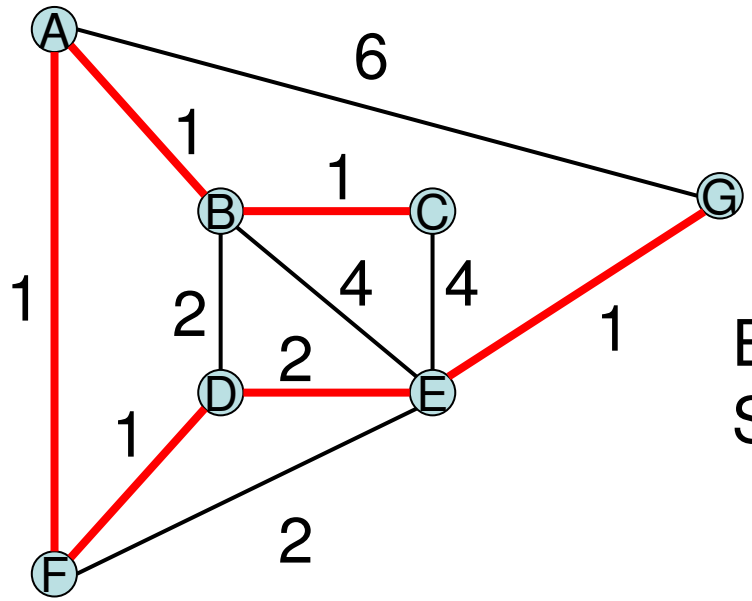
Minimale Spannbäume

- Eingabe: gew. Graph G(V,E,δ: E → R)
- Problem: Finde einen minimalen Spannbaum (MST) von G!



Minimale Spannbäume

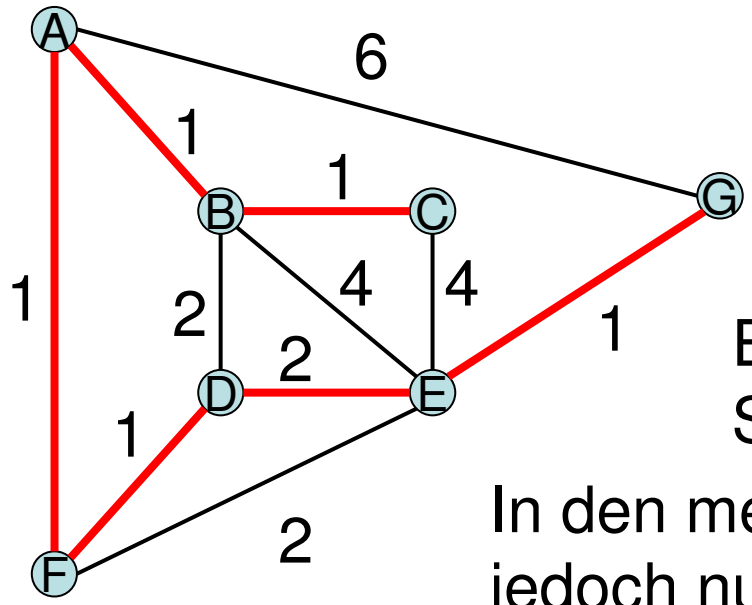
- Eingabe: gew. Graph G(V,E,δ: E → R)
- Problem: Finde einen minimalen Spannbaum (MST) von G!



Es kann viele minimale Spannbäume geben!

Minimale Spannbäume

- Eingabe: gew. Graph G(V,E,δ: E → R)
- Problem: Finde einen minimalen Spannbaum (MST) von G!



Es kann viele minimale Spannbäume geben!

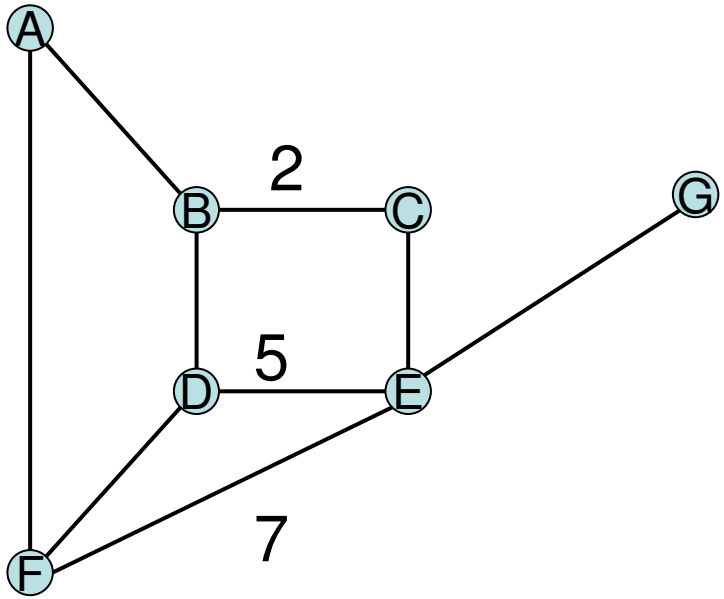
In den meisten Fällen wird jedoch nur einer benötigt.

Minimale Spannbäume

- Interessante Eigenschaft eines MST:
  - Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.

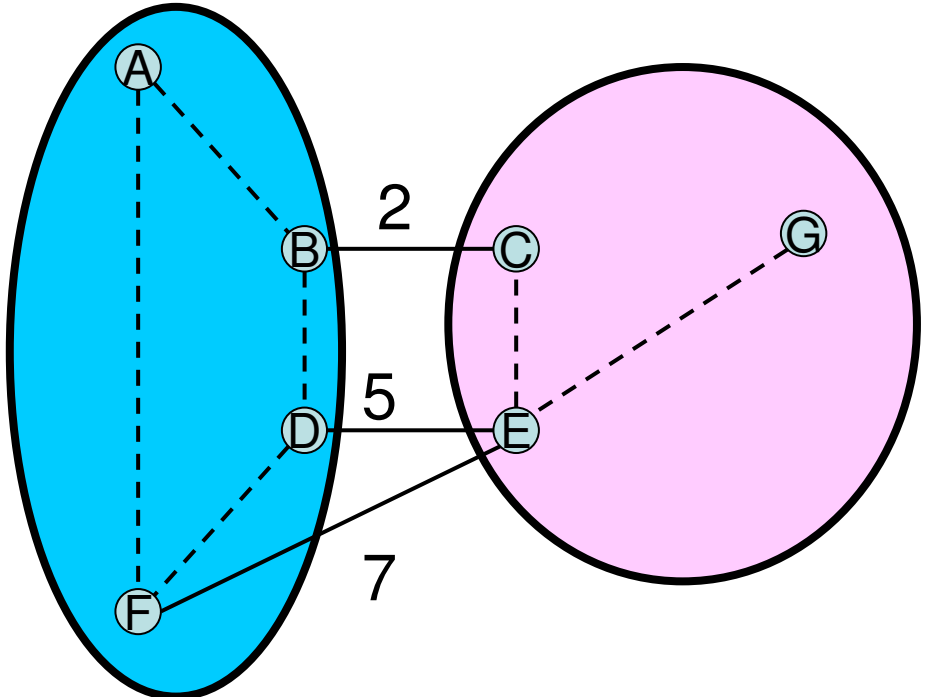
Minimale Spannbäume

- Interessante Eigenschaft eines MST:
  - Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.



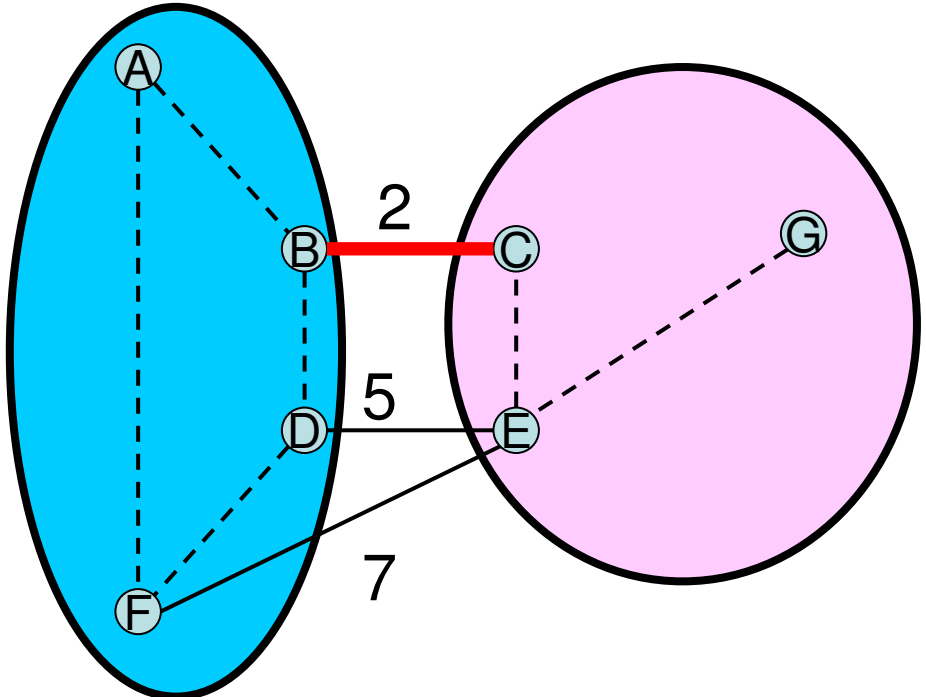
Minimale Spannbäume

- Interessante Eigenschaft eines MST:
  - Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.



Minimale Spannbäume

- Interessante Eigenschaft eines MST:
  - Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.



Minimale Spannbäume

- Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.
- Beweis durch Widerspruch:

Minimale Spannbäume

- Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.
- Beweis durch Widerspruch:
  - s sei die echt kürzeste Kante, die zwei Mengen A und B verbindet

Minimale Spannbäume

- Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.
- Beweis durch Widerspruch:
  - s sei die echt kürzeste Kante, die zwei Mengen A und B verbindet
  - Ann: es gibt einen MST(V,E‘) und s ist nicht darin enthalten

Minimale Spannbäume

- Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.
- Beweis durch Widerspruch:
  - s sei die echt kürzeste Kante, die zwei Mengen A und B verbindet
  - Ann: es gibt einen MST(V,E‘) und s ist nicht darin enthalten
  - Füge s zu diesem MST hinzu : enthält einen Zyklus

Minimale Spannbäume

- Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.
- Beweis durch Widerspruch:
  - s sei die echt kürzeste Kante, die zwei Mengen A und B verbindet
  - Ann: es gibt einen MST(V,E‘) und s ist nicht darin enthalten
  - Füge s zu diesem MST hinzu : enthält einen Zyklus
  - Insbesondere enthält G neben s eine Kante, die die Mengen A und B verbindet

Minimale Spannbäume

- Für jede gegebene Zerlegung eines Graphen in zwei Mengen enthält der MST die kürzeste der Kanten, die Knoten aus der einen Menge mit denen der anderen verbindet.
- Beweis durch Widerspruch:
  - s sei die echt kürzeste Kante, die zwei Mengen A und B verbindet
  - Ann: es gibt einen MST(V,E‘) und s ist nicht darin enthalten
  - Füge s zu diesem MST hinzu : enthält einen Zyklus
  - Insbesondere enthält G neben s eine Kante, die die Mengen A und B verbindet
  - Das Löschen dieser Kante und Hinzufügen von s ergibt einen kürzeren Spannbaum!

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

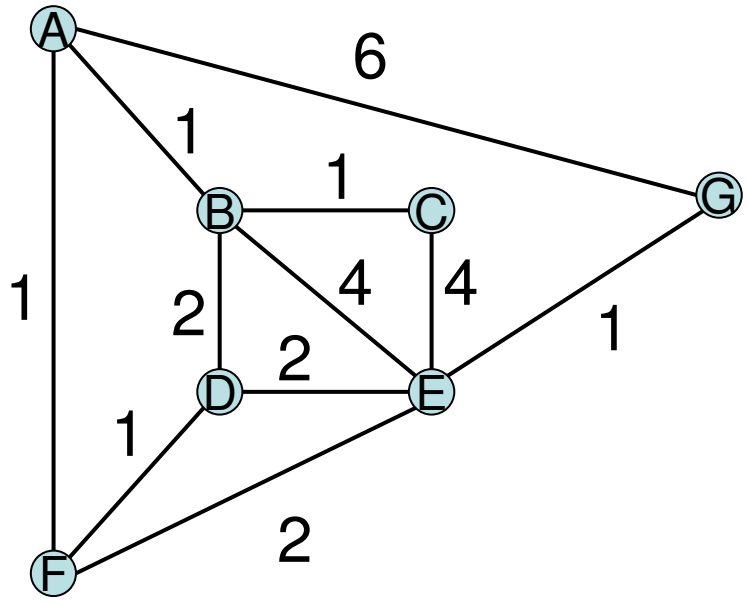
Prim

- Wir verwenden die Eigenschaft für den folgenden Algorithmus:
  - Beginne bei einem bel. Knoten
  - Füge immer den Knoten mit minimalem Abstand zum MST hinzu
- Wir benötigen dafür eine PriorityQueue (PQ)

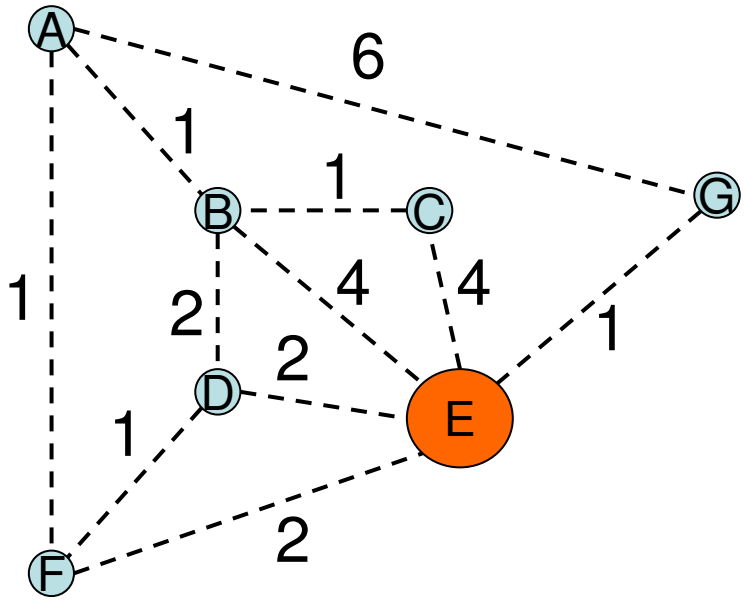
Prim

- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)

- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)



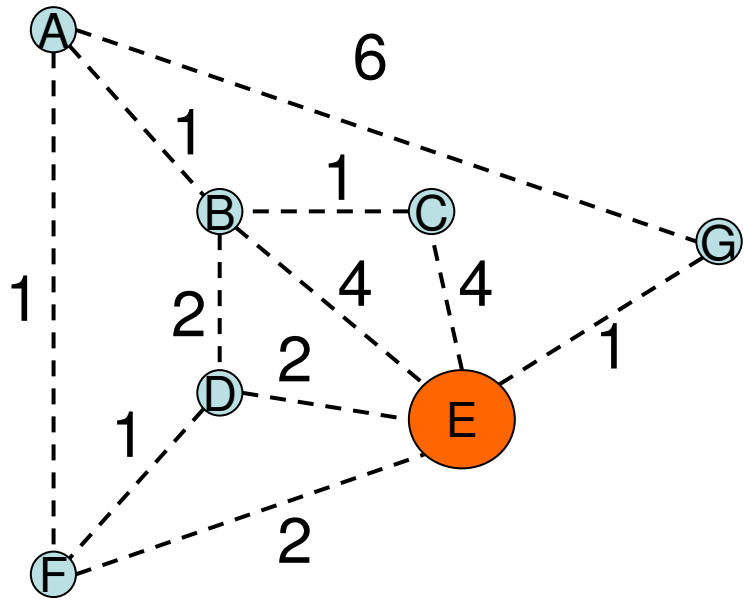
- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)



0

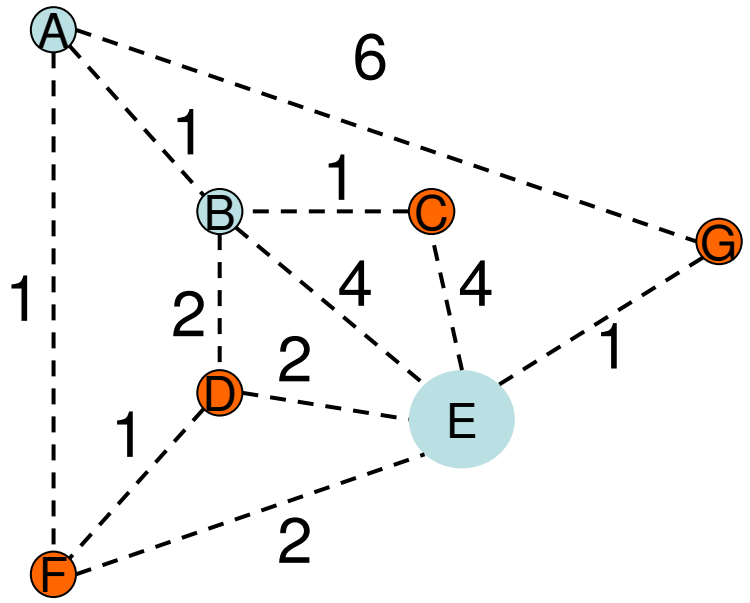
PQ

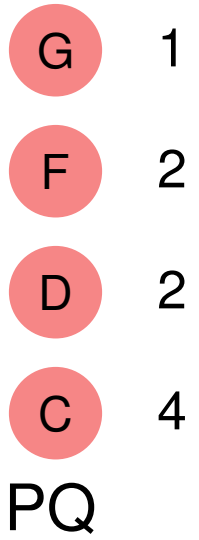
- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)



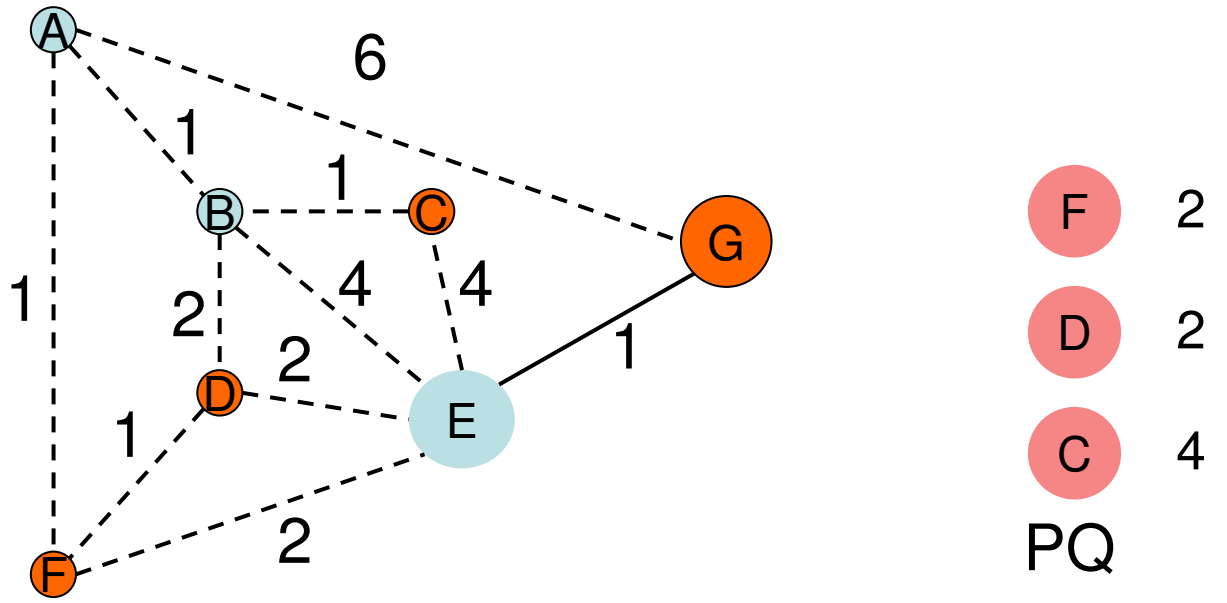
PQ

- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)

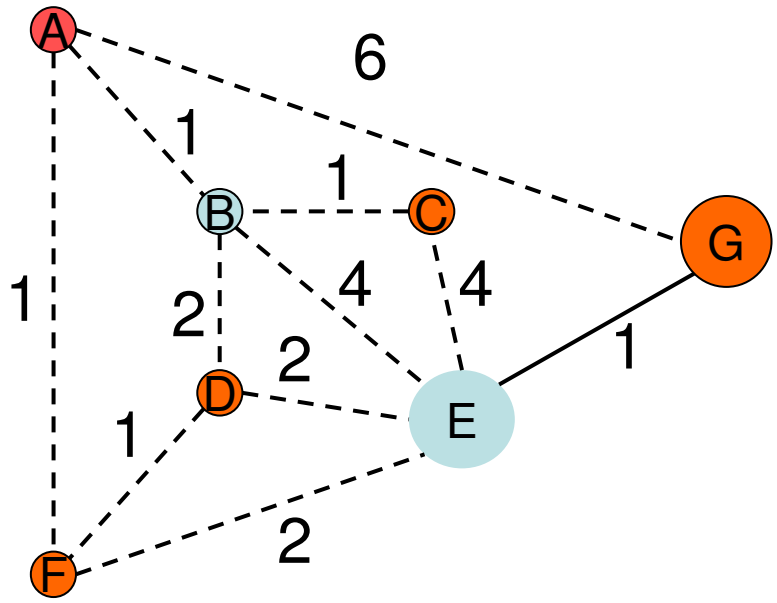


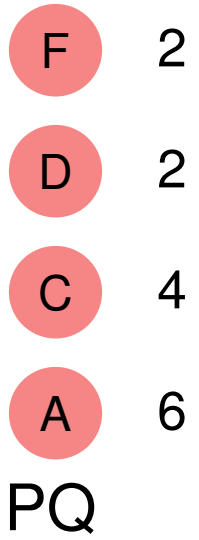


- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)

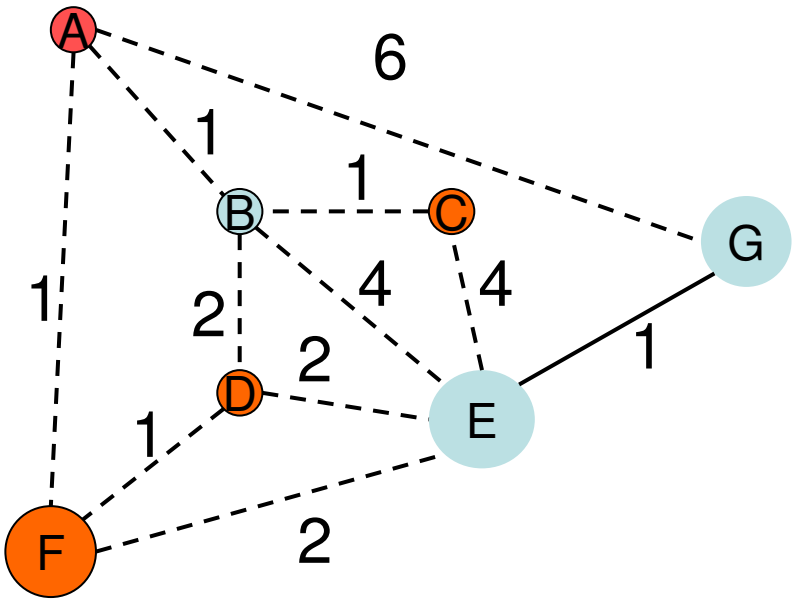


- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)

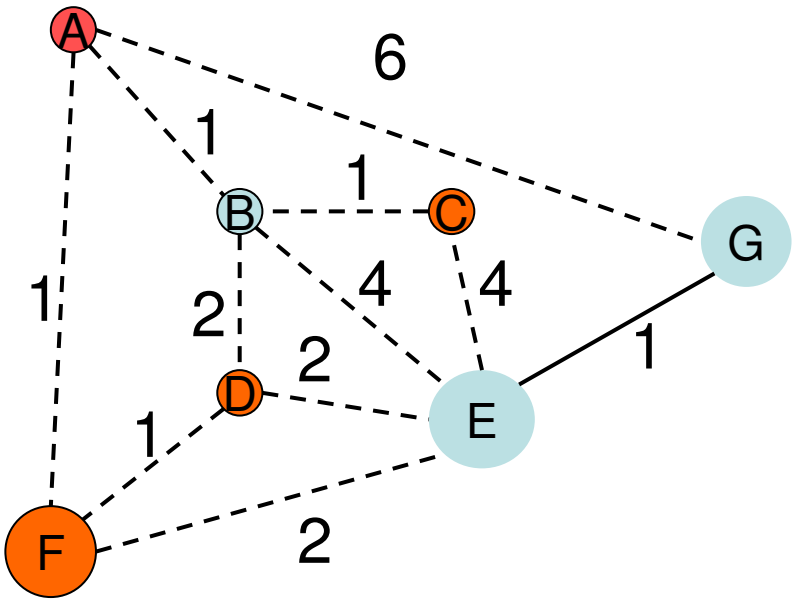


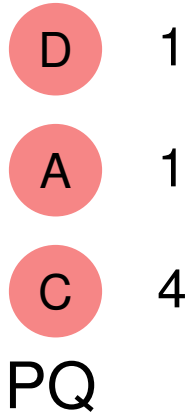


- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)

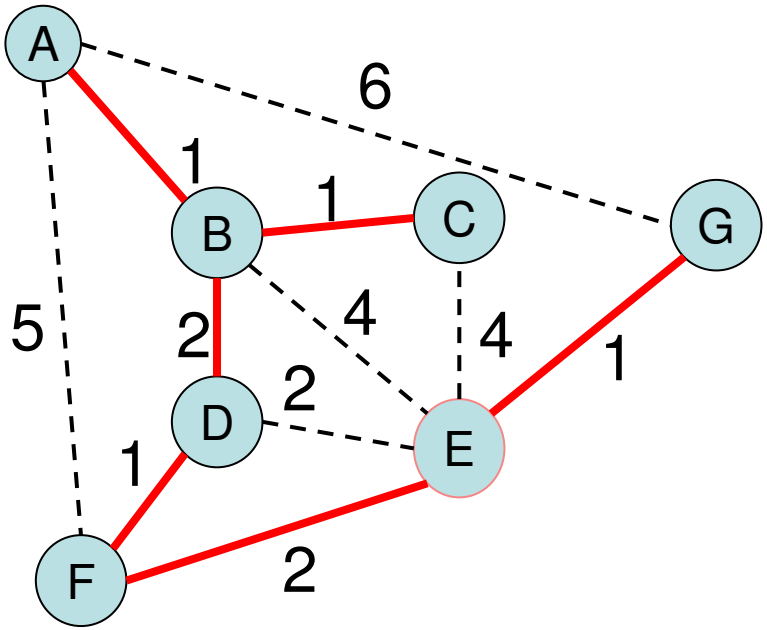


- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)





- Start: Wähle beliebigen Knoten N
- Füge N in die PQ (Abstand 0)
- Solange PQ nicht leer:
  - Füge Knoten mit minimalem Abstand zum MST hinzu (extract_min)
  - Füge nicht-MST Nachbarn des neuen Knoten in PQ (insert bzw. decrease_key)



PQ

Prim – Komplexität

- Hängt von der Implementierung der PQ ab
  - Standardfall: $O((|E| + |V|) \log |V|)$
  - AF-Heap sogar: $O(|E| + |V|)$

Prim – Komplexität

- Hängt von der Implementierung der PQ ab
  - Standardfall: $O((|E| + |V|) \log |V|)$
  - ~~AF-Heap sogar: $O(|E| + |V|)$~~
    - (nur theoretisch interessant)

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

Kruskal

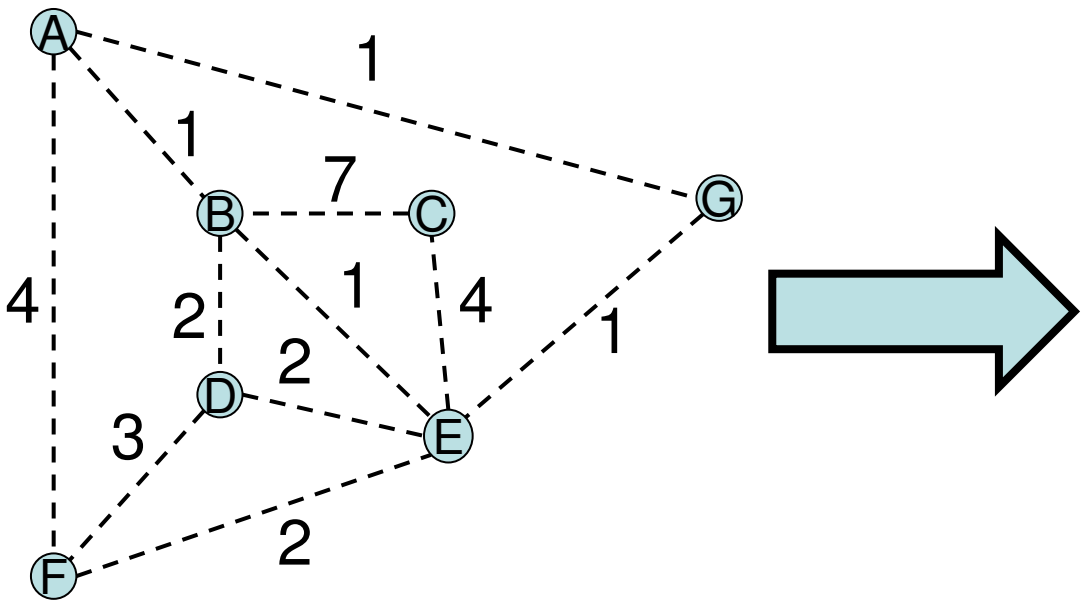
- Anderer Ansatz:
  - Wähle die „kürzeste“ noch nicht betrachtete Kante e
  - Ist das jetzige Ergebnis + e kreisfrei?
    - Wenn ja, füge e zum MST hinzu

Kruskal

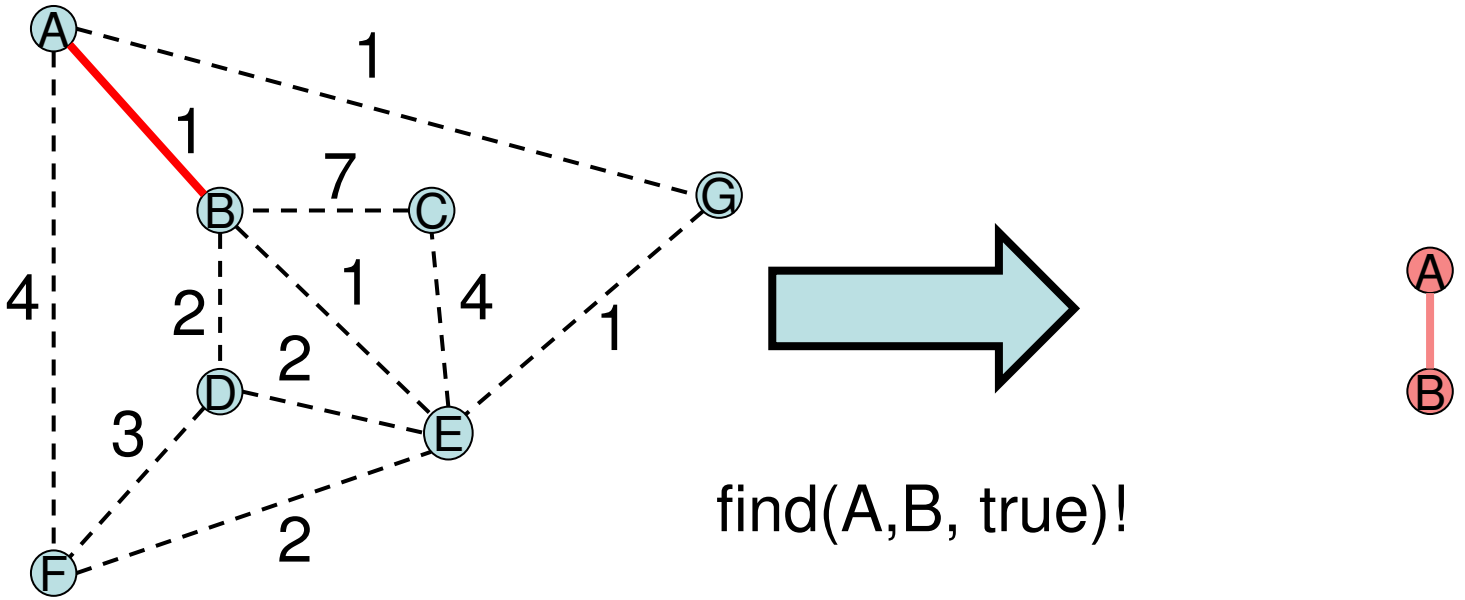
- Überlegungen:
  - Wie erhalten wir immer die „kürzeste“ Kante?
    - Wir sortieren die Kanten einfach
  - Wie prüfen wir, ob der entstehende MST nach dem Hinzufügen noch kreisfrei ist?
    - Wir verwenden unsere Union/Find Datenstruktur!

Kruskal (Code)

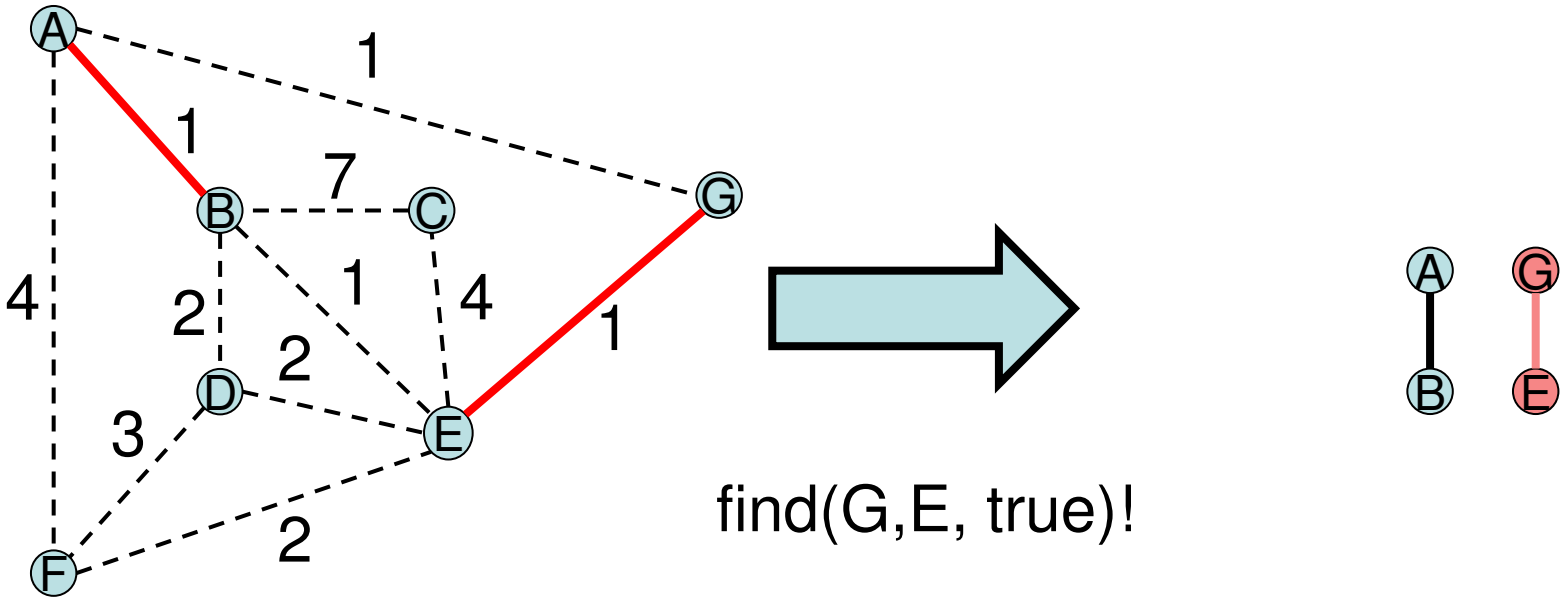
- Start: Sortierte Liste mit den Kanten
- Solange MST nicht vollständig & Liste nicht leer
  - Nimm Kante k aus der Liste
  - find(k.start, k.ende, true)
  - Bei union: nimm Kante in MST auf



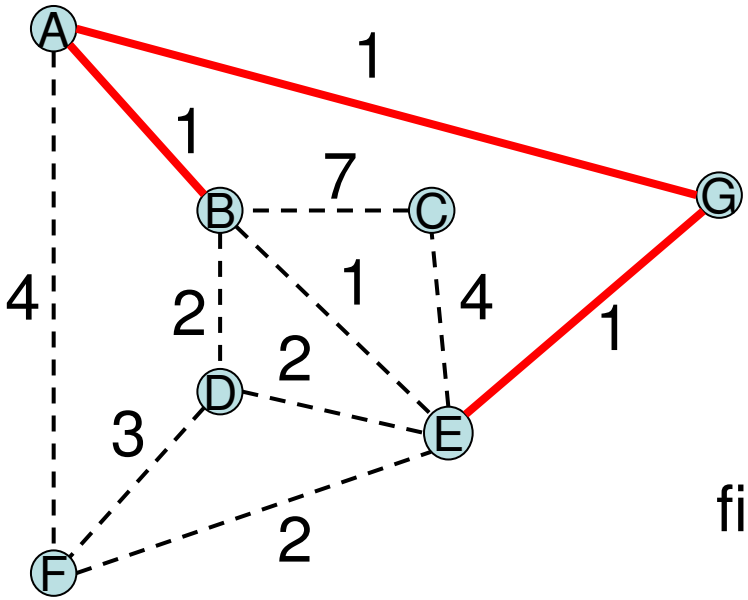
- Start: Sortierte Liste mit den Kanten
- Solange MST nicht vollständig & Liste nicht leer
  - Nimm Kante k aus der Liste
  - find(k.start, k.ende, true)
  - Bei union: nimm Kante in MST auf

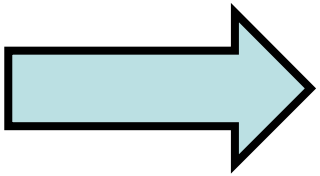


- Start: Sortierte Liste mit den Kanten
- Solange MST nicht vollständig & Liste nicht leer
  - Nimm Kante k aus der Liste
  - find(k.start, k.ende, true)
  - Bei union: nimm Kante in MST auf

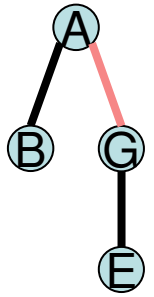


- Start: Sortierte Liste mit den Kanten
- Solange MST nicht vollständig & Liste nicht leer
  - Nimm Kante k aus der Liste
  - find(k.start, k.ende, true)
  - Bei union: nimm Kante in MST auf

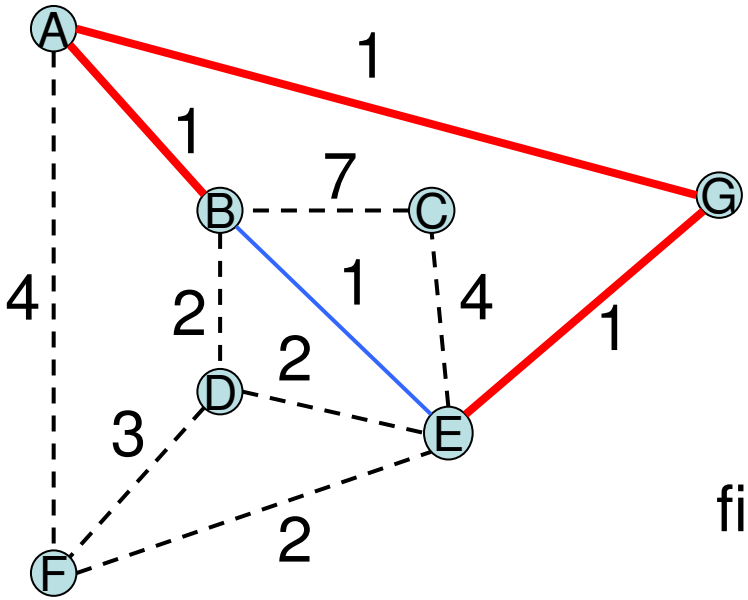


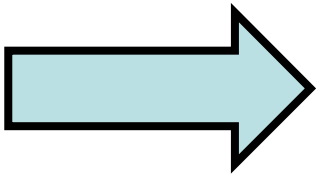


find(A,G, true)!

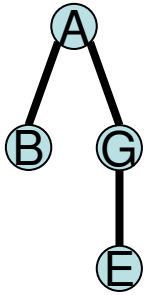


- Start: Sortierte Liste mit den Kanten
- Solange MST nicht vollständig & Liste nicht leer
  - Nimm Kante k aus der Liste
  - find(k.start, k.ende, true)
  - Bei union: nimm Kante in MST auf

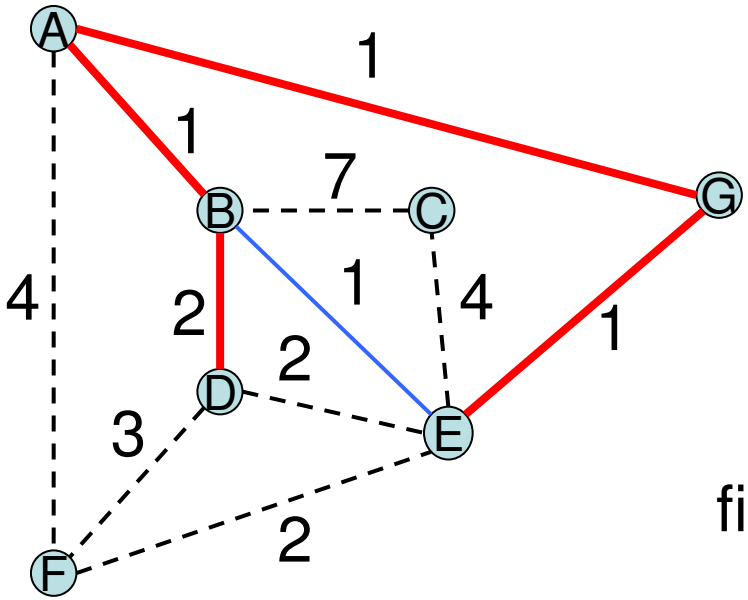




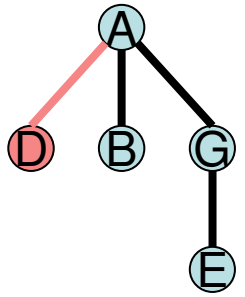
find(B,E, true)!



- Start: Sortierte Liste mit den Kanten
- Solange MST nicht vollständig & Liste nicht leer
  - Nimm Kante k aus der Liste
  - find(k.start, k.ende, true)
  - Bei union: nimm Kante in MST auf

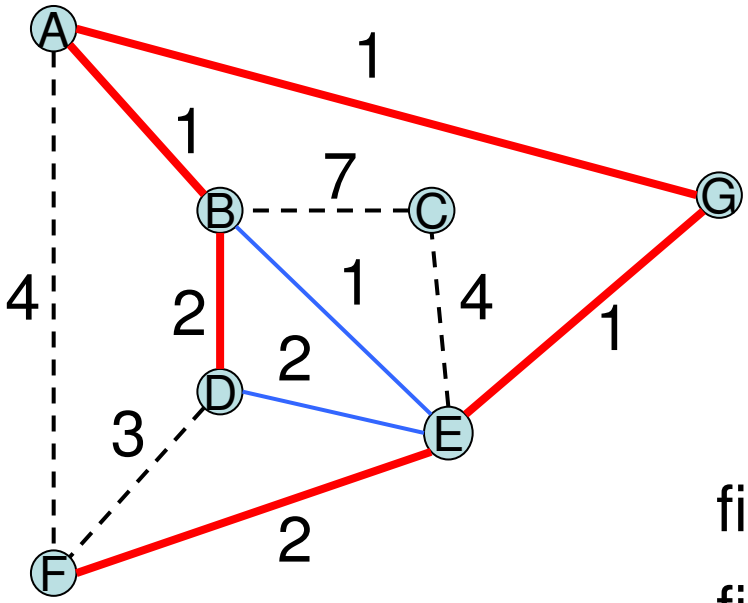


find(B,D, true)!



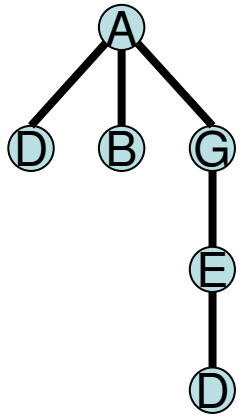
(path compression)

- Start: Sortierte Liste mit den Kanten
- Solange MST nicht vollständig & Liste nicht leer
  - Nimm Kante k aus der Liste
  - find(k.start, k.ende, true)
  - Bei union: nimm Kante in MST auf



find(D,E, true)!

find(E,F, true)!



- Start: Sortierte Liste mit den Kanten
- Solange MST nicht vollständig & Liste nicht leer
  - Nimm Kante k aus der Liste
  - find(k.start, k.ende, true)
  - Bei union: nimm Kante in MST auf

Kruskal - Komplexität

J. Kruskal
(Princeton)

- Der Algorithmus von Kruskal läuft in $O(|E| \log |E|)$
  - Erinnerung: Prim braucht $O(|E| + |V| \log |V|)$

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

Kürzeste Wege

- *Eingabe*: ger. Graph $G(V,E,\delta: E \rightarrow R)$
- *Problem*: Finde den kürzesten Weg von A nach B! (shortest path)
- *Problem*: Finde die kürzesten Wege von A zu allen anderen Knoten! (single source shortest path, sssp)
- *Problem*: Finde die kürzesten Wege zwischen allen Paaren von Knoten! (all pairs shortest path, apsp)

Kürzeste Wege

- *Eingabe*: ger. Graph G(V,E,δ: E → R)
- *Problem*: Finde den kürzesten Weg von A nach B! (shortest path)
- *Problem*: Finde die kürzesten Wege von A zu allen anderen Knoten! (single source shortest path, sssp)
- *Problem*: Finde die kürzesten Wege zwischen allen Paaren von Knoten! (all pairs shortest path, apsp)

Kürzeste Wege

- *Eingabe*: ger. Graph G(V,E,δ: E → R)
- *Problem*: Finde den kürzesten Weg von A nach B! (shortest path)
- *Problem*: Finde die kürzesten Wege von A zu allen anderen Knoten! (single source shortest path, sssp)
- *Problem*: Finde die kürzesten Wege zwischen allen Paaren von Knoten! (all pairs shortest path, apsp)

Kürzeste Wege

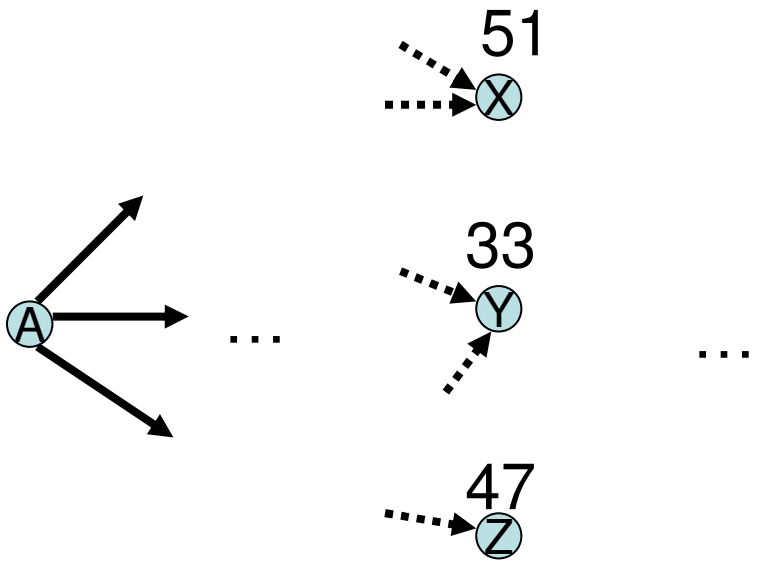
- *Eingabe*: ger. Graph G(V,E,δ: E → R)
- *Problem*: Finde den kürzesten Weg von A nach B! (shortest path)
- *Problem*: Finde die kürzesten Wege von A zu allen anderen Knoten! (single source shortest path, sssp)
- *Problem*: Finde die kürzesten Wege zwischen allen Paaren von Knoten! (all pairs shortest path, apsp)

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

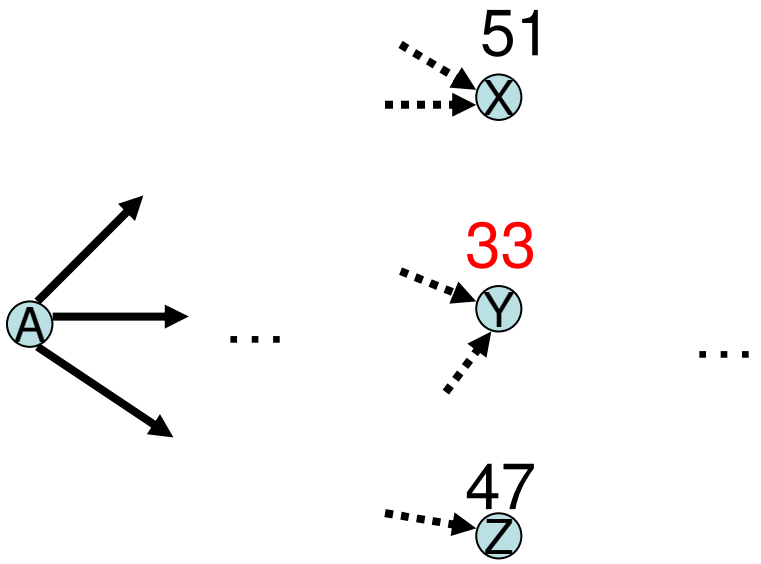
Dijkstra (sssp)

- Grundidee: Update nur da, wo bestimmt ein kürzester Weg gefunden wurde



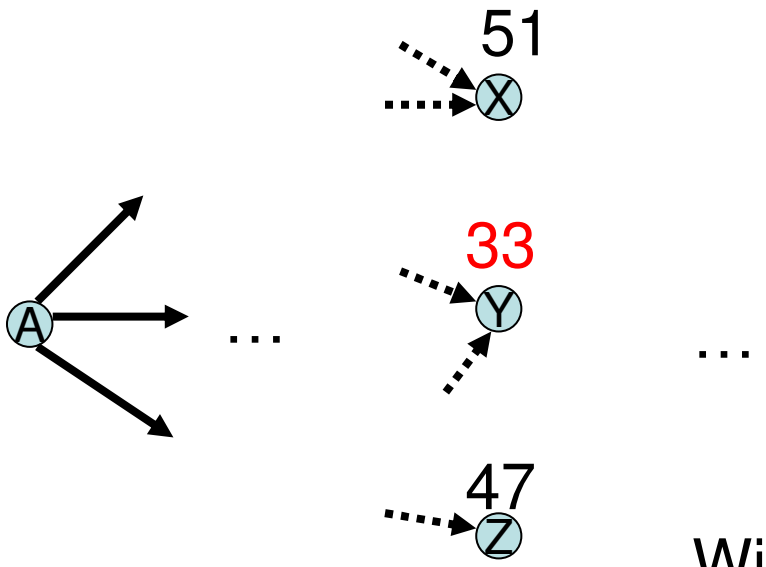
Dijkstra (sssp)

- Grundidee: Update nur da, wo bestimmt ein kürzester Weg gefunden wurde



Dijkstra (sssp)

- Grundidee: Update nur da, wo bestimmt ein kürzester Weg gefunden wurde



Wir brauchen wieder eine Priority Queue!

Dijkstra (sssp)

- Modifzierte Breitensuche, s Startknoten:

- Verwalte die Knoten in einer PriorityQueue PQ
- Abstandsarray d, initialisiere auf ∞, d[s] = 0
- Solange PQ nicht leer:
  - Entferne Knoten k mit minimalem Abstand
    - Für alle Nachbarn k‘ von k in PQ:
      - Wenn $d[k] + \delta(\{k, k'\}) < d[k'] : d[k'] = d[k] + \delta(\{k, k'\})$

Beispiel (an der Tafel)

Dijkstra - Komplexität

- $O((|E| + |V|) \log |V|)$

Dijkstra - Komplexität

- $O((|E| + |V|) \log |V|)$

- Triviale Fragen:
  - Was passiert, wenn:

Dijkstra - Komplexität

- $O((|E| + |V|) \log |V|)$

- Triviale Fragen:
  - Was passiert, wenn:
    - Der Graph nicht zusammenhängend ist?

Dijkstra - Komplexität

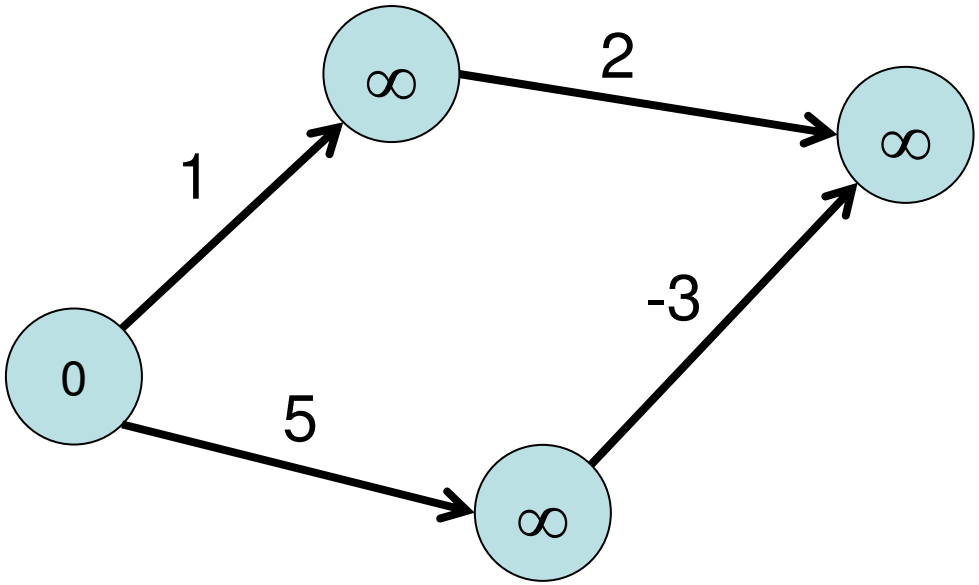
- $O((|E| + |V|) \log |V|)$

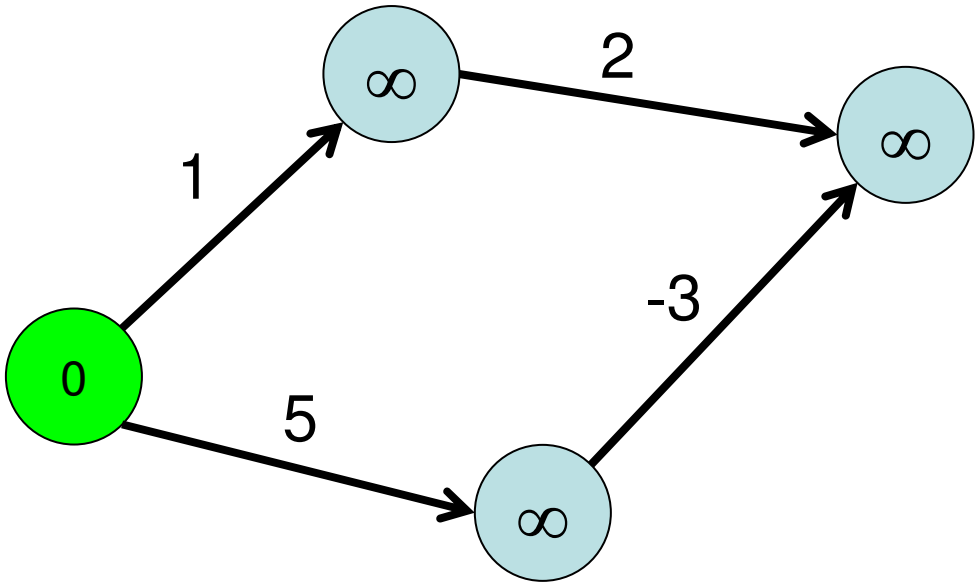
- Triviale Fragen:
  - Was passiert, wenn:
    - Der Graph nicht zusammenhängend ist?
    - Die Kantengewichte negativ sind?

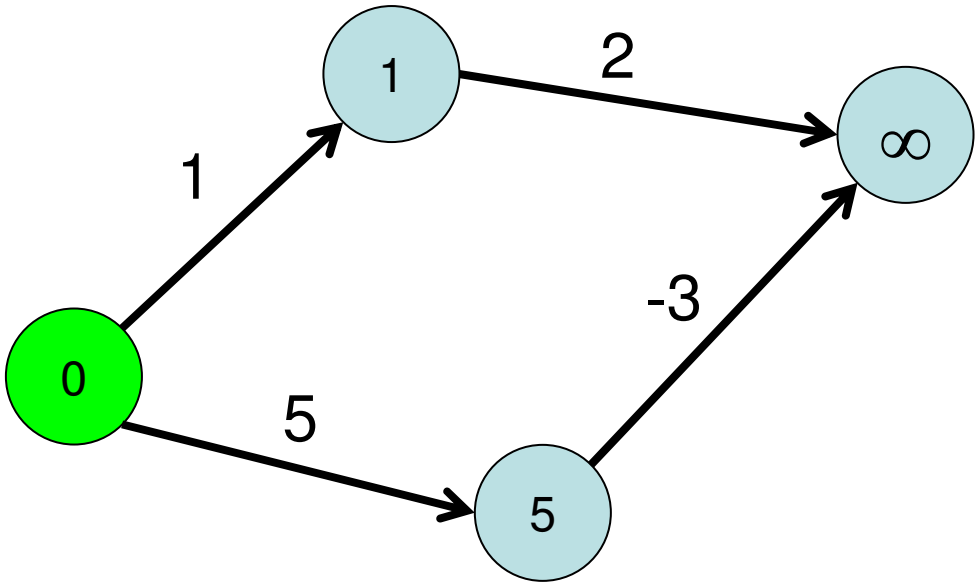
Dijkstra - Komplexität

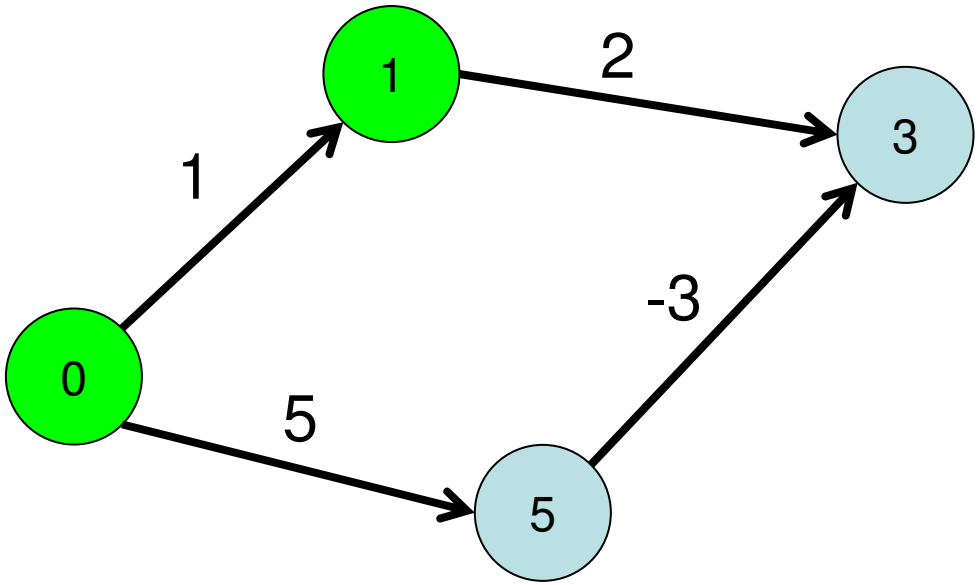
- $O((|E| + |V|) \log |V|)$

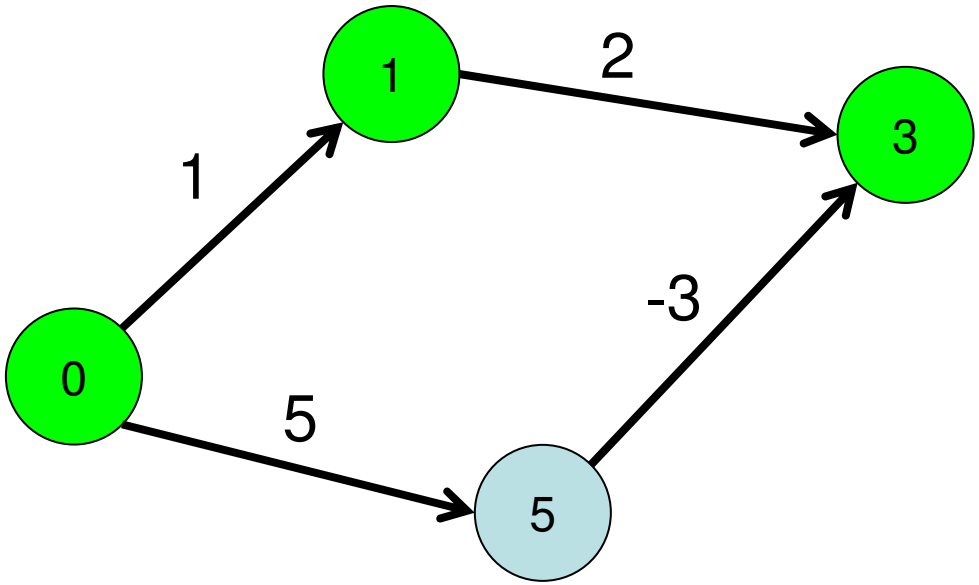
- Triviale Fragen:
  - Was passiert, wenn:
    - Der Graph nicht zusammenhängend ist?
    - Die Kantengewichte negativ sind?
      - Diese Frage scheint gar nicht so trivial zu sein…

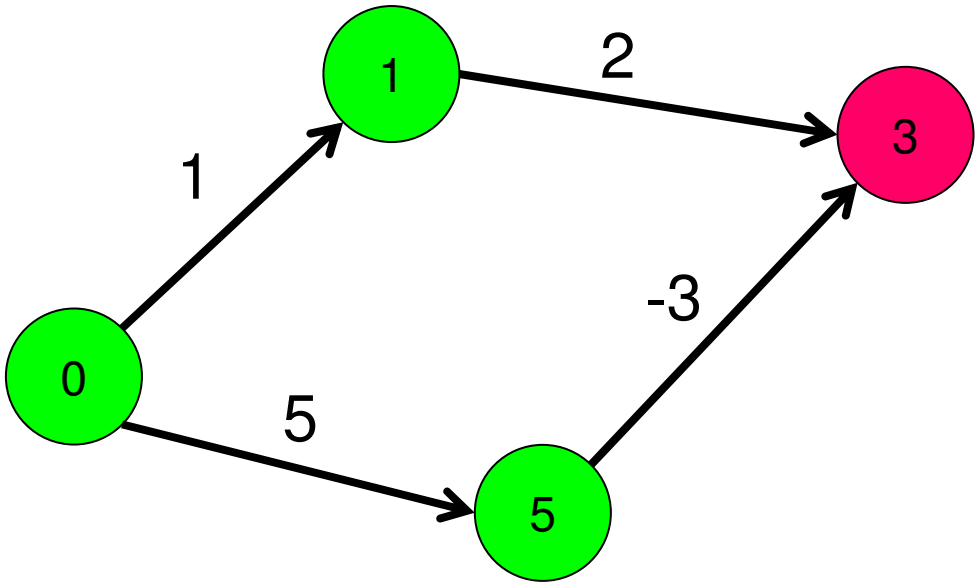

∞
2
∞
1
-3
0
5
∞

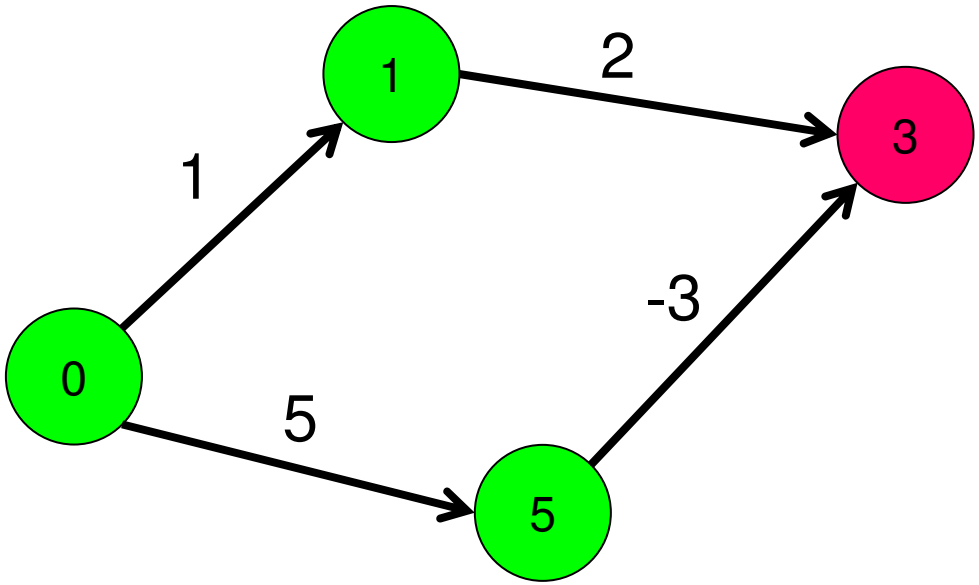

∞
2
∞
1
-3
0
5
∞


1
2
∞
1
-3
0
5
5

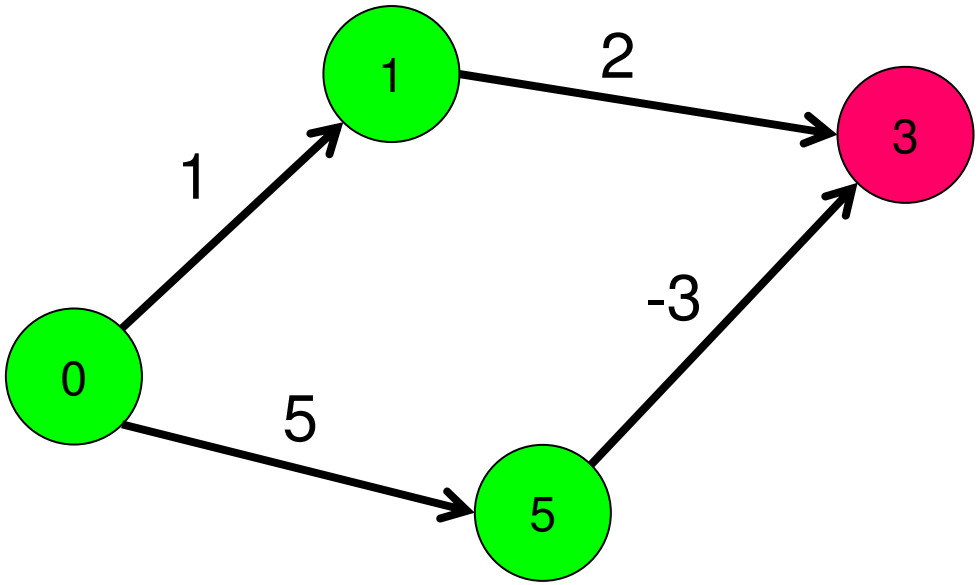

1
2
3
1
-3
0
5
5


1
2
3
1
-3
0
5
5


1
2
3
1
-3
0
5
5

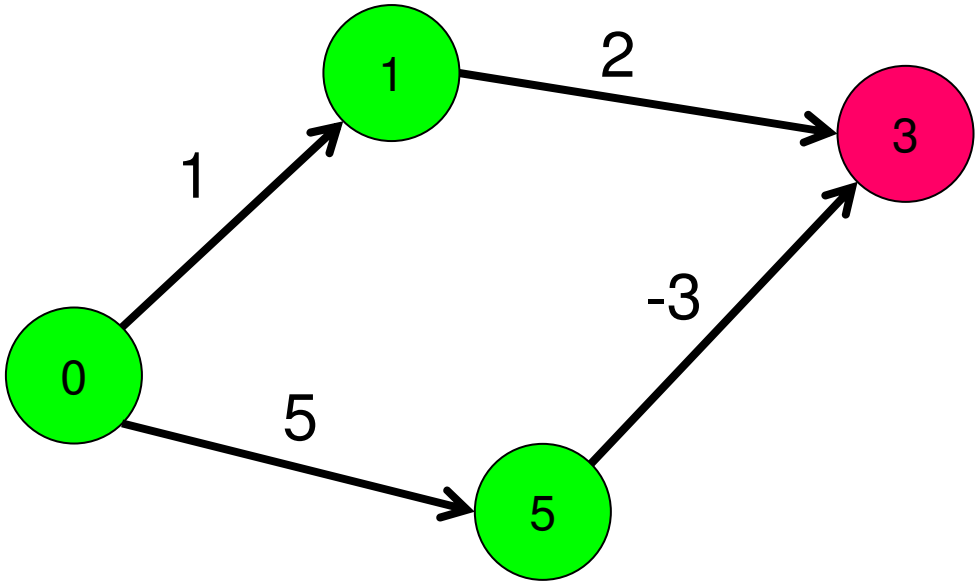


Was ist der kürzeste Weg?



Was ist der kürzeste Weg?

→ Dijkstra funktioniert bei negativen Kantengewichten nicht!



Was ist der kürzeste Weg?

→ Dijkstra funktioniert bei negativen Kantengewichten nicht!

→ Algorithmus von Bellman/Ford

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

Bellman/Ford

- Berechnet sssp in O(|V| * |E|)
- Auch bei negativen Kanten

- Bei negativen Kreisen: Ergebnis stimmt zwar nicht, Fall kann aber erkannt werden

Bellman/Ford

- Wiederhole (|V| - 1) mal
  - Für jede Kante (u,v) in E
    - wenn $d(u) + \delta(u,v) < d(v)$
      - $d(v) := d(u) + \delta(u,v)$

Bellman/Ford

- Wiederhole (|V| - 1) mal
  - Für jede Kante (u,v) in E
    - wenn $d(u) + \delta(u,v) < d(v)$
      - $d(v) := d(u) + \delta(u,v)$

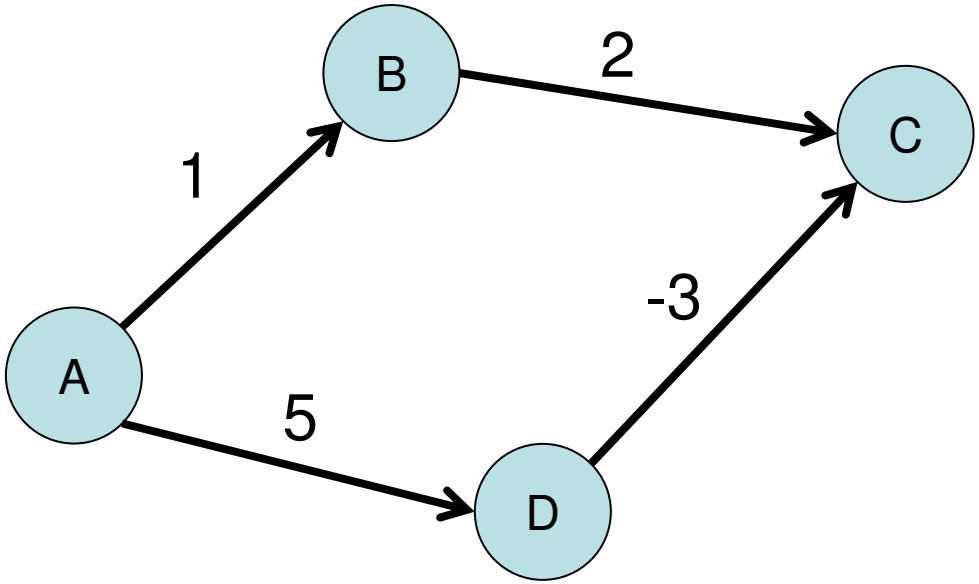
Wie können jetzt negative Kreise erkannt werden?

Bellman/Ford

- Wiederhole (|V| - 1) mal
  - Für jede Kante (u,v) in E
    - wenn $d(u) + \delta(u,v) < d(v)$
      - $d(v) := d(u) + \delta(u,v)$

Wie können jetzt negative Kreise erkannt werden?

- Für jede Kante (u,v) in E
  - wenn $d(u) + \delta(u,v) < d(v)$
    - NEGATIVER KREIS!



An der Tafel!

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall

Floyd-Warshall

- Was interessiert uns noch?

Floyd-Warshall

- Was interessiert uns noch?
  - Die kürzesten Wege aller Paare

Floyd-Warshall

- Was interessiert uns noch?
  - Die kürzesten Wege aller Paare
  - Die transitive Hülle

Floyd-Warshall

- Was interessiert uns noch?
  - Die kürzesten Wege aller Paare
  - Die transitive Hülle

- Gewichtsmatrix betrachten!

Floyd-Algorithmus für apsp

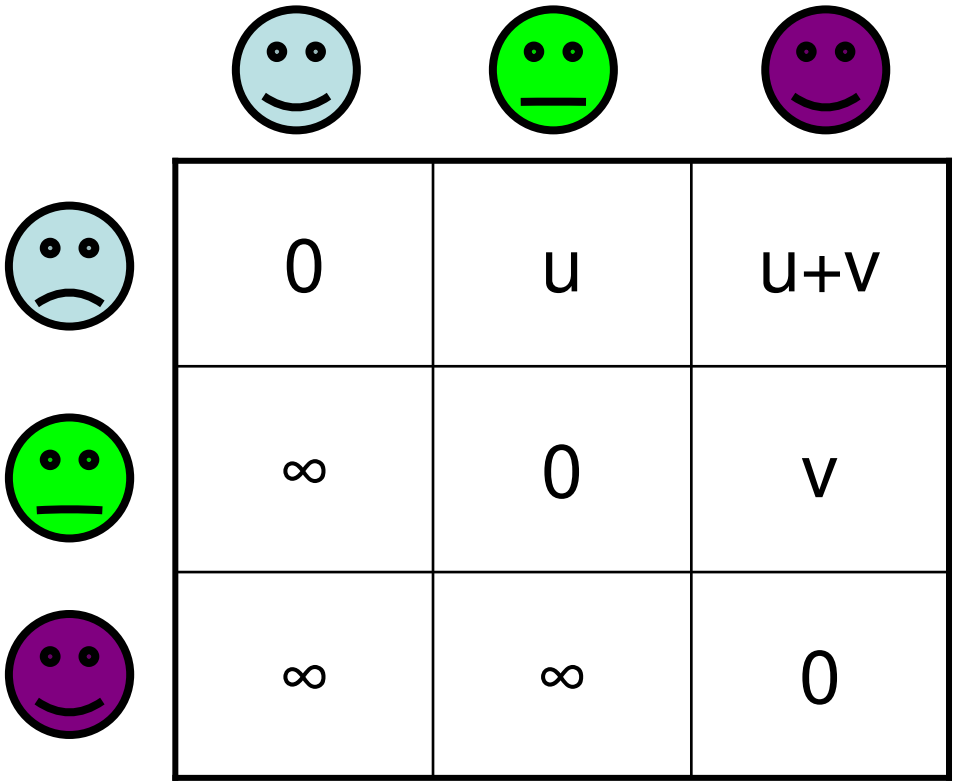
- Dynamische Programmierung
  - Was ist unsere Substruktur?

Floyd-Algorithmus für apsp

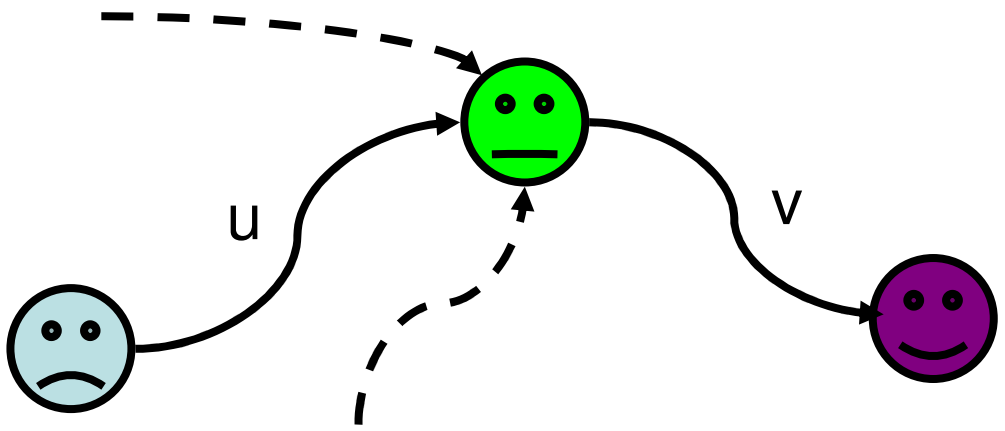
- Dynamische Programmierung
  - Was ist unsere Substruktur?
  - Optimale Teilpfade, denn: Geht der kürzeste Weg von *u* nach *v* durch *w*, dann sind die enthaltenen Teilpfade von *u* nach *w* und von *w* nach *v* schon minimal.

Floyd-Algorithmus für apsp

- Dynamische Programmierung
  - Was ist unsere Substruktur?
  - Optimale Teilpfade, denn: Geht der kürzeste Weg von *u* nach *v* durch *w*, dann sind die enthaltenen Teilpfade von *u* nach *w* und von *w* nach *v* schon minimal.
  - Betrachte die Gewichtsmatrix des Graphen!



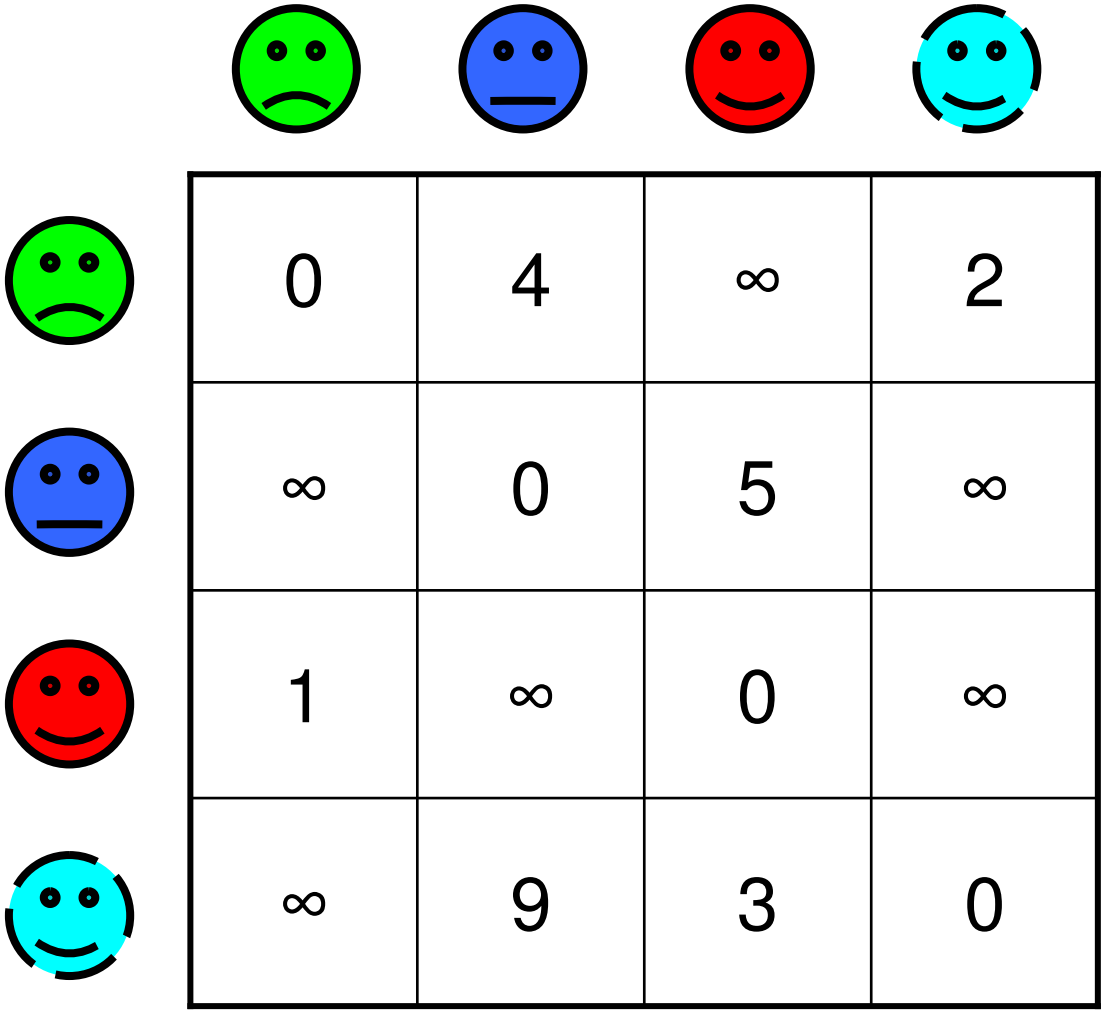
0	u	u+v
∞	0	v
∞	∞	0



Algorithmus von Floyd

- informell:

  - repeat für alle Knoten k
    - repeat für alle Paare von Knoten i,j
      - Ist der Weg von i nach k und von k nach j kürzer als der bisherige Weg von i nach j?
        - Falls ja: update distanz



0	4	∞	2
∞	0	5	∞
1	∞	0	∞
∞	9	3	0

An der Tafel!

Algorithmus von Floyd

- Berechnet apsp in $O(|V|^3)$
- Auch mit negativen Kanten
- Negative Kreise erkennen: negative Zahlen an den Diagonalen

- Wie kann man den Algorithmus so abändern, dass die transitive Hülle berechnet wird?

Graphalgorithmen II

- Union/Find
- Minimale Spannbäume
  - Prim
  - Kruskal
- Kürzeste Wege
  - Dijkstra
  - Bellman/Ford
  - Floyd-Warshall
- Aufgaben!

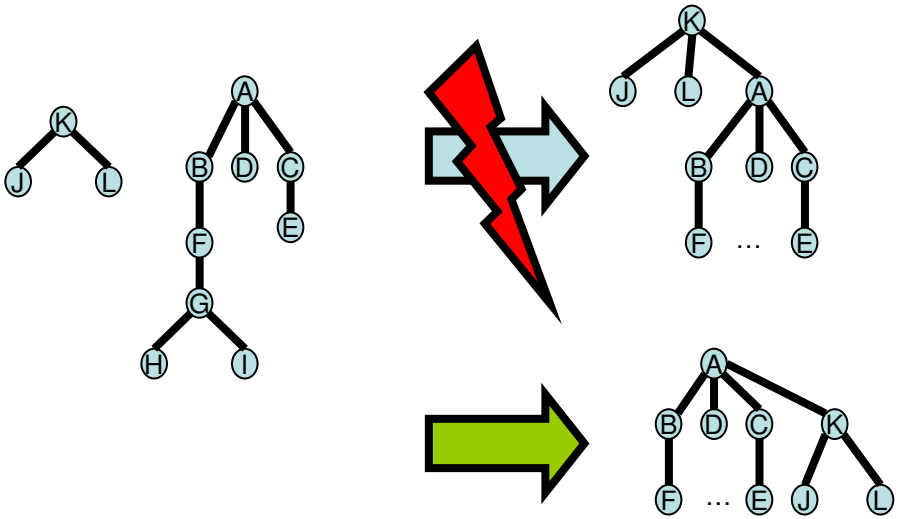
Zum Aufwärmen

- Funktioniert Union/Find auch bei gerichteten Graphen?
- Welchen Algorithmus kann man für „single destination shortest path“ verwenden?
- Ist ein MST immer sssp-Spannbaum eines Knotens?
- Geben Kruskal und Prim immer das gleiche Ergebnis?

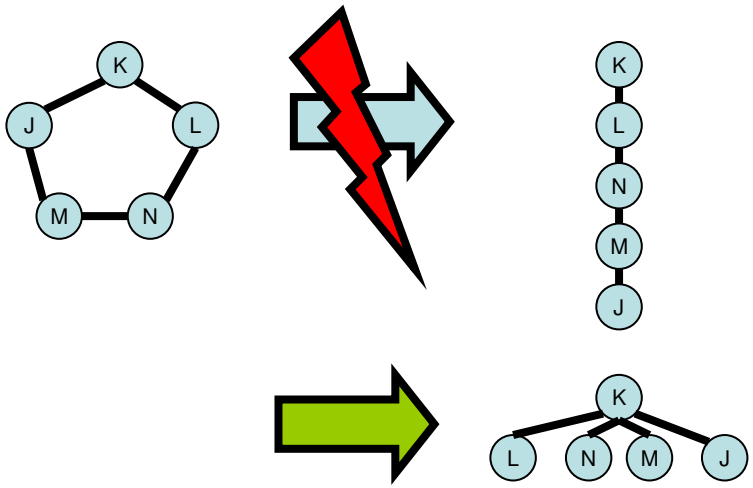
Union/Find mit Heuristiken

- Zur Erinnerung:

Balancing:



Path Compression:



Union/Find mit Heuristiken

```
int find(int x, int y, bool union) {
  int i = x, j = y;
  while (dad[i] > 0) i = dad[i];
  while (dad[j] > 0) j = dad[j];
  if(union && (i != j)) dad[j] = i;
  return (i != j);
}
```

10583 – Ubiquitous Religions

There are so many different religions in the world today that it is difficult to keep track of them all. You are interested in finding out how many different religions students in your university believe in.

You know that there are n students in your university ($0 < n \leq 50000$). It is infeasible for you to ask every student their religious beliefs. Furthermore, many students are not comfortable expressing their beliefs. One way to avoid these problems is to ask m ($0 \leq m \leq n(n-1)/2$) pairs of students and ask them whether they believe in the same religion (e.g. they may know if they both attend the same church). From this data, you may not know what each person believes in, but you can get an idea of the upper bound of how many different religions can be possibly represented on campus. You may assume that each student subscribes to at most one religion.

For each test case, print on a single line the case number (starting with 1) followed by the maximum number of different religions that the students in the university believe in.

10583 – Ubiquitious Religions

- Gegeben: Informationen der Form a ~ b
  - a hat die selbe Religion wie b
- Wie viele verschiedene Religionen?
- Achtung: bis zu ($50000^2/2$) Kanten!

10987 - Antifloyd

You have been hired as a systems administrator for a large company. The company head office has **n** computers connected by a network of **m** cables. Each cable connects two different computers, and there is at most one cable connecting any given pair of computers. Each cable has a latency, measured in micro-seconds, that determines how long it takes for a message to travel along that cable. The network protocol is set up in a smart way, so that when sending a message from computer A to computer B, the message will travel along the path that has the smallest total latency, so that it arrives at B as soon as possible. The cables are bi-directional and have the same latency in both directions.

As your first order of business, you need to determine which computers are connected to each other, and what the latency is along each of the **m** cables. You soon discover that this is a difficult task because the building has many floors, and the cables are hidden inside walls. So here is what you decide to do. You will send a message from every computer A to every other computer B and measure the latency. This will give you **n**(**n**-1)/2 measurements. From this data, you will determine which computers are connected by cables, and what the latency along each cable is. You would like your model to be simple, so you want to use as few cables as possible.

10987 - Antifloyd

- „Shortest-Path-Matrix“ ist bekannt

- Wie kann man daraus das ursprügnliche Netzwerk berechnen? (minimale Anzahl an Kanten)

- Achtung: Kann auch unmöglich sein! Wann?

Andere Aufgaben zum Trainieren

- 10600 - *ACM Contest & Blackout*
- 10462 - *Is There A Second Way Left?*
- 10801 - *Lift Hopping*
- 10986 - *Sending eMail*
- 10805 - *Cockroach Escape Networks*
- 3505 - *Buy or Build*
- 10158 - *War*
- 10537 - *Toll! Revisited*
- 10850 - *The Gossipy Gossipers Gossip Gossips*
- 10381 - *The Rock*
- 10224 - *Return of the Jedi*
- 10857 - *Easter Eggs*

Quellen

- Sedgewick: Algorithms in C++
- Skiena: Algorithm Design Manual
- Wikipedia
- „Hallo-Welt“ Seminare FAU
- UVA & Algorithmist
- Ubahnpläne: MVV-Homepage