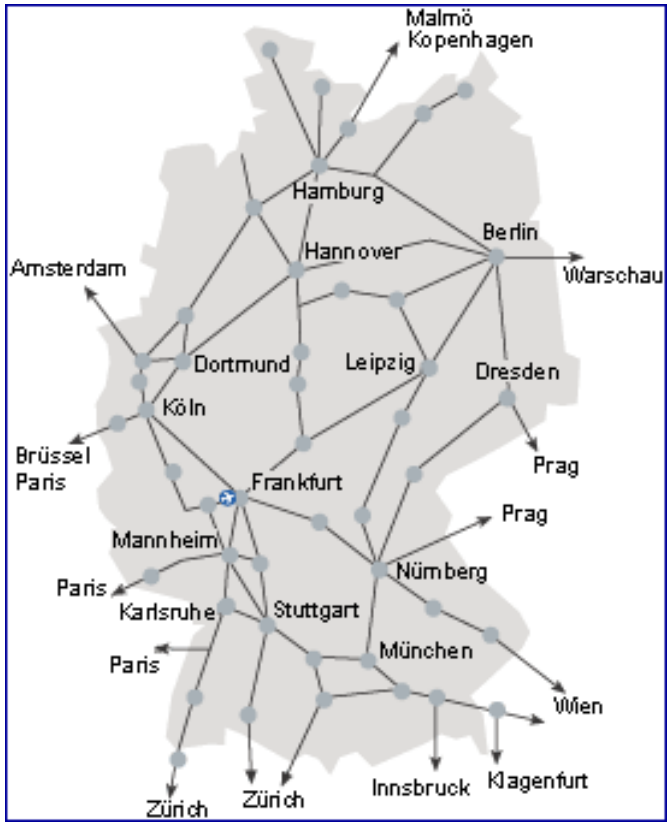
Kürzeste Wege in Graphen

Maurice Duvigneau
Otto-von-Guericke Universität
Fakultät für Informatik

Gliederung

- Einleitung
- Definitionen
- Algorithmus von Dijkstra
- Bellmann-Ford Algorithmus
- Floyd-Warshall Algorithmus
- Quellen

Einleitung



- Schienennetz
  - Als Kantenwichtung kann Fahrtzeit oder Fahrtkosten dienen
- Problem: schnellster bzw. günstigster Weg

Definitionen - **Was ist ein Weg?**

- Sei $G=(V, E)$ ein Graph und $W=(v_1,...,v_n)$ eine Folge von Knoten aus V, mit der Eigenschaft, dass für alle i aus $\{1,...,n\text{-}1\}$ gilt:
- das v_i und v_{i+1} durch eine Kante verbunden sind.
- Dann bezeichnet man W als ungerichteten Weg in G, falls G ungerichtet ist, und als gerichteten Weg in G, falls G gerichtet ist.
- Eine andere Bezeichnung für Weg ist Kantenfolge. Den Knoten v_1 nennt man Startknoten von W und den Knoten v_n Endknoten von W.

Definitionen – Länge eines Weges

- In Graphen ohne gewichtete Kanten bezeichnet man mit n-1 die Länge eines Weges, wobei W=(v_1,...,v_n).
- Anschaulich zählt man also die Anzahl zugehöriger Kanten.
- In kantengewichteten Graphen bezeichnet man als Länge eines Weges die Summe der Kantengewichte aller zugehörigen Kanten.
- Als einen kürzesten Weg bezeichnet man einen Weg von *s* nach *t*, dessen Länge minimal ist.
- Die Länge eines kürzesten Weges nennt man dann Abstand oder Distanz von *s* nach *t*.

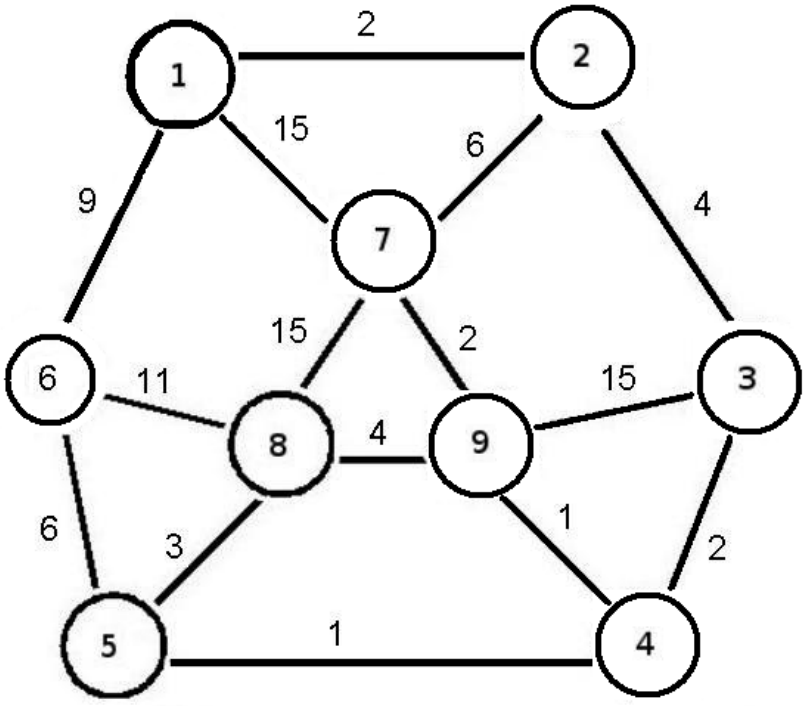
Algorithmus von Dijkstra

- Nach seinem Erfinder Edsger W. Dijkstra
- Gehört zur Klasse der Greedy-Algorithmen.
- Sukzessive wird der nächst beste Knoten in eine Ergebnismenge aufgenommen und aus der Menge der noch zu bearbeitenden Knoten entfernt.
- Es lässt sich eine Verwandtschaft zur Breitensuche feststellen.

Algorithmus von Dijkstra

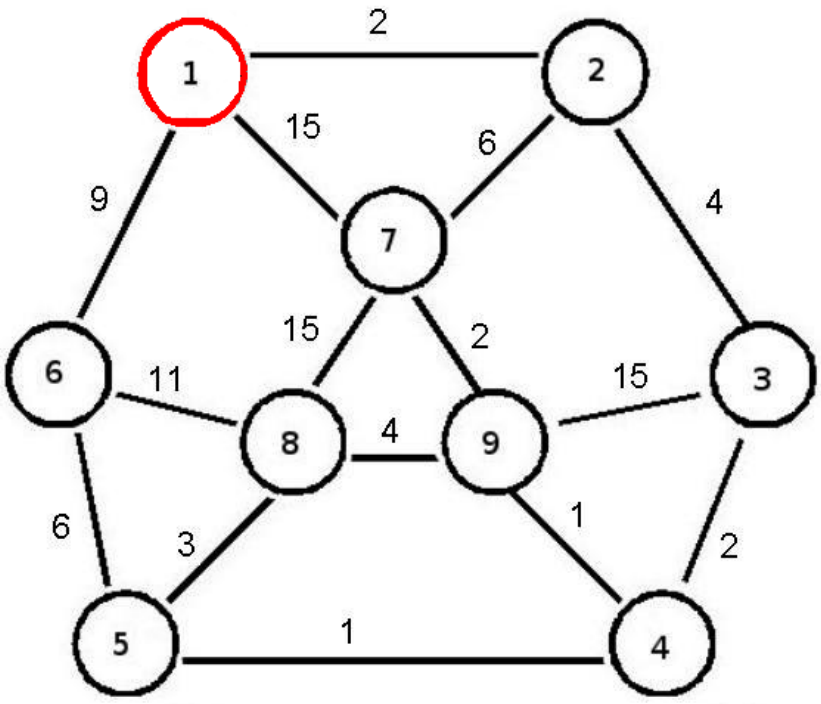
```
01  für jedes v aus V
02      Distanz(v) := unendlich, Vorgänger(v) := kein
03  Distanz(s) := 0, Vorgänger(s) := 'begin', U := V
04  M := EMPTYSET
05  für jedes (s,v) aus E mit v aus U
06      Distanz(v) := Kosten(s,v)
07      M := M UNION {v}
08      Vorgänger(v) := s
09
10  solange M nicht leer
11      wähle u aus M mit Distanz(u) minimal
12      M := M - {u}
13      U := U - {u}
14      wenn u = z dann STOP   # optional
15      für jedes (u,v) aus E mit v aus U
16          M := M UNION {v}
17          wenn Distanz(u) + Kosten(u,v) < Distanz(v) dann
18              Distanz(v) := Distanz(u) + Kosten(u,v)
19              Vorgänger(v) := u
```

Algorithmus von Dijkstra - Beispiel



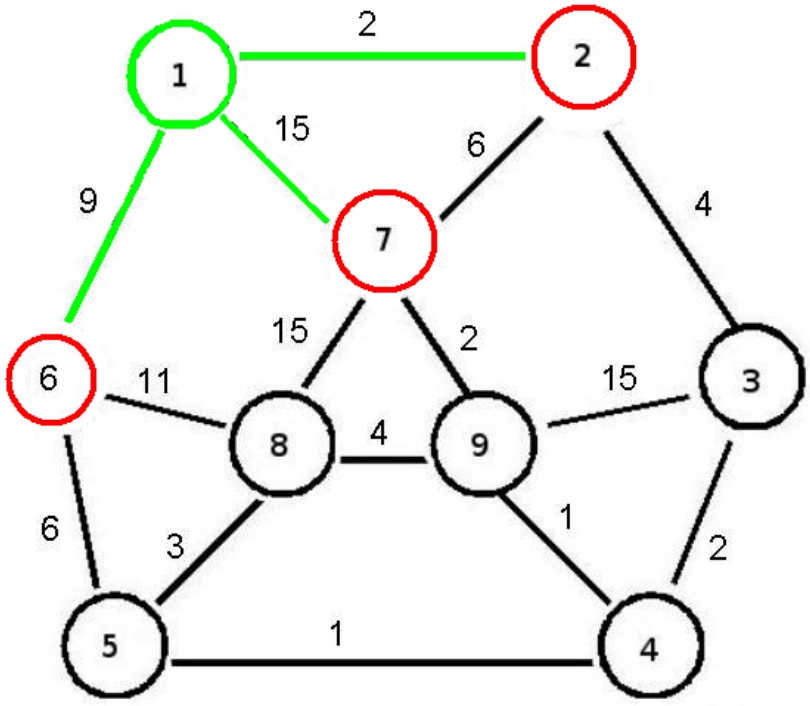
- Beispielhaft Berechnung des kürzesten Weges vom Knoten 1 zum Knoten 4

Algorithmus von Dijkstra - Beispiel



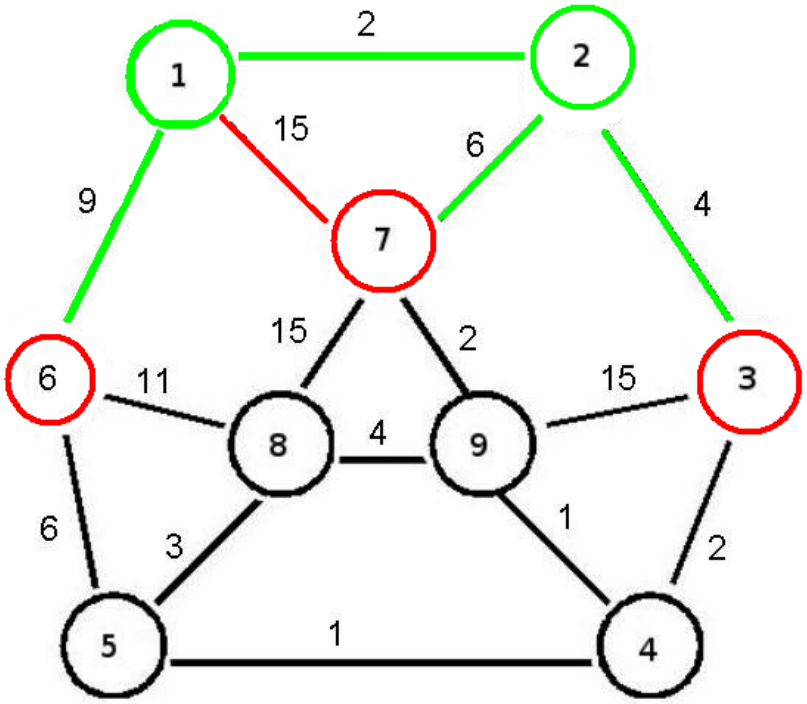
- 1. Schritt:
  - der Startknoten wird Rot gefärbt

Algorithmus von Dijkstra - Beispiel



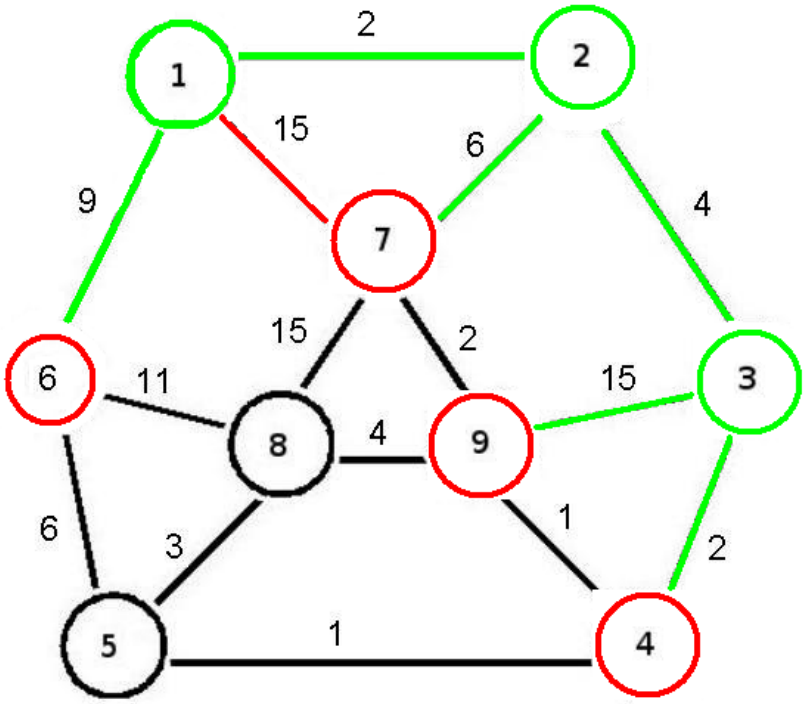
- 2. Schritt:
  - Dann wird der Startknoten grün gefärbt.
  - Seine Nachbarn 2, 6 und 7 werden rot gefärbt.
  - Die zu 2, 6 und 7 führenden Kanten werden grün, da es die kürzesten Wege zu diesen Knoten sind

Algorithmus von Dijkstra - Beispiel



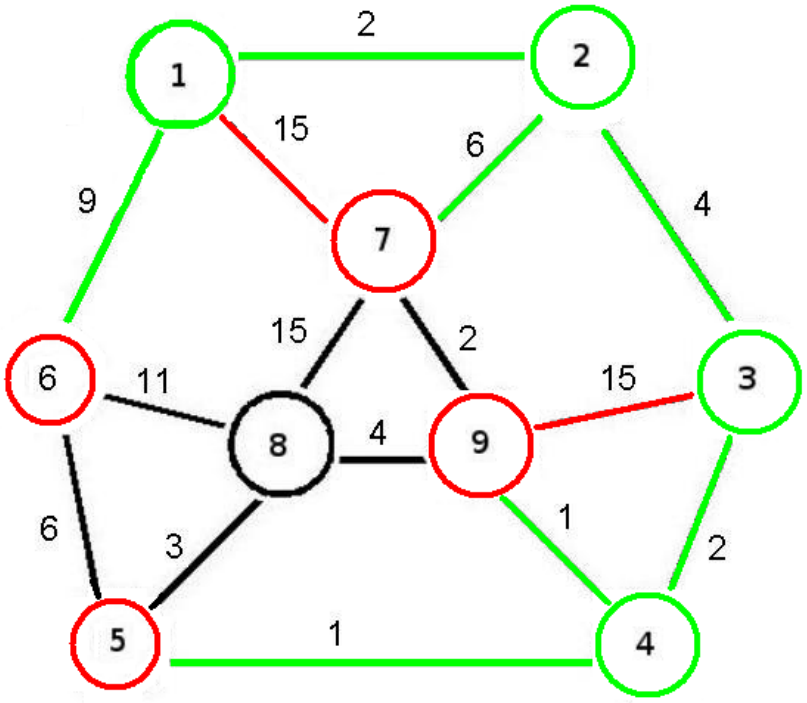
- 3. Schritt:
  - Der Knoten 2 ist der rote Knoten mit geringstem Abstand zu 1.
  - Er wird grün gefärbt.
  - Gleichzeitig wird sein Nachbar 3 rot.
  - Die Kante zu 3 wird grün, da sie auf dem bislang kürzesten Weg zu 3 liegt.
  - Die Kante zu 7 wird ebenfalls grün, da der neue Weg über 2 eine Länge von 8 aufweist, die bisherige Länge jedoch 15
  - Als Konsequenz dieser Grünfärbung mit neuer kürzester Länge wird die Kante von 1 zu 7 rot gefärbt

Algorithmus von Dijkstra - Beispiel



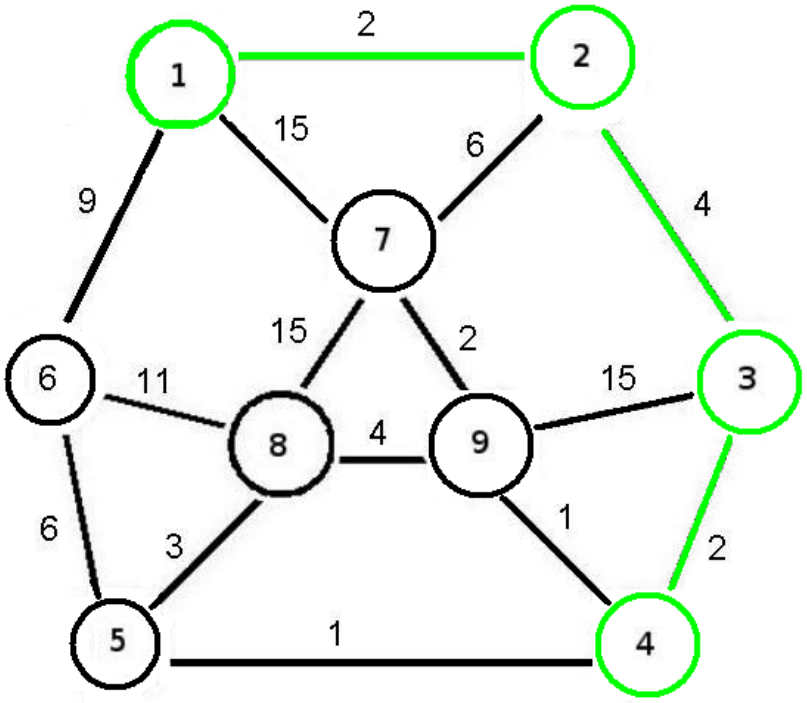
- 4. Schritt:
  - Knoten 3 ist nun der Knoten mit der minimalen Entfernung zu 1.
  - Dieser wird grün gefärbt
  - Seine Nachbarn 4 und 9 werden rot gefärbt
  - Kanten zu 4 und 9 erhalten grüne Färbung

Algorithmus von Dijkstra - Beispiel



- 5. Schritt:
  - Minimal sind 4 und 7
  - Es wird zufällig 4 gewählt.
  - Also wird 4 grün und 5 und 9 rot.
  - Die bisherige Entfernung 21 mit der 9 erreicht wird, wird mit der neuen Entfernung über 4, also mit 9 verglichen.
  - Da die neue kürzer ist, wird die Kante 4 nach 9 grün, die Kante 3 nach 9 rot,

Algorithmus von Dijkstra - Beispiel



- Kürzesten Weg gefunden
- Länge: 8
- Kantenfolge (1,2),(2,3),(3,4)

Algorithmus von Dijkstra – Einschränkungen + Komplexität

- Der Algorithmus von Dijkstra arbeitet nur für nichtnegative Kantengewichte korrekt.
- **Zeitkomplexität:**

  Im Folgenden sei m die Anzahl der Kanten und n die Anzahl der Knoten. Der Algorithmus von Dijkstra muss n-mal den nächsten minimalen Knoten u bestimmen. Eine Möglichkeit wäre jedes Mal diesen mittels Durchlaufen durch eine Knotenliste zu bestimmen. Die Laufzeit beträgt dann $O(n^2)$. Eine effizientere Möglichkeit zur Liste bietet die Verwendung eines Fibonacci-Heap. Die Laufzeit beträgt dann lediglich $O(m+n\log_n)$.

Bellmann-Ford Algorithmus

- nach seinen Erfindern Richard Bellmann und Lester Ford
- Gelegentlich wird auch vom **Moore-Bellman-Ford-Algorithmus** gesprochen, da auch Edward Moore zu seiner Entwicklung beigetragen hat.
- Hier auch negative Kantengewichte zugelassen

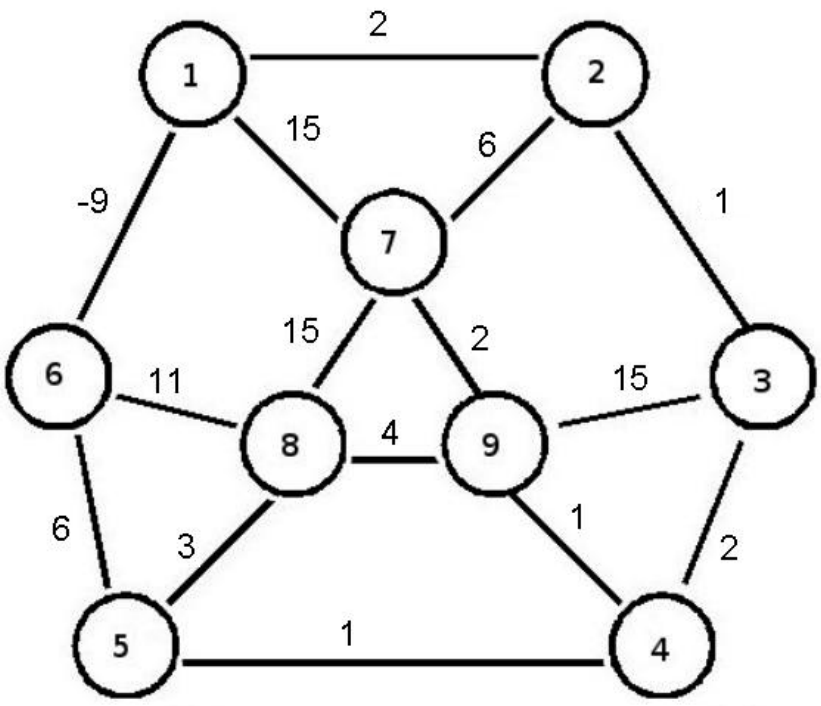
Bellmann-Ford Algorithmus

- Kann als Verallgemeinerung des Algorithmus von Dijkstra verstanden werden.
- Auch hier wird ein Teilgraph über den Ausgangsgraphen wachsen gelassen. Im Unterschied zu Dijkstra werden die Knoten zu keinem Zeitpunkt abschließend betrachtet. Die kürzeste Entfernung eines Knotens zum Ausgangsknoten steht also erst dann fest, wenn der Algorithmus endet.
- Falls ein Knoten vom Startknoten aus nicht erreichbar ist, wird der Abstand formal als unendlich gesetzt.

Bellmann-Ford Algorithmus

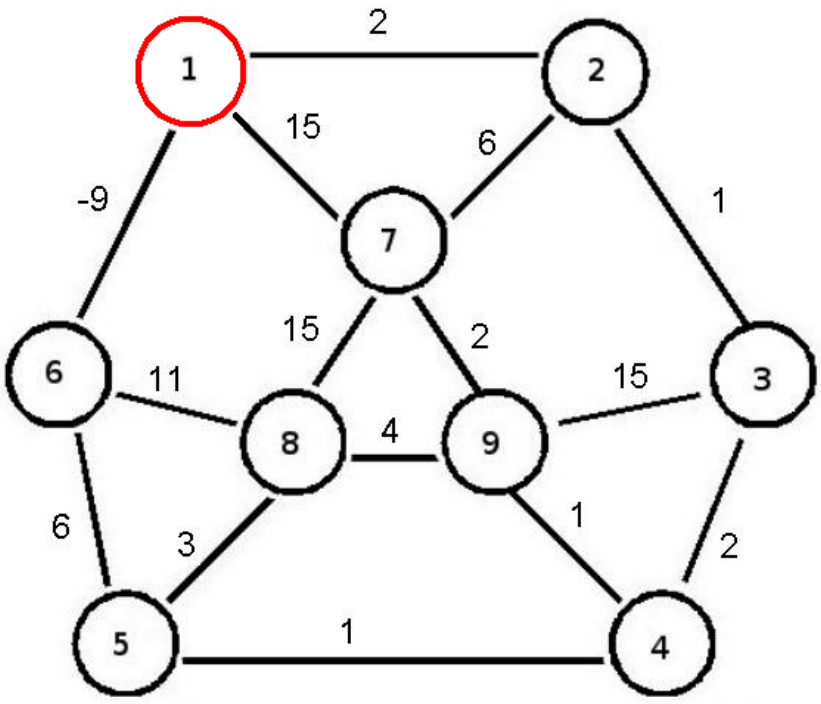
```
01  für jedes v aus V
02      Distanz(v) := unendlich, Vorgänger(v) := kein
03  Distanz(s) := 0
04  wiederhole n - 1 mal
05      für jedes (u,v) aus E
06          wenn Distanz(u) + Gewicht(u,v) < Distanz(v)
07          dann
08              Distanz(v) := Distanz(u) + Gewicht(u,v)
09              Vorgänger(v) := u
10  für jedes (u,v) aus E
11      wenn Distanz(u) + Gewicht(u,v) < Distanz(v) dann
12          STOP mit Ausgabe "Zyklus negativer Länge gefunden"
13  Ausgabe Distanz
```

Bellmann-Ford Algorithmus - Beispiel



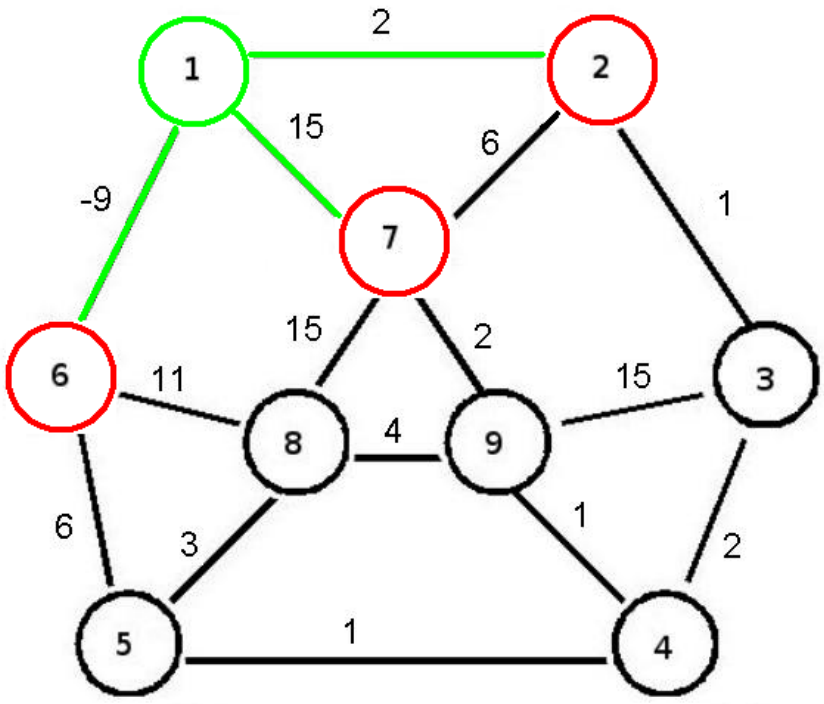
- Beispielhaft Berechnung des kürzesten Weges vom Knoten 1 zum Knoten 4

Bellmann-Ford Algorithmus - Beispiel



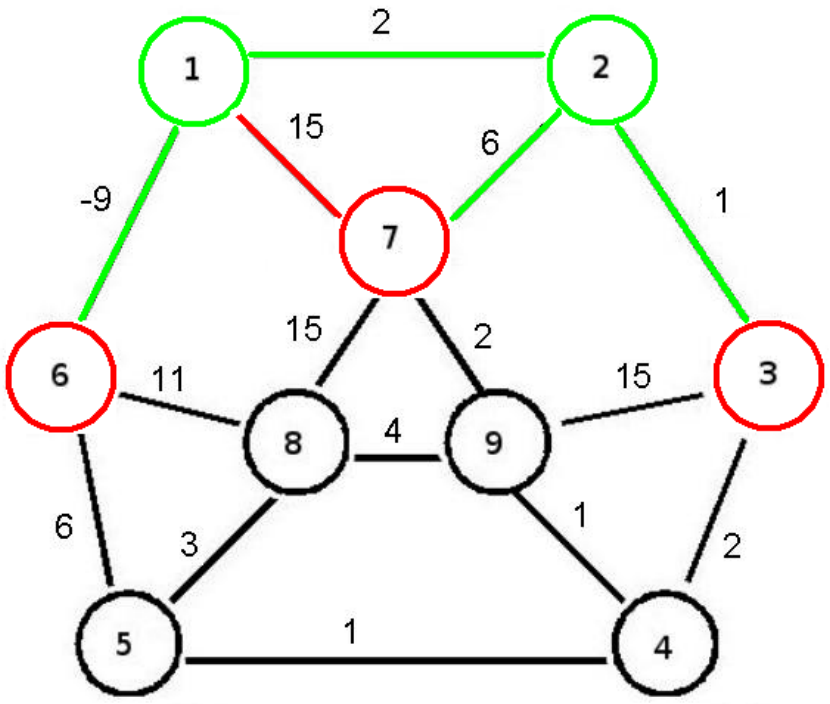
- Ausgangsknoten wird rot gefärbt

Bellmann-Ford Algorithmus - Beispiel



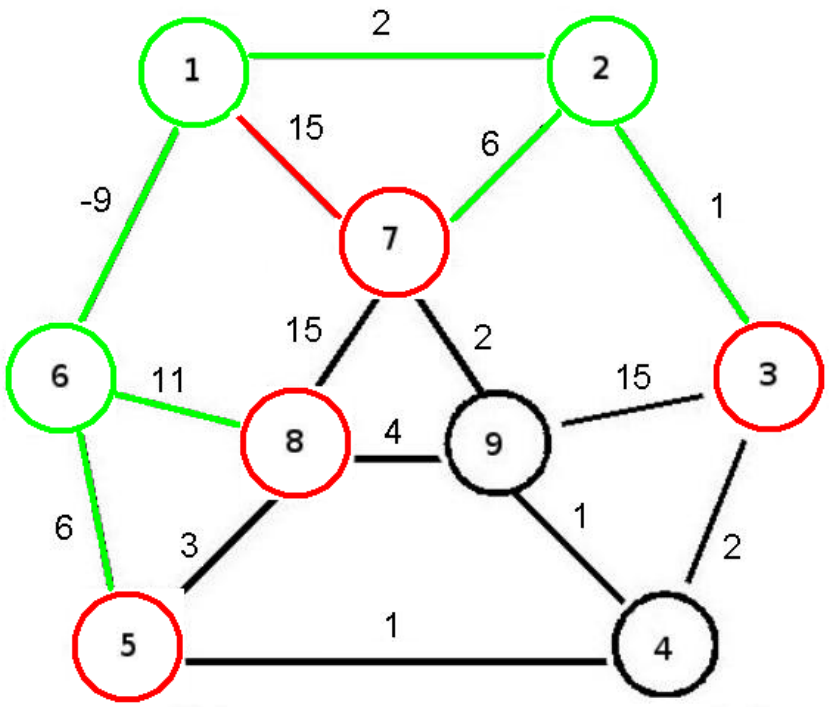
- Es werden vom einzigen roten Knoten die Kanten zu den Nachbarknoten 2, 6 und 7 betrachtet.
- Dabei wird überprüft, ob ein kürzerer Weg entsteht als bisher.
- Das ist bei allen der Fall
- Sie werden also rot gefärbt
- Für die Entfernung von Knoten 1 gab es keine Änderung, er wird grün gefärbt

Bellmann-Ford Algorithmus - Beispiel



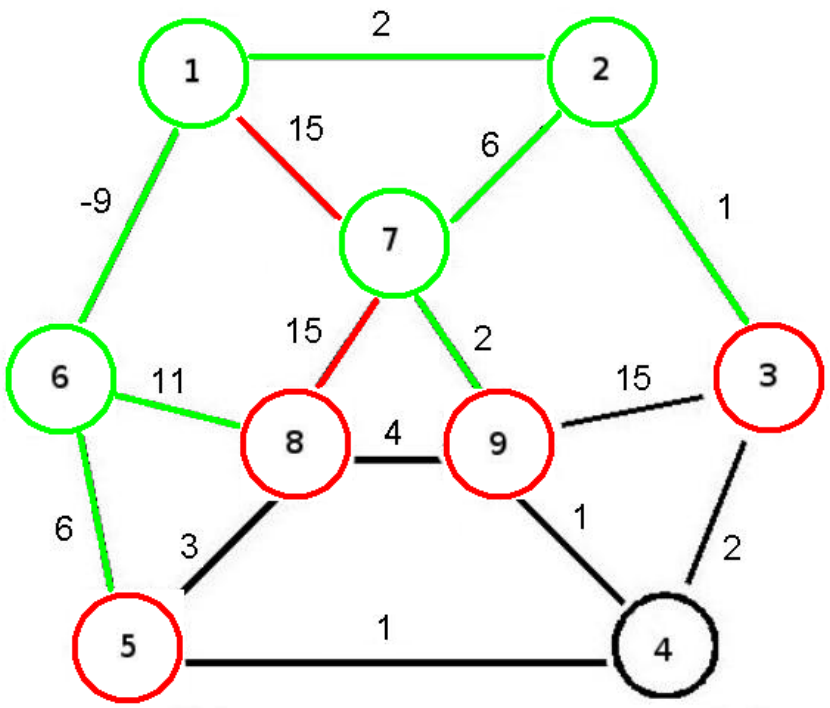
- Jetzt sind die Knoten 2, 6 und 7 rot.
- Sie werden nacheinander besucht, beginnend mit Knoten 2.
- Damit wird 2 grün.
- Knoten 3 wird erstmalig erreicht und rot.
- Knoten 7 wird über 2 kürzer erreicht. Er bleibt deshalb rot

Bellmann-Ford Algorithmus - Beispiel



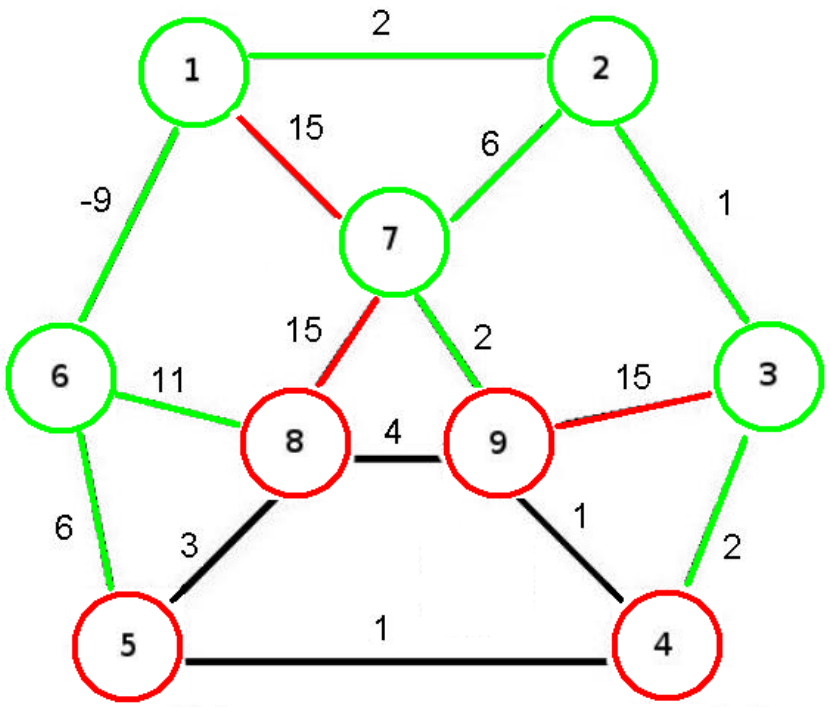
- Knoten 6:
  - Er wird grün.
  - Knoten 5 und 8 werden rot, da sie das erste mal besucht werden

Bellmann-Ford Algorithmus - Beispiel



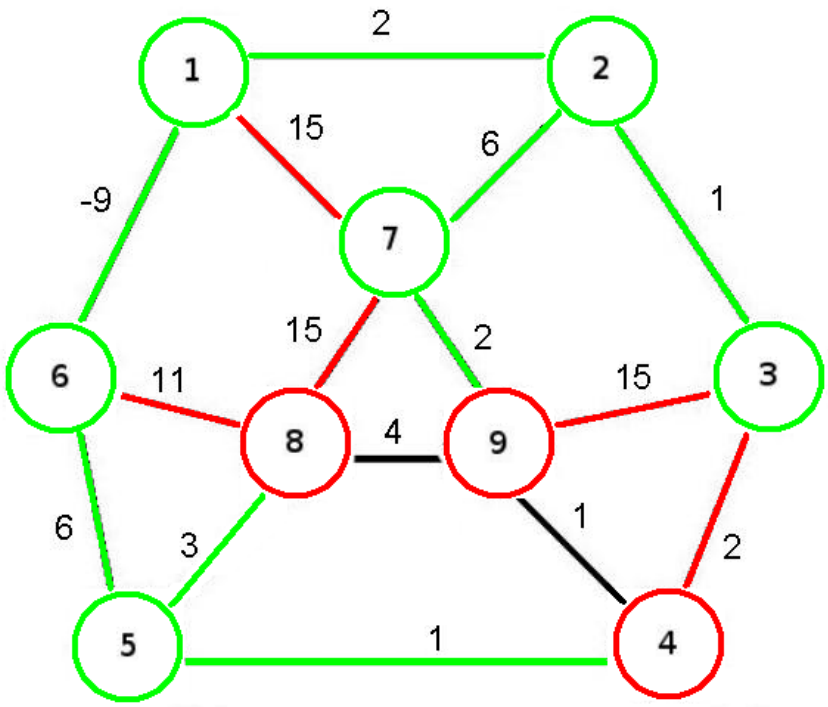
- Knoten 7:
  - Er wird grün.
  - Für Knoten 8 bleibt der Weg über 6 der kürzeste.
  - Knoten 9 wird erstmalig erreicht und rot

Bellmann-Ford Algorithmus - Beispiel



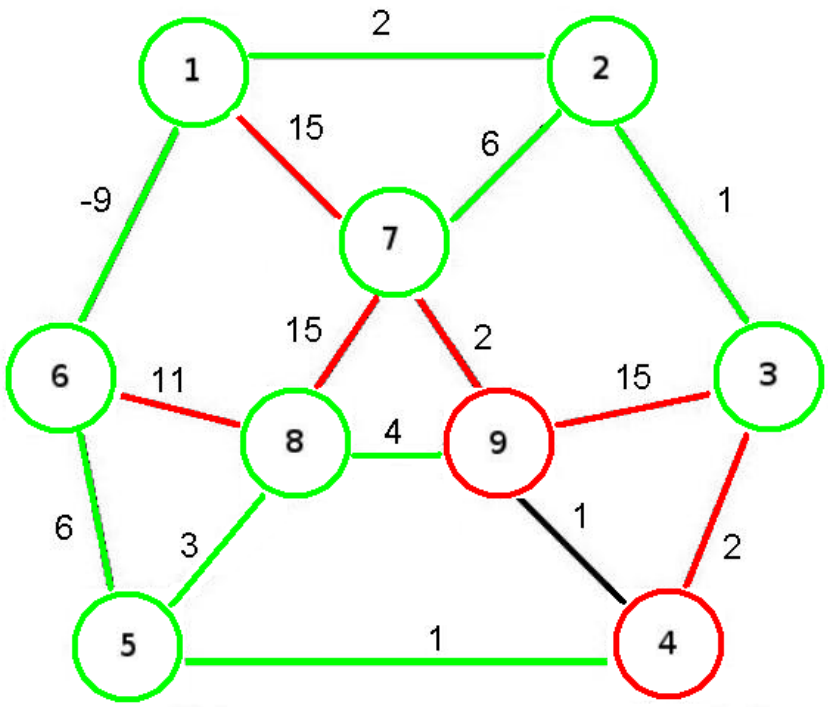
- 3 wird also grün.
- Für Knoten 4 geht der erste und damit kürzeste Weg über 3
- Er wird rot.
- Für Knoten 9 ändert sich nichts. Er bleibt rot

Bellmann-Ford Algorithmus - Beispiel



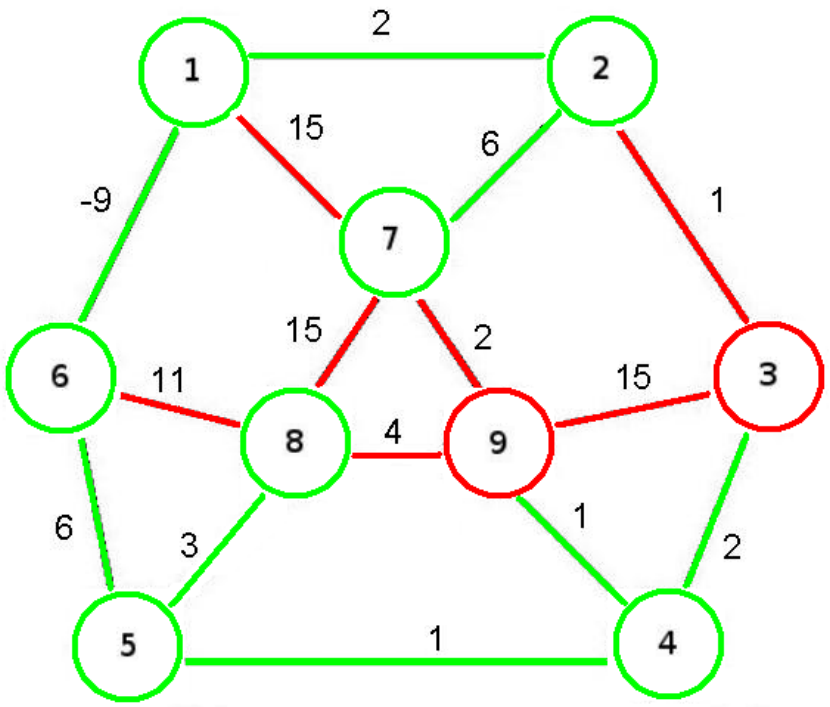
- Knoten 5:
  - Seine Nachbarn 4 und 8 werden beide kürzer erreicht, mit neuer Entfernung über 5:
  - Knoten 5 wird grün, die beiden anderen bleiben rot

Bellmann-Ford Algorithmus - Beispiel



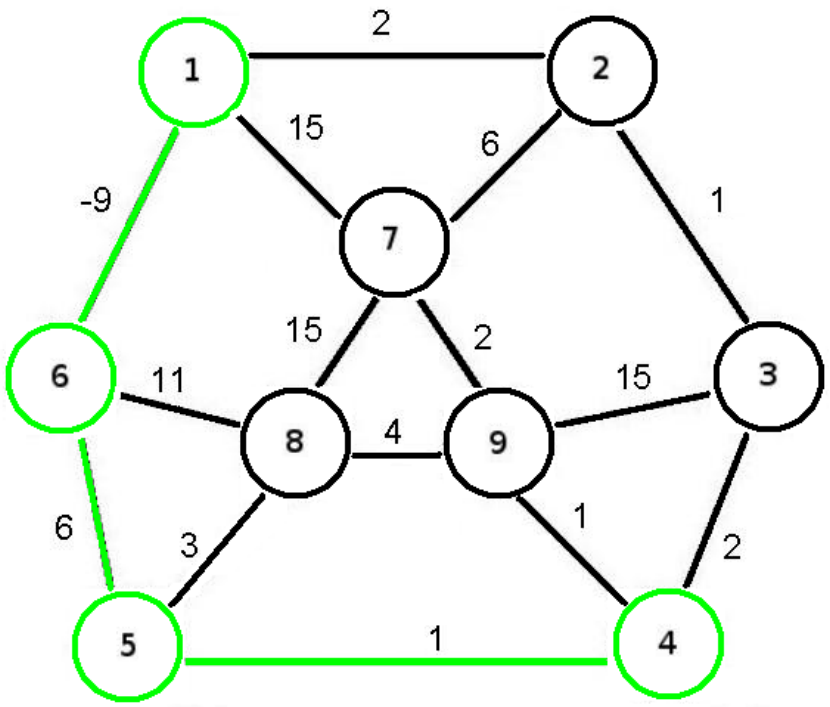
- Knoten 8:
  - wird grün.
  - Sein Nachbar 7 bleibt grün
  - Wohingegen 9 kürzer über 8 erreicht wird.
  - Er bleibt also rot

Bellmann-Ford Algorithmus - Beispiel



- Knoten 4 wird grün.
- Seine Nachbarn 3 und 9 werden über 4 kürzer als bisher erreicht.
- Sie werden beide rot

Bellmann-Ford Algorithmus - Beispiel



- Kürzesten Weg gefunden
- Länge: -2
- Kantenfolge (1,6),(6,5),(5,4)

Bellmann-Ford Algorithmus – Einschränkungen + Komplexität

- Es darf im Graphen keinen Zyklus mit negativer Länge geben
- Die Laufzeit des Algorithmus ist in O(n·m), wobei *n* die Anzahl der Knoten und *m* die Anzahl der Kanten im Graphen sind.

Floyd-Warshall Algorithmus

- (auch *Tripel-Algorithmus*)
- benannt nach Robert Floyd und Stephen Warshall
- Eigentlich 2 Algorithmen:
  - Er findet die Länge der kürzesten Wege zwischen allen Paaren von Knoten eines gewichteten Graphen in der Version Floyds
  - oder die reflexive, transitive Hülle eines Graphen in der Version Warshalls.
  - Beide Versionen wurden im Jahr 1962 vorgestellt und gehen eigentlich zurück auf einen Algorithmus von Stephen Kleene aus dem Jahr 1956.

Floyd-Warshall Algorithmus

- 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.
- Nimmt man also an, man kennt schon die kürzesten Wege zwischen allen Knotenpaaren, die nur über Knoten mit Index kleiner als *k* führen
- Und man sucht alle kürzesten Wege über Knoten mit Index kleiner oder gleich *k*, dann hat man für einen Pfad von *u* nach *v* zwei Möglichkeiten:
  - Entweder er geht über den Knoten *k*, dann setzt er sich zusammen aus schon bekannten Pfaden von *u* nach *k* und von *k* nach *v*
  - Oder es ist der schon bekannte Weg von *u* nach *v* über Knoten kleiner als *k*.

Floyd - Algorithmus

- der Graph ist gegeben durch seine Gewichtsmatrix *w*.
- *w[i,j]* ist das Gewicht der Kante von *i* nach *j*, falls eine solche Kante existiert.
  - *w[i,j]* ist unendlich, falls es keine Kante von *i* nach *j* gibt.
- Dann kann man die Matrix *d* der kürzesten Distanzen durch folgendes Verfahren bestimmen:
- (1) Für alle i,j : d[i,j] = w[i,j]
- (2) Für k = 1 bis *n*
- (3) Für alle Paare *i,j*
- (4) d[i,j] = min (d[i,j],d[i,k] + d[k,j])

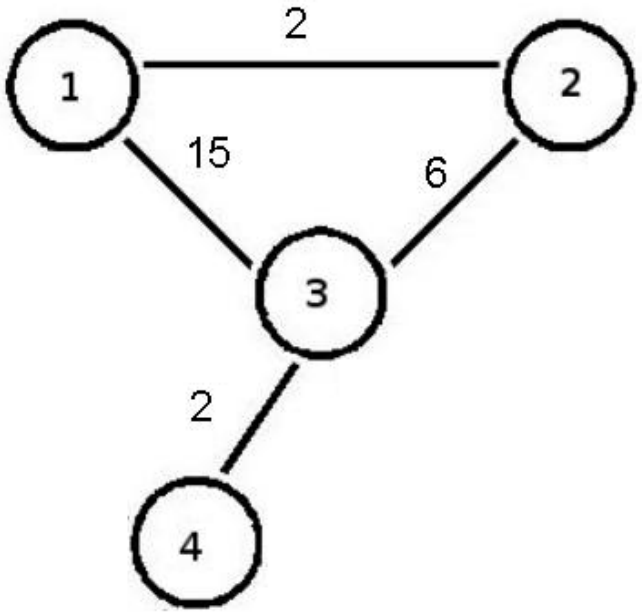
Warshall - Algorithmus

- *w* ist die Adjazenzmatrix, das heißt *w[i,j]* ist 1 falls eine Kante von *i* nach *j* existiert, 0 falls keine Kante existiert.
- Die Matrix *d* wird so berechnet, dass *d[i,j]* gleich 1, genau dann, wenn ein Pfad von *i* nach *j* existiert:
- (1) Für alle i,j : d[i,j] = w[i,j]
- (2) Für k = 1 bis *n*
- (3) Für alle Paare *i,j*
- (4) Falls d[i,j] = 0
- (5) d[i,j] = d[i,k] * d[k,j]

Floyd-Warshall Algorithmus

- Der Floyd-Algorithmus berechnet in dieser Form nur die Länge des kürzesten Weges.
- Um den kürzesten Weg selbst zu konstruieren, wird parallel zu *A* eine weitere Matrix *F* geführt, in der als Eintrag *F[i,j]* jeweils der vorherige Knoten auf dem kürzesten Weg von *i* nach *j* steht.
- Jedesmal, wenn der Floyd-Algorithmus einen kürzeren Weg von *i* nach *j* als den bisher bekannten findet, wird *F[i,j]* aktualisiert.

Floyd-Warshall Algorithmus - Beispiel



- Beispielhaft Berechnung des kürzesten Weges vom Knoten 1 zum Knoten 4
- 1. Schritt
  - Die Wichtungsmatrix:

W[i,j]	1	2	3	4
1	0	2	15	inf
2	2	0	6	inf
3	15	6	0	2
4	inf	inf	2	0

Floyd-Warshall Algorithmus - Beispiel

W[i,j]	1	2	3	4
1	0	2	8	inf
2	2	0	6	inf
3	8	6	0	2
4	inf	inf	2	0

F[i,j]	1	2	3	4
1	X	1	2	
2	2	X	2	
3	2	3	X	3
4			4	x

- Berechnung mit k = 1 ergibt keine Änderung
- Berechnung mit k = 2 ergibt neuen kürzesten Weg von 1 nach 3 über 2

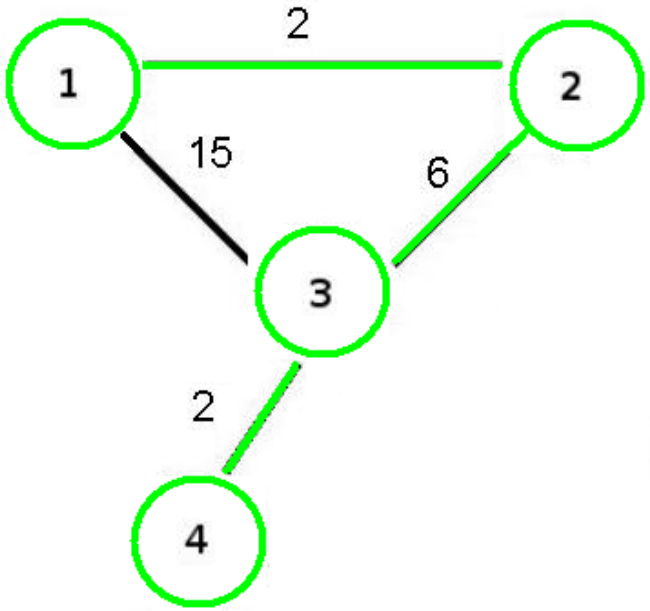
Floyd-Warshall Algorithmus - Beispiel

W[i,j]	1	2	3	4
1	0	2	8	10
2	2	0	6	8
3	8	6	0	2
4	10	8	2	0

F[i,j]	1	2	3	4
1	X	1	2	3
2	2	X	2	3
3	2	3	X	3
4	3	3	4	x

- Berechnung mit k = 3 ergibt neue kürzeste Wege:
  - von 1 nach 4 über 3 und
  - Von 2 nach 4 über 3
- Für k = 4 gibt es keine Veränderung

Floyd-Warshall Algorithmus - Beispiel



- Kürzesten Weg gefunden
- Länge: 10
- Kantenfolge (1,2),(2,3),(3,4)

Floyd-Warshall Algorithmus – Einschränkungen + Komplexität

- Der Floyd-Algorithmus funktioniert auch, wenn die Kanten negatives Gewicht haben können, allerdings werden Zyklen mit negativer Länge nicht erkannt.
- Die Laufzeit des Floyd-Warshall-Algorithmus ist $O(n^3)$

Quellen

- de.wikipedia.org
- www.mcgods.de
- http://www.iti.fh-flensburg.de/lang/algorithmen/graph/warshall.htm
- „Graphentheorie“, Reinhard Diestel, Springer Verlag 2000

Folien

- Die Folien stehen als pdf bzw. ppt unter
- www.cs.uni-magdeburg.de/~duvignea/praesi.ppt
- www.cs.uni-magdeburg.de/~duvignea/praesi.pdf