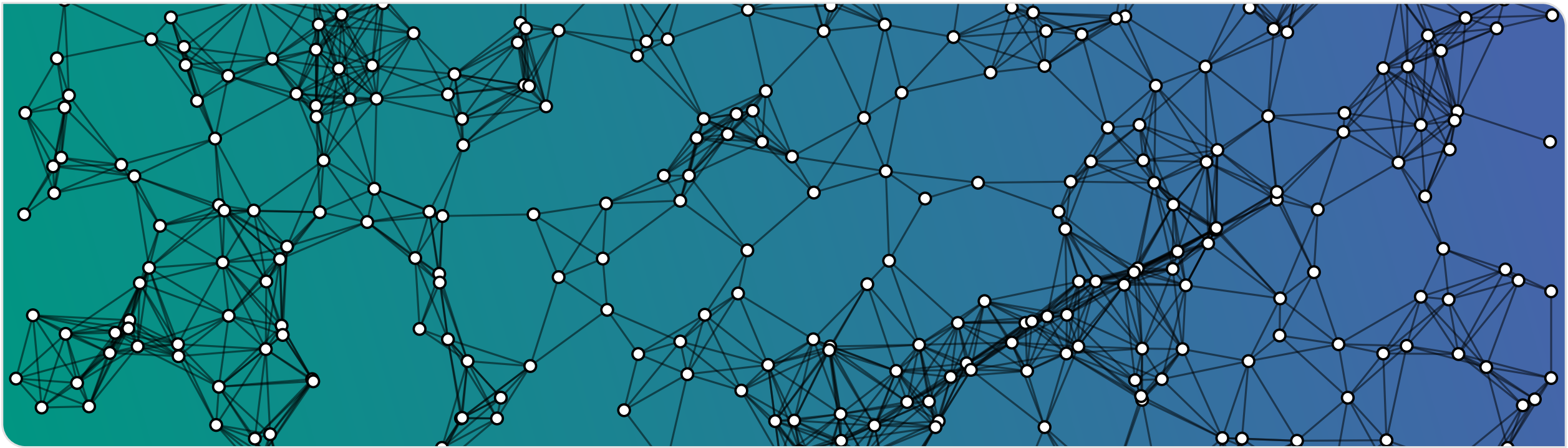
Karlsruher Institut für Technologie

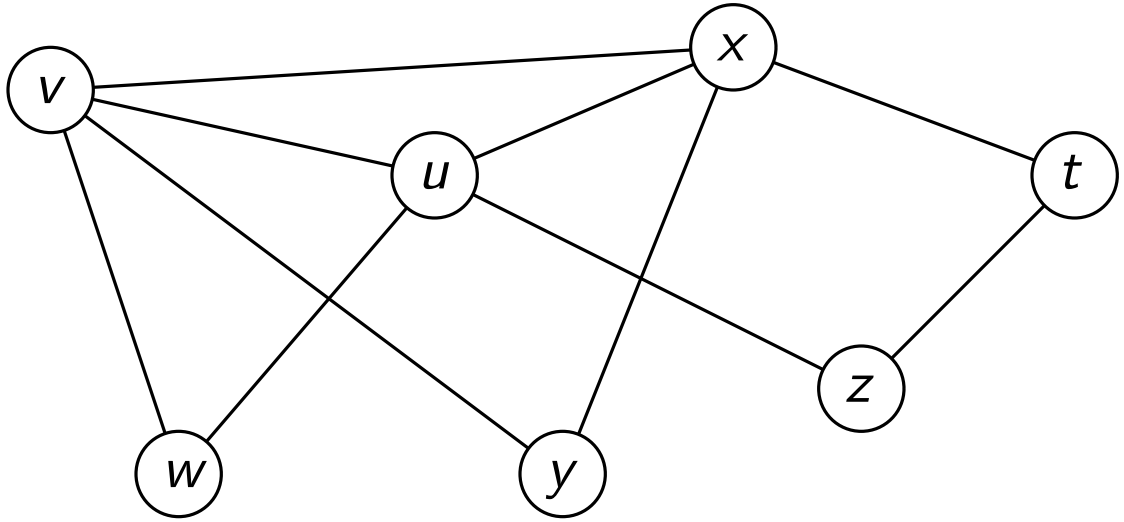
Algorithmen Zusatztutorium Woche 4

Graphen, kürzeste Wege und minimale Spannbäume



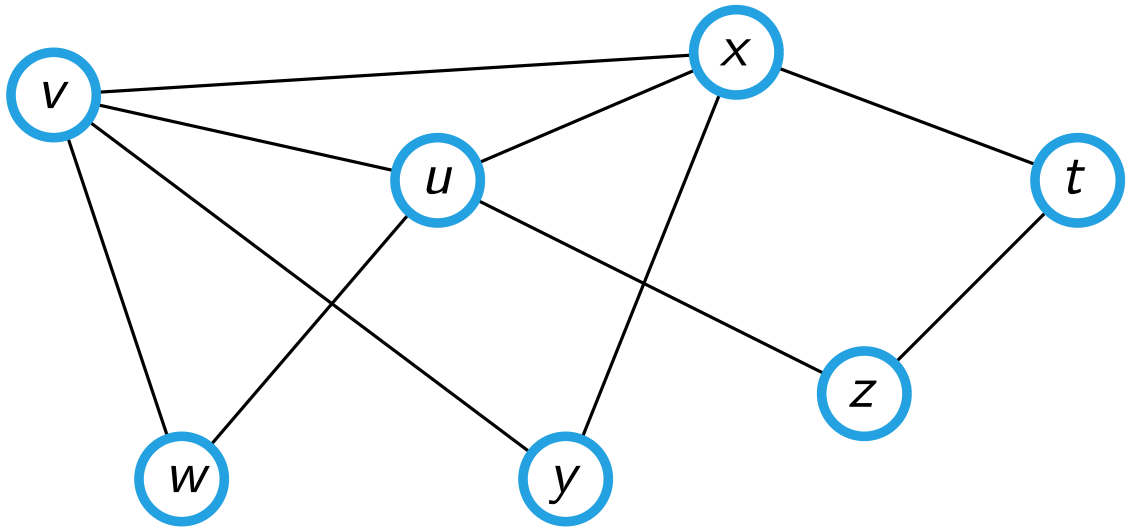
Graphen: Definition

Graph G besteht aus



Graphen: Definition

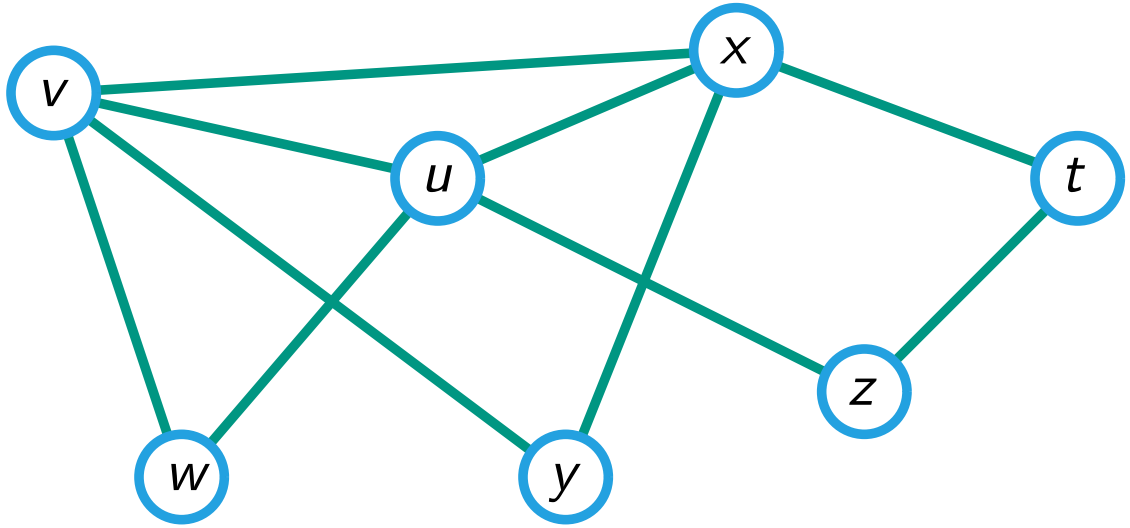
Graph G besteht aus



Knotenmenge V $= \{t, u, v, w, x, y, z\}$

Graphen: Definition

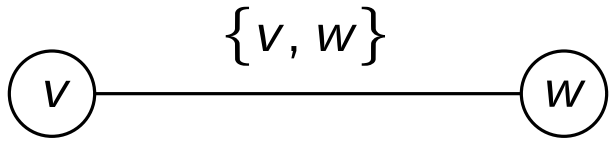
Graph G besteht aus



Knotenmenge V $= \{t, u, v, w, x, y, z\}$ **Kantenmenge E** $= \{\{v, w\}, \{v, u\}, \{v, x\}, \{v, y\}, \{w, u\}, \{u, x\}, \{y, x\}, \{u, z\}, \{x, t\}, \{x, t\}\}$

Kantenrichtung und Teilgraphen: Definition

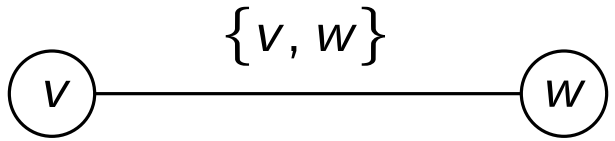
Kanten können **ungerichtet**



In beide Richtungen traversierbar

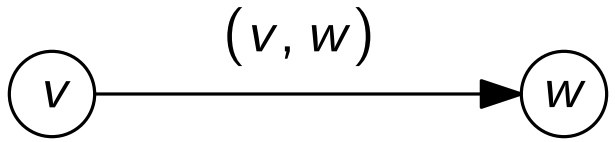
Kantenrichtung und Teilgraphen: Definition

Kanten können **ungerichtet**



In beide Richtungen traversierbar

oder **gerichtet** sein

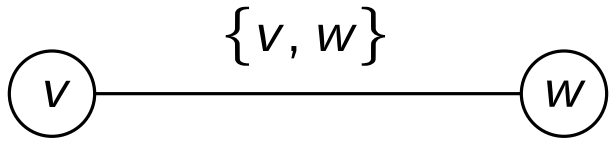


Nur in einer Richtung traversierbar

z.B Einbahnstraßen

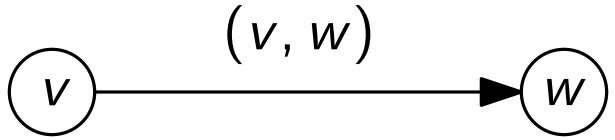
Kantenrichtung und Teilgraphen: Definition

Kanten können **ungerichtet**



In beide Richtungen traversierbar

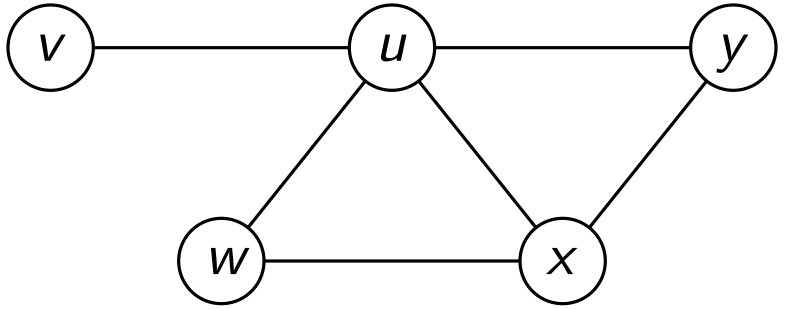
oder **gerichtet** sein



Nur in einer Richtung traversierbar

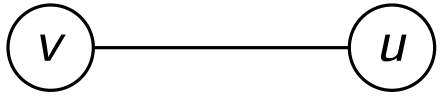
z.B Einbahnstraßen

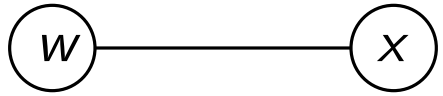
Teilgraph T von Graph **G** ist



Teilmenge $W \subseteq V$ der Knoten

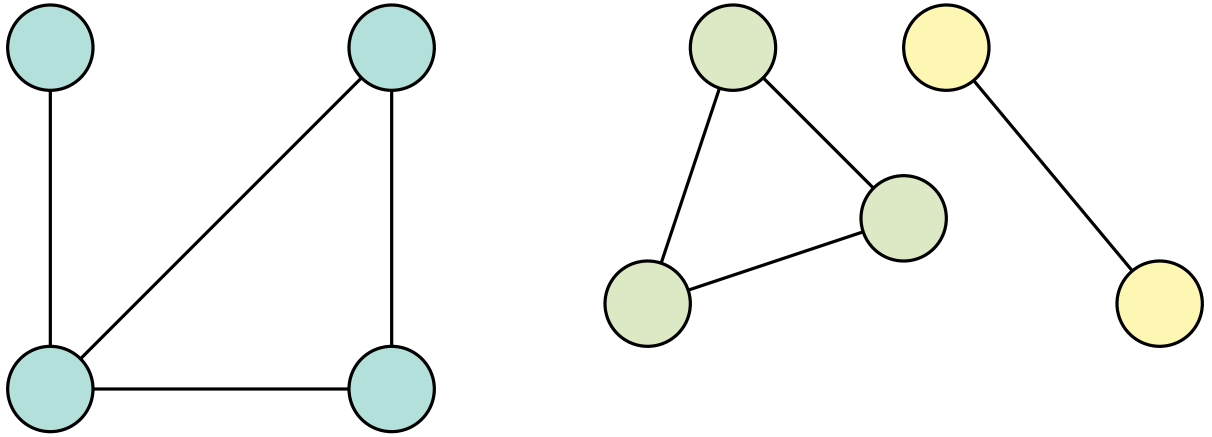
Teilmenge $F \subseteq E$ der Kanten





Zusammenhangskomponenten: Definition

ungerichteter Graph G

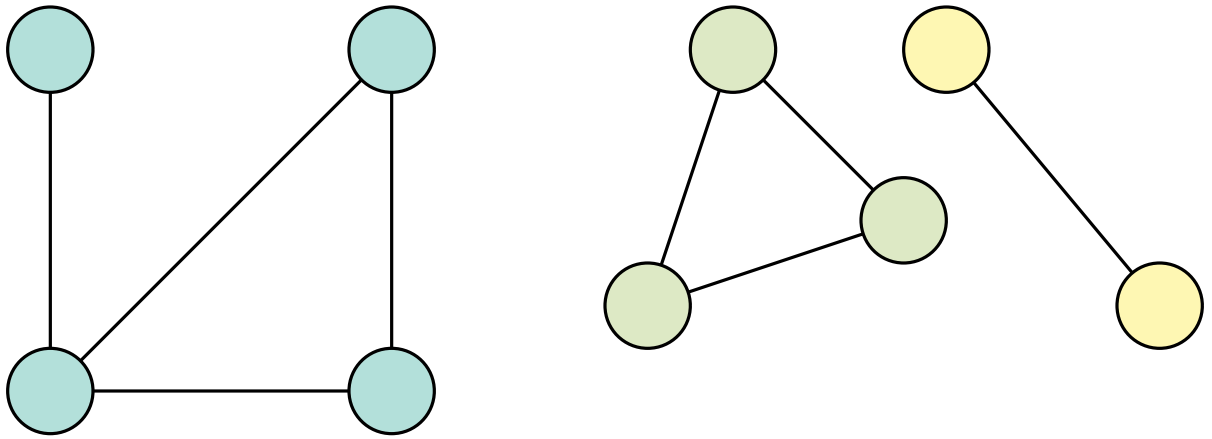


Zusammenhangskomponente von v:

Teilgraph aller von v erreichbaren Knoten und Kanten

Zusammenhangskomponenten: Definition

ungerichteter Graph G

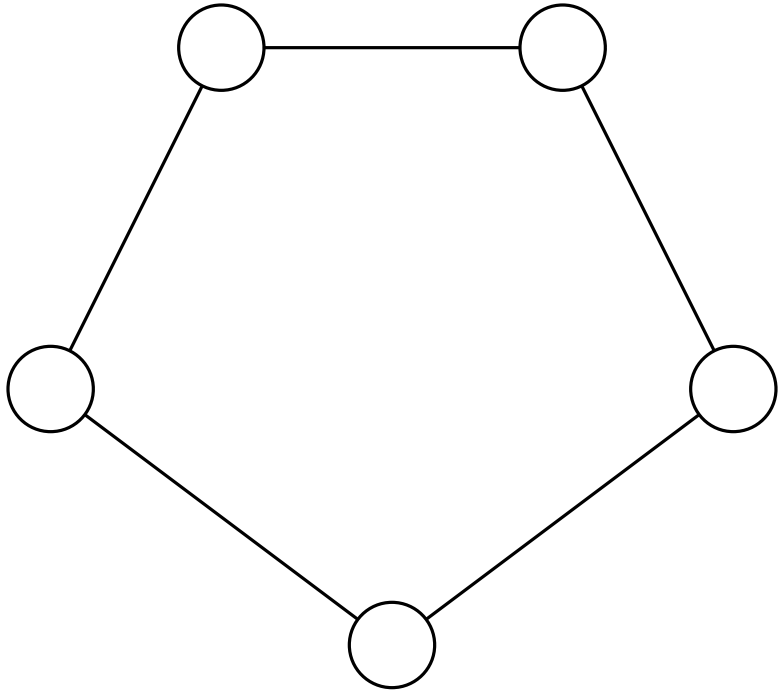


Zusammenhangskomponente von v:
Teilgraph aller von v erreichbaren Knoten und Kanten

Graph **unzusammenhängend** $\iff$ Graph hat mehr als eine Zusammenhangskomponente

Spezielle Graphentypen

gerichteter / ungerichteter **Kreis** ist

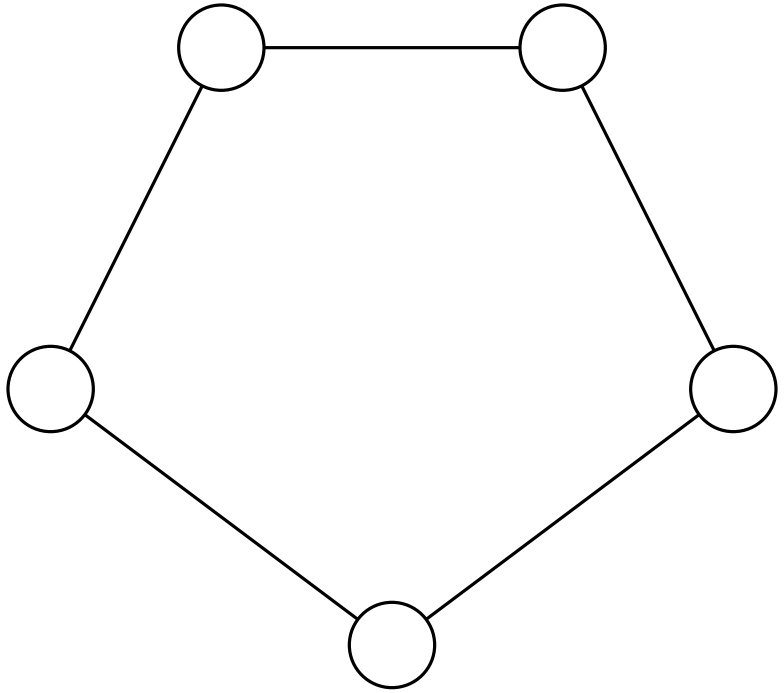


Graph mit Knoten $v_1, \ldots, v_n$

Kanten $\{v_1, v_2\}, \ldots, \{v_{n-1}, v_2\}, \{v_n, v_1\}$

Spezielle Graphentypen

gerichteter / ungerichteter **Kreis** ist

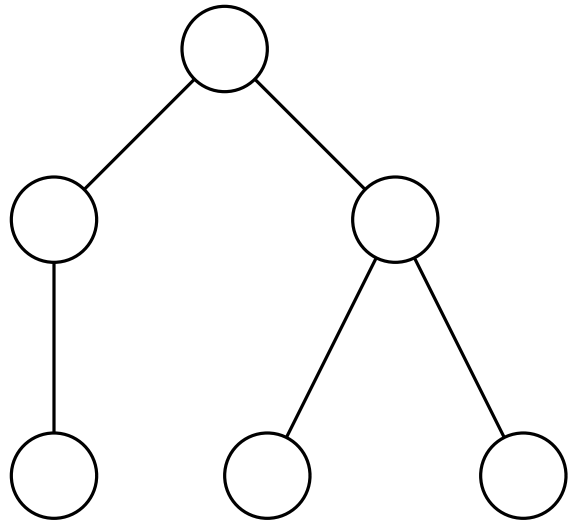


Graph mit Knoten $v_1, \ldots, v_n$

Kanten $\{v_1, v_2\}, \ldots, \{v_{n-1}, v_2\}, \{v_n, v_1\}$

Baum ist Graph

mit folgenden Eigenschaften:

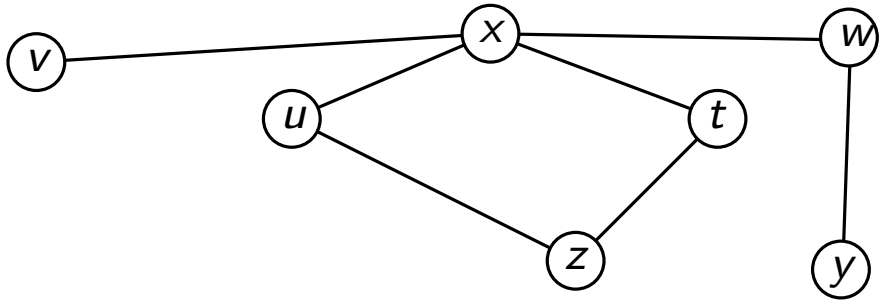


kreisfrei: Kein Kreis als Teilgraph

zusammenhängend: Nur eine Zusammenhangskomponente

Wege und Pfade: Definition

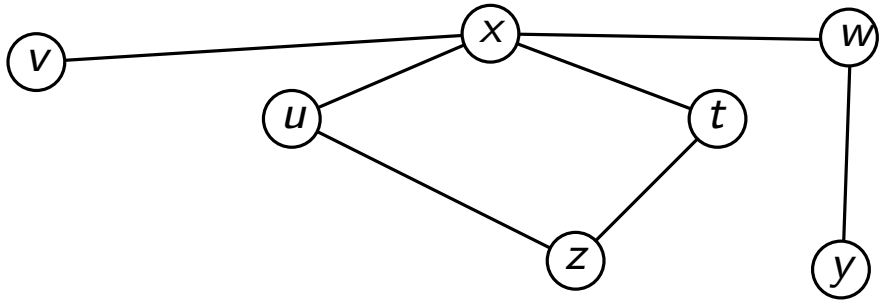
Weg W ist Folge von Knoten $(v_1, \ldots, v_j)$ sodass $\{v_i, v_{i+1}\}$ Kante ist



z.B (v, x, t, z, u, x, w, y)

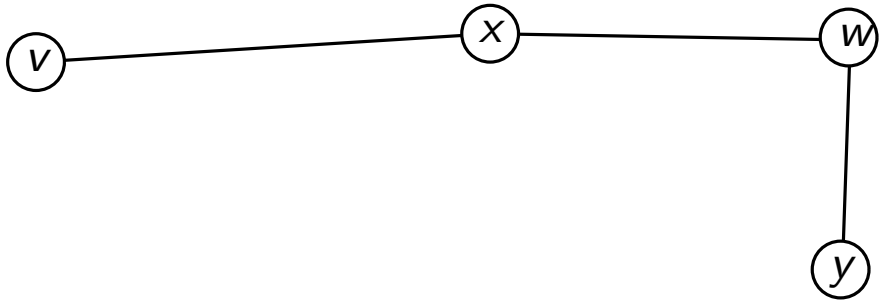
Wege und Pfade: Definition

Weg W ist Folge von Knoten $(v_1, \ldots, v_j)$ sodass $\{v_i, v_{i+1}\}$ Kante ist



z.B (v, x, t, z, u, x, w, y)

Pfad P ist Weg ohne doppelte Knoten



z.B (v, x, w, y)

Graphen in der echten Welt

Situationen der echten Welt lassen sich als Graphen modellieren

z.B Straßennetze

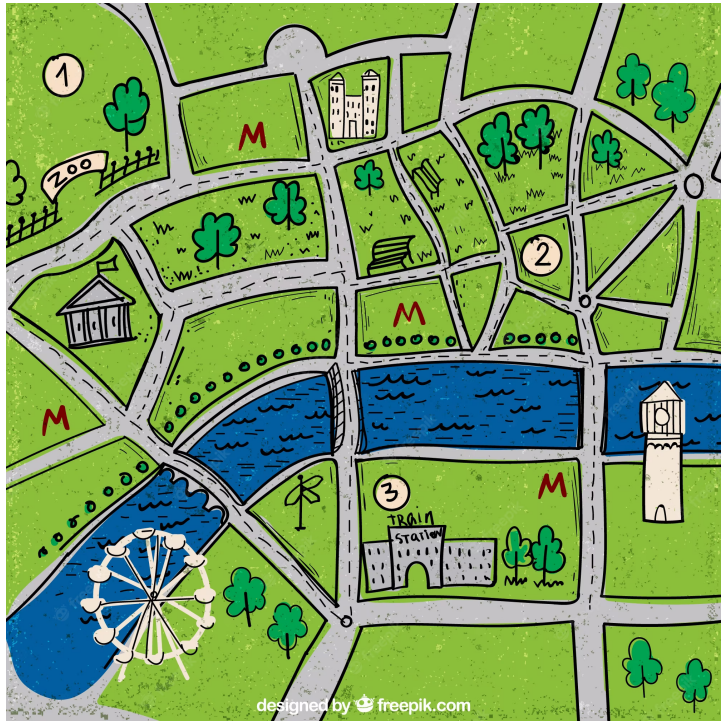


Bild von freepik.com

Graphen in der echten Welt

Situationen der echten Welt lassen sich als Graphen modellieren

z.B Straßennetze

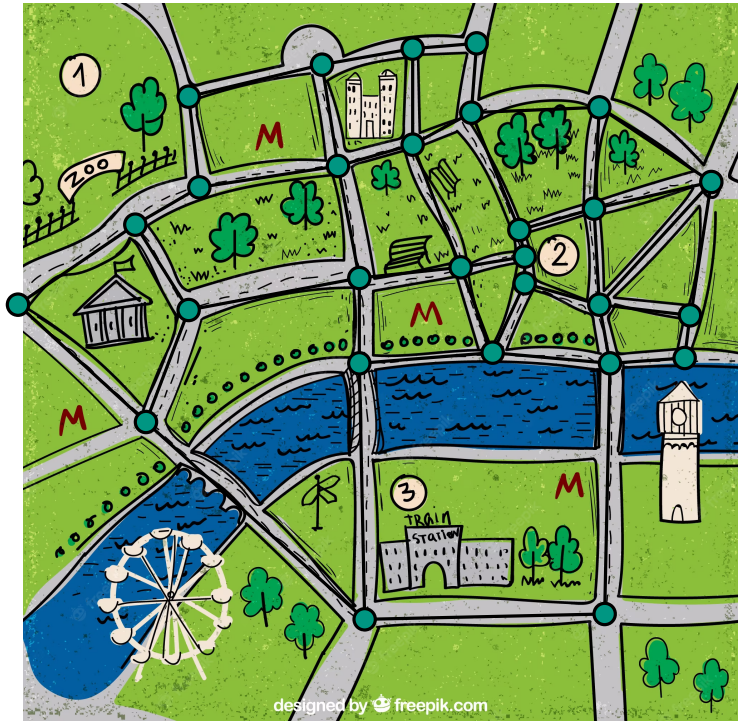
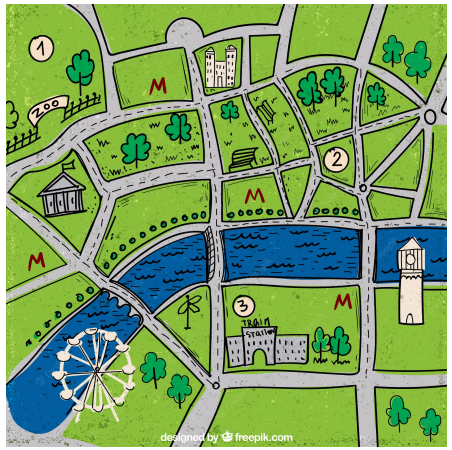


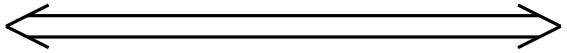
Bild von freepik.com

Graphen in der echten Welt

Situationen der echten Welt lassen sich als Graphen modellieren

z.B Straßennetze





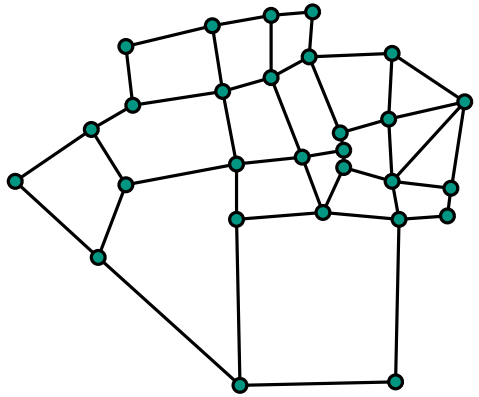
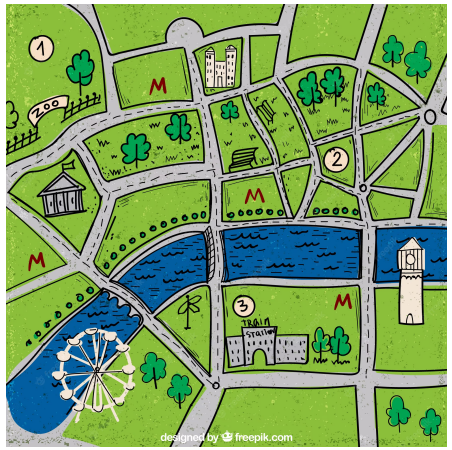


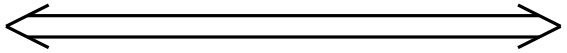
Bild von freepik.com

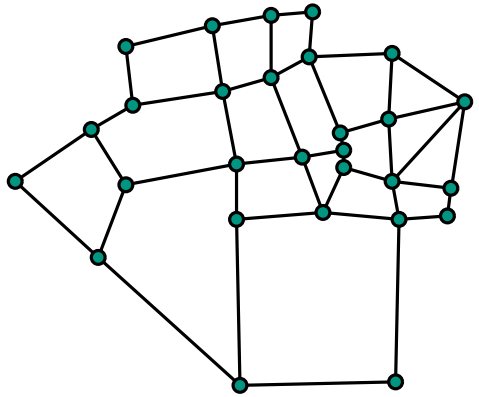
Graphen in der echten Welt

Situationen der echten Welt lassen sich als Graphen modellieren

z.B Straßennetze







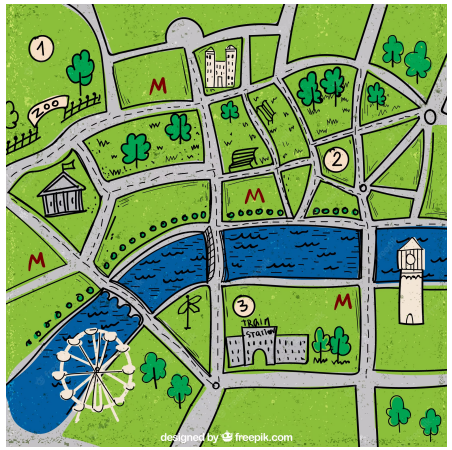
Probleme der echten Welt, lassen sich als Probleme auf Graphen formulieren

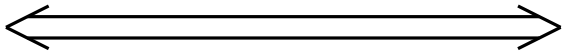
Bild von freepik.com

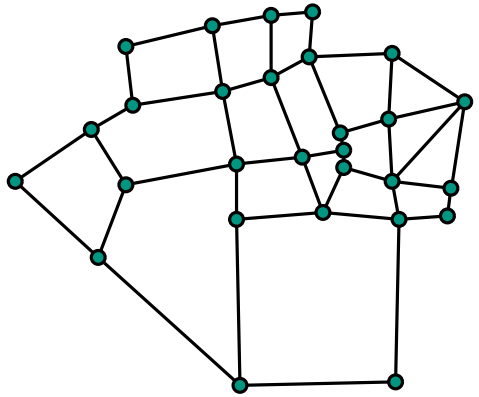
Graphen in der echten Welt

Situationen der echten Welt lassen sich als Graphen modellieren

z.B Straßennetze







Probleme der echten Welt, lassen sich als Probleme auf Graphen formulieren

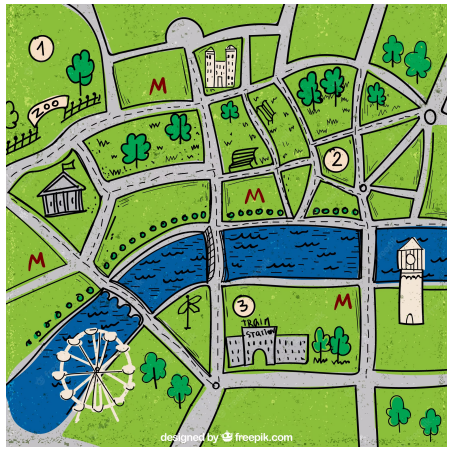
Wie komme ich am schnellsten von A nach B? $\Leftrightarrow$ Was ist der kürzeste Weg von Knoten A nach Knoten B?

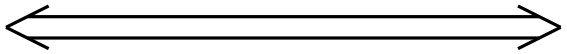
Bild von freepik.com

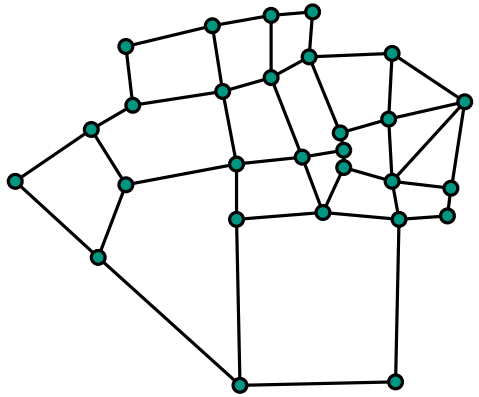
Graphen in der echten Welt

Situationen der echten Welt lassen sich als Graphen modellieren

z.B Straßennetze







Probleme der echten Welt, lassen sich als Probleme auf Graphen formulieren

Wie komme ich am schnellsten von A nach B? $\Leftrightarrow$ Was ist der kürzeste Weg von Knoten A nach Knoten B?

Was ist das günstigte Stromnetz? $\Leftrightarrow$ Was ist der minimale Spannbaum?

Bild von freepik.com

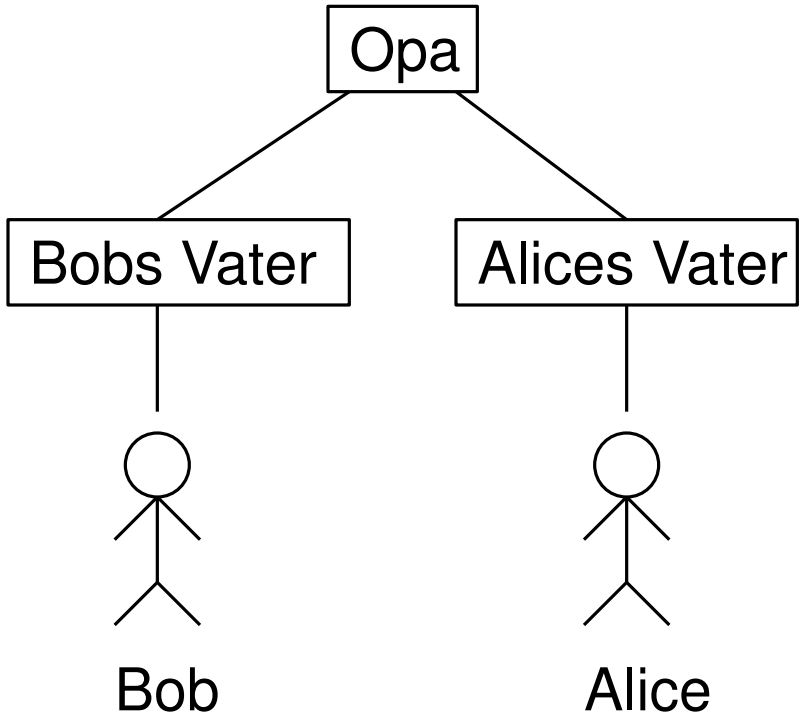
Über wie viele Ecken verwandt?

Bob und Alice wollen wissen, über wie viele Ecken sie verwandt sind

Über wie viele Ecken verwandt?

Bob und Alice wollen wissen, über wie viele Ecken sie verwandt sind

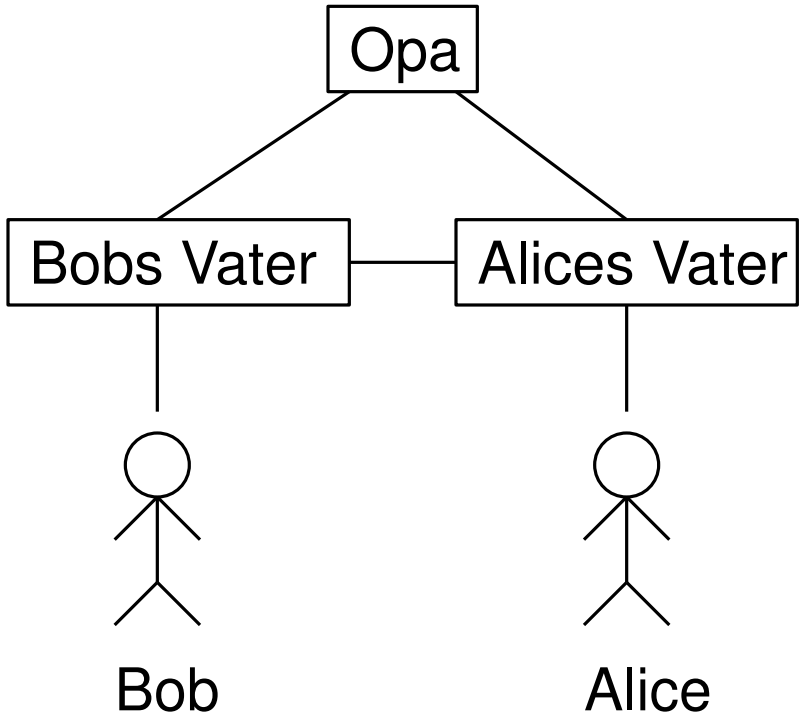
- Bob weiß, dass Alice die Tochter des Sohnes des Vaters seines Vaters ist
  - Also sind sie über 3 Ecken verwandt (?)



Über wie viele Ecken verwandt?

Bob und Alice wollen wissen, über wie viele Ecken sie verwandt sind

- Bob weiß, dass Alice die Tochter des Sohnes des Vaters seines Vaters ist
  - Also sind sie über 3 Ecken verwandt (?)
- Alices Vater ist der Bruder von Bobs Vater
  - Also sind sie über 2 Ecken verwandt!

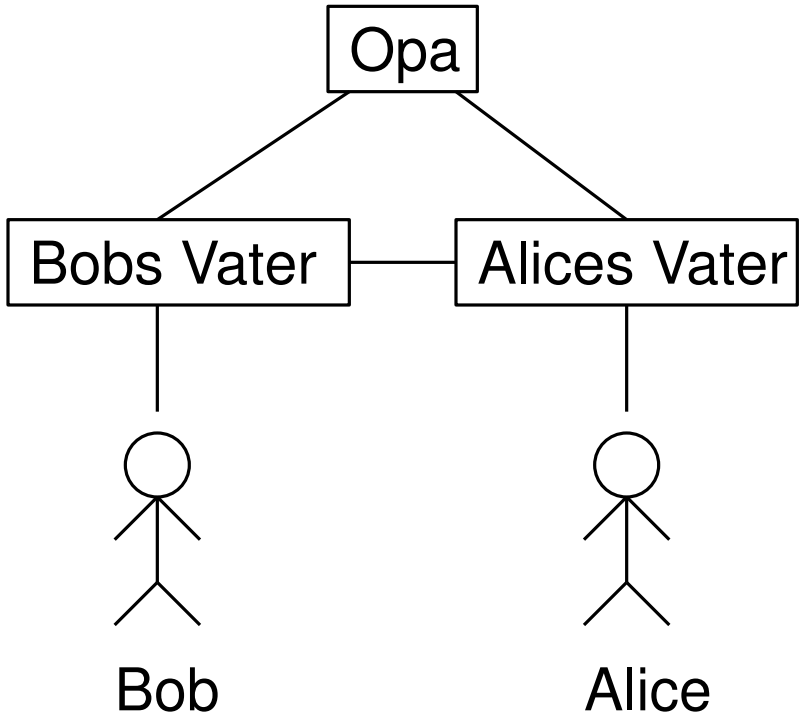


Über wie viele Ecken verwandt?

Bob und Alice wollen wissen, über wie viele Ecken sie verwandt sind

- Bob weiß, dass Alice die Tochter des Sohnes des Vaters seines Vaters ist
  - Also sind sie über 3 Ecken verwandt (?)
- Alices Vater ist der Bruder von Bobs Vater
  - Also sind sie über 2 Ecken verwandt!

Wie können wir für beliebige Personen den Verwandheitsgrad bestimmen?



Verwandheit: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste an Menschen
- Liste an direkten Verwandheiten
  - Eltern, Kinder, Geschwister, ...

Verwandheit: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste an Menschen
- Liste an direkten Verwandheiten
  - Eltern, Kinder, Geschwister, ...

Eingabegraph:

- Ein Knoten pro Mensch
- Kante zwischen zwei Menschen mit direkter Verwandheit

Verwandheit: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste an Menschen
- Liste an direkten Verwandheiten
  - Eltern, Kinder, Geschwister, ...

Problemstellung:

- Finde kürzeste Verwandheitskette zwischen zwei Menschen

Eingabegraph:

- Ein Knoten pro Mensch
- Kante zwischen zwei Menschen mit direkter Verwandheit

Verwandheit: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste an Menschen
- Liste an direkten Verwandheiten
  - Eltern, Kinder, Geschwister, ...

Eingabegraph:

- Ein Knoten pro Mensch
- Kante zwischen zwei Menschen mit direkter Verwandheit

Problemstellung:

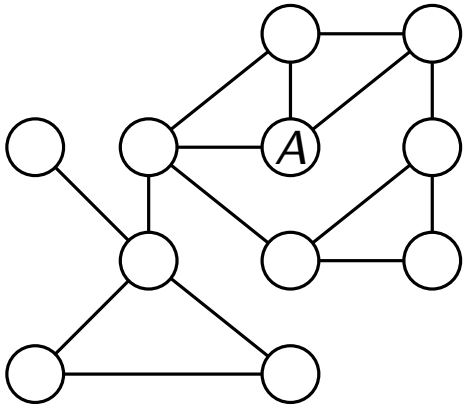
- Finde kürzeste Verwandheitskette zwischen zwei Menschen

Graphproblem:

- Finde kürzesten Weg zwischen zwei Knoten

Kürzeste Wege in ungewichteten Graphen

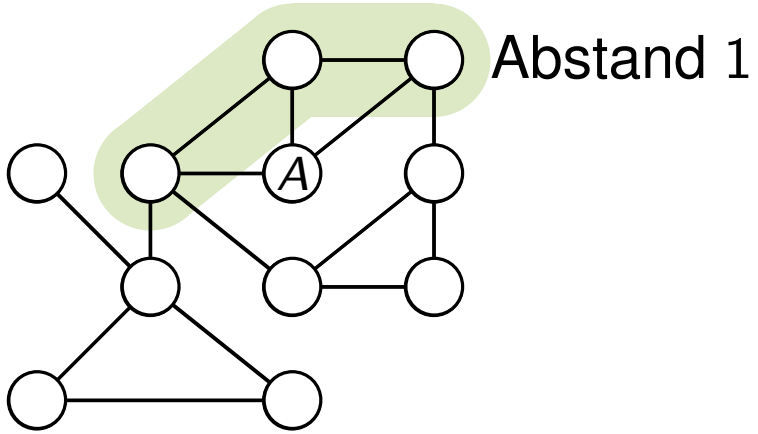
Wir suchen den kürzesten Weg zwischen A und B in einem ungewichteten Graphen



Kürzeste Wege in ungewichteten Graphen

Wir suchen den kürzesten Weg zwischen A und B in einem ungewichteten Graphen

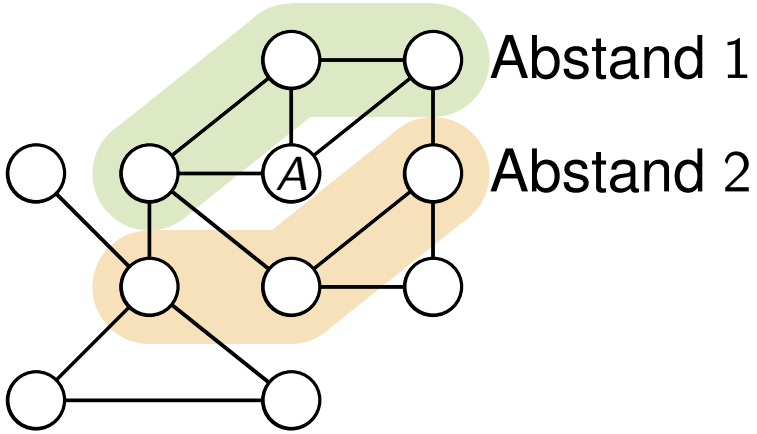
- Beobachtung:
  - *A* und *B* haben Abstand 1 $\iff$ *A* ist Nachbar von *B*



Kürzeste Wege in ungewichteten Graphen

Wir suchen den kürzesten Weg zwischen A und B in einem ungewichteten Graphen

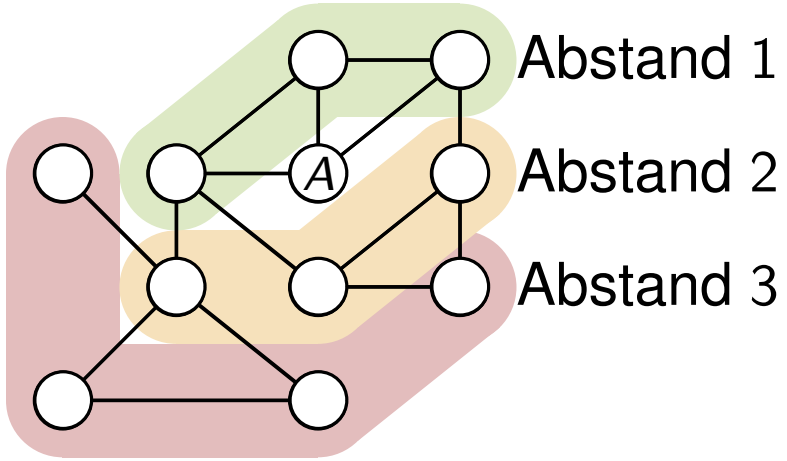
- Beobachtung:
  - *A* und *B* haben Abstand 1 $\iff$ *A* ist Nachbar von *B*
  - *A* und *B* haben Abstand 2 $\iff$ *A* ist Nachbar eines Nachbars von *B*



Kürzeste Wege in ungewichteten Graphen

Wir suchen den kürzesten Weg zwischen A und B in einem ungewichteten Graphen

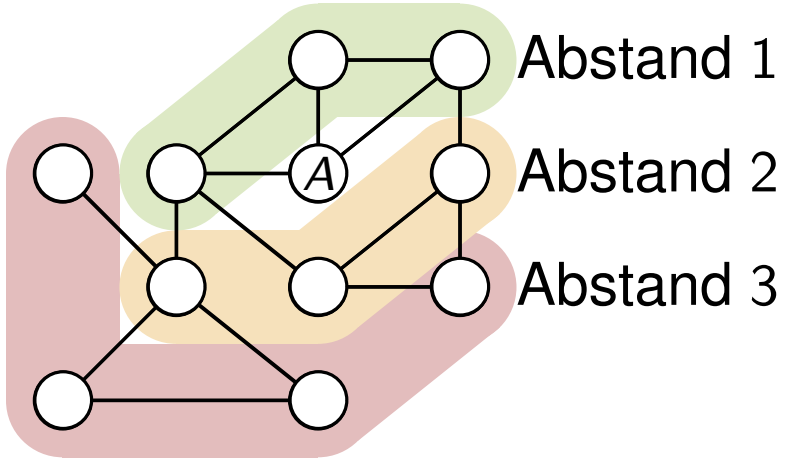
- Beobachtung:
  - *A* und *B* haben Abstand 1 $\iff$ *A* ist Nachbar von *B*
  - *A* und *B* haben Abstand 2 $\iff$ *A* ist Nachbar eines Nachbars von *B*
  - *A* und *B* haben Abstand 3 $\iff$ *A* ist Nachbar eines Nachbars eines Nachbars von *B*
  - $\vdots$



Kürzeste Wege in ungewichteten Graphen

Wir suchen den kürzesten Weg zwischen A und B in einem ungewichteten Graphen

- Beobachtung:
  - A und B haben Abstand 1 $\iff$ A ist Nachbar von B
  - A und B haben Abstand 2 $\iff$ A ist Nachbar eines Nachbars von B
  - A und B haben Abstand 3 $\iff$ A ist Nachbar eines Nachbars eines Nachbars von B
  - $\vdots$



- Idee:
  - Teile Graph in Schichten von Nachbarschaften auf
  - Knoten A hat Abstand 0 zu sich selber
  - unentdeckter Nachbar von Knoten mit Abstand n hat Abstand $n + 1$

Breitensuche: Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

Breitensuche: Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

- Knoten werden nach der Reihenfolge in der Queue abgearbeitet
- Früher in der Queue $\iff$ weniger Abstand zu s

Breitensuche: Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

- Startknoten ist erster Knoten und hat Abstand 0 zu sich selbst.

Breitensuche: Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return  z.layer
```

- Gucke immer nächst-näheren Knoten an

Breitensuche: Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

- Für jeden neuen Nachbarn, aktualisiere Distanz
- Falls der Zielknoten gefunden ist, haben wir gewonnen

Breitensuche: Laufzeit

Formulierung in Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

Welche Laufzeit hat dieser Algorithmus?

Breitensuche: Laufzeit

Formulierung in Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten landet maximal 1 mal in der Queue

Breitensuche: Laufzeit

Formulierung in Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten landet maximal 1 mal in der Queue
- Für jeden Knoten wird jede ausgehende Kante 1 mal angeguckt

Breitensuche: Laufzeit

Formulierung in Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten landet maximal 1 mal in der Queue
- Für jeden Knoten wird jede ausgehende Kante 1 mal angeguckt

$\Longrightarrow$ O(Für jeden Knoten: #ausgehender Kanten)

Breitensuche: Laufzeit

Formulierung in Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten landet maximal 1 mal in der Queue
- Für jeden Knoten wird jede ausgehende Kante 1 mal angeguckt

$\Longrightarrow$ O(Für jeden Knoten: #ausgehender Kanten)

$= O(\sum_{v \in V} deg(v))$

Breitensuche: Laufzeit

Formulierung in Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten landet maximal 1 mal in der Queue
- Für jeden Knoten wird jede ausgehende Kante 1 mal angeguckt

$\Longrightarrow$ O(Für jeden Knoten: #ausgehender Kanten)

$$= O(\sum_{v \in V} deg(v)) \quad = O(2|E|)$$

heißt Handshake-Lemma

Breitensuche: Laufzeit

Formulierung in Pseudocode

```
BFS(Graph G, Start s, Goal z)
    Queue Q := empty queue
    Q.push(s)
    s.layer = 0
    while Q ≠ ∅ do
        u := Q.pop()
        for Node v in N(u) do
            if v.layer = −∞ then
                Q.push(v)
                v.layer = u.layer + 1
            if v = z then
                return z.layer
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten landet maximal 1 mal in der Queue
- Für jeden Knoten wird jede ausgehende Kante 1 mal angeguckt

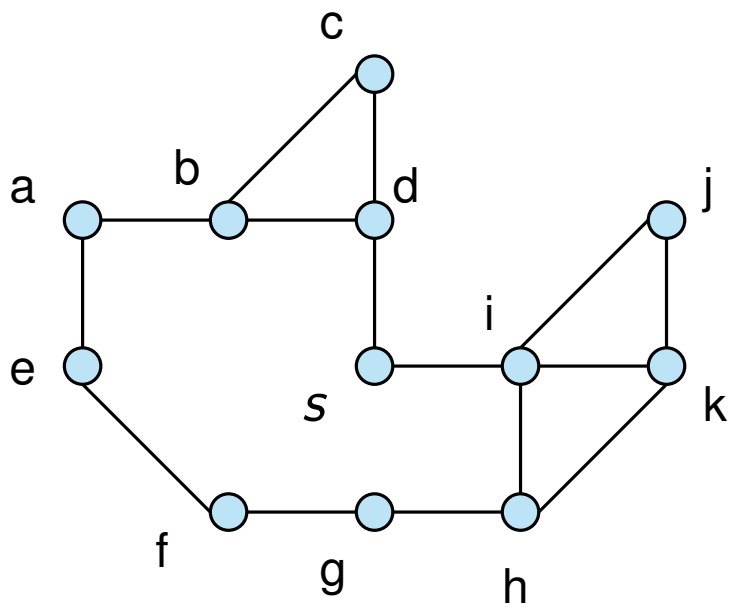
$\Longrightarrow$ O(Für jeden Knoten: #ausgehender Kanten)

$$= O(\sum_{v \in V} deg(v)) \quad = O(2|E|)$$

heißt Handshake-Lemma

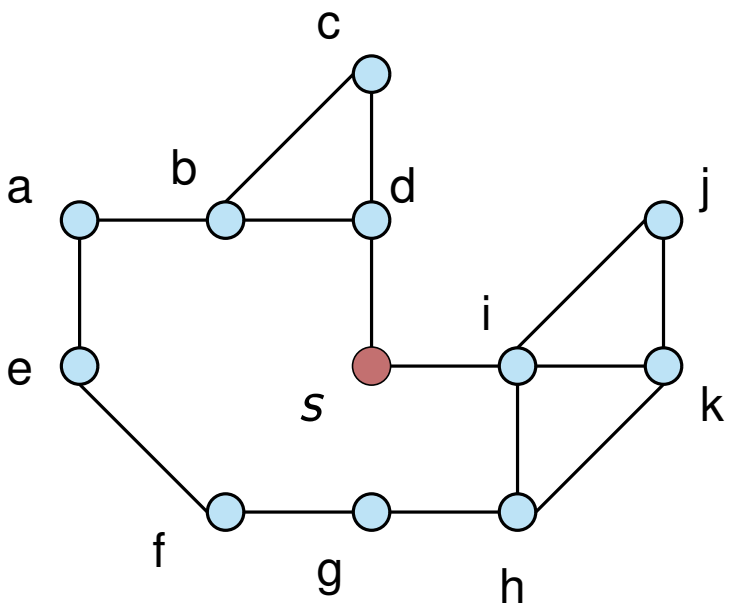
BFS hat eine Laufzeit von $O(|E|)$

BFS: Beispiel

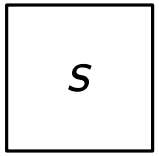


Queue Q

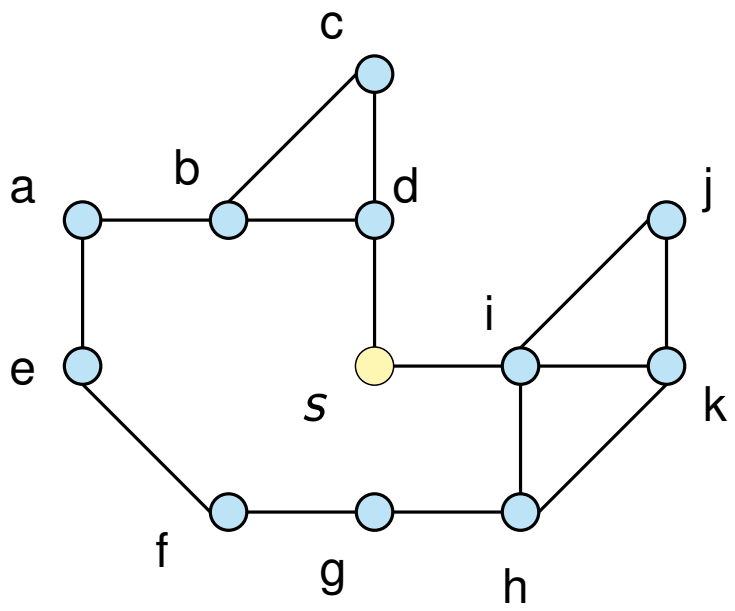
BFS: Beispiel



Queue Q

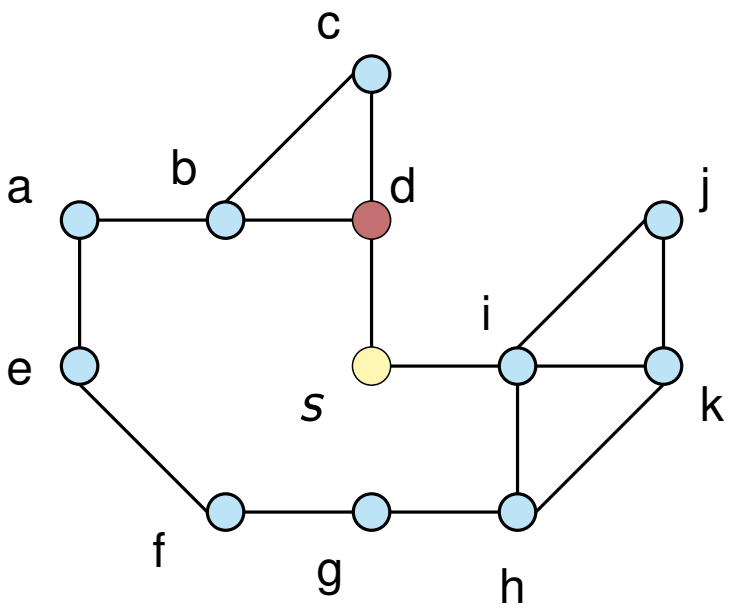


BFS: Beispiel

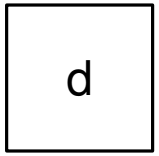


Queue Q

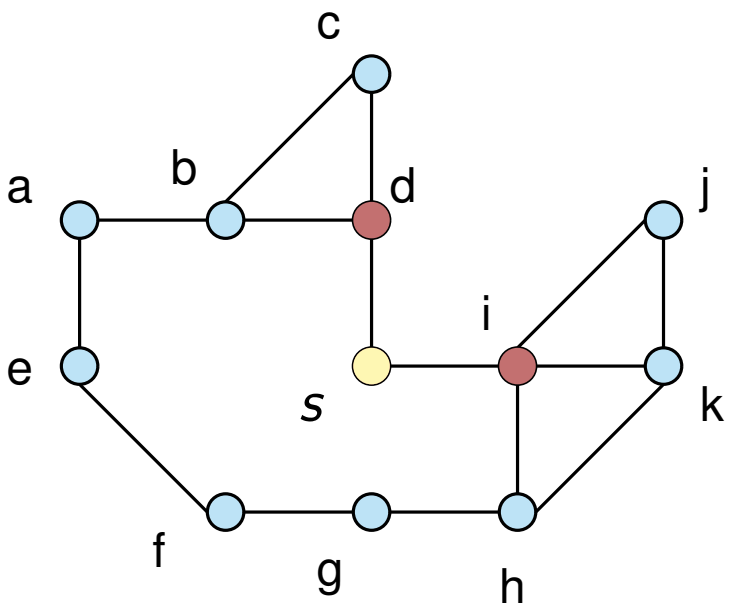
BFS: Beispiel



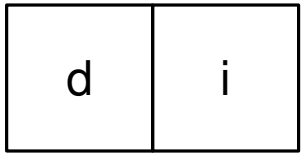
Queue Q



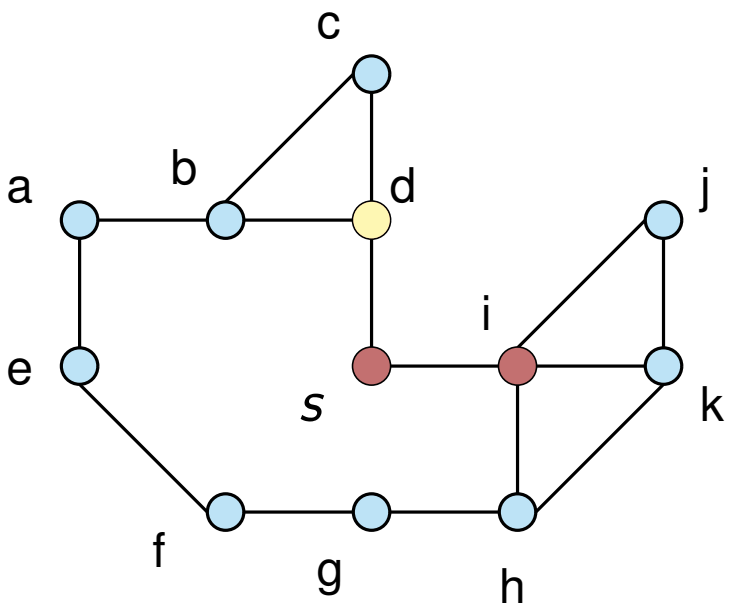
BFS: Beispiel



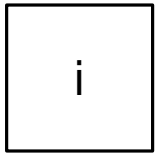
Queue Q



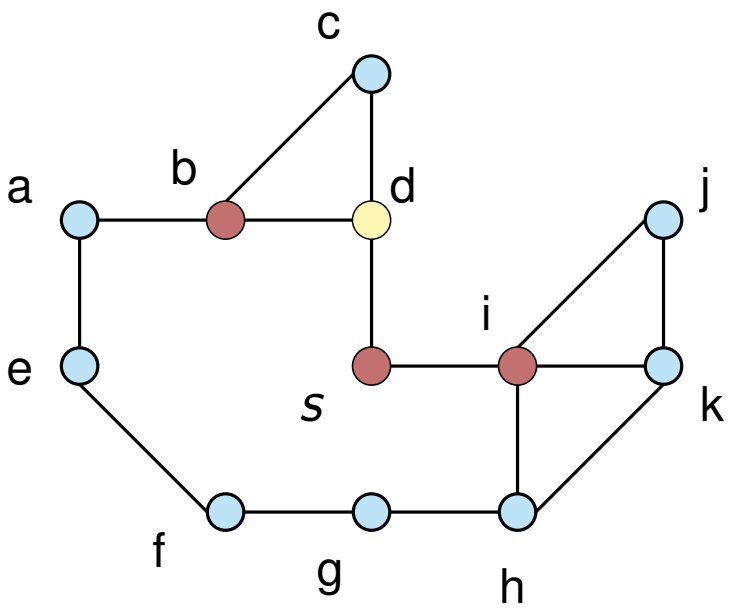
BFS: Beispiel



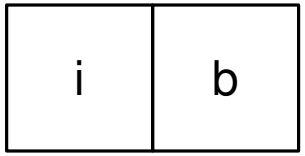
Queue Q



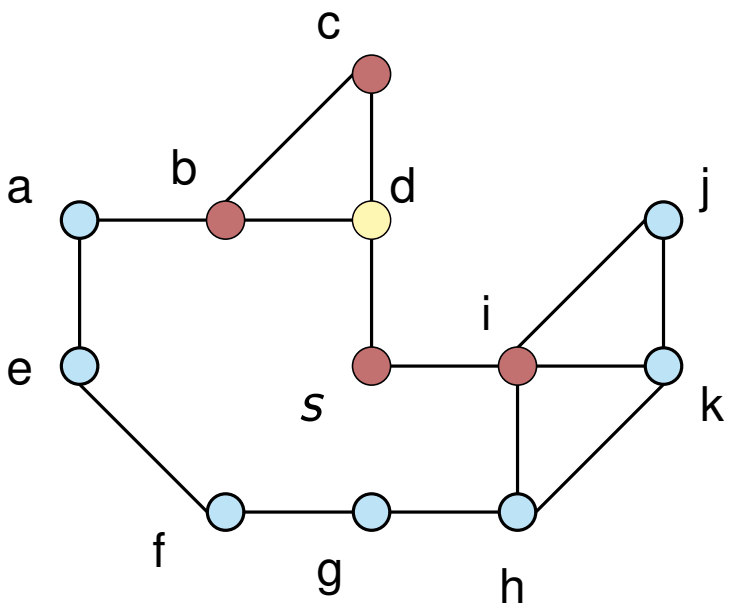
BFS: Beispiel



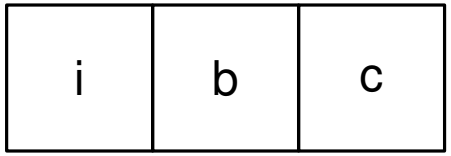
Queue Q



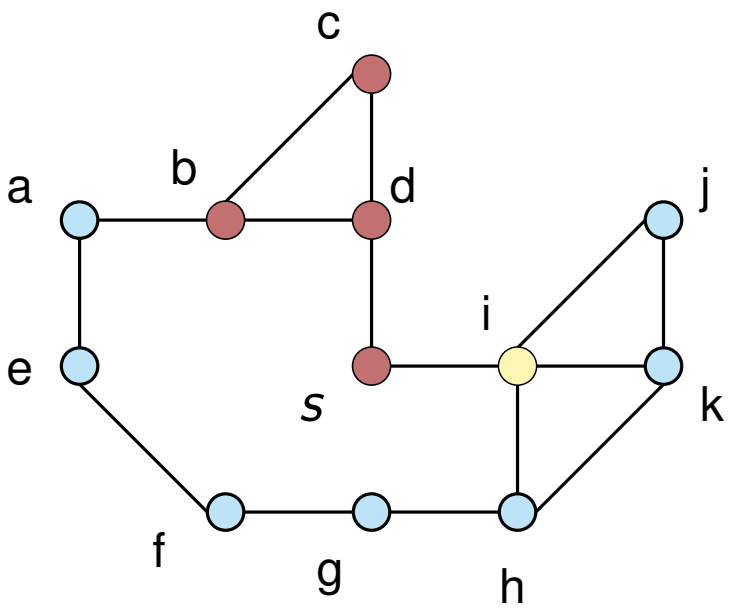
BFS: Beispiel



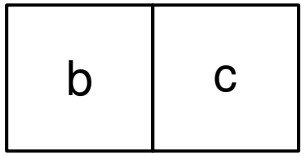
Queue Q



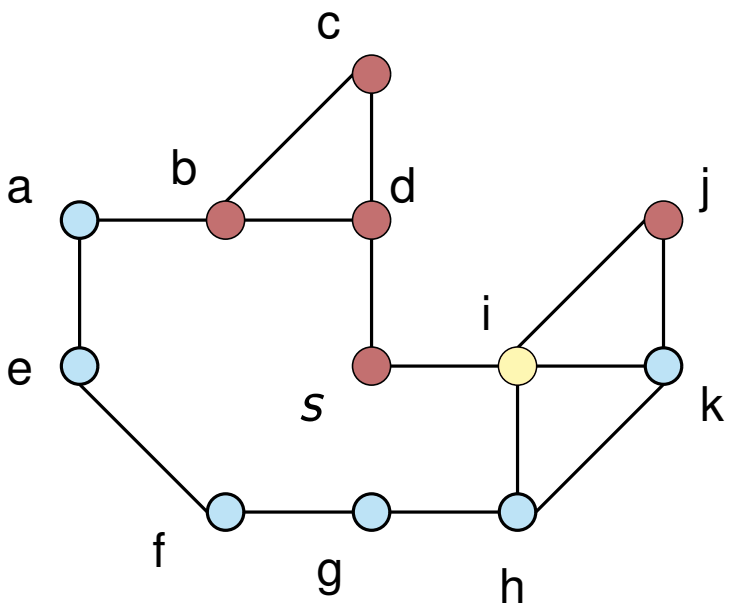
BFS: Beispiel



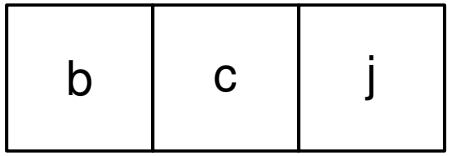
Queue Q



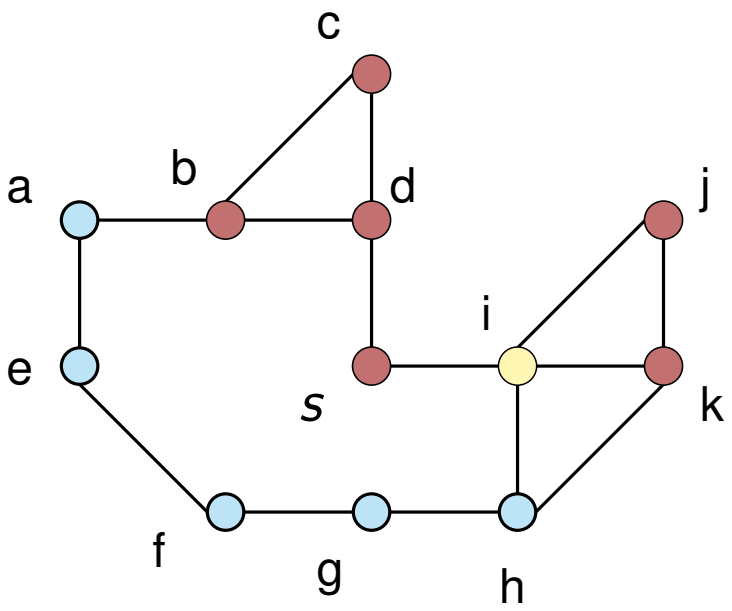
BFS: Beispiel



Queue Q



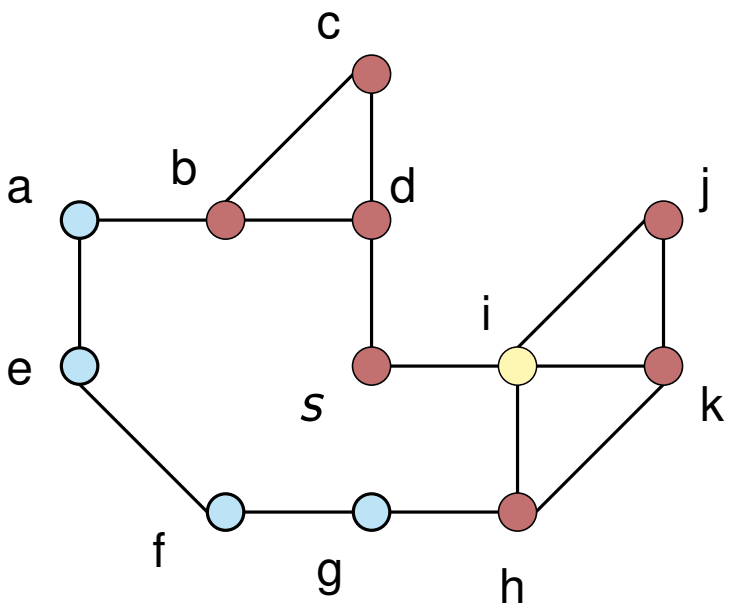
BFS: Beispiel



Queue Q

b	c	j	k

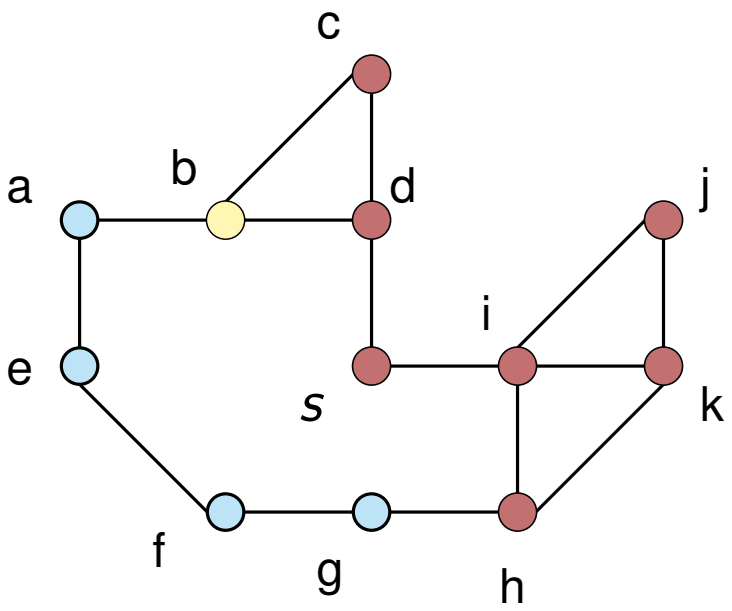
BFS: Beispiel



Queue Q

b	c	j	k	h

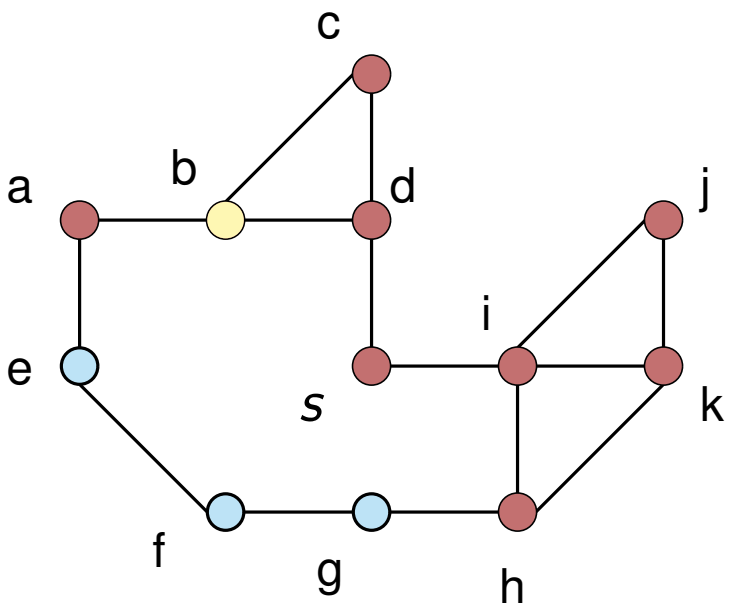
BFS: Beispiel



Queue Q

c	j	k	h

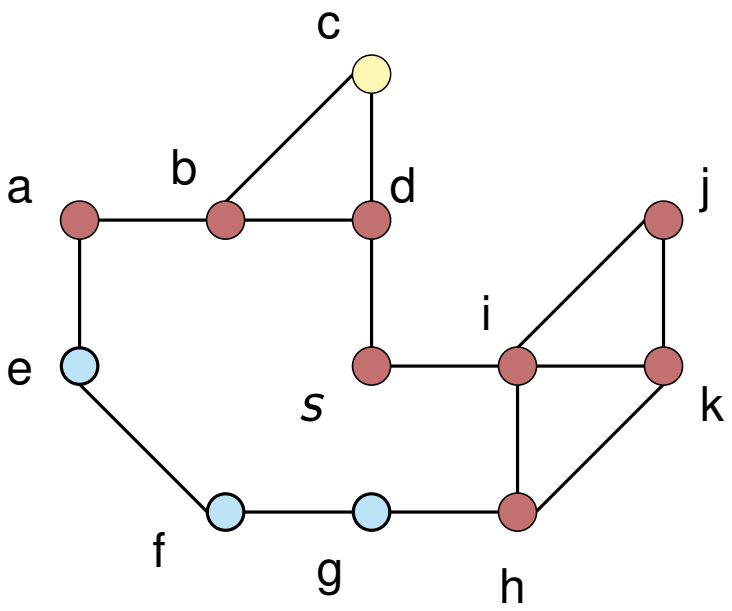
BFS: Beispiel



Queue Q

c	j	k	h	a

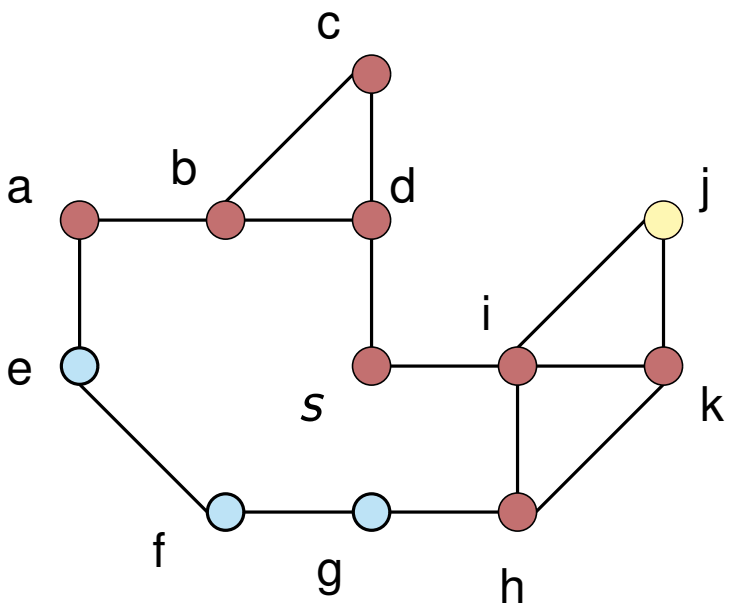
BFS: Beispiel



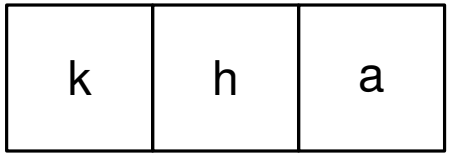
Queue Q

j	k	h	a

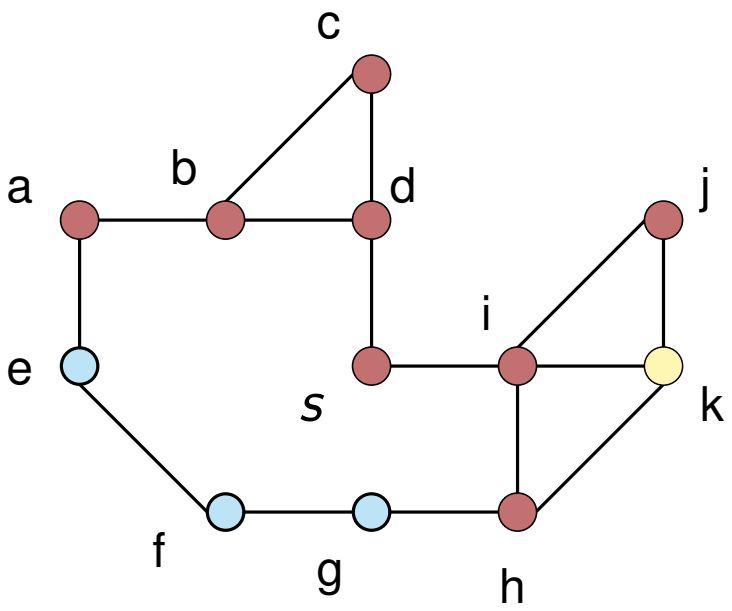
BFS: Beispiel



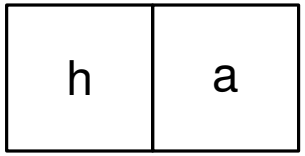
Queue Q



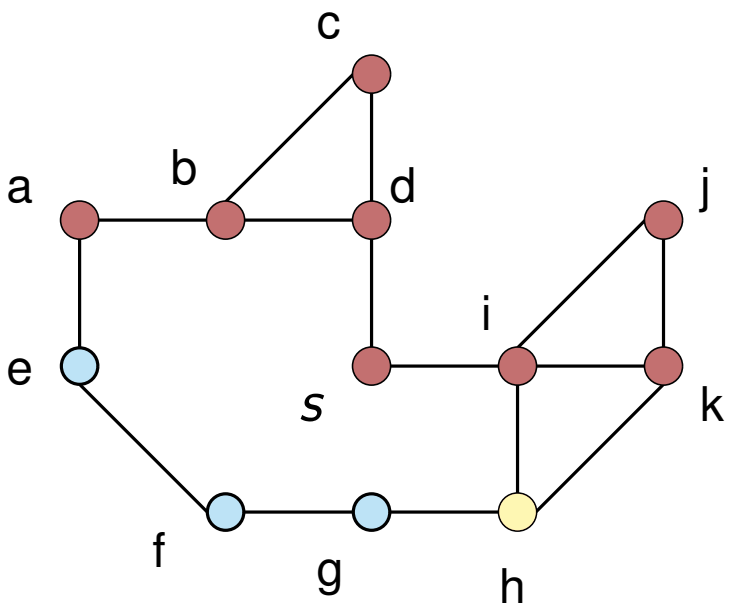
BFS: Beispiel



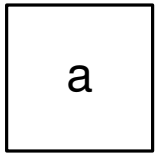
Queue Q



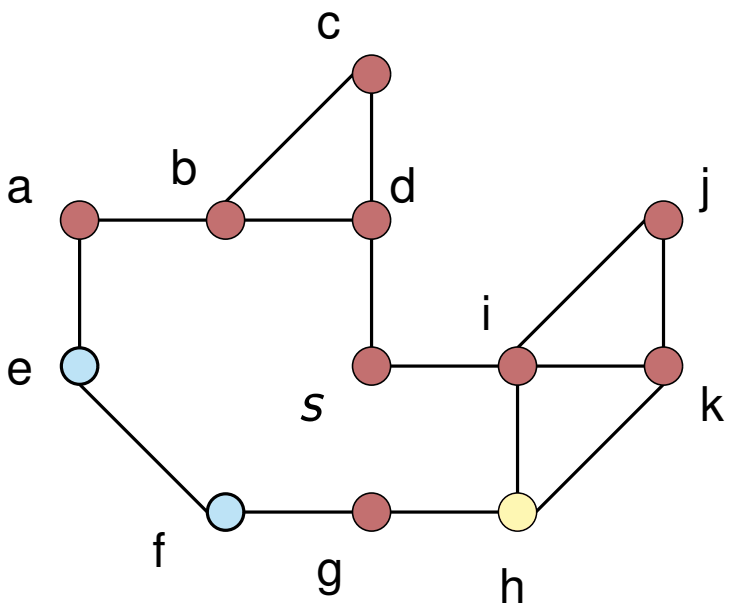
BFS: Beispiel



Queue Q



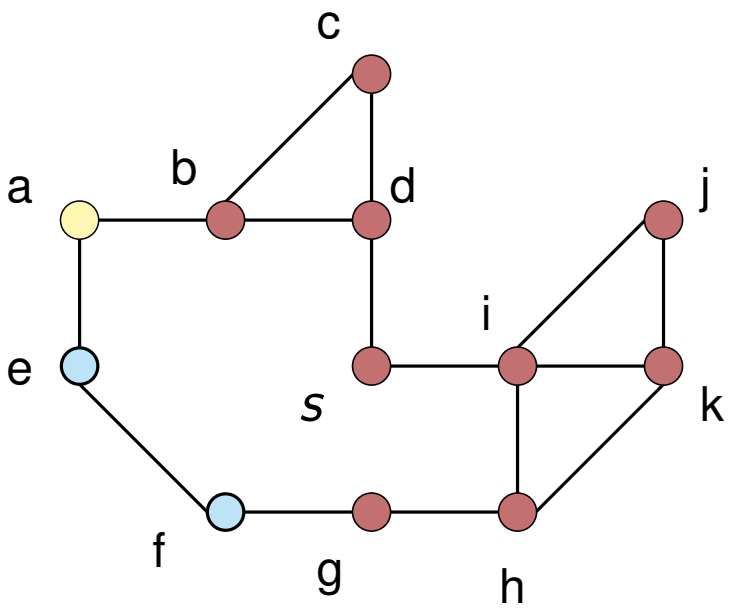
BFS: Beispiel



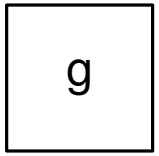
Queue Q

a	g

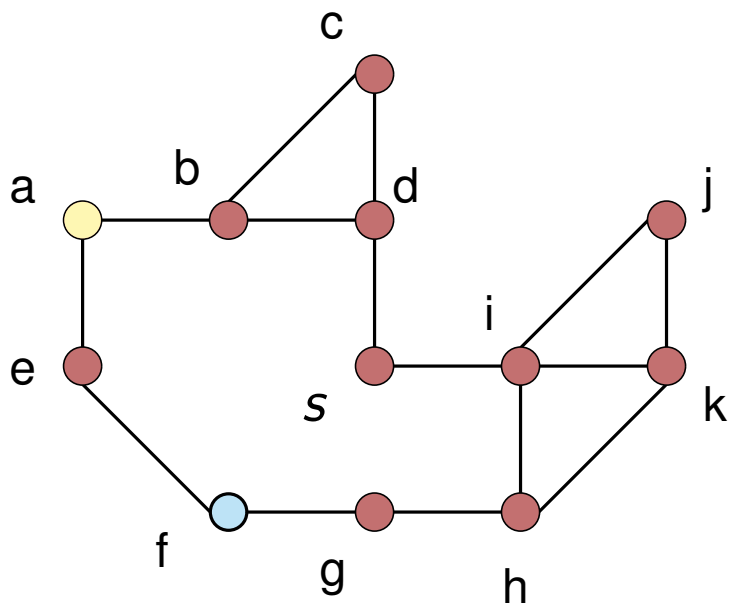
BFS: Beispiel



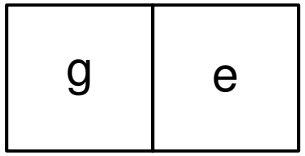
Queue Q



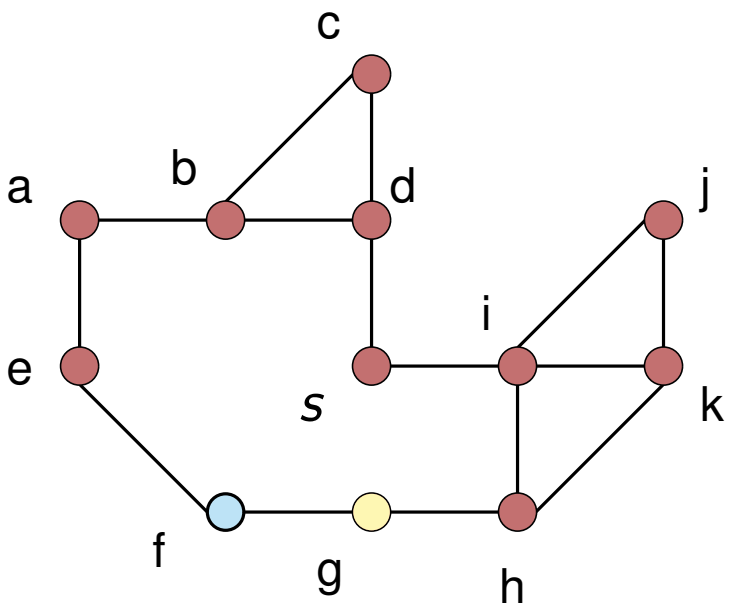
BFS: Beispiel



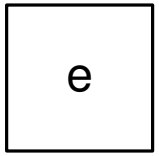
Queue Q



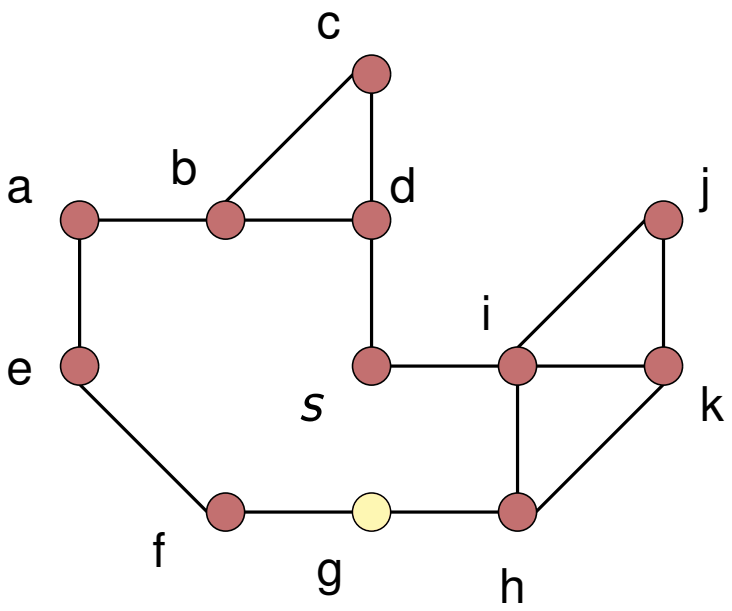
BFS: Beispiel



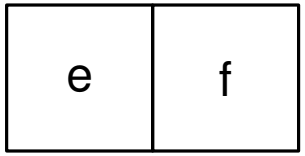
Queue Q



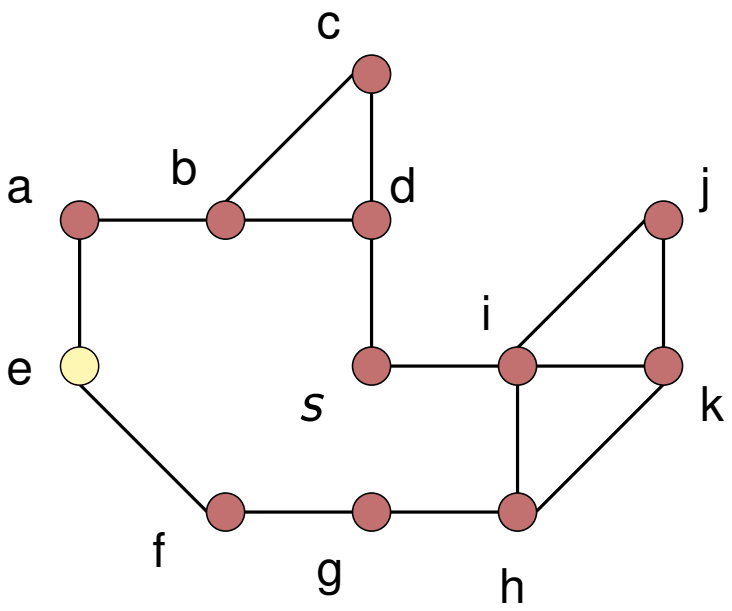
BFS: Beispiel



Queue Q

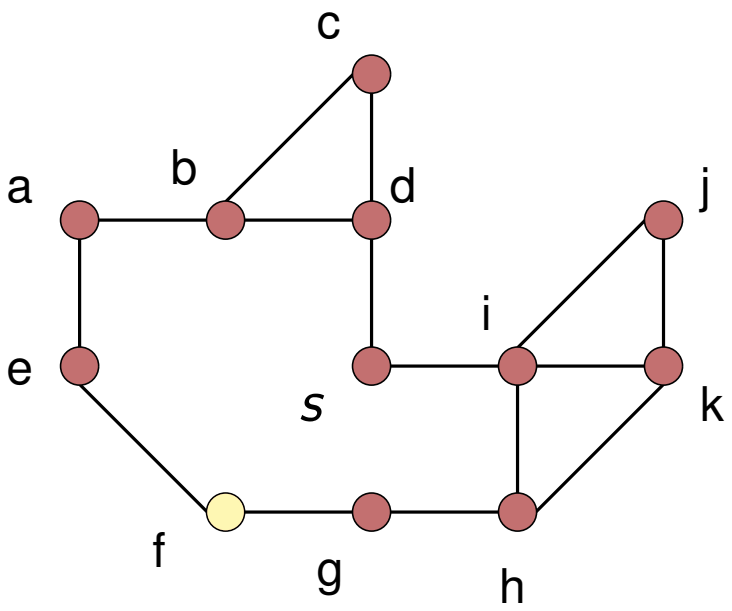


BFS: Beispiel



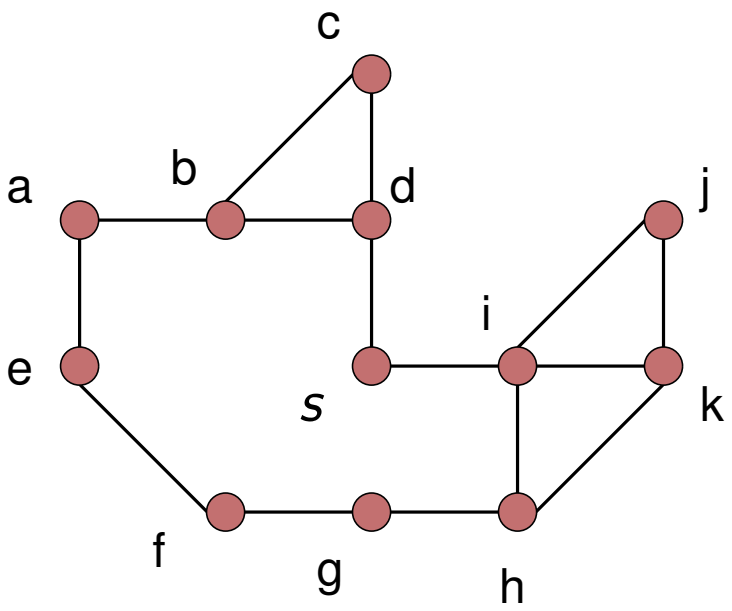
Queue Q

BFS: Beispiel



Queue Q

BFS: Beispiel

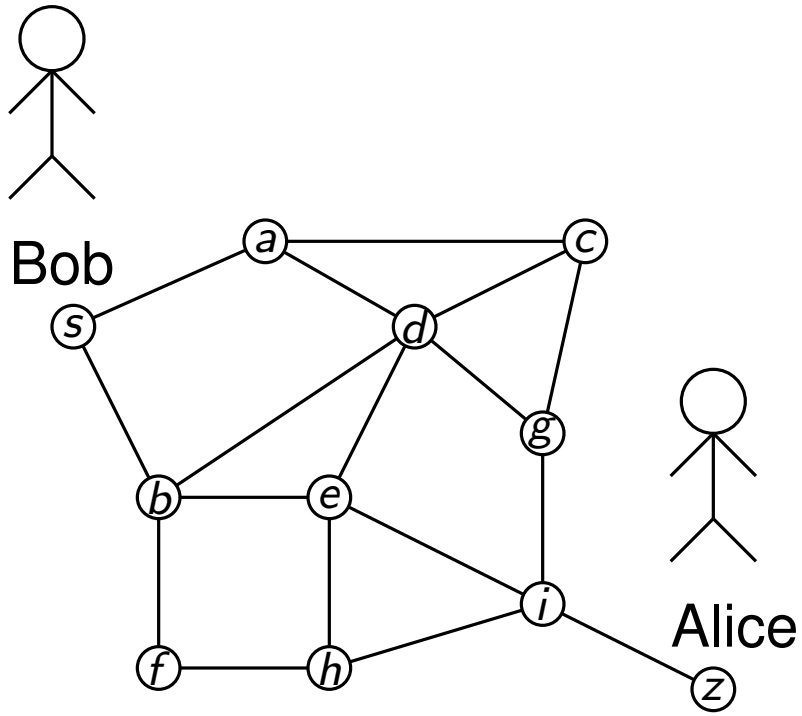


Queue Q

Navi spielen

Bob würde gerne Alice besuchen

- Bob wohnt in Startstadt, Alice in Zielcity
- Bob will keine Zeit verschwenden und möglichst schnell bei Alice ankommen



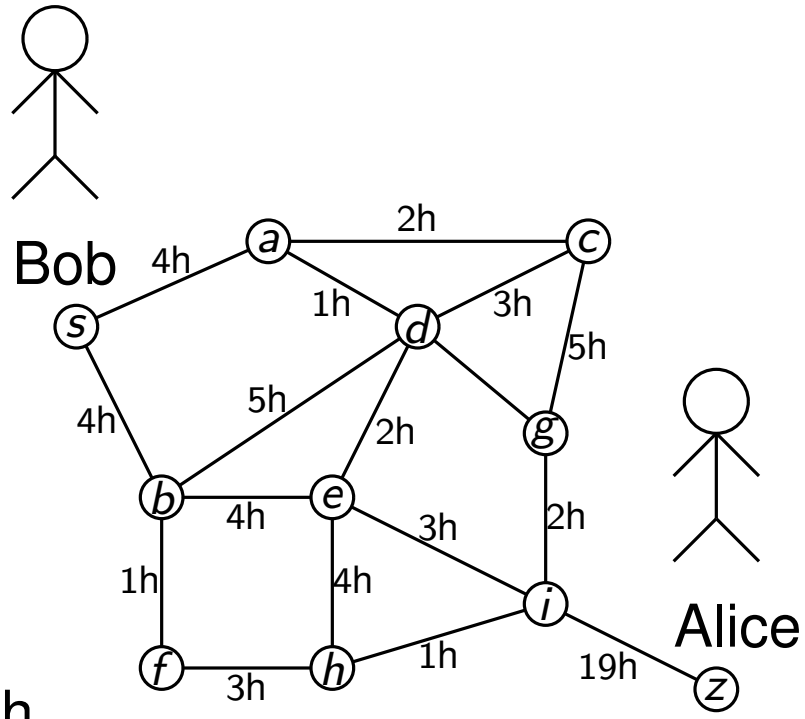
Navi spielen

Bob würde gerne Alice besuchen

- Bob wohnt in Startstadt, Alice in Zielcity
- Bob will keine Zeit verschwenden und möglichst schnell bei Alice ankommen

Bedingungen und Problemformulierung

- Jede Strecke zwischen zwei Städten hat Dauer der Fahrt
- Gesucht ist eine Folge von Strecken von Startstadt nach Zielcity, sodass die Fahrt möglichst schnell ist



Navi spielen: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste Städten oder Kreuzungen
- Liste an Strecken zwischen den Städten oder Kreuzungen
- Für jede Strecke eine Durchfahrdauer

Navi spielen: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste Städten oder Kreuzungen
- Liste an Strecken zwischen den Städten oder Kreuzungen
- Für jede Strecke eine Durchfahrdauer

Eingabegraph:

- Ein Knoten pro Stadt / Kreuzungen
- Kante zwischen Städten / Krezungen mit Strecke
- Jede Kante hat ein positives Gewicht, für die zugewiesene Streckendauer

Navi spielen: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste Städten oder Kreuzungen
- Liste an Strecken zwischen den Städten oder Kreuzungen
- Für jede Strecke eine Durchfahrdauer

Eingabegraph:

- Ein Knoten pro Stadt / Kreuzungen
- Kante zwischen Städten / Krezungen mit Strecke
- Jede Kante hat ein positives Gewicht, für die zugewiesene Streckendauer

Problemstellung:

- Finde kürzeste Folge an Strecken zwischen zwei gegebenen Städten

Navi spielen: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste Städten oder Kreuzungen
- Liste an Strecken zwischen den Städten oder Kreuzungen
- Für jede Strecke eine Durchfahrdauer

Problemstellung:

- Finde kürzeste Folge an Strecken zwischen zwei gegebenen Städten

Eingabegraph:

- Ein Knoten pro Stadt / Kreuzungen
- Kante zwischen Städten / Krezungen mit Strecke
- Jede Kante hat ein positives Gewicht, für die zugewiesene Streckendauer

Graphproblem:

- Finde kürzesten Weg zwischen zwei Knoten
- minimale Summe der Kantengewichte

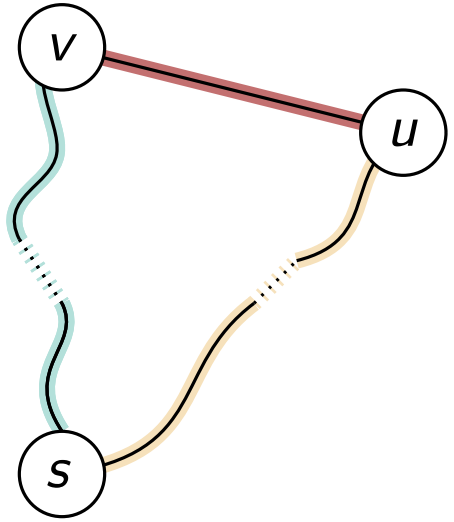
Dijkstras Algorithmus: Strategie

Arbeite immer unexplorierten Knoten mit aktuell kürzester Distanz ab

Dijkstras Algorithmus: Strategie

Arbeite immer unexplorierten Knoten mit aktuell kürzester Distanz ab

- Aktualisiere Distanzen der Nachbarn, wenn ein neuer Knoten exploriert wird

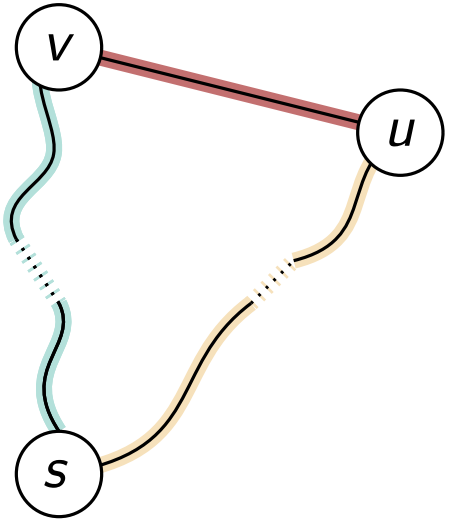


vorläufige $s-v$ und $s-u$ Pfade gegeben

Dijkstras Algorithmus: Strategie

Arbeite immer unexplorierten Knoten mit aktuell kürzester Distanz ab

- Aktualisiere Distanzen der Nachbarn, wenn ein neuer Knoten exploriert wird



vorläufige $s-v$ und $s-u$ Pfade gegeben

- v wird exploriert

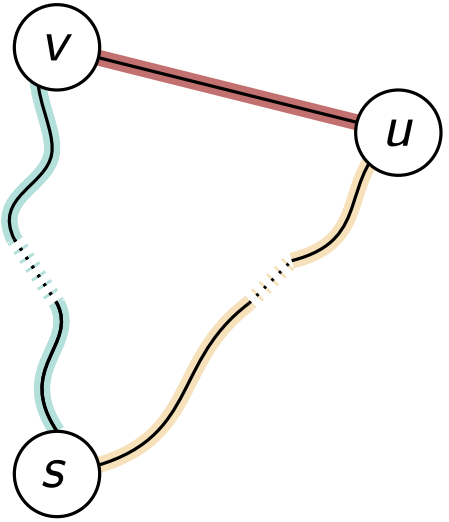
$$\Longrightarrow \quad d(s,u) = \min(d(s,u), d(s,v) + \text{len}(v,u))$$

  - " Ist es billiger über v zu u zu gehen? "

Dijkstras Algorithmus: Strategie

Arbeite immer unexplorierten Knoten mit aktuell kürzester Distanz ab

- Aktualisiere Distanzen der Nachbarn, wenn ein neuer Knoten exploriert wird
- u ist unexplorierter Knoten mit aktuell kürzester Distanz
  - Alle Knoten mit kürzerer Distanz sind exploriert
  - Dann ist $d(s, u)$ bereits optimal
    - Wenn Kantengewichte positiv



vorläufige $s-v$ und $s-u$ Pfade gegeben

- v wird exploriert

$$\Longrightarrow \quad d(s,u) = \min(d(s,u), d(s,v) + \text{len}(v,u))$$

  - " Ist es billiger über v zu u zu gehen? "

Dijkstras Algorithmus: Pseudocode

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

Dijkstras Algorithmus: Pseudocode

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

- In $d[v]$ speichern wir uns die aktuelle beste Distanz von s zu v
- Nach dem Algorithmus wird d ausgegeben

Dijkstras Algorithmus: Pseudocode

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

- In Q speichern wir die unexplorierten Knoten
- Sortiert nach der aktuellen besten Distanz zu s

Dijkstras Algorithmus: Pseudocode

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

- Wir explorieren den Knoten mit aktuell kürzester Distanz zu s

Dijkstras Algorithmus: Pseudocode

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

- Aktualisiere Distanz zu Nachbarn, falls besserer Weg gefunden wurde

Dijkstras Algorithmus: Laufzeit

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

Welche Laufzeit hat dieser Algorithmus?

Dijkstras Algorithmus: Laufzeit

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten wird genau 1 mal gepusht

Dijkstras Algorithmus: Laufzeit

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten wird genau 1 mal gepusht
- Jeder Knoten wird genau einmal entfernt

Dijkstras Algorithmus: Laufzeit

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten wird genau 1 mal gepusht
- Jeder Knoten wird genau einmal entfernt
- Für jeden Knoten wird jede ausgehende Kante betrachtet

Dijkstras Algorithmus: Laufzeit

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten wird genau 1 mal gepusht
- Jeder Knoten wird genau einmal entfernt
- Für jeden Knoten wird jede ausgehende Kante betrachtet
  - Für jede Kante könnte **decPrio** aufgerufen werden

Dijkstras Algorithmus: Laufzeit

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten wird genau 1 mal gepusht
- Jeder Knoten wird genau einmal entfernt
- Für jeden Knoten wird jede ausgehende Kante betrachtet
- Für jede Kante könnte **decPrio** aufgerufen werden

$$\overset{\text{buildHeap}}{\Longrightarrow} O(n + n\log(n) + m\log(n))$$

Dijkstras Algorithmus: Laufzeit

```
Dijkstra(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    PriorityQueue Q := empty priority queue
    for Node v in V do
        Q.push(v, d[v])
    while Q ≠ ∅ do
        u := Q.popMin()
        for Node v in N(u) do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
                Q.decPrio(v, d[v])
```

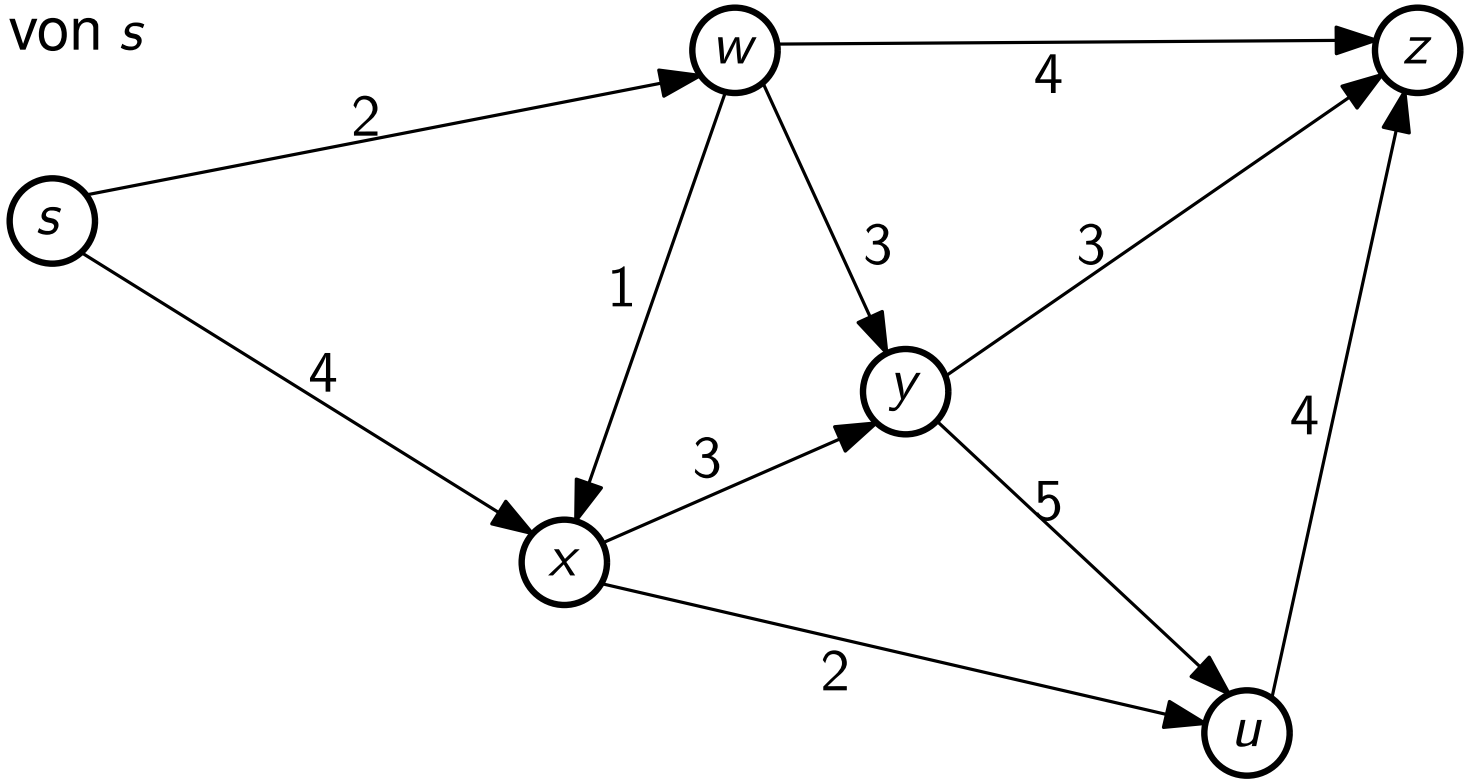
Welche Laufzeit hat dieser Algorithmus?

- Jeder Knoten wird genau 1 mal gepusht
- Jeder Knoten wird genau einmal entfernt
- Für jeden Knoten wird jede ausgehende Kante betrachtet
  - Für jede Kante könnte **decPrio** aufgerufen werden

$$\implies O(n + n\log(n) + m\log(n))$$
$$= O((n+m)\log(n))$$

Dijkstra Algorithmus: Beispiel

Dijkstra von *s*



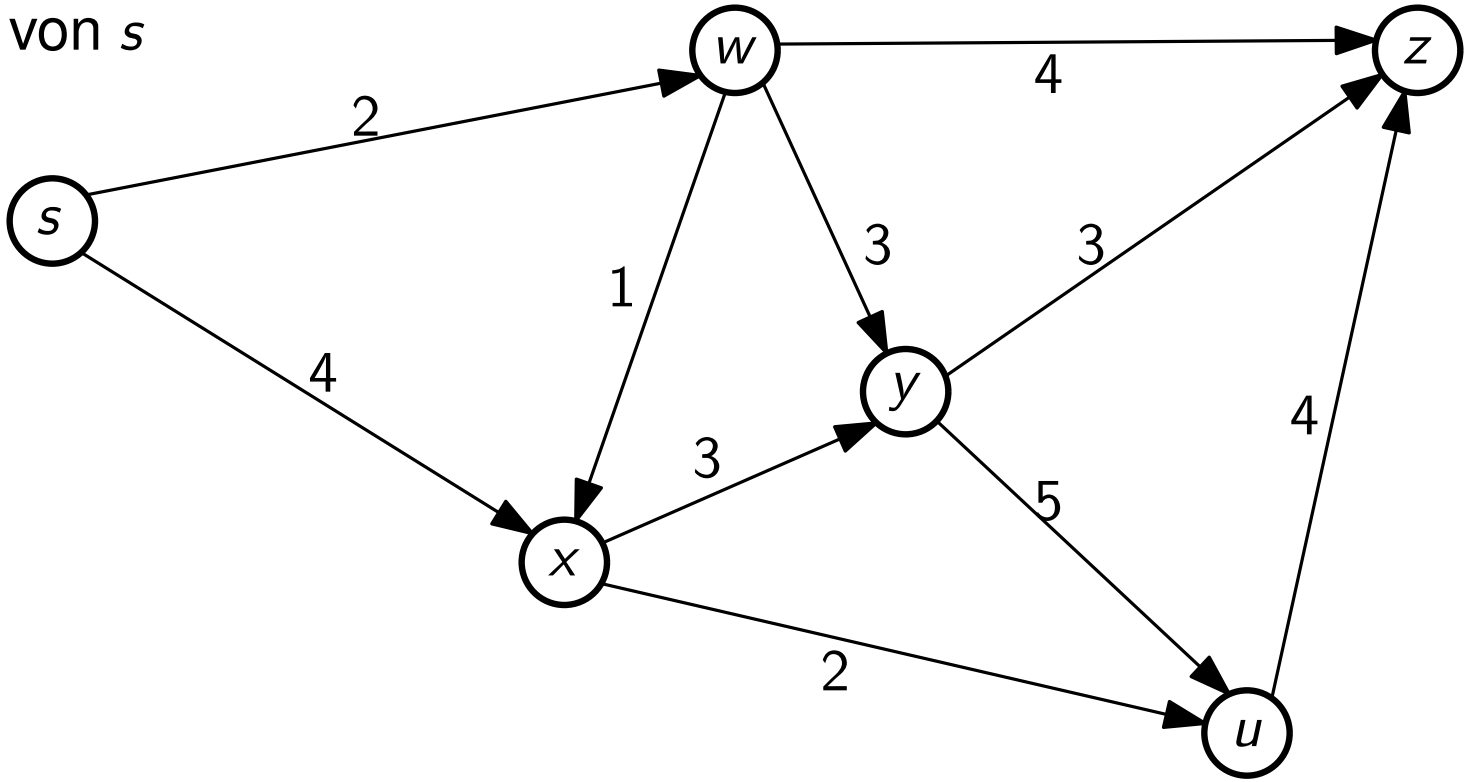
Distanzen

u	*s*	*w*	*x*	*y*	*z*

Priority Queue

Dijkstra Algorithmus: Beispiel

Dijkstra von *s*



Distanzen

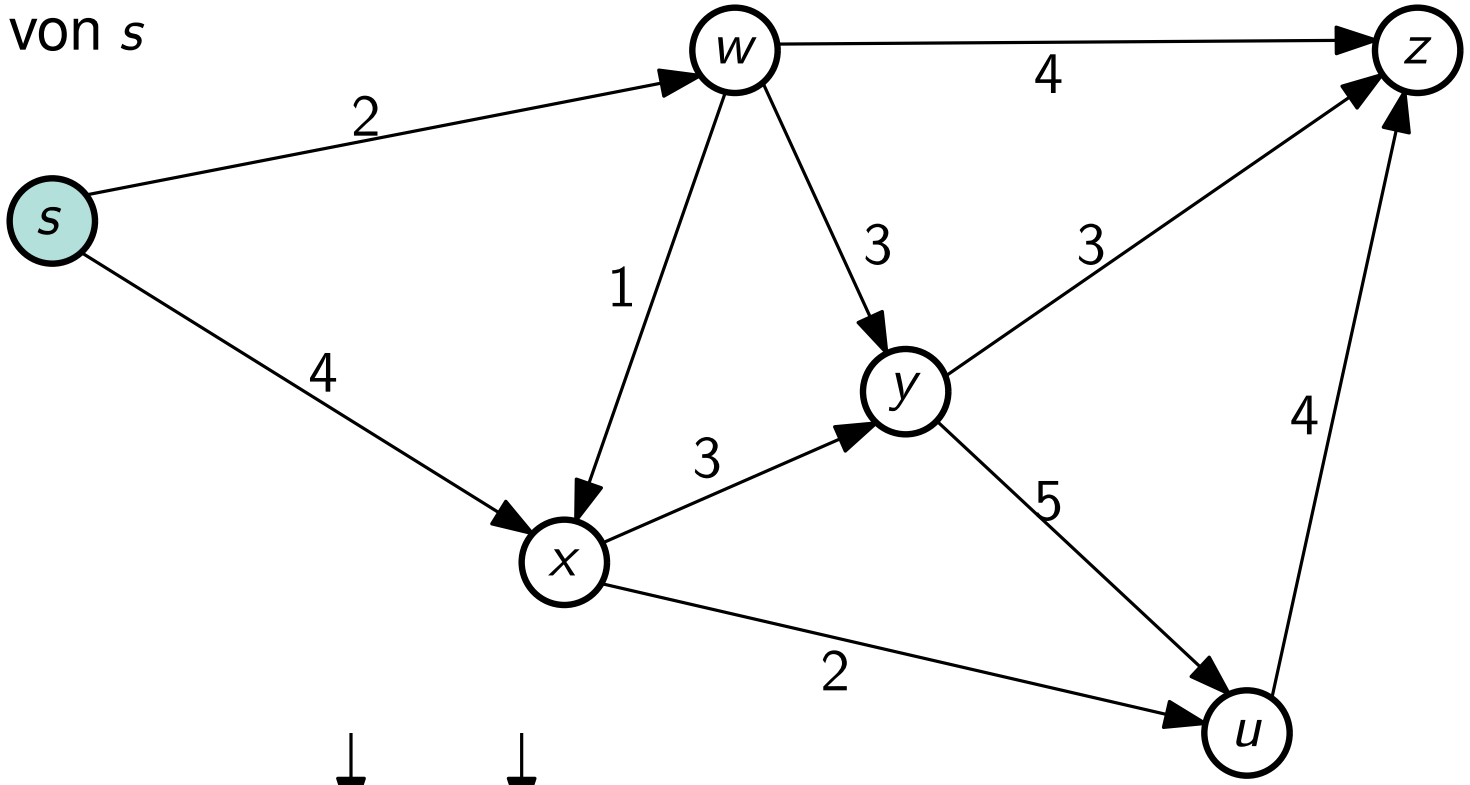
u	*s*	*w*	*x*	*y*	*z*
∞	0	∞	∞	∞	∞

Priority Queue

s	*u*	*w*	*x*	*y*	*z*
0	∞	∞	∞	∞	∞

Dijkstra Algorithmus: Beispiel

Dijkstra von s



Distanzen

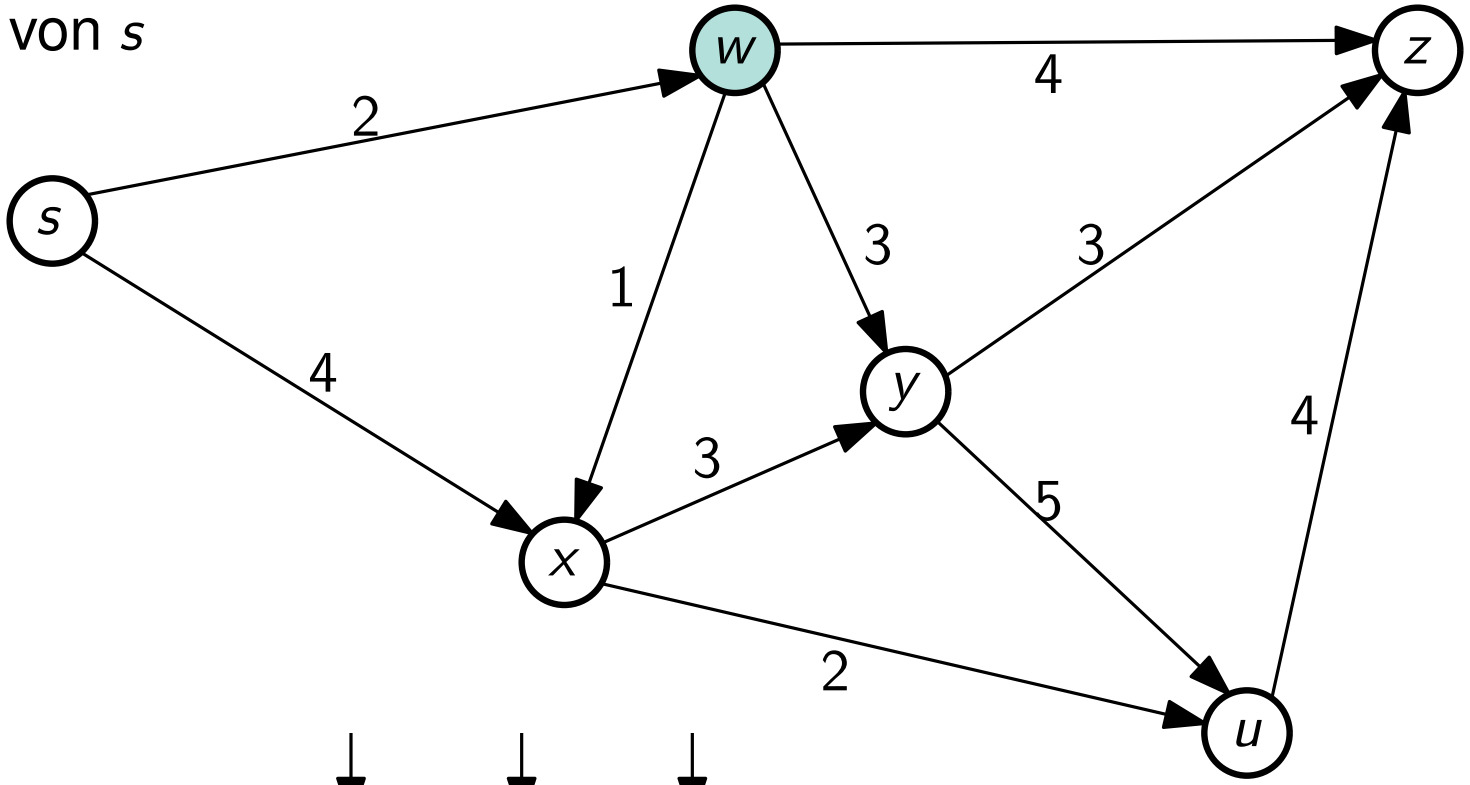
u	s	w	x	y	z
∞	0	2	4	∞	∞

Priority Queue

w	x	u	y	z
2	4	∞	∞	∞

Dijkstra Algorithmus: Beispiel

Dijkstra von s



Distanzen

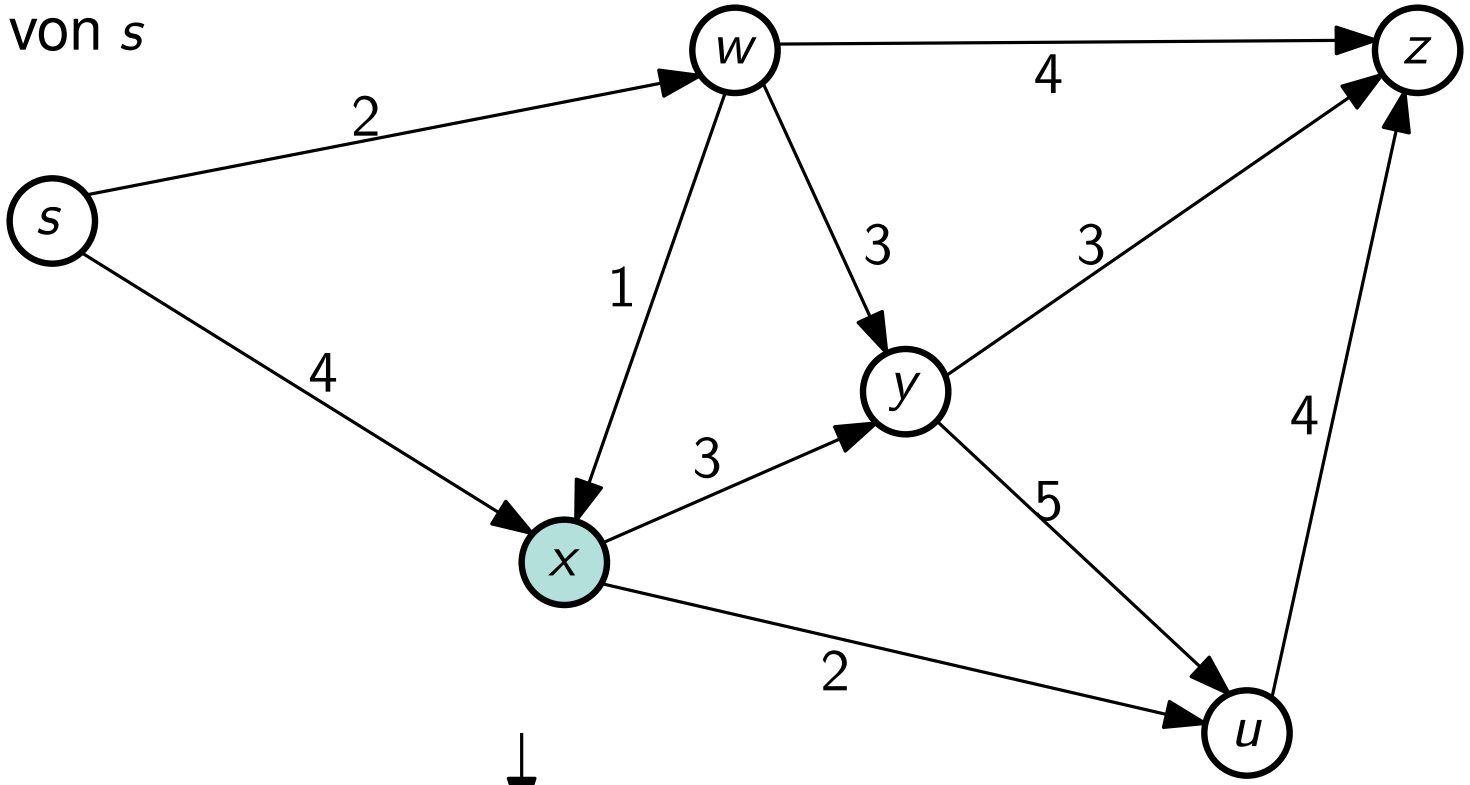
u	s	w	x	y	z
∞	0	2	3	5	6

Priority Queue

x	y	z	u
3	5	6	∞

Dijkstra Algorithmus: Beispiel

Dijkstra von *s*



Distanzen

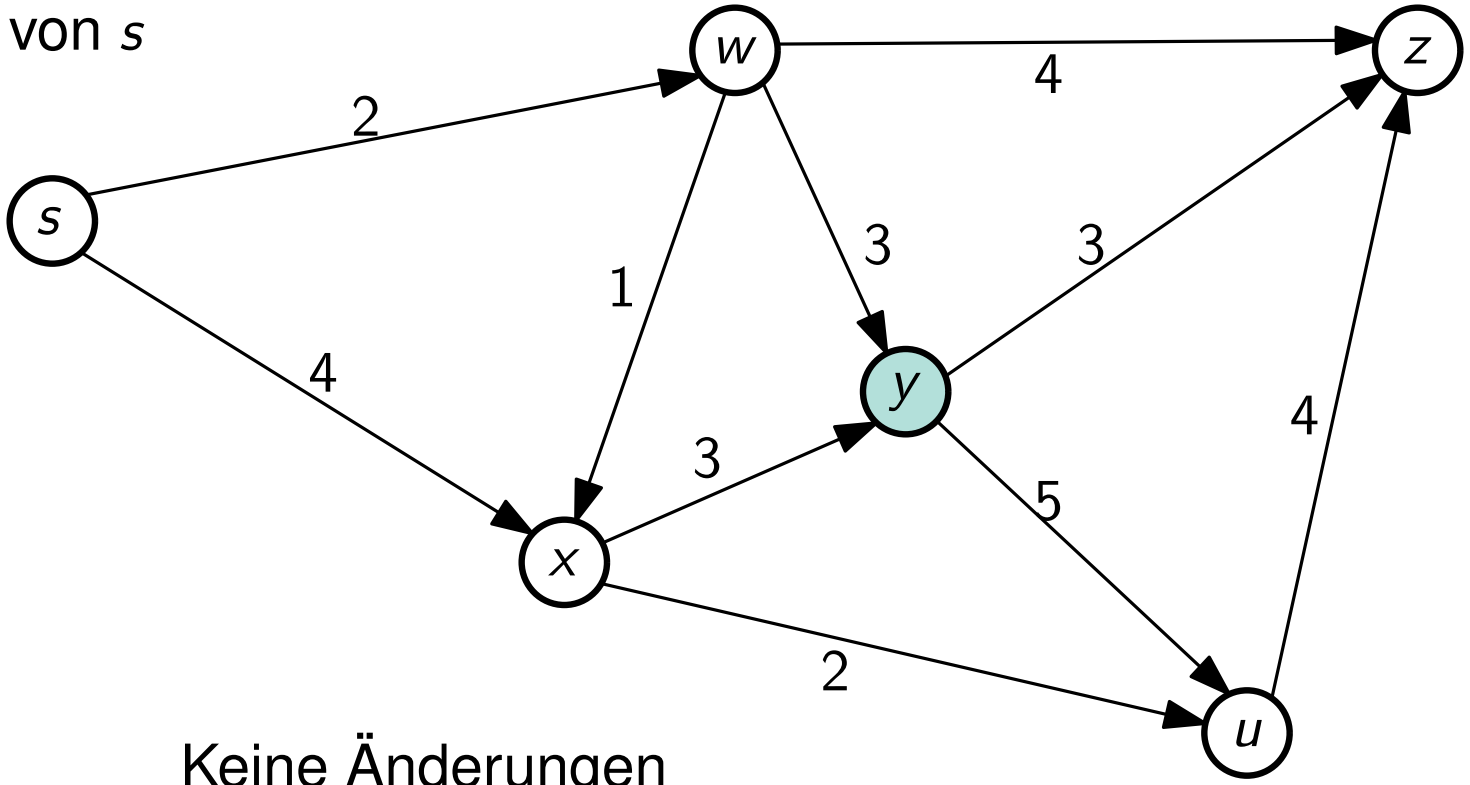
u	*s*	*w*	*x*	*y*	*z*
5	0	2	3	5	6

Priority Queue

y	*u*	*z*
5	5	6

Dijkstra Algorithmus: Beispiel

Dijkstra von *s*



Distanzen

u	*s*	*w*	*x*	*y*	*z*
5	0	2	3	5	6

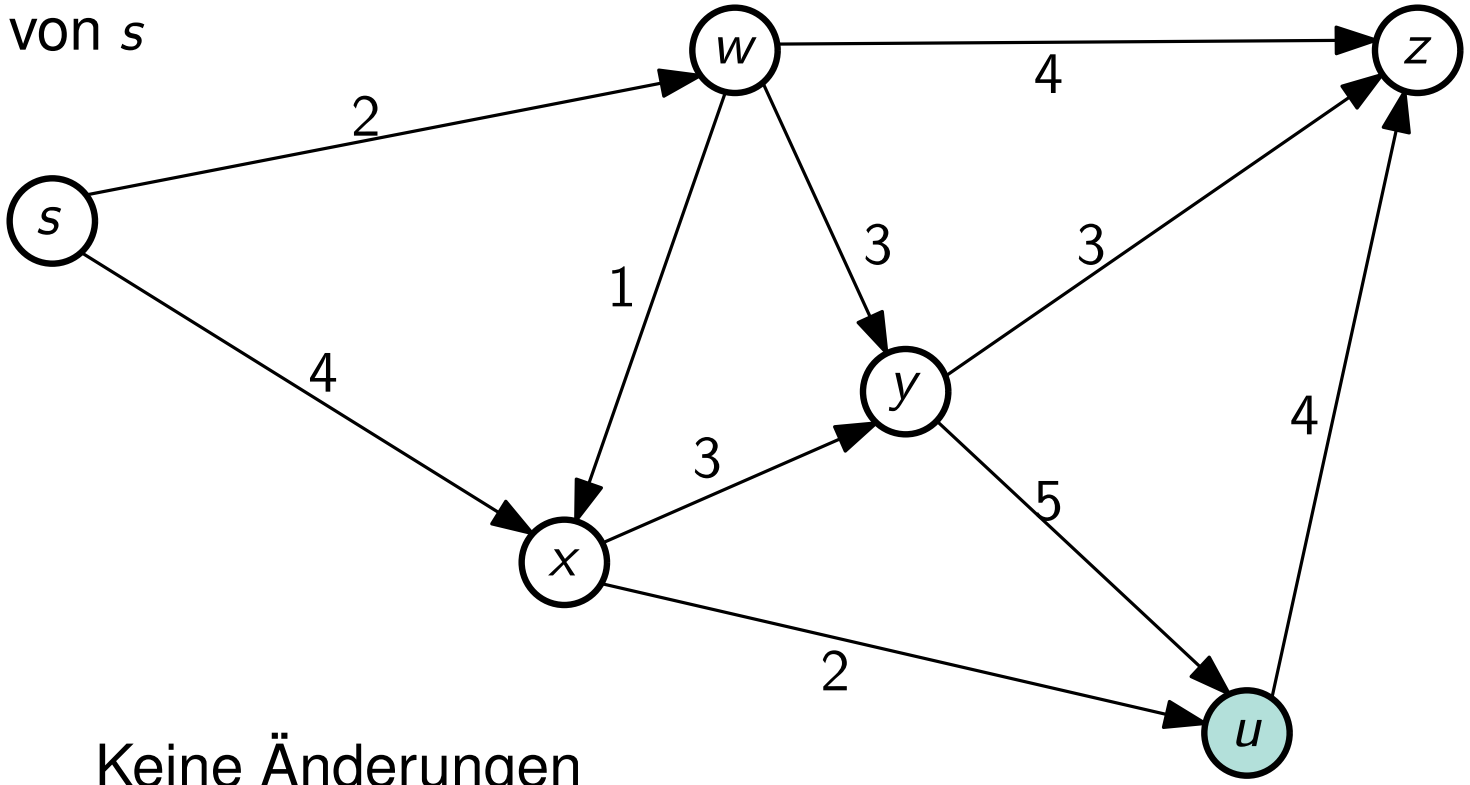
Keine Änderungen

Priority Queue

u	*z*
5	6

Dijkstra Algorithmus: Beispiel

Dijkstra von *s*



Distanzen

u	*s*	*w*	*x*	*y*	*z*
5	0	2	3	5	6

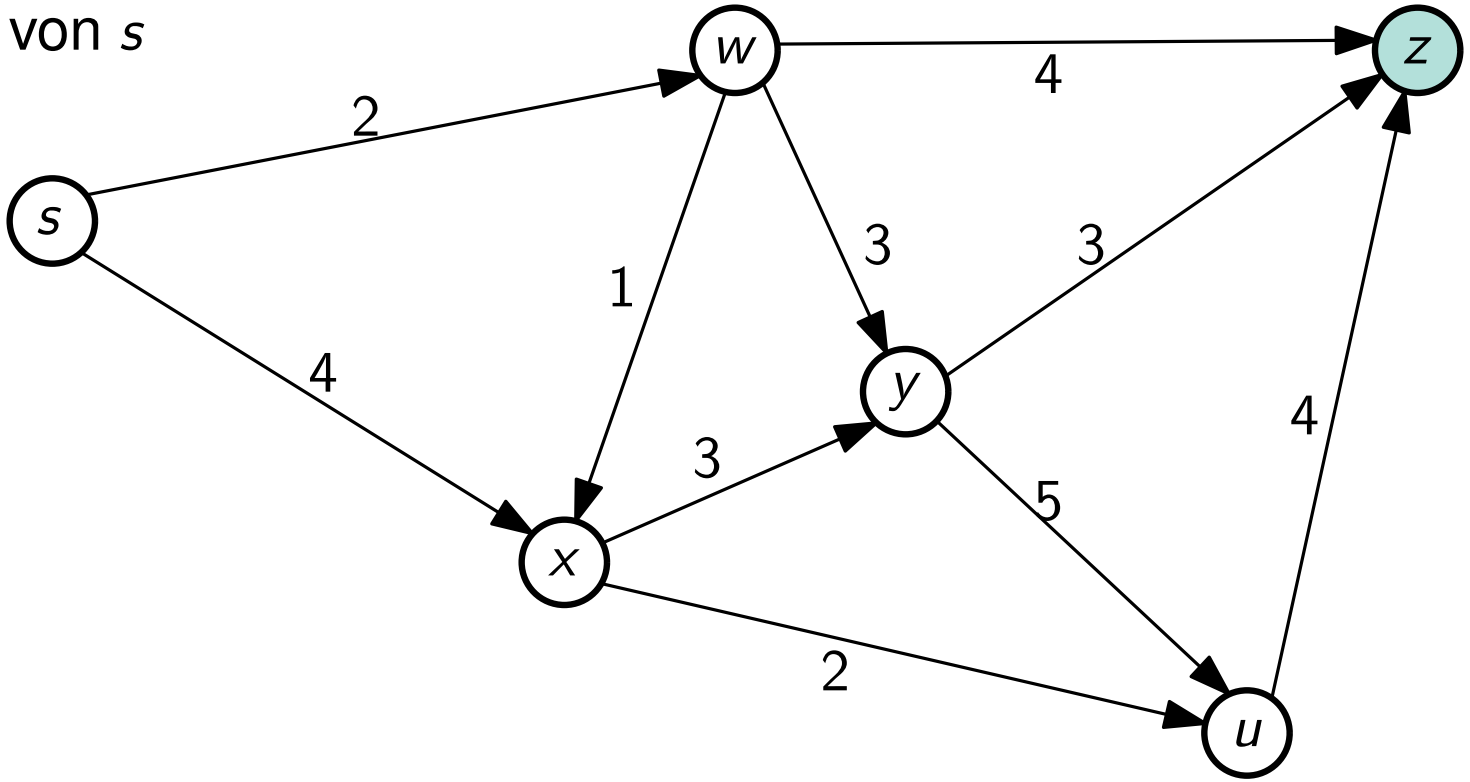
Keine Änderungen

Priority Queue

z
6

Dijkstra Algorithmus: Beispiel

Dijkstra von *s*



Distanzen

u	*s*	*w*	*x*	*y*	*z*
5	0	2	3	5	6

Priority Queue

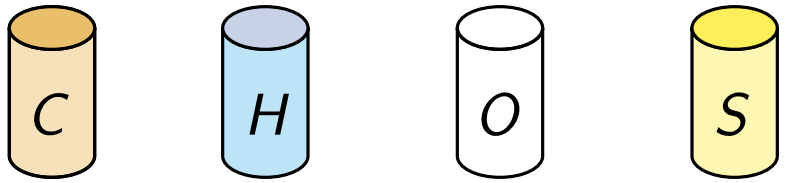
Chemikalien mixen

Bob und Alice wollen $C_3H_8O_{10}P_2$ synthetisieren

Chemikalien mixen

Bob und Alice wollen $C_3H_8O_{10}P_2$ synthetisieren

- Dafür haben sie potentiall unendlich viele Basisstoffe



Chemikalien mixen

Bob und Alice wollen $C_3H_8O_{10}P_2$ synthetisieren

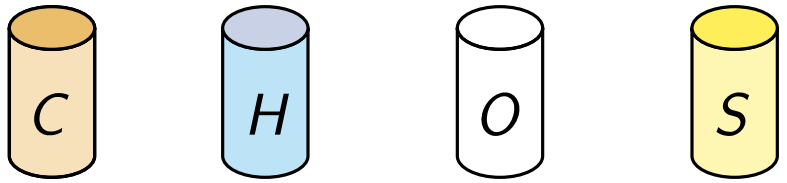
- Dafür haben sie potentiall unendlich viele Basisstoffe
- Jede Reaktion ist entweder endothermisch oder exothermisch
  - Braucht oder produziert also Hitze

$$Fe_2O_3 + 4H_2S \rightarrow 2FeS_2 + 3H_2O + H_2$$

+3 Heat

$$2Cu^2 + R - CHO + 4OH^- \rightarrow Cu_2O + R - COOH + 2H_2O$$

-15 Heat



Chemikalien mixen

Bob und Alice wollen $C_3H_8O_{10}P_2$ synthetisieren

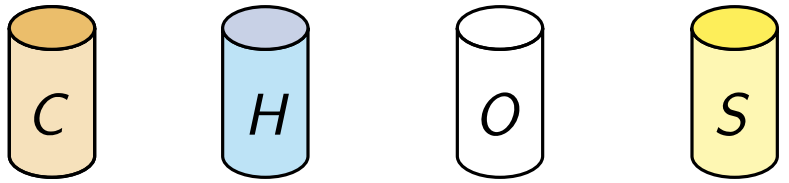
- Dafür haben sie potentiall unendlich viele Basisstoffe
- Jede Reaktion ist entweder endothermisch oder exothermisch
  - Braucht oder produziert also Hitze
- Sie wollen nun herausfinden wie viel Hitze sie mindestens benötigen, um diesen Stoff zu synthetisieren

$$Fe_2O_3 + 4H_2S \rightarrow 2FeS_2 + 3H_2O + H_2$$

+3 Heat

$$2Cu^2 + R - CHO + 4OH^- \rightarrow Cu_2O + R - COOH + 2H_2O$$

-15 Heat



Chemikalien mixen: Graphmodellierung

Das kann man als Graph modellieren! (sort-of)

Eingabe:

- Liste an möglichen chemischen Stoffen
- Liste an chemischen Reaktionen
  - positiver oder negativer Hitzebedarf

Chemikalien mixen: Graphmodellierung

Das kann man als Graph modellieren! (sort-of)

Eingabe:

- Liste an möglichen chemischen Stoffen
- Liste an chemischen Reaktionen
  - positiver oder negativer Hitzebedarf

Eingabegraph:

- Ein Knoten pro Stoff
- Kante zwischen zwei Stoffen, falls der eine aus dem anderen per Reaktion hervorgeht
  - positives oder negatives Kantengewicht je nach Hitzebedarf

Chemikalien mixen: Graphmodellierung

Das kann man als Graph modellieren! (sort-of)

Eingabe:

- Liste an möglichen chemischen Stoffen
- Liste an chemischen Reaktionen
  - positiver oder negativer Hitzebedarf

Problemstellung:

- Finde Folge von Reaktionen zu einem Zielstoff mit möglichst niedriger Gesamthitze

Eingabegraph:

- Ein Knoten pro Stoff
- Kante zwischen zwei Stoffen, falls der eine aus dem anderen per Reaktion hervorgeht
  - positives oder negatives Kantengewicht je nach Hitzebedarf

Chemikalien mixen: Graphmodellierung

Das kann man als Graph modellieren! (sort-of)

Eingabe:

- Liste an möglichen chemischen Stoffen
- Liste an chemischen Reaktionen
  - positiver oder negativer Hitzebedarf

Problemstellung:

- Finde Folge von Reaktionen zu einem Zielstoff mit möglichst niedriger Gesamthitze

Eingabegraph:

- Ein Knoten pro Stoff
- Kante zwischen zwei Stoffen, falls der eine aus dem anderen per Reaktion hervorgeht
  - positives oder negatives Kantengewicht je nach Hitzebedarf

Graphproblem:

- Finde kürzesten Weg zwischen zwei Knoten

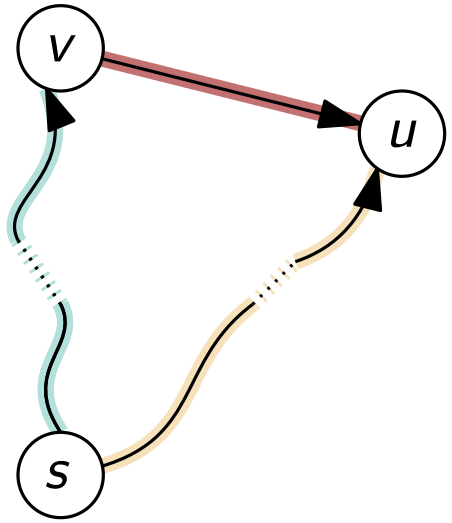
Kürzeste Wege: Negative Kantengewichte

Jetzt könnten Kanten negative Gewichte haben

Kürzeste Wege: Negative Kantengewichte

Jetzt könnten Kanten negative Gewichte haben

- Gehe wie in Dijkstras Algorithmus vor:



vorläufige $s-v$ und $s-u$ Pfade gegeben

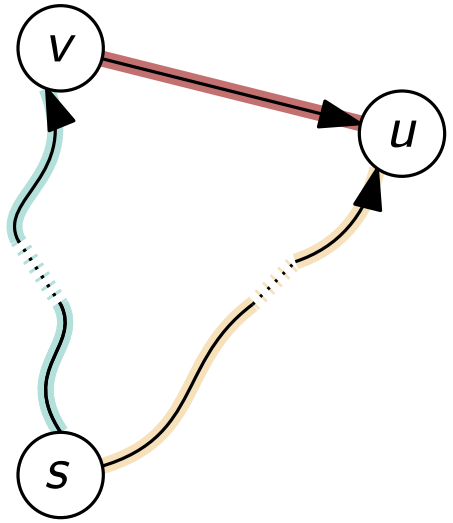
Setze $d(s,u) = \min(d(s,u), d(s,v) + \text{len}(v,u))$

- “ Ist es billiger über v zu u zu gehen? ”

Kürzeste Wege: Negative Kantengewichte

Jetzt könnten Kanten negative Gewichte haben

- Gehe wie in Dijkstras Algorithmus vor:



vorläufige $s-v$ und $s-u$ Pfade gegeben

Setze $d(s,u) = \min(d(s,u), d(s,v) + \text{len}(v,u))$

- “ Ist es billiger über v zu u zu gehen? ”

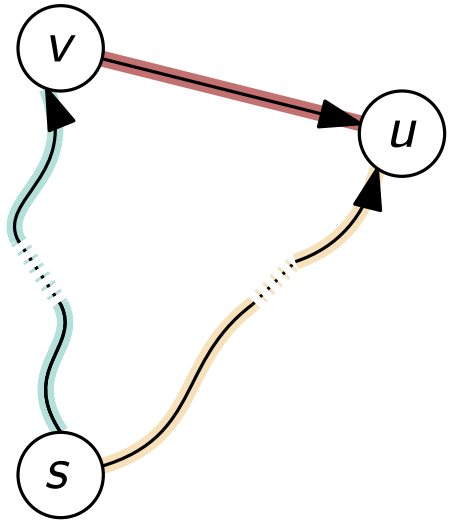
Dijkstras Arbeitsannahme:
Wenn ein Knoten exploriert wird, ist seine Distanz optimal

- Stimmt nicht mehr!

Kürzeste Wege: Negative Kantengewichte

Jetzt könnten Kanten negative Gewichte haben

- Gehe wie in Dijkstras Algorithmus vor:



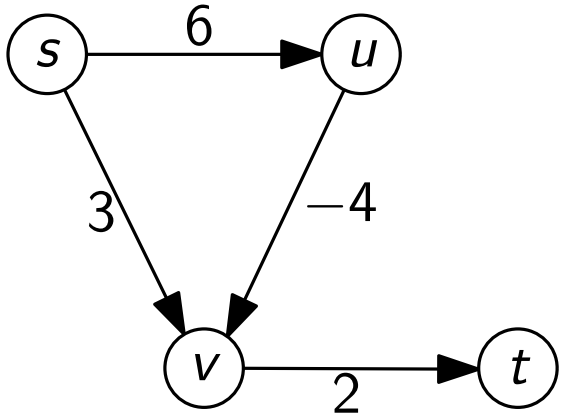
vorläufige $s-v$ und $s-u$ Pfade gegeben

Setze $d(s,u) = \min(d(s,u), d(s,v) + \text{len}(v,u))$

- " Ist es billiger über v zu u zu gehen? "

Dijkstras Arbeitsannahme:
Wenn ein Knoten exploriert wird, ist seine Distanz optimal

- Stimmt nicht mehr!



- v wird zuerst mit Distanz 3 und danach t mit Distanz 5 entdeckt
- Aber $d(s,v) = 2$ und $d(s,t) = 4$

Wie viele Relaxierungen reichen?

Einmalige Relaxierungen der Kanten reicht nicht aus

Pro Kante (u, v): Setze $d(s, u) = \min(d(s, u), d(s, v) + \text{len}(v, u))$

- Nach einer Runde Relaxierung könnten sich $d(s, u)$ und $d(s, v)$ verändert haben
  - Kante (u, v) könnte sich jetzt lohnen

Wie viele Relaxierungen reichen?

Einmalige Relaxierungen der Kanten reicht nicht aus

Pro Kante (u, v): Setze $d(s, u) = \min(d(s, u), d(s, v) + \text{len}(v, u))$

- Nach einer Runde Relaxierung könnten sich $d(s, u)$ und $d(s, v)$ verändert haben
  - Kante (u, v) könnte sich jetzt lohnen

$\Longrightarrow$ Relaxiere nochmal alle Kanten

- Nach wie vielen Relaxierungen sind wir fertig?

Wie viele Relaxierungen reichen?

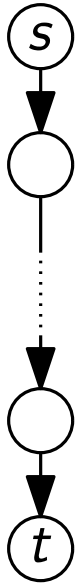
Einmalige Relaxierungen der Kanten reicht nicht aus

Pro Kante (u, v): Setze $d(s, u) = \min(d(s, u), d(s, v) + \text{len}(v, u))$

- Nach einer Runde Relaxierung könnten sich $d(s, u)$ und $d(s, v)$ verändert haben
  - Kante (u, v) könnte sich jetzt lohnen

$\Longrightarrow$ Relaxiere nochmal alle Kanten

- Nach wie vielen Relaxierungen sind wir fertig?



Keine negativen Kreise:
Ein kürzester Weg ist kreisfrei

Wie viele Relaxierungen reichen?

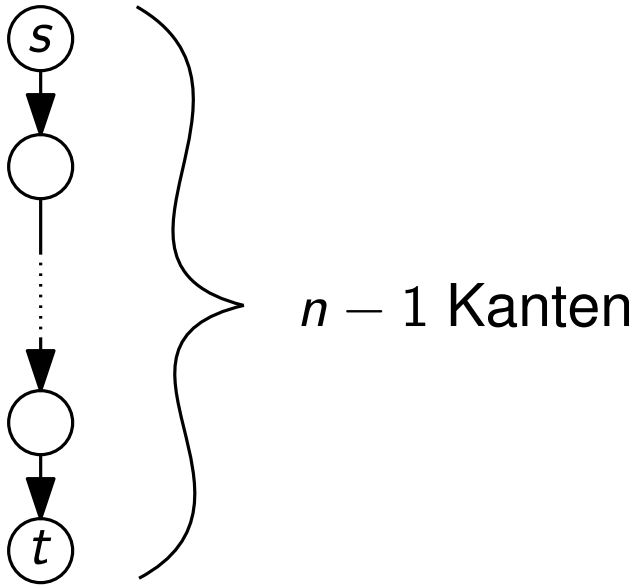
Einmalige Relaxierungen der Kanten reicht nicht aus

Pro Kante (u, v): Setze $d(s, u) = \min(d(s, u), d(s, v) + \text{len}(v, u))$

- Nach einer Runde Relaxierung könnten sich $d(s, u)$ und $d(s, v)$ verändert haben
  - Kante (u, v) könnte sich jetzt lohnen

$\Longrightarrow$ Relaxiere nochmal alle Kanten

- Nach wie vielen Relaxierungen sind wir fertig?



Keine negativen Kreise:
Ein kürzester Weg ist kreisfrei

$\Longrightarrow$ Maximal $n - 1$ Kanten

$\Longrightarrow$ $n - 1$ Iterationen reichen

Wie viele Relaxierungen reichen?

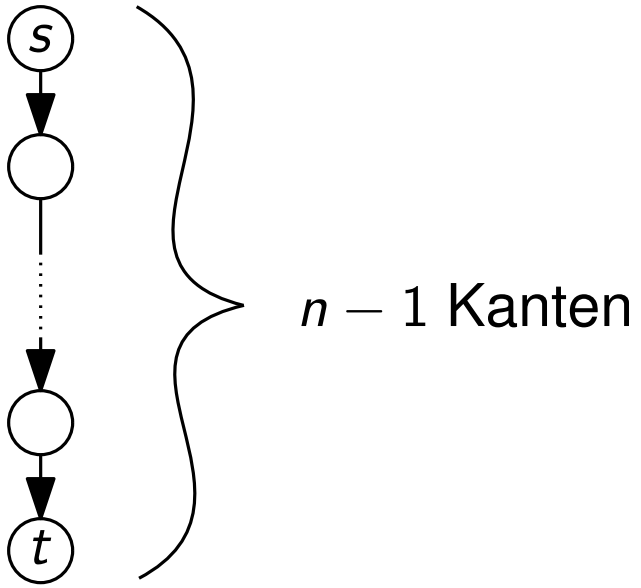
Einmalige Relaxierungen der Kanten reicht nicht aus

Pro Kante (u, v): Setze $d(s, u) = \min(d(s, u), d(s, v) + \text{len}(v, u))$

- Nach einer Runde Relaxierung könnten sich $d(s, u)$ und $d(s, v)$ verändert haben
  - Kante (u, v) könnte sich jetzt lohnen

$\Longrightarrow$ Relaxiere nochmal alle Kanten

- Nach wie vielen Relaxierungen sind wir fertig?



Keine negativen Kreise:
Ein kürzester Weg ist kreisfrei

$\Longrightarrow$ Maximal $n - 1$ Kanten

$\Longrightarrow$ $n - 1$ Iterationen reichen

Neues Problem: Negative Kreise erkennen

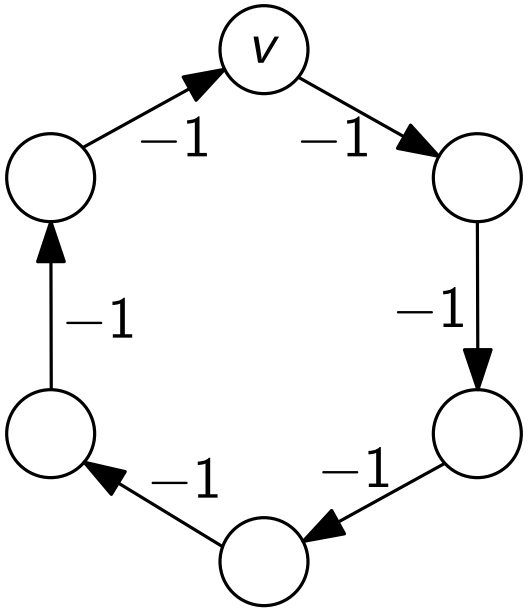
Negative Kreise

Warum sind negative Kreise böse?

Negative Kreise

Warum sind negative Kreise böse?

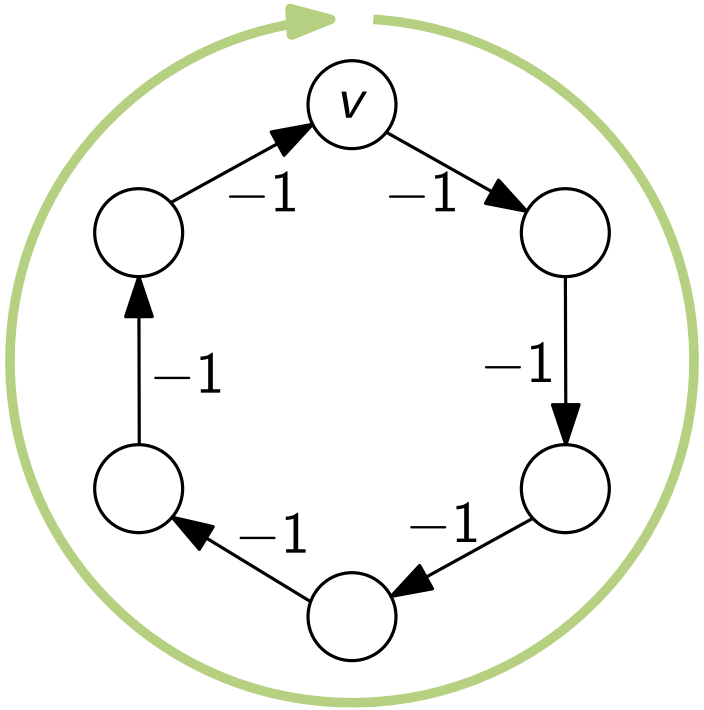
- Distanzen stabilisieren sich nicht



Negative Kreise

Warum sind negative Kreise böse?

- Distanzen stabilisieren sich nicht

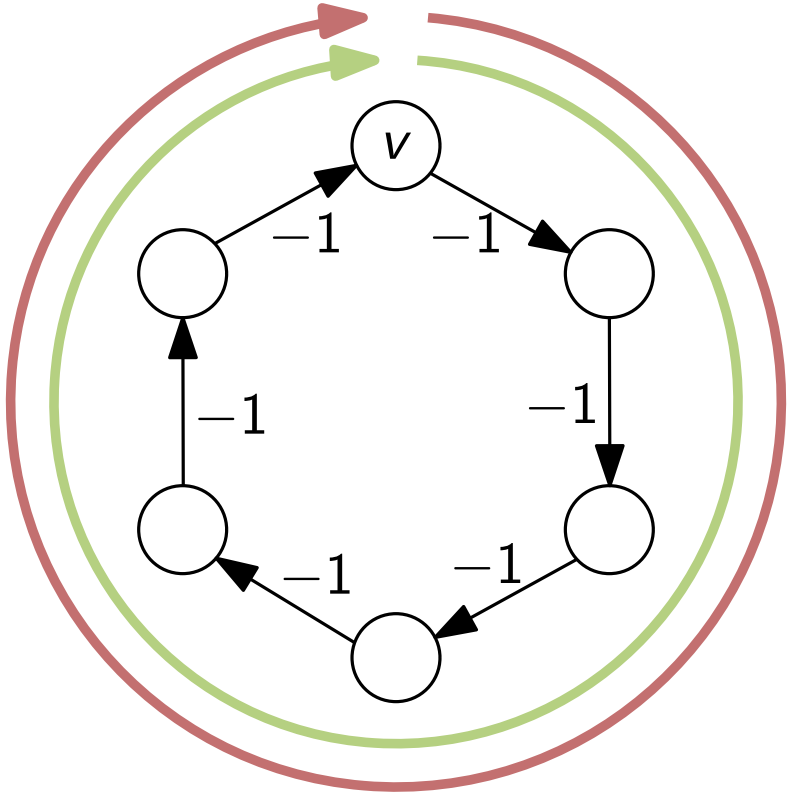


Eine Umdrehung: $d(v, v) = -6$

Negative Kreise

Warum sind negative Kreise böse?

- Distanzen stabilisieren sich nicht



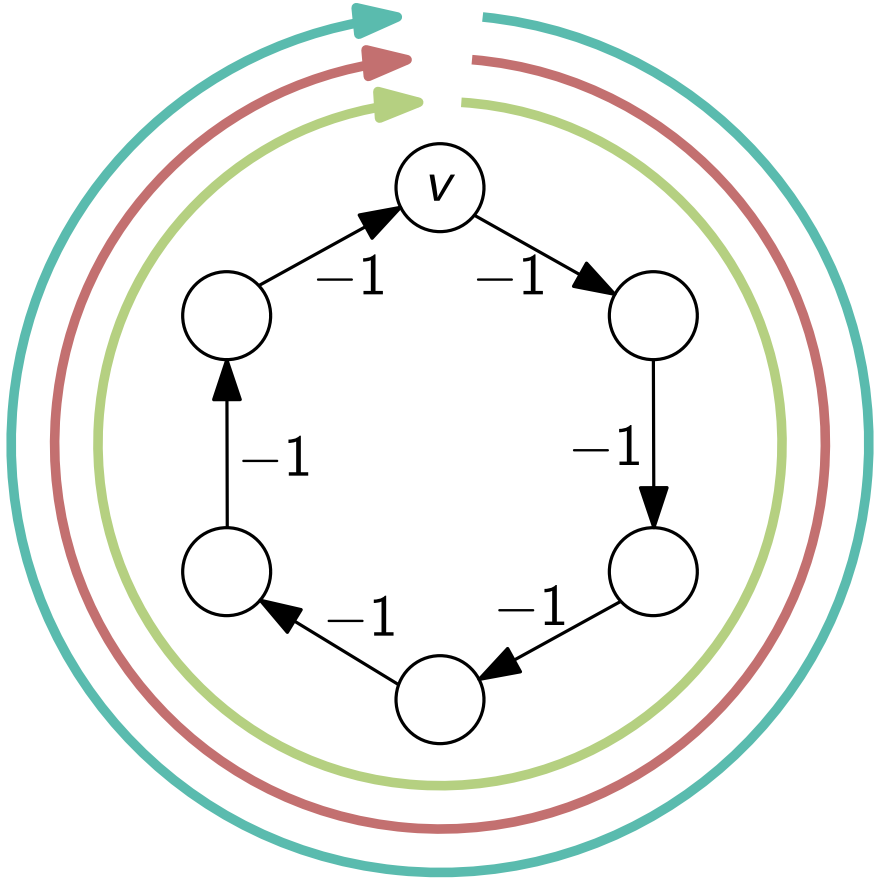
Eine Umdrehung: $d(v, v) = -6$

Zwei Umdrehungen: $d(v, v) = -12$

Negative Kreise

Warum sind negative Kreise böse?

- Distanzen stabilisieren sich nicht



Eine Umdrehung: $d(v, v) = -6$

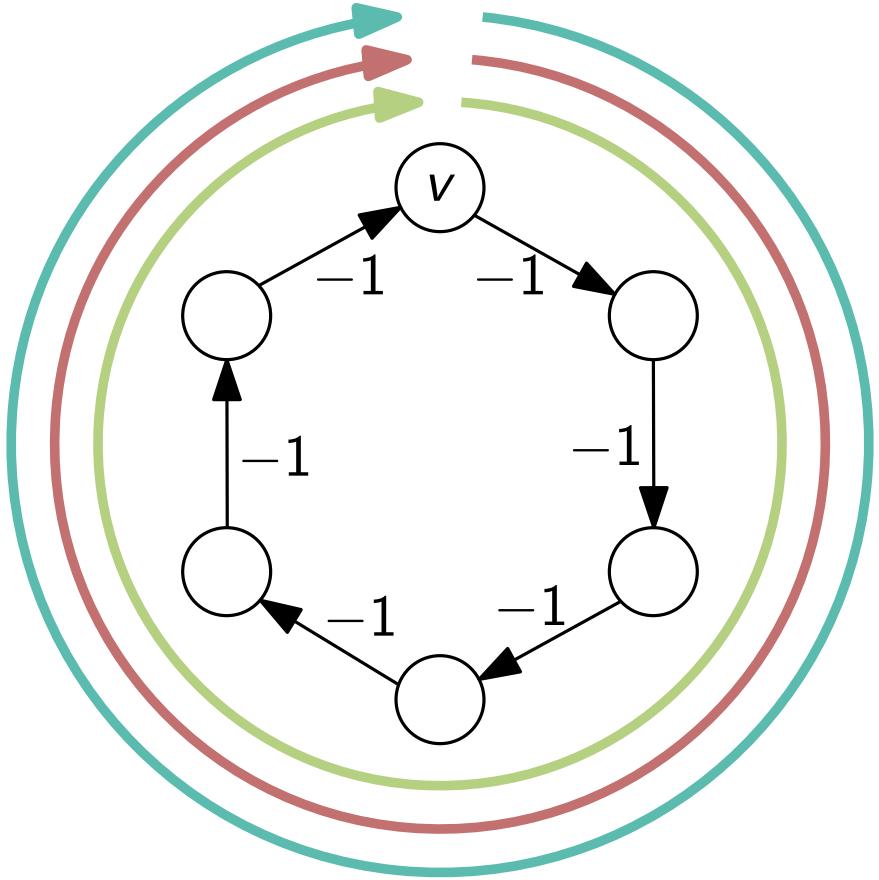
Zwei Umdrehungen: $d(v, v) = -12$

Drei Umdrehungen: $d(v, v) = -18$

Negative Kreise

Warum sind negative Kreise böse?

- Distanzen stabilisieren sich nicht
- $d(s,t) = -\infty$ **was???**



Eine Umdrehung: $d(v,v) = -6$

Zwei Umdrehungen: $d(v,v) = -12$

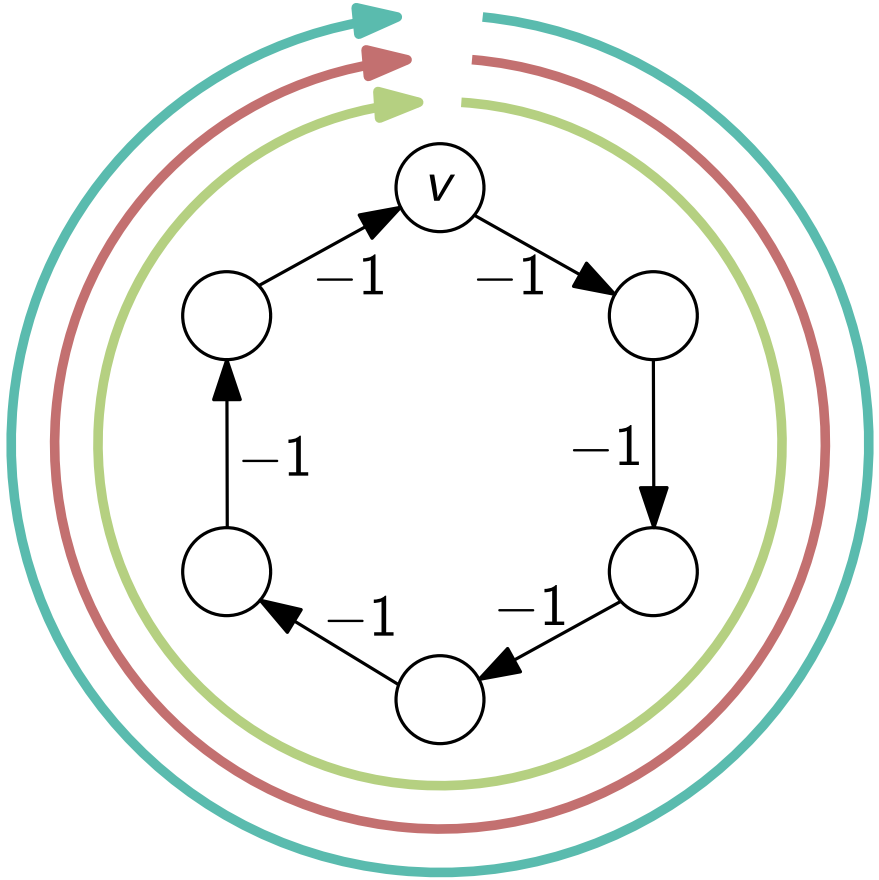
Drei Umdrehungen: $d(v,v) = -18$

Negative Kreise

Warum sind negative Kreise böse?

- Distanzen stabilisieren sich nicht
- $d(s,t) = -\infty$ **was???**

Wie kann man negative Kreise erkennen?



Eine Umdrehung: $d(v,v) = -6$

Zwei Umdrehungen: $d(v,v) = -12$

Drei Umdrehungen: $d(v,v) = -18$

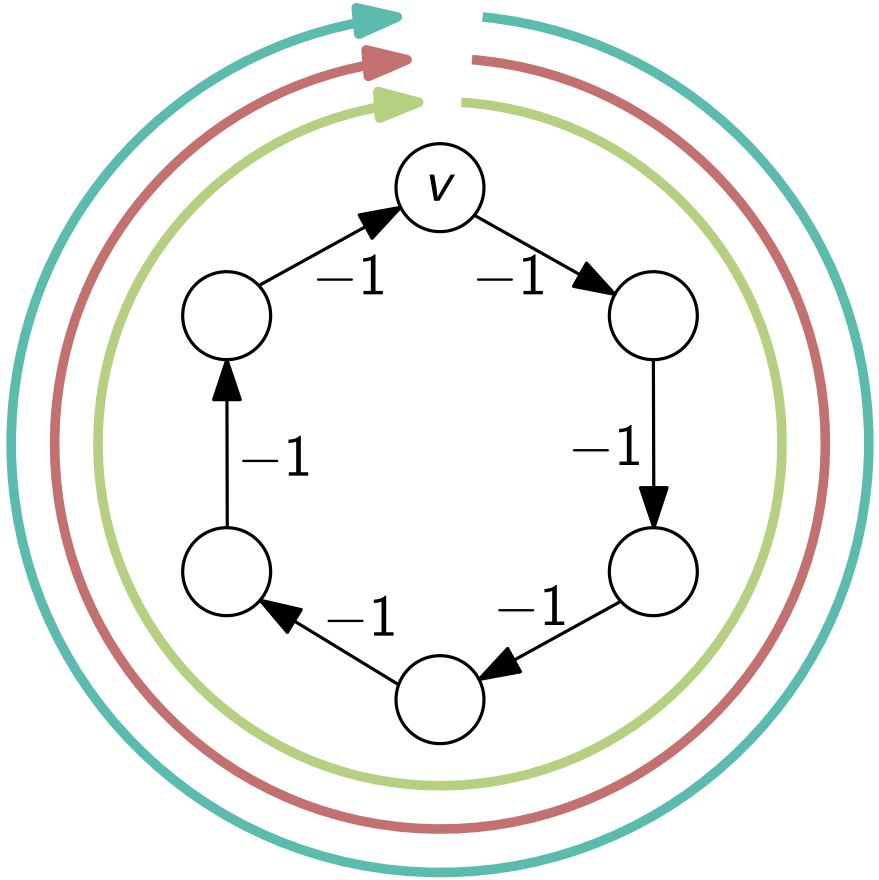
Negative Kreise

Warum sind negative Kreise böse?

- Distanzen stabilisieren sich nicht
- $d(s,t) = -\infty$ **was???**

Wie kann man negative Kreise erkennen?

- Keine negativen Kreise $\Longrightarrow$
  Nach $n-1$ Iteration sind Distanzen stabil
- Negative Kreise $\Longrightarrow$
  Nach n-ter Iteration ändern sich Distanzen



Eine Umdrehung: $d(v,v) = -6$

Zwei Umdrehungen: $d(v,v) = -12$

Drei Umdrehungen: $d(v,v) = -18$

Negative Kreise

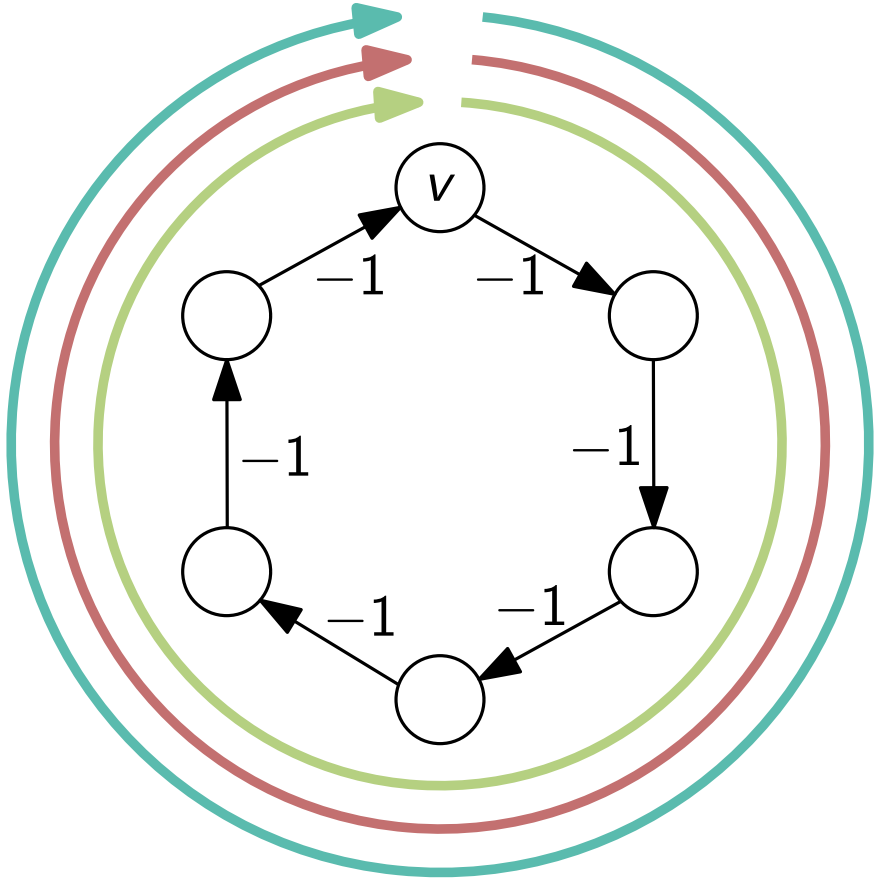
Warum sind negative Kreise böse?

- Distanzen stabilisieren sich nicht
- $d(s,t) = -\infty$ **was???**

Wie kann man negative Kreise erkennen?

- Keine negativen Kreise $\Longrightarrow$
  Nach $n-1$ Iteration sind Distanzen stabil
- Negative Kreise $\Longrightarrow$
  Nach n-ter Iteration ändern sich Distanzen

Führe n-te Iteration durch und prüfe, ob sich Distanzen ändern



Eine Umdrehung: $d(v,v) = -6$

Zwei Umdrehungen: $d(v,v) = -12$

Drei Umdrehungen: $d(v,v) = -18$

Bellman-Fords Algorithmus: Pseudocode

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

Bellman-Fords Algorithmus: Pseudocode

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

- In $d[v]$ speichern wir uns die aktuelle beste Distanz von s zu v
- Nach dem Algorithmus wird d ausgegeben

Bellman-Fords Algorithmus: Pseudocode

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

- Jeder Pfad kann maximal $n - 1$ Kanten haben
- Nach $n - 1$ Iterationen sind wir definitiv fertig
  - Falls wir keinen negativen Kreis haben

Bellman-Fords Algorithmus: Pseudocode

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

- Relaxiere jede Kante pro Iteration
- Falls ein besserer Weg gefunden wurde, aktualisiere Distanz

Bellman-Fords Algorithmus: Pseudocode

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

- Falls sich nach einer n-ten Iteration Distanzen verändern, gibt es einen negativen Kreis

Bellman-Fords Algorithmus: Laufzeit

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

Welche Laufzeit hat dieser Algorithmus?

Bellman-Fords Algorithmus: Laufzeit

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

Welche Laufzeit hat dieser Algorithmus?

- n Iterationen der Kantenrelaxierungen

Bellman-Fords Algorithmus: Laufzeit

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

Welche Laufzeit hat dieser Algorithmus?

- n Iterationen der Kantenrelaxierungen
- In jeder Iteration wird jede der m Kanten betrachtet

Bellman-Fords Algorithmus: Laufzeit

```
BellmanFord(Graph G, Node s)
    d := Array of size n initialized with ∞
    d[s] := 0
    for n − 1 iterations do
        for Edge (u, v) ∈ E do
            if d[v] > d[u] + len(u, v) then
                d[v] := d[u] + len(u, v)
    // test for negative cycle
    for Edge (u, v) ∈ E do
        if d[v] > d[u] + len(u, v) then
            return negative cycle
    return d
```

Welche Laufzeit hat dieser Algorithmus?

- n Iterationen der Kantenrelaxierungen
- In jeder Iteration wird jede der m Kanten betrachtet

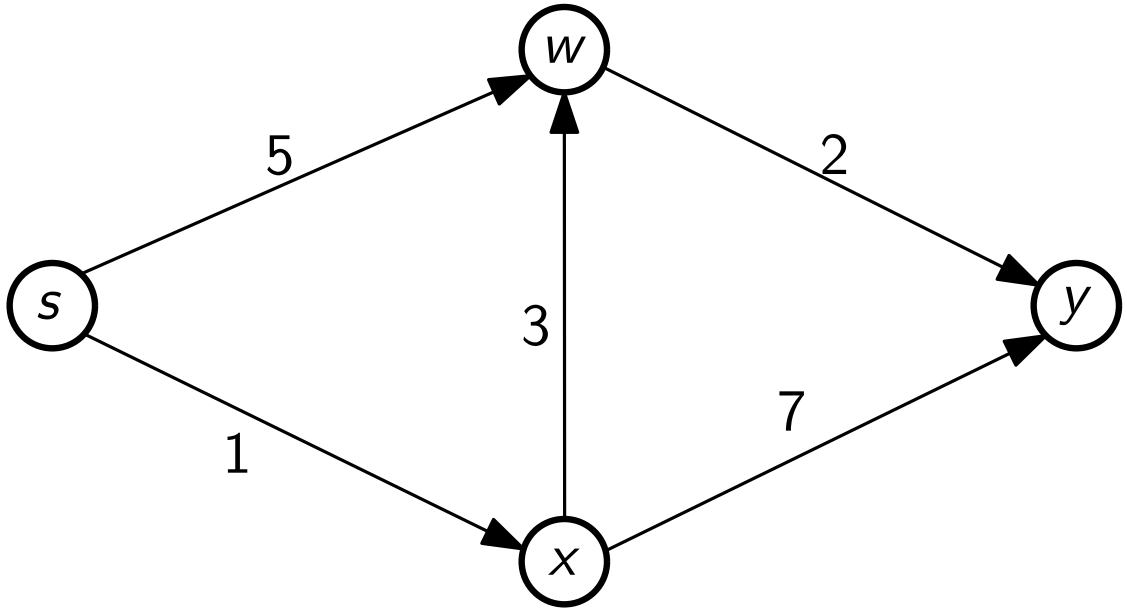
Insgesamt $\Theta(n \cdot m)$

Bellman-Fords Algorithmus hat eine Laufzeit von $\Theta(nm)$

Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	∞	∞	∞



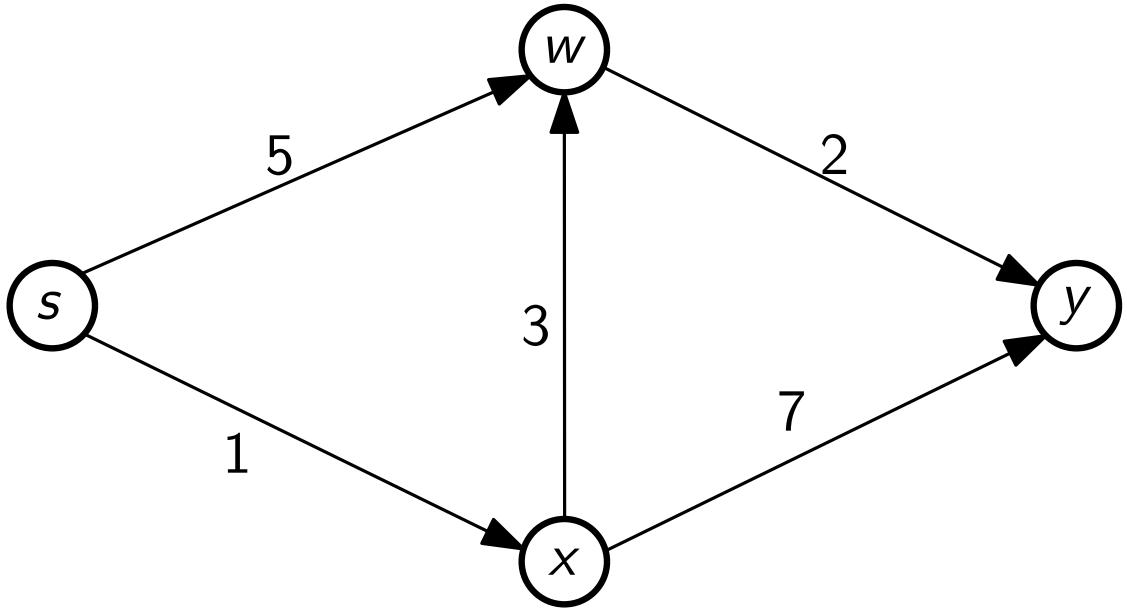
Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			

Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	5	∞	∞

Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			



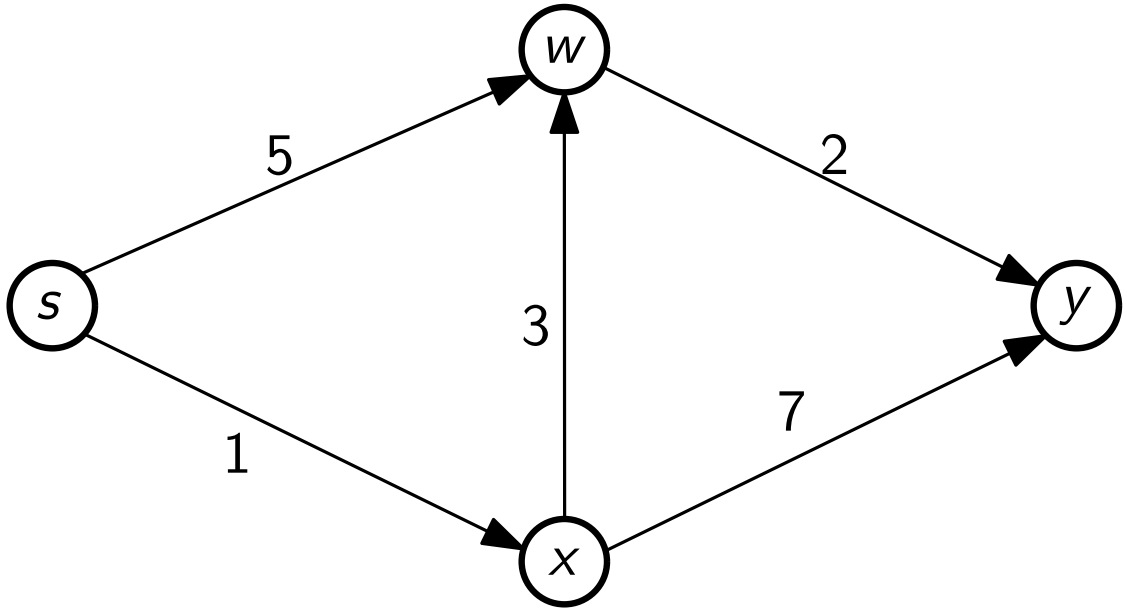
Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	5	∞	∞

Iteration: 1 2 3 4

Kante:
(s, w)
(x, w)
(w, y)
(s, x)
(x, y)

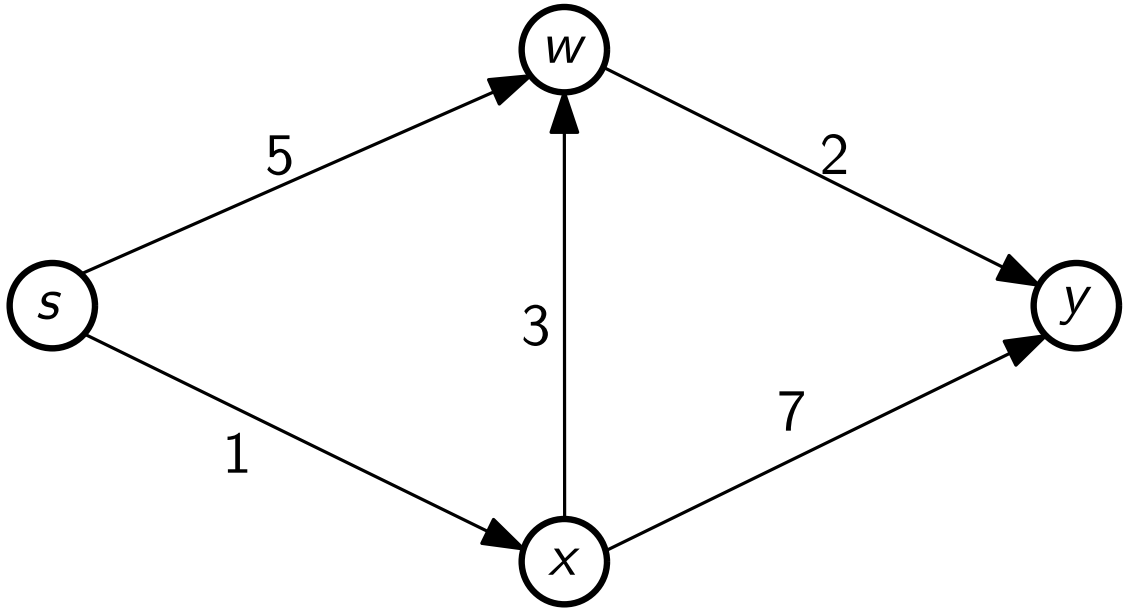


Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	5	∞	7

Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			

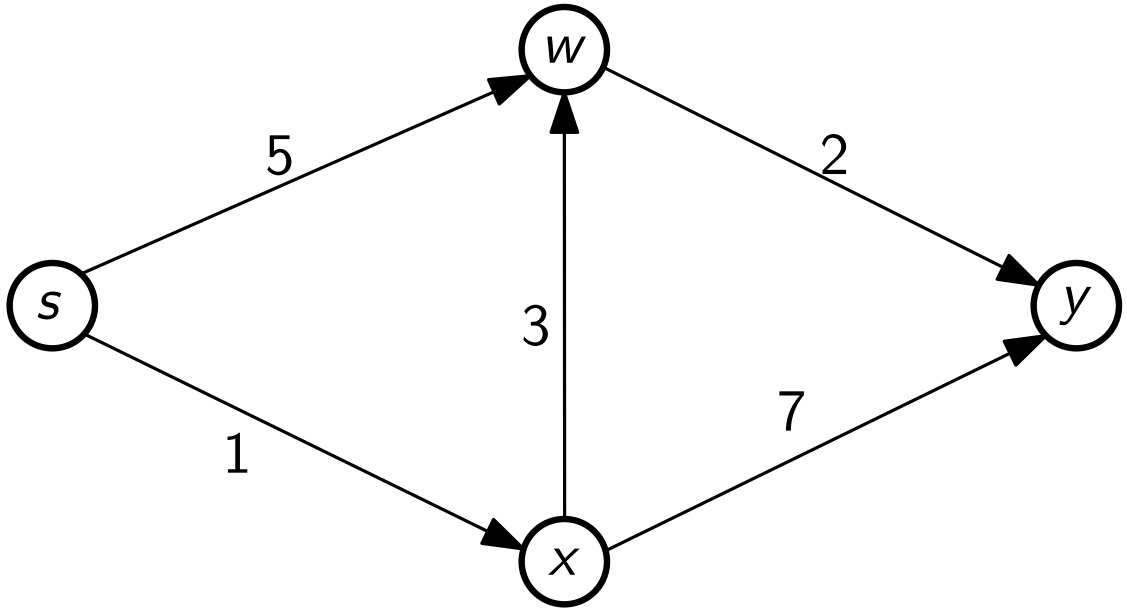


Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	5	1	7

Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			



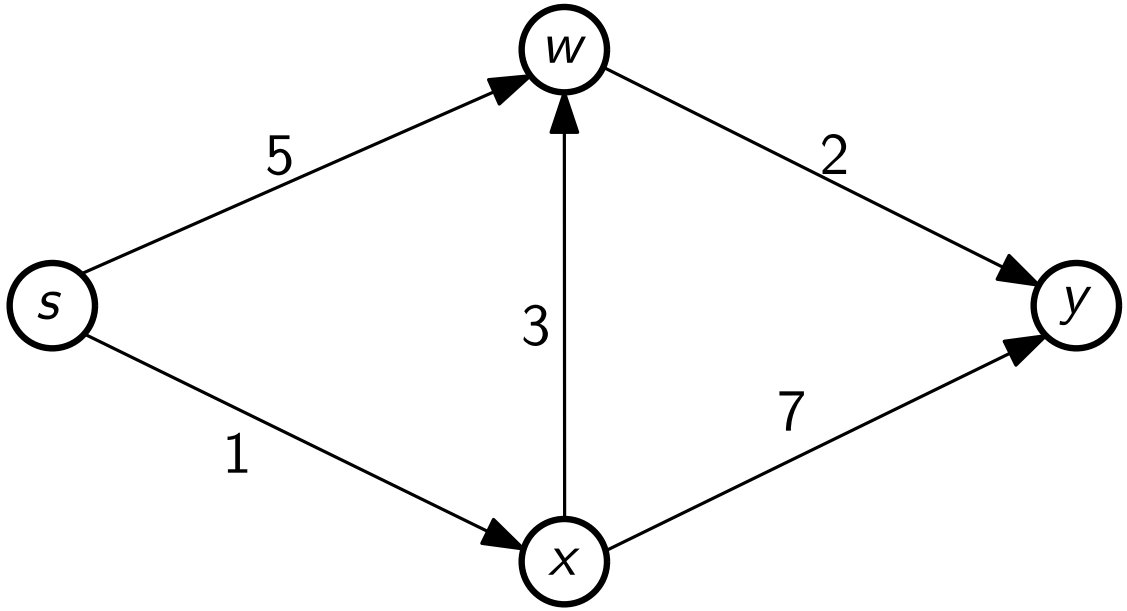
Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	5	1	7

Iteration: 1 2 3 4

Kante: (s, w) (x, w) (w, y) (s, x) (x, y)

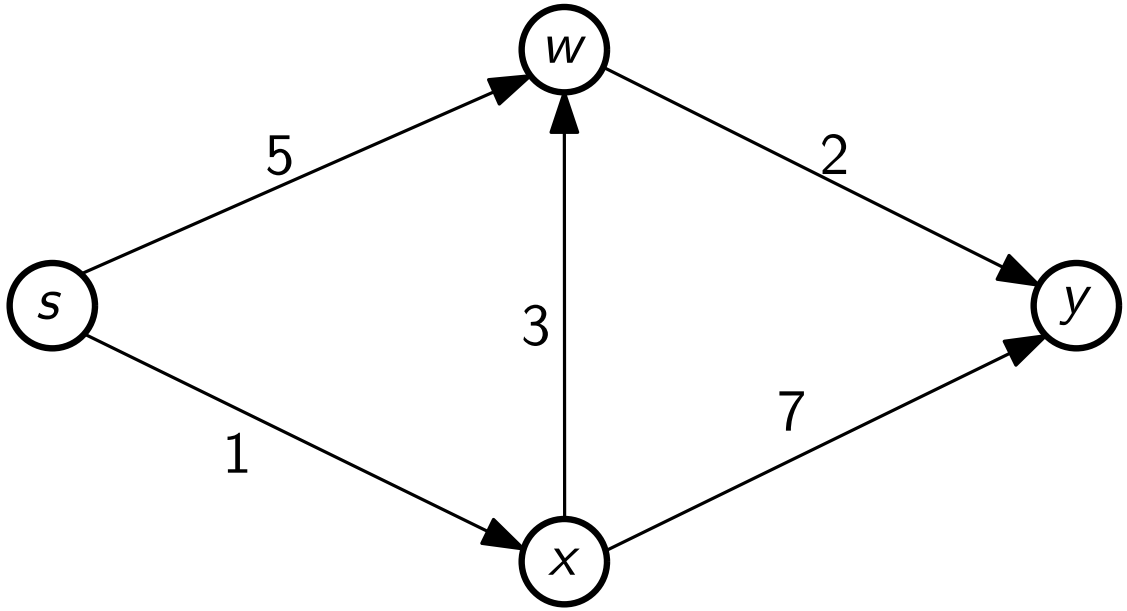


Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	5	1	7

Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			

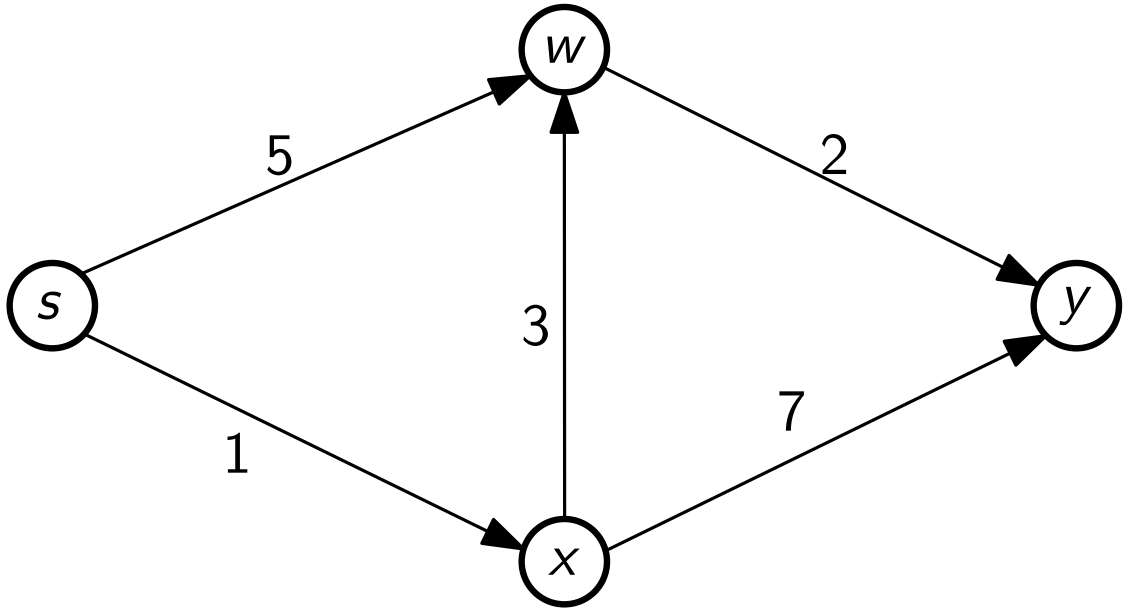


Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	4	1	7

Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			

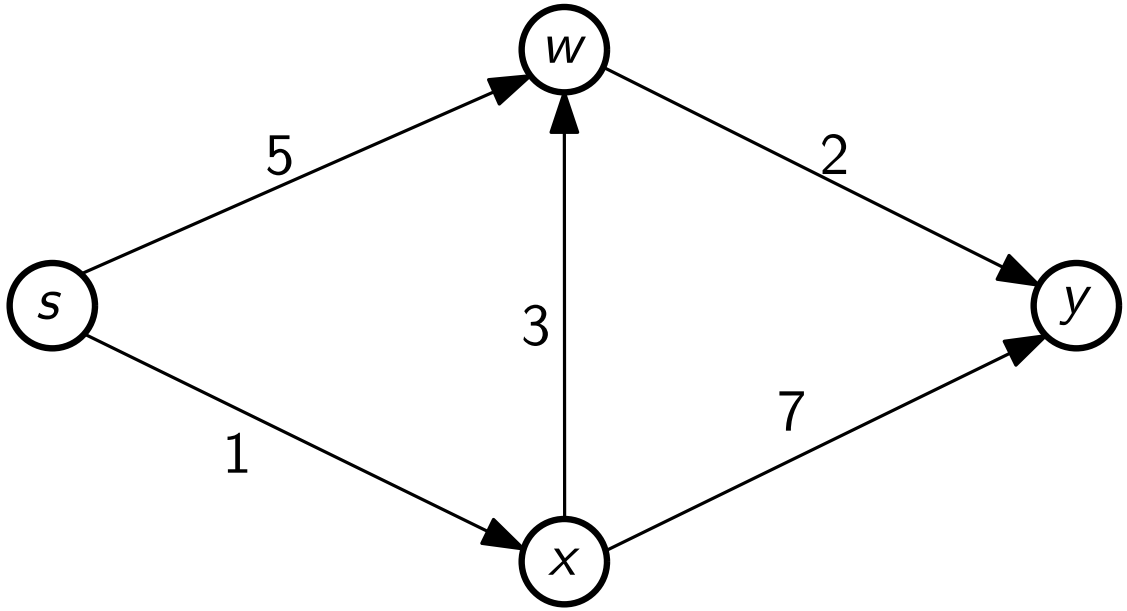


Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	4	1	6

Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			

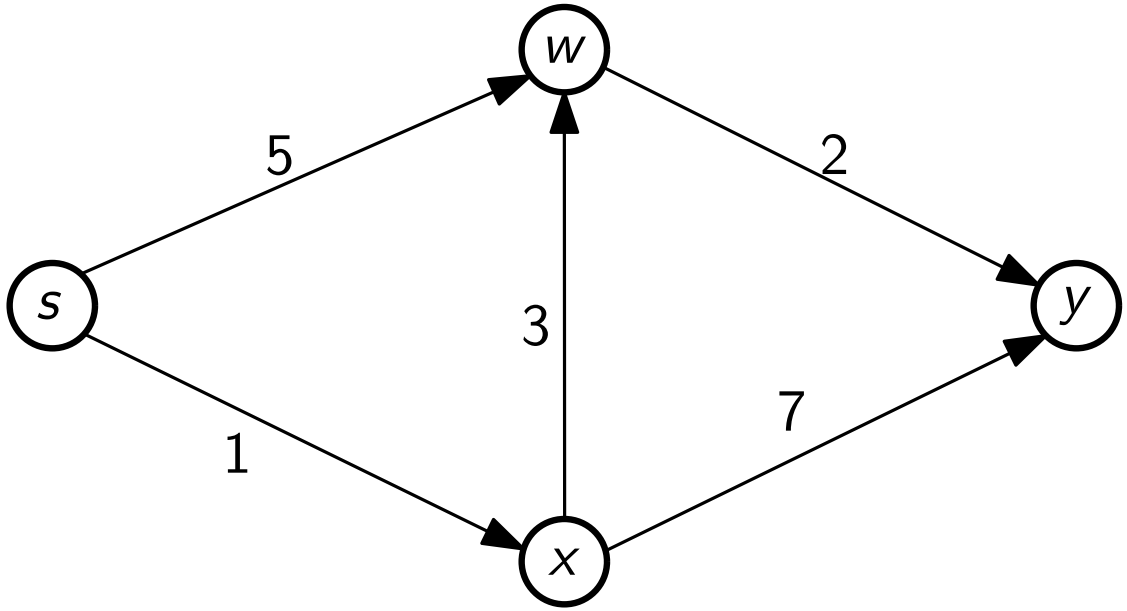


Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	4	1	6

Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			

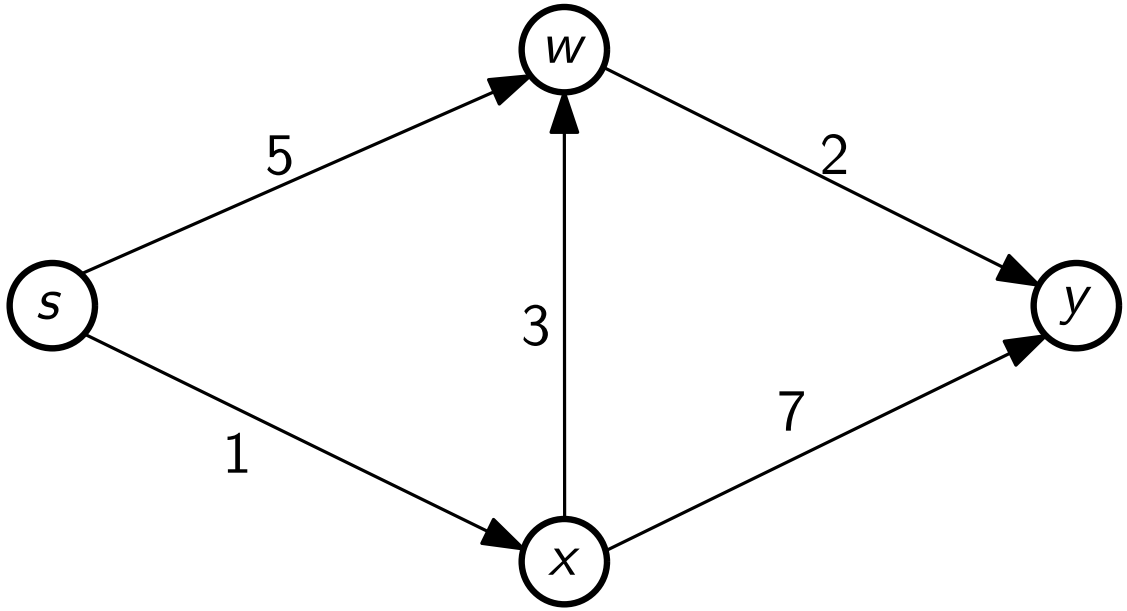


Bellman-Fords Algorithmus: Beispiel

Distanzen

s	w	x	y
0	4	1	6

Iteration:	1	2	3	4
Kante:	(s, w)			
	(x, w)			
	(w, y)			
	(s, x)			
	(x, y)			



Bellman-Fords Algorithmus: Beispiel

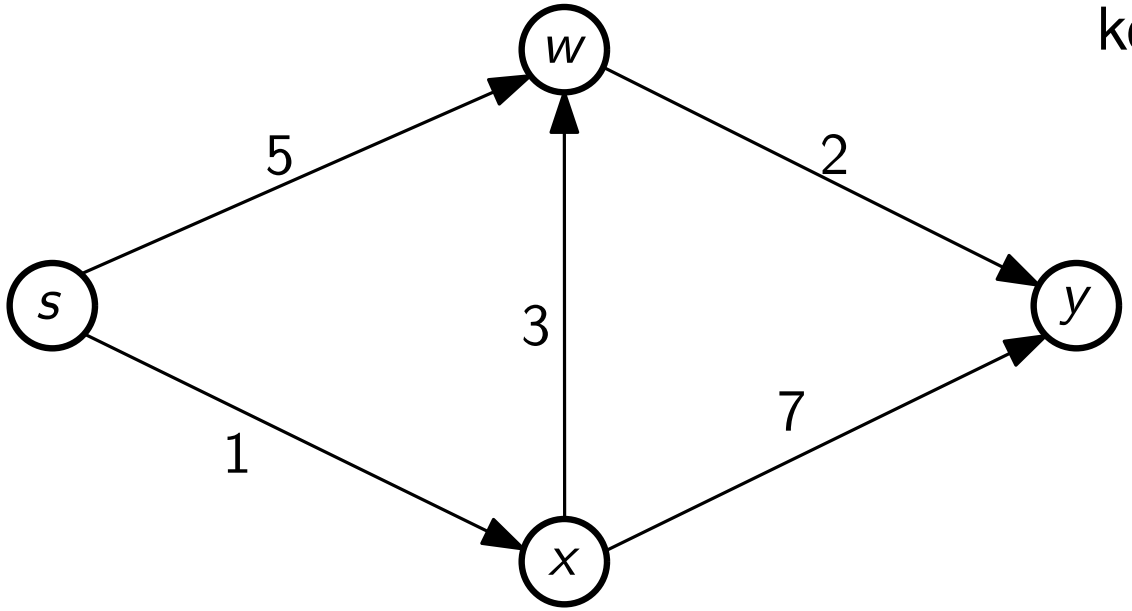
Distanzen

s	w	x	y
0	4	1	6

keine Änderungen!

Iteration: 1 2 3 4

Kante: (s, w) (x, w) (w, y) (s, x) (x, y)



Bellman-Fords Algorithmus: Beispiel

Distanzen

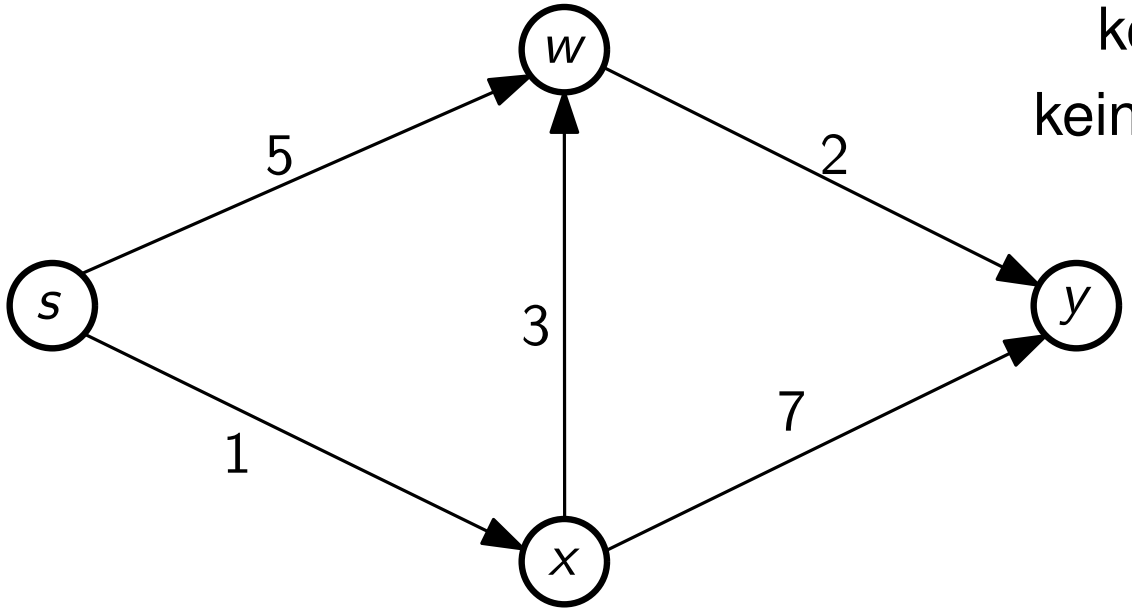
s	w	x	y
0	4	1	6

keine Änderungen!

keine negativen Kreise!

Iteration: 1 2 3 4

Kante: (s, w) (x, w) (w, y) (s, x) (x, y)



Lieferrouten

Bobs neues Logistikunternehmen möchte Lieferrouten bestimmen

Lieferrouten

Bobs neues Logistikunternehmen möchte Lieferrouten bestimmen

- Kürzeste-Wege-Algorithmen hilfreich
- Aber: Wir wollen nicht jedes mal einen Algorithmus laufen lassen (teuer)
- Berechne ein mal alle kürzesten Wege vor und frage danach nur Ergebnisse ab

Lieferrouten

Bobs neues Logistikunternehmen möchte Lieferrouten bestimmen

- Kürzeste-Wege-Algorithmen hilfreich
- Aber: Wir wollen nicht jedes mal einen Algorithmus laufen lassen (teuer)
- Berechne ein mal alle kürzesten Wege vor und frage danach nur Ergebnisse ab

SSSP (**S**ingle-**S**ource-**S**hortest-**P**ath) ⟶ **APSP** (**A**ll-**P**airs-**S**hortest-**P**ath)

APSP Strategien

Naive Strategie: n mal Bellman-Ford

Für jeden Knoten einmal

- Laufzeit: $\Theta(n \cdot nm)$

APSP Strategien

Naive Strategie: n mal Bellman-Ford

Für jeden Knoten einmal

- Laufzeit: $\Theta(n \cdot nm)$

Bessere Strategie: Nutze Zwischenergebnisse von Bellman-Ford

- Gleiche Strategie wie Bellman-Ford: Kanten relaxieren
- Benutze jeden Knoten als Zwischenstop und probiere alle Paare von (Start,Ziel) aus

APSP Strategien

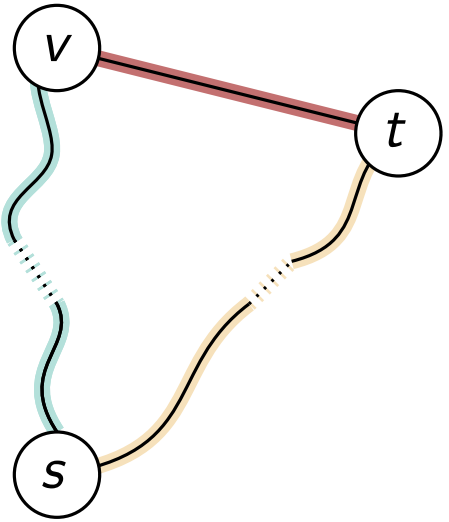
Naive Strategie: n mal Bellman-Ford

Für jeden Knoten einmal

- Laufzeit: $\Theta(n \cdot nm)$

Bessere Strategie: Nutze Zwischenergebnisse von Bellman-Ford

- Gleiche Strategie wie Bellman-Ford: Kanten relaxieren
- Benutze jeden Knoten als Zwischenstop und probiere alle Paare von (Start,Ziel) aus



vorläufige $s-v$ und $s-t$ Pfade gegeben

- Betrachte v als Zwischenstop

$$\Longrightarrow \quad d(s,t) = \min(d(s,t), d(s,v) + \text{len}(v,t))$$

  - " Ist es billiger über v zu t zu gehen? "
- Variiere s und t um alle Paare abzudecken

Floyd-Warshalls Algorithmus: Pseudocode

FloydWarshall(*Graph* G)

$D := n \times n$ *Matrix* initialized with ∞
for $(u, v) \in E$ **do** $D[u][v] := \text{len}(u, v)$
for $v \in V$ **do** $D[v][v] := 0$
for $i := 1, \ldots, n$ **do**
  for all pairs of nodes $(u, v) \in V \times V$ **do**
    $D[u][v] := \min(D[u][v], D[u][v_i] + D[v_i][v])$
return D

Floyd-Warshalls Algorithmus: Pseudocode

FloydWarshall(*Graph* G)

$D := n \times n$ *Matrix* initialized with ∞
for $(u, v) \in E$ **do** $D[u][v] := \text{len}(u, v)$
for $v \in V$ **do** $D[v][v] := 0$
for $i := 1, \ldots, n$ **do**
  for all pairs of nodes $(u, v) \in V \times V$ **do**
    $D[u][v] := \min(D[u][v], D[u][v_i] + D[v_i][v])$
return D

- In $D[u][v]$ speichern wir uns die aktuell beste Distanz von u nach v

Floyd-Warshalls Algorithmus: Pseudocode

FloydWarshall(*Graph* G)

$D := n \times n$ *Matrix* initialized with ∞

for $(u, v) \in E$ **do** $D[u][v] := \text{len}(u, v)$

for $v \in V$ **do** $D[v][v] := 0$

for $i := 1, \ldots, n$ **do**

  for all pairs of nodes $(u, v) \in V \times V$ **do**

    $D[u][v] := \min(D[u][v], D[u][v_i] + D[v_i][v])$

return D

- Populiere die Distanzmatrix mit bekannten oberen Schranken
- Kanten (u, v) sind ein erster Weg von u nach v
- Knoten haben Distanz 0 zu sich selber

Floyd-Warshalls Algorithmus: Pseudocode

FloydWarshall(*Graph* G)

$D := n \times n$ *Matrix* initialized with ∞
for $(u, v) \in E$ **do** $D[u][v] := \text{len}(u, v)$
for $v \in V$ **do** $D[v][v] := 0$
for $i := 1, \ldots, n$ **do**
  for all pairs of nodes $(u, v) \in V \times V$ **do**
    $D[u][v] := \min(D[u][v], D[u][v_i] + D[v_i][v])$
return D

- Jeder Knoten wird einmal als Zwischenstopp benutzt

Floyd-Warshalls Algorithmus: Pseudocode

FloydWarshall(*Graph* G)

$D := n \times n$ *Matrix* initialized with ∞
for $(u, v) \in E$ **do** $D[u][v] := \text{len}(u, v)$
for $v \in V$ **do** $D[v][v] := 0$
for $i := 1, \ldots, n$ **do**
  for all pairs of nodes $(u, v) \in V \times V$ **do**
    $D[u][v] := \min(D[u][v], D[u][v_i] + D[v_i][v])$
return D

- Für jedes Paar an Start- und Zielknoten wird geprüft, ob es schneller ist, über den Zwischenknoten zu gehen

Floyd-Warshalls Algorithmus: Laufzeit

FloydWarshall(*Graph* G)

```
D := n × n Matrix initialized with ∞
for (u, v) ∈ E do D[u][v] := len(u, v)
for v ∈ V do D[v][v] := 0
for i := 1, . . . , n do
    for all pairs of nodes (u, v) ∈ V × V do
        D[u][v] := min(D[u][v], D[u][v_i] + D[v_i][v])
return D
```

Welche Laufzeit hat dieser Algorithmus?

Floyd-Warshalls Algorithmus: Laufzeit

FloydWarshall(*Graph* G)

```
D := n × n Matrix initialized with ∞
for (u, v) ∈ E do D[u][v] := len(u, v)
for v ∈ V do D[v][v] := 0
for i := 1, ..., n do
    for all pairs of nodes (u, v) ∈ V × V do
        D[u][v] := min(D[u][v], D[u][v_i] + D[v_i][v])
return D
```

Welche Laufzeit hat dieser Algorithmus?

- Matrix mit n^2 Einträgen erstellen in $\Theta(n^2)$
- Jede Kante eintragen in $\Theta(m)$
- Distanz von Knoten zu sich selber in $\Theta(n)$ eintragen

Floyd-Warshalls Algorithmus: Laufzeit

FloydWarshall(*Graph* G)

$D := n \times n$ *Matrix* initialized with ∞
for $(u, v) \in E$ **do** $D[u][v] := \text{len}(u, v)$
for $v \in V$ **do** $D[v][v] := 0$
for $i := 1, \ldots, n$ **do**
  for all pairs of nodes $(u, v) \in V \times V$ **do**
    $D[u][v] := \min(D[u][v], D[u][v_i] + D[v_i][v])$
return D

Welche Laufzeit hat dieser Algorithmus?

- Matrix mit n^2 Einträgen erstellen in $\Theta(n^2)$
- Jede Kante eintragen in $\Theta(m)$
- Distanz von Knoten zu sich selber in $\Theta(n)$ eintragen
- Jeder der n Knoten wird einmal als Zwischenstopp benutzt
- Pro Zwischenstopp wird jedes der n^2 Knotenpaare betrachtet

Floyd-Warshalls Algorithmus: Laufzeit

FloydWarshall(*Graph* G)

$D := n \times n$ *Matrix* initialized with ∞
for $(u, v) \in E$ **do** $D[u][v] := \text{len}(u, v)$
for $v \in V$ **do** $D[v][v] := 0$
for $i := 1, \ldots, n$ **do**
  for all pairs of nodes $(u, v) \in V \times V$ **do**
    $D[u][v] := \min(D[u][v], D[u][v_i] + D[v_i][v])$
return D

Insgesamt $\Theta(n^2 + m + n + n \cdot n^2)$

Welche Laufzeit hat dieser Algorithmus?

- Matrix mit n^2 Einträgen erstellen in $\Theta(n^2)$
- Jede Kante eintragen in $\Theta(m)$
- Distanz von Knoten zu sich selber in $\Theta(n)$ eintragen
- Jeder der n Knoten wird einmal als Zwischenstopp benutzt
- Pro Zwischenstopp wird jedes der n^2 Knotenpaare betrachtet

Floyd-Warshalls Algorithmus: Laufzeit

FloydWarshall(*Graph* G)

$D := n \times n$ *Matrix* initialized with ∞
for $(u, v) \in E$ **do** $D[u][v] := \text{len}(u, v)$
for $v \in V$ **do** $D[v][v] := 0$
for $i := 1, \ldots, n$ **do**
  for all pairs of nodes $(u, v) \in V \times V$ **do**
    $D[u][v] := \min(D[u][v], D[u][v_i] + D[v_i][v])$
return D

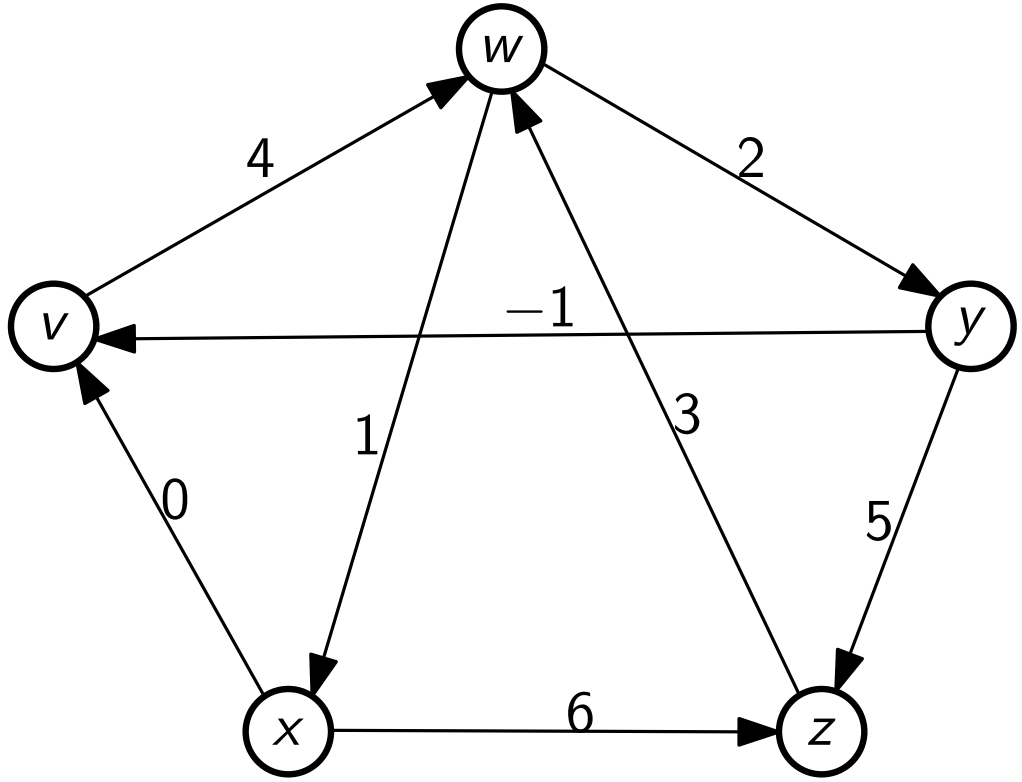
Insgesamt $\Theta(n^2 + m + n + n \cdot n^2) = \Theta(n^3)$

Floyd-Warshalls Algorithmus hat eine Laufzeit von $\Theta(n^3)$

Welche Laufzeit hat dieser Algorithmus?

- Matrix mit n^2 Einträgen erstellen in $\Theta(n^2)$
- Jede Kante eintragen in $\Theta(m)$
- Distanz von Knoten zu sich selber in $\Theta(n)$ eintragen
- Jeder der n Knoten wird einmal als Zwischenstopp benutzt
- Pro Zwischenstopp wird jedes der n^2 Knotenpaare betrachtet

Floyd Warshalls Algorithmus: Beispiel

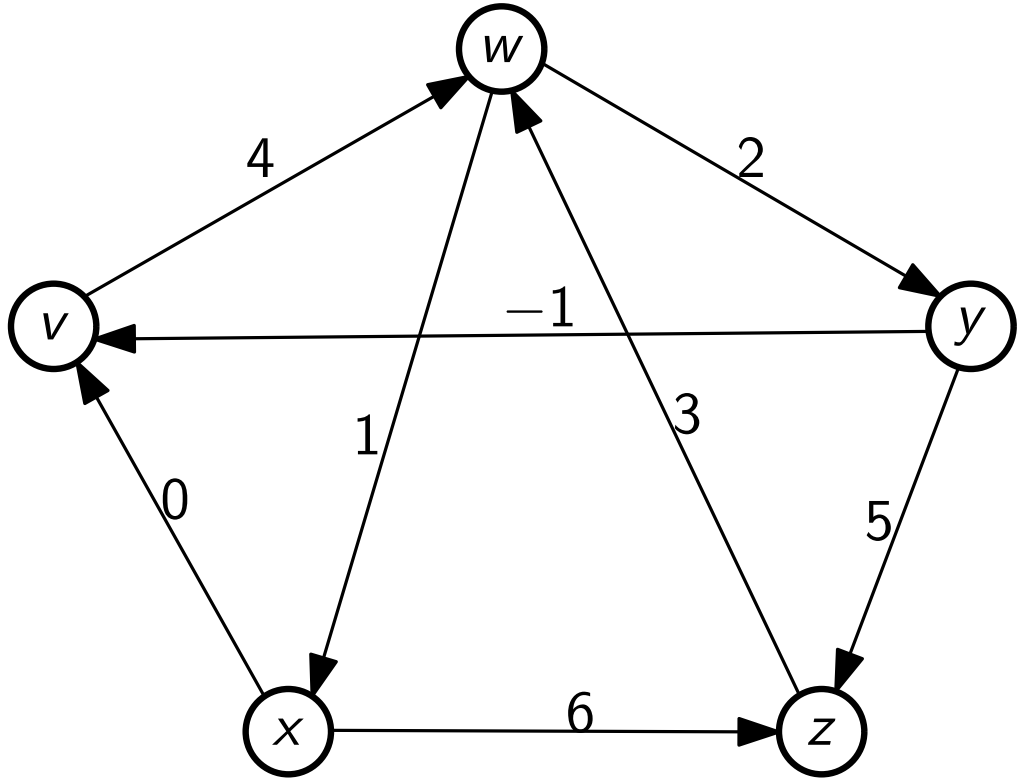


Distanzen

	v	w	x	y	z
v	∞	∞	∞	∞	∞
w	∞	∞	∞	∞	∞
x	∞	∞	∞	∞	∞
y	∞	∞	∞	∞	∞
z	∞	∞	∞	∞	∞

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel

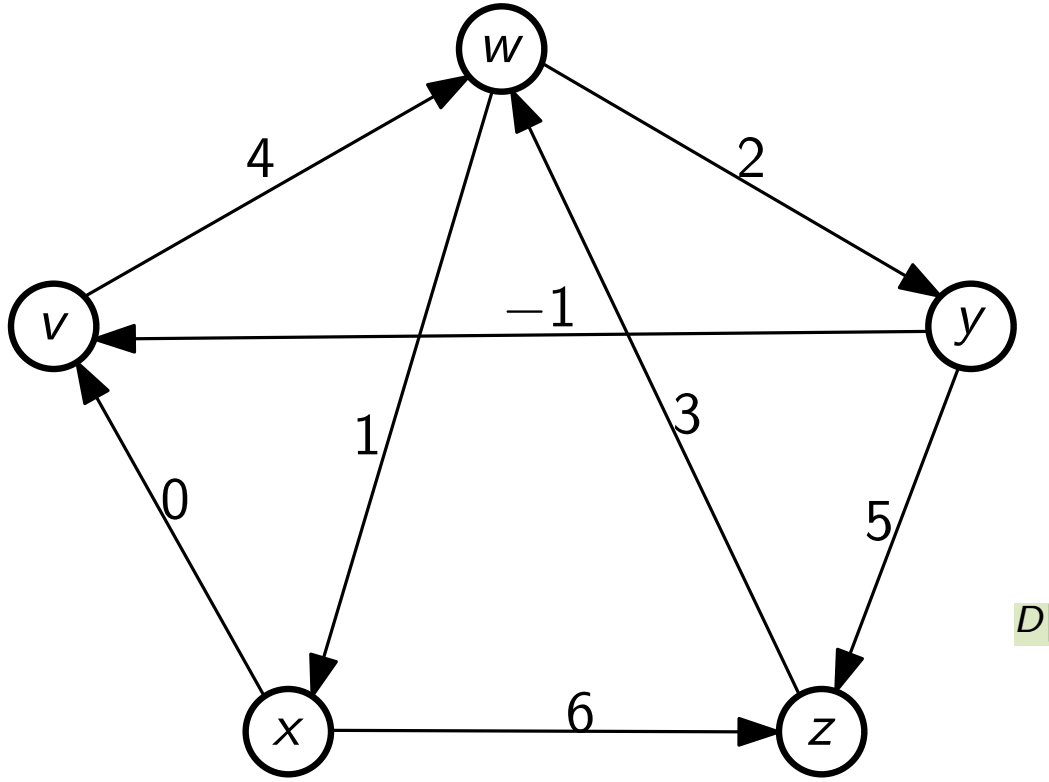


Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



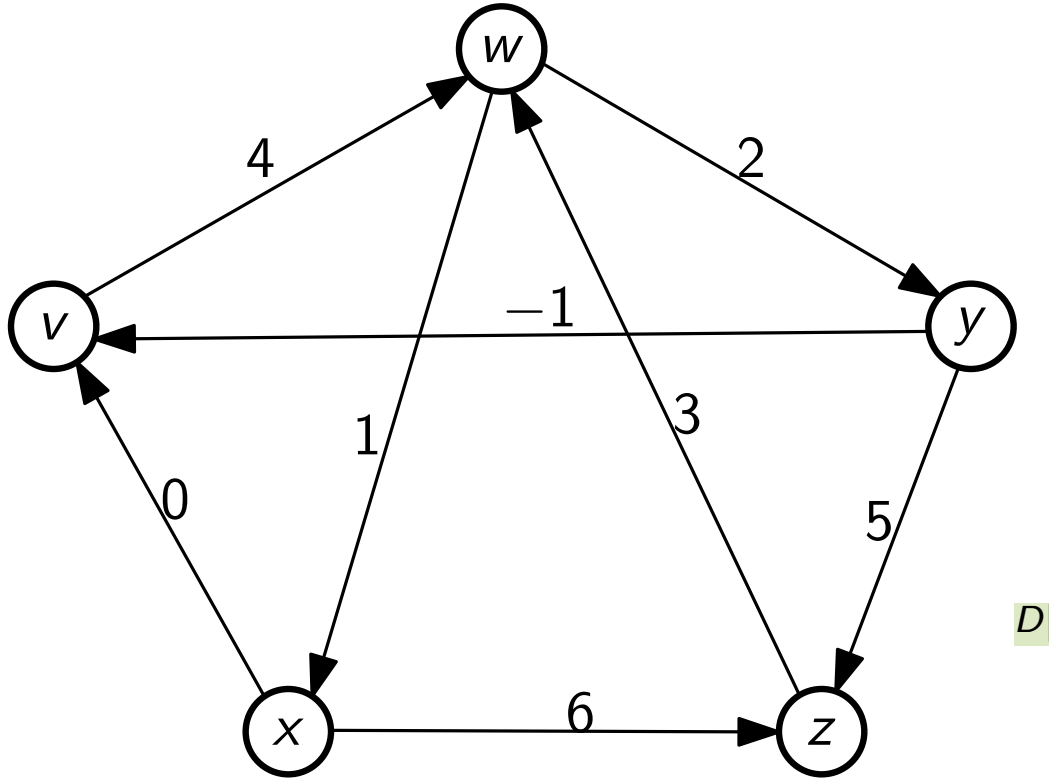
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	−1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



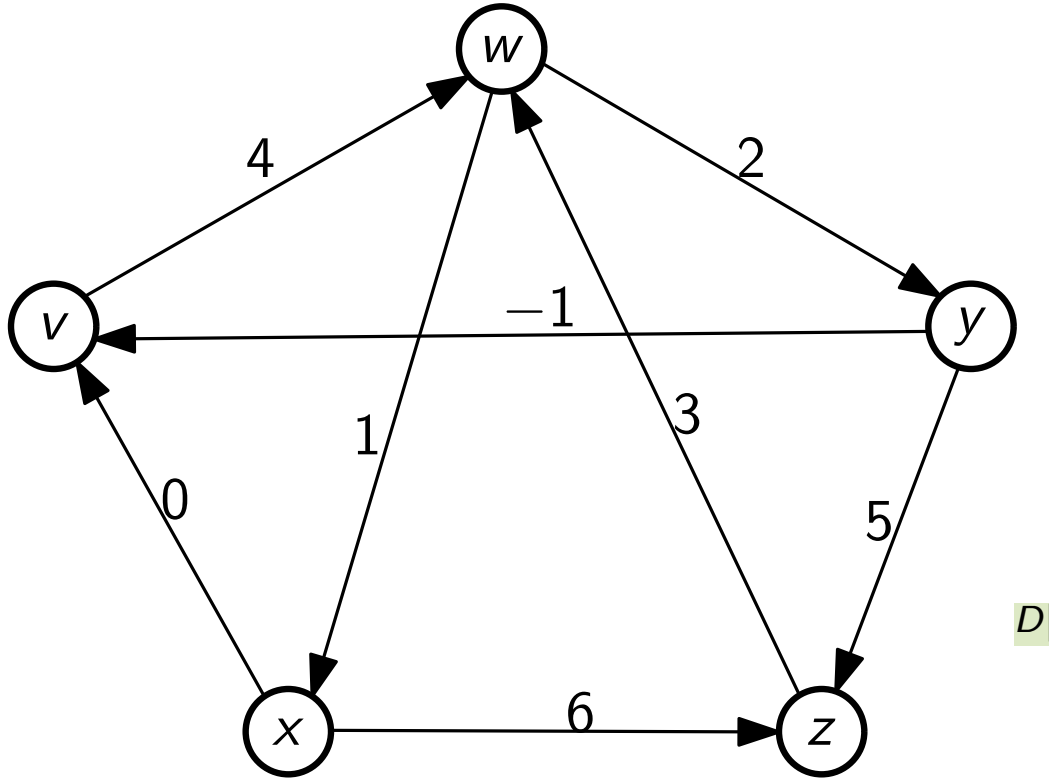
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	−1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



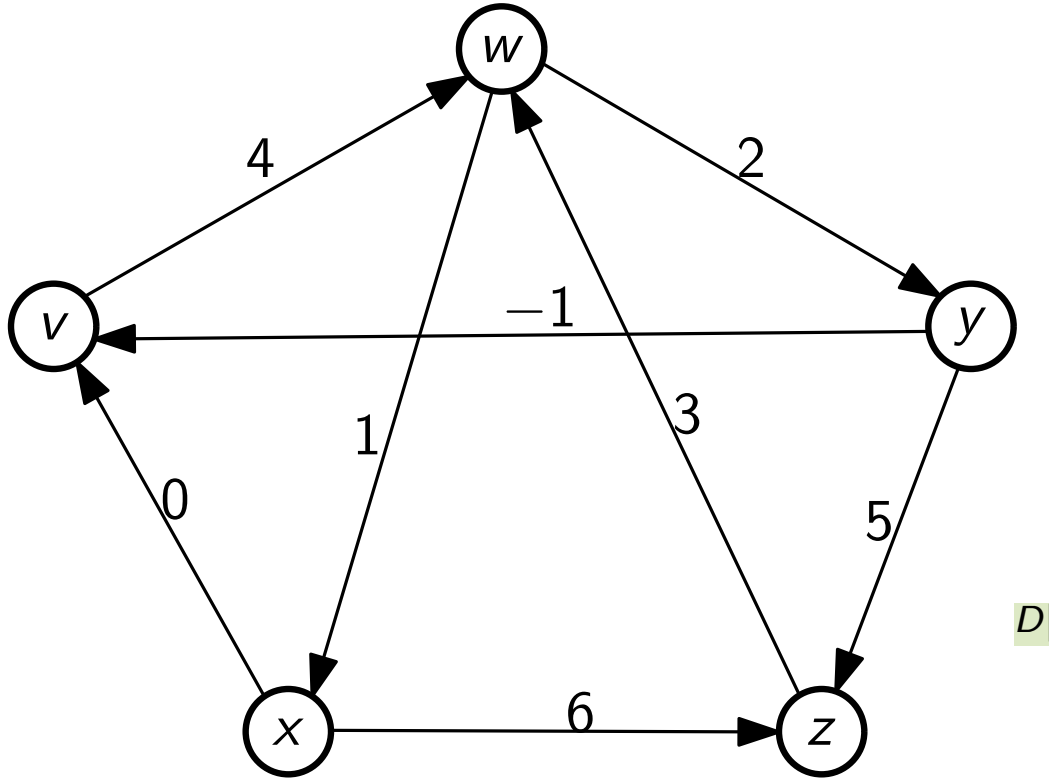
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



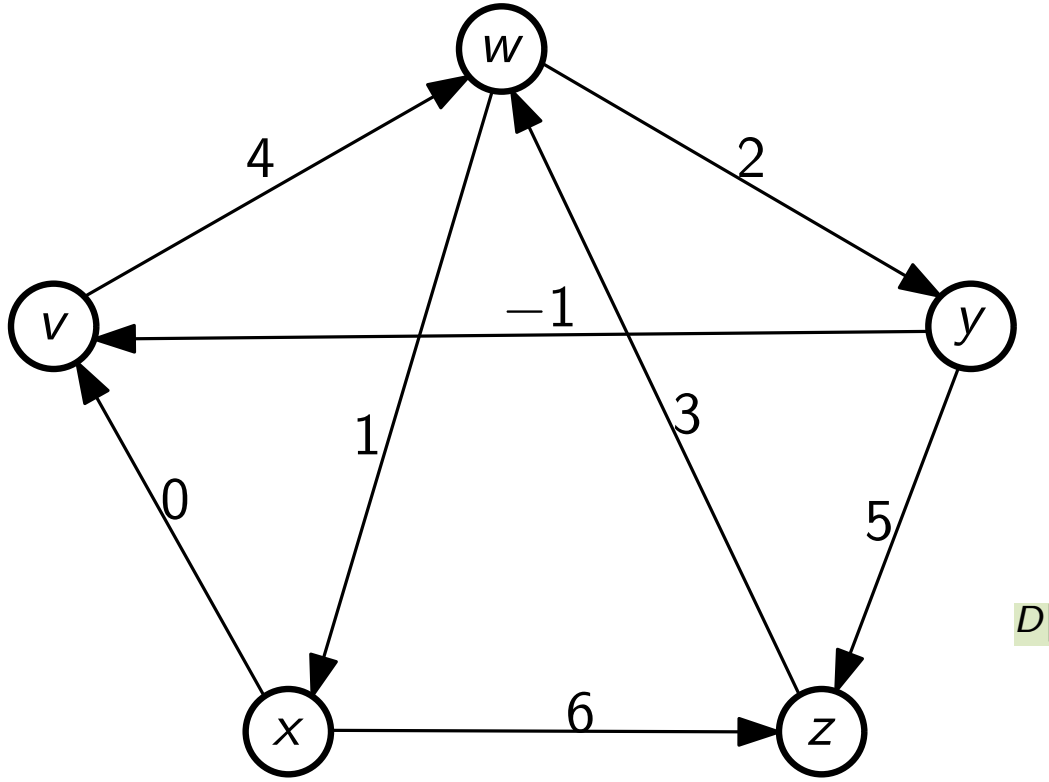
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



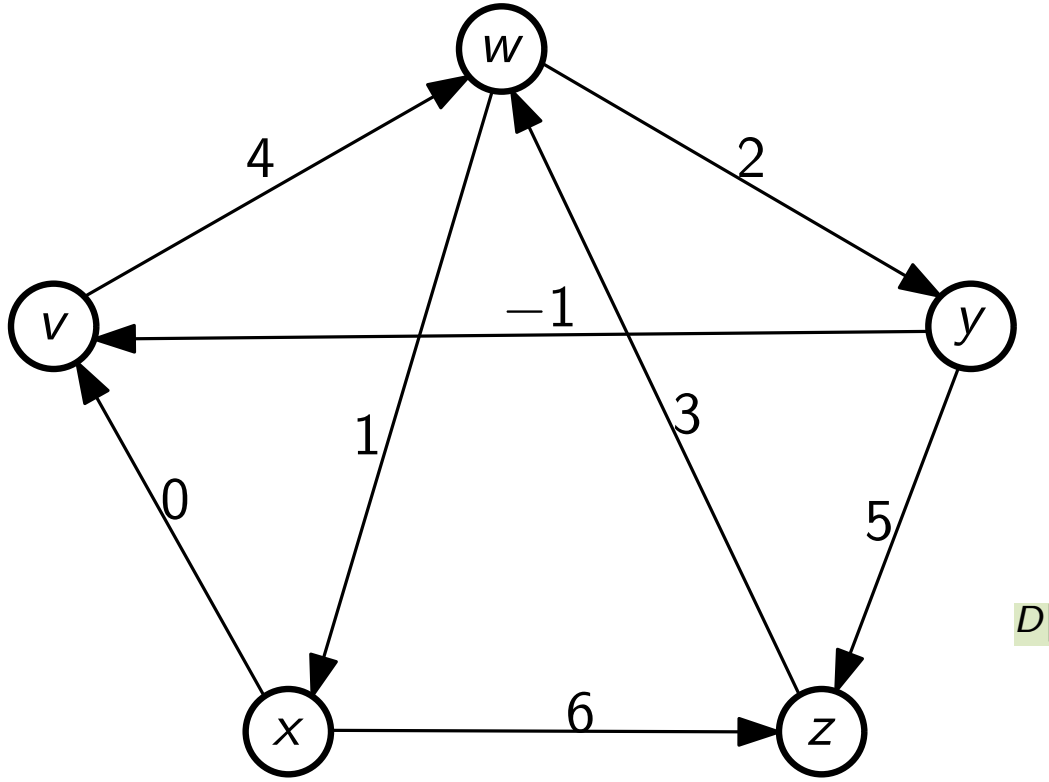
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



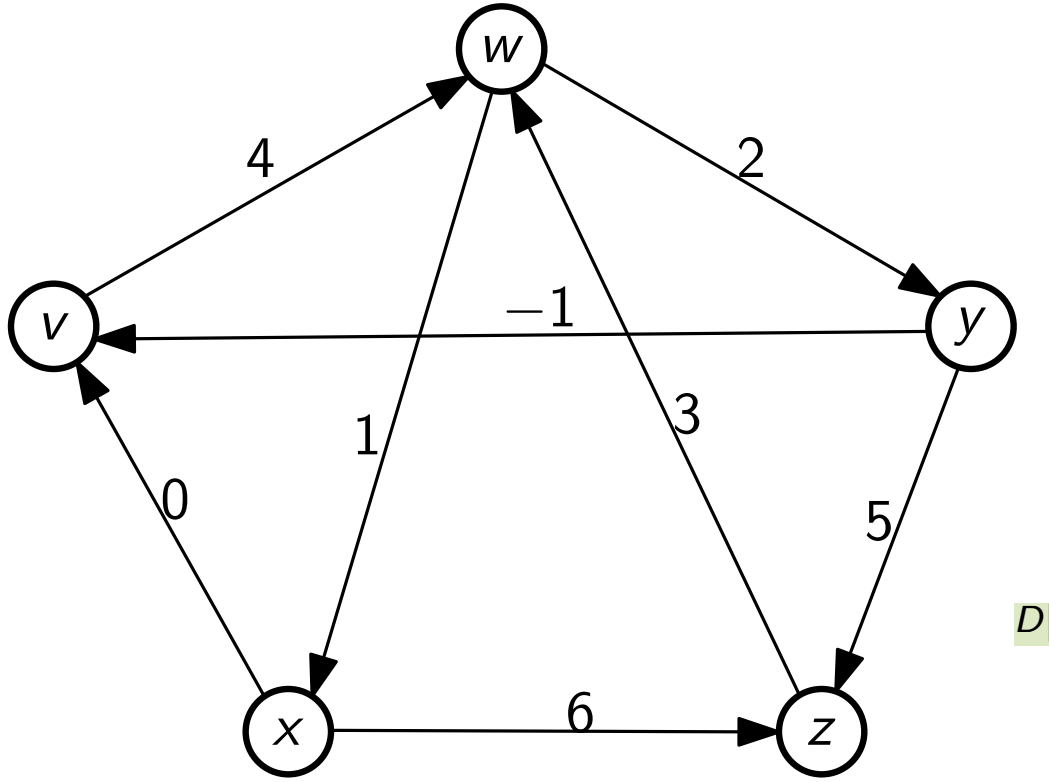
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



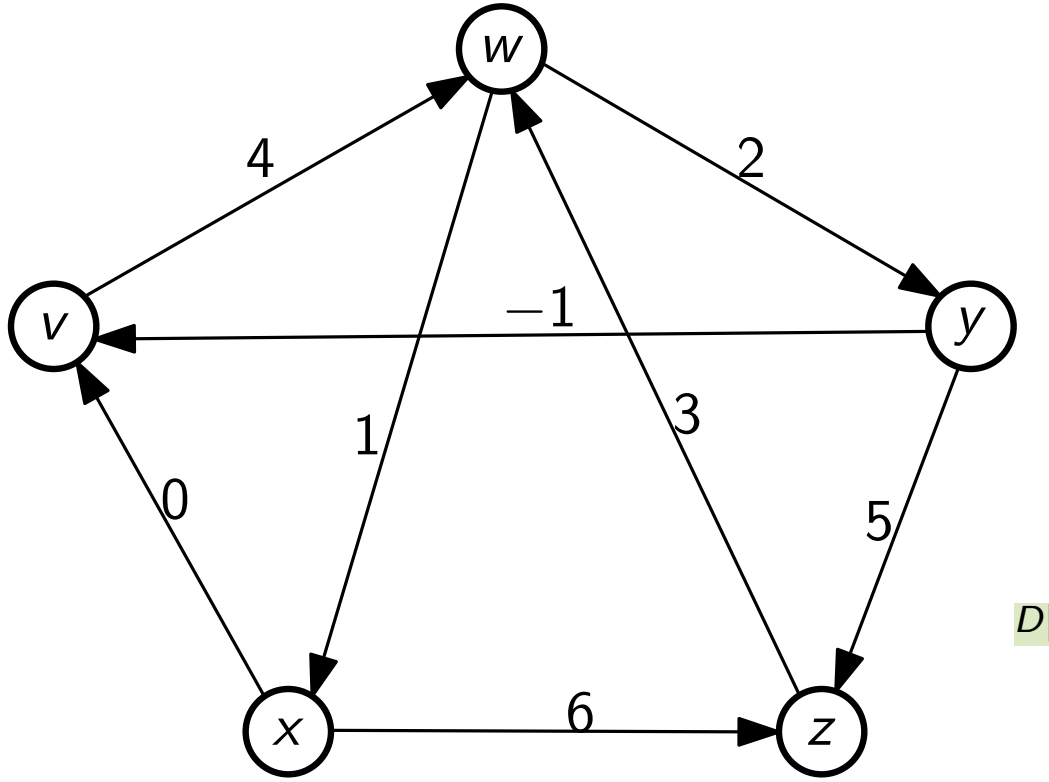
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c]+D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



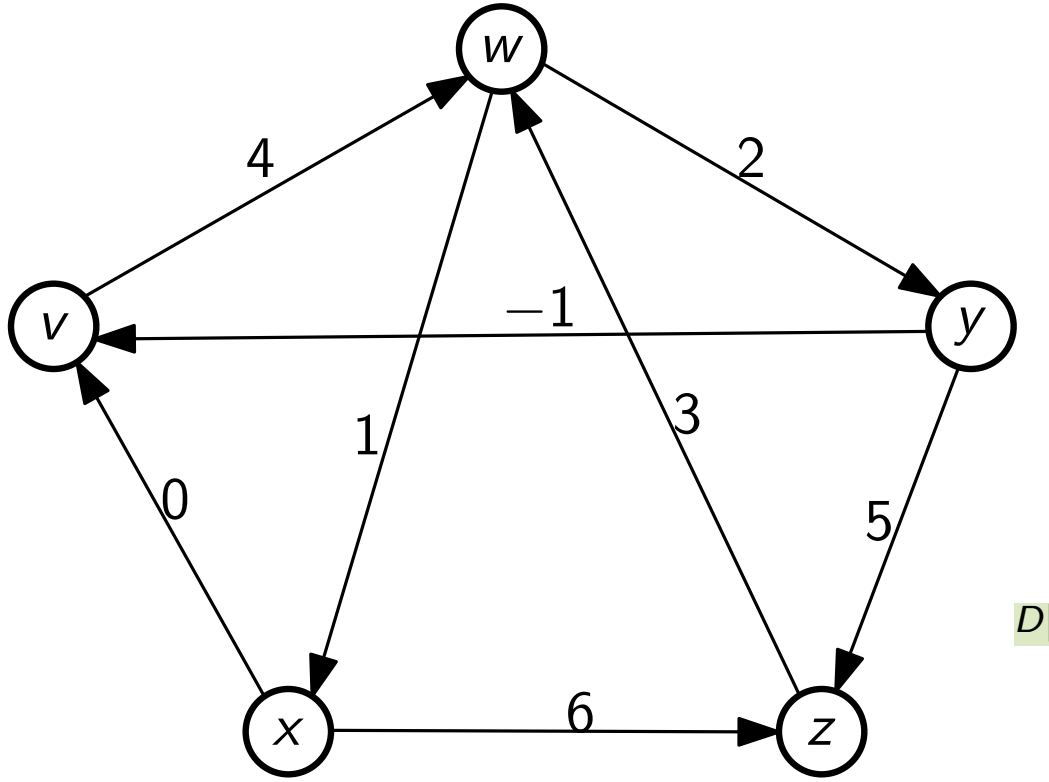
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



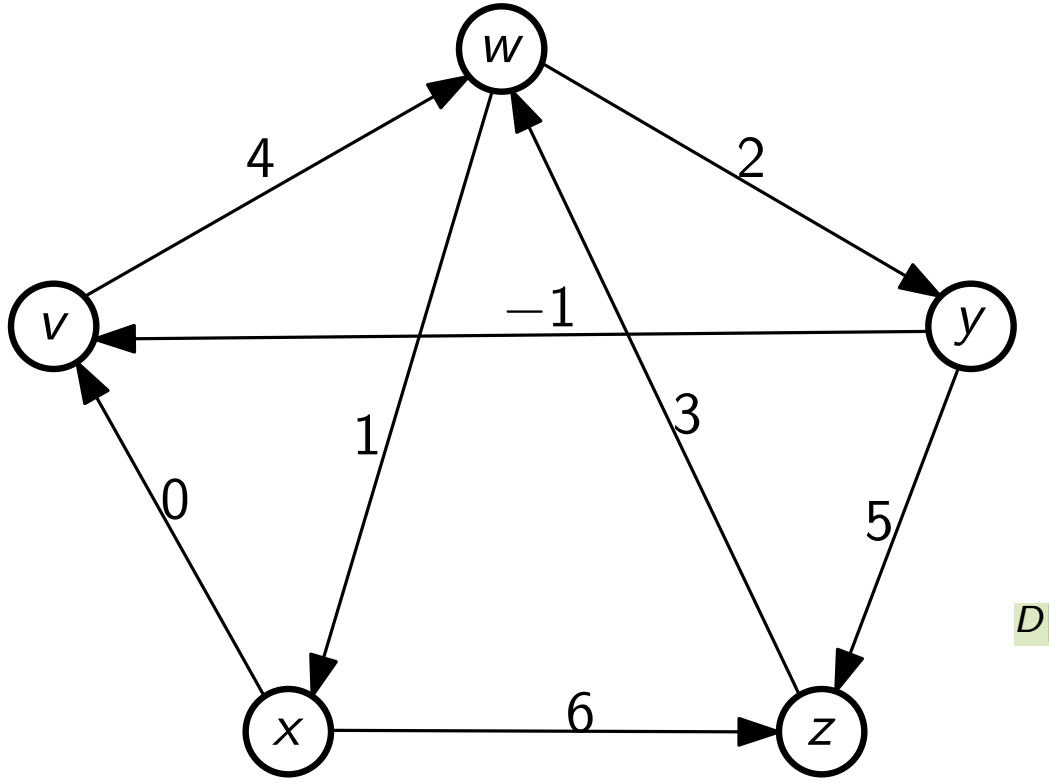
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



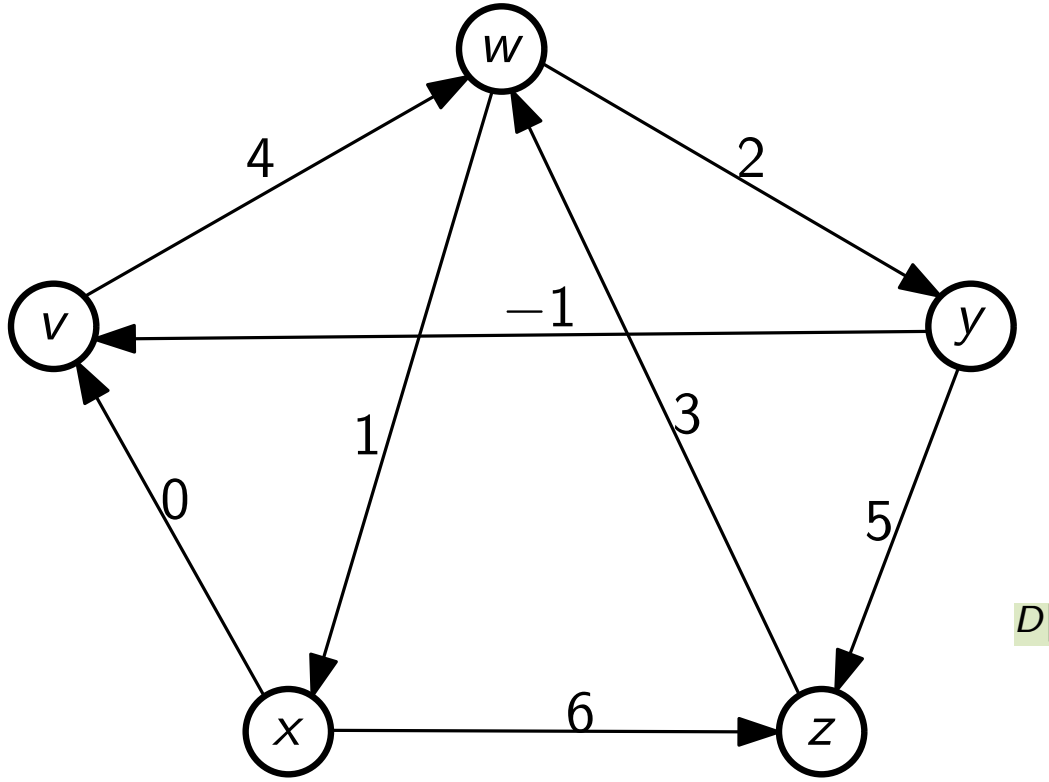
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



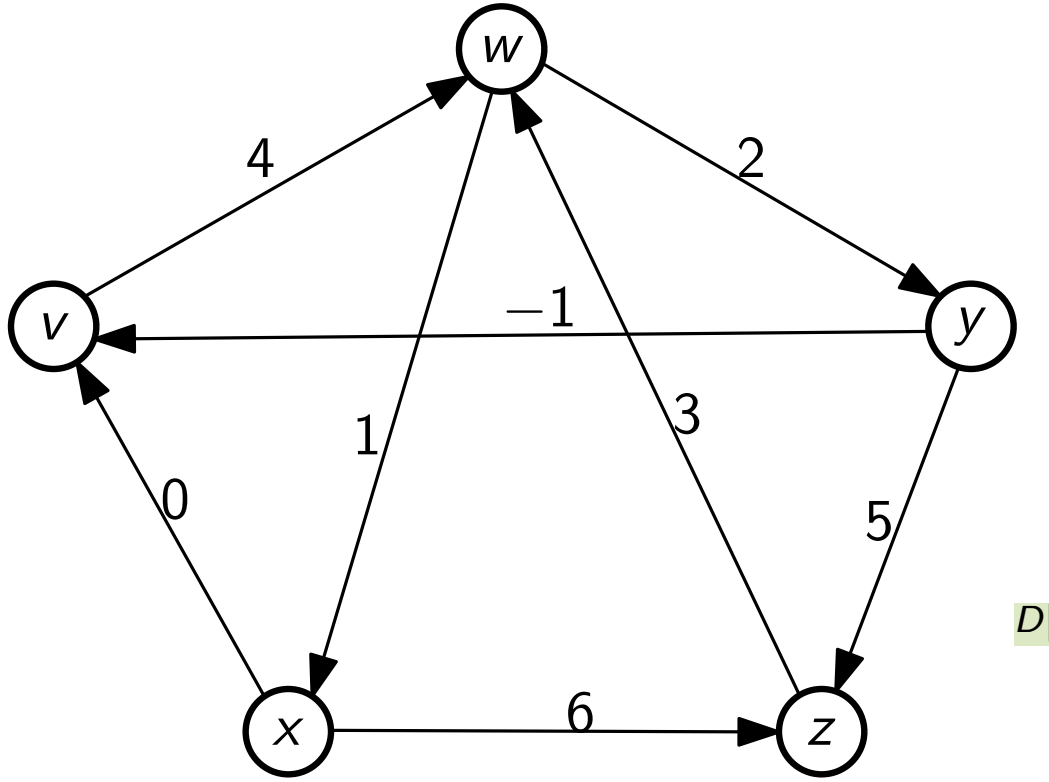
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	∞	0	∞	6
y	−1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



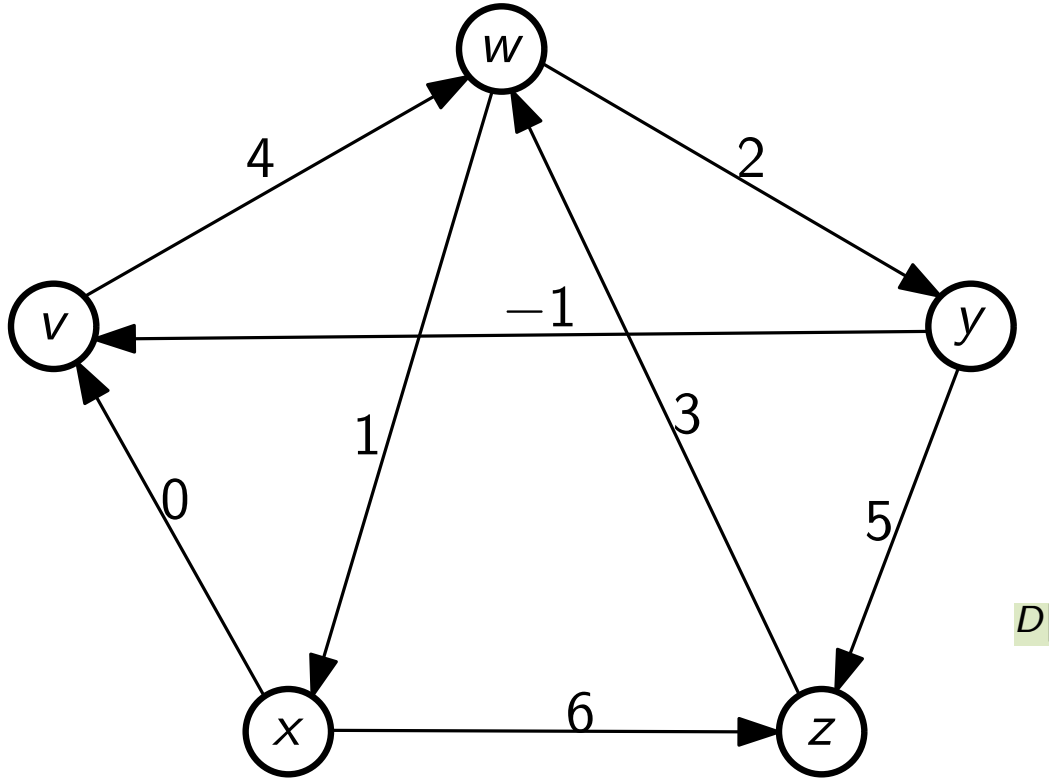
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	−1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



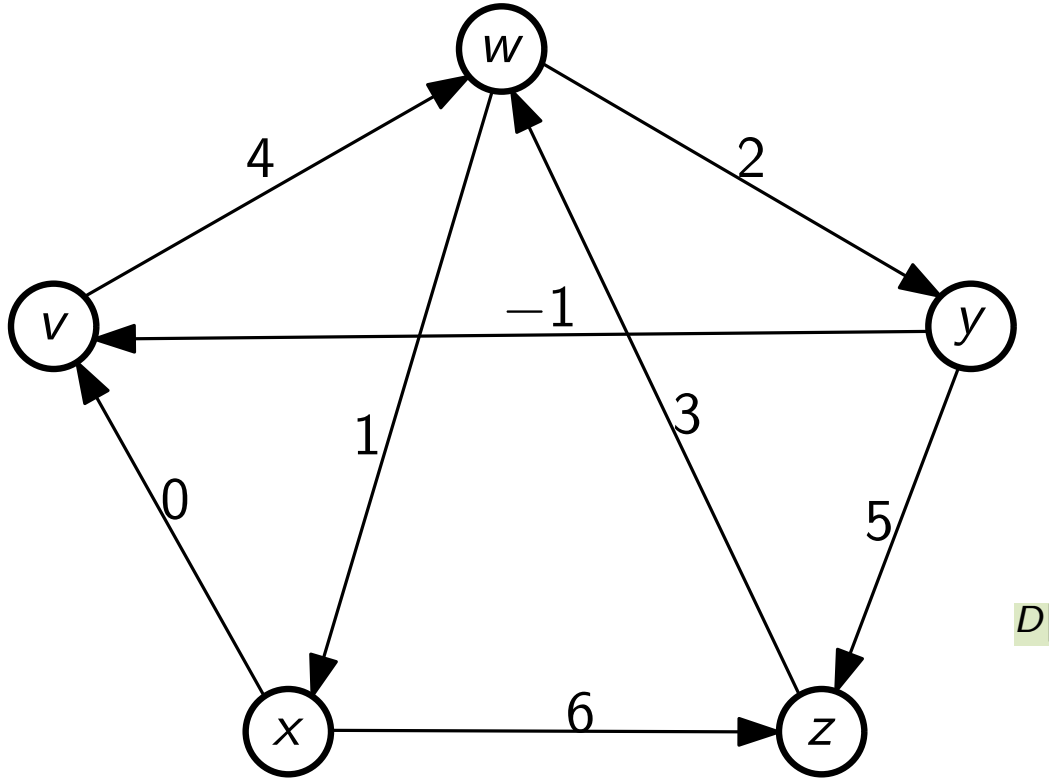
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



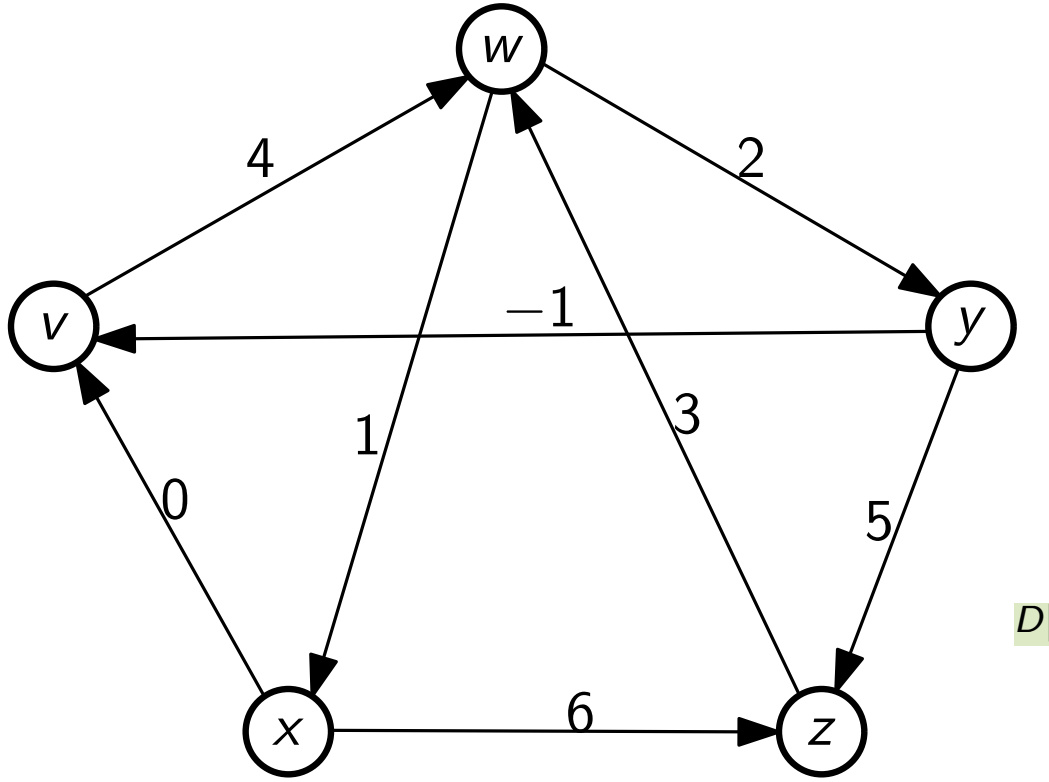
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



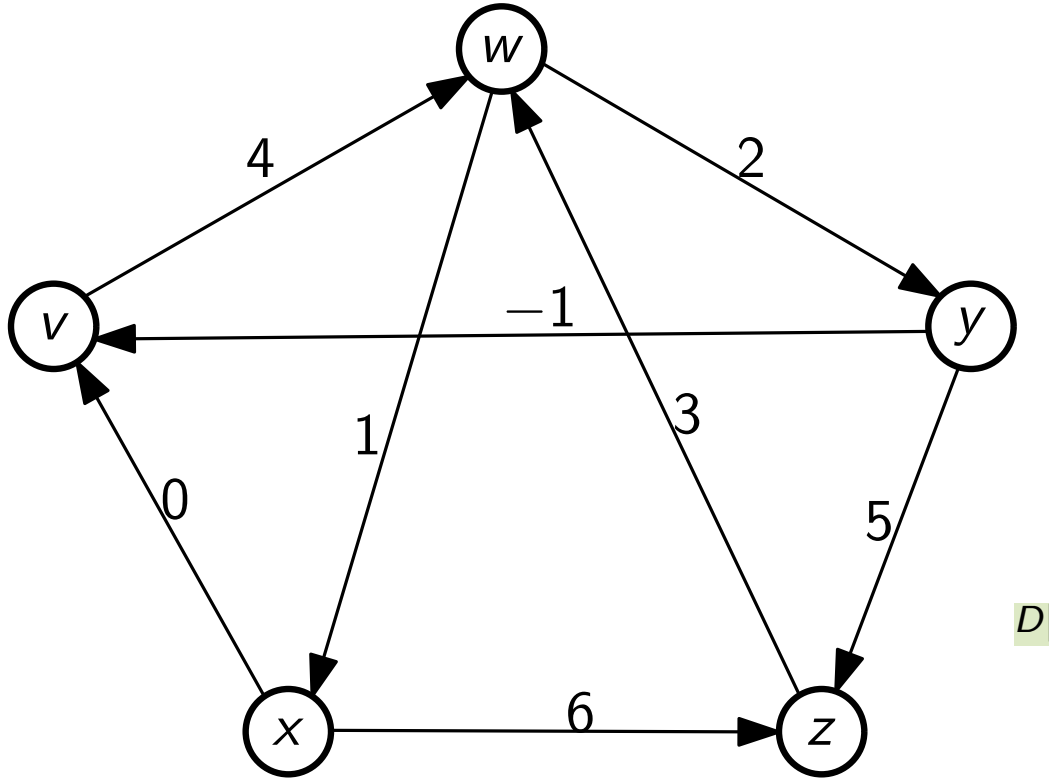
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



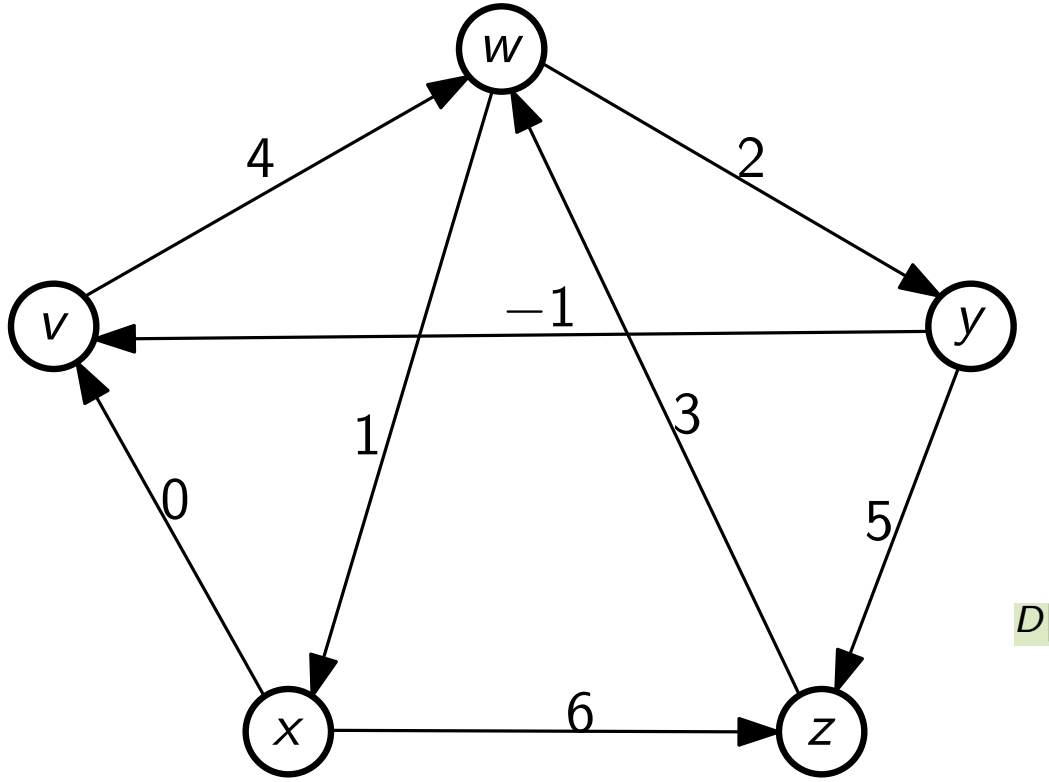
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	−1	∞	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



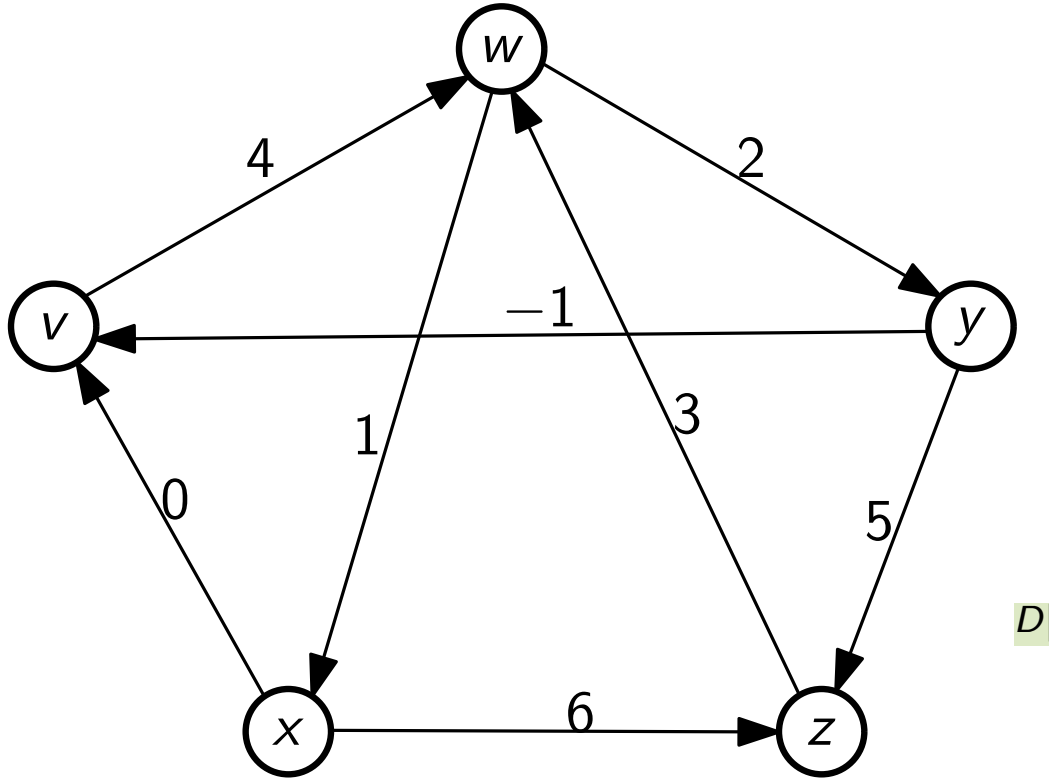
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



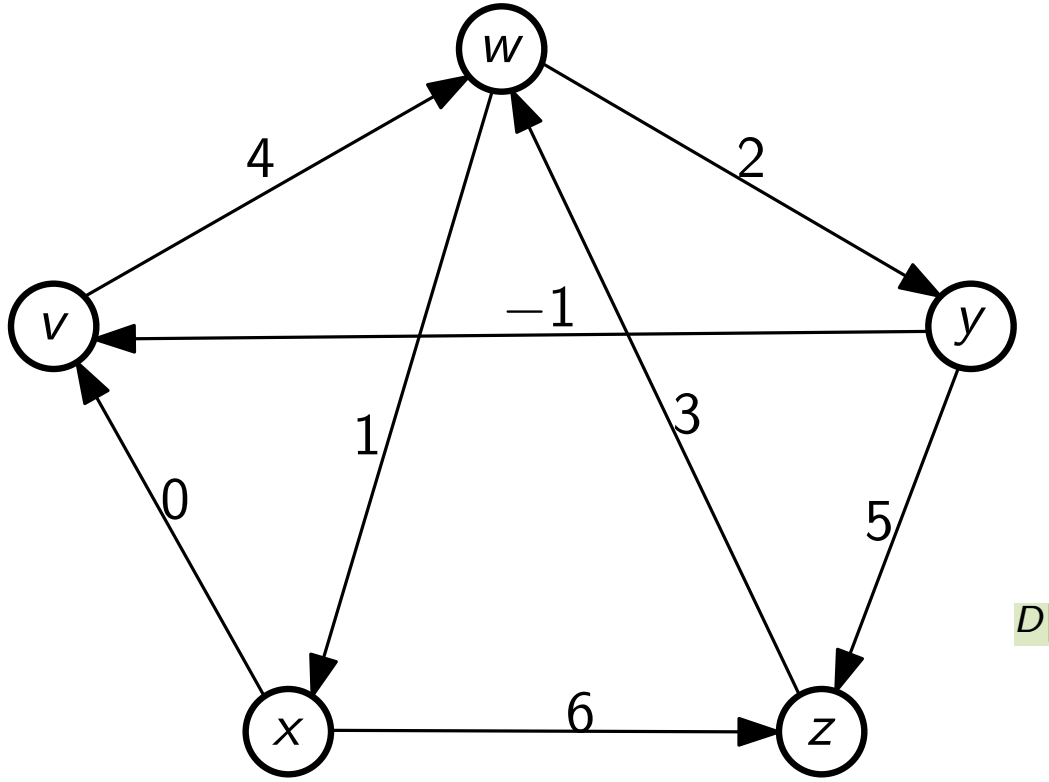
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



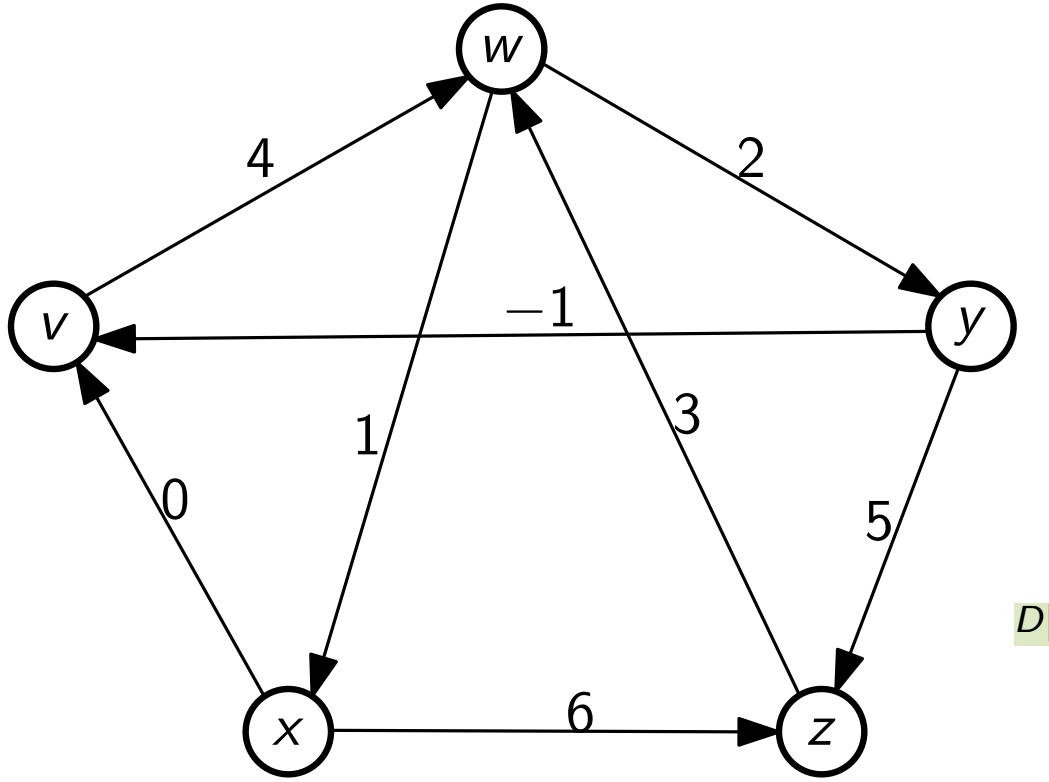
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



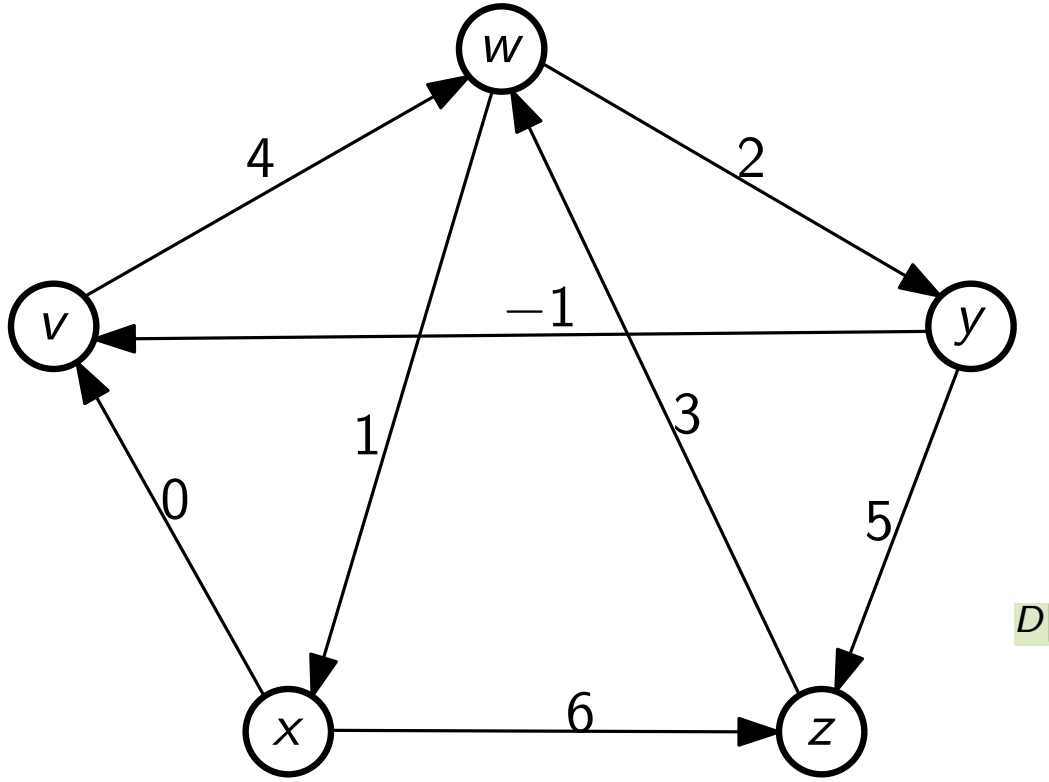
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



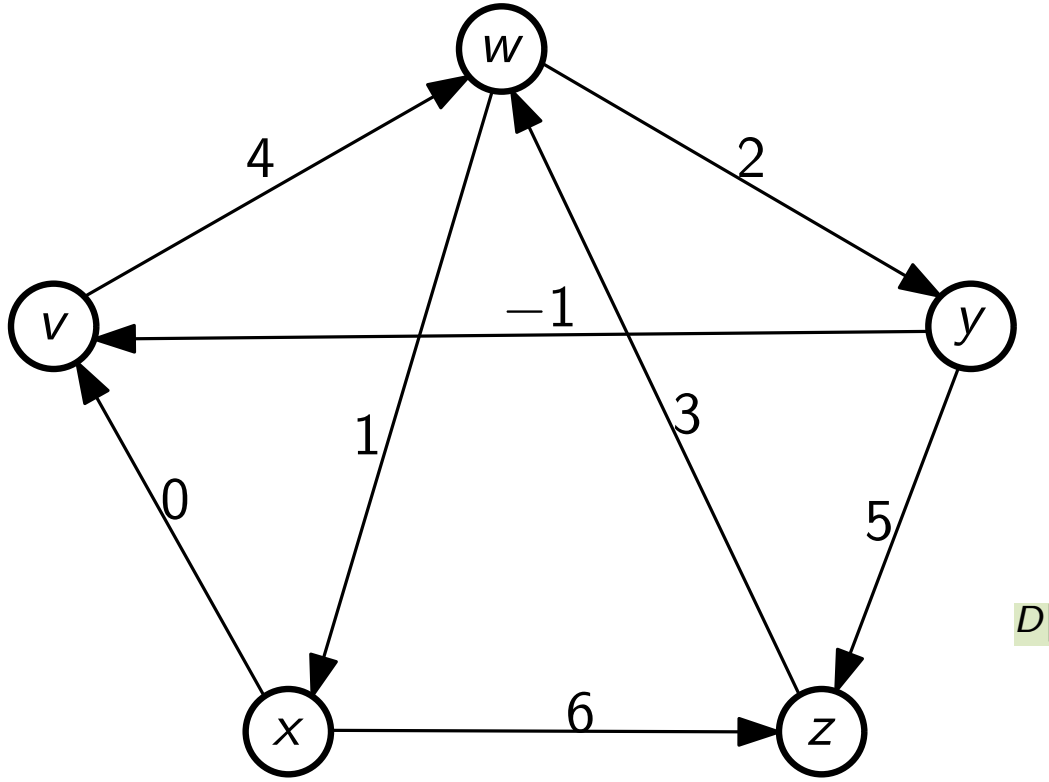
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c]+D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



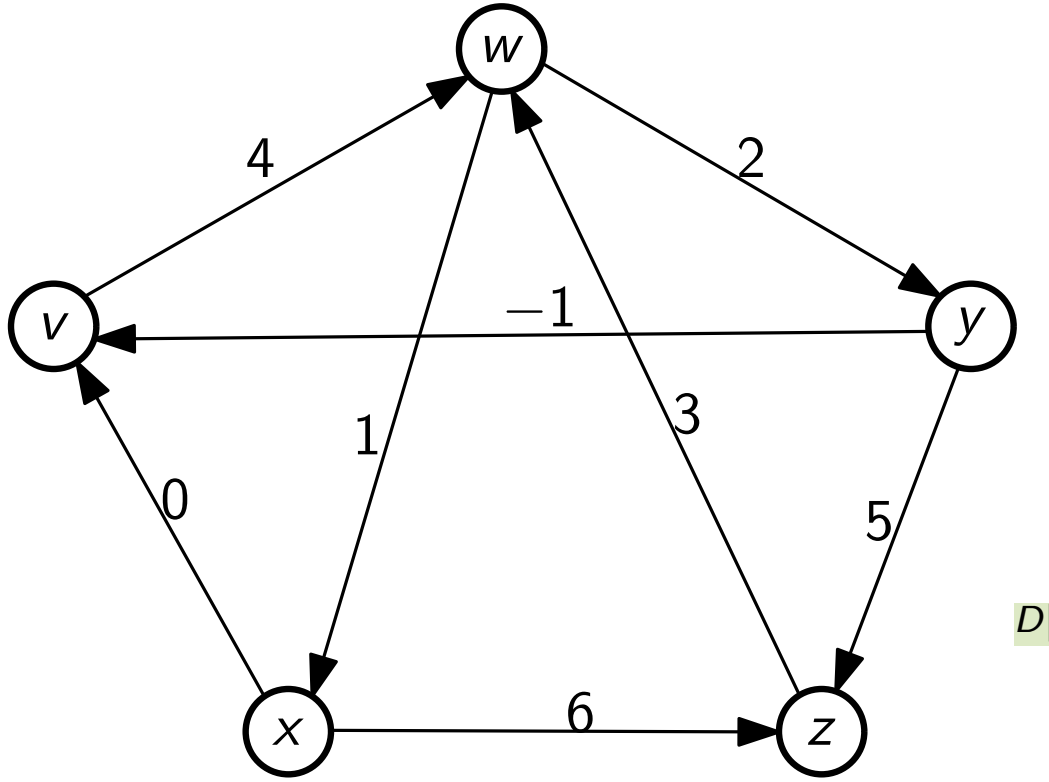
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



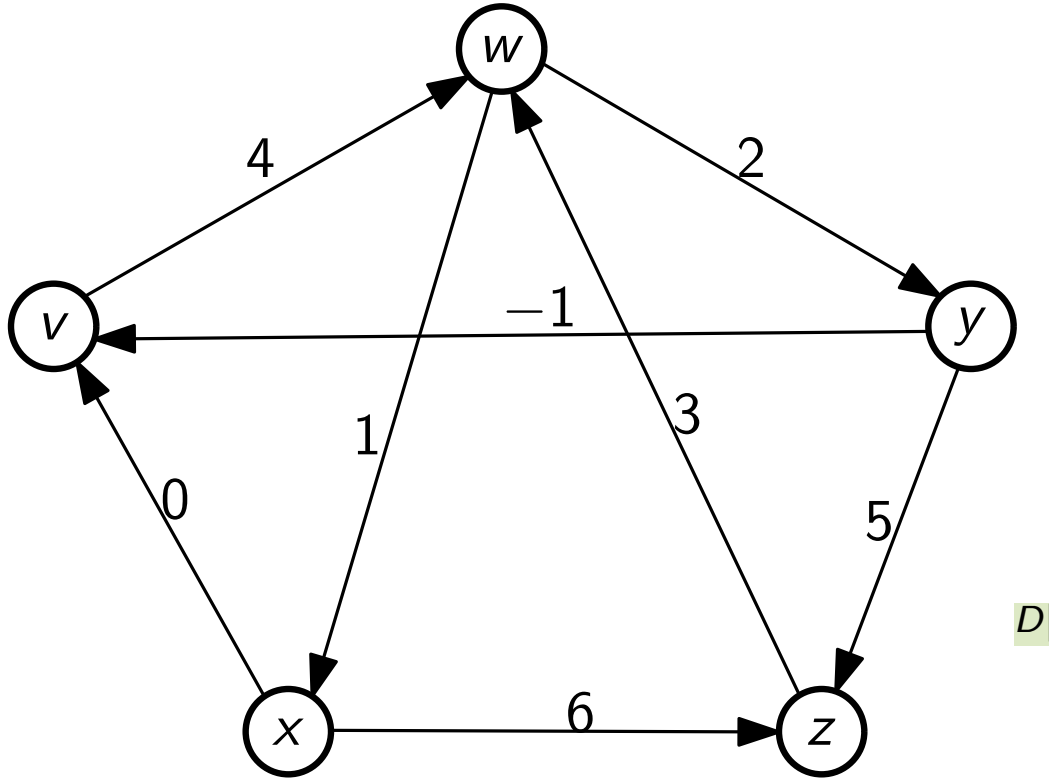
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



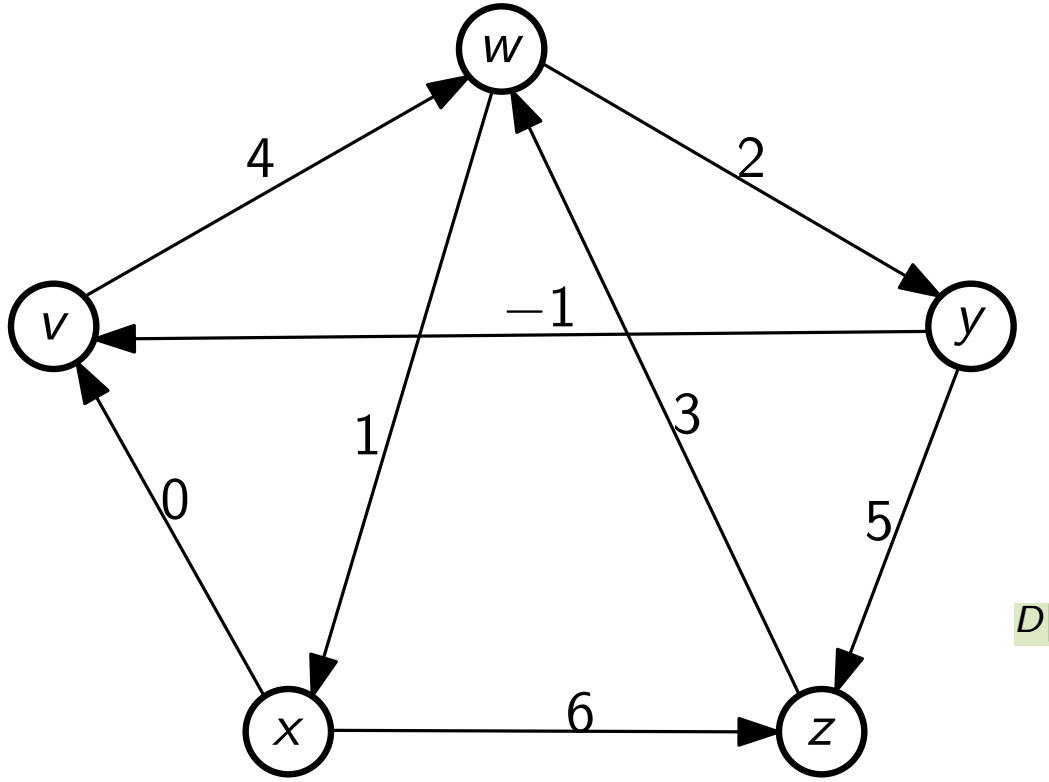
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



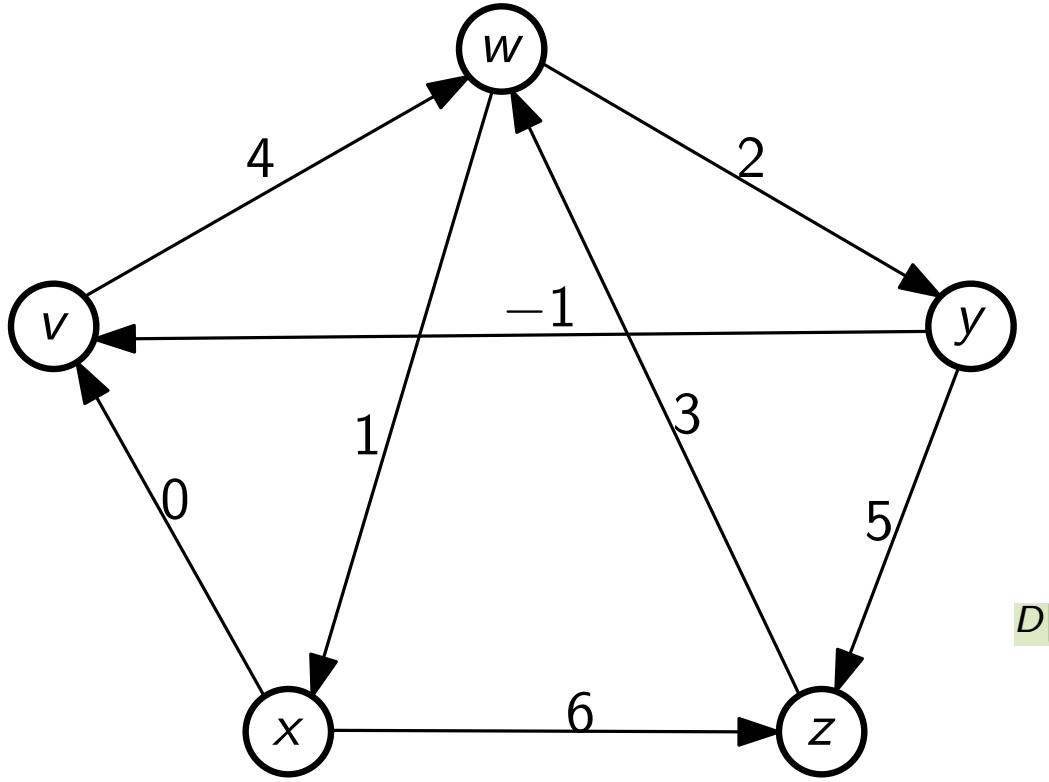
Distanzen

	v	w	x	y	z
v	0	4	∞	∞	∞
w	∞	0	1	2	∞
x	0	4	0	∞	6
y	-1	3	∞	0	5
z	∞	3	∞	∞	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



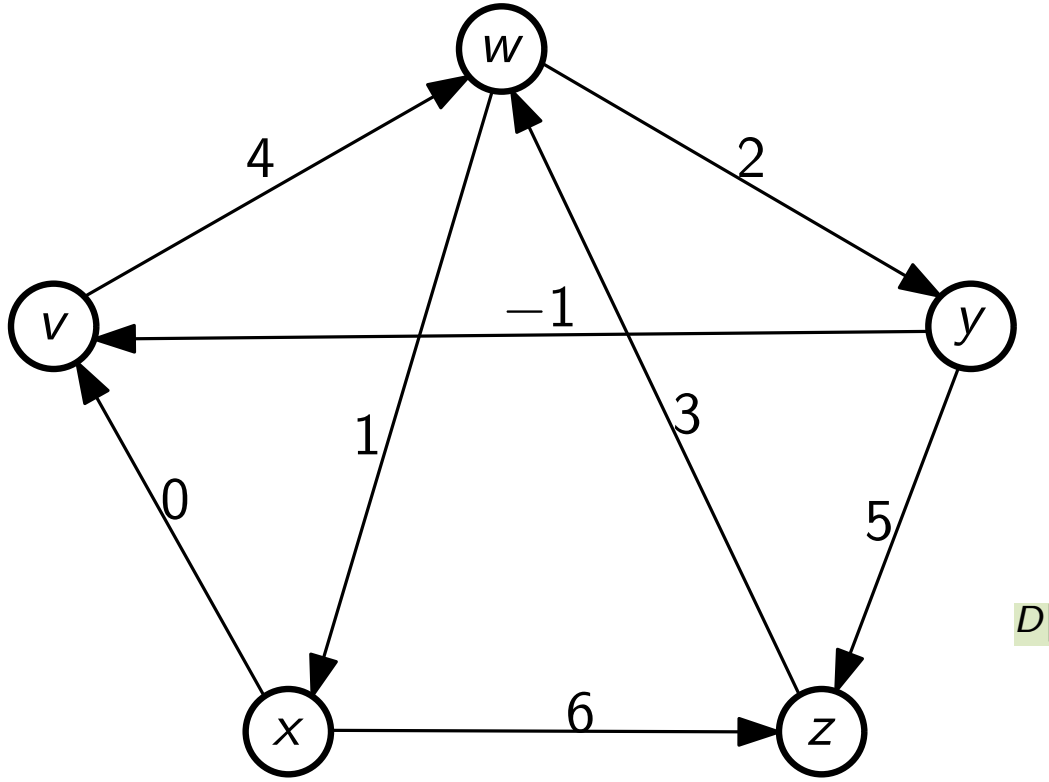
Distanzen

	v	w	x	y	z
v	0	4	5	6	∞
w	∞	0	1	2	∞
x	0	4	0	6	6
y	−1	3	4	0	5
z	∞	3	4	5	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



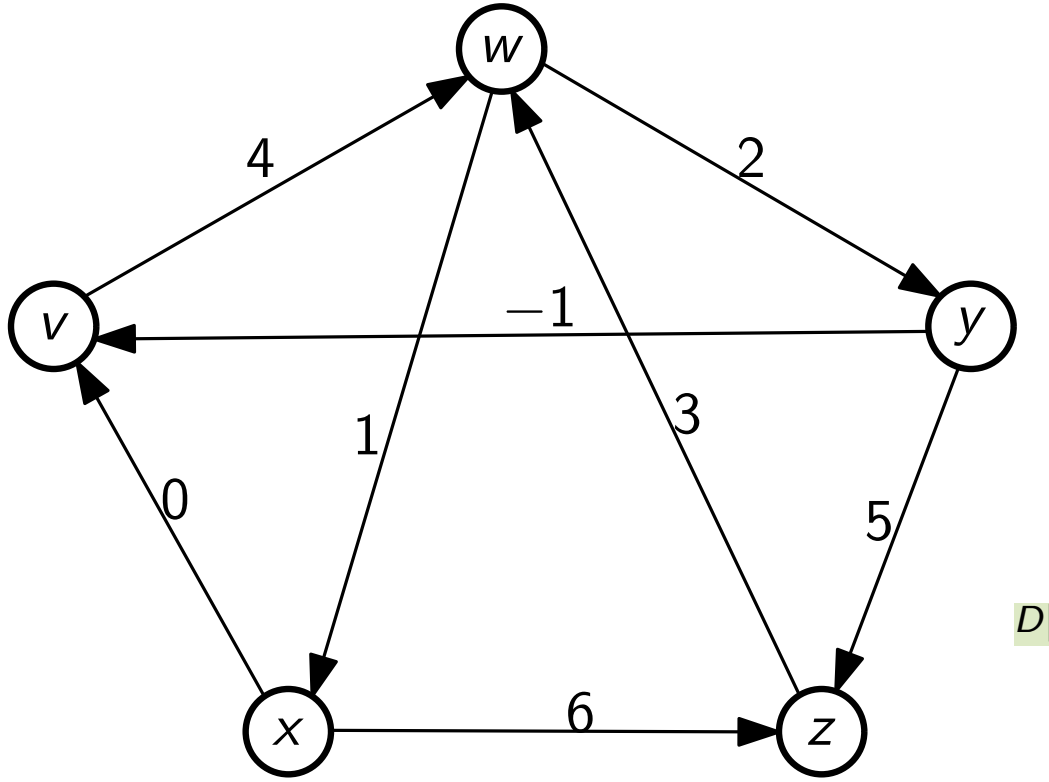
Distanzen

	v	w	x	y	z
v	0	4	5	6	11
w	1	0	1	2	7
x	0	4	0	6	6
y	−1	3	4	0	5
z	4	3	4	5	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



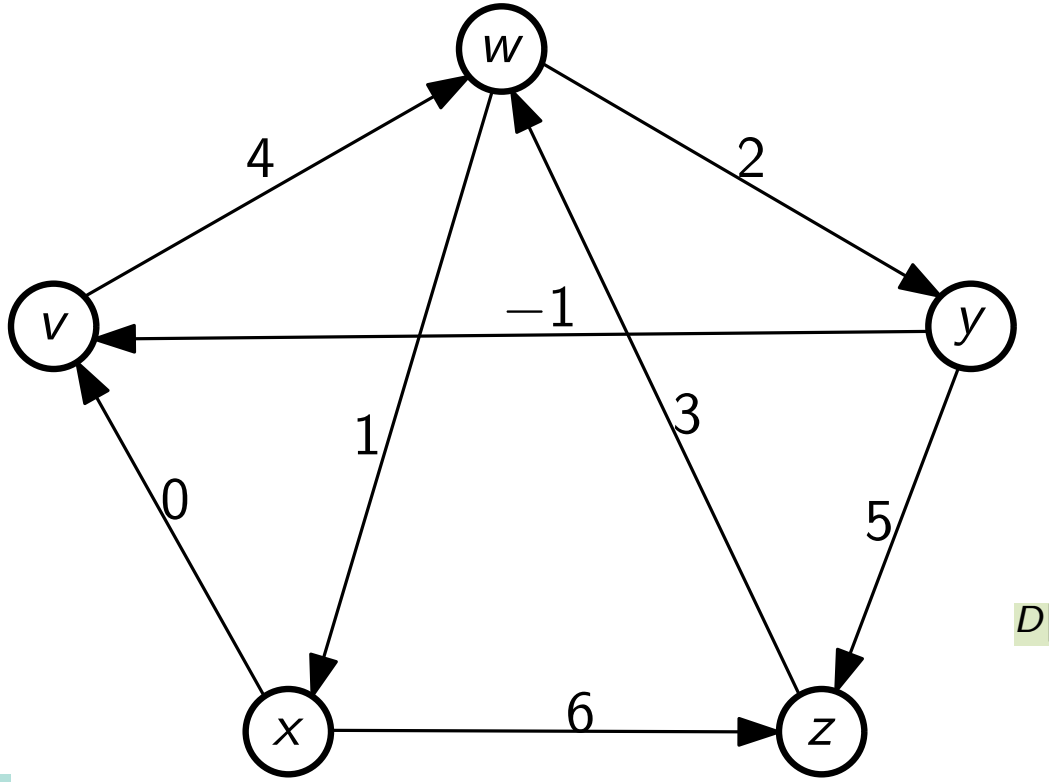
Distanzen

	v	w	x	y	z
v	0	4	5	6	11
w	1	0	1	2	7
x	0	4	0	6	6
y	−1	3	4	0	5
z	4	3	4	5	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

Floyd Warshalls Algorithmus: Beispiel



Distanzen

	v	w	x	y	z
v	0	4	5	6	11
w	1	0	1	2	7
x	0	4	0	6	6
y	−1	3	4	0	5
z	4	3	4	5	0

$D[a][b] = min(D[a][b], D[a][c] + D[c][b])$

Betrachte: v w x y z

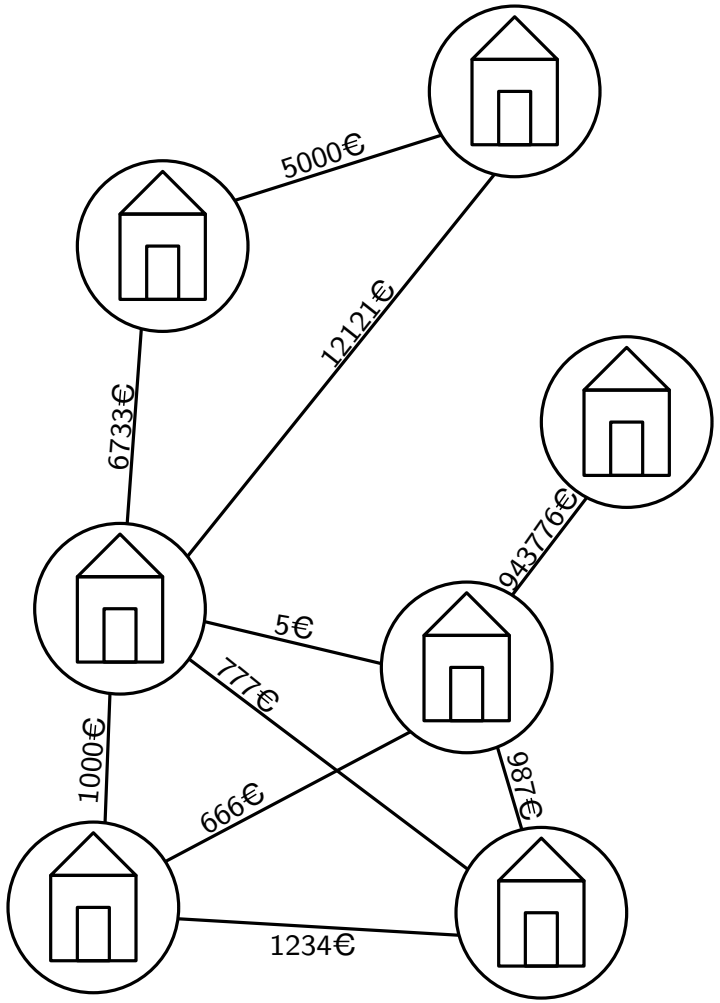
Glasfaser verlegen

Wir wollen Glasfaser verlegen

Glasfaser verlegen

Wir wollen Glasfaser verlegen

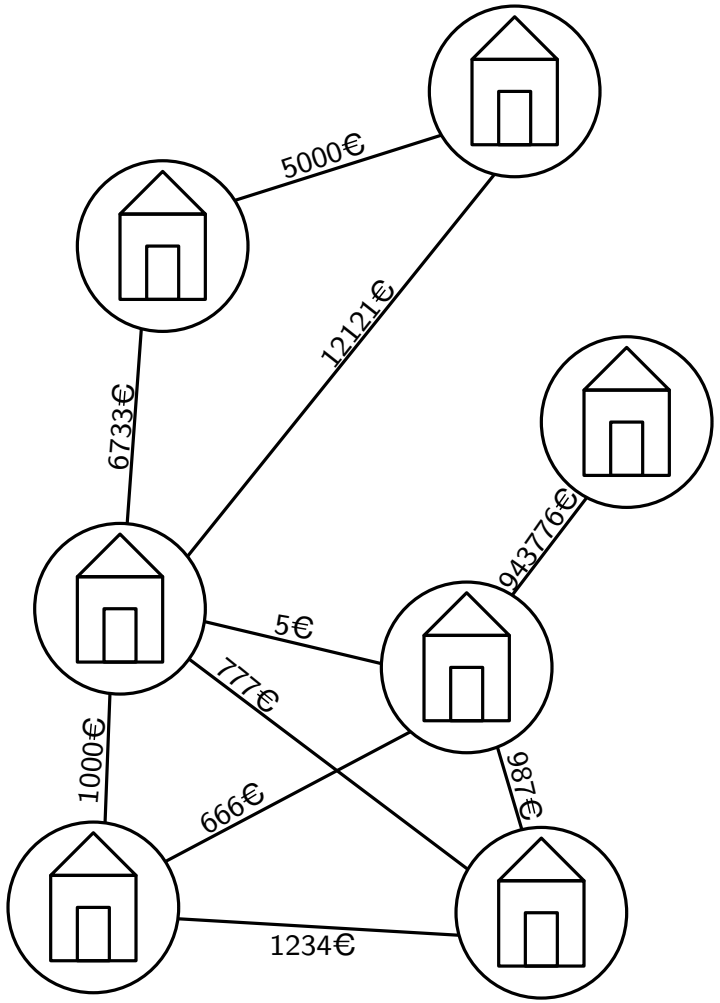
- Gegeben:
  - Liste an Häusern
  - Liste an Paaren von Häusern zum möglichen Verlegen von Leitungen
  - Für jede mögliche Leitung, Kosten fürs bauen dieser Leitung



Glasfaser verlegen

Wir wollen Glasfaser verlegen

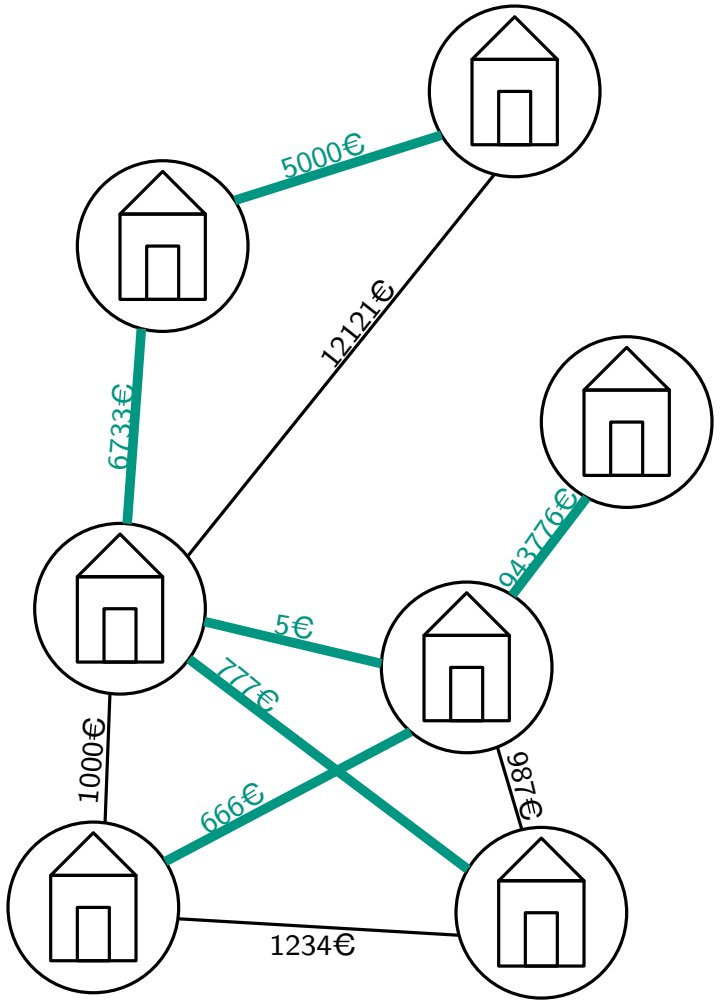
- Gegeben:
  - Liste an Häusern
  - Liste an Paaren von Häusern zum möglichen Verlegen von Leitungen
  - Für jede mögliche Leitung, Kosten fürs bauen dieser Leitung
- Gesucht:
  - Wahl von Leitungen die gebaut werden
    - Alle Häuser haben Internet
    - Ausbau kostet möglichst wenig



Glasfaser verlegen

Wir wollen Glasfaser verlegen

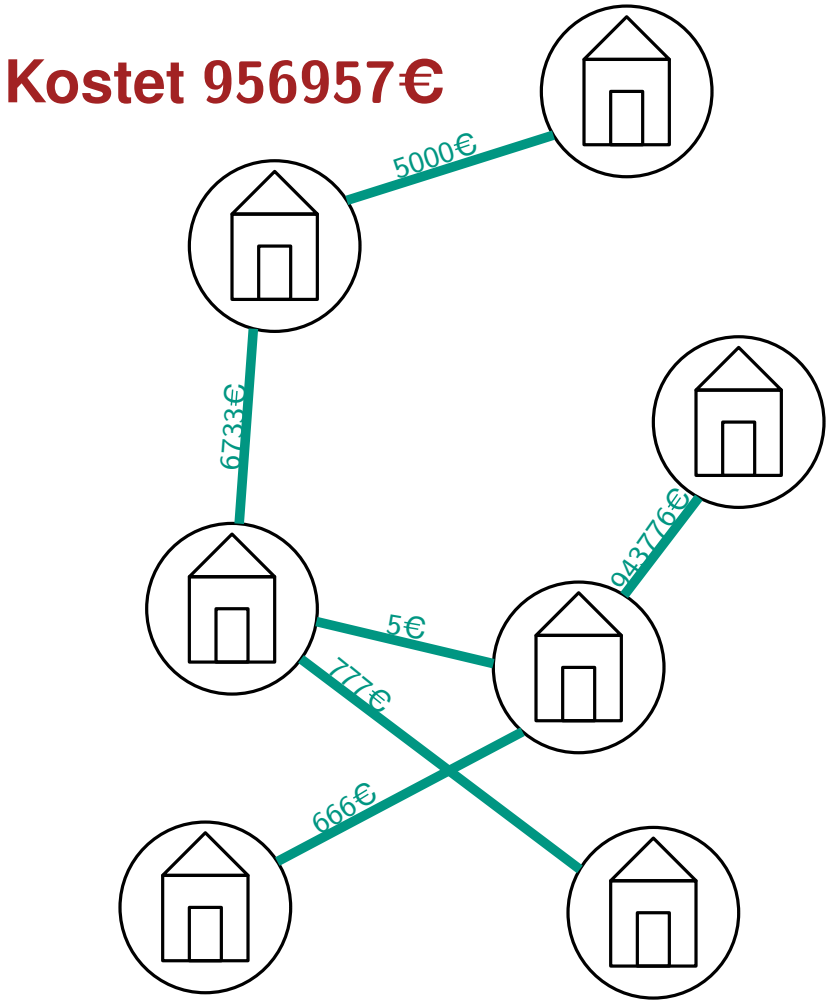
- Gegeben:
  - Liste an Häusern
  - Liste an Paaren von Häusern zum möglichen Verlegen von Leitungen
  - Für jede mögliche Leitung, Kosten fürs bauen dieser Leitung
- Gesucht:
  - Wahl von Leitungen die gebaut werden
    - Alle Häuser haben Internet
    - Ausbau kostet möglichst wenig



Glasfaser verlegen

Wir wollen Glasfaser verlegen

- Gegeben:
  - Liste an Häusern
  - Liste an Paaren von Häusern zum möglichen Verlegen von Leitungen
  - Für jede mögliche Leitung, Kosten fürs bauen dieser Leitung
- Gesucht:
  - Wahl von Leitungen die gebaut werden
    - Alle Häuser haben Internet
    - Ausbau kostet möglichst wenig



Glasfaser verlegen: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste an Wohnhaus
- Liste an möglichen Leitungen
  - Kosten für jede mögliche Leitung

Glasfaser verlegen: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste an Wohnhaus
- Liste an möglichen Leitungen
  - Kosten für jede mögliche Leitung

Eingabegraph:

- Ein Knoten pro Wohnhaus
- Kante pro möglicher Leitung
- Jede Kante hat ein (positives) Gewicht, für die Kosten der Leitung

Glasfaser verlegen: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste an Wohnhaus
- Liste an möglichen Leitungen
  - Kosten für jede mögliche Leitung

Problemstellung:

- Wähle Leitungen zum bauen, sodass:
  - Jedes Haus ist erreicht
  - Die Summe an Kosten ist minimal

Eingabegraph:

- Ein Knoten pro Wohnhaus
- Kante pro möglicher Leitung
- Jede Kante hat ein (positives) Gewicht, für die Kosten der Leitung

Glasfaser verlegen: Graphmodellierung

Das kann man als Graph modellieren!

Eingabe:

- Liste an Wohnhaus
- Liste an möglichen Leitungen
  - Kosten für jede mögliche Leitung

Problemstellung:

- Wähle Leitungen zum bauen, sodass:
  - Jedes Haus ist erreicht
  - Die Summe an Kosten ist minimal

Eingabegraph:

- Ein Knoten pro Wohnhaus
- Kante pro möglicher Leitung
- Jede Kante hat ein (positives) Gewicht, für die Kosten der Leitung

Graphproblem:

- Wähle Teilgraph, sodass:
  - Jeder Knoten in der gleichen Zusammenhangskomponente
  - Summe der Kantengewichte minimal

Minimale Spannbäume

Suche: minimaler Teilgraph der alle Knoten verbindet

Minimale Spannbäume

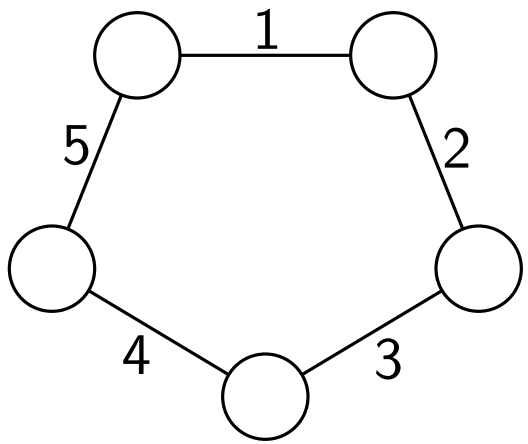
Suche: minimaler Teilgraph der alle Knoten verbindet

- Damit alle Knoten verbunden sind: Mindestens $n - 1$ Kanten

Minimale Spannbäume

Suche: minimaler Teilgraph der alle Knoten verbindet

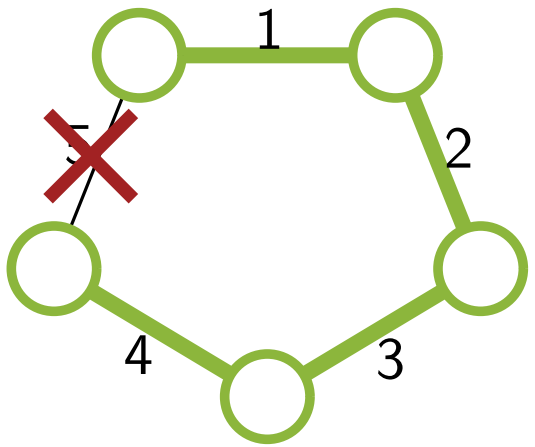
- Damit alle Knoten verbunden sind: Mindestens $n - 1$ Kanten
- Aus Kreisen kann man immer 1 Kante weglassen und bleibt verbunden



Minimale Spannbäume

Suche: minimaler Teilgraph der alle Knoten verbindet

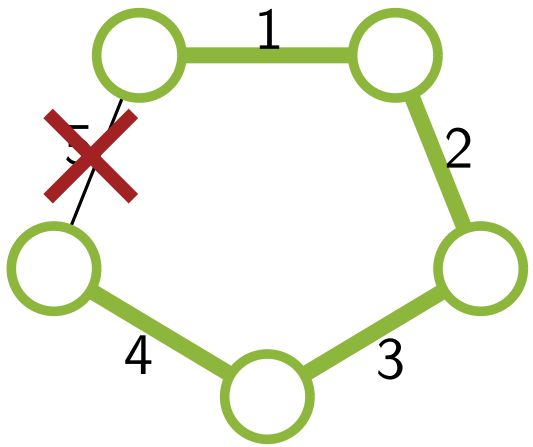
- Damit alle Knoten verbunden sind: Mindestens $n - 1$ Kanten
- Aus Kreisen kann man immer 1 Kante weglassen und bleibt verbunden



Minimale Spannbäume

Suche: minimaler Teilgraph der alle Knoten verbindet

- Damit alle Knoten verbunden sind: Mindestens $n - 1$ Kanten
- Aus Kreisen kann man immer 1 Kante weglassen und bleibt verbunden



- Erreicht jeden Knoten
- Minimal

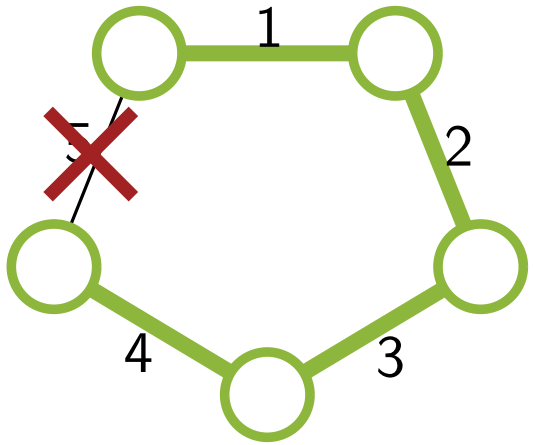
Minimale Spannbäume

Suche: minimaler Teilgraph der alle Knoten verbindet

- Damit alle Knoten verbunden sind: Mindestens $n - 1$ Kanten
- Aus Kreisen kann man immer 1 Kante weglassen und bleibt verbunden

$\Longrightarrow$ Teilgraph ist kreisfrei also Baum

  - Heißt Spannbaum
  - Durch Minimalität: Minimaler Spannbaum (MST)



- Erreicht jeden Knoten
- Minimal

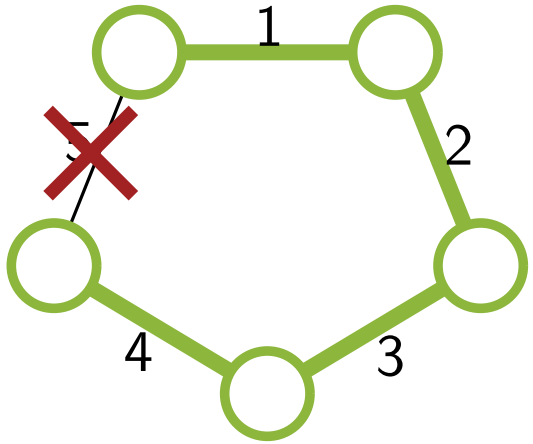
Minimale Spannbäume

Suche: minimaler Teilgraph der alle Knoten verbindet

- Damit alle Knoten verbunden sind: Mindestens $n - 1$ Kanten
- Aus Kreisen kann man immer 1 Kante weglassen und bleibt verbunden

$\Longrightarrow$ Teilgraph ist kreisfrei also Baum

  - Heißt Spannbaum
  - Durch Minimalität: Minimaler Spannbaum (MST)



- Erreicht jeden Knoten
- Minimal

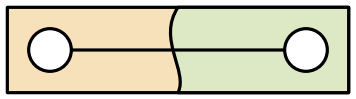
Kleine Einschränkung:
Alle Kantengewichte unterschiedlich

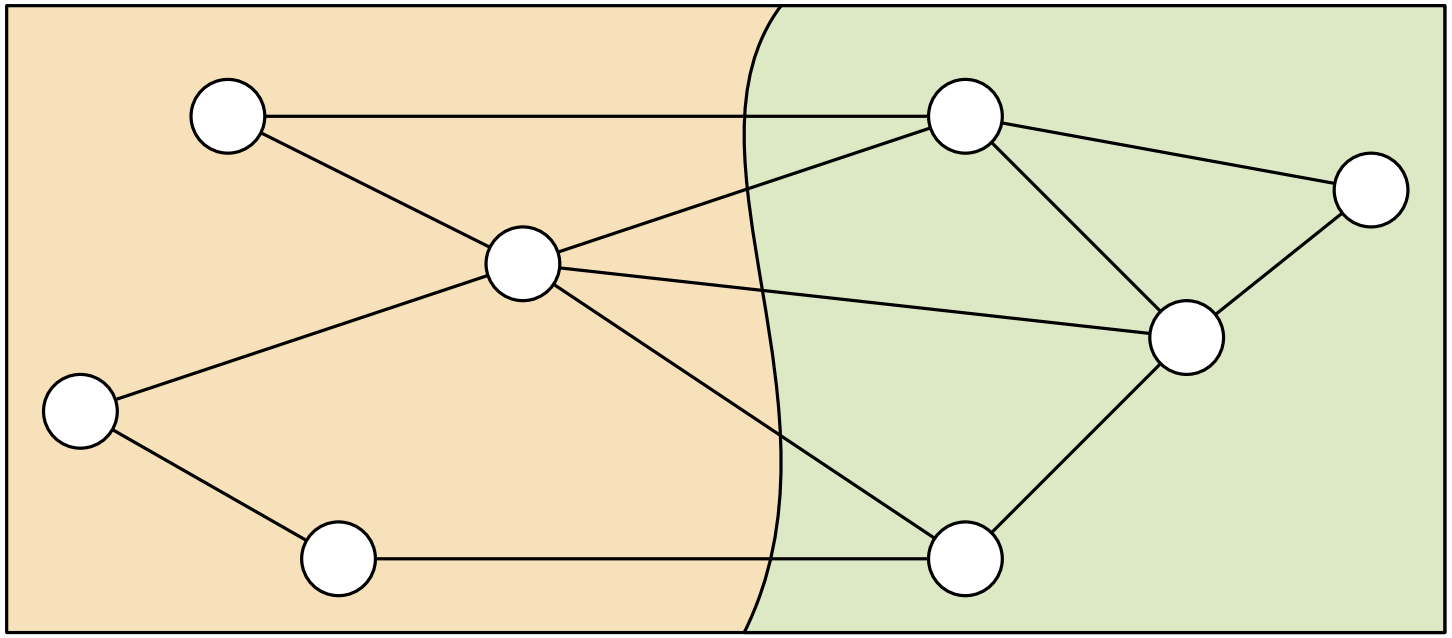
- Dadurch ist MST eindeutig
- Keine wirkliche Einschränkung: Algorithmen funktionieren auch ohne Einzigartigkeit

Minimale Spannbäume: Schnitteigenschaft

MST muss immer kleinste Kante jedes Schnitts enthalten

Schnittkante:



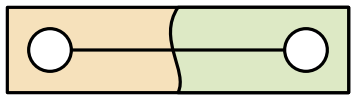


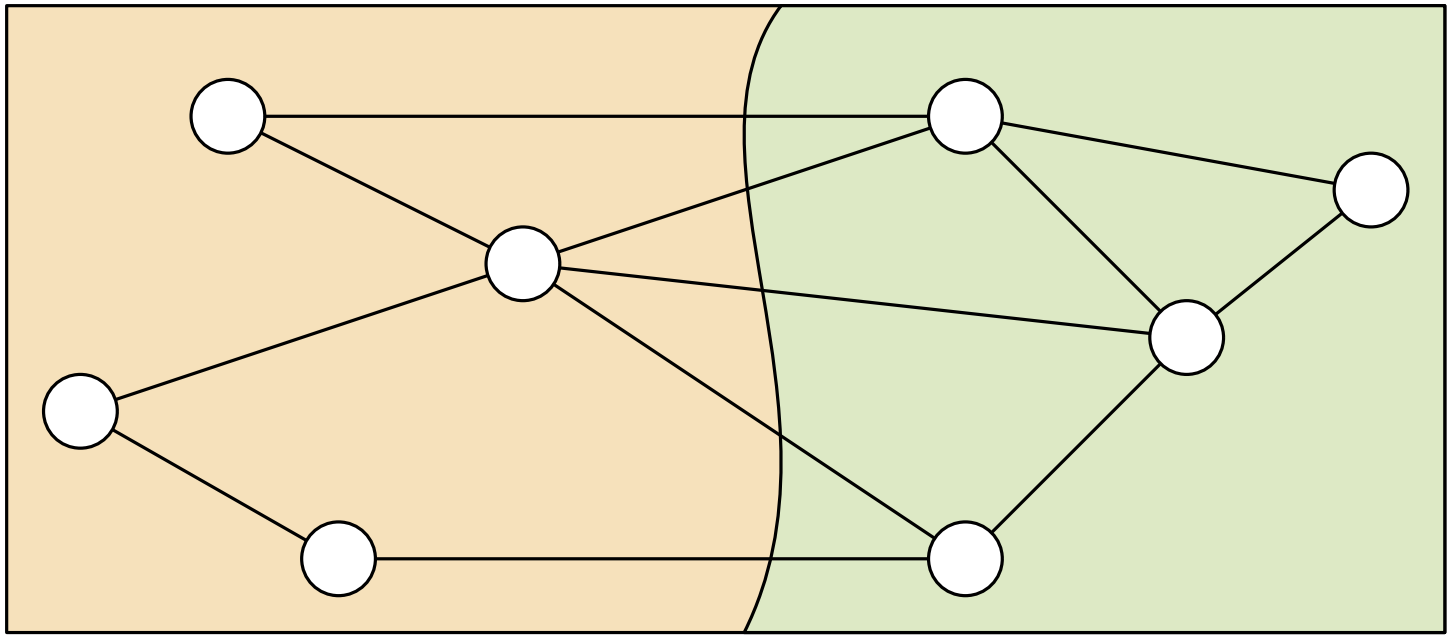
Minimale Spannbäume: Schnitteigenschaft

MST muss immer kleinste Kante jedes Schnitts enthalten

- MST muss alle Knoten verbinden

Schnittkante:



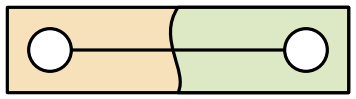


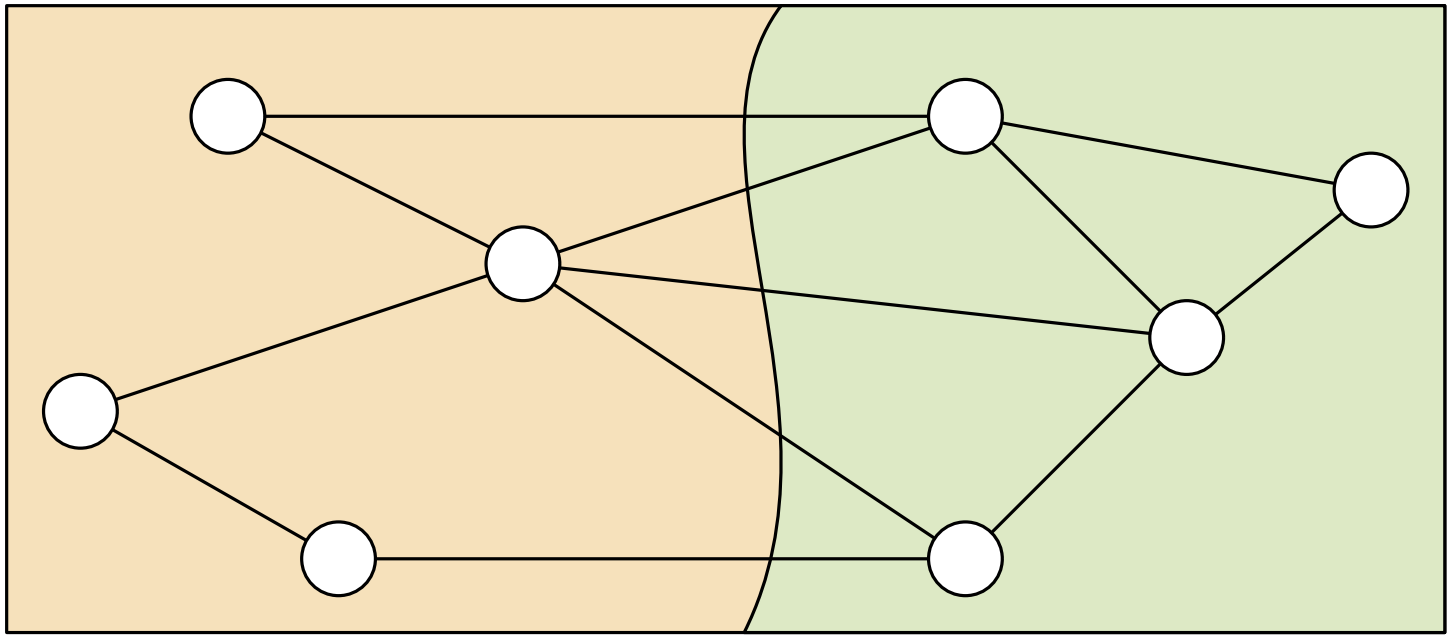
Minimale Spannbäume: Schnitteigenschaft

MST muss immer kleinste Kante jedes Schnitts enthalten

- MST muss alle Knoten verbinden
  - Mindestens eine Schnittkante muss gewählt werden
- Idee: Wähle kleinste Schnittkante

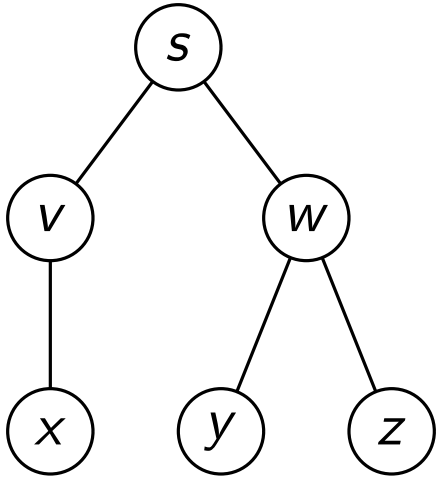
Schnittkante:





Einschub: Baumrepräsentation

Wie repräsentiert man einen Baum?

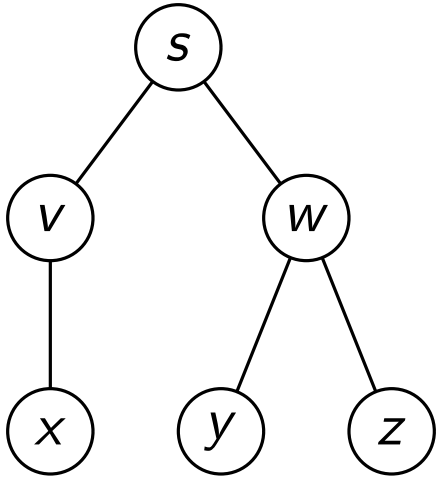


Einschub: Baumrepräsentation

Wie repräsentiert man einen Baum?

Erste Idee: Liste an Kanten

$\{s, v\}, \{s, w\}, \{v, x\}, \{w, y\}, \{w, z\}$

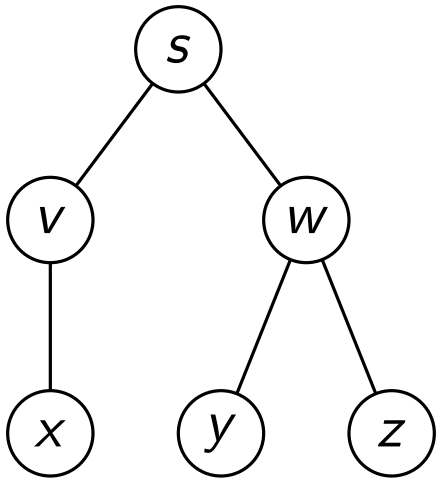


Einschub: Baumrepräsentation

Wie repräsentiert man einen Baum?

Erste Idee: Liste an Kanten

$\{s, v\}, \{s, w\}, \{v, x\}, \{w, y\}, \{w, z\}$



- Vorteile:
  - Eltern sehen Kinder
  - Kinder sehen Eltern
- Nachteile:
  - Abfragen in $O(m)$
  - Falls sortiert, trotzdem $\log(m)$

Zweite Idee: Speichere Elternknoten

s	v	w	x	y	z
	s	s	v	w	w

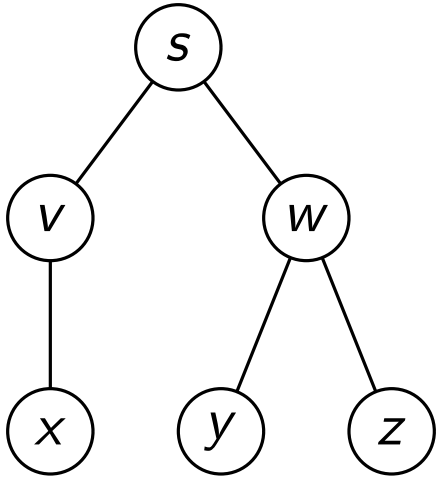
Einschub: Baumrepräsentation

Wie repräsentiert man einen Baum?

Erste Idee: Liste an Kanten

$\{s, v\}, \{s, w\}, \{v, x\}, \{w, y\}, \{w, z\}$

- Vorteile:
  - Eltern sehen Kinder
  - Kinder sehen Eltern
- Nachteile:
  - Abfragen in $O(m)$
  - Falls sortiert, trotzdem $\log(m)$



Zweite Idee: Speichere Elternknoten

s	v	w	x	y	z
	s	s	v	w	w

- Vorteile:
  - Zugriff in $\Theta(m)$
- Nachteile:
  - Eltern sehen Kinder nicht

Algorithmus von Prim: Pseudocode

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

Algorithmus von Prim: Pseudocode

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

- Priority Queue speichert Gewicht der aktuell kleinsten Kante zu den Knoten
- Am Anfang hat jeder Knoten Distanz ∞ bevor er entdeckt wird

Algorithmus von Prim: Pseudocode

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

- Um den MST zu rekonstruieren, speichern wir ihn als Parent-Array

Algorithmus von Prim: Pseudocode

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

- Der unexplorierte Knoten mit minimalster Distanz wird als nächstes zum MST hinzugefügt

Algorithmus von Prim: Pseudocode

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

- Nach jedem neuen Knoten müssen Distanzen der Nachbarn und mögliche Elternknoten aktualisiert werden

Algorithmus von Prim: Laufzeit

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

Welche Laufzeit hat dieser Algorithmus?

Algorithmus von Prim: Laufzeit

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

Welche Laufzeit hat dieser Algorithmus?

- Leere Priority-Queue erstellen in $\Theta(1)$
- Array der Größe n in $\Theta(n)$ erstellen
- n Knoten mit **buildHeap** in $\Theta(n)$ einfügen

Algorithmus von Prim: Laufzeit

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

Welche Laufzeit hat dieser Algorithmus?

- Leere Priority-Queue erstellen in $\Theta(1)$
- Array der Größe n in $\Theta(n)$ erstellen
- n Knoten mit **buildHeap** in $\Theta(n)$ einfügen
- Jeder der n Knoten wird einmal mit **popMin** entfernt

Algorithmus von Prim: Laufzeit

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

Welche Laufzeit hat dieser Algorithmus?

- Leere Priority-Queue erstellen in $\Theta(1)$
- Array der Größe n in $\Theta(n)$ erstellen
- n Knoten mit **buildHeap** in $\Theta(n)$ einfügen
- Jeder der n Knoten wird einmal mit **popMin** entfernt
- Jede der m Kanten wird maximal ein mal zum aktualisieren der Distanz benutzt
- Pro Aktualisierung wird **decPrio** aufgerufen

Algorithmus von Prim: Laufzeit

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

Welche Laufzeit hat dieser Algorithmus?

- Leere Priority-Queue erstellen in $\Theta(1)$
- Array der Größe n in $\Theta(n)$ erstellen
- n Knoten mit **buildHeap** in $\Theta(n)$ einfügen
- Jeder der n Knoten wird einmal mit **popMin** entfernt
- Jede der m Kanten wird maximal ein mal zum aktualisieren der Distanz benutzt
- Pro Aktualisierung wird **decPrio** aufgerufen

Insgesamt $O(1 + n + n + n\log(n) + m\log(n))$

Algorithmus von Prim: Laufzeit

Algorithmus von Prim(*Graph* G)

```
PriorityQueue Q := empty PriorityQueue
parent := Array of size n initialized with 0
for Node v in V do
    Q.push(v, ∞)
while Q ≠ ∅ do
    u := Q.popMin()
    color(u)
    for Node v in N(u) do
        if (v is uncolored) ∧ (len(u, v) < Q.Prio(v)) then
            parent[v] = u
            Q.decPrio(v, len(u, v))
```

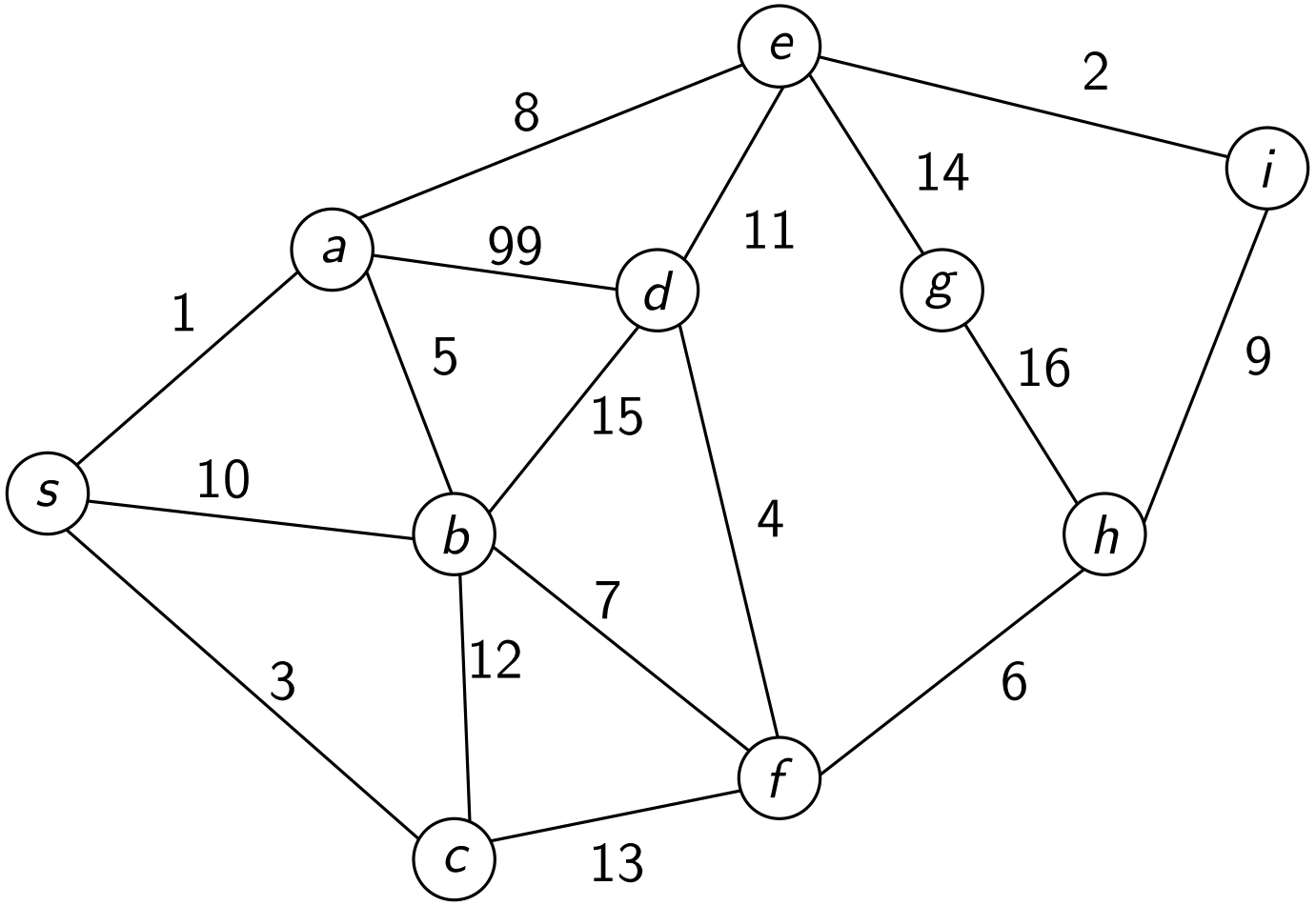
Welche Laufzeit hat dieser Algorithmus?

- Leere Priority-Queue erstellen in $\Theta(1)$
- Array der Größe n in $\Theta(n)$ erstellen
- n Knoten mit **buildHeap** in $\Theta(n)$ einfügen
- Jeder der n Knoten wird einmal mit **popMin** entfernt
- Jede der m Kanten wird maximal ein mal zum aktualisieren der Distanz benutzt
- Pro Aktualisierung wird **decPrio** aufgerufen

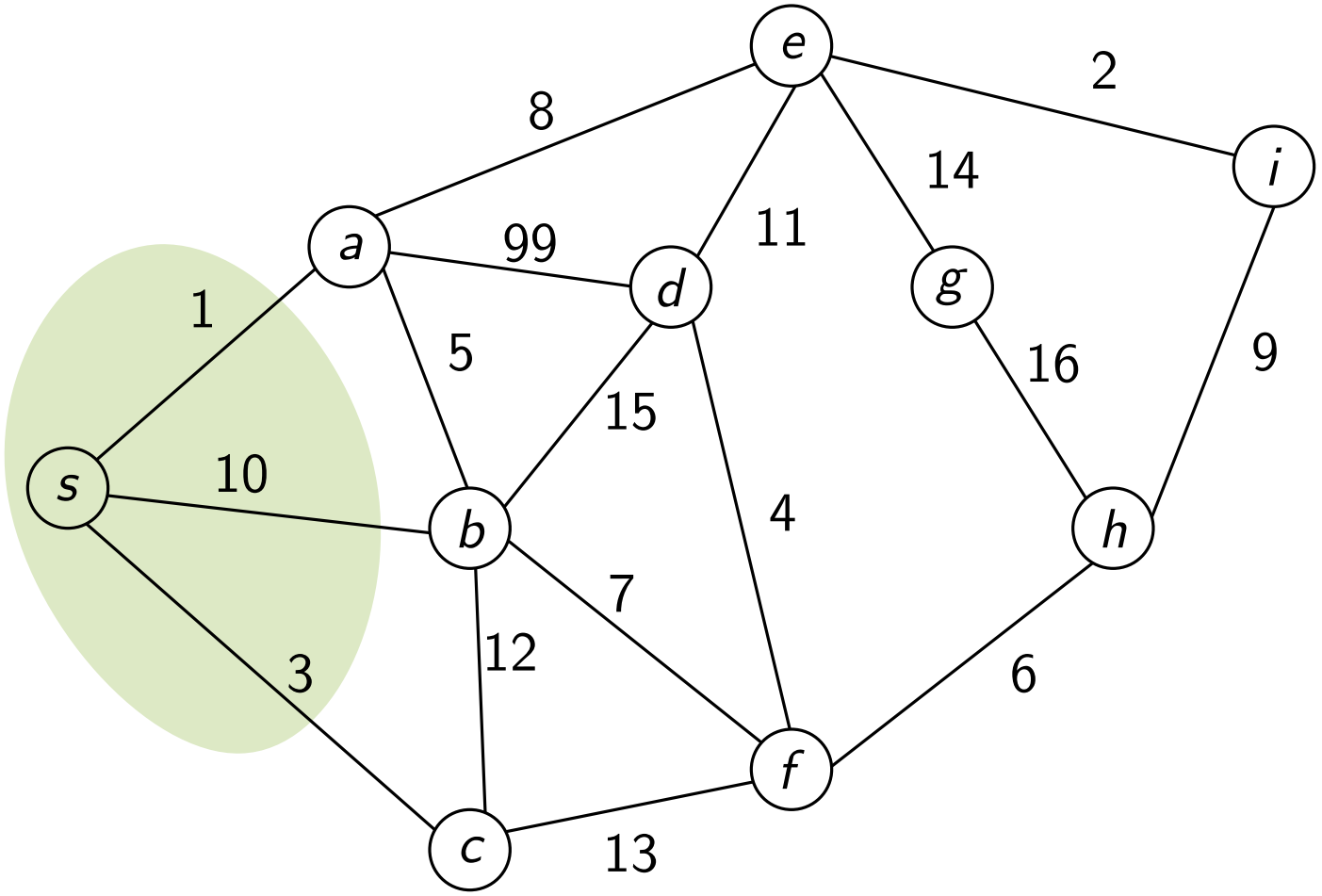
Insgesamt $O(1 + n + n + n\log(n) + m\log(n))$

$= O((n + m)\log(n))$

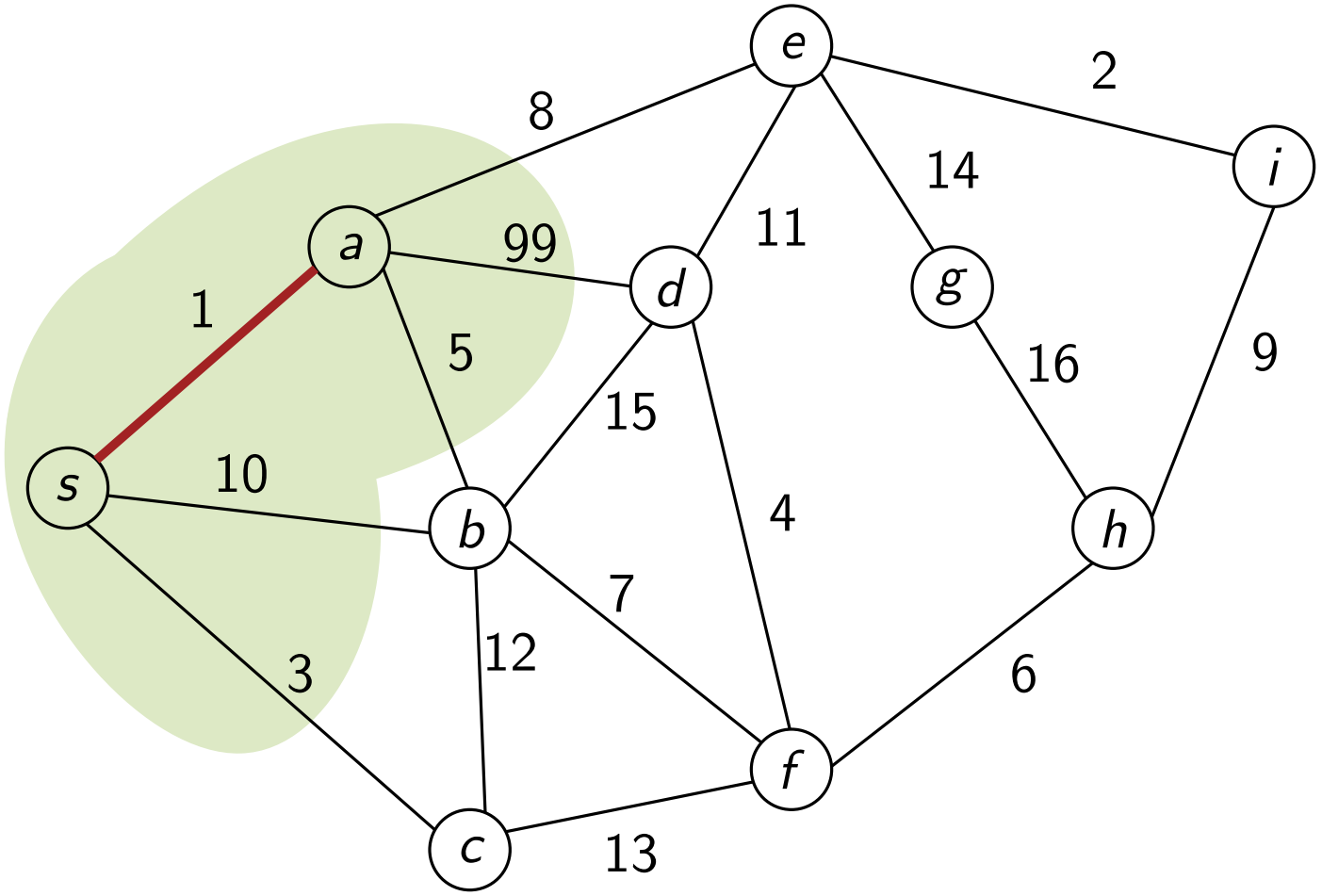
Algorithmus von Prim: Beispiel



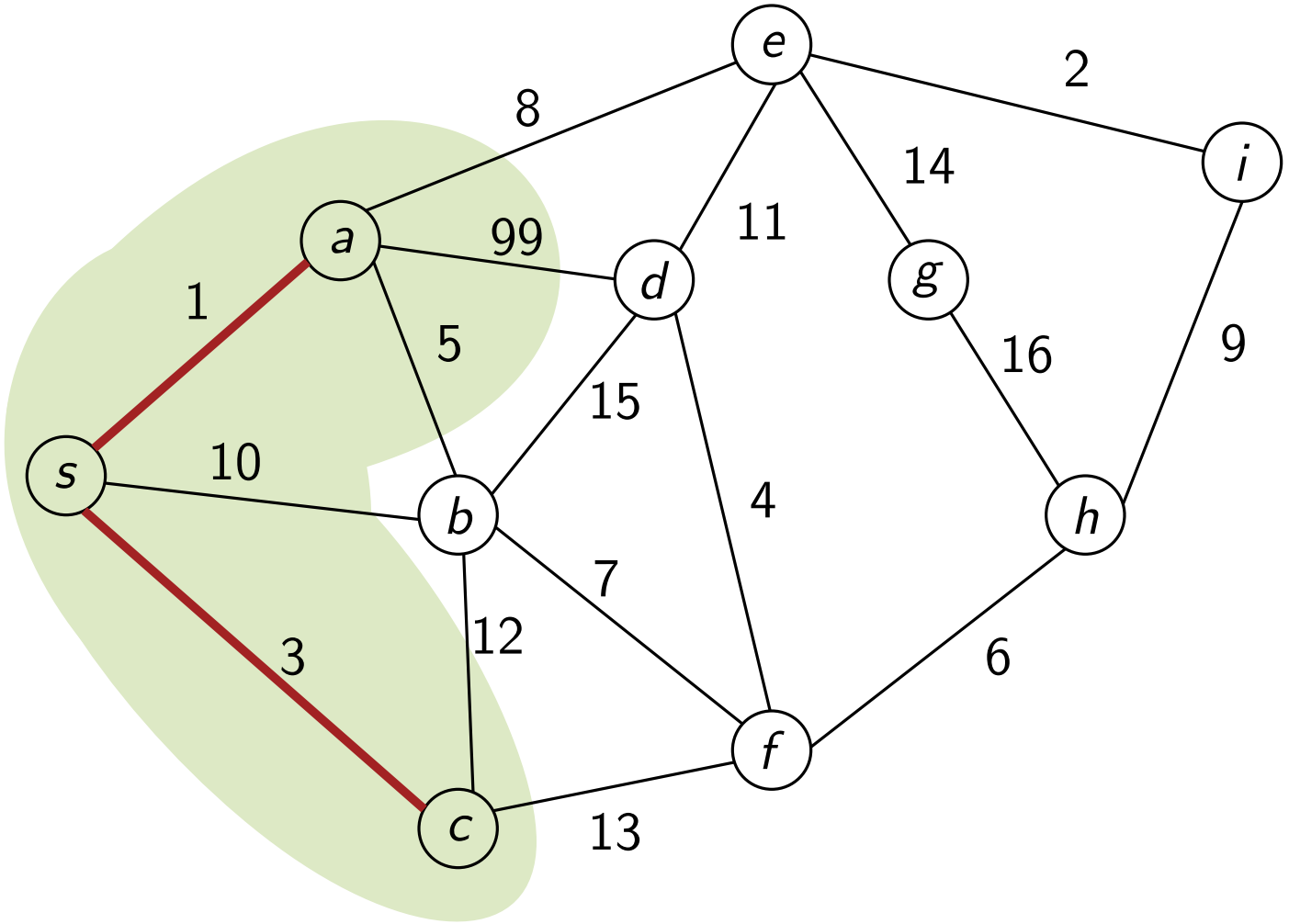
Algorithmus von Prim: Beispiel



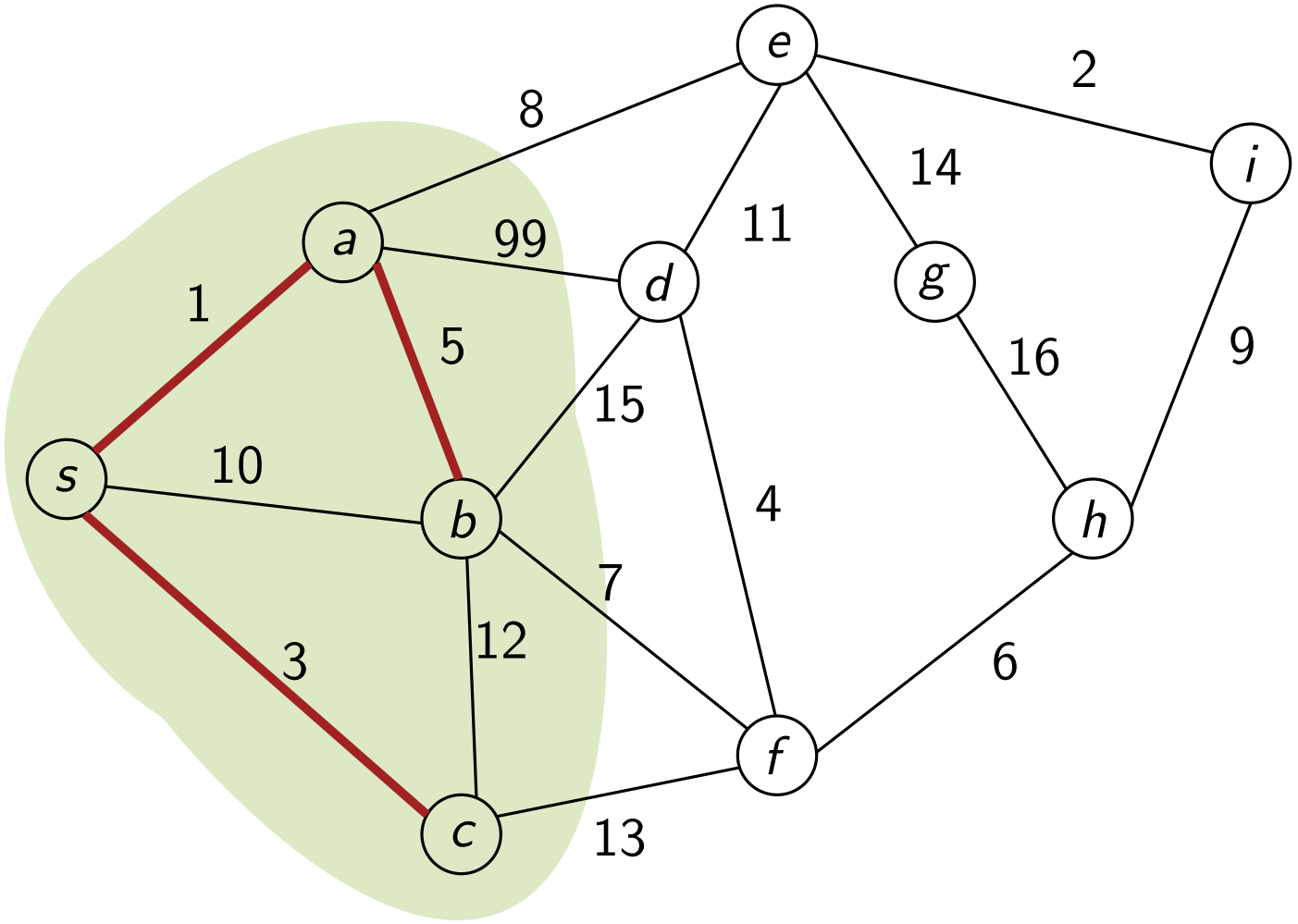
Algorithmus von Prim: Beispiel



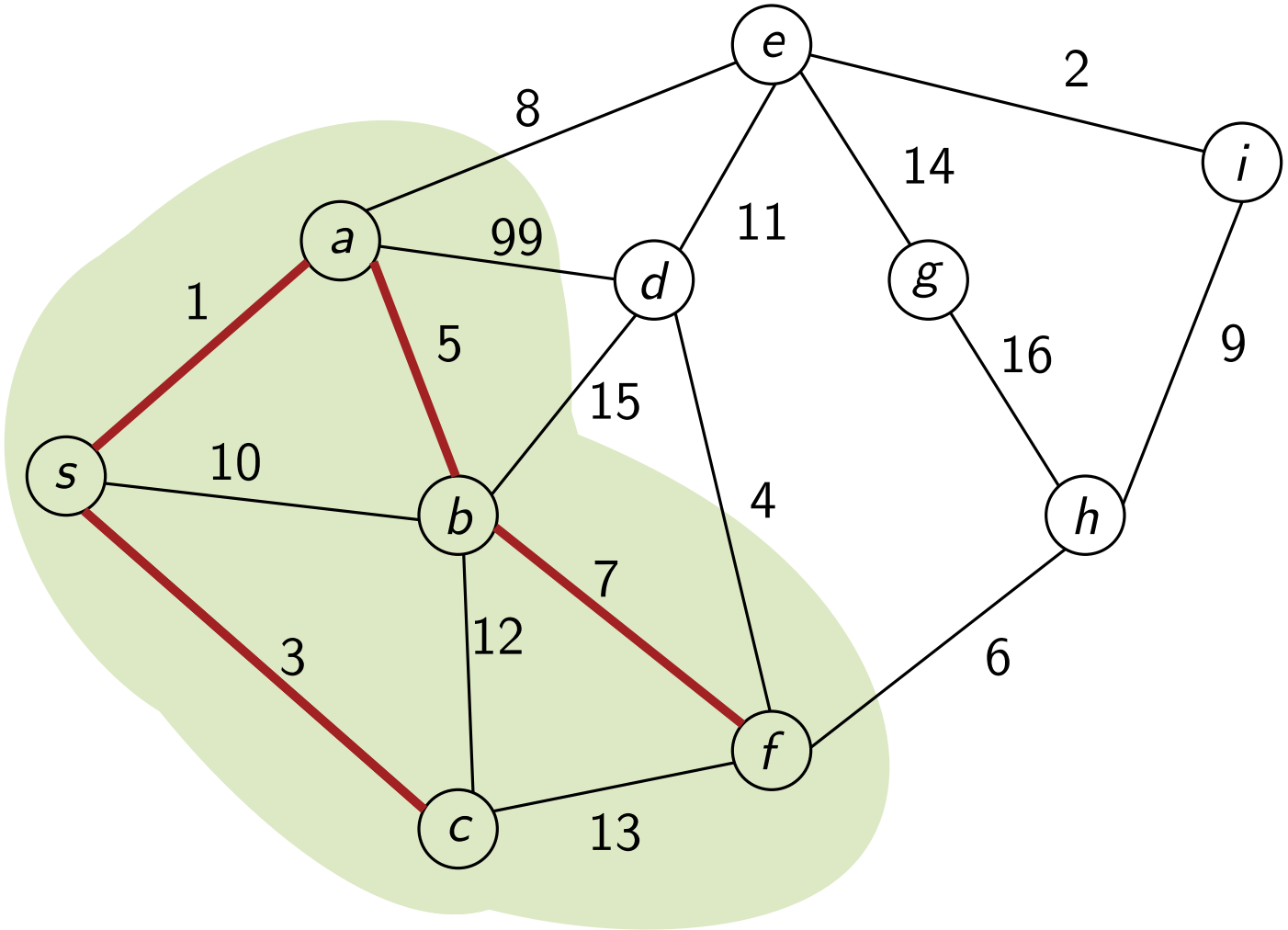
Algorithmus von Prim: Beispiel



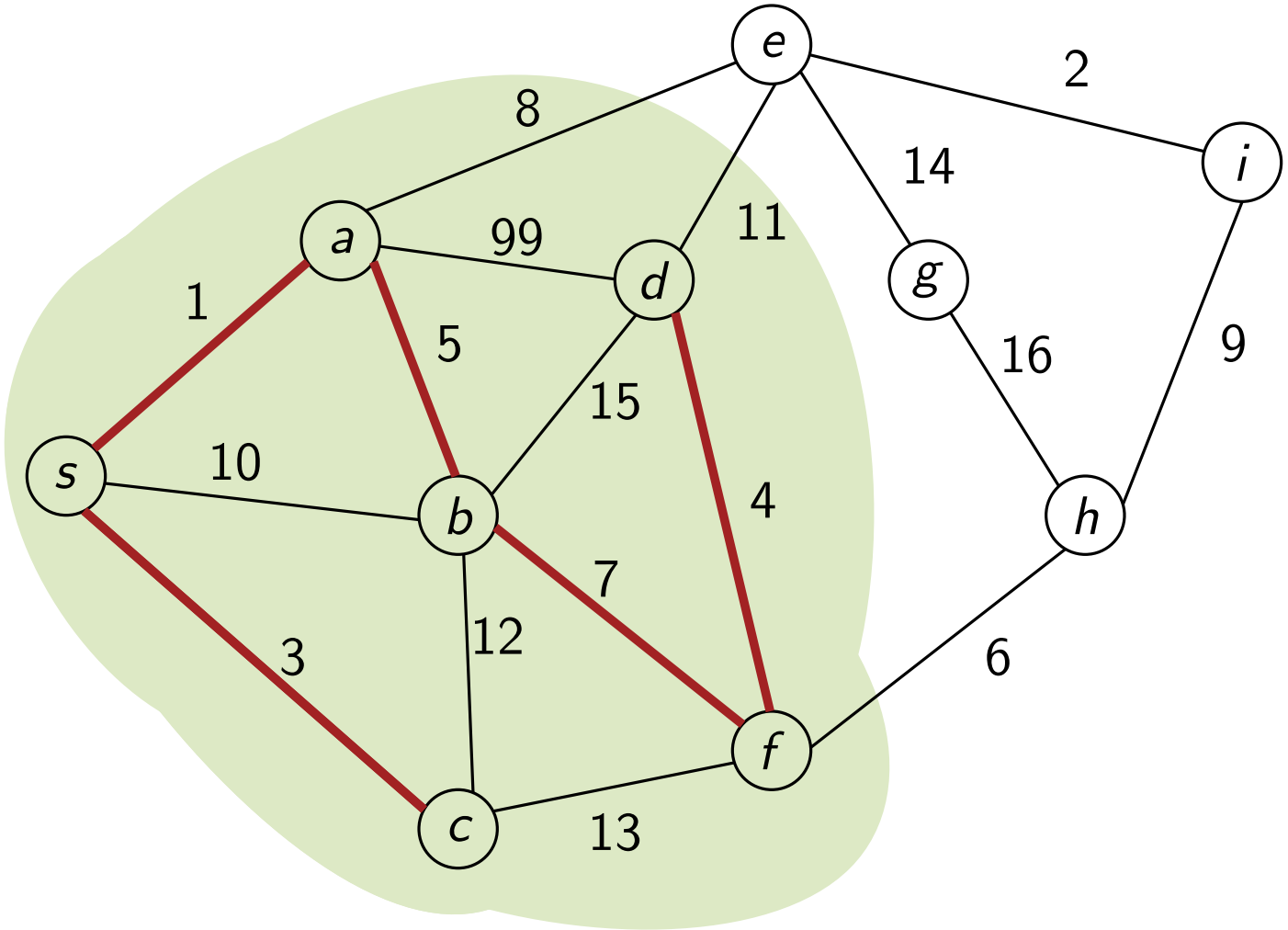
Algorithmus von Prim: Beispiel



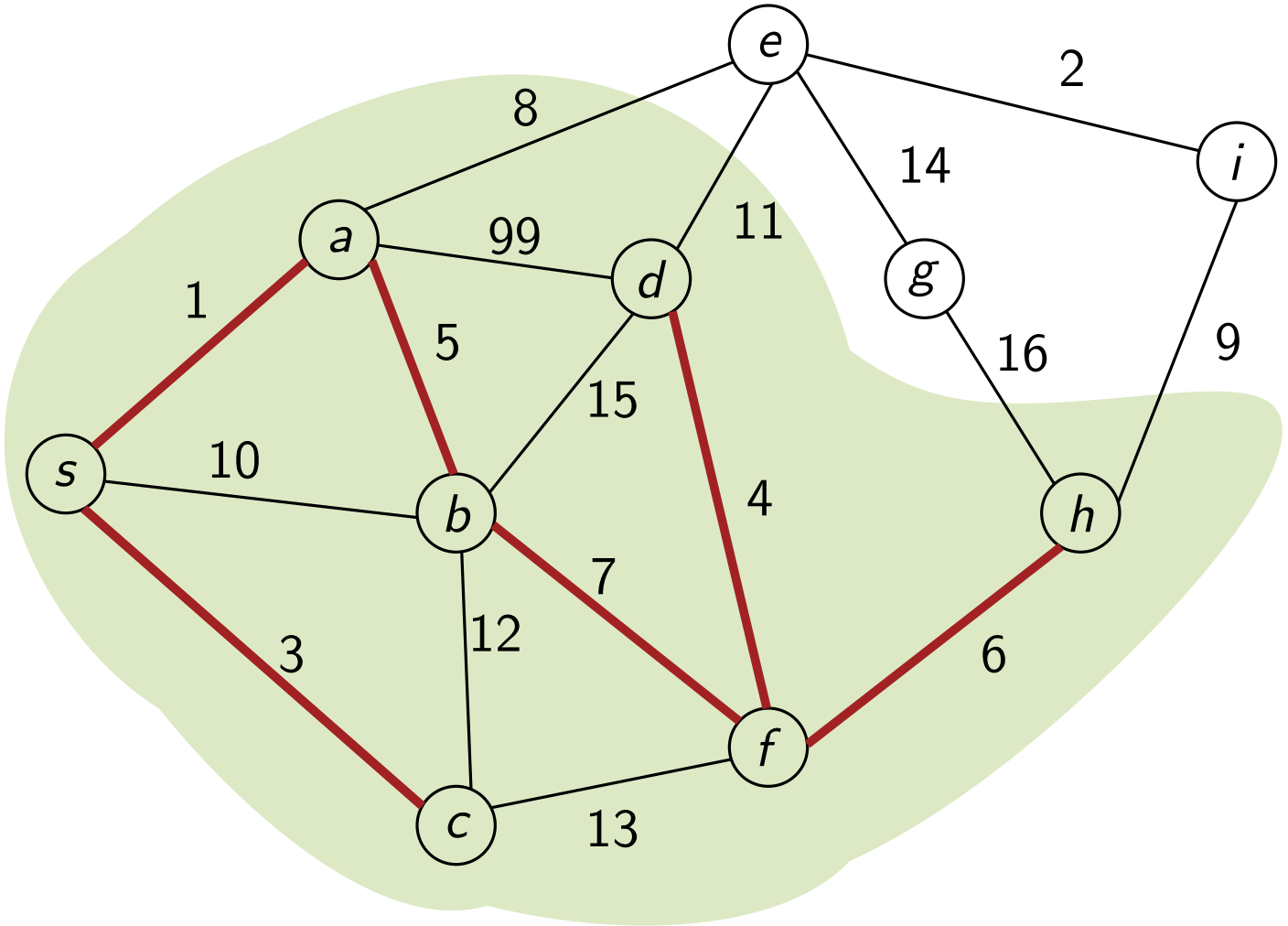
Algorithmus von Prim: Beispiel



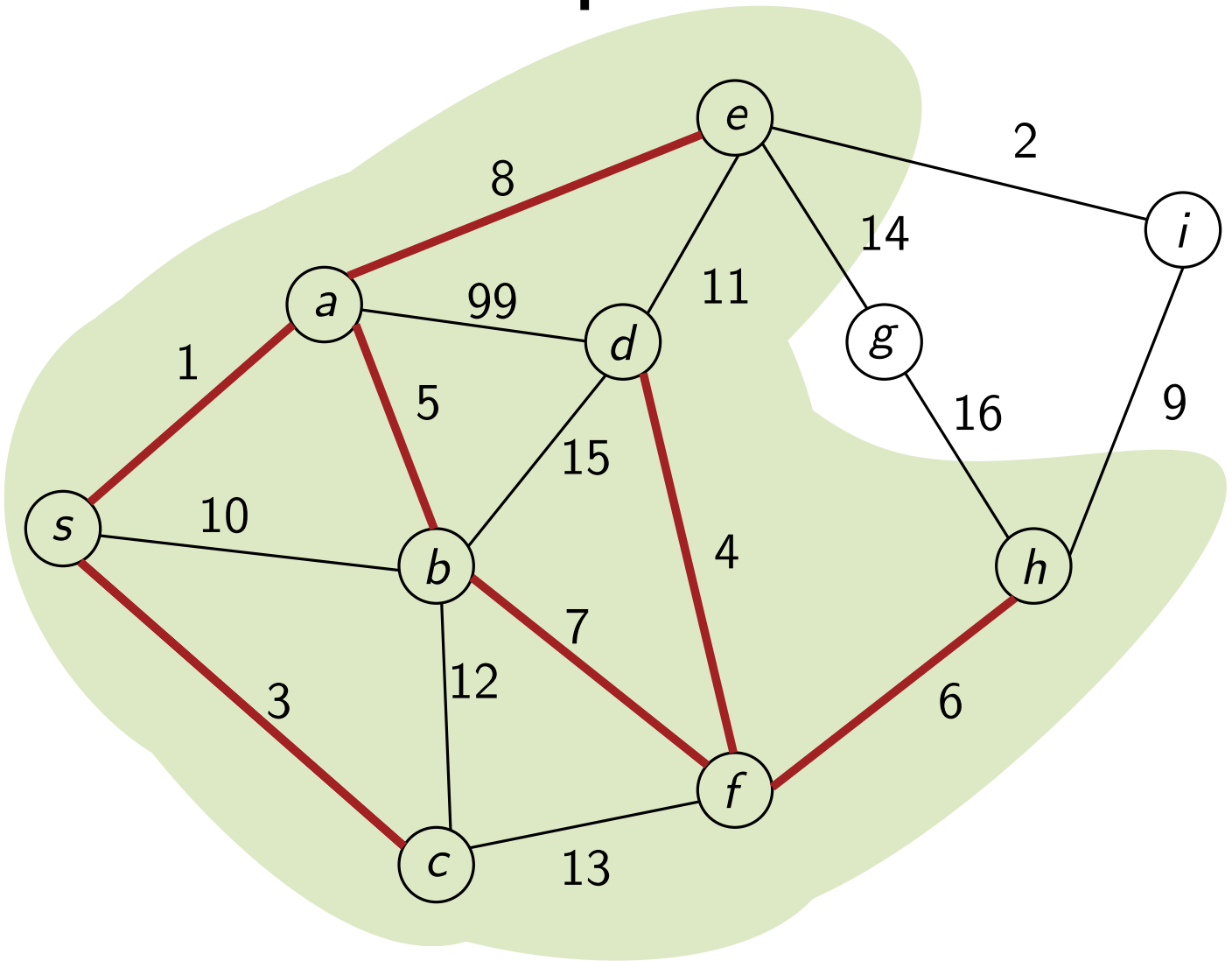
Algorithmus von Prim: Beispiel



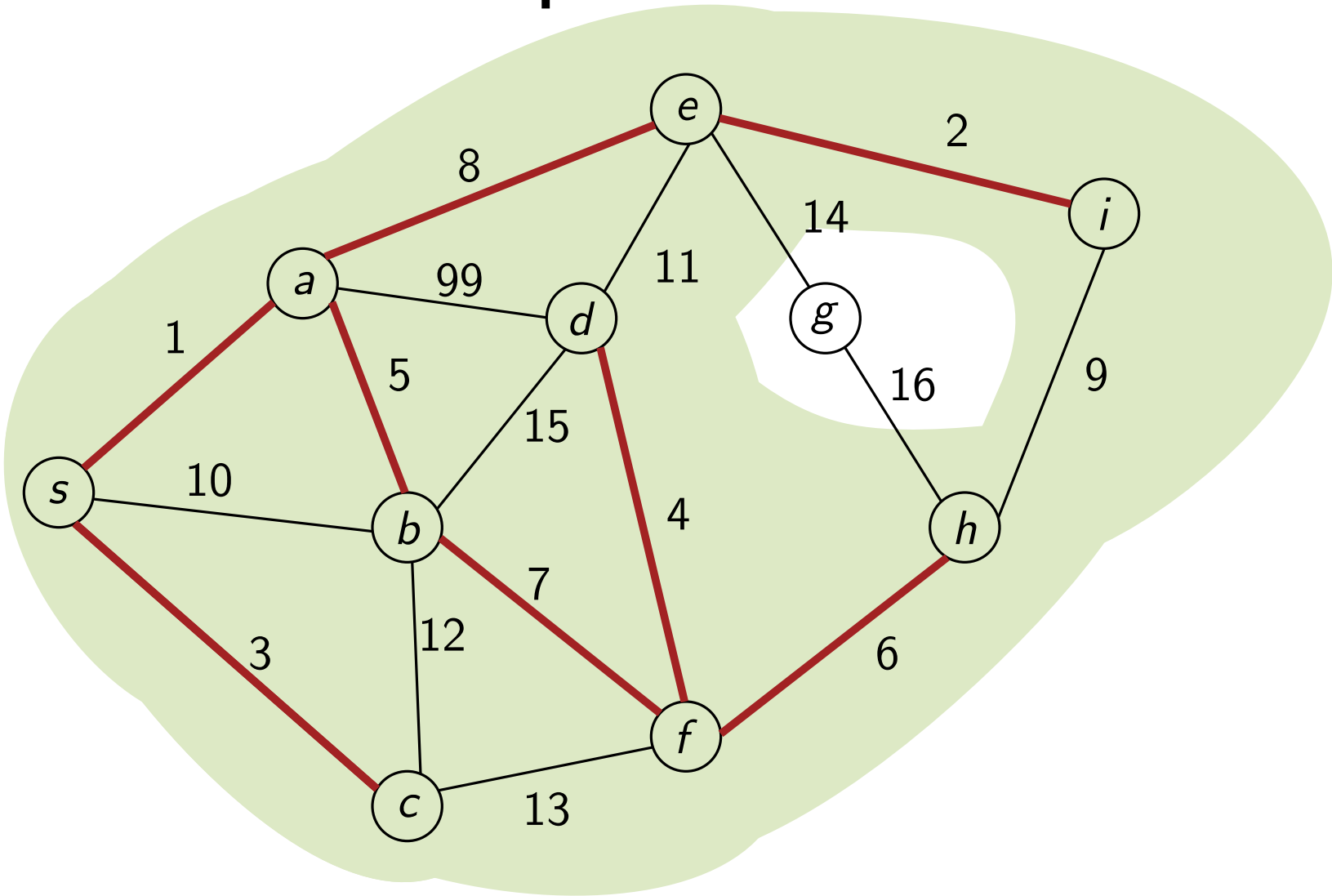
Algorithmus von Prim: Beispiel



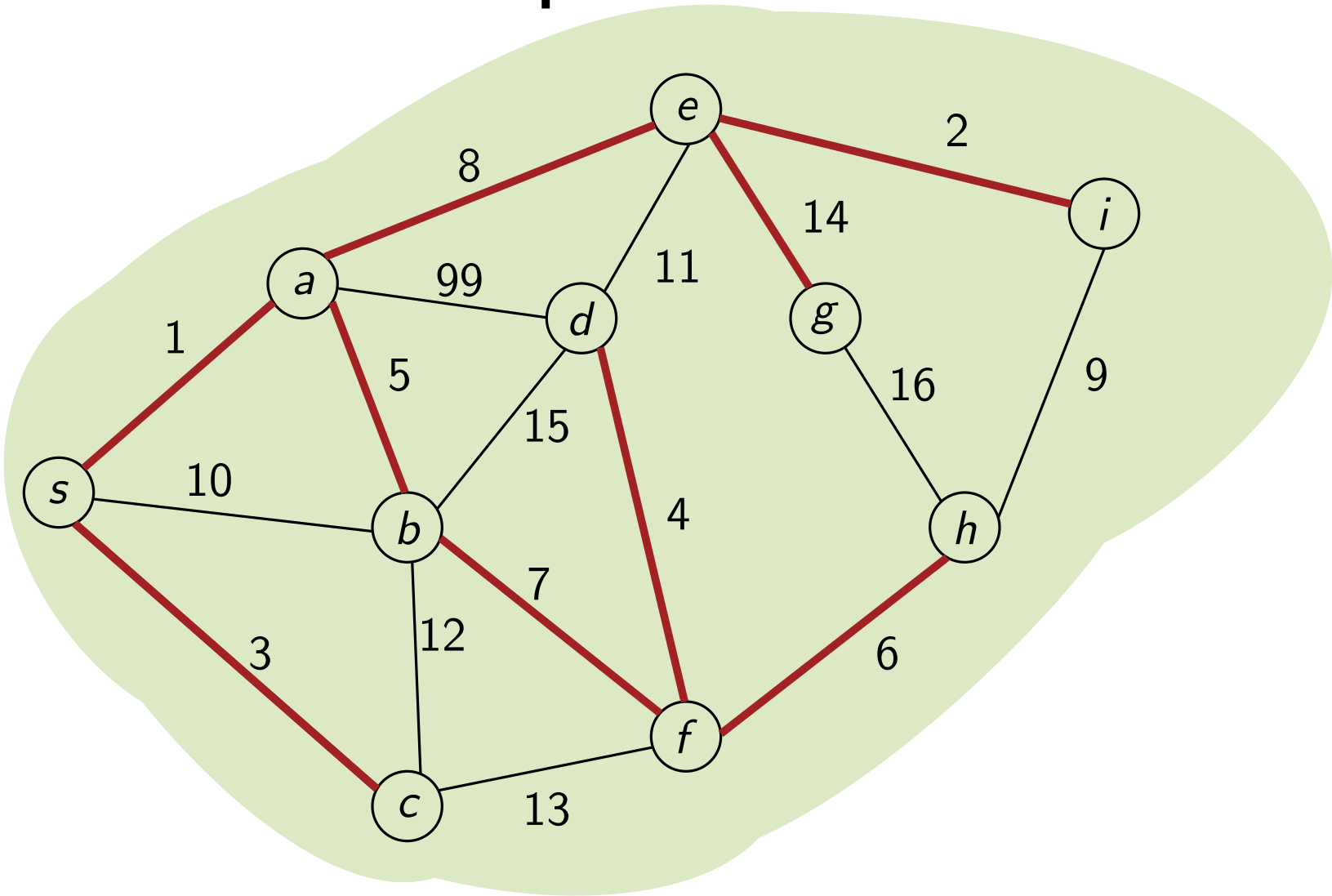
Algorithmus von Prim: Beispiel



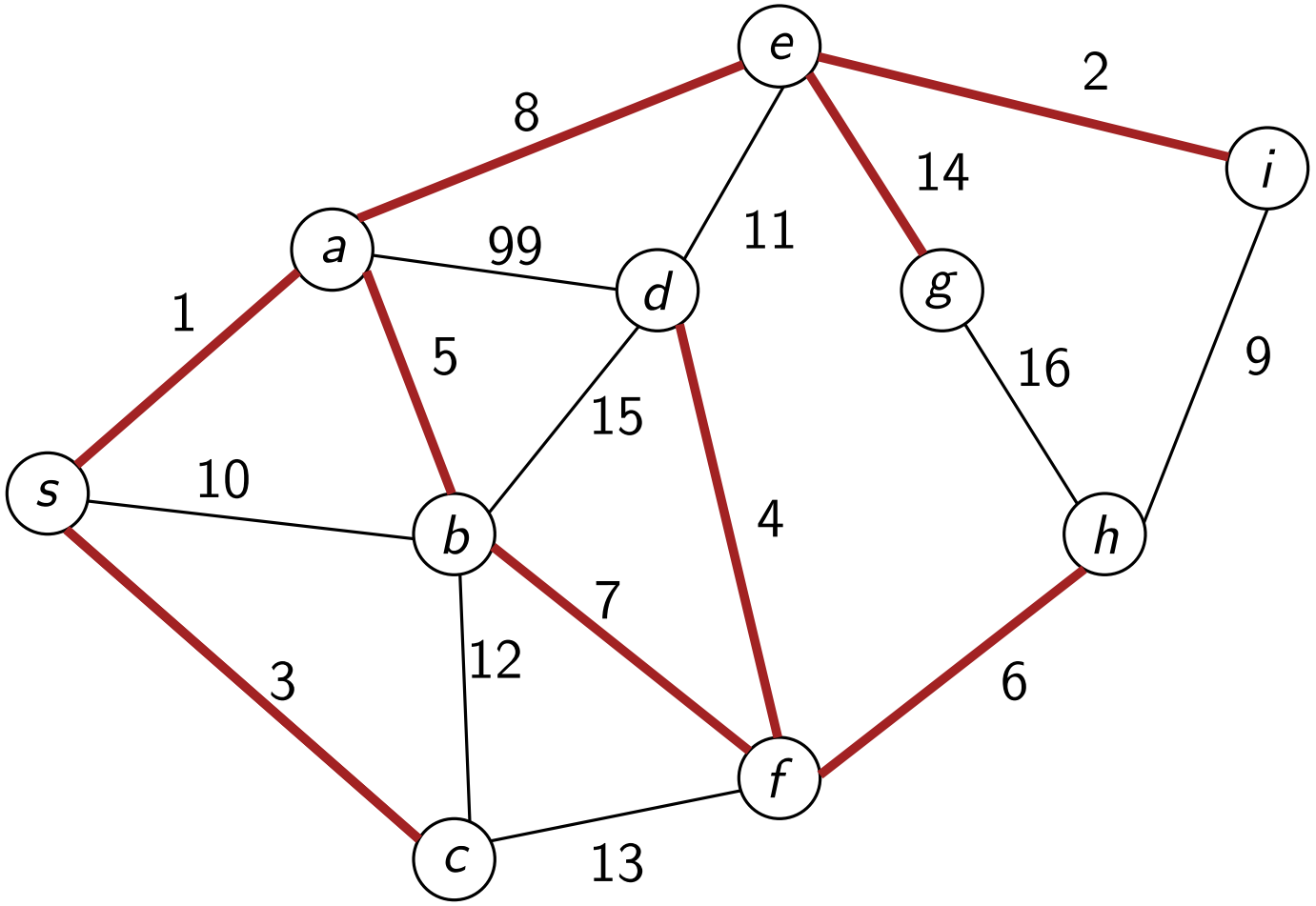
Algorithmus von Prim: Beispiel



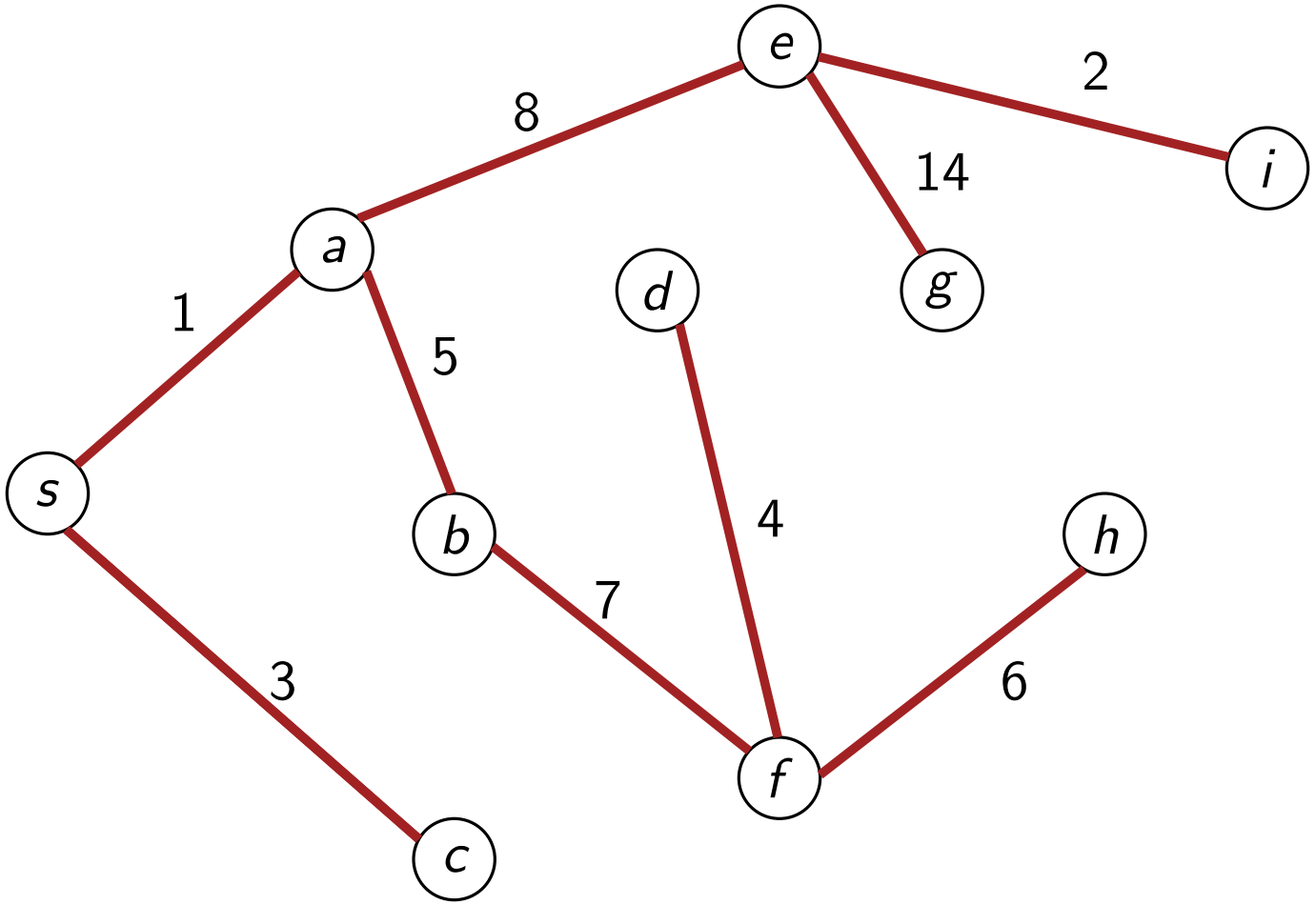
Algorithmus von Prim: Beispiel



Algorithmus von Prim: Beispiel



Algorithmus von Prim: Beispiel



Minimale Spannbäume: Lokale oder globale Strategie

Lokale Strategie: Algorithmus von Prim

- Startet bei einem Knoten
- Erweitere MST um neuen “billigsten” Knoten

Minimale Spannbäume: Lokale oder globale Strategie

Lokale Strategie: Algorithmus von Prim

- Startet bei einem Knoten
- Erweitere MST um neuen “billigsten” Knoten

Globale Strategie: Algorithmus von Kruskal

- Wähle die billigste Kante die keinen Kreis bildet

Minimale Spannbäume: Lokale oder globale Strategie

Lokale Strategie: Algorithmus von Prim

- Startet bei einem Knoten
- Erweitere MST um neuen “billigsten” Knoten

- Baut sich aus einer Ecke durch lokale Betrachtungen auf

Globale Strategie: Algorithmus von Kruskal

- Wähle die billigste Kante die keinen Kreis bildet

- Treffe global beste Entscheidung

Minimale Spannbäume: Lokale oder globale Strategie

Lokale Strategie: Algorithmus von Prim

- Startet bei einem Knoten
- Erweitere MST um neuen “billigsten” Knoten

Globale Strategie: Algorithmus von Kruskal

- Wähle die billigste Kante die keinen Kreis bildet

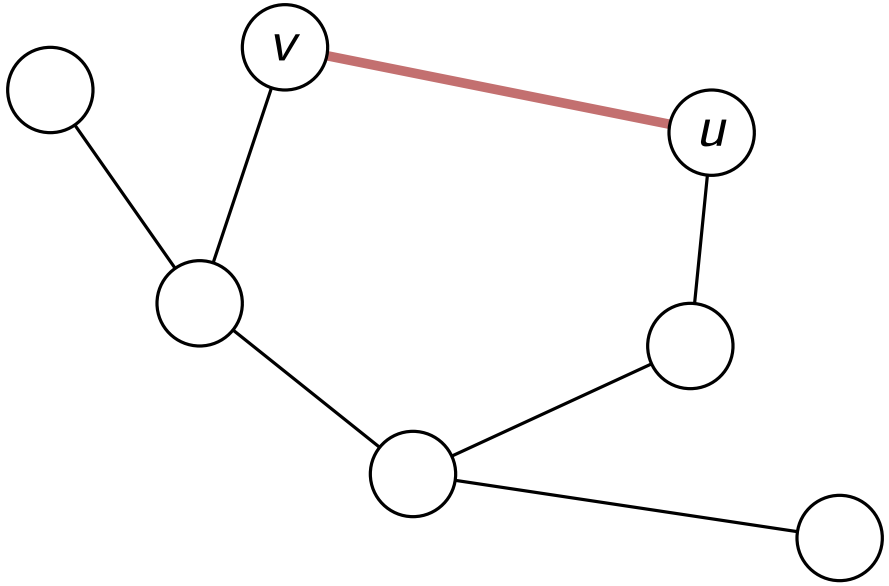
- Baut sich aus einer Ecke durch lokale Betrachtungen auf
- Treffe global beste Entscheidung

Für MSTs funktionieren die meisten Greedy-Strategien

Kreisfreiheit überprüfen

Globale Strategie:
Algorithmus von Kruskal

- Wähle die billigste Kante die keinen Kreis bildet



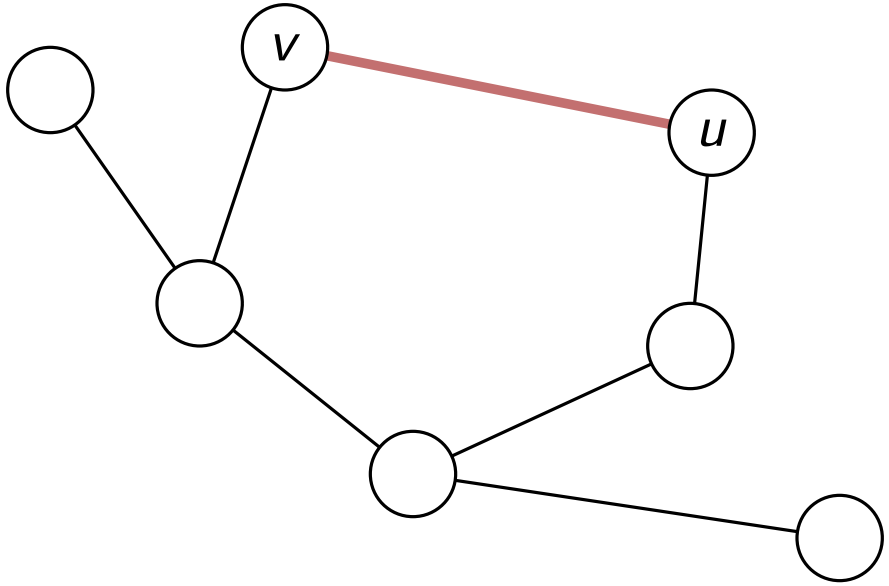
- Wie überprüft man, ob eine Kante $\{u, v\}$ einen Kreis schließen würde?

Kreisfreiheit überprüfen

Globale Strategie:
Algorithmus von Kruskal

- Wähle die billigste Kante die keinen Kreis bildet

- Wie überprüft man, ob eine Kante $\{u, v\}$ einen Kreis schließen würde?



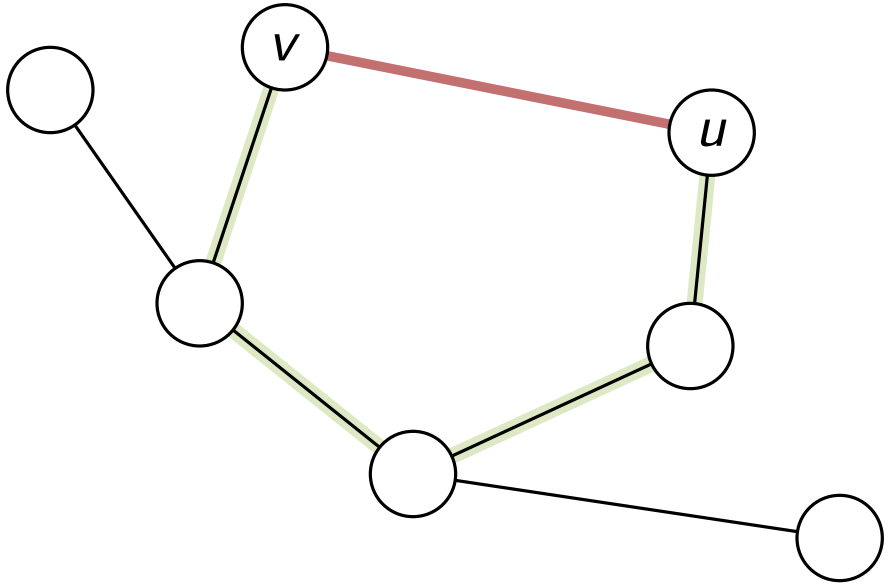
Beobachtung: Wenn $\{u, v\}$ einen Kreis schließen würde...

Kreisfreiheit überprüfen

Globale Strategie:
Algorithmus von Kruskal

- Wähle die billigste Kante die keinen Kreis bildet

- Wie überprüft man, ob eine Kante $\{u, v\}$ einen Kreis schließen würde?



Beobachtung: Wenn $\{u, v\}$ einen Kreis schließen würde...

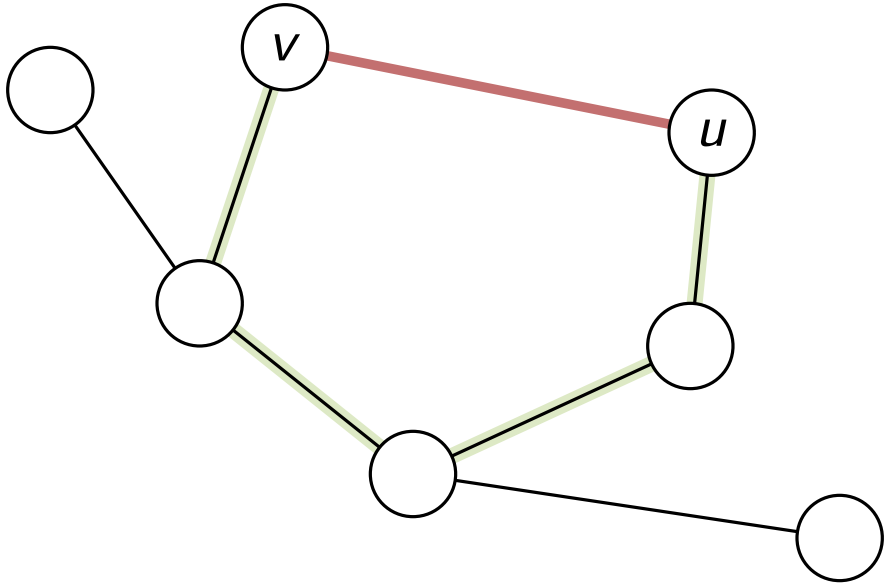
- ... dann gäbe es schon einen $u - v$-Pfad ohne $\{u, v\}$
- Also liegen u und v schon in der gleichen Zusammenhangskomponente

Kreisfreiheit überprüfen

Globale Strategie:
Algorithmus von Kruskal

- Wähle die billigste Kante die keinen Kreis bildet

- Wie überprüft man, ob eine Kante $\{u, v\}$ einen Kreis schließen würde?
  - Prüfe mit Union-Find, ob v und u in der gleichen Zusammenhangskomponente liegen



Beobachtung: Wenn $\{u, v\}$ einen Kreis schließen würde...

- ... dann gäbe es schon einen $u - v$-Pfad ohne $\{u, v\}$
- Also liegen u und v schon in der gleichen Zusammenhangskomponente

Algorithmus von Kruskal: Pseudocode

```
Algorithmus von Kruskal(Graph G)
    U := Union-Find with nodes of G
    PriorityQueue Q := empty PriorityQueue
    List L := empty List
    for (u, v) ∈ E do
        Q.push((u, v), len(e))
    while Q ≠ ∅ do
        (u, v) := Q.popMin()
        if U.find(v) ≠ U.find(u) do
            L.add((u, v))
            U.union(v, u)
```

Algorithmus von Kruskal: Pseudocode

```
Algorithmus von Kruskal(Graph G)
    U := Union-Find with nodes of G
    PriorityQueue Q := empty PriorityQueue
    List L := empty List
    for (u, v) ∈ E do
        Q.push((u, v), len(e))
    while Q ≠ ∅ do
        (u, v) := Q.popMin()
        if U.find(v) ≠ U.find(u) do
            L.add((u, v))
            U.union(v, u)
```

- Union-Find Datenstruktur zum Prüfen der Kreisfreiheit
- Aus Q bekommen wir immer die kleinste nicht-geprüfte Kante
- In L speichern wir uns die Kanten für den MST

Algorithmus von Kruskal: Pseudocode

```
Algorithmus von Kruskal(Graph G)
    U := Union-Find with nodes of G
    PriorityQueue Q := empty PriorityQueue
    List L := empty List
    for (u, v) ∈ E do
        Q.push((u, v), len(e))
    while Q ≠ ∅ do
        (u, v) := Q.popMin()
        if U.find(v) ≠ U.find(u) do
            L.add((u, v))
            U.union(v, u)
```

- Am Anfang müssen wir alle Kanten in Q einfügen

Algorithmus von Kruskal: Pseudocode

```
Algorithmus von Kruskal(Graph G)
    U := Union-Find with nodes of G
    PriorityQueue Q := empty PriorityQueue
    List L := empty List
    for (u, v) ∈ E do
        Q.push((u, v), len(e))
    while Q ≠ ∅ do
        (u, v) := Q.popMin()
        if U.find(v) ≠ U.find(u) do
            L.add((u, v))
            U.union(v, u)
```

- Betrachte die kleinste ungeprüfte Kante
- Falls sie keinen Kreis, schließt, füge sie in den MST ein und füge die Zusammenhangskomponenten zusammen

Algorithmus von Kruskal: Laufzeit

```
Algorithmus von Kruskal(Graph G)
    U := Union-Find with nodes of G
    PriorityQueue Q := empty PriorityQueue
    List L := empty List
    for (u, v) ∈ E do
        Q.push((u, v), len(e))
    while Q ≠ ∅ do
        (u, v) := Q.popMin()
        if U.find(v) ≠ U.find(u) do
            L.add((u, v))
            U.union(v, u)
```

Welche Laufzeit hat dieser Algorithmus?

Algorithmus von Kruskal: Laufzeit

Algorithmus von Kruskal(*Graph* G)

```
U := Union-Find with nodes of G
PriorityQueue Q := empty PriorityQueue
List L := empty List
for (u, v) ∈ E do
    Q.push((u, v), len(e))
while Q ≠ ∅ do
    (u, v) := Q.popMin()
    if U.find(v) ≠ U.find(u) do
        L.add((u, v))
        U.union(v, u)
```

Welche Laufzeit hat dieser Algorithmus?

- Erstellen der Union-Find Datenstruktur in $\Theta(n)$
- Erstellen der anderen Datenstrukturen in $\Theta(1)$

Algorithmus von Kruskal: Laufzeit

Algorithmus von Kruskal(*Graph* G)

```
U := Union-Find with nodes of G
PriorityQueue Q := empty PriorityQueue
List L := empty List
for (u, v) ∈ E do
    Q.push((u, v), len(e))
while Q ≠ ∅ do
    (u, v) := Q.popMin()
    if U.find(v) ≠ U.find(u) do
        L.add((u, v))
        U.union(v, u)
```

Welche Laufzeit hat dieser Algorithmus?

- Erstellen der Union-Find Datenstruktur in $\Theta(n)$
- Erstellen der anderen Datenstrukturen in $\Theta(1)$
- Mit **buildHeap** werden die m Kanten in $\Theta(m)$ eingefügt

Algorithmus von Kruskal: Laufzeit

Algorithmus von Kruskal(*Graph* G)

```
U := Union-Find with nodes of G
PriorityQueue Q := empty PriorityQueue
List L := empty List
for (u, v) ∈ E do
    Q.push((u, v), len(e))
while Q ≠ ∅ do
    (u, v) := Q.popMin()
    if U.find(v) ≠ U.find(u) do
        L.add((u, v))
        U.union(v, u)
```

Welche Laufzeit hat dieser Algorithmus?

- Erstellen der Union-Find Datenstruktur in $\Theta(n)$
- Erstellen der anderen Datenstrukturen in $\Theta(1)$
- Mit **buildHeap** werden die m Kanten in $\Theta(m)$ eingefügt
- Jede der m Kanten wird maximal ein mal angeguckt

Algorithmus von Kruskal: Laufzeit

Algorithmus von Kruskal(*Graph* G)

```
U := Union-Find with nodes of G
PriorityQueue Q := empty PriorityQueue
List L := empty List
for (u, v) ∈ E do
    Q.push((u, v), len(e))
while Q ≠ ∅ do
    (u, v) := Q.popMin()
    if U.find(v) ≠ U.find(u) do
        L.add((u, v))
        U.union(v, u)
```

Welche Laufzeit hat dieser Algorithmus?

- Erstellen der Union-Find Datenstruktur in $\Theta(n)$
- Erstellen der anderen Datenstrukturen in $\Theta(1)$
- Mit **buildHeap** werden die m Kanten in $\Theta(m)$ eingefügt
- Jede der m Kanten wird maximal ein mal angeguckt
- Für jede Kante gibt es maximal 2 **find** und 1 **union** Operation

Algorithmus von Kruskal: Laufzeit

Algorithmus von Kruskal(*Graph* G)

```
U := Union-Find with nodes of G
PriorityQueue Q := empty PriorityQueue
List L := empty List
for (u, v) ∈ E do
    Q.push((u, v), len(e))
while Q ≠ ∅ do
    (u, v) := Q.popMin()
    if U.find(v) ≠ U.find(u) do
        L.add((u, v))
        U.union(v, u)
```

Welche Laufzeit hat dieser Algorithmus?

- Erstellen der Union-Find Datenstruktur in $\Theta(n)$
- Erstellen der anderen Datenstrukturen in $\Theta(1)$
- Mit **buildHeap** werden die m Kanten in $\Theta(m)$ eingefügt
- Jede der m Kanten wird maximal ein mal angeguckt
- Für jede Kante gibt es maximal 2 **find** und 1 **union** Operation

Insgesamt $O(n + 1 + m + m \log^*(n))$

Algorithmus von Kruskal: Laufzeit

Algorithmus von Kruskal(*Graph* G)

```
U := Union-Find with nodes of G
PriorityQueue Q := empty PriorityQueue
List L := empty List
for (u, v) ∈ E do
    Q.push((u, v), len(e))
while Q ≠ ∅ do
    (u, v) := Q.popMin()
    if U.find(v) ≠ U.find(u) do
        L.add((u, v))
        U.union(v, u)
```

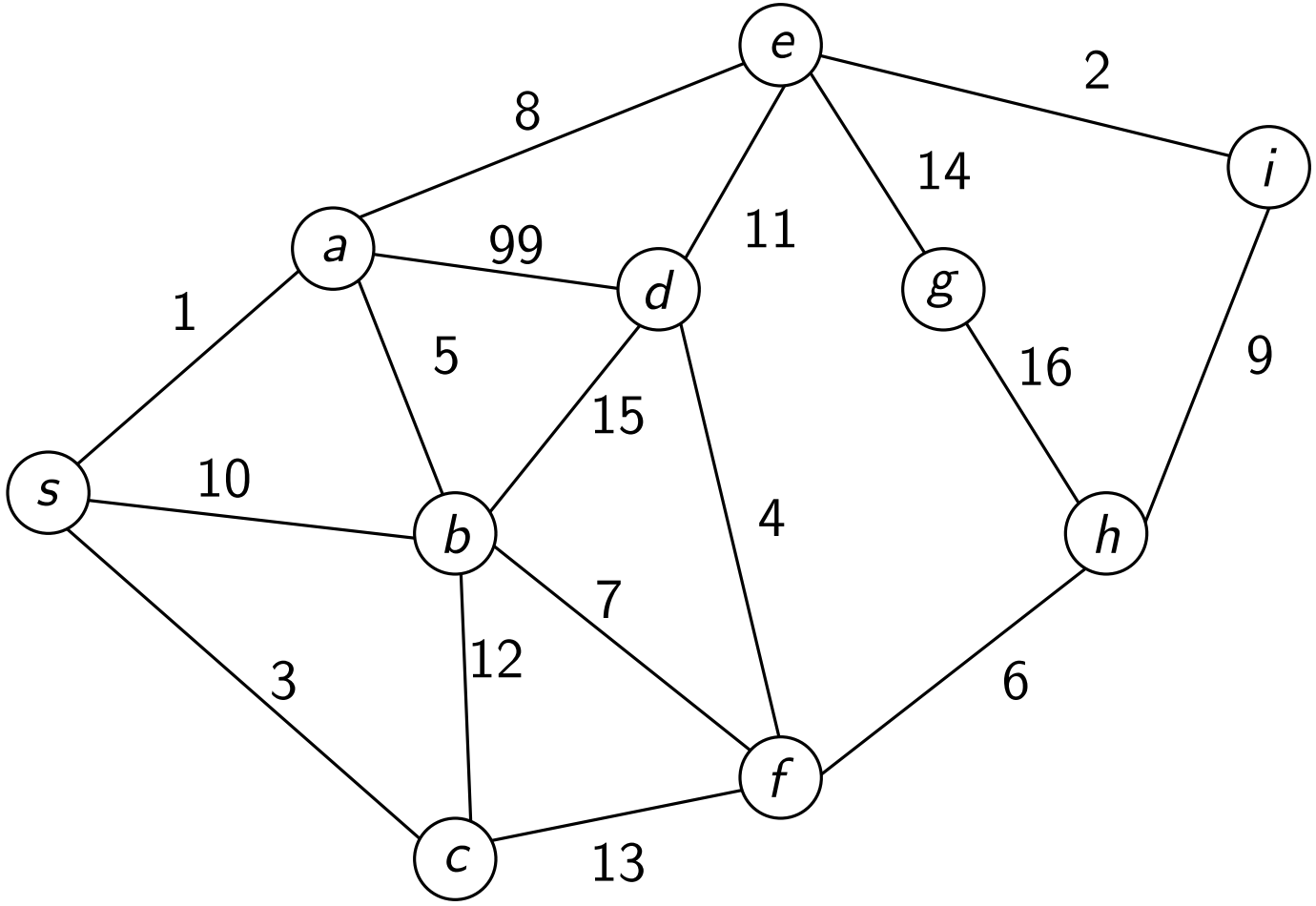
Welche Laufzeit hat dieser Algorithmus?

- Erstellen der Union-Find Datenstruktur in $\Theta(n)$
- Erstellen der anderen Datenstrukturen in $\Theta(1)$
- Mit **buildHeap** werden die m Kanten in $\Theta(m)$ eingefügt
- Jede der m Kanten wird maximal ein mal angeguckt
- Für jede Kante gibt es maximal 2 **find** und 1 **union** Operation

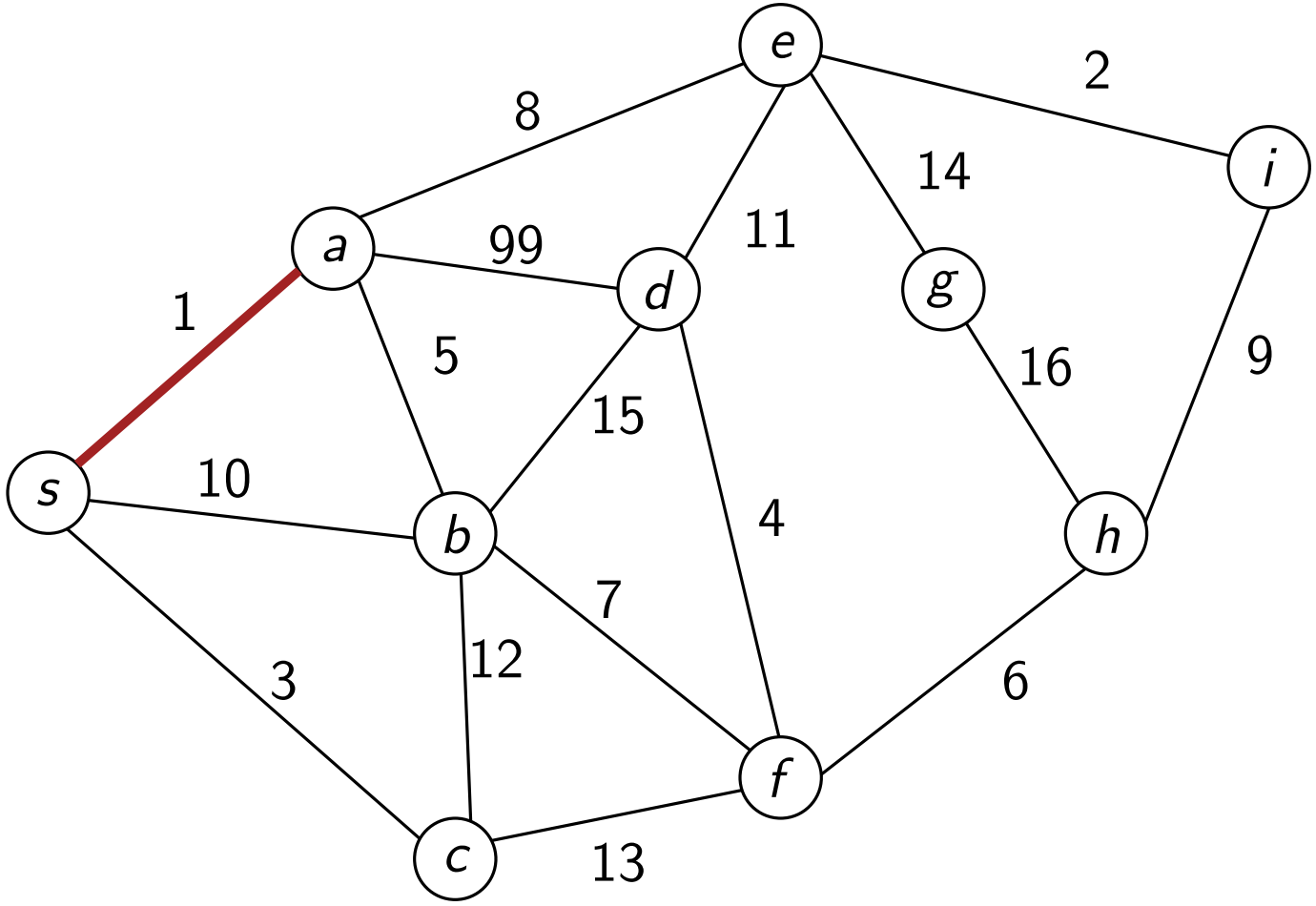
Insgesamt $O(n + 1 + m + m \log^*(n))$

$= O(m \log^*(n))$

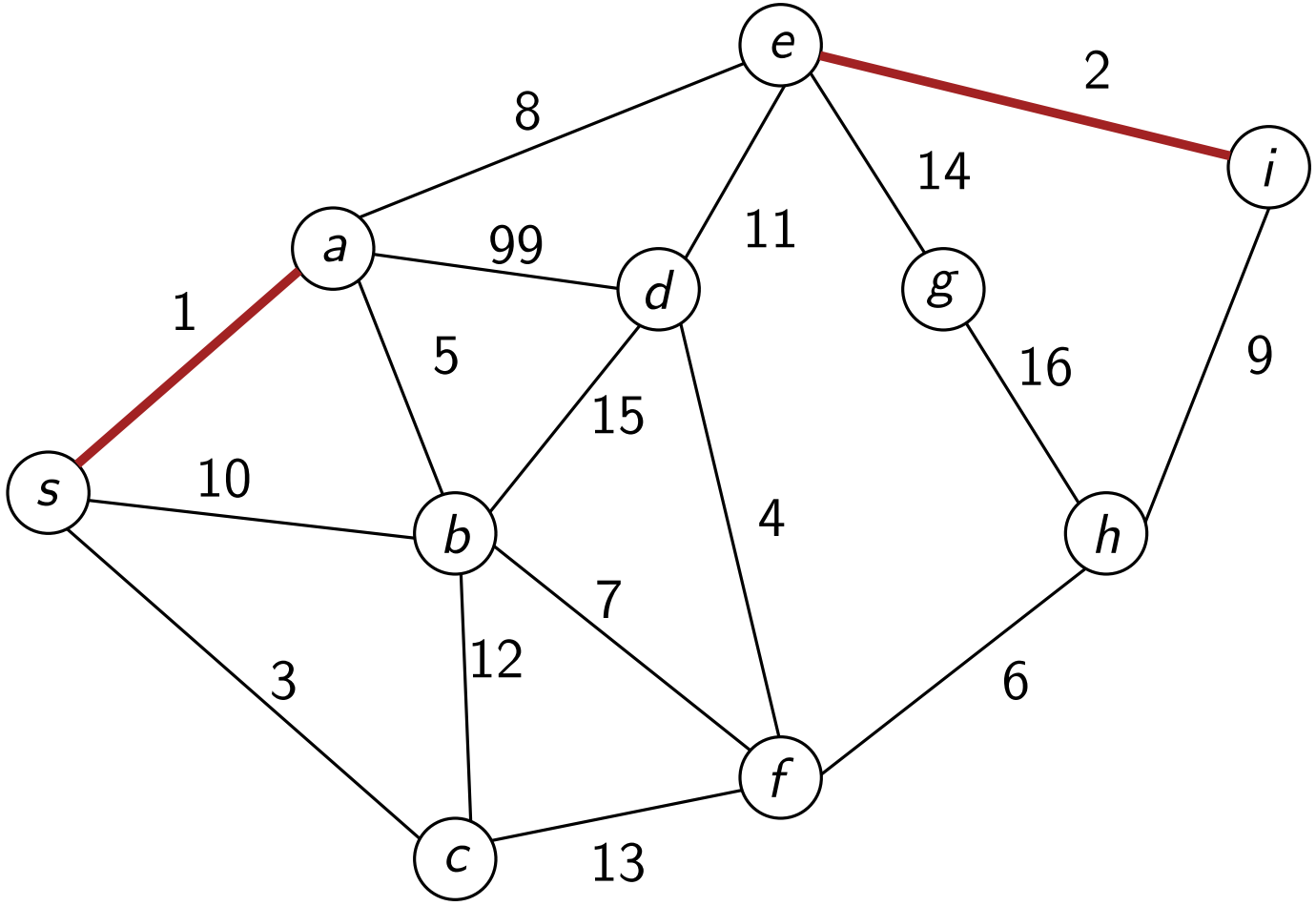
Algorithmus von Kruskal: Beispiel



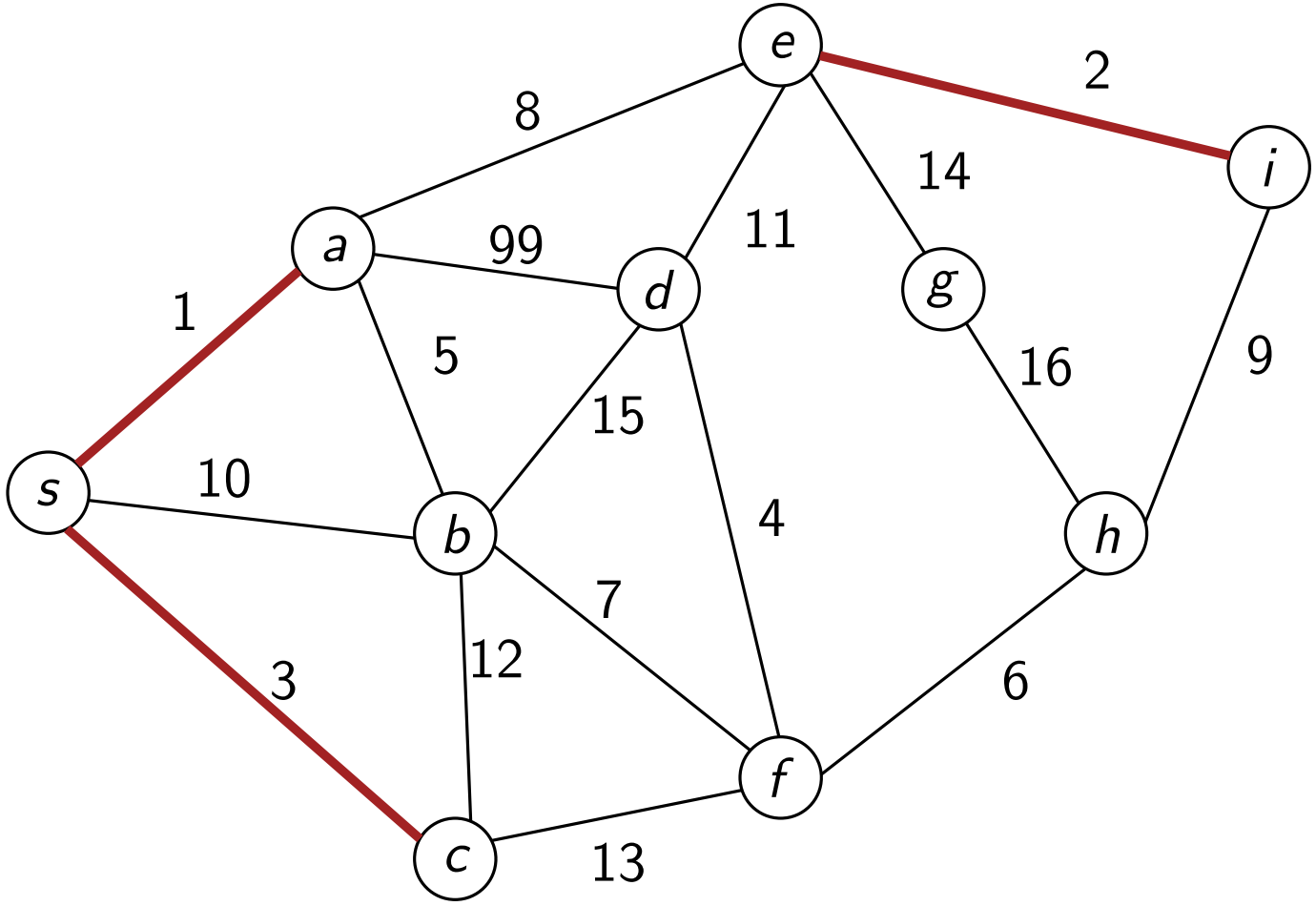
Algorithmus von Kruskal: Beispiel



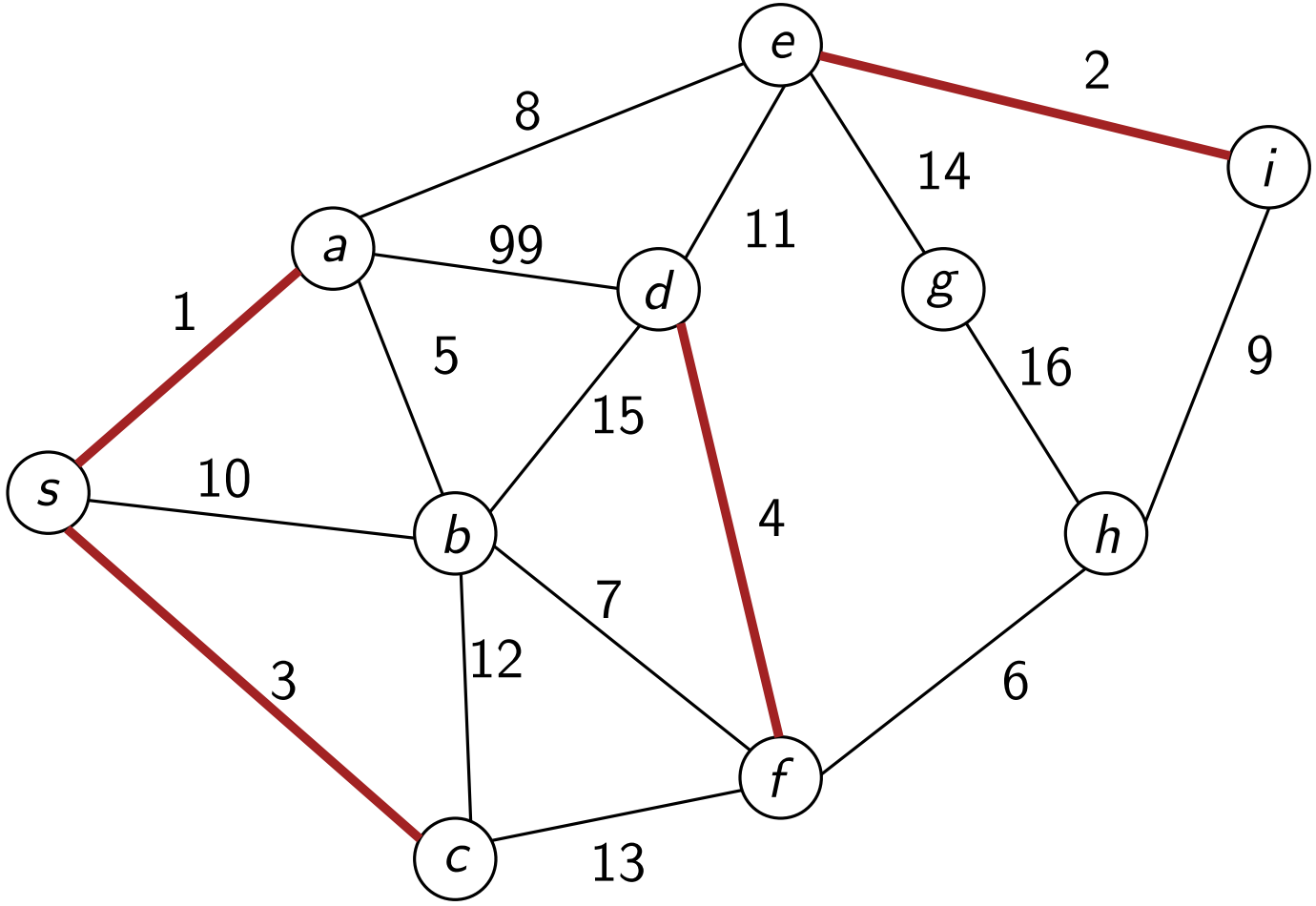
Algorithmus von Kruskal: Beispiel



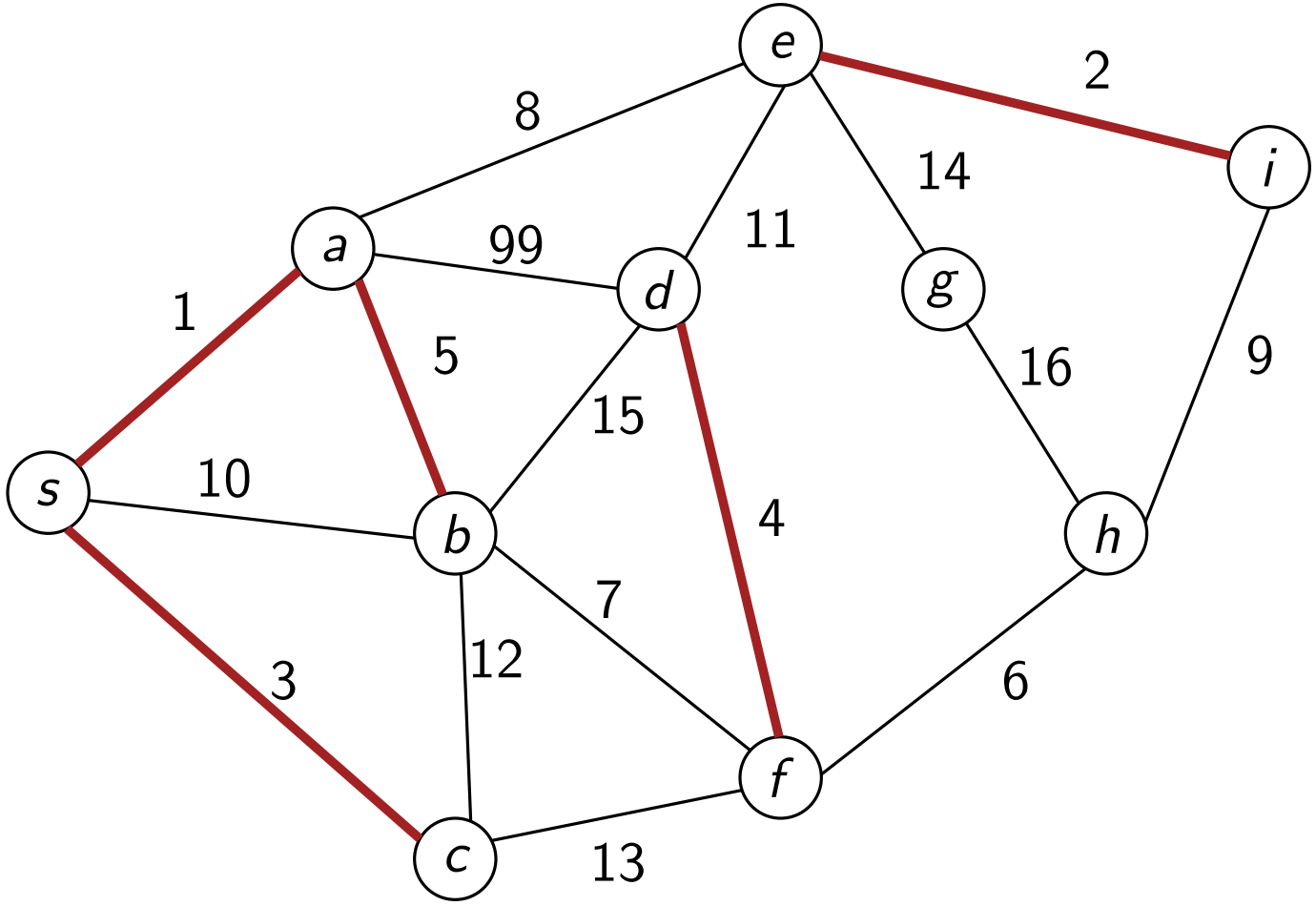
Algorithmus von Kruskal: Beispiel



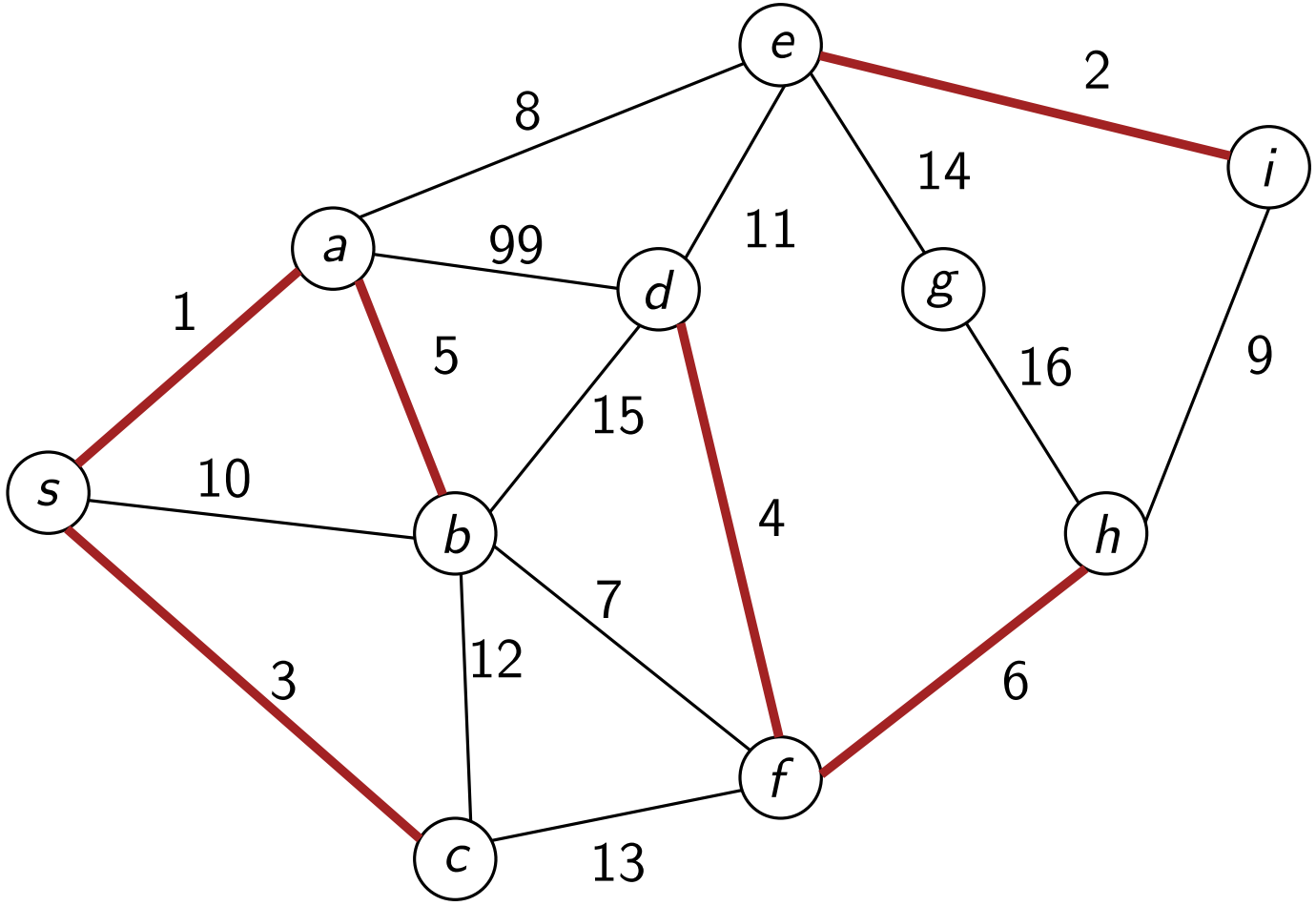
Algorithmus von Kruskal: Beispiel



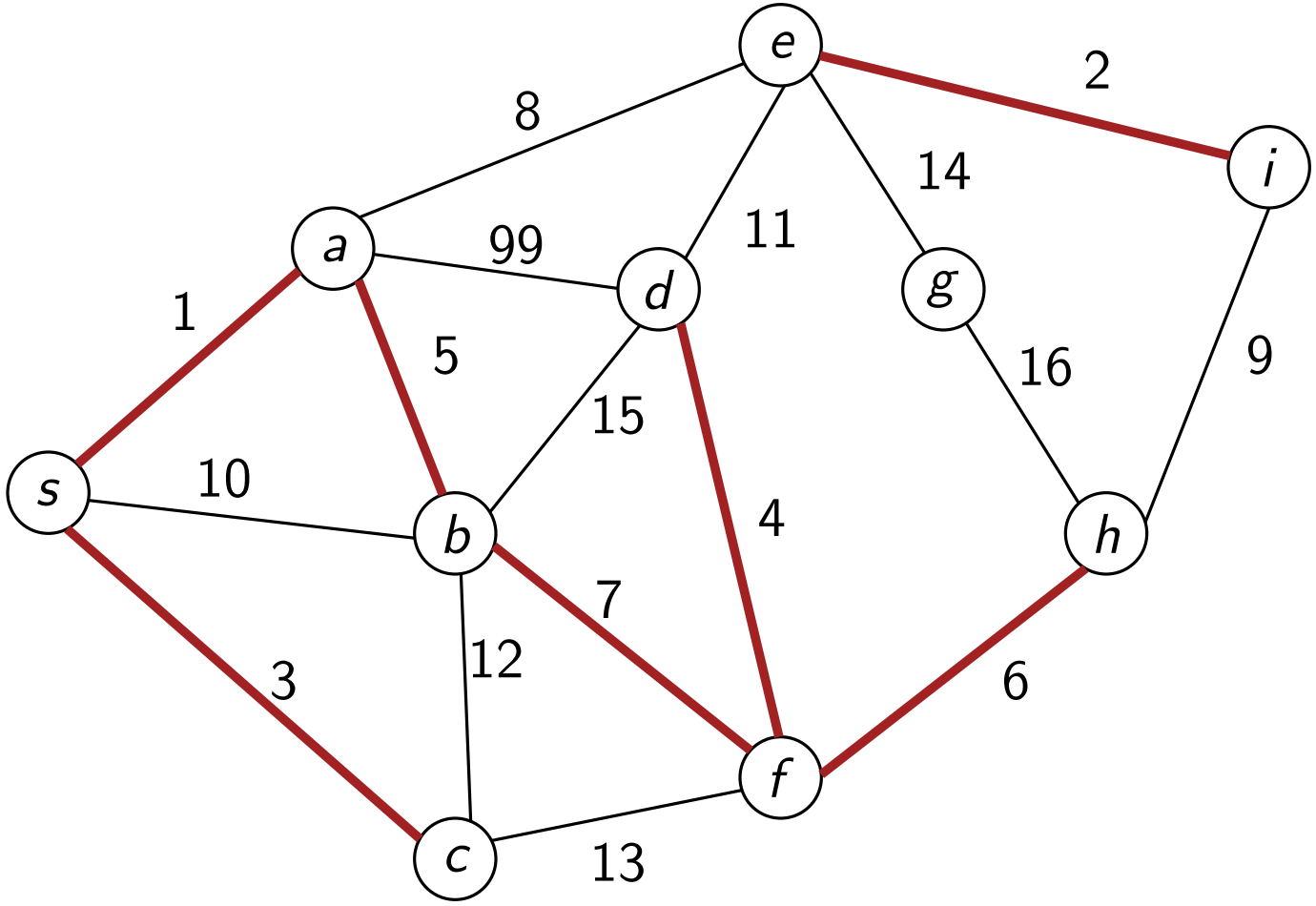
Algorithmus von Kruskal: Beispiel



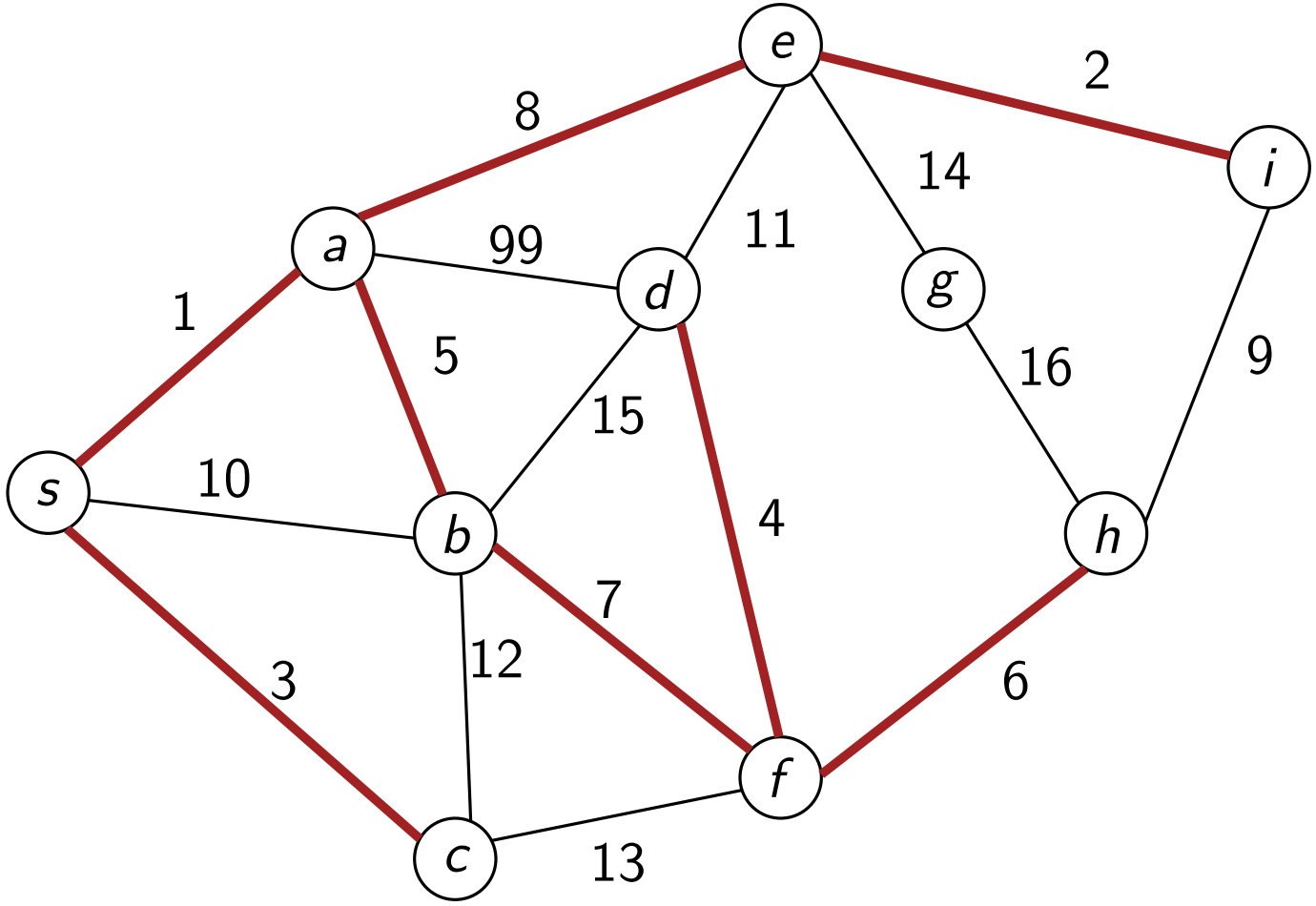
Algorithmus von Kruskal: Beispiel



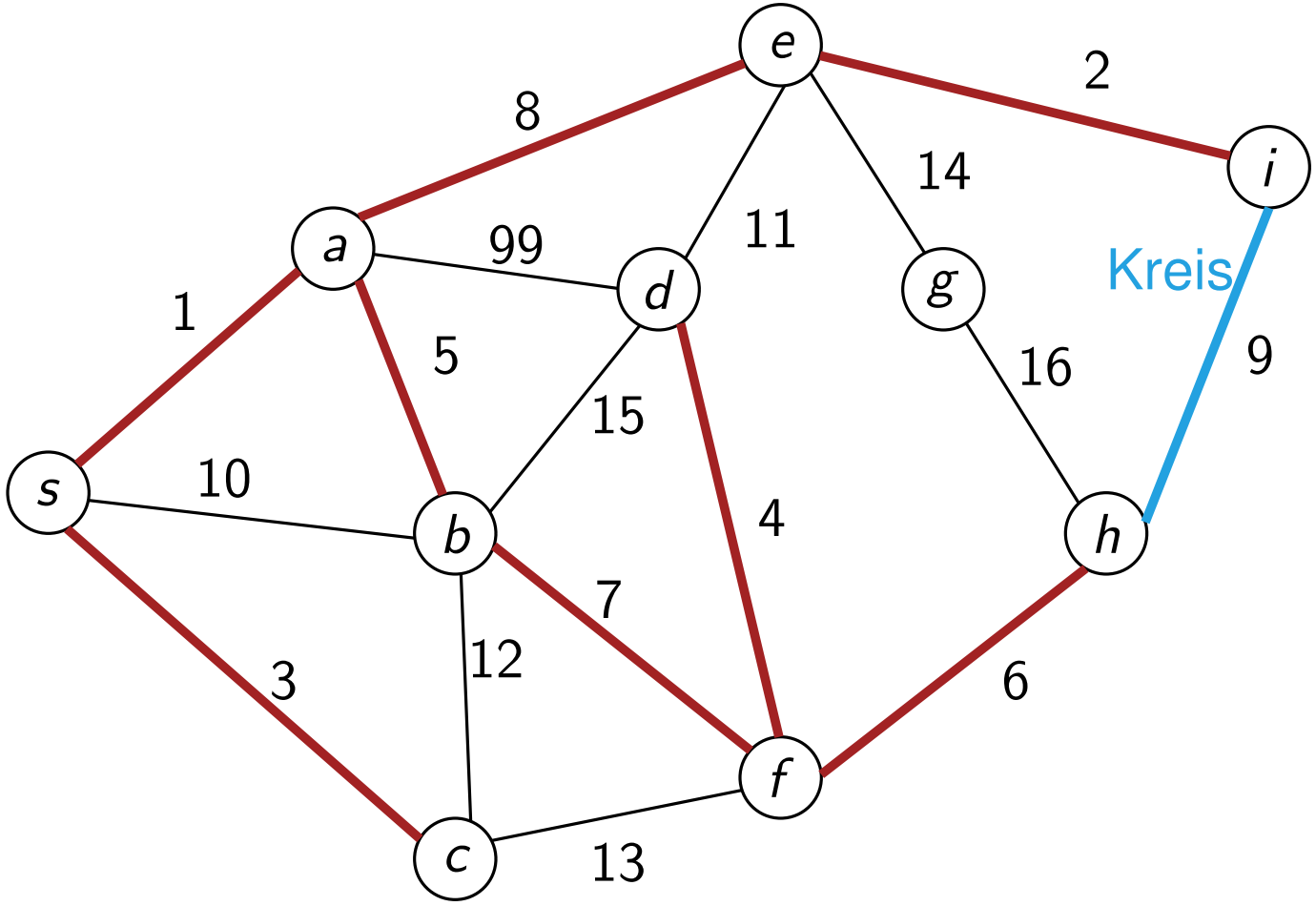
Algorithmus von Kruskal: Beispiel



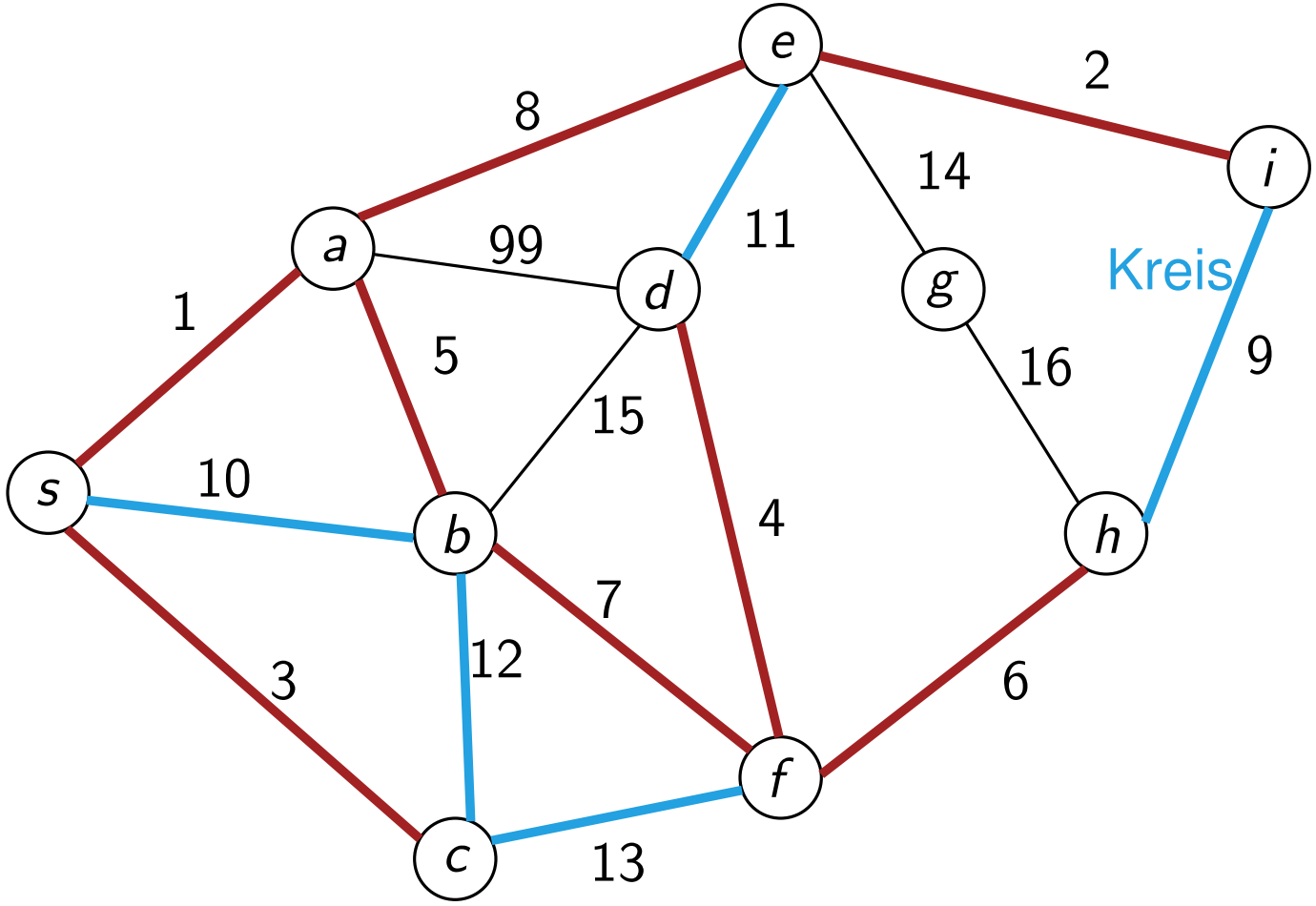
Algorithmus von Kruskal: Beispiel



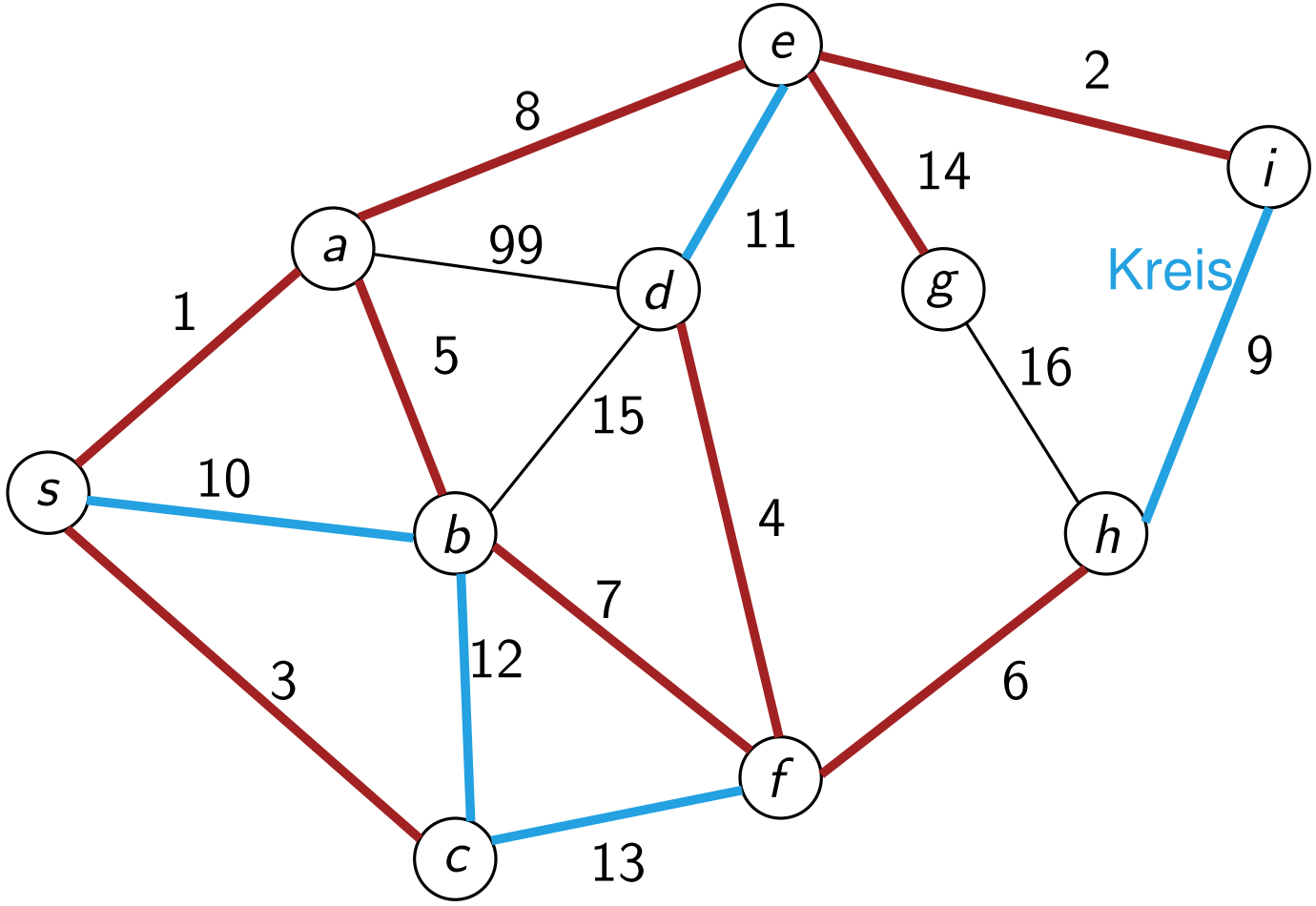
Algorithmus von Kruskal: Beispiel



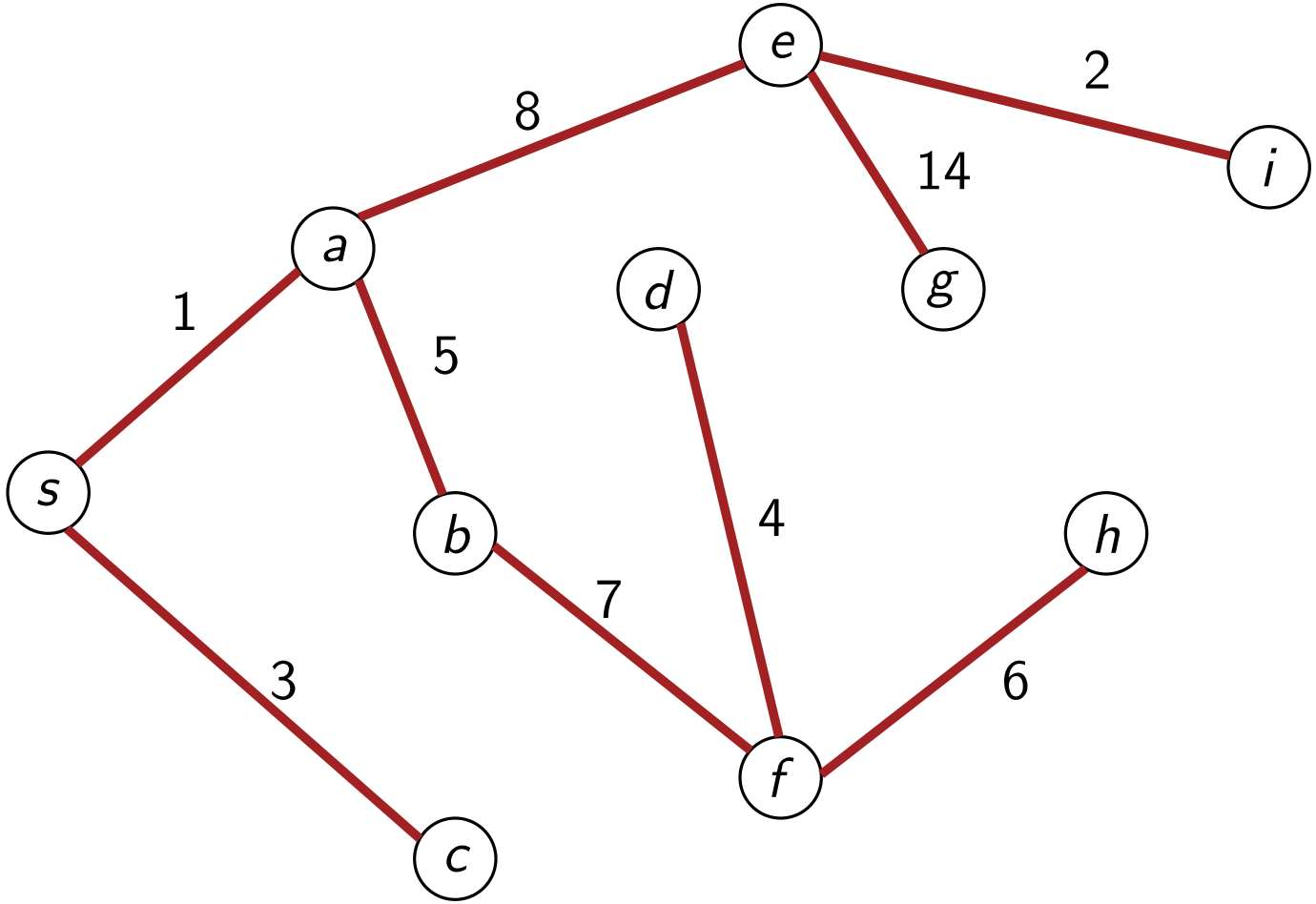
Algorithmus von Kruskal: Beispiel

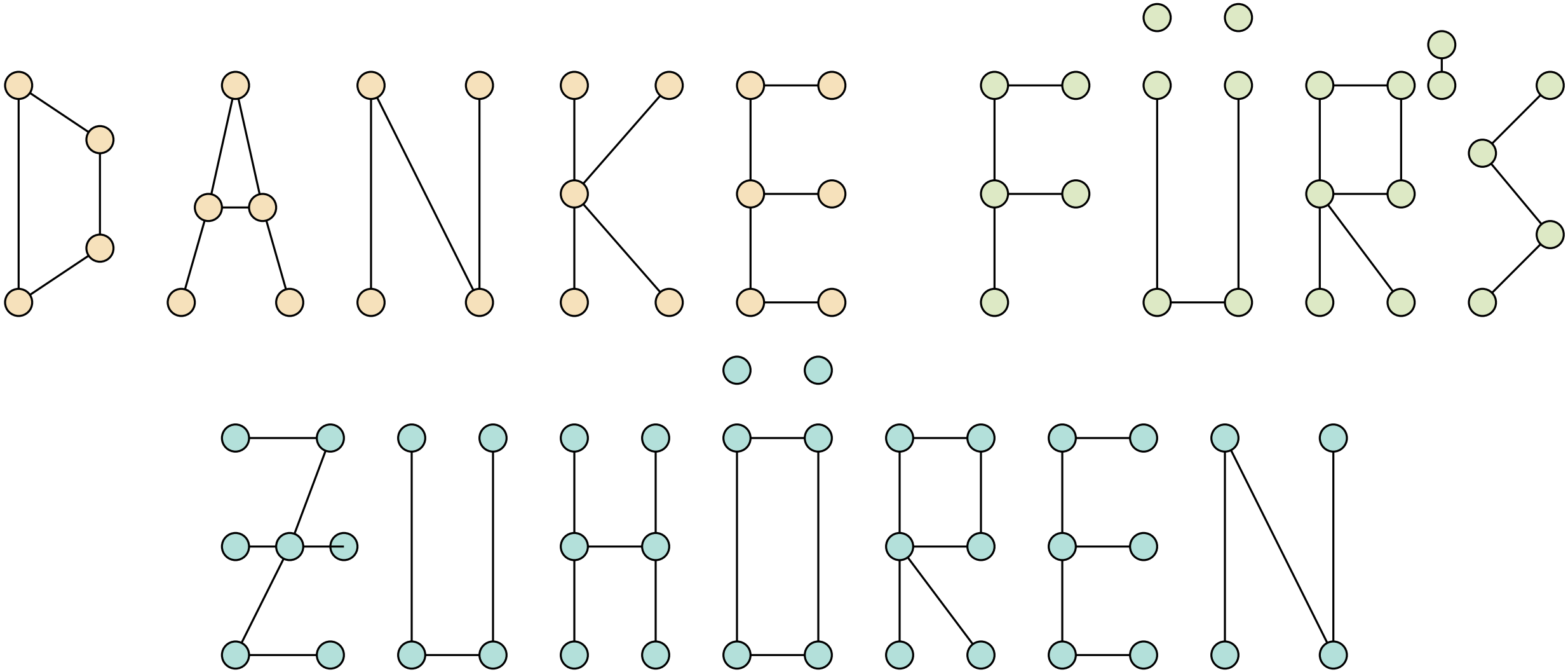


Algorithmus von Kruskal: Beispiel



Algorithmus von Kruskal: Beispiel




DANKE FÜRS
ZUHÖREN