Graphen

Algorithmen und Datenstrukturen

Prof. Dr. Stephan Trahasch

Übersicht

- Motivation
- Definitionen
- Kürzeste Wege
  - Dijkstras Algorithmus
  - Bellman-Ford-Algorithmus

Ziele

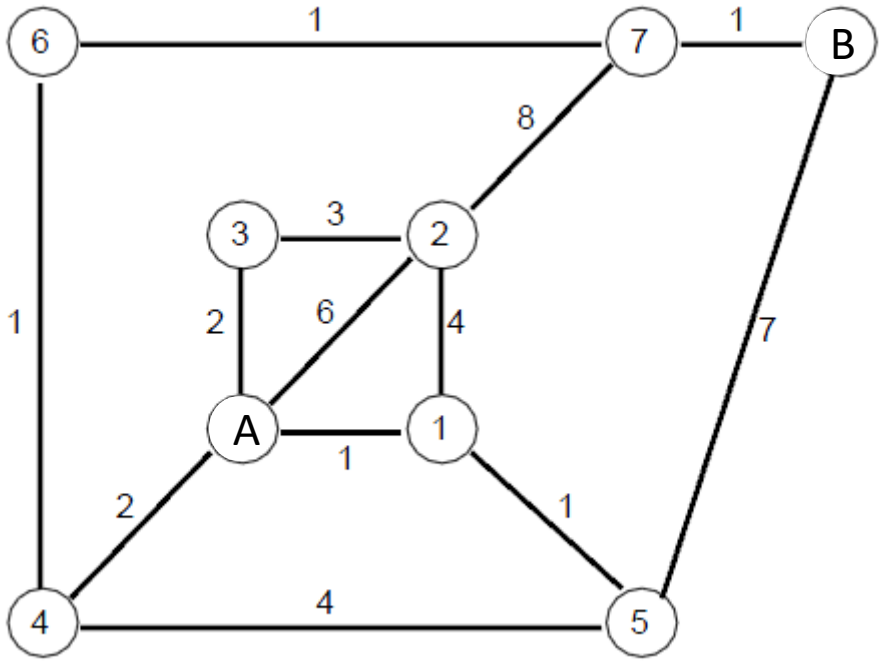
- Datenstruktur-Operationen implementieren können
- Geeignete Probleme auf Graphen abstrahieren können

Quellen

- Stefan Trahasch, Vorlesungsfolien Algorithmen und Datenstrukturen, Hochschule Offenburg
- Thomas Ottmann und Peter Widmayer, Algorithmen und Datenstrukturen, 4. Auflage, Spektrum, Berlin, 2002

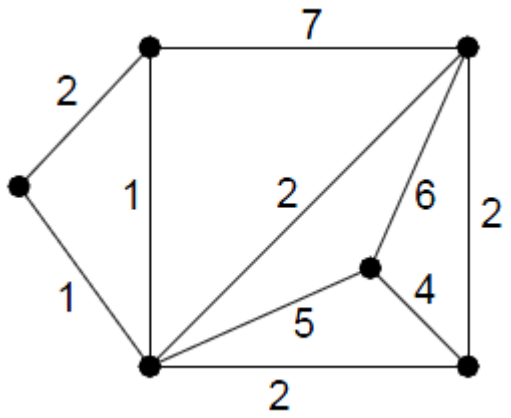
Routenplanung

Finde in einem Straßennetz die kürzeste Verbindung (Länge bzw. Fahrzeit) von einer Stadt A zu einer Stadt B.

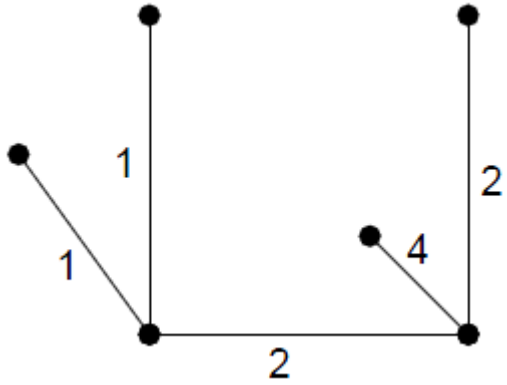


Netzplanung

Zwischen n Orten soll ein Versorgungsnetz (Kommunikation, Strom, Wasser, etc.) aufgebaut werden, so dass je 2 Orte direkt oder indirekt miteinander verbunden sind. Die Verbindungskosten zwischen 2 Orte seien bekannt. Gesucht ist ein Versorgungsnetz mit den geringsten Kosten.



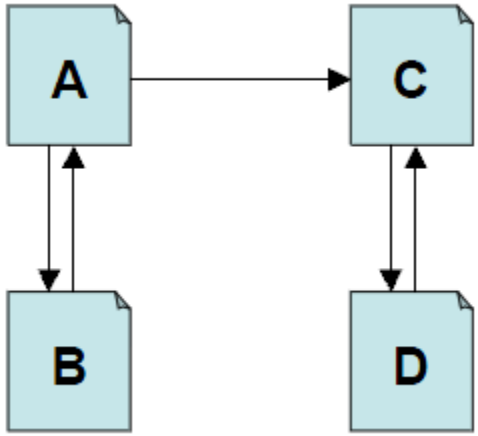
Orte mit Verbindungskosten



Günstigstes Versorgungsnetz

Suchmaschinen

Webseiten mit Links:



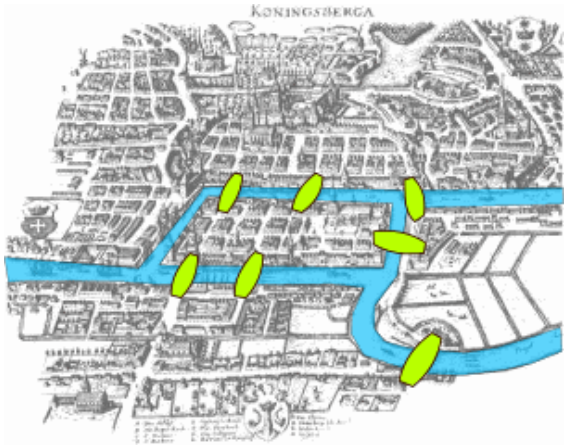
Besuche alle Internetseiten, extrahiere Schlüsselwörter und speichere diese in einen Index.

Motivation

- Wie komme ich am besten von Freiburg nach Ulm?
- Was ist die kürzeste Rundreise durch eine gegebene Menge von Städten?
- Welche Menge an Wasser kann die Kanalisation von Offenburg maximal verkraften?
- Gibt es einen Rundweg über die Brücken von Königsberg (Kaliningrad), derart dass jede Brücke nur einmal überquert wird und man zum Ausgangspunkt zurückgelangt?

Diese und viele andere Probleme lassen sich als Graphenprobleme definieren.

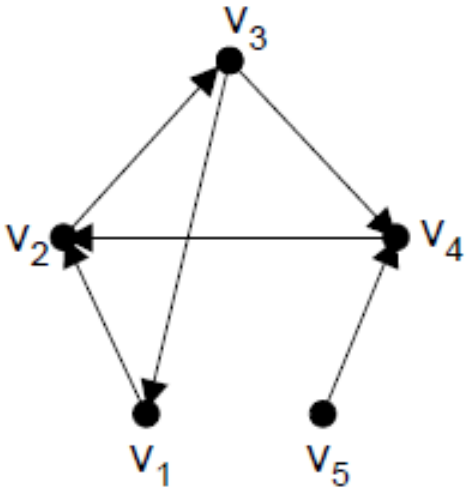
Das Königsberger Brückenproblem



Gibt es einen Weg, bei dem man alle sieben Brücken über den Pregel genau einmal überquert, und wenn ja, gibt es auch einen Rundweg, bei dem man wieder zum Ausgangspunkt gelangt?

Definition: Gerichteter Graph

Ein ***gerichteter Graph*** *G = (V,E)* (englisch: *digraph*) besteht aus einer Menge *V* = {1, 2, . . . , |*V* |} von ***Knoten*** (englisch: *vertices*) und einer Menge $E \subseteq V \times V$ von *Pfeilen* oder ***Kanten*** (englisch: *edges, arcs*).
Ein Paar $(v, v') \subseteq E$ heißt *Pfeil* oder ***Kante*** *von v nach v´*.



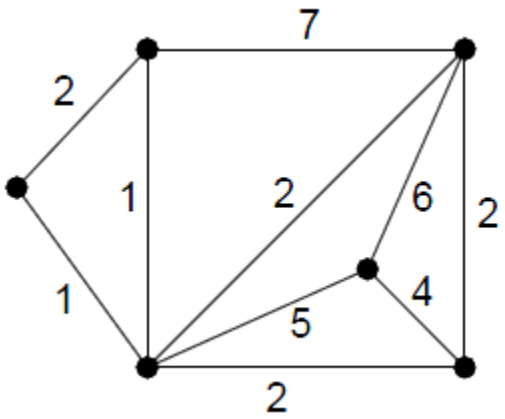
$V = \{v_1, v_2, v_3, v_4, v_5\}$
$E = \{(v_1,v_2), (v_3,v_1), (v_2,v_3), (v_4,v_2), (v_3,v_4), (v_5,v_4)\}$

Ungerichteter Graph

- Bei einem ungerichteten Graphen sind die Kanten ungerichtet. Eine Kante zwischen v und w kann als 2-elementige Menge {v, w} dargestellt werden.
- Der Einfachheit wegen wollen wir jedoch auch ungerichtete Kanten als Paar (v, w) darstellen, wobei im Kontext festgelegt wird, ob Kanten gerichtet oder ungerichtet zu verstehen sind.

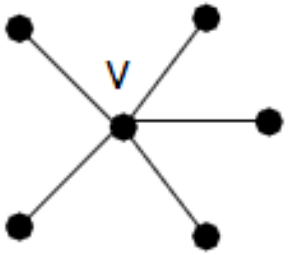
Gewichteter Graph

- Ist jeder Kante (v,w) eine reelle Zahl c(v,w) als Kosten oder Gewicht zugeordnet, spricht man von einem gewichteten Graphen.
- Sind darüber hinaus die Gewichte ≥ 0, dann heißt der Graph auch Distanzgraph.

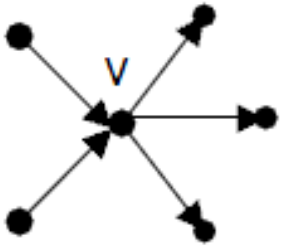


Nachbarn und Grad

Zwei Knoten v und w, die durch eine (gerichtete oder ungerichtete) Kante verbunden sind, heißen **Nachbarn** bzw. sind adjazent. Bei einem gerichteten Graphen können Nachbarn in Vorgänger und Nachfolger unterschieden werden.
Der **Grad eines Knoten** ist die Anzahl seiner Nachbarn. Bei einem gerichteten Graphen kann zwischen Eingangs- und Ausgangsgrad unterschieden werden.



Der Knoten v hat 5 Nachbarn.
Der Grad von v ist damit 5.



Der Knoten v hat 5 Nachbarn:
2 Vorgänger und 3 Nachfolger.
Der Eingangsgrad von v ist damit 2 und der Ausgangsgrad 3.

Weg und Zyklus

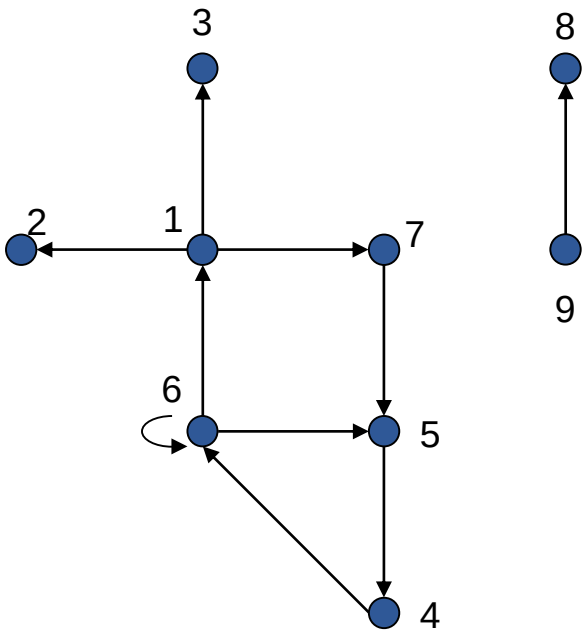
- Ein Weg (engl. path) ist eine Folge von Knoten $v_1, v_2, \ldots, v_n$, so dass $(v_i, v_{i+1}) \in E$ für $1 \leq i < n$.
- Die Länge dieses Weges ist gleich der Anzahl der Kanten und damit n-1.
- Ein Weg heißt einfacher Weg, falls alle Knoten unterschiedlich sind.
- Ein Weg $v_1, v_2, \ldots, v_n$ heißt geschlossen oder auch Zyklus (engl. cycle), falls Anfangsknoten v_1 und Endknoten v_n identisch sind.
- Zyklenfreie Graphen werden auch azyklisch genannt.

Adjazenzmatrizen

- Adjazenzmatrizen dienen der Speicherung von Graphen.
- Ein Graph $G = (V,E)$ wird in einer Boole'schen $|V| \times |V|$-Matrix $A_G = (a_{ij})$, mit $1 \leq i \leq |V|$, $1 \leq j \leq |V|$ gespeichert, wobei

$$a_{ij} = \begin{cases} 1 \text{ falls } (i,j) \in E \\ 0 \text{ sonst} \end{cases}$$

Beispiel einer Adjazenzmatrix



	1	2	3	4	5	6	7	8	9
1									
2									
3									
4									
5									
6									
7									
8									
9									

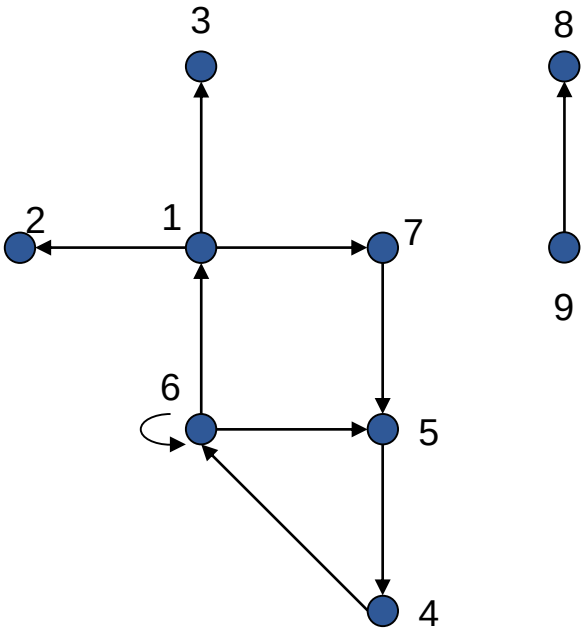
Eigenschaften von Adjazenzmatrizen

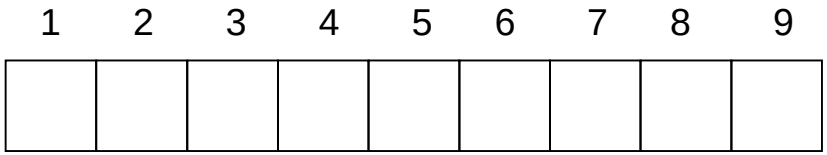
- Bei der Speicherung eines Graphen mit Knotenmenge V in einer Adjazenzmatrix ergibt sich ein Speicherbedarf von $\Theta(|V|^2)$.
- Speicherbedarf ist nicht abhängig von der Anzahl der Kanten im Graphen.
- Demnach sind Adjazenzmatrizen ungünstig, wenn der Graph vergleichsweise wenige Kanten enthält.
- Wegen der erforderlichen Initialisierung der Matrix oder der Berücksichtigung aller Einträge der Matrix benötigen die meisten Algorithmen $\Omega(|V|^2)$ Rechenschritte.

Adjazenzlisten

- Bei Adjazenzlisten wird für jeden Knoten eine lineare, verkettete Liste der von diesem Knoten ausgehenden Kanten gespeichert.
- Die Knoten werden als lineares Feld von $|V|$ Anfangszeigern auf je eine solche Liste verwaltet.
- Die *i*-te Liste enthält ein Listenelement mit Eintrag *j* für jeden Endknoten eines Pfeils $(i, j) \in E$.
- Adjazenzlisten unterstützen viele Operationen, z.B. das Verfolgen von Pfeilen in Graphen, sehr gut.
- Andere Operationen dagegen werden nur schlecht unterstützt, insbesondere das Hinzufügen und Entfernen von Knoten.

Beispiel für Adjazenzliste





Gliederung

- Motivation
- Definitionen
- Kürzeste Wege
  - Dijkstras Algorithmus
  - Bellmann-Ford-Algorithmus

Problemstellung

Länge eines Weges

- Gewichteter Graph: Summe der Kantengewichte
- Ungewichteter Graph: Anzahl der Kanten.
- Ein ungewichteter Graph entspricht damit einem gewichteten Graphen, bei dem jede Kante die Kosten 1 hat

Kürzester Weg und Distanz

Im allgemeinen gibt es zwischen zwei Knoten u und v mehrere Wege. Ein Weg von u nach v heißt kürzester Weg, falls der Weg minimale Länge hat.

Die Distanz $\gamma(u,v)$ zwischen u und v wird als die Länge eines kürzesten Weges definiert. Falls es keinen Weg von u nach v gibt, ist $\gamma(u,v) = \infty$.

Kürzeste Wege

Der **kürzeste Weg** von einem Startknoten s zu einem Zielknoten z in einem gerichteten gewichteten Graphen G = (V, E, γ) mit
γ : E → N ist derjenige Pfad, der s mit z verbindet und bei dem die Summe der Kantengewichte minimal ist.

Ein **Pfad P** in einem Graphen von s nach z ist definiert als eine Folge von aneinander stoßenden Kanten

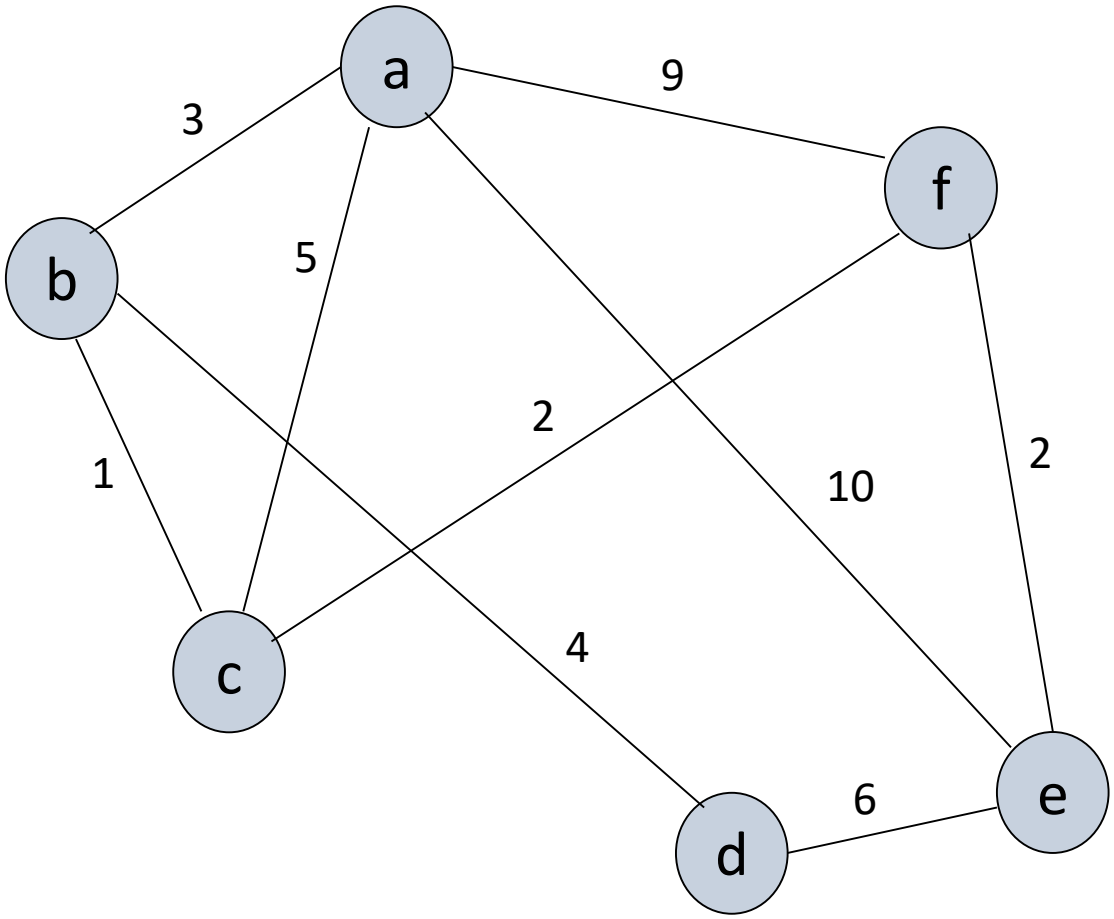
$$P = \mathbf{(}(s, v_1), (v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, z)\mathbf{)}$$

Das **Gewicht w** eines Pfades P ist definiert als die Summe der einzelnen Kantengewichte (mit $v_0 = s$ und $v_{n+1} = z$):

$$w(P) = \sum_{i=0}^{n-1} \gamma(v_i, v_{i+1})$$

Die **Distanz d(s, z)** zweier Knoten s und z ist definiert als das Minimum der Gewichte aller Pfade von s nach z.

Beispiel



Gewichte einer Kante

Länge eines Weges muss nicht die wörtliche Bedeutung haben.

Die Gewichte (Kosten) der Kanten und damit die Weglängen können in Abhängigkeit von der Anwendung ganz unterschiedliche Bedeutungen haben.
Beispiele:

- Streckenlänge
- Zeitspannen
- Kosten
- Profit: Gewinn/Verlust (Gewichte können positiv und negativ sein)
- Wahrscheinlichkeiten

Problemvarianten

1. Kürzeste Wege zwischen zwei Knoten
   → Single pair shortest path
2. Kürzeste Wege zwischen einem Knoten und allen anderen Knoten (Dijkstra)
   → Single source shortest path problem
3. Kürzeste Wege zwischen allen Knotenpaaren
   → all pairs shortest path

- Für Problem (1) kennt man keine bessere Lösung, als einen Algorithmus für Lösung (2) zu nehmen, der abgebrochen wird, sobald der Zielknoten erreicht wird.
- Problem (3) kann auf Problem (2) zurückgeführt werden.

Single-Source-Shortest-Path

Das **Single-Source-Shortest-Path-Problem** besteht darin, für einen Graph $G = (V,E)$ und einen Knoten v die kürzesten Pfade von v zu allen anderen Knoten in G zu bestimmen.

Annahmen: Gewichte der Kanten > 0.

Überlegung

Optimalitätsprinzip:

Für jeden kürzesten Weg $p = (v_0, v_1, \ldots, v_k)$ von v_0 nach v_k ist jeder Teilweg $p' = (v_i, \ldots, v_j)$, $0 \leq i < j \leq k$ ein kürzester Weg von v_i nach v_j .

Begründung:

1. Wäre dies nicht so, gäbe es also einen kürzeren Weg p'' von v_i nach v_j , so könnte auch in p der Teilweg p' durch p'' ersetzt werden und der entstehende Weg von v_0 nach v_k wäre kürzer als p.
2. Dies ist aber ein Widerspruch zu der Annahme, dass p ein kürzester Weg von v_0 nach v_k ist.

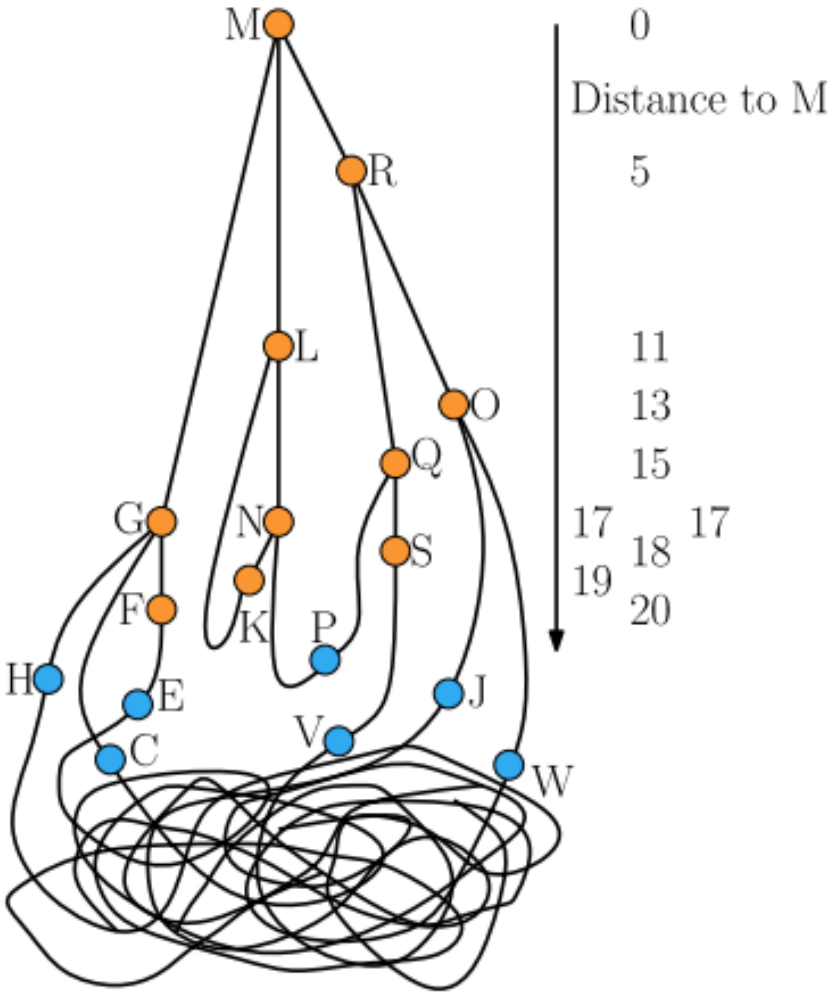
Edsger Wybe Dijkstra

- 1972 ACM Turingpreis
- THE: das erste Mulitasking-OS
- Semaphor
- Philosophen-Problem
- GOTO Statement Considered Harmful

Dijkstras Algorithmus

Idee:

Iterative Erweiterung einer Menge von "billig" erreichbaren Kanten.
Basiert auf dem Greedy-Prinzip (immer das nächstbeste Stück nehmen).
Weiterentwicklung der Breitensuche.



Dijkstras Algorithmus

1. Beginne mit Startknoten s als aktuellen Knoten a
2. Bestimme die Distanzen *der nur eine Kante entfernten Knoten als Minimum ihrer bisherigen Distanz (anfänglich ∞) und der Distanz von a plus dem Gewicht der Kante von a zu dem Knoten.*
3. *Betrachte a nicht weiter und wähle den Knoten mit dem niedrigsten Distanzwert für die nächste Iteration* (weiter Schritt 2).

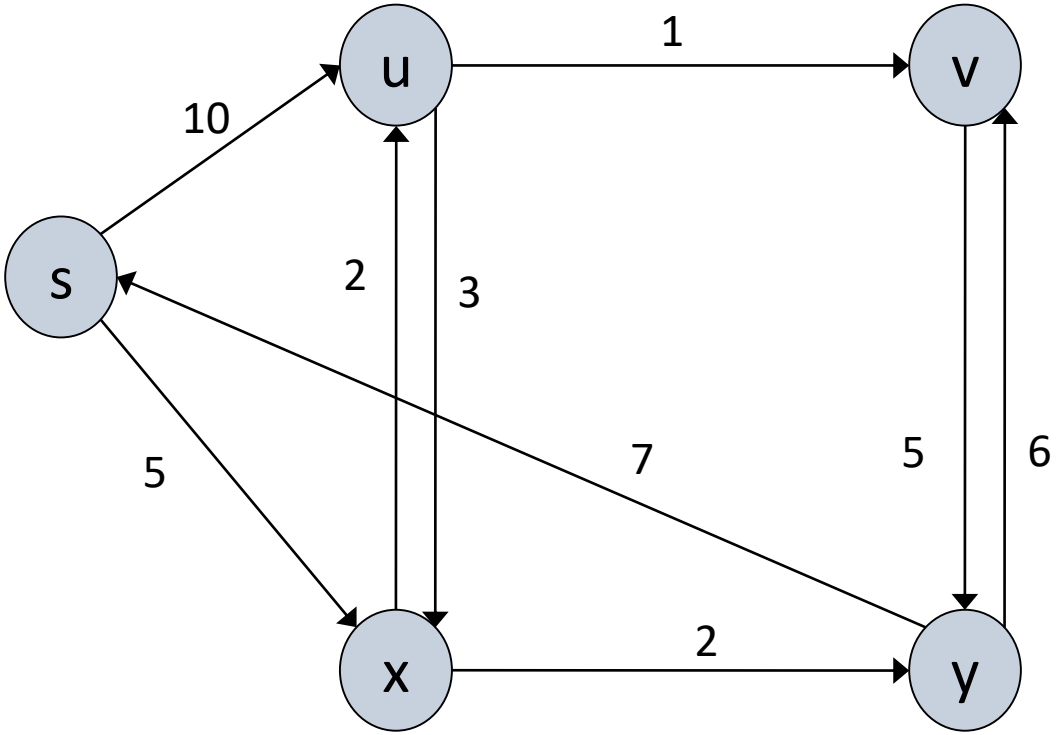
Dijkstras Algorithmus

```
Algorithmus Dijkstra (G,s)
Eingabe: Graph G=(V,E) mit Startknoten s

for each Knoten u ∈ G\s do
  D[u]:=∞
od;
D[s]:= 0; PriorityQueue Q:= V;

while isNotEmpty(Q) do
  u:=extractMinimum(Q);
  foreach v ∈ ZielknotenAusgehenderKanten von u ∩ Q do
      if D[u]+γ((u,v)) < D[v] then
            D[v]:= D[u]+γ((u,v));
            adjustiere Q an neuen Wert von D[v]
      fi
  od
od
```

Beispiel



Benötigte Datenstruktur

- Wir merken uns für jeden Knoten v die bisher berechnete, vorläufige *Entfernung d*(*v*) zum Anfangsknoten *s*.
- Weiter speichern wir den *Vorgänger* von *v* auf dem bisher berechneten vorläufig kürzesten Weg.
- Weiter benötigen wir eine Datenstruktur, um die noch zu bearbeitenden Knoten zu speichern. Dazu verwenden wir eine **Priority Queue**.

Laufzeitanalyse

Annahme:

Operationen auf der Priority Queue lassen sich in O(1) ausführen.

Graph G = (V,E) mit

- Anzahl der Knoten: |V| = n
- Anzahl der Kanten: |E| = m

⇨ Laufzeit für Dijsktras Algorithmus in O(n+m) unter der Annahme, dass sich Operationen auf der Priority Queue in O(1) ausführen lassen.

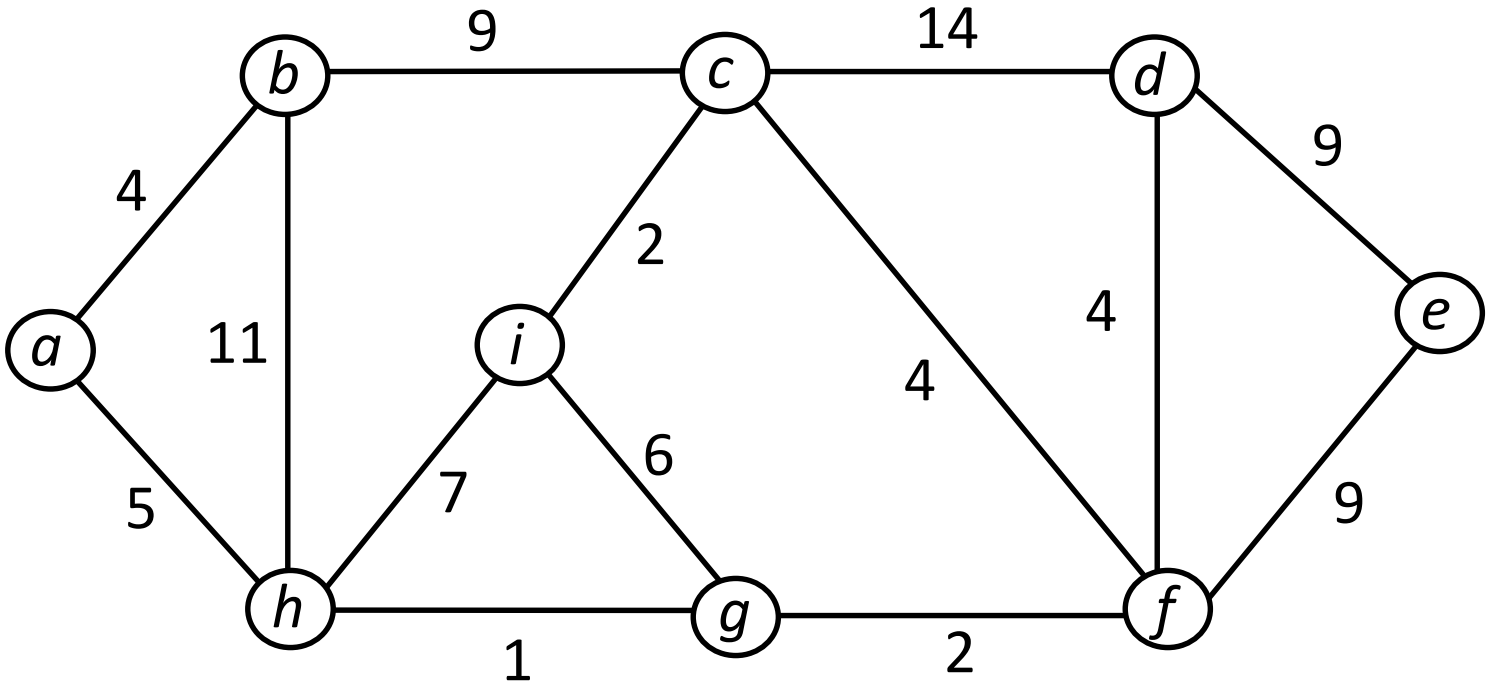
⇨ Laufzeit, wenn ein Fibonacci-Heap zur Implementierung der Priority Queue verwendet wird: **O(n*log n + m)**

Zusammenfassung

- Kürzeste Wege zwischen einem Knoten und allen anderen Knoten wird mit Dijkstra-Algorithmus berechnet.
- Aus einer Priority-Queue werden das Minimum extrahiert und dann die Pfade aktualisiert (greedy)
- Funktioniert nur, wenn Kantengewichte > 0
- Bei negativen Kantengewichten verwendet man den Bellman-Ford-Algorithmus


Übung

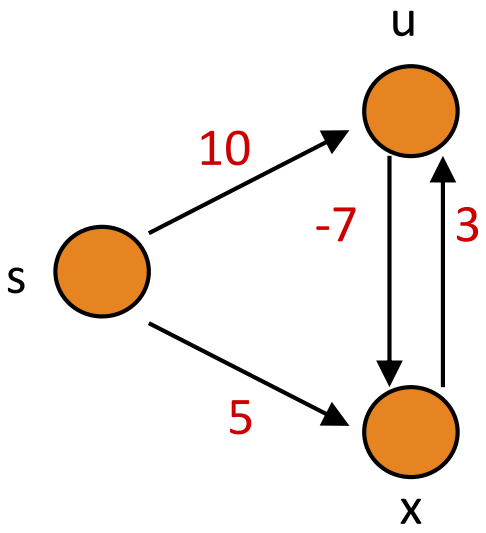
Geben Sie in dem folgenden Graphen die Längen der kürzesten Wege vom Knoten a zu allen anderen Knoten in einer möglichen Reihenfolge expandierter Knoten des Algorithmus von Dijkstra an.



Negative Kantengewichte

Der Dijkstra-Algorithmus funktioniert nicht für Graphen, bei denen negative Kantengewichte erlaubt sind (wenn also γ: E → Z).
Dann kann nicht garantiert werden, dass die Distanz durch zusätzliche Kanten noch weiter verringert werden kann.

Greedy-Algorithmen wie Dijkstras Algorithmus versagen, wenn eine lokal ungünstige Auswahl (etwa eine „teure“ Kante zu einem Nachfolge-knoten) nachträglich noch verbessert werden können (etwa durch ein negatives Kantengewicht ausgehend vom Nachfolgeknoten).
Um negative Kantengewichte korrekt zu berücksichtigen benötigt es daher einen anderen Algorithmus, der alle möglichen Pfade betrachtet und danach den kürzesten auswählt.



Bellman-Ford-Algorithmus

Im Bellman-Ford-Algorithmus wird in mehreren Durchläufen über die Kanten im Graph die beste bis zu diesem Zeitpunkt (für die betrachteten Kanten) mögliche Verbindung gesucht. In der i-ten Iteration werden dabei alle Pfade der Länge i vom Startknoten betrachtet.

Der längste mögliche Weg ohne Zyklus hat die Länge |V|-1, beinhaltet also eine Kante weniger als Knoten im Graphen vorhanden sind. Dabei würde der Pfad über alle Knoten verlaufen. Bei der Analyse aller möglicher Pfade kann man sich daher auf die Pfade bis zur Länge |V|-1 beschränken.

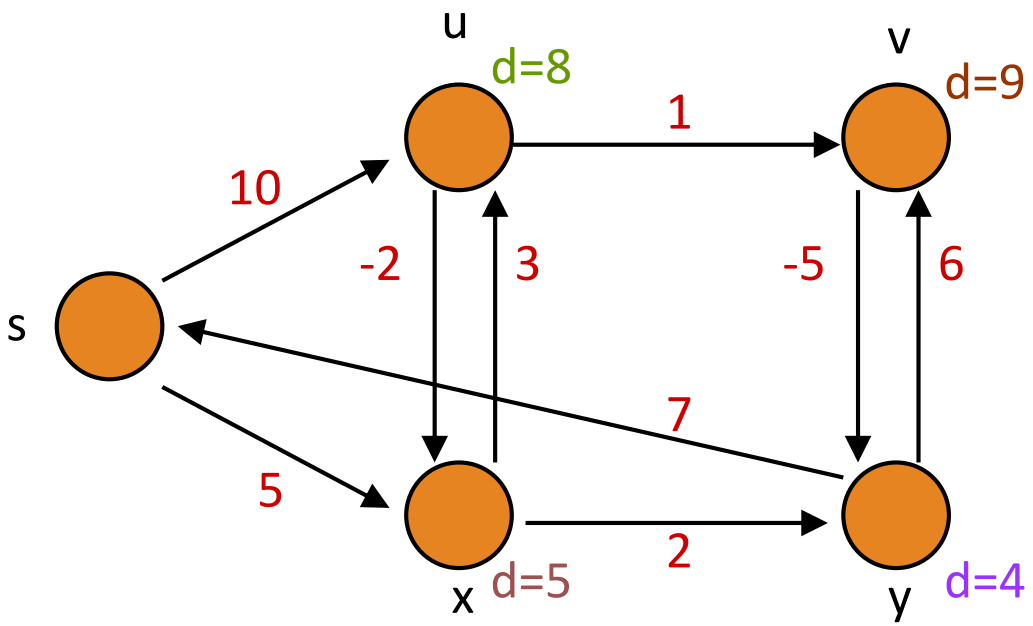
Bellman-Ford-Algorithmus (Pseudocode)

```
algorithm Bellman-Ford (G, s)
Eingabe: Graph G und Startknoten s∈V

for each Knoten u ∈ V-s do
  D(u):=∞
od

D(s):=0;
for i:= 1 to |V|-1 do
  for each (u, v) ∈ E do
       if D(u) + γ((u,v)) < D(v) then
             D(v) := D(u) + γ((u,v));
       fi
  od
od
```

Beispiel



Pfade der Länge 1

Pfade der Länge 2

Pfade der Länge 3

Pfade der Länge 4

Verbesserung des kürzesten Pfades

Nach dem letzten Schritt der Iteration, d.h. nach der Betrachtung aller Pfade der Länge |V|-1 lässt sich eine Verbesserung des kürzesten Pfades nur dann erreichen, wenn es einen Zyklus der Länge |V| gibt, der ein negative Länge besitzt. Dann wird bei jedem Durchlauf des Zyklus der Wert der Länge weiter verringert (in diesem Fall besitzt der Graph keinen endlichen Pfad mit minimaler Länge).