Algorithmen und Datenstrukturen
Graphen - Einführung

Matthias Teschner
Graphische Datenverarbeitung
Institut für Informatik
Universität Freiburg

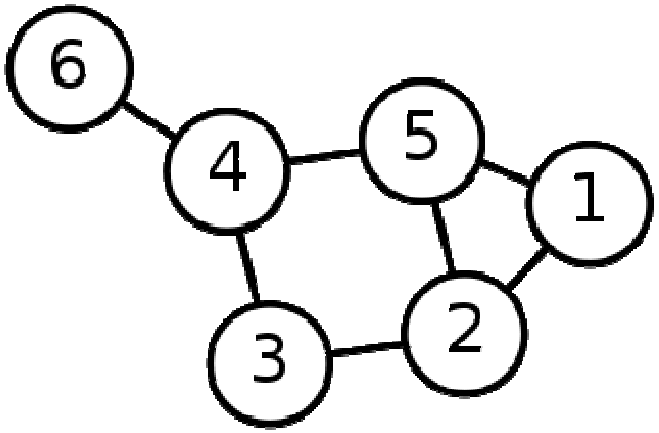
SS 12

Überblick

- Definition / Eigenschaften
- Anwendungen
- Repräsentation
- Traversierung
- Kürzester Weg

Definition Graph

- Graph G ist ein Paar zweier Mengen G = (V, E) mit
  - V = $\{v_0, ..., v_{n-1}\}$ Menge von n unterschiedlichen Knoten (Vertices / Vertex)
  - E = $\{e_0, ..., e_{m-1}\}$ Menge von m Kanten mit $e_i = (v_j, v_k)$ (Edge, Arc)



V = {1, 2, 3, 4, 5, 6}

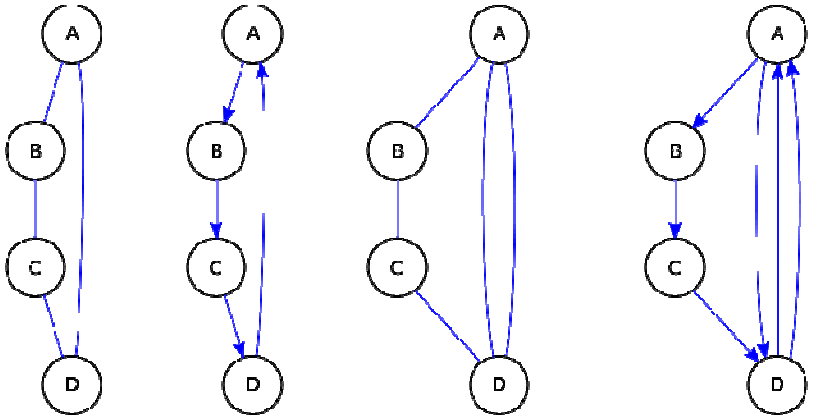
E = { (1, 2), (1, 5), (2, 3), (2, 5), (3, 4), (4, 5), (4, 6) }

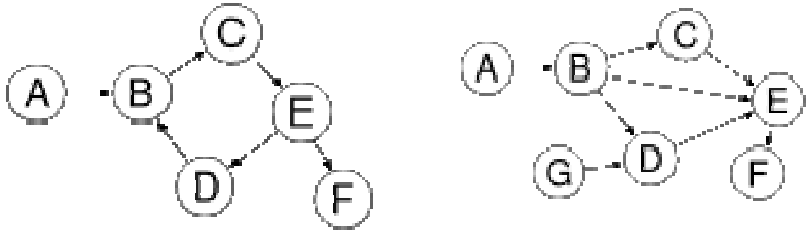
Wikipedia: Graph (Graphentheorie)

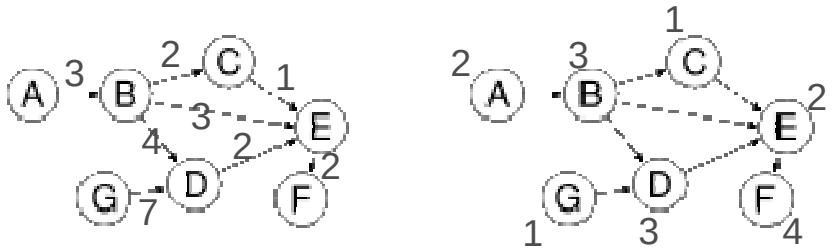
Spezielle Graphen

Wikipedia: Graph (Graphentheorie)

- gerichtet, ungerichtet
- mit / ohne Mehrfachkanten
- zyklisch / azyklisch
- gewichtet / ungewichtet
  - kantengewichtet, knotengewichtet







Nachbarschaft

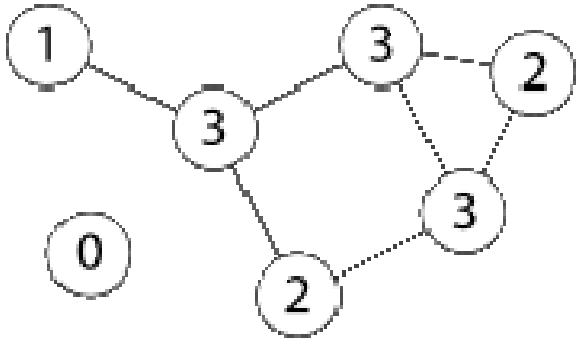
- v_i und v_k sind in einem ungerichteten Graphen benachbart, wenn $(v_i, v_k) \in E$ oder $(v_k, v_i) \in E$
- v_i ist Vorgänger von v_k in einem gerichteten Graphen, wenn $(v_i, v_k) \in E$
- zwei Kanten sind im ungerichteten Graphen benachbart, wenn sie einen gemeinsamen Knoten enthalten
- die Menge aller Nachbarn von v_i wird als Nachbarschaft von v_i bezeichnet
- analog Vorgängermenge (Eingangsmenge) bzw. Nachfolgermenge (Ausgangsmenge) im gerichteten Graphen

Grad (Valenz)

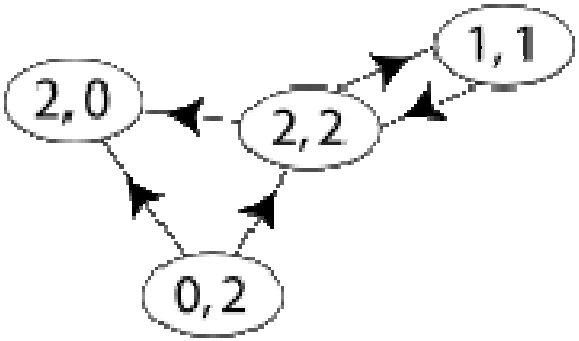
- ungerichteter Graph
  - Grad (v) entspricht Anzahl der Kanten mit $(v, w) \in E$ oder $(w, v) \in E$
  - in einem Graph ohne Mehrfachkanten entspricht Grad (v) der Zahl der Nachbarn von v
- gerichteter Graph
  - analog zum ungerichteten Graph, allerdings Unterscheidung in Eingangsgrad und Ausgangsgrad
  - in einem Graph ohne Mehrfachkanten:
    - Eingangsgrad von v entspricht der Zahl der Knoten, für die v Nachfolger ist
    - Ausgangsgrad von v entspricht der Zahl der Nachfolger von v

Grad (Valenz)

Wikipedia: Nachbarschaft und Grad in Graphen



Knotengrade in einem ungerichteten Graph



Eingangs- und Ausgangsgrade
in einem gerichteten Graph

Überblick

- Definition / Eigenschaften
- Anwendungen
- Repräsentation
- Traversierung
- Kürzester Weg

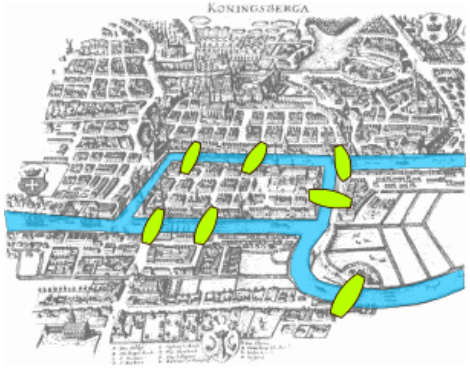
Anwendung

- Routenplanung
  - kürzester Weg von A nach B
  - kürzeste Rundreise durch A, B, C, ..., A
  - Existenz eines Rundwegs (jeder Ort wird genau einmal besucht)
- Planung von Fahrplänen
- Wieviel Wasser verkraftet eine Kanalisation?
- Planungsprobleme
  - A vor B, B vor D, A vor E, D vor E $\rightarrow$ A, B, C, D, E
- Verarbeitung triangulierter Oberflächenrepräsentationen

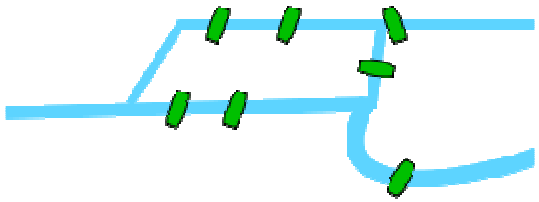
Königsberger Brückenproblem

Wikipedia: Königsberger Brückenproblem

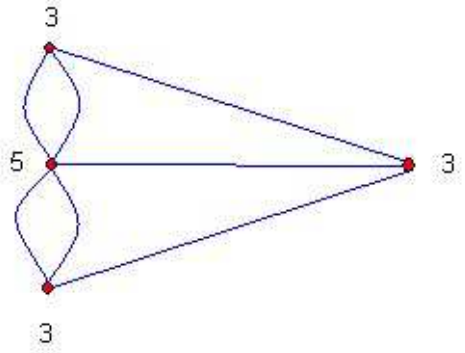
- Existiert ein Weg, bei dem jede Brücke genau einmal überquert wird?



Karte von Königsberg vor 1945



schematische Darstellung

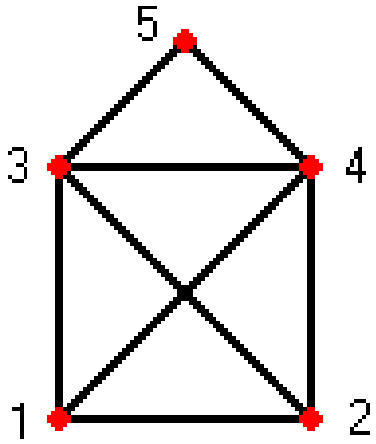


Graphdarstellung mit Angabe der Grade / Valenzen (ungerichtet, ungewichtet, zyklisch, Mehrfachkanten)

- Eulerweg: besuche jede Brücke einmal
- Eulerkreis: Eulerweg, der am Ausgangspunkt endet
- beides existiert im Beispiel nicht
  - Eulerweg: kein oder zwei Knoten von ungeradem Grad
  - Eulerkreis: alle Knoten von geradem Grad

Haus vom Nikolaus

Wikipedia: Eulerkreisproblem



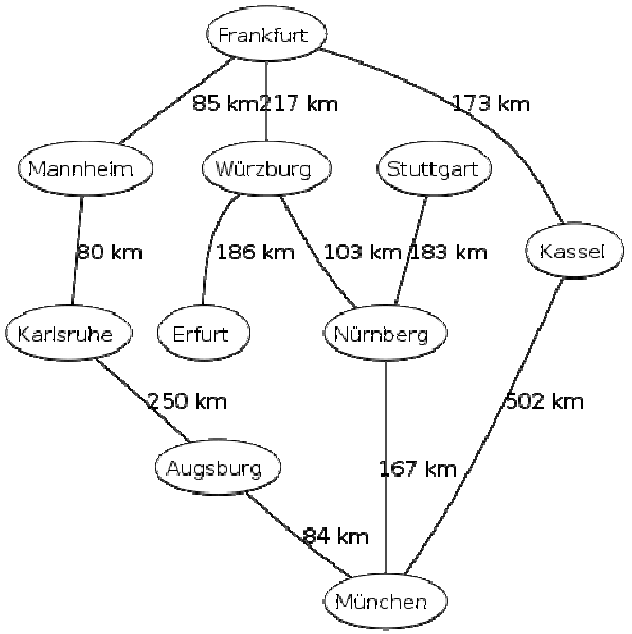
- zwei Knoten von ungeradem Grad,
  alle weiteren Knoten von geradem Grad
  - Eulerweg existiert
  - Eulerkreis existiert nicht
  - Knoten von ungeradem Grad müssen Start- und Endpunkt sein

Routenplanung

Wikipedia: Dijkstra-Algorithmus

- Graphdarstellung des Straßennetzes
- Ermittlung der kürzesten Distanz zwischen zwei Orten

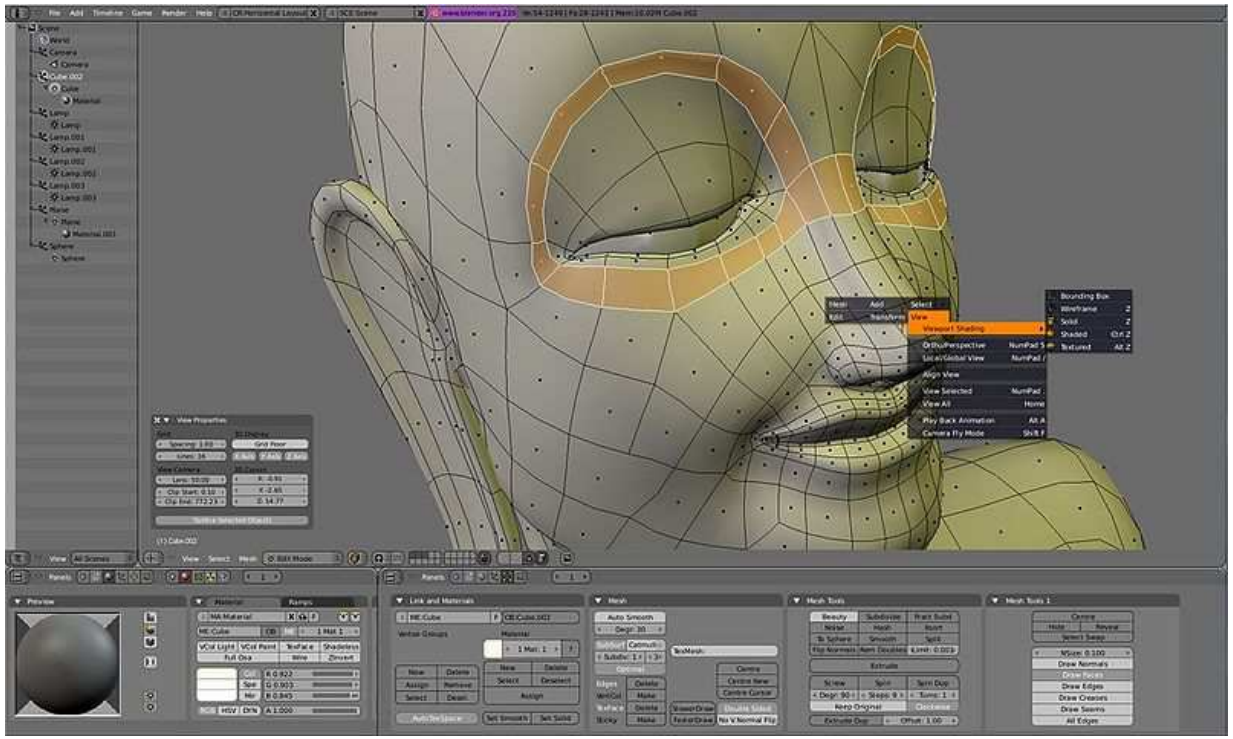
Kartendarstellung eines Straßennetzes



Graphdarstellung eines Straßennetzes

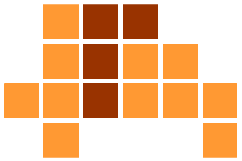
Geometrische Modellierung

- Bearbeitung / Modellierung von polygonalen Netzen
  - Operationen benötigen als Eingabe polygonale Netze mit bestimmten Eigenschaften

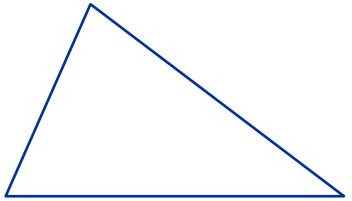


Wikipedia:
Computergrafik
(Blender Screenshot)

Polygon / Polygonales Netz

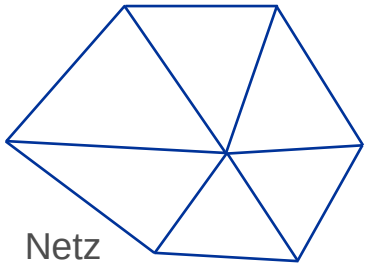


- Polygon:
  - geometrischer Graph Q = (V, E) mit $v_i \in R^d$ $d \geq 2$ und $E = \{ (v_0, v_1), (v_1, v_2), \ldots, (v_{n-2}, v_{n-1}) \}$
  - planar, wenn alle Knoten in einer Ebene liegen
  - geschlossen, wenn $v_0 = v_{n-1}$
  - einfach, wenn Kanten sich nicht schneiden

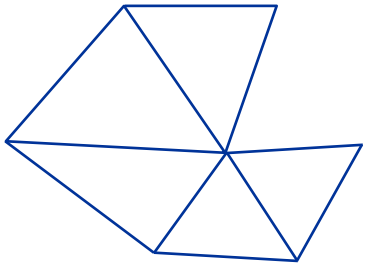


geschlossen,
einfach, planar

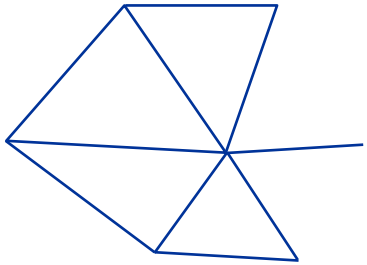
- Polygonales Netz
  - endliche Menge geschlossener, einfacher Polygone
  - Schnitt eingeschlossener Regionen zweier Polygone ist leer
  - Schnitt zweier Polygone ist leer, ein Knoten oder eine Kante
  - Jede Kante ist Teil von mindestens einem Polygon



Netz



Netz



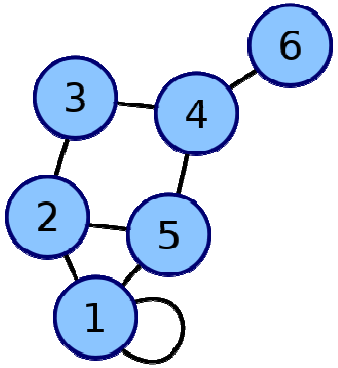
kein Netz

Überblick

- Definition / Eigenschaften
- Anwendungen
- Repräsentation
- Traversierung
- Kürzester Weg

Adjazenzmatrizen

- zur Repräsentation / Implementierung von Graphen
- Matrix A der Größe n x n mit n = |V| Zahl der Knoten
- $a_{ik} = 1$, falls $(v_i, v_k) \in E$ oder $(v_k, v_i) \in E$, sonst $a_{ik} = 0$



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Wikipedia: Repräsentation von Graphen

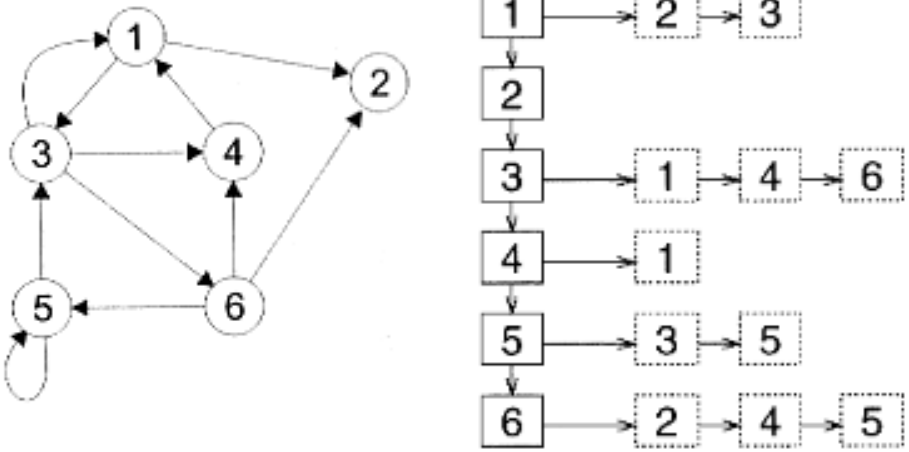
- symmetrisch für ungerichtete Graphen
- in gerichteten Graphen können eingehende Kanten (-1) und ausgehende Kanten (1) unterschieden werden

Adjazenzmatrizen

- vergleichsweise ineffizient
- Speicherbedarf von $\Theta (|V|^2)$
  - unabhängig von der Zahl der Kanten
  - wegen Initialisierung und Traversierung aller Einträge der Matrix liegt die Laufzeit von Operationen üblicherweise in $\Omega (|V|^2)$
- alternative Repräsentation sinnvoll, wenn durchschnittliche Zahl der Nachbarn (bzw. Nachfolger und Vorgänger) klein gegenüber der Gesamtzahl der Knoten ist
  - dünner Graph (im Gegensatz zum dichten Graph)
  - in dem Fall ist die Matrix dünn besetzt

Adjanzenzlisten

- Speichern einer linearen Liste von Kanten für jeden Knoten
- lineares Feld oder Liste der Größe |V| (Zahl der Knoten) mit Zeigern bzw. Referenzen auf lineare Listen, in denen Nachfolgeknoten von Kanten gespeichert sind

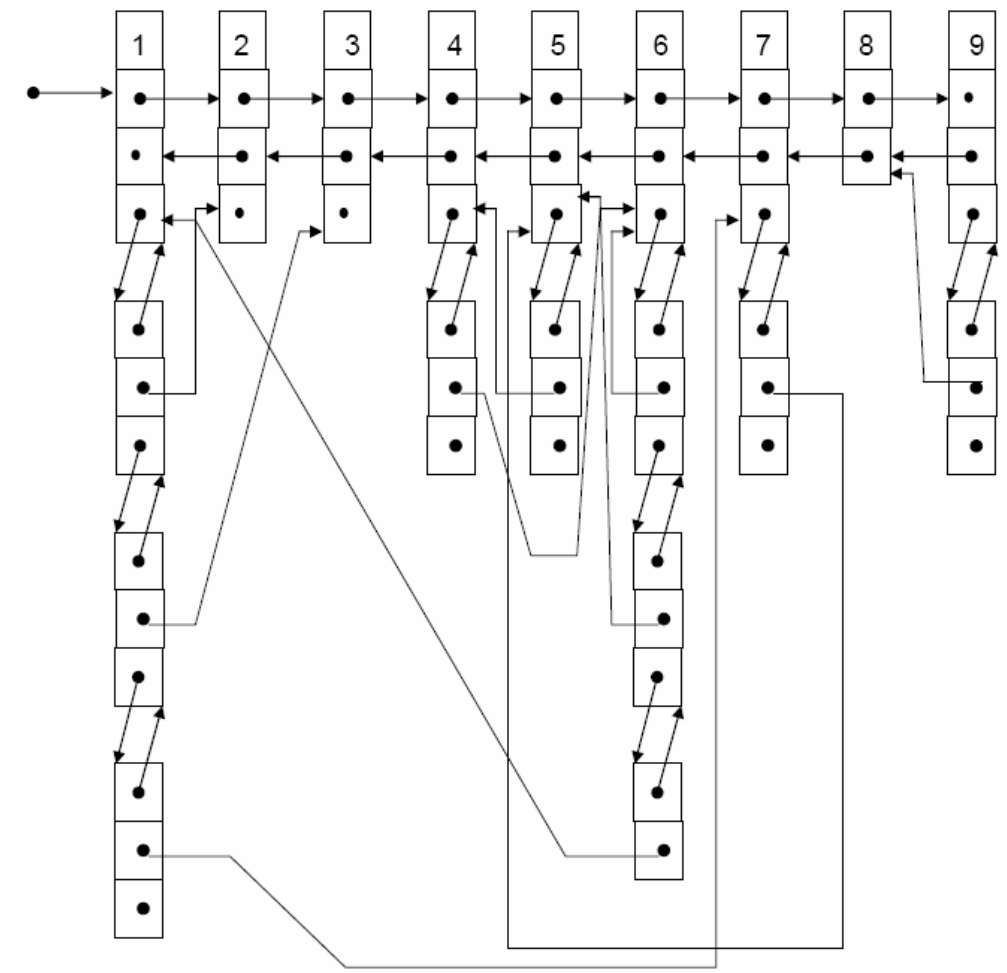


Speicherbedarf
Θ (|V| + |E|)

Saake, Sattler, "Algorithmen und Datenstrukturen"

Adjazenzlisten

- doppelt verkettet
- Kanteneinträge enthalten lediglich Referenzen auf Nachfolgeknoten

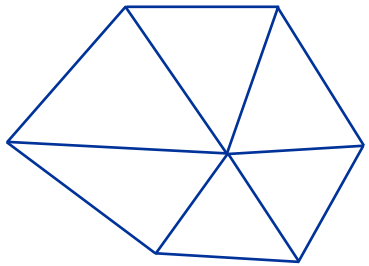


Diskussion

- Adjazenzmatrizen und Adjazenzlisten sind mögliche Datenstrukturen zur Repräsentationen von Graphen (dynamische Mengen)
- Operationen auf Graphen sind unterschiedlich effizient für verschiedene Repräsentationen
  - Einfügen / Löschen von Knoten / Kanten
  - Suchen von Knoten / Kanten (Traversieren)
  - Suchen von Wegen
  - ...
- Wahl der Datenstruktur hängt von den Operationen einer Anwendung ab

Diskussion - Polygonnetz

- mögliche Repräsentationen
  - Knotenliste
    (Liste von Knoten und Polygonen)
  - Kantenliste
    (Liste von Knoten, Kanten, Polygonen)
  - winged edge
    (zusätzliches Speichern von Nachbarkanten)
- zu unterstützende Suchanfragen
  - Kante → Endpunkte
  - Kante → Polygone
  - Kante → Nachbarkanten
  - Knoten → Kanten
  - Knoten → Polygone
  - Polygon → Knoten, Kanten, Nachbarpolygone



Polygonnetz

Überblick

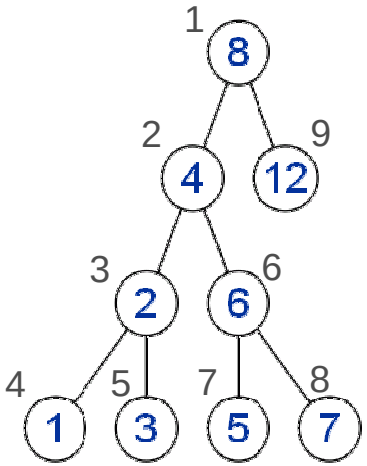
- Definition / Eigenschaften
- Anwendungen
- Repräsentation
- Traversierung
- Kürzester Weg

Einführung

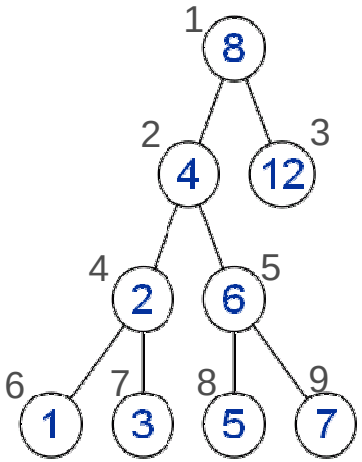
- manche Operationen (z. B. Suchen eines Knotens) müssen alle Knoten eines Graphen besuchen / betrachten
- vollständige Traversierung des Graphen
- unterschiedliche Strategien je nach Anwendung
- Breitensuche (BFS breadth-first search)
  - besuche zunächst alle Nachbarn / Nachfolger eines Knotens und erst danach die Nachbarn / Nachfolger der Nachbarn / Nachfolger
- Tiefensuche (DFS depth-first search)
  - besuche erst die Nachfolger eines Nachfolgers des aktuellen Knotens, dann weitere Nachfolger des aktuellen Knotens
- weitere Strategien, wenn nicht zwangsläufig der gesamte Graph durchlaufen werden muss

Spezialfall Baum

- Breitensuche
  - besuche Geschwister eines Knotens vor seinen Kindern
- Tiefensuche
  - besuche Kinder eines Knotens vor seinen Geschwistern



Tiefensuche / Preorder



Breitensuche

Traversieren von Graphen

- im Gegensatz zu Bäumen können Zyklen auftreten
  - markieren von bereits besuchten Knoten zur Vermeidung von Schleifen
- Hilfs-Datenstruktur zur Verwaltung aller noch zu besuchenden Nachfolger eines Knotens
  - Stapel für DFS
  - Schlange für BFS

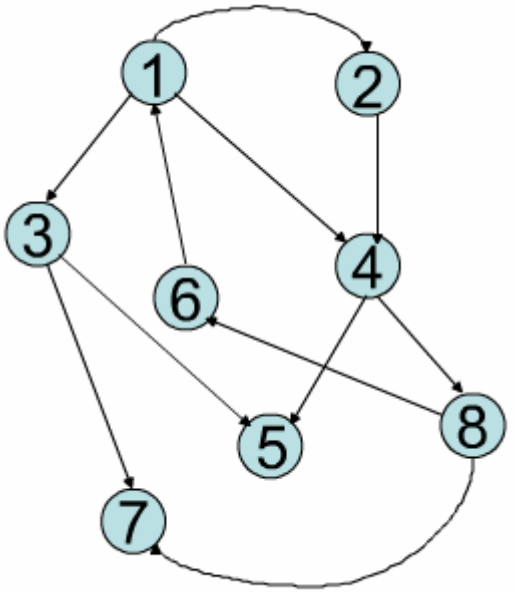
Tiefensuche

- rekursive Implementierung
  - Initialisierung aller Knoten v mit v.besucht = false

- ```
  Tiefensuche (v) {
    v.besucht = true;
    foreach (Nachfolger w von v) do
      if (w.besucht == false)
        Tiefensuche(w);
  }
  ```
  ```

Tiefensuche

- Implementierung mit einem Stapel S
  - Initialisierung aller Knoten v mit v.besucht = false

- ```
  Tiefensuche (v) {
    S.push(v);
    while (!S.empty) do {
      u = S.pop();
      if (u.besucht == false) {
        u.besucht = true;
        foreach (Nachfolger w von u) do
          S.push(w);
      }
    }
  }
  ```
  ```

Beispiel

- push(1), S = {1}
- 1 = pop(), **1 besucht**, push(2), push(3), push(4)
  S = {2, 3, 4}
- 4 = pop(), **4 besucht**, push(5), push(8)
  S = {2, 3, 5, 8}
- 8 = pop(), **8 besucht**, push(6), push(7)
  S = {2, 3, 5, 6, 7}
- 7 = pop(), **7 besucht** // kein Nachfolger, tue nichts
  S = {2, 3, 5, 6}
- 6 = pop(), **6 besucht**, push(1)
  S = {2, 3, 5, 1}
- 1 = pop(), 1 bereits besucht // tue nichts
  S = {2, 3, 5}
- 5 = pop(), **5 besucht**
  S = {2, 3}
- ...



Ottmann, Universität Freiburg,
Informatik II, SS 2008

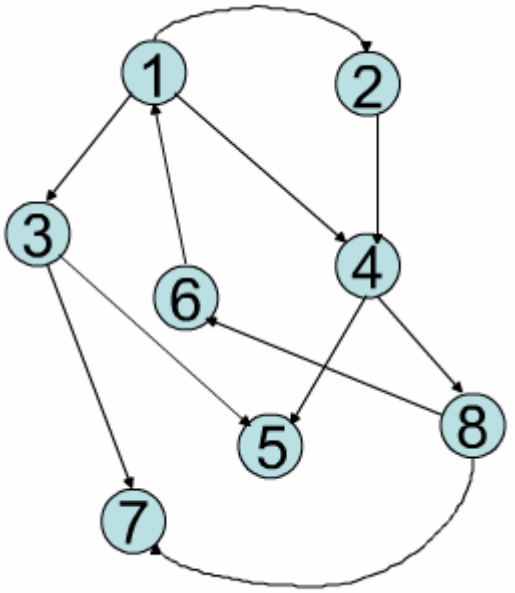
Breitensuche

- Implementierung mit einer Schlange Q
  - Initialisierung aller Knoten v mit v.besucht = false

```
Breitensuche (v) {
  v.besucht = true;
  Q.enqueue(v);
  while (!Q.empty) do {
    u = Q.dequeue();
    foreach (Nachfolger w von u) do
      if (w.besucht == false) {
        w.besucht = true;
        Q.enqueue(w);
      }
  }
}
```

Beispiel

- enq(1), **1 besucht**, Q = {1}
- 1 = deq (), enq(2), **2 besucht**, enq(3), **3 besucht**, enq(4), **4 besucht**, Q = {2, 3, 4}
- 2 = deq() // 4 bereits besucht, tue nichts
  Q = {3, 4}
- 3 = deq(), enq(5), **5 besucht**, enq(7), **7 besucht**
  Q = {4, 5, 7}
- 4 = deq(), enq(8), **8 besucht** // 5 bereits besucht
  Q = {5, 7, 8}
- 5 = deq() // kein Nachfolger
  Q = {7, 8}
- 7 = deq() // kein Nachfolger
  Q = {8}
- 8 = deq(), enq(6), **6 besucht** // 7 bereits besucht
  Q = {6}
- 6 = deq() // tue nichts
- fertig



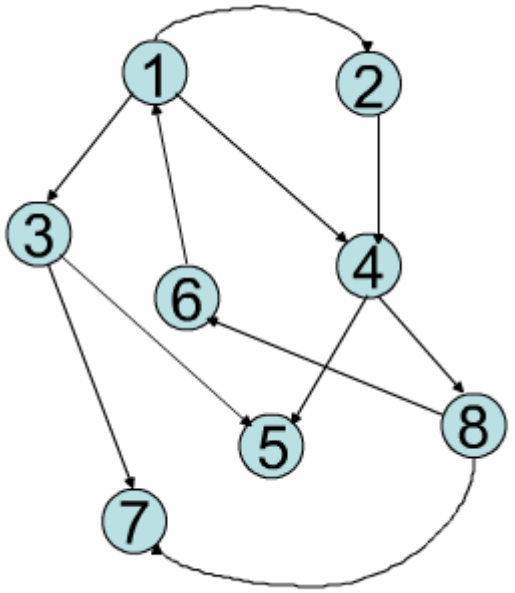
Ottmann, Universität Freiburg, Informatik II, SS 2008

Überblick

- Definition / Eigenschaften
- Anwendungen
- Repräsentation
- Traversierung
- Kürzester Weg

Entfernungen in ungewichteten Graphen

- alle Kanten haben gleiche Kosten (beispielsweise 1)
- Bestimmung aller Entfernungen zu einem Knoten (Single-source shortest-path problem)
- Beispiel
  - Entfernungen zum Knoten 1
  - $1 \rightarrow 1$ (Entfernung 0)
  - $1 \rightarrow 2$ (Entfernung 1)
  - $1 \rightarrow 3$ (Entfernung 1)
  - $1 \rightarrow 4$ (Entfernung 1)
  - $1 \rightarrow 4 \rightarrow 5$ (Entfernung 2)
  - $1 \rightarrow 4 \rightarrow 8 \rightarrow 6$ (Entfernung 3)
  - $1 \rightarrow 4 \rightarrow 8 \rightarrow 7$ (Entfernung 3)
  - $1 \rightarrow 4 \rightarrow 8$ (Entfernung 2)



Entfernungen in ungewichteter Graph

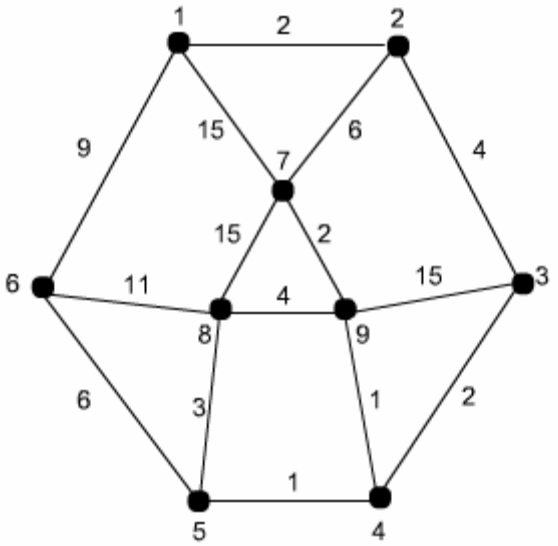
- Knoten v ist von sich selbst 0 Schritte entfernt
- Nachfolger von v sind 1 Schritt entfernt
- Knoten der Entfernung j sind Knoten, die von den in j-1 Schritten entfernten Knoten in genau einem Schritt erreicht werden können
- Realisierung durch Breitensuche
  - statt v.besucht wird v.distanz gespeichert
  - foreach (w ist Nachfolger von v) do
    if (w.distanz == -1) w.distanz = v.distanz + 1;
- Laufzeit O (|V| + |E|) für Adjazenzlisten

Kürzester Weg

- Sind Distanzen aller Knoten zu einem Knoten v gegeben, kann der kürzeste Weg von jedem Knoten zu v angegeben werden.
- Pfad wird rückwärts rekonstruiert.
- Starte bei u
- Betrachte alle Vorgänger / Nachbarn von u
- w ist der Vorgänger / Nachbar mit der kleinsten Distanz
  $\rightarrow$ w liegt auf kürzestem Pfad
- Betrachte alle Vorgänger / Nachbarn von w
- ...
- Halte, wenn v erreicht

Entfernungen in gewichteten Graphen

- Kanten sind mit reelwertigen, nichtnegativen Zahlen bewertet
  - Kosten, Distanzen, ...
- Beispiel
  - $1 \rightarrow 7$ (Kosten 15)
  - $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 9 \rightarrow 7$ (Kosten 11)
  - $1 \rightarrow 2 \rightarrow 7$ (Kosten 8)
- im Unterschied zu ungewichteten Graphen können Pfade mit mehr Knoten kürzer sein als Pfade mit weniger Knoten



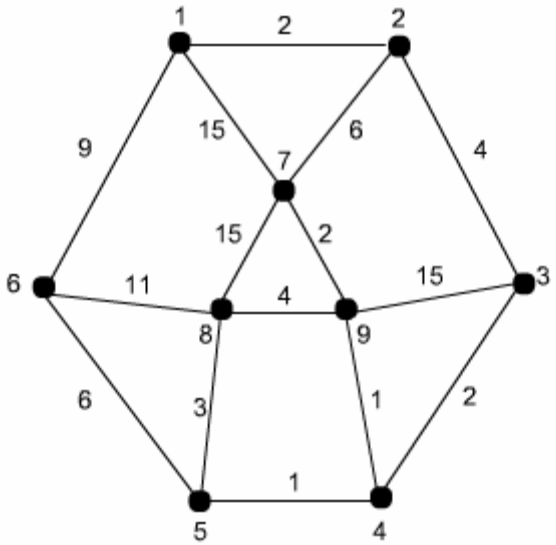
Ottmann, Universität Freiburg, Informatik II, SS 2008

Beobachtungen

- verwende bereits bekannte kürzeste Wege
  und erweitere sie schrittweise
  - funktioniert nur für nichtnegative Kantengewichte (Terminierung!)
- Optimalitätsprinzip:
  Für jeden kürzesten Weg p = (v_0, v_1, ..., v_k) von v_0 nach v_k ist jeder Teilweg p' = (v_i, ..., v_j) ein kürzester Weg von v_i nach v_j
- Begründung:
  Wäre dies nicht so, könnte man den Teilweg (v_i, ..., v_j) in p durch einen kürzeren Weg p' ersetzen, um einen neuen Weg p'' = (v_0, v_1, ..., v_k) zu erhalten, der kürzer ist als p
  $\rightarrow$ Widerspruch

Beobachtungen

- 1. Für kürzesten Weg sp(s, v) und alle Kanten (v,v') gilt: distanz (sp(s,v)) + distanz (v, v') ≥ distanz (sp(s,v'))
- 2. Für kürzesten Weg sp(s, v) existiert eine Kante (v,v') mit: distanz (sp(s,v)) + distanz (v, v') = distanz (sp(s,v'))



zu 1.
Trivial. Jede Distanz zwischen zwei Knoten s und v' ist größer oder gleich der minimalen Distanz zwischen den beiden Knoten.

zu 2.
Für mindestens einen Nachfolgerknoten v' von v muss (v, v') die kürzeste Distanz zwischen v und v' darstellen.
distanz (1, 2) = 2 ist optimal, da distanz (1, 7) = 15 und distanz (1, 6) = 9 bereits größer als 2 sind und alle Kantengewichte positiv sind..

Dijkstra-Algorithmus

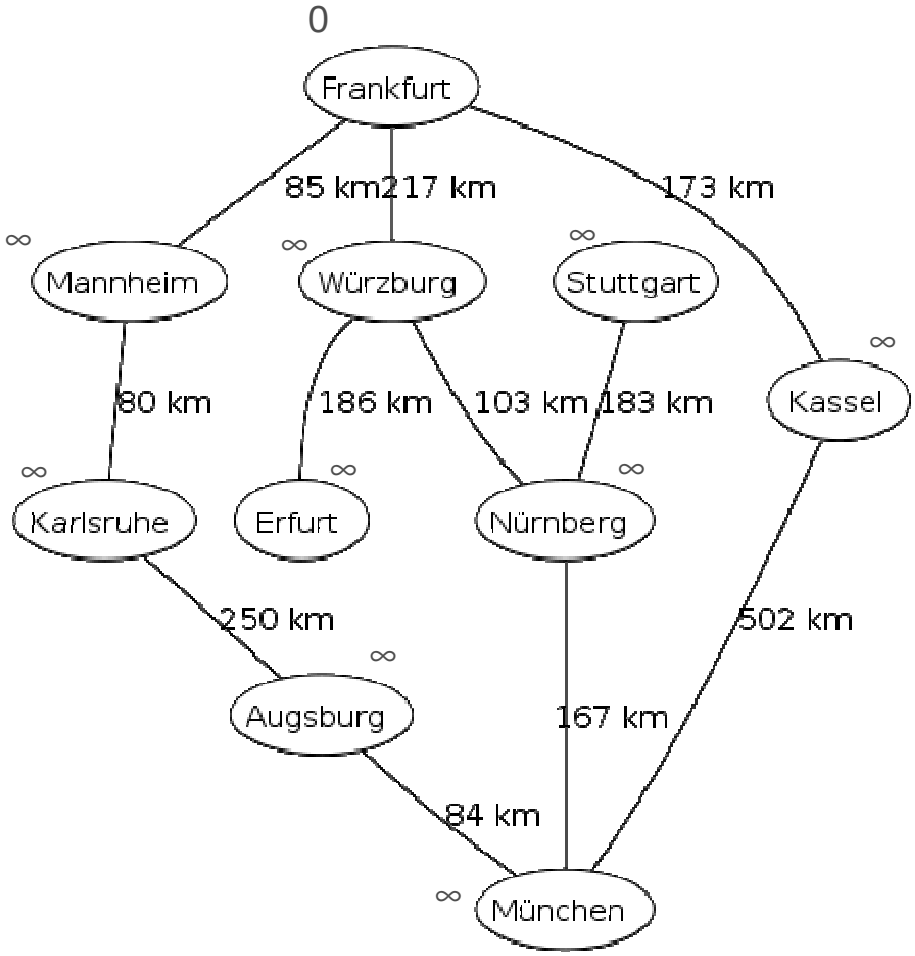
Edsger Wybe Dijkstra (1930 - 2002) niederländischer Informatiker

- Initialisierung:
  - alle Knoten v außer Startknoten mit v.distanz = ∞
  - Startknoten w mit w.distanz = 0
  - alle Knoten v mit v.vorgänger = nichtInitialisiert
  - alle Knoten v sind Element der Menge M (nicht besuchte Knoten)
- Wiederhole, solange M nicht leer ist
  - entferne Knoten v mit minimaler Distanz aus M
  - für alle Nachfolger w von v:
    if (v.distanz + distanz(v,w) < w.distanz)
    { w.distanz = v.distanz + distanz(v,w); w.vorgänger = v; }

Beispiel

Wikipedia: Dijkstra-Algorithmus

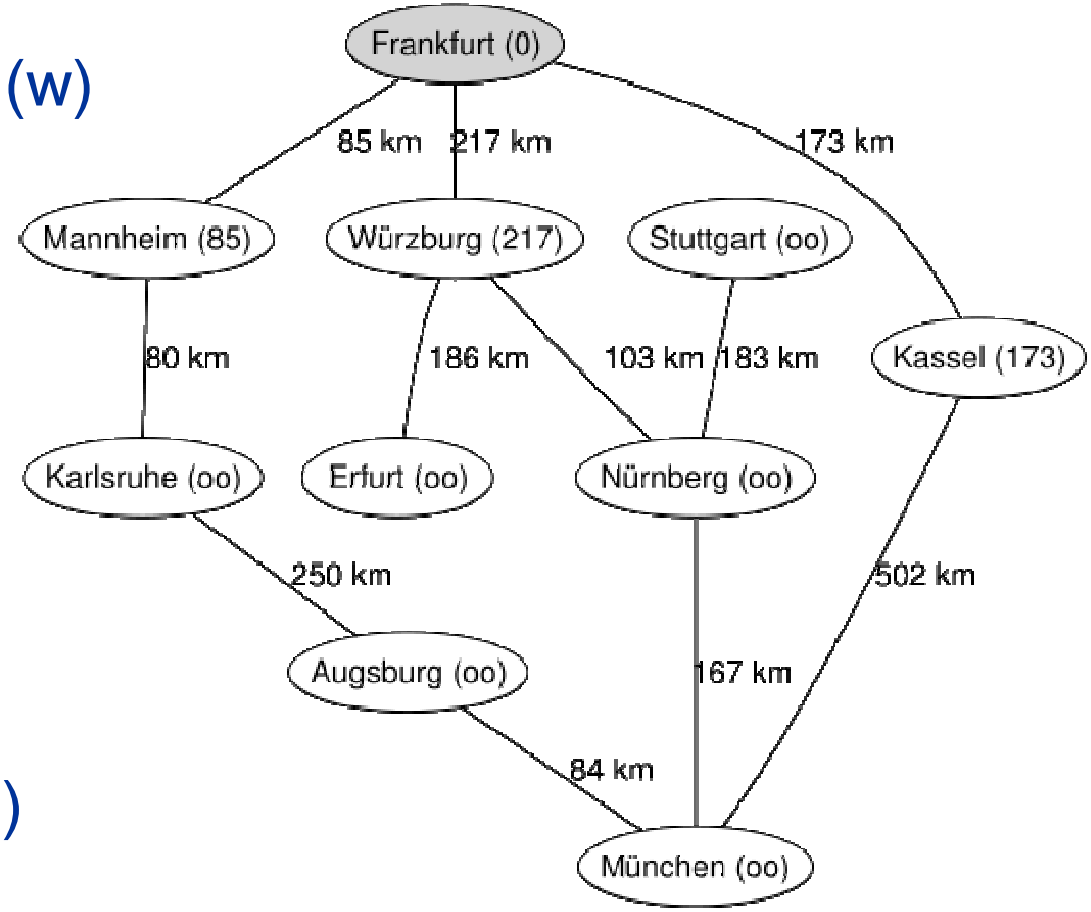
- Initialisierung
  - alle Orte sind in M



Schritt 1

Wikipedia: Dijkstra-Algorithmus

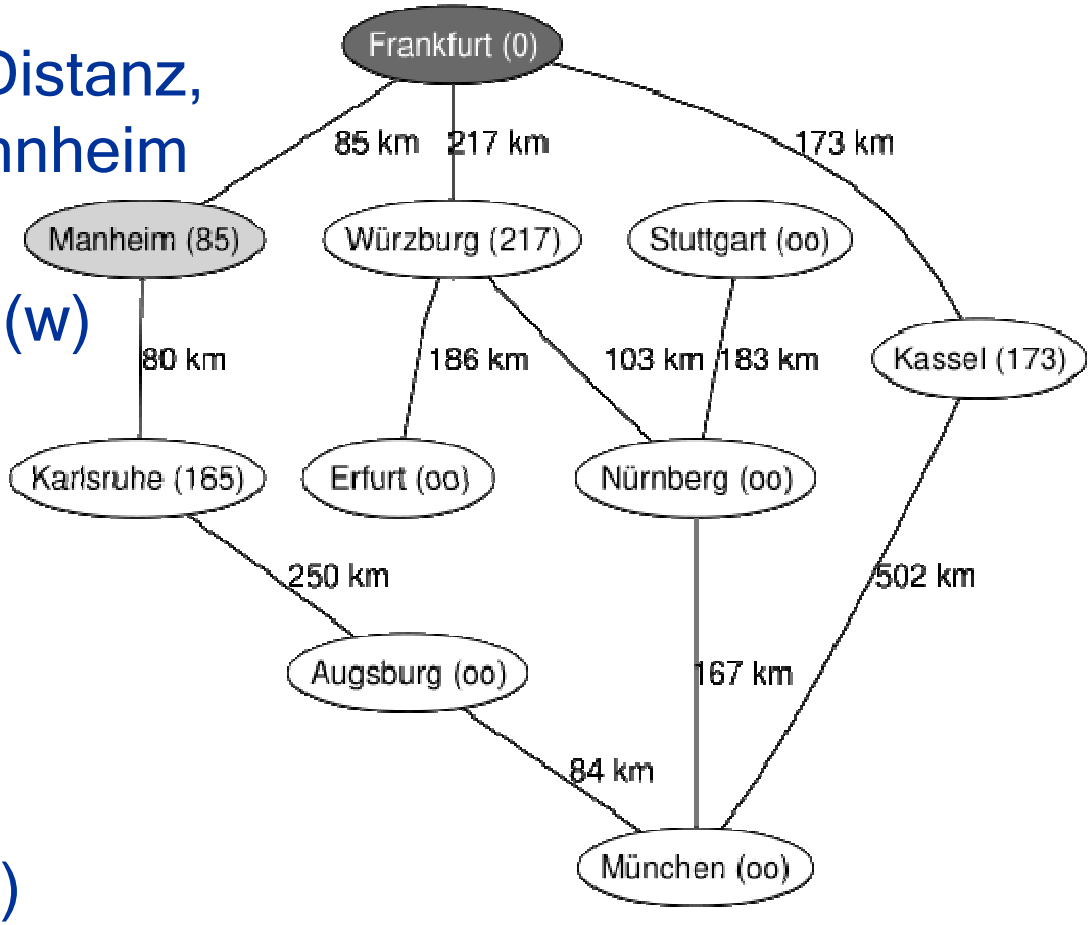
- entferne Frankfurt (v)
- aktualisiere Nachfolger (w)
  - Mannheim
  - Würzburg
  - Kassel
- aktualisiere jeweils Distanz und Vorgänger-Information
- w.distanz = min(w.distanz, v.distanz + distanz(v,w))



Schritt 2

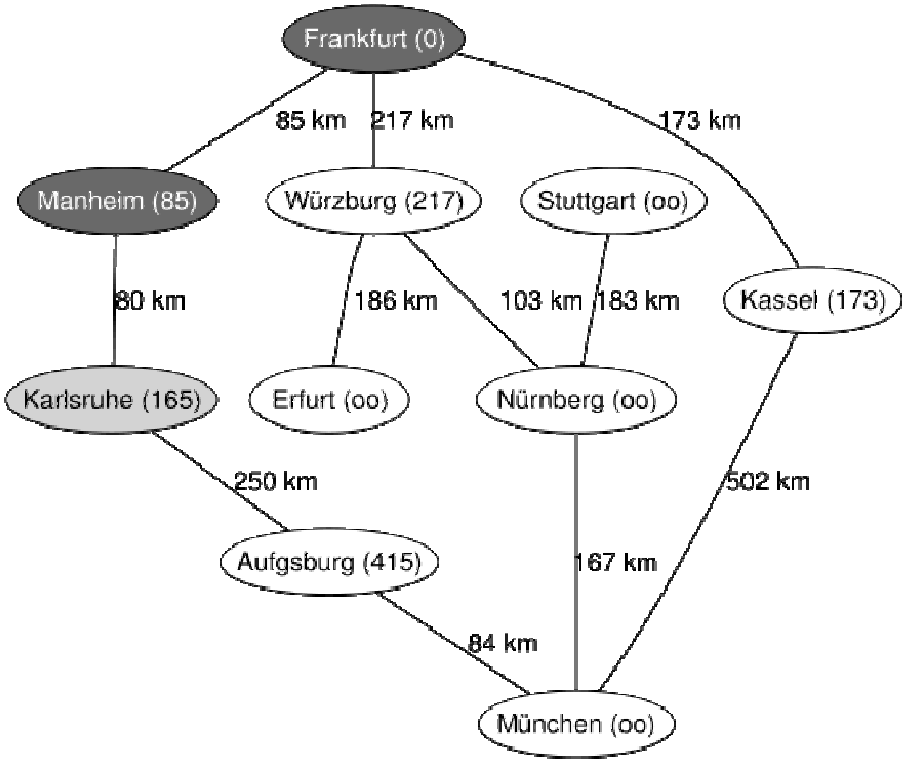
Wikipedia: Dijkstra-Algorithmus

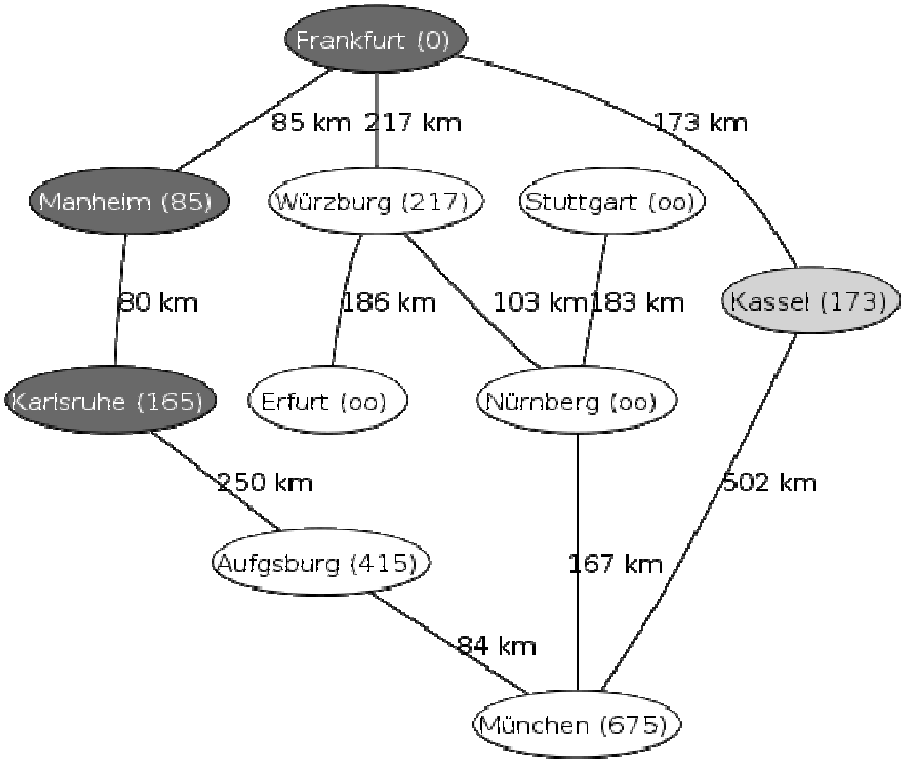
- entferne Mannheim (v)
  (Knoten mit minimaler Distanz, kürzerer Weg nach Mannheim nicht möglich)
- aktualisiere Nachfolger (w)
  - Karlsruhe
- aktualisiere jeweils Distanz und Vorgänger-Information
- w.distanz = min(w.distanz, v.distanz + distanz(v,w))



Schritt 3 und 4

Wikipedia: Dijkstra-Algorithmus

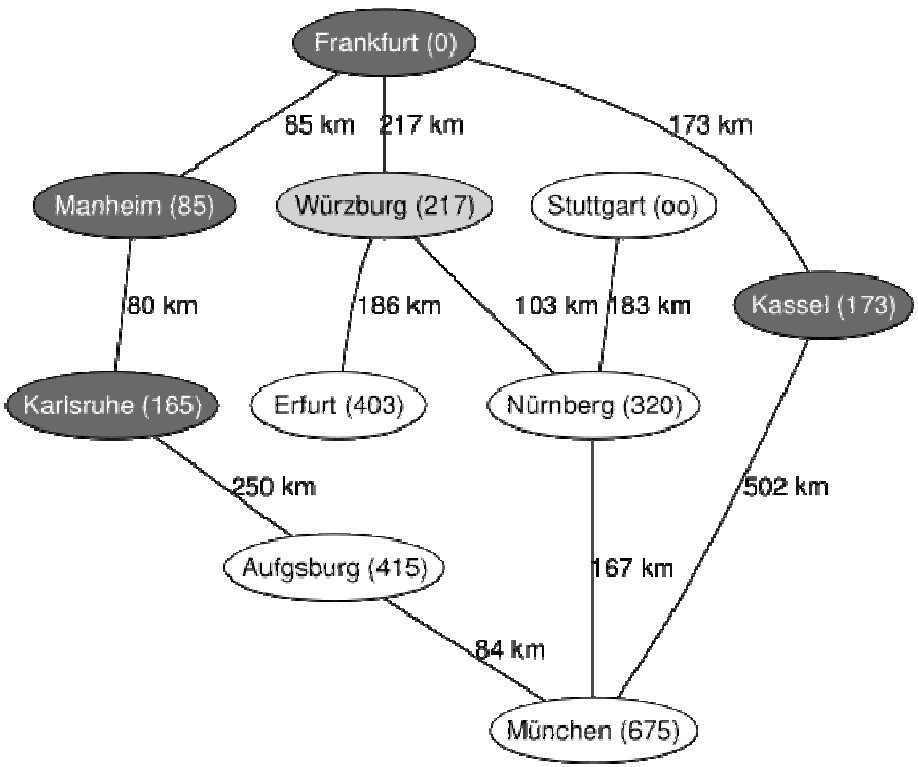


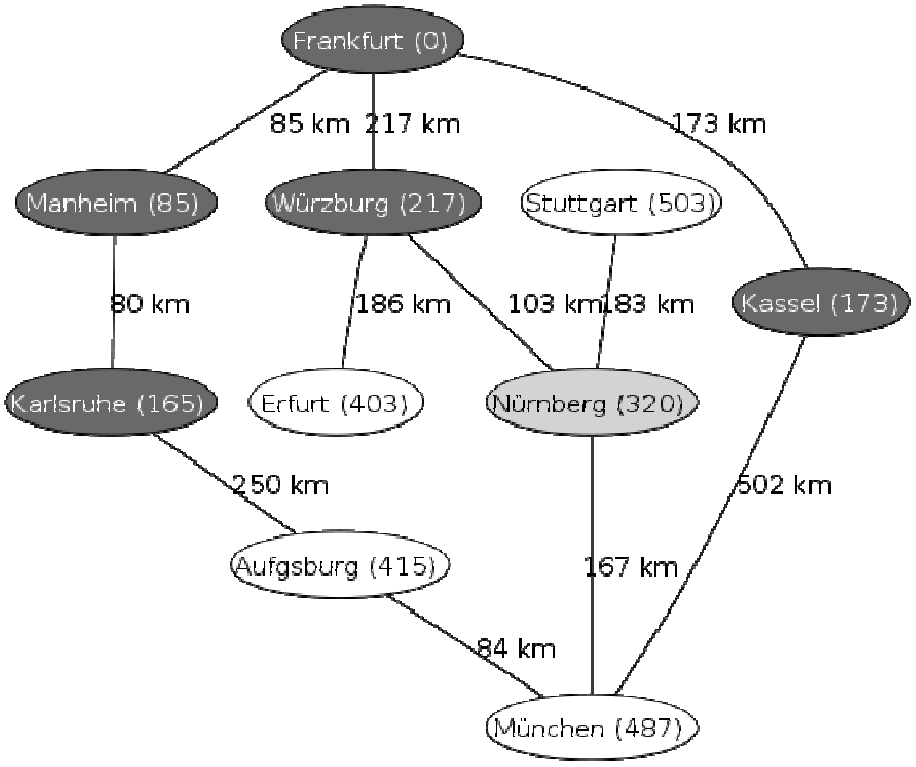


Kassel ist Vorgänger von München.

Schritt 5 und 6

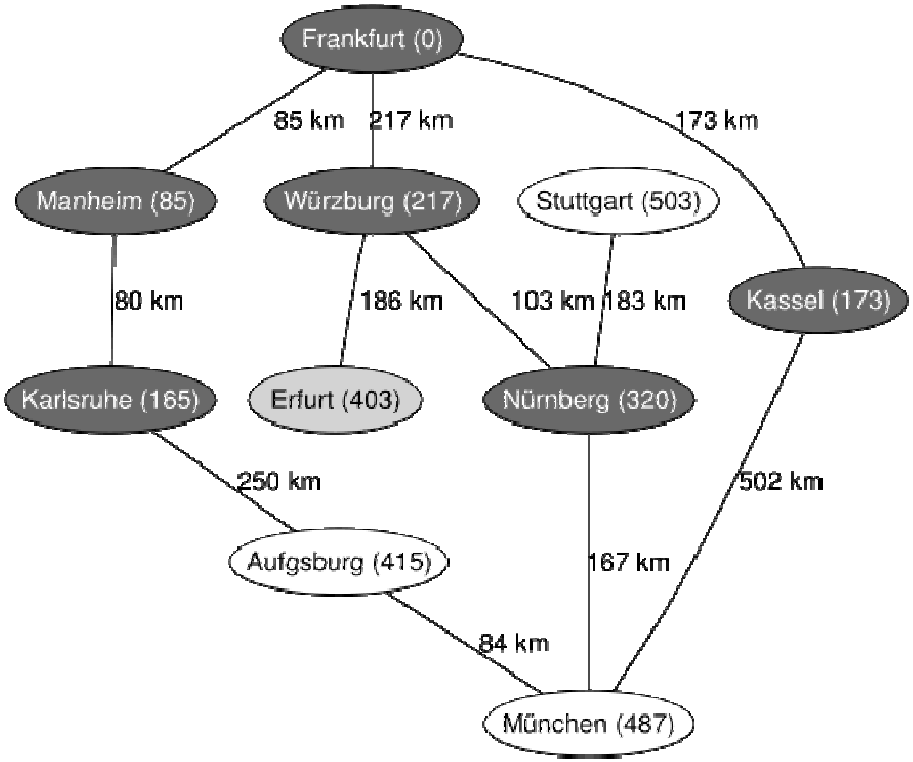
Wikipedia: Dijkstra-Algorithmus

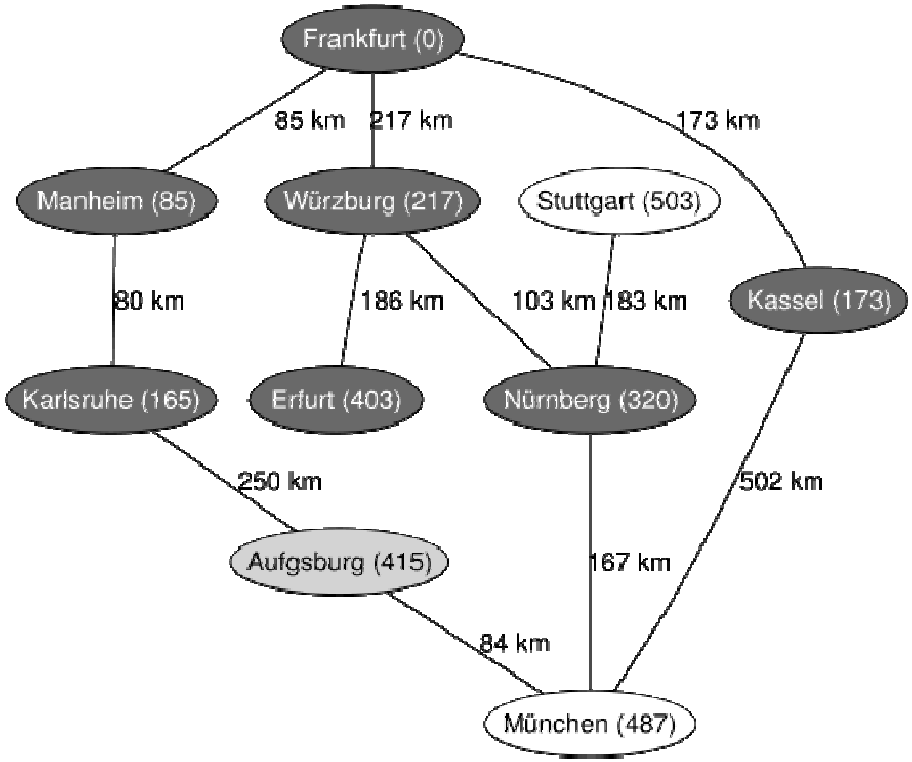




320 + 167 = 487 < 675
Aktualisierung von München
(Nürnberg ist jetzt Vorgänger
von München.)

Schritt 7 und 8



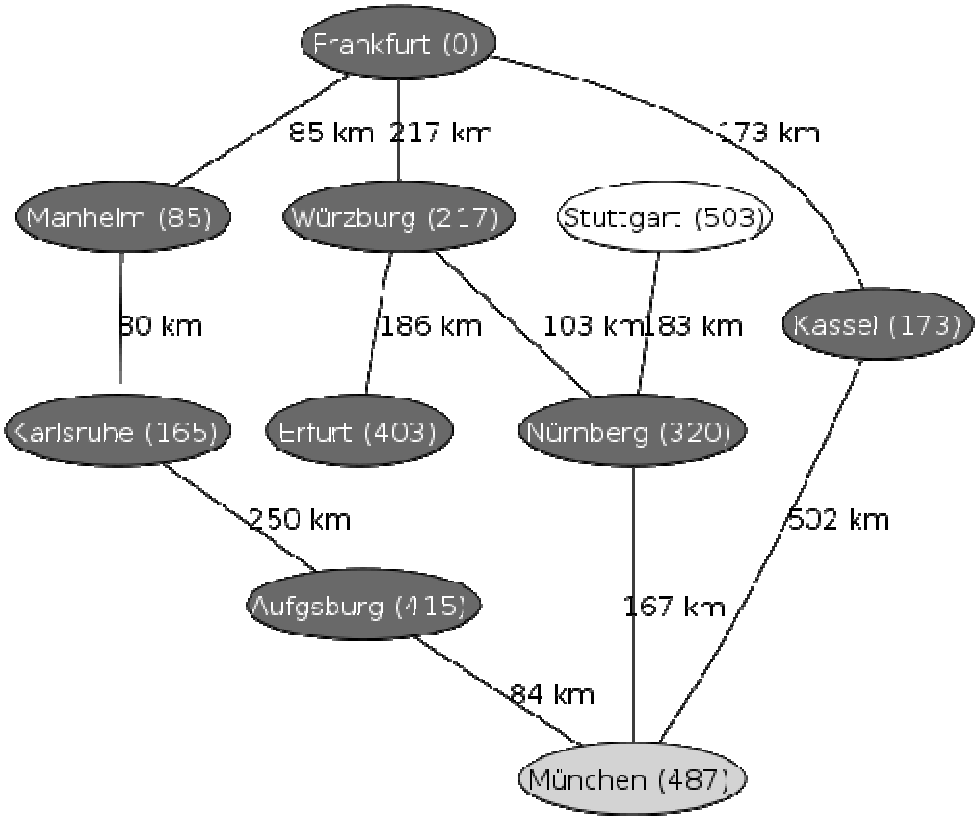


Wikipedia: Dijkstra-Algorithmus

415 + 84 > 487
München wird nicht verändert.

Schritt 9 und 10

Wikipedia: Dijkstra-Algorithmus



Stuttgart wird entfernt (kein Bild).

Diskussion

- Knoten v in M mit minimaler Distanz
  - zu diesem Knoten kann es keinen kürzeren Weg geben, da Wege über alle anderen Knoten in M länger wären (bei nichtnegativen Kantengewichten)
- Entfernung zu Knoten kann auch unendlich sein
  - bei unzusammenhängenden Graphen
- Menge M durch Prioritätswarteschlange repräsentiert
  - effizientes Entfernen des minimalen Elements
  - Implementierung durch Minimum-Heap
- Zeitkomplexität bei Verwendung von Adjazenzliste und Prioritätswarteschlange $O (|V| \cdot \log |V| + |E|)$

Zusammenfassung

- Graph
  - Paar von Knoten- und Kantenmenge
- Repräsentation
  - Adjazenzmatrix
  - Adjazenzliste
  - winged edge
- Traversierung
  - Tiefensuche
  - Breitensuche
- Kürzester Weg
  - Breitensuche für ungewichtete Graphen
  - Dijkstra-Algorithmus für Graphen mit nicht-negativen Gewichten

Nächstes Thema

- Algorithmen / Datenstrukturen
  - Bereichsbäume