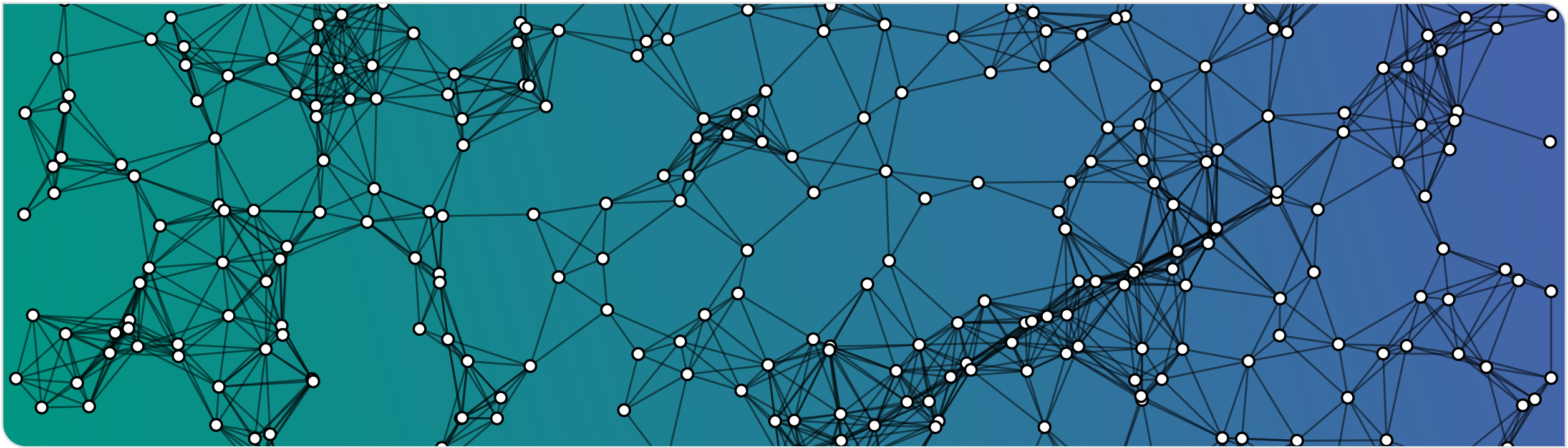
Karlsruher Institut für Technologie

Algorithmen Zusatztutorium Woche 4

Graphen, kürzeste Wege und minimale Spannbäume



Diese Woche

- Graphen und Begrifflichkeiten

Diese Woche

- Graphen und Begrifflichkeiten
- kürzeste Wege
  - Warum ist das interessant?
  - Problemversionen und Algorithmen
    - BFS, Dijkstra, Bellman-Ford, Floyd-Warshall

Diese Woche

- Graphen und Begrifflichkeiten
- kürzeste Wege
  - Warum ist das interessant?
  - Problemversionen und Algorithmen
    - BFS, Dijkstra, Bellman-Ford, Floyd-Warshall
- minimale Spannbäume
  - Warum ist das interessant?
  - Wie finden wir einen minimalen Spannbaum?

Graphen 1

- Graph $G = (V, E)$ besteht aus:

Graphen 1

- Graph $G = (V, E)$ besteht aus:
  - Knotenmenge V, z.B Städte, Kreuzungen, Zustände

Graphen 1

- Graph $G = (V, E)$ besteht aus:
  - Knotenmenge V, z.B Städte, Kreuzungen, Zustände
  - Kantenmenge E, z.B Straßen zwischen Städten, Kreuzungen, Übergänge zwischen Zuständen
    - Kante ist Verbindung zwischen zwei Knoten

Graphen 1

- Graph $G = (V, E)$ besteht aus:
  - Knotenmenge V, z.B Städte, Kreuzungen, Zustände
  - Kantenmenge E, z.B Straßen zwischen Städten, Kreuzungen, Übergänge zwischen Zuständen
    - Kante ist Verbindung zwischen zwei Knoten
- Graphen können gemalt werden, Knoten sind Punkte, Kanten sind Verbindungen zwischen den Punkten

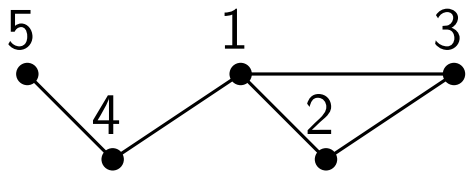
Graphen 1

- Graph $G = (V, E)$ besteht aus:
  - Knotenmenge V, z.B Städte, Kreuzungen, Zustände
  - Kantenmenge E, z.B Straßen zwischen Städten, Kreuzungen, Übergänge zwischen Zuständen
    - Kante ist Verbindung zwischen zwei Knoten
- Graphen können gemalt werden, Knoten sind Punkte, Kanten sind Verbindungen zwischen den Punkten

z.B:

$V = \{1, 2, 3, 4, 5\}$
$E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}\{2, 3\}, \{4, 5\}\}$

bildlich →
← formal



Graphen 2

- Kanten können gerichtet oder ungerichtet sein
  - gerichtet geht nur in eine Richtung, ungerichtet in beide
  - z.B bei Einbahnstraßen

Graphen 2

- Kanten können gerichtet oder ungerichtet sein
  - gerichtet geht nur in eine Richtung, ungerichtet in beide
  - z.B bei Einbahnstraßen
- Teilgraph ist Teilmenge der Knoten und Kanten

Graphen 2

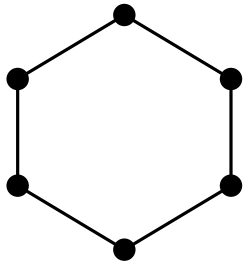
- Kanten können gerichtet oder ungerichtet sein
  - gerichtet geht nur in eine Richtung, ungerichtet in beide
  - z.B bei Einbahnstraßen
- Teilgraph ist Teilmenge der Knoten und Kanten

- Zusammenhangskomponente ist maximaler Teilgraph, sodass zwischen je zwei Knoten v, w ein Pfad existiert

Graphen 3

- Kreis ist Folge von Knoten $v_1, v_2, \cdots v_n$ und $v_i v_{i+1}$ und zusätzlich $v_n v_1$ Kante

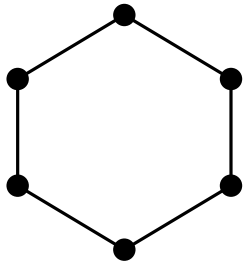
z.B.:



Graphen 3

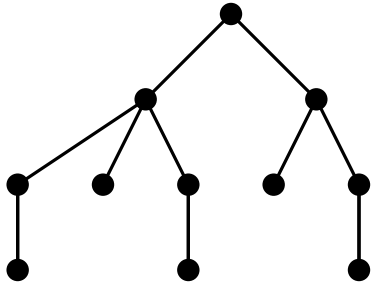
- Kreis ist Folge von Knoten $v_1, v_2, \cdots v_n$ und $v_i v_{i+1}$ und zusätzlich $v_n v_1$ Kante

z.B.:



- Ein Baum ist ein kreisfreier zusammenhängender Graph

z.B.:



Wege in Graphen

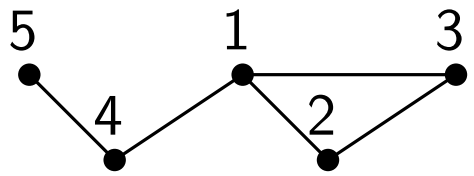
- Gegeben ein Graph G, ein Weg $W = (v_0, v_1, \cdots v_n)$ ist eine Folge von Knoten, sodass v_i Knoten in G sind und v_i, v_{i+1} eine Kante in G ist.

- Gegeben ein Graph G, ein Pfad P ist ein Weg, in dem kein Knoten zwei mal besucht wird

Wege in Graphen

- Gegeben ein Graph G, ein Weg $W = (v_0, v_1, \cdots v_n)$ ist eine Folge von Knoten, sodass v_i Knoten in G sind und v_i, v_{i+1} eine Kante in G ist.

- Gegeben ein Graph G, ein Pfad P ist ein Weg, in dem kein Knoten zwei mal besucht wird



z.B. $(1, 3, 2, 1, 4)$ ist ein Weg in diesem Graphen

z.B. $(3, 2, 1, 4, 5)$ ist ein Pfad in diesem Graph

kürzeste Wege

- Graphen können viele Sachverhalte modelieren
  - z.B. Straßennetze, mögliche Zustände in einem System, etc.

kürzeste Wege

- Graphen können viele Sachverhalte modelieren
  - z.B. Straßennetze, mögliche Zustände in einem System, etc.
- Probleme der Sachverhalte sind auf Probleme der Graphen zurückführbar
  - optimales Layout zum Ausbau von Glasfaser $=$ Minimaler Spannbaum

kürzeste Wege

- Graphen können viele Sachverhalte modelieren
  - z.B. Straßennetze, mögliche Zustände in einem System, etc.
- Probleme der Sachverhalte sind auf Probleme der Graphen zurückführbar
  - optimales Layout zum Ausbau von Glasfaser $=$ Minimaler Spannbaum
  - schnellste Route von Karlsruhe nach Berlin $=$ kürzester Weg in Straßengraph

kürzeste Wege

- Graphen können viele Sachverhalte modelieren
  - z.B. Straßennetze, mögliche Zustände in einem System, etc.
- Probleme der Sachverhalte sind auf Probleme der Graphen zurückführbar
  - optimales Layout zum Ausbau von Glasfaser $=$ Minimaler Spannbaum
  - schnellste Route von Karlsruhe nach Berlin $=$ kürzester Weg in Straßengraph
- Im Folgenden betrachten wir kürzeste-Wege-Probleme
  - Möglichst schnelle Beantwortung von Anfragen: Startknoten x, Zielknoten y, wie komme ich am schnellsten von x nach y ?

Das Kohlkopfproblem 1

- Sie sind Farmer und haben eine Ziege, einen Wolf und ein Kohlkopf

Das Kohlkopfproblem 1

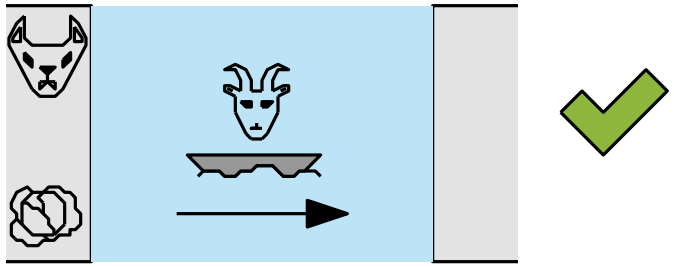
- Sie sind Farmer und haben eine Ziege, einen Wolf und ein Kohlkopf
- Sie möchten den Fluss überqueren, haben jedoch nur ein kleines Boot
  - Sie müssen immer mitfahren um das Boot zu lenken, sonst können Sie noch einen weiteren Mitfahrer mitnehmen

Das Kohlkopfproblem 1

- Sie sind Farmer und haben eine Ziege, einen Wolf und ein Kohlkopf
- Sie möchten den Fluss überqueren, haben jedoch nur ein kleines Boot
  - Sie müssen immer mitfahren um das Boot zu lenken, sonst können Sie noch einen weiteren Mitfahrer mitnehmen
- Falls Sie den Wolf und die Ziege alleine am Ufer lassen, frisst der Wolf die Ziege
  - Ebenso frisst die Ziege den Kohlkopf, falls alleine gelassen

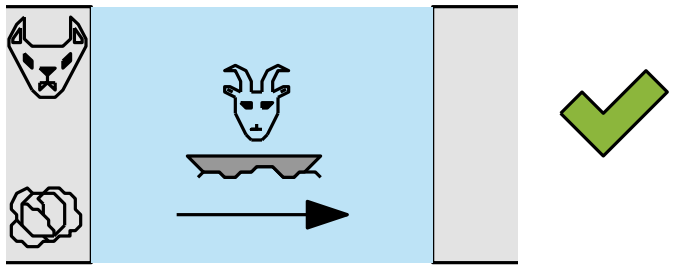
Das Kohlkopfproblem 1

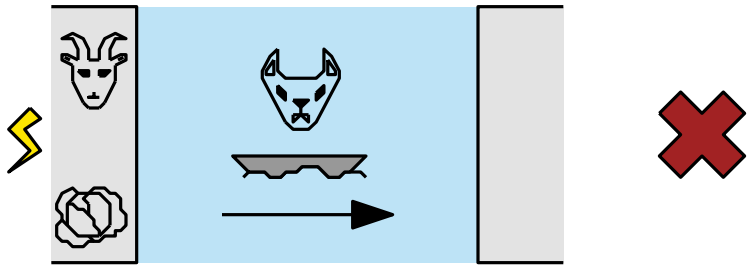
- Sie sind Farmer und haben eine Ziege, einen Wolf und ein Kohlkopf
- Sie möchten den Fluss überqueren, haben jedoch nur ein kleines Boot
  - Sie müssen immer mitfahren um das Boot zu lenken, sonst können Sie noch einen weiteren Mitfahrer mitnehmen
- Falls Sie den Wolf und die Ziege alleine am Ufer lassen, frisst der Wolf die Ziege
  - Ebenso frisst die Ziege den Kohlkopf, falls alleine gelassen



Das Kohlkopfproblem 1

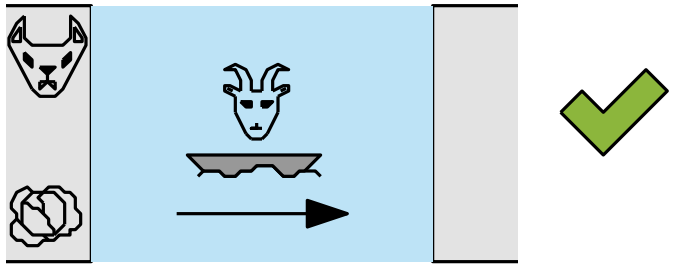
- Sie sind Farmer und haben eine Ziege, einen Wolf und ein Kohlkopf
- Sie möchten den Fluss überqueren, haben jedoch nur ein kleines Boot
  - Sie müssen immer mitfahren um das Boot zu lenken, sonst können Sie noch einen weiteren Mitfahrer mitnehmen
- Falls Sie den Wolf und die Ziege alleine am Ufer lassen, frisst der Wolf die Ziege
  - Ebenso frisst die Ziege den Kohlkopf, falls alleine gelassen

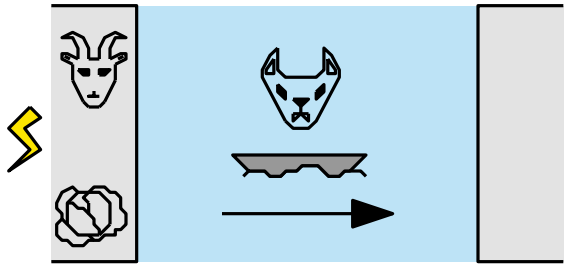




Das Kohlkopfproblem 1

- Sie sind Farmer und haben eine Ziege, einen Wolf und ein Kohlkopf
- Sie möchten den Fluss überqueren, haben jedoch nur ein kleines Boot
  - Sie müssen immer mitfahren um das Boot zu lenken, sonst können Sie noch einen weiteren Mitfahrer mitnehmen
- Falls Sie den Wolf und die Ziege alleine am Ufer lassen, frisst der Wolf die Ziege
  - Ebenso frisst die Ziege den Kohlkopf, falls alleine gelassen
- Gibt es Abfolge von Flussüberquerungen, sodass alle drei Mitfahrer heil auf der anderen Seite ankommen?





Das Kohlkopfproblem 2

- Kohlkopf-Problem kann man auch als Graph modellieren!

Das Kohlkopfproblem 2

- Kohlkopf-Problem kann man auch als Graph modellieren!
  - Knoten sind Zustände, also was auf welcher Seite ist und wohin das Boot gerade fährt

Das Kohlkopfproblem 2

- Kohlkopf-Problem kann man auch als Graph modellieren!
  - Knoten sind Zustände, also was auf welcher Seite ist und wohin das Boot gerade fährt
  - Kante zwischen zwei Zuständen, falls man anderen Zustand durch eine gültige Fluss-überquerung erreichen kann

Das Kohlkopfproblem 2

- Kohlkopf-Problem kann man auch als Graph modellieren!
  - Knoten sind Zustände, also was auf welcher Seite ist und wohin das Boot gerade fährt
  - Kante zwischen zwei Zuständen, falls man anderen Zustand durch eine gültige Flussüberquerung erreichen kann
- Welche Folge von Überquerungen führt am Schnellsten zum Ziel?

Das Kohlkopfproblem 2

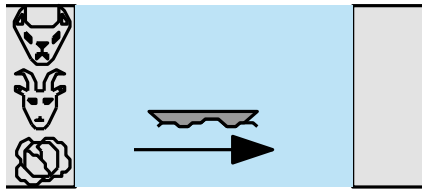
- Kohlkopf-Problem kann man auch als Graph modellieren!
  - Knoten sind Zustände, also was auf welcher Seite ist und wohin das Boot gerade fährt
  - Kante zwischen zwei Zuständen, falls man anderen Zustand durch eine gültige Flussüberquerung erreichen kann
- Welche Folge von Überquerungen führt am Schnellsten zum Ziel?

$\Longrightarrow$ kürzester Weg zwischen Start und Ende

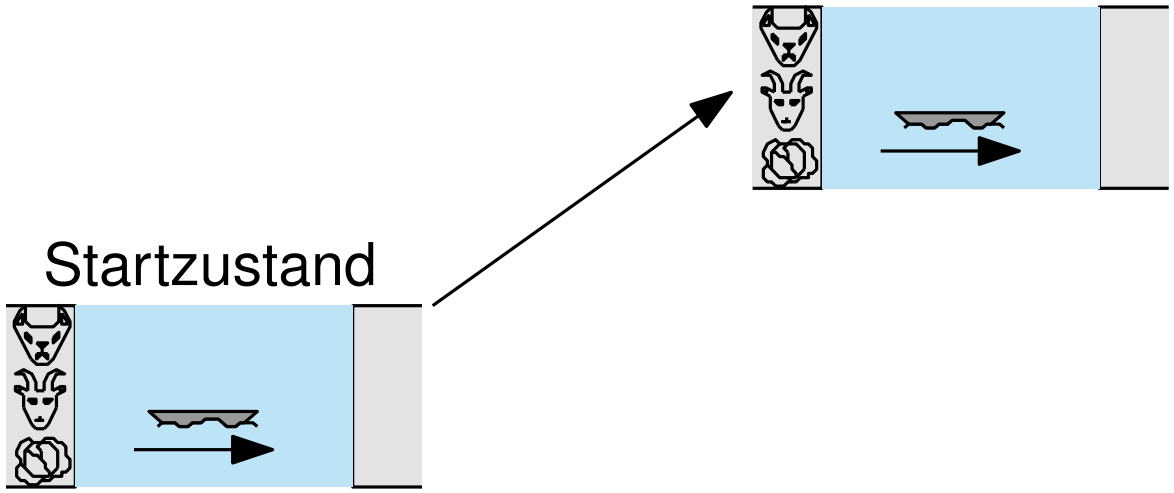
  - Startknoten: Alle Mitfahrer sind auf linker Seite
  - Zielknoten: Alle Mitfahrer sind auf rechter Seite

Das Kohlkopfproblem 3

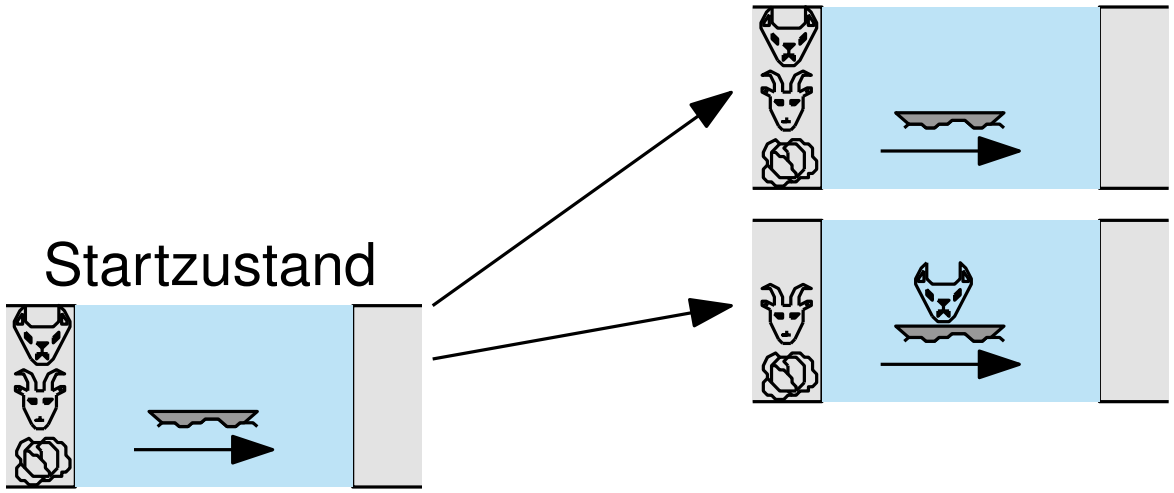
Startzustand



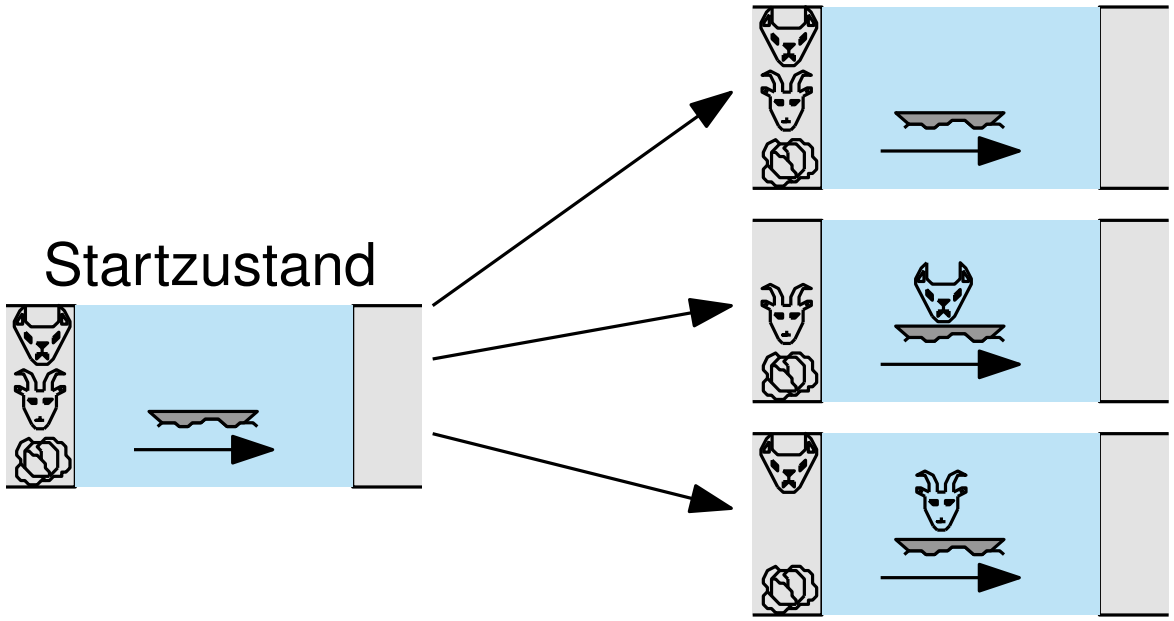
Das Kohlkopfproblem 3



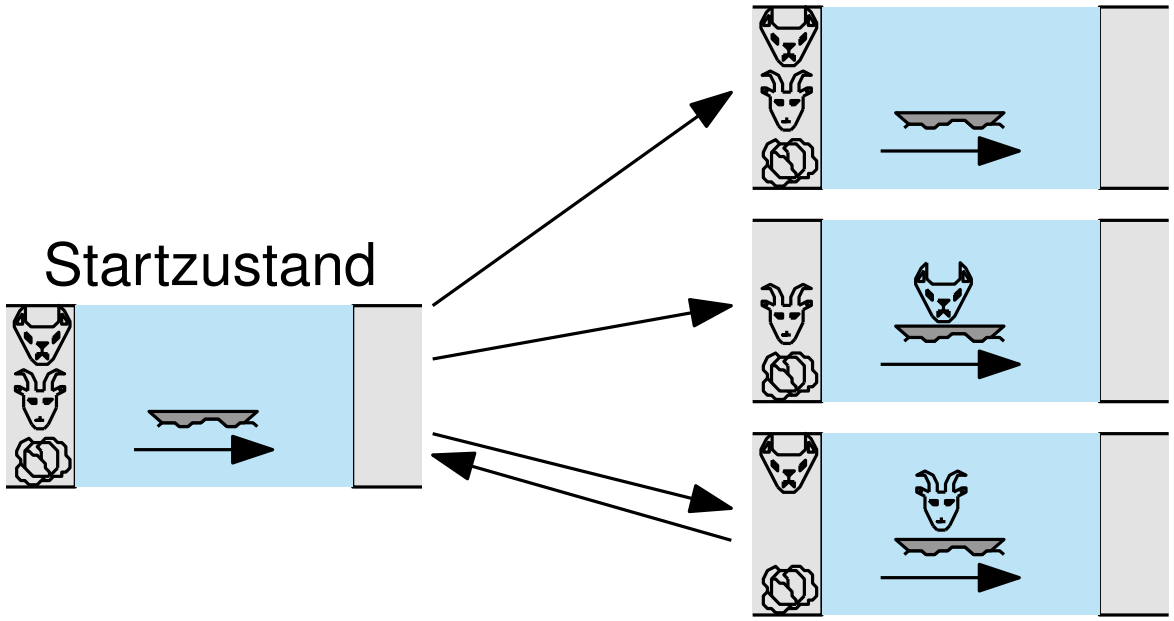
Das Kohlkopfproblem 3



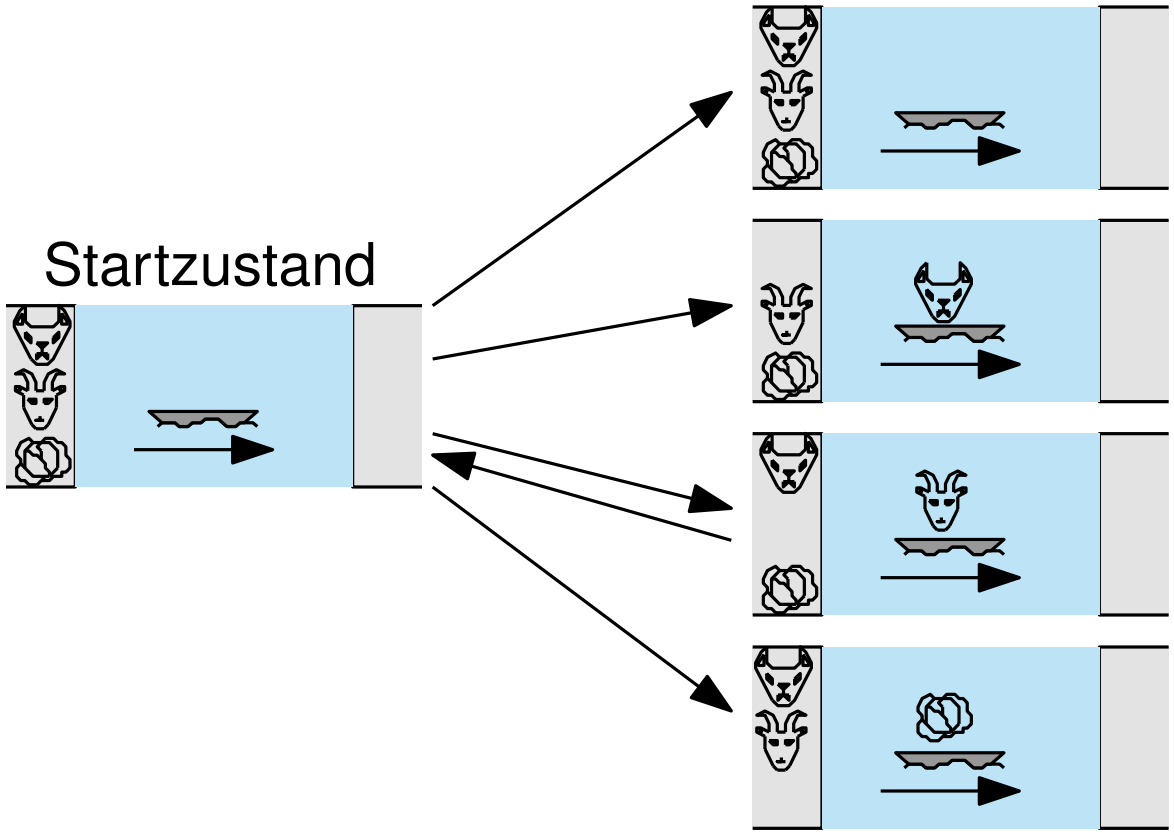
Das Kohlkopfproblem 3



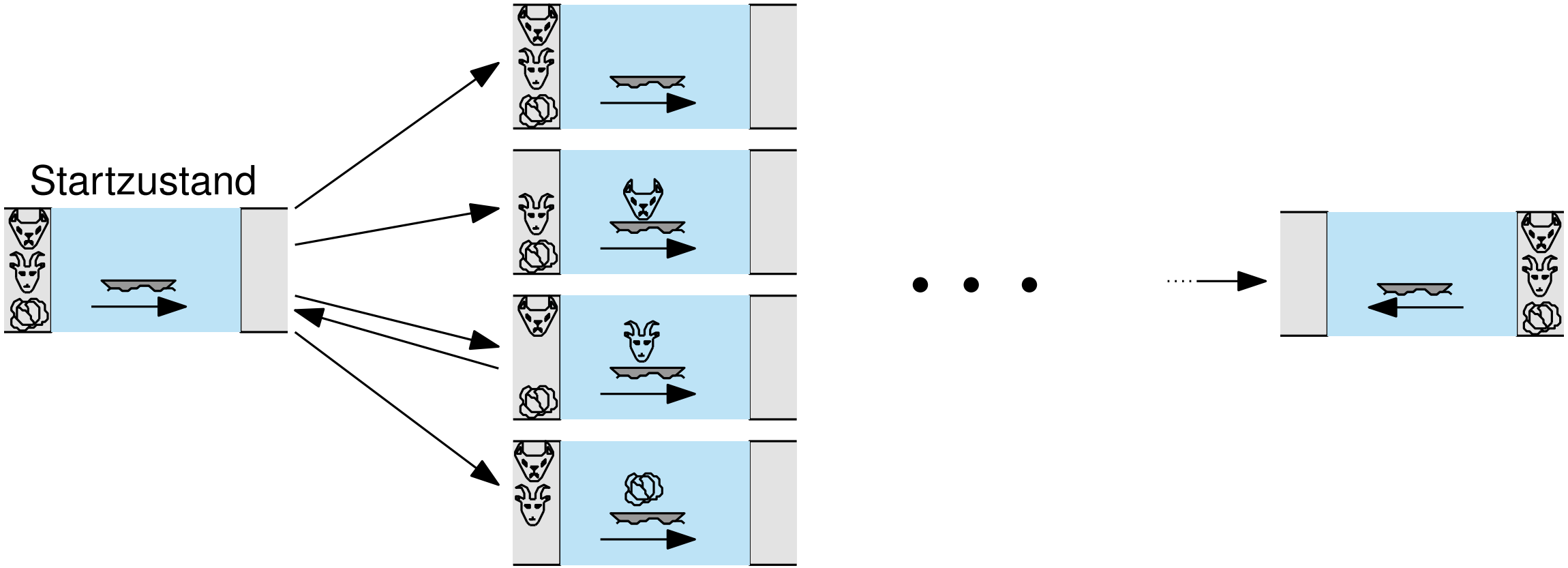
Das Kohlkopfproblem 3



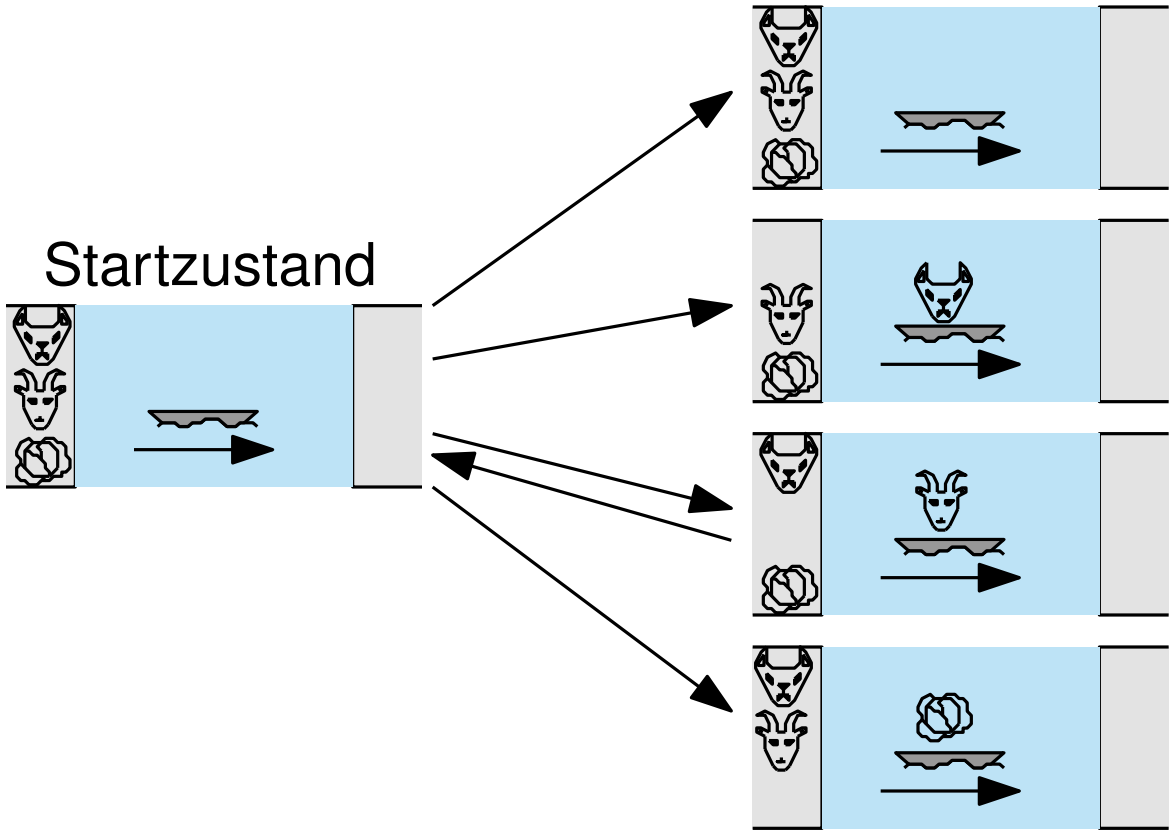
Das Kohlkopfproblem 3



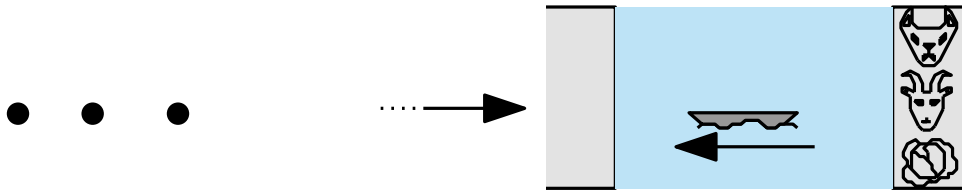
Das Kohlkopfproblem 3



Das Kohlkopfproblem 3



Was ist die optimale Lösung?



Das Kohlkopfproblem 4

- Wir können das Kohlkopfproblem in 7 Flussüberquerungen lösen

Das Kohlkopfproblem 4

- Wir können das Kohlkopfproblem in 7 Flussüberquerungen lösen
  - Zuerst die Ziege auf die andere Seite, dann kehren wir leer zurück

Das Kohlkopfproblem 4

- Wir können das Kohlkopfproblem in 7 Flussüberquerungen lösen
  - Zuerst die Ziege auf die andere Seite, dann kehren wir leer zurück
  - Nun entweder den Wolf oder den Kohlkopf

Das Kohlkopfproblem 4

- Wir können das Kohlkopfproblem in 7 Flussüberquerungen lösen
  - Zuerst die Ziege auf die andere Seite, dann kehren wir leer zurück
  - Nun entweder den Wolf oder den Kohlkopf
  - Wenn der Wolf oder Kohlkopf auf der anderen Seite ist, nehmen wir die Ziege wieder mit zurück

Das Kohlkopfproblem 4

- Wir können das Kohlkopfproblem in 7 Flussüberquerungen lösen
  - Zuerst die Ziege auf die andere Seite, dann kehren wir leer zurück
  - Nun entweder den Wolf oder den Kohlkopf
  - Wenn der Wolf oder Kohlkopf auf der anderen Seite ist, nehmen wir die Ziege wieder mit zurück
  - Dann ist die Ziege wieder auf der linken Seite und nehmen entweder den Kohlkopf oder den Wolf mit

Das Kohlkopfproblem 4

- Wir können das Kohlkopfproblem in 7 Flussüberquerungen lösen
  - Zuerst die Ziege auf die andere Seite, dann kehren wir leer zurück
  - Nun entweder den Wolf oder den Kohlkopf
  - Wenn der Wolf oder Kohlkopf auf der anderen Seite ist, nehmen wir die Ziege wieder mit zurück
  - Dann ist die Ziege wieder auf der linken Seite und nehmen entweder den Kohlkopf oder den Wolf mit
  - Dann fahren wir wieder leer zurück und holen in der letzten Überquerung wieder die Ziege mit

Breitensuche 1

- Kohlkopf-Problem hat Kanten ohne Kosten, kürzester Weg wird nur an Anzahl Kanten gemessen

Breitensuche 1

- Kohlkopf-Problem hat Kanten ohne Kosten, kürzester Weg wird nur an Anzahl Kanten gemessen
- Also untersuchen wir: kürzeste Wege in ungewichteten Graphen
  - Start und Ziel haben Distanz 1 $\iff$ Ziel ist Nachbar des Startknoten

Breitensuche 1

- Kohlkopf-Problem hat Kanten ohne Kosten, kürzester Weg wird nur an Anzahl Kanten gemessen
- Also untersuchen wir: kürzeste Wege in ungewichteten Graphen
  - Start und Ziel haben Distanz 1 $\iff$ Ziel ist Nachbar des Startknoten
  - Start und Ziel haben Distanz 2 $\iff$ Ziel ist Nachbar eines Nachbarn des Startknoten

Breitensuche 1

- Kohlkopf-Problem hat Kanten ohne Kosten, kürzester Weg wird nur an Anzahl Kanten gemessen
- Also untersuchen wir: kürzeste Wege in ungewichteten Graphen
  - Start und Ziel haben Distanz 1 $\iff$ Ziel ist Nachbar des Startknoten
  - Start und Ziel haben Distanz 2 $\iff$ Ziel ist Nachbar eines Nachbarn des Startknoten
  - Start und Ziel haben Distanz 3 $\iff$ Ziel ist Nachbar eines $\cdots$

$\vdots$

Breitensuche 1

- Kohlkopf-Problem hat Kanten ohne Kosten, kürzester Weg wird nur an Anzahl Kanten gemessen
- Also untersuchen wir: kürzeste Wege in ungewichteten Graphen
  - Start und Ziel haben Distanz 1 $\iff$ Ziel ist Nachbar des Startknoten
  - Start und Ziel haben Distanz 2 $\iff$ Ziel ist Nachbar eines Nachbarn des Startknoten
  - Start und Ziel haben Distanz 3 $\iff$ Ziel ist Nachbar eines $\cdots$

$\vdots$

$\implies$ Wir untersuchen Nachbarschaften vom Startknoten aus

Breitensuche 2

Konzeptuelle Idee des Algorithmus

- Starte bei s
- Gucke alle Nachbarn von s an

Breitensuche 2

Konzeptuelle Idee des Algorithmus

- Starte bei s
- Gucke alle Nachbarn von s an
- Falls z gefunden, gib aktuelle Distanz aus

Breitensuche 2

Konzeptuelle Idee des Algorithmus

- Starte bei s
- Gucke alle Nachbarn von s an
- Falls z gefunden, gib aktuelle Distanz aus
- Falls nicht, erhöhe Distanz um 1 und gucke Nachbarn der Nachbarn an
- Ignoriere bereits gefundene Knoten

Breitensuche 2

Konzeptuelle Idee des Algorithmus

- Starte bei s
- Gucke alle Nachbarn von s an
- Falls z gefunden, gib aktuelle Distanz aus
- Falls nicht, erhöhe Distanz um 1 und gucke Nachbarn der Nachbarn an
- Ignoriere bereits gefundene Knoten

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
```

Breitensuche 3

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
```

- Wie lange dauert das?

Breitensuche 3

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
```

- Wie lange dauert das?
  - Hauptüberlegung: Jeder Knoten landet maximal 1 mal in der Queue

Breitensuche 3

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
```

- Wie lange dauert das?
  - Hauptüberlegung: Jeder Knoten landet maximal 1 mal in der Queue
  - Für jeden Knoten wird jede ausgehende Kante 1 mal angeguckt

Breitensuche 3

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
```

- Wie lange dauert das?
  - Hauptüberlegung: Jeder Knoten landet maximal 1 mal in der Queue
  - Für jeden Knoten wird jede ausgehende Kante 1 mal angeguckt

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

Breitensuche 3

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
```

- Wie lange dauert das?
  - Hauptüberlegung: Jeder Knoten landet maximal 1 mal in der Queue
  - Für jeden Knoten wird jede ausgehende Kante 1 mal angeguckt

$$\implies BFS \in O(\sum_{v \in V} deg(v)) = O(2 \cdot |E|) = O(m)$$

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

heißt Handshake-Lemma und war Übungsblattaufgabe

Dijkstras Algorithmus 1

- Nun haben Sie den Beruf gewechselt und sind LKW-Fahrer. Sie sind aktuell in Stuttgart und wollen nach Tübingen fahren.

Dijkstras Algorithmus 1

- Nun haben Sie den Beruf gewechselt und sind LKW-Fahrer. Sie sind aktuell in Stuttgart und wollen nach Tübingen fahren.

- Dabei wollen Sie natürlich den kürzesten Weg fahren

Dijkstras Algorithmus 1

- Nun haben Sie den Beruf gewechselt und sind LKW-Fahrer. Sie sind aktuell in Stuttgart und wollen nach Tübingen fahren.
- Dabei wollen Sie natürlich den kürzesten Weg fahren
- An jeder Kreuzung können Sie entscheiden, welche Straße Sie als nächstes nehmen

Dijkstras Algorithmus 1

- Nun haben Sie den Beruf gewechselt und sind LKW-Fahrer. Sie sind aktuell in Stuttgart und wollen nach Tübingen fahren.
- Dabei wollen Sie natürlich den kürzesten Weg fahren
- An jeder Kreuzung können Sie entscheiden, welche Straße Sie als nächstes nehmen
- Jeder Straßenabschnitt zwischen zwei Kreuzungen hat eine Länge

Dijkstras Algorithmus 1

- Nun haben Sie den Beruf gewechselt und sind LKW-Fahrer. Sie sind aktuell in Stuttgart und wollen nach Tübingen fahren.
- Dabei wollen Sie natürlich den kürzesten Weg fahren
- An jeder Kreuzung können Sie entscheiden, welche Straße Sie als nächstes nehmen
- Jeder Straßenabschnitt zwischen zwei Kreuzungen hat eine Länge
- Welche Folge von Straßen, startend in Stuttgart und endend in Tübingen, hat die kürzeste Distanz?

Dijkstras Algorithmus 2

- Das Problem können wir als gewichteten Graphen modellieren

Dijkstras Algorithmus 2

- Das Problem können wir als gewichteten Graphen modellieren
  - Kreuzungen sind Knoten
  - Straßenabschnitte zwischen zwei Kreuzungen sind Kanten
  - zusätzlich hat jede Kante e ein Gewicht w, das ist genau die Länge des Straßenabschnitts

Dijkstras Algorithmus 2

- Das Problem können wir als gewichteten Graphen modellieren
  - Kreuzungen sind Knoten
  - Straßenabschnitte zwischen zwei Kreuzungen sind Kanten
  - zusätzlich hat jede Kante e ein Gewicht w, das ist genau die Länge des Straßenabschnitts
- Wir starten an einer bestimmten Kreuzung s in Stuttgart und wollen zu einer bestimmten Kreuzung t in Tübingen

Dijkstras Algorithmus 2

- Das Problem können wir als gewichteten Graphen modellieren
  - Kreuzungen sind Knoten
  - Straßenabschnitte zwischen zwei Kreuzungen sind Kanten
  - zusätzlich hat jede Kante e ein Gewicht w, das ist genau die Länge des Straßenabschnitts
- Wir starten an einer bestimmten Kreuzung s in Stuttgart und wollen zu einer bestimmten Kreuzung t in Tübingen
- Jede Kante hat ein positives Gewicht, denn es gibt keine Straßen mit negativer Länge

Dijkstras Algorithmus 3

- Wir suchen die Länge des billigsten Weges von s nach t, also den s-t-Pfad P mit $w(P) = \sum_{e \in P} w(e)$ minimal

Dijkstras Algorithmus 3

- Wir suchen die Länge des billigsten Weges von s nach t, also den s-t-Pfad P mit $w(P) = \sum_{e \in P} w(e)$ minimal

- Unsere Idee

Dijkstras Algorithmus 3

- Wir suchen die Länge des billigsten Weges von s nach t, also den s-t-Pfad P mit $w(P) = \sum_{e \in P} w(e)$ minimal

- Unsere Idee
  - Wir besuchen nacheinander Knoten mit niedrigster Distanz zu s
  - Dann ist erster Besuch bei t auch kürzester Pfad

Dijkstras Algorithmus 3

- Wir suchen die Länge des billigsten Weges von s nach t, also den s-t-Pfad P mit $w(P) = \sum_{e \in P} w(e)$ minimal

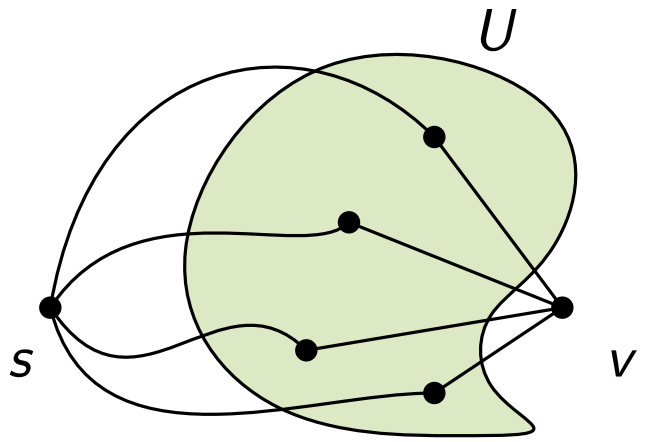
- Unsere Idee
  - Wir besuchen nacheinander Knoten mit niedrigster Distanz zu s
  - Dann ist erster Besuch bei t auch kürzester Pfad
  - Also: wie finden wir Distanz von beliebigen Knoten zu s?

Dijkstras Algorithmus 4

- Wie finden wir Distanz von Knoten s zu v?

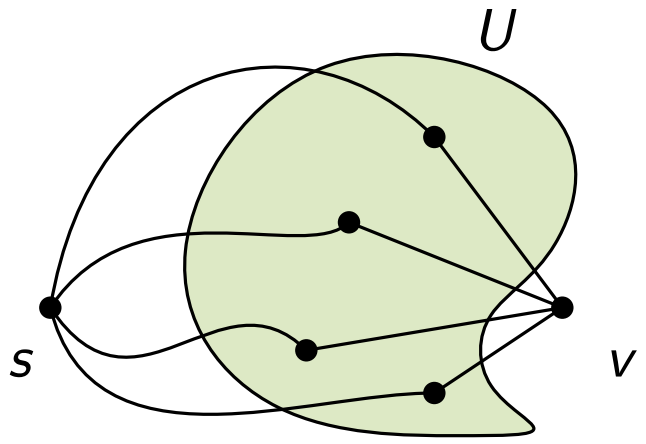
Dijkstras Algorithmus 4

- Wie finden wir Distanz von Knoten s zu v?
- Angenommen wir wissen schon kürzeste Distanzen von s zu Menge U von Nachbarn von v



Dijkstras Algorithmus 4

- Wie finden wir Distanz von Knoten s zu v?
- Angenommen wir wissen schon kürzeste Distanzen von s zu Menge U von Nachbarn von v
- Dann suchen wir minimale Länge des Pfades von s zu Knoten $u \in U$ und dann Kante (u, v)



$$d(s, v) = min_{u \in U}(d(s, u) + len(u, v))$$

Dijsktras Algorithmus 5

- Formulierung des Algorithmus
  - Starte bei s

Dijsktras Algorithmus 5

- Formulierung des Algorithmus
  - Starte bei s
  - Wähle immer Nachbar von bereits explorierten Knoten mit minimaler Distanz zu s

Dijsktras Algorithmus 5

- Formulierung des Algorithmus
  - Starte bei s
  - Wähle immer Nachbar von bereits explorierten Knoten mit minimaler Distanz zu s
  - Falls neuer Knoten exploriert wurde, ändern sich minimale Distanzen, also eventuelles aktualisieren

Dijsktras Algorithmus 5

- Formulierung des Algorithmus
  - Starte bei s
  - Wähle immer Nachbar von bereits explorierten Knoten mit minimaler Distanz zu s
  - Falls neuer Knoten exploriert wurde, ändern sich minimale Distanzen, also eventuelles aktualisieren

  Distanz von s über explorierten Knoten v:
  $d(s,u) = d(s,v) + len(v,u)$

```
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])
```

Dijsktras Algorithmus 5

- Formulierung des Algorithmus
  - Starte bei s
  - Wähle immer Nachbar von bereits explorierten Knoten mit minimaler Distanz zu s
  - Falls neuer Knoten exploriert wurde, ändern sich minimale Distanzen, also eventuelles aktualisieren

Distanz von s über explorierten Knoten v:
$d(s,u) = d(s,v) + len(v,u)$

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 6

```
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])
```

- Wie lange dauert das?

Dijkstras Algorithmus 6

```
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])
```

- Wie lange dauert das?
  - Jeder Knoten wird genau 1 mal gepusht $\implies$ n PUSH

Dijkstras Algorithmus 6

```
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])
```

- Wie lange dauert das?
  - Jeder Knoten wird genau 1 mal gepusht $\implies$ n PUSH
  - Jeder Knoten wird genau einmal entfernt $\implies$ n popMIN

Dijkstras Algorithmus 6

```
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])
```

- Wie lange dauert das?
  - Jeder Knoten wird genau 1 mal gepusht
    $\Longrightarrow$ n PUSH
  - Jeder Knoten wird genau einmal entfernt
    $\Longrightarrow$ n popMIN
  - Jede Kante wird von beiden Seiten betrachtet
    $\Longrightarrow$ m decPRIO

Dijkstras Algorithmus 6

Dijkstra(*Graph* G, *Node* s)

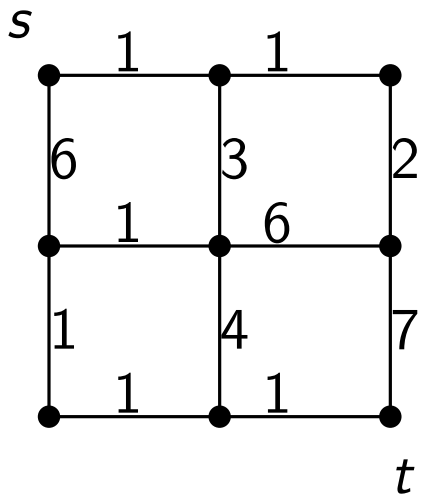
```
d := Array of size n initialized with ∞
d[s] := 0
PriorityQueue Q := empty priority queue
for Node v in V do
    Q.push(v, d[v])
while Q ≠ ∅ do
    u := Q.popMin()
    for Node v in N(u) do
        if d[v] > d[u] + len(u, v) then
            d[v] := d[u] + len(u, v)
            Q.decPrio(v, d[v])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

- Wie lange dauert das?
  - Jeder Knoten wird genau 1 mal gepusht $\Longrightarrow$ n PUSH
  - Jeder Knoten wird genau einmal entfernt $\Longrightarrow$ n popMIN
  - Jede Kante wird von beiden Seiten betrachtet $\Longrightarrow$ m decPRIO

$\Longrightarrow$ insgesamt $\Theta(n + n\log(n) + m\log(n)) = \Theta((n+m)\log(n))$

Kürzester Pfad

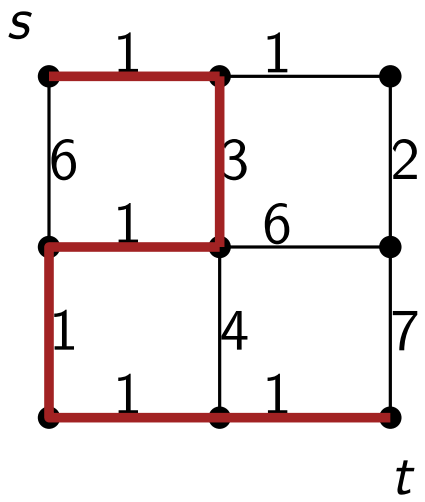
Wie lange sind die kürzesten s-t-Pfade in diesen Graphen?



Graph G_1

Kürzester Pfad

Wie lange sind die kürzesten s-t-Pfade in diesen Graphen?

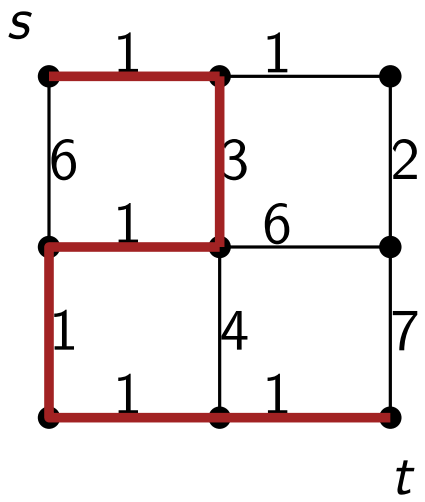


Graph G_1

kürzester s-t-Pfad hat Länge 8

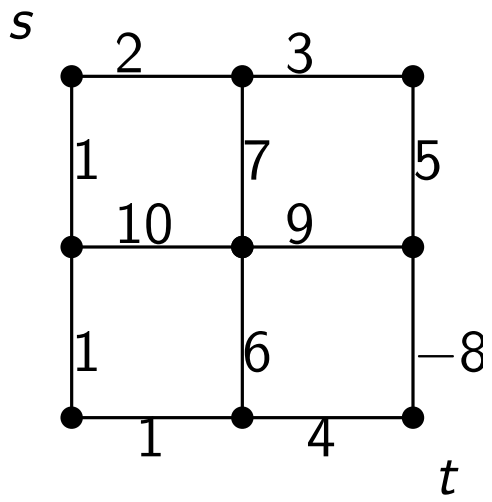
Kürzester Pfad

Wie lange sind die kürzesten s-t-Pfade in diesen Graphen?



Graph G_1

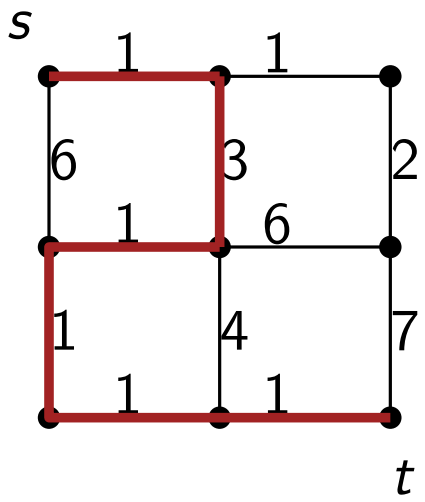
kürzester s-t-Pfad hat Länge 8



Graph G_2

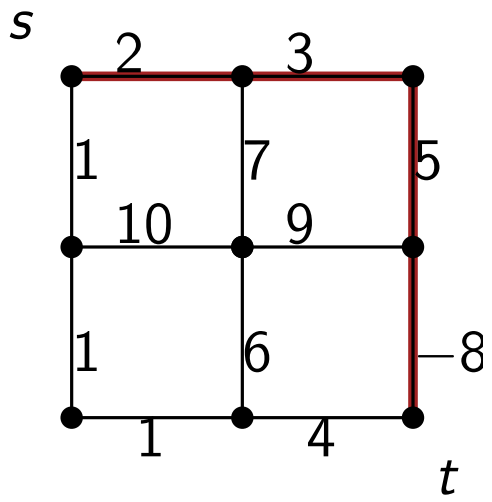
Kürzester Pfad

Wie lange sind die kürzesten s-t-Pfade in diesen Graphen?



Graph G_1

kürzester s-t-Pfad hat Länge 8

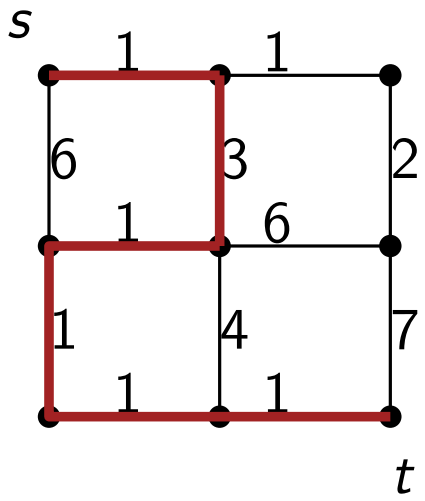


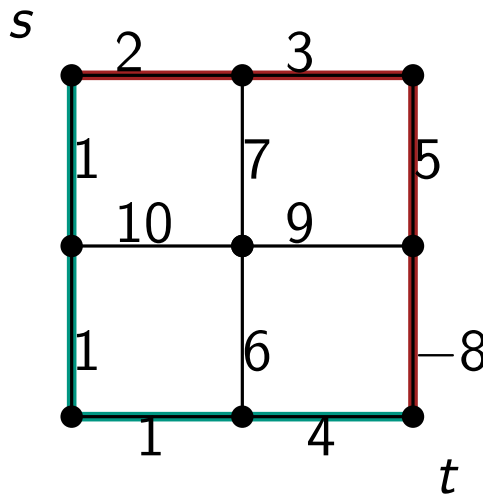
Graph G_2

kürzester s-t-Pfad hat Länge 2

Kürzester Pfad

Wie lange sind die kürzesten s-t-Pfade in diesen Graphen?





Graph G_1

kürzester s-t-Pfad hat Länge 8

Graph G_2

kürzester s-t-Pfad hat Länge 2

Dijkstra hätte einen Weg mit Länge 7 gefunden

Bellman-Fords Algorithmus

- Da Sie ihren neuen Job als LKW-Fahrer so gut ausüben, dürfen Sie nun in einem neuen Elektro-LKW fahren

Bellman-Fords Algorithmus

- Da Sie ihren neuen Job als LKW-Fahrer so gut ausüben, dürfen Sie nun in einem neuen Elektro-LKW fahren
- Neuerdings gibt es auch Autobahnen auf denen ihr LKW laden kann, während er fährt

Bellman-Fords Algorithmus

- Da Sie ihren neuen Job als LKW-Fahrer so gut ausüben, dürfen Sie nun in einem neuen Elektro-LKW fahren
- Neuerdings gibt es auch Autobahnen auf denen ihr LKW laden kann, während er fährt
- Dadurch verliert er nicht nur keine Ladung während er fährt, er gewinnt auf diesen Strecken nun sogar Ladung hinzu!

Bellman-Fords Algorithmus

- Da Sie ihren neuen Job als LKW-Fahrer so gut ausüben, dürfen Sie nun in einem neuen Elektro-LKW fahren
- Neuerdings gibt es auch Autobahnen auf denen ihr LKW laden kann, während er fährt
- Dadurch verliert er nicht nur keine Ladung während er fährt, er gewinnt auf diesen Strecken nun sogar Ladung hinzu!
- Optimalerweise wollen wir auf der Route vom Start zum Ziel möglichst wenig Ladung verbrauchen
  - Im Szenario davor: meiste Ladung bei Ankunft = kürzeste Route

Bellman-Fords Algorithmus

- Mit Dijkstras Algorithmus: erster Besuch bei Zielknoten t = kürzester Pfad

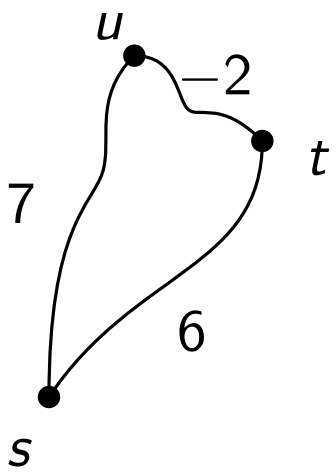
Bellman-Fords Algorithmus

- Mit Dijkstras Algorithmus: erster Besuch bei Zielknoten t = kürzester Pfad
- Mit negativen Kantengewichten stimmt dies nicht mehr!

Bellman-Fords Algorithmus

- Mit Dijkstras Algorithmus: erster Besuch bei Zielknoten t = kürzester Pfad
- Mit negativen Kantengewichten stimmt dies nicht mehr!

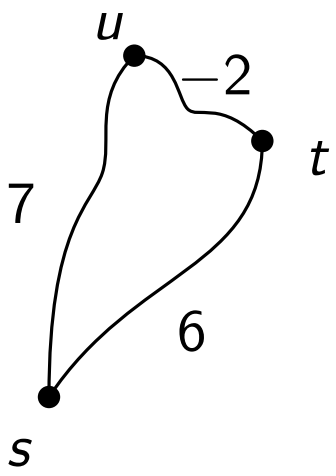
z.B.:



Bellman-Fords Algorithmus

- Mit Dijkstras Algorithmus: erster Besuch bei Zielknoten t = kürzester Pfad
- Mit negativen Kantengewichten stimmt dies nicht mehr!

z.B.:

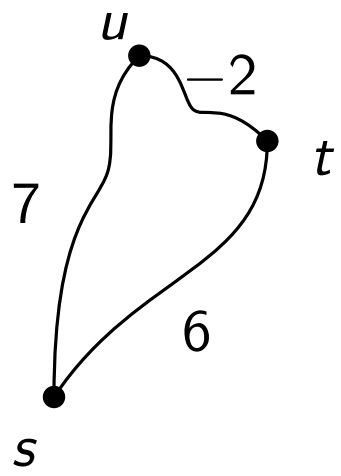


Dijkstra findet t zuerst über Kante $\{s, t\}$ und meint damit, kürzester s-t-Pfad hat Länge 6

Bellman-Fords Algorithmus

- Mit Dijkstras Algorithmus: erster Besuch bei Zielknoten t = kürzester Pfad
- Mit negativen Kantengewichten stimmt dies nicht mehr!

z.B.:



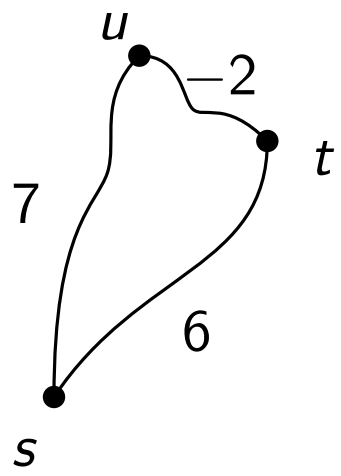
Dijkstra findet t zuerst über Kante $\{s, t\}$ und meint damit, kürzester s-t-Pfad hat Länge 6

Aber zuerst $\{s, u\}$ und dann $\{u, t\}$ zu nehmen ist billiger, dieser Pfad hat Länge 5

Bellman-Fords Algorithmus

- Mit Dijkstras Algorithmus: erster Besuch bei Zielknoten t = kürzester Pfad
- Mit negativen Kantengewichten stimmt dies nicht mehr!

z.B.:



Dijkstra findet t zuerst über Kante $\{s, t\}$ und meint damit, kürzester s-t-Pfad hat Länge 6

Aber zuerst $\{s, u\}$ und dann $\{u, t\}$ zu nehmen ist billiger, dieser Pfad hat Länge 5

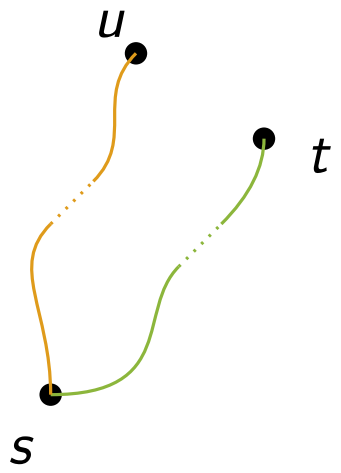
$\Longrightarrow$ Wir brauchen eine neue Strategie

Bellman-Fords Algorithmus

- Neue Idee: Wir entscheiden für jede Kante, ob wir sie nehmen oder nicht

Bellman-Fords Algorithmus

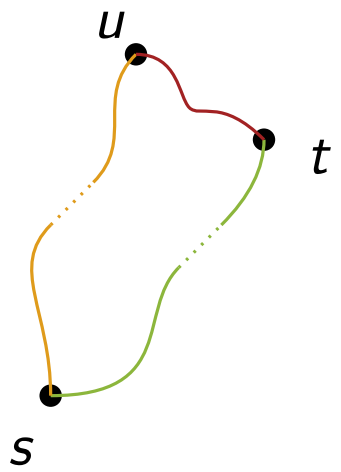
- Neue Idee: Wir entscheiden für jede Kante, ob wir sie nehmen oder nicht



  - Angenommen wir haben s-t-Pfad und s-u-Pfad

Bellman-Fords Algorithmus

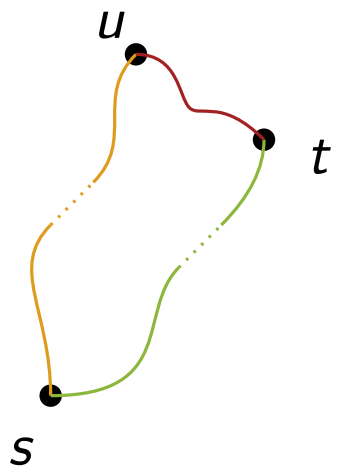
- Neue Idee: Wir entscheiden für jede Kante, ob wir sie nehmen oder nicht



  - Angenommen wir haben s-t-Pfad und s-u-Pfad
  - Betrachte Kante $\{u, t\}$

Bellman-Fords Algorithmus

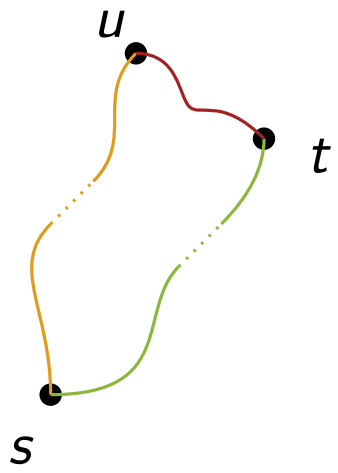
- Neue Idee: Wir entscheiden für jede Kante, ob wir sie nehmen oder nicht



  - Angenommen wir haben s-t-Pfad und s-u-Pfad
  - Betrachte Kante $\{u, t\}$
  - Falls $dist(s, u) + len(u, t) < dist(s, t)$ so ist s-u-Pfad $+$ Kante $\{u, t\}$ billiger als urpsprünglicher s-t-Pfad

Bellman-Fords Algorithmus

- Neue Idee: Wir entscheiden für jede Kante, ob wir sie nehmen oder nicht



  - Angenommen wir haben s-t-Pfad und s-u-Pfad
  - Betrachte Kante $\{u, t\}$
  - Falls $dist(s, u) + len(u, t) < dist(s, t)$ so ist s-u-Pfad $+$ Kante $\{u, t\}$ billiger als urpsprünglicher s-t-Pfad

- Gehe Prinzip für alle Kanten $\{v, w\}$ durch: Falls $dist(s, v) + len(v, w) < dist(s, w)$ so benutze Kante $\{v, w\}$ und $dist(s, w) = dist(s, v) + len(v, w)$

Bellman-Fords Algorithmus

- Nun haben sich manche Distanzen verändert!
- Abschätzung $dist(s,u) + len(u,t) < dist(s,t)$ ist eventuell nicht die gleiche, da sich $dist(s,u)$ und $dist(s,t)$ verändert haben können

Bellman-Fords Algorithmus

- Nun haben sich manche Distanzen verändert!
- Abschätzung $dist(s,u) + len(u,t) < dist(s,t)$ ist eventuell nicht die gleiche, da sich $dist(s,u)$ und $dist(s,t)$ verändert haben können

$\Longrightarrow$ Prinzip nochmal für alle Kanten wiederholen

Bellman-Fords Algorithmus

- Nun haben sich manche Distanzen verändert!
- Abschätzung $dist(s,u) + len(u,t) < dist(s,t)$ ist eventuell nicht die gleiche, da sich $dist(s,u)$ und $dist(s,t)$ verändert haben können

$\Longrightarrow$ Prinzip nochmal für alle Kanten wiederholen

- Da sich Distanzen nur verkleinern, enden wir irgendwann $(*)$
- Falls $n = |V|$ die Anzahl der Knoten, so sollten wir nach $n-1$ Wiederholungen fertig sein $(*)$

Bellman-Fords Algorithmus

- Nun haben sich manche Distanzen verändert!
- Abschätzung $dist(s,u) + len(u,t) < dist(s,t)$ ist eventuell nicht die gleiche, da sich $dist(s,u)$ und $dist(s,t)$ verändert haben können

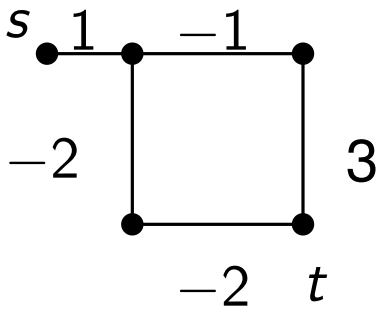
$\Longrightarrow$ Prinzip nochmal für alle Kanten wiederholen

- Da sich Distanzen nur verkleinern, enden wir irgendwann $(*)$
- Falls $n = |V|$ die Anzahl der Knoten, so sollten wir nach $n-1$ Wiederholungen fertig sein $(*)$

$*$ Außer wir haben negative Kreise

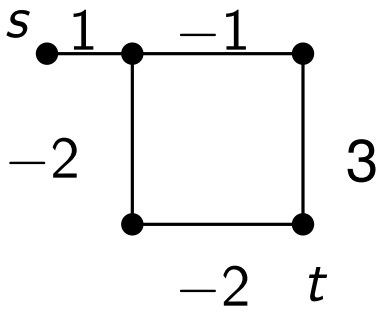
Bellman-Fords Algorithmus

- Was passiert bei negativen Kreisen?



Bellman-Fords Algorithmus

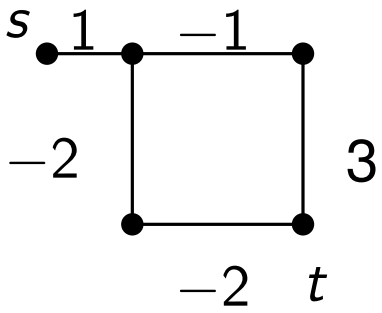
- Was passiert bei negativen Kreisen?



Kreis wird nur teilweise durchlaufen $\implies dist(s,t) = -3$

Bellman-Fords Algorithmus

- Was passiert bei negativen Kreisen?

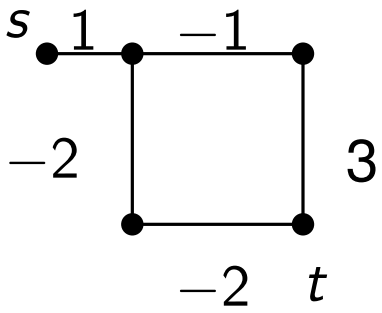


Kreis wird nur teilweise durchlaufen $\implies dist(s,t) = -3$

Kreis wird einmal durchlaufen $\implies dist(s,t) = -5$

Bellman-Fords Algorithmus

- Was passiert bei negativen Kreisen?



Kreis wird nur teilweise durchlaufen $\implies dist(s,t) = -3$

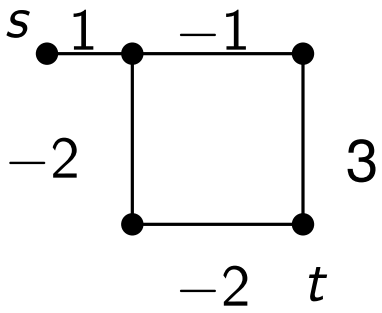
Kreis wird einmal durchlaufen $\implies dist(s,t) = -5$

Kreis wird zweimal durchlaufen $\implies dist(s,t) = -7$

$\vdots$

Bellman-Fords Algorithmus

- Was passiert bei negativen Kreisen?



Kreis wird nur teilweise durchlaufen $\implies dist(s,t) = -3$

Kreis wird einmal durchlaufen $\implies dist(s,t) = -5$

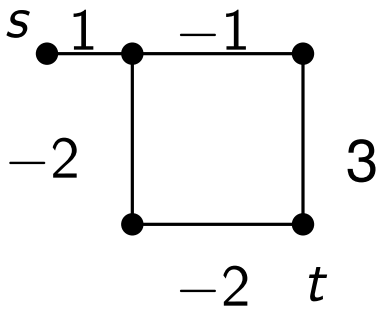
Kreis wird zweimal durchlaufen $\implies dist(s,t) = -7$

$\vdots$

- Kürzeste Distanzen würden sich nicht stabilisieren, da negativer Kreis unendlich oft durchlaufen würde

Bellman-Fords Algorithmus

- Was passiert bei negativen Kreisen?



Kreis wird nur teilweise durchlaufen $\implies dist(s,t) = -3$

Kreis wird einmal durchlaufen $\implies dist(s,t) = -5$

Kreis wird zweimal durchlaufen $\implies dist(s,t) = -7$

$\vdots$

- Kürzeste Distanzen würden sich nicht stabilisieren, da negativer Kreis unendlich oft durchlaufen würde

- Also funktioniert unser Verfahren nur für Graphen ohne negative Kreise

Bellman-Fords Algorithmus

- Wie können wir erkennen, ob ein Graph negative Kreise hat?

Bellman-Fords Algorithmus

- Wie können wir erkennen, ob ein Graph negative Kreise hat?
  - Falls er keine negative Kreise hat, so ändern sich die Distanzen von s zu allen anderen Knoten nach der $n-1$ten Iteration nicht mehr

Bellman-Fords Algorithmus

- Wie können wir erkennen, ob ein Graph negative Kreise hat?
  - Falls er keine negative Kreise hat, so ändern sich die Distanzen von s zu allen anderen Knoten nach der $n-1$ten Iteration nicht mehr
  - Sollte der Graph negative Kreise haben, so ändert sich mindestens eine Distanz nach der $n$ten Iteration

Bellman-Fords Algorithmus

- Wie können wir erkennen, ob ein Graph negative Kreise hat?
  - Falls er keine negative Kreise hat, so ändern sich die Distanzen von s zu allen anderen Knoten nach der $n-1$ten Iteration nicht mehr
  - Sollte der Graph negative Kreise haben, so ändert sich mindestens eine Distanz nach der $n$ten Iteration

$\Longrightarrow$ Wir führen unser Verfahren einfach ein $n$tes mal durch

Bellman-Fords Algorithmus

- Wie können wir erkennen, ob ein Graph negative Kreise hat?
  - Falls er keine negative Kreise hat, so ändern sich die Distanzen von s zu allen anderen Knoten nach der $n-1$ten Iteration nicht mehr
  - Sollte der Graph negative Kreise haben, so ändert sich mindestens eine Distanz nach der $n$ten Iteration

$\Longrightarrow$ Wir führen unser Verfahren einfach ein $n$tes mal durch

- Falls wir negative Kreise haben, so ändert sich mindestens eine Distanz und wir geben einen Fehler aus

Bellman Fords-Algorithmus

- Formulieren wir nun unseren Algorithmus:
- Wir speichern die aktuellen Distanzen von s zu jedem anderen Knoten

Bellman Fords-Algorithmus

- Formulieren wir nun unseren Algorithmus:
- Wir speichern die aktuellen Distanzen von s zu jedem anderen Knoten
- Wir testen für jede Kante (u, v) ob $dist(s, u) + len(u, v) < dist(s, v)$ und ändern gegebenenfalls $dist(s, v)$

Bellman Fords-Algorithmus

- Formulieren wir nun unseren Algorithmus:
- Wir speichern die aktuellen Distanzen von s zu jedem anderen Knoten
- Wir testen für jede Kante (u, v) ob $dist(s, u) + len(u, v) < dist(s, v)$ und ändern gegebenenfalls $dist(s, v)$
- Dies führen wir $n - 1$ mal aus

Bellman Fords-Algorithmus

- Formulieren wir nun unseren Algorithmus:
- Wir speichern die aktuellen Distanzen von s zu jedem anderen Knoten
- Wir testen für jede Kante (u, v) ob $dist(s, u) + len(u, v) < dist(s, v)$ und ändern gegebenenfalls $dist(s, v)$
- Dies führen wir $n - 1$ mal aus
- Dann führen wir es ein n-tes mal aus, falls eine Distanz sich ändert, gibt es einen negativen Kreis

Bellman Fords-Algorithmus

- Formulieren wir nun unseren Algorithmus:
- Wir speichern die aktuellen Distanzen von s zu jedem anderen Knoten
- Wir testen für jede Kante (u, v) ob $dist(s, u) + len(u, v) < dist(s, v)$ und ändern gegebenenfalls $dist(s, v)$
- Dies führen wir $n - 1$ mal aus
- Dann führen wir es ein n-tes mal aus, falls eine Distanz sich ändert, gibt es einen negativen Kreis

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

BellmanFord(*Graph* G, *Node* s)

$d :=$ *Array* of size n initialized with ∞

$d[s] := 0$

for $n - 1$ iterations **do**

  for *Edge* $(u, v) \in E$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

// test for negative cycle

for *Edge* $(u, v) \in E$ **do**

  if $d[v] > d[u] + \text{len}(u, v)$ **then**

    return negative cycle

return d

- Wie lange dauert das?

Bellman-Fords Algorithmus

BellmanFord(*Graph* G, *Node* s)

$d :=$ *Array* of size n initialized with ∞

$d[s] := 0$

for $n - 1$ iterations **do**

  for *Edge* $(u, v) \in E$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

// test for negative cycle

for *Edge* $(u, v) \in E$ **do**

  if $d[v] > d[u] + \text{len}(u, v)$ **then**

    return negative cycle

return d

- Wie lange dauert das?
  - Array der Größe n erstellen in $O(n)$

Bellman-Fords Algorithmus

BellmanFord(*Graph* G, *Node* s)

$d :=$ *Array* of size n initialized with ∞

$d[s] := 0$

for $n - 1$ iterations **do**

  for *Edge* $(u, v) \in E$ **do**

    if $d[v] > d[u] + \text{len}(u, v)$ **then**

      $d[v] := d[u] + \text{len}(u, v)$

// test for negative cycle

for *Edge* $(u, v) \in E$ **do**

  if $d[v] > d[u] + \text{len}(u, v)$ **then**

    return negative cycle

return d

- Wie lange dauert das?
  - Array der Größe n erstellen in $O(n)$
  - n Schleifendurchläufe

Bellman-Fords Algorithmus

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
```

- Wie lange dauert das?
  - Array der Größe n erstellen in $O(n)$
  - n Schleifendurchläufe
    - In jedem Schleifendurchlauf werden alle m Kanten betrachtet

Bellman-Fords Algorithmus

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
```

- Wie lange dauert das?
  - Array der Größe n erstellen in $O(n)$
  - n Schleifendurchläufe
    - In jedem Schleifendurchlauf werden alle m Kanten betrachtet
- Insgesamt $\Theta(n + n \cdot m) = \Theta(nm)$

Floyd-Warshalls Algorithmus

- So erfolgreich wie Sie als LKW-Fahrer sind, werden Sie in viele verschiedenen Routen eingespannt

Floyd-Warshalls Algorithmus

- So erfolgreich wie Sie als LKW-Fahrer sind, werden Sie in viele verschiedenen Routen eingespannt
- Anstelle für jede Route den optimalen Weg neu zu berechnen, wollen Sie nun einmal für alle Wege den besten Weg berechnen und speichern

Floyd-Warshalls Algorithmus

- So erfolgreich wie Sie als LKW-Fahrer sind, werden Sie in viele verschiedenen Routen eingespannt
- Anstelle für jede Route den optimalen Weg neu zu berechnen, wollen Sie nun einmal für alle Wege den besten Weg berechnen und speichern
- Dann können Sie in Zukunft einfach den optimalen Weg abfragen

Floyd-Warshalls Algorithmus

- So erfolgreich wie Sie als LKW-Fahrer sind, werden Sie in viele verschiedenen Routen eingespannt
- Anstelle für jede Route den optimalen Weg neu zu berechnen, wollen Sie nun einmal für alle Wege den besten Weg berechnen und speichern
- Dann können Sie in Zukunft einfach den optimalen Weg abfragen
- Früher haben wir $SSSP$ betrachtet (Single-Source-Shortest-Path), also kürzeste Pfade von einem Knoten aus

Floyd-Warshalls Algorithmus

- So erfolgreich wie Sie als LKW-Fahrer sind, werden Sie in viele verschiedenen Routen eingespannt
- Anstelle für jede Route den optimalen Weg neu zu berechnen, wollen Sie nun einmal für alle Wege den besten Weg berechnen und speichern
- Dann können Sie in Zukunft einfach den optimalen Weg abfragen
- Früher haben wir $SSSP$ betrachtet (Single-Source-Shortest-Path), also kürzeste Pfade von einem Knoten aus
- Nun betrachten wir $APSP$ (All-Pairs-Shortest-Path) also kürzeste Pfade zwischen allen Knotenpaaren

Floyd-Warshalls Algorithmus

- Idee wie bei Bellman-Ford: Startknoten s, Zielknoten t, für jeden Knoten v berechnen wir: 'Ist es schneller zuerst von s zu v und dann von v zu t zu gehen?'

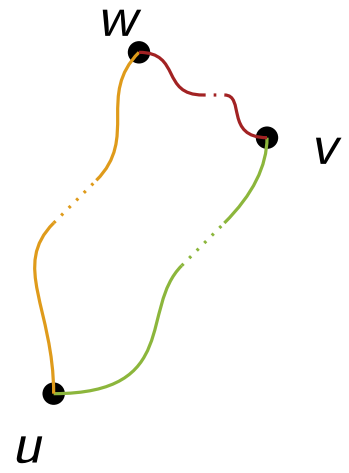
Floyd-Warshalls Algorithmus

- Idee wie bei Bellman-Ford: Startknoten s, Zielknoten t, für jeden Knoten v berechnen wir: 'Ist es schneller zuerst von s zu v und dann von v zu t zu gehen?'

- Nun aber für alle Paare (u, v) als Start und Ende

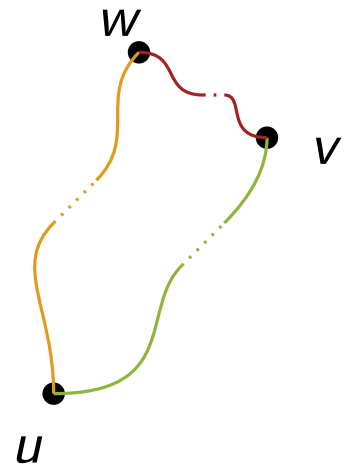
Floyd-Warshalls Algorithmus

- Idee wie bei Bellman-Ford: Startknoten s, Zielknoten t, für jeden Knoten v berechnen wir: 'Ist es schneller zuerst von s zu v und dann von v zu t zu gehen?'
- Nun aber für alle Paare (u, v) als Start und Ende
- Wieder für jeden Knoten w gleicher Test: ist $dist(u, w) + dist(w, v) < dist(u, v)$



Floyd-Warshalls Algorithmus

- Idee wie bei Bellman-Ford: Startknoten s, Zielknoten t, für jeden Knoten v berechnen wir: 'Ist es schneller zuerst von s zu v und dann von v zu t zu gehen?'
- Nun aber für alle Paare (u, v) als Start und Ende
- Wieder für jeden Knoten w gleicher Test: ist $dist(u, w) + dist(w, v) < dist(u, v)$
  - Dann setze $dist(u, v) = dist(u, w) + dist(w, v)$

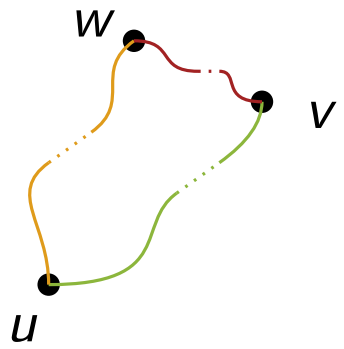


Floyd-Warshalls Algorithmus

- Formulierung unseres Algorithmus:
  - Schon bekannte Wege speichern, also genau Kanten und ihre Längen

Floyd-Warshalls Algorithmus

- Formulierung unseres Algorithmus:
  - Schon bekannte Wege speichern, also genau Kanten und ihre Längen
  - Nun, für jeden Knoten v und jedes Knotenpaar (s, t) betrachten, ob $dist(s, v) + dist(v, t) < dist(s, t)$ und entsprechend den kürzesten Weg aktualisieren

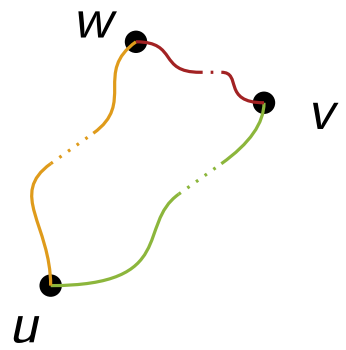


Floyd-Warshalls Algorithmus

- Formulierung unseres Algorithmus:
  - Schon bekannte Wege speichern, also genau Kanten und ihre Längen
  - Nun, für jeden Knoten v und jedes Knotenpaar (s, t) betrachten, ob $dist(s, v) + dist(v, t) < dist(s, t)$ und entsprechend den kürzesten Weg aktualisieren

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

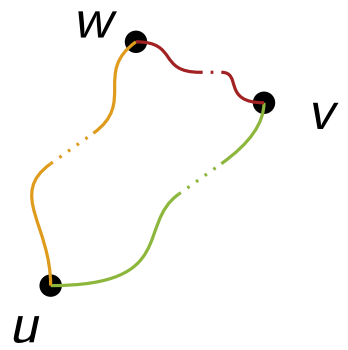


Floyd-Warshalls Algorithmus

- Formulierung unseres Algorithmus:
  - Schon bekannte Wege speichern, also genau Kanten und ihre Längen
  - Nun, für jeden Knoten v und jedes Knotenpaar (s, t) betrachten, ob $dist(s, v) + dist(v, t) < dist(s, t)$ und entsprechend den kürzesten Weg aktualisieren

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

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

- Laufzeit?

Floyd-Warshalls Algorithmus

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

- Laufzeit?
  - Am Anfang für jede Kante und jeden Knoten eine Distanz initialisieren $\implies \Theta(m + n)$

Floyd-Warshalls Algorithmus

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

- Laufzeit?
  - Am Anfang für jede Kante und jeden Knoten eine Distanz initialisieren $\implies \Theta(m + n)$
  - Für jeden der n Knoten betrachten wir jedes der n^2 Knotenpaare

Floyd-Warshalls Algorithmus

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

- Laufzeit?
  - Am Anfang für jede Kante und jeden Knoten eine Distanz initialisieren $\implies \Theta(m + n)$
  - Für jeden der n Knoten betrachten wir jedes der n^2 Knotenpaare

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

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.

Minimale Spannbäume 1

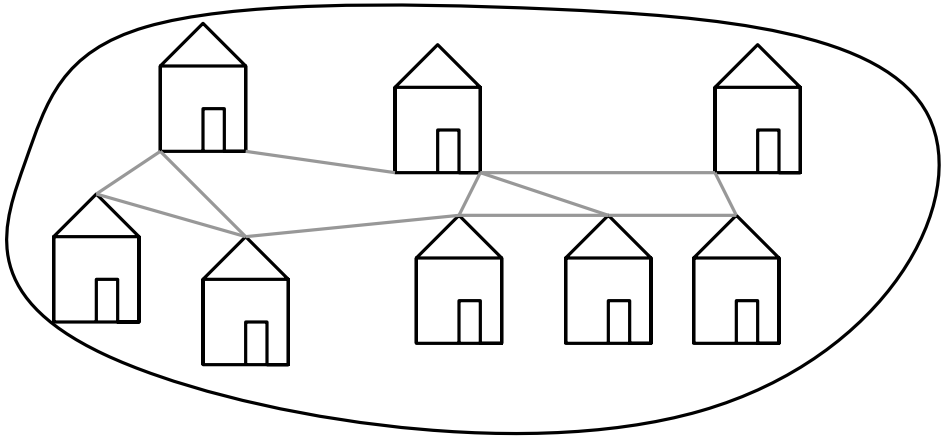
- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen

Minimale Spannbäume 1

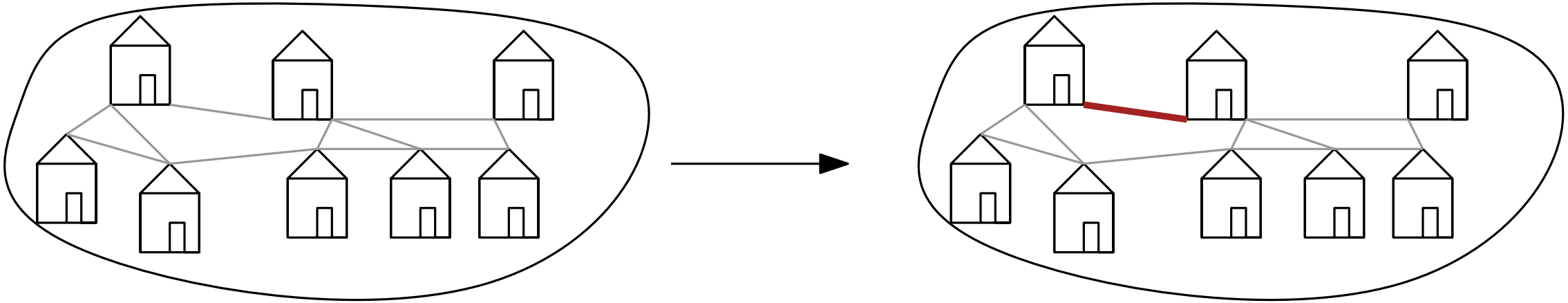
- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen



Nachbarschaft

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen

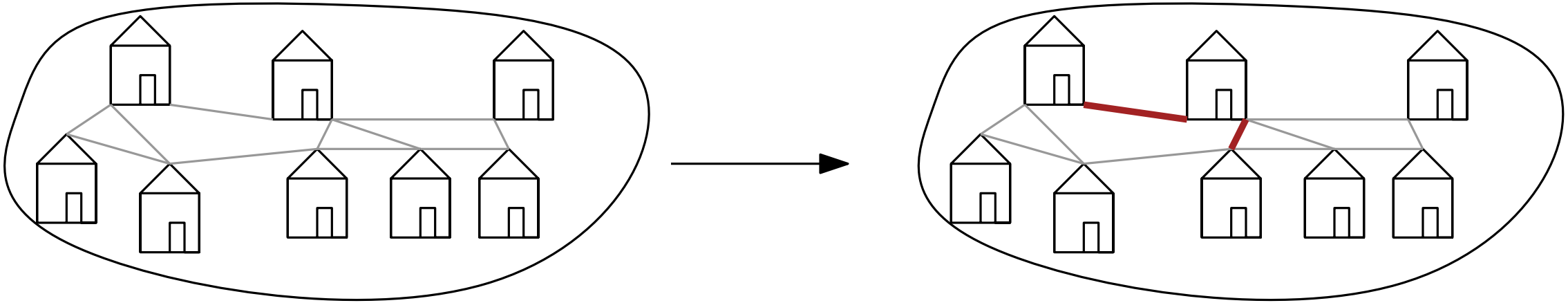


Nachbarschaft

Ihr geplantes Leitungsnetz

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen

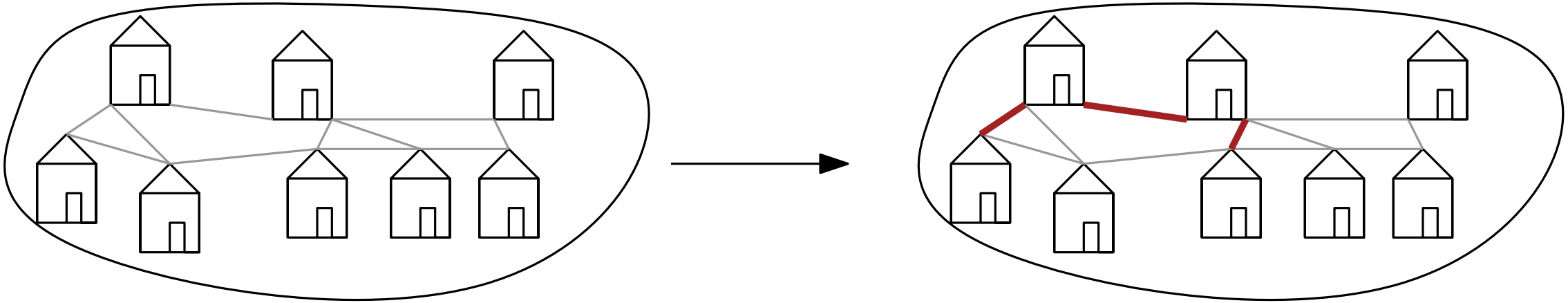


Nachbarschaft

Ihr geplantes Leitungsnetz

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen

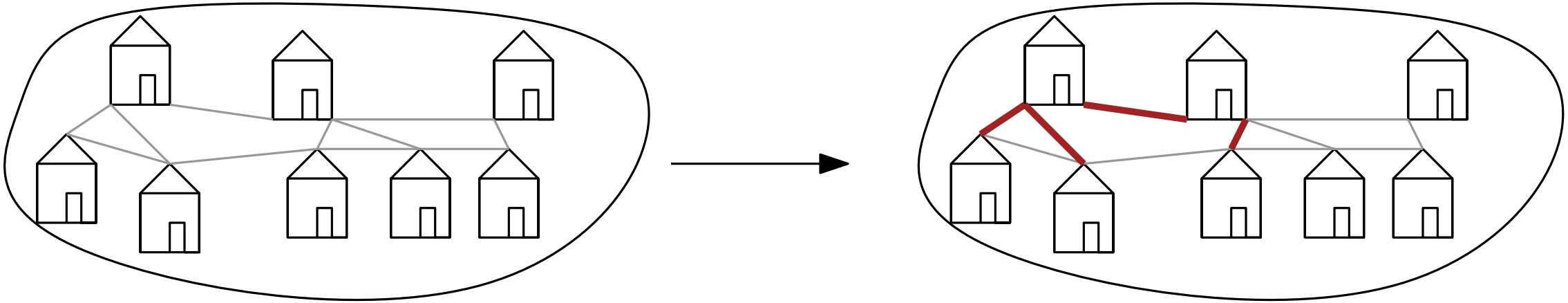


Nachbarschaft

Ihr geplantes Leitungsnetz

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen

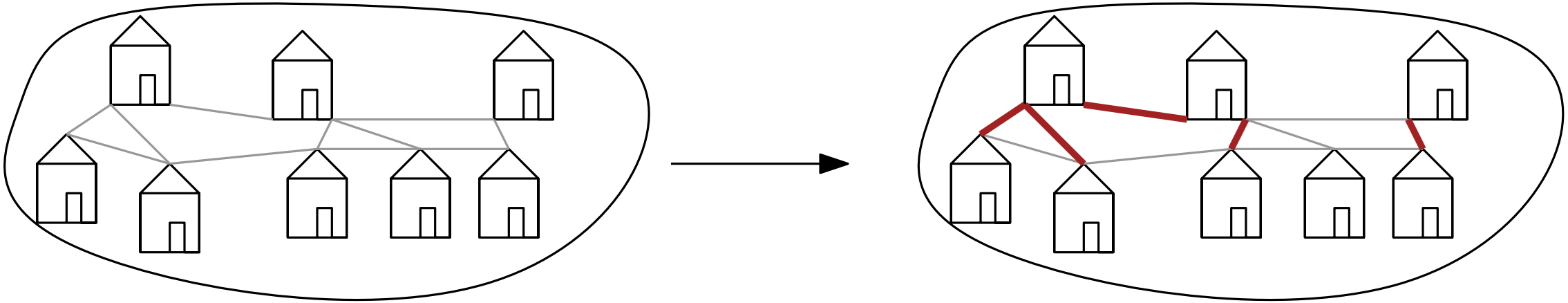


Nachbarschaft

Ihr geplantes Leitungsnetz

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen

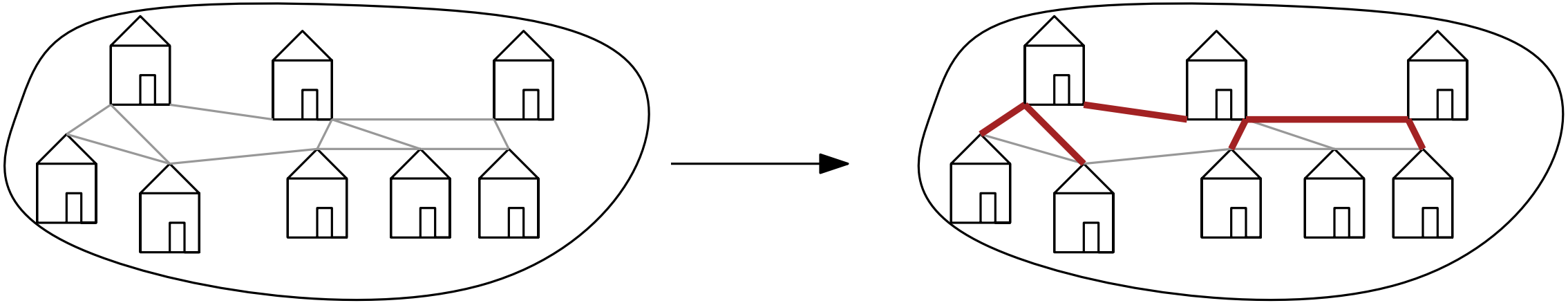


Nachbarschaft

Ihr geplantes Leitungsnetz

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen

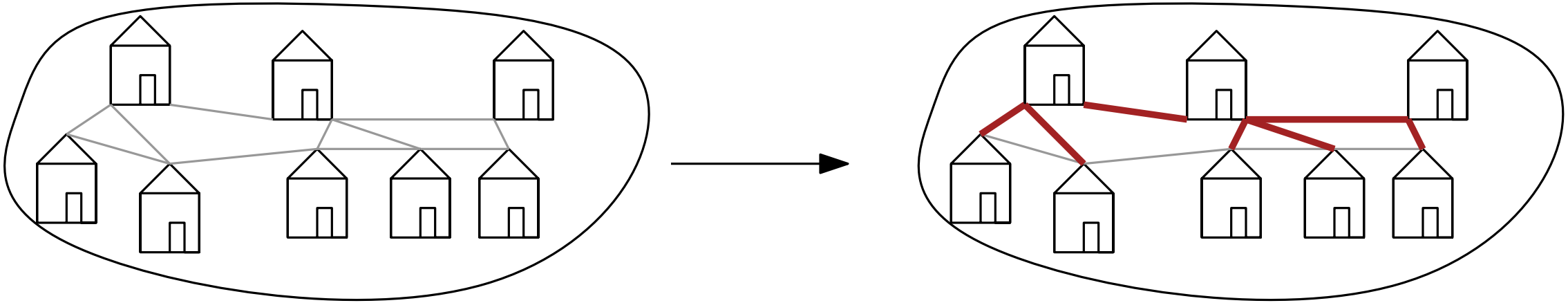


Nachbarschaft

Ihr geplantes Leitungsnetz

Minimale Spannbäume 1

- Nach ihrer Karriere als LKW-Fahrer, wollen sie etwas Neues probieren.
- Natürlich entscheiden sie sich Planner für die Telekom zu werden, ihre Aufgabe wird es sein, neue Leitungen zu planen
- Sie kriegen dabei eine Karte der Nachbarschaft mit den Häusern die einen Anschluss wollen und eine Liste von möglichen Leitungen zwischen den Häusern und die Kosten, diese zu verlegen



Nachbarschaft

Ihr geplantes Leitungsnetz

Minimale Spannbäume 2

- Das kann man als Graphenproblem modellieren!

Minimale Spannbäume 2

- Das kann man als Graphenproblem modellieren!
  - Die Häuser sind Knoten

Minimale Spannbäume 2

- Das kann man als Graphenproblem modellieren!
  - Die Häuser sind Knoten
  - Die möglichen Leitungen und ihre Kosten zwischen den Häusern sind Kanten mit Gewichten die den Kosten entsprechen

Minimale Spannbäume 2

- Das kann man als Graphenproblem modellieren!
  - Die Häuser sind Knoten
  - Die möglichen Leitungen und ihre Kosten zwischen den Häusern sind Kanten mit Gewichten die den Kosten entsprechen
- Dann wollen wir einen minimalen Spannbaum finden, also ein Teilgraph der:

Minimale Spannbäume 2

- Das kann man als Graphenproblem modellieren!
  - Die Häuser sind Knoten
  - Die möglichen Leitungen und ihre Kosten zwischen den Häusern sind Kanten mit Gewichten die den Kosten entsprechen
- Dann wollen wir einen minimalen Spannbaum finden, also ein Teilgraph der:
  - Alle Knoten erreicht

Minimale Spannbäume 2

- Das kann man als Graphenproblem modellieren!
  - Die Häuser sind Knoten
  - Die möglichen Leitungen und ihre Kosten zwischen den Häusern sind Kanten mit Gewichten die den Kosten entsprechen
- Dann wollen wir einen minimalen Spannbaum finden, also ein Teilgraph der:
  - Alle Knoten erreicht
  - Zusammenhängend und kreisfrei ist, also ein Baum

Minimale Spannbäume 2

- Das kann man als Graphenproblem modellieren!
  - Die Häuser sind Knoten
  - Die möglichen Leitungen und ihre Kosten zwischen den Häusern sind Kanten mit Gewichten die den Kosten entsprechen
- Dann wollen wir einen minimalen Spannbaum finden, also ein Teilgraph der:
  - Alle Knoten erreicht
  - Zusammenhängend und kreisfrei ist, also ein Baum
  - Unter allen Spannbäumen derjenige, dessen Summe der Kantengewichte minimal ist

Minimale Spannbäume 3

- Kleine Einschränkung: Alle Kantengewichte einzigartig, also $e_1, e_2 \in E$ und $e_1 \neq e_2$, dann ist $w(e_1) \neq w(e_2)$

Minimale Spannbäume 3

- Kleine Einschränkung: Alle Kantengewichte einzigartig, also $e_1, e_2 \in E$ und $e_1 \neq e_2$, dann ist $w(e_1) \neq w(e_2)$
  - Macht den Spannbaum eindeutig und erleichtert Auswahl.

Minimale Spannbäume 3

- Kleine Einschränkung: Alle Kantengewichte einzigartig, also $e_1, e_2 \in E$ und $e_1 \neq e_2$, dann ist $w(e_1) \neq w(e_2)$
  - Macht den Spannbaum eindeutig und erleichtert Auswahl.
  - Ändert nichts an den Algorithmen, auch für nicht-einzigartige Kantenwichte anwendbar

Minimale Spannbäume 3

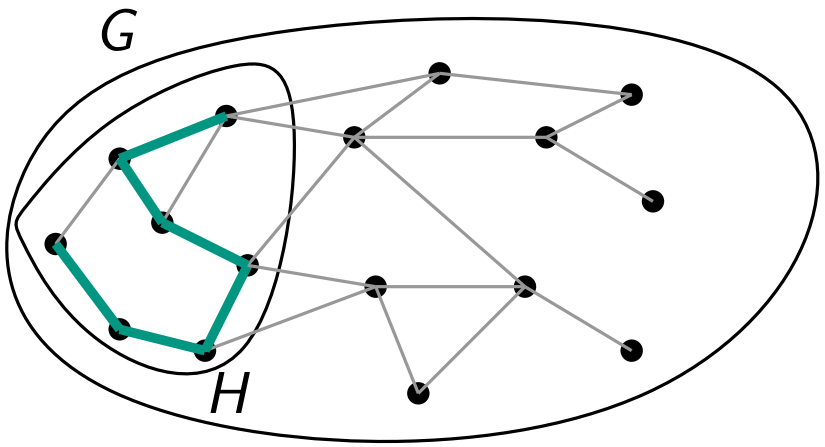
- Kleine Einschränkung: Alle Kantengewichte einzigartig, also $e_1, e_2 \in E$ und $e_1 \neq e_2$, dann ist $w(e_1) \neq w(e_2)$
  - Macht den Spannbaum eindeutig und erleichtert Auswahl.
  - Ändert nichts an den Algorithmen, auch für nicht-einzigartige Kantenwichte anwendbar
- Im Folgenden zwei Greedy-Algorithmen um MSTs zu finden
- MST-Problem ist nett, viele greedy Strategien funktionieren auch

Minimale Spannbäume 4

- Erste Überlegung:

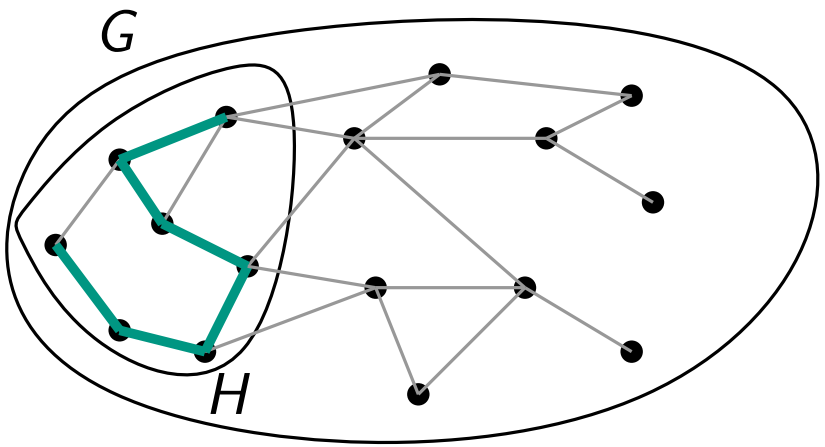
Minimale Spannbäume 4

- Erste Überlegung:
  - Angenommen wir haben schon MST für Teilgraph H



Minimale Spannbäume 4

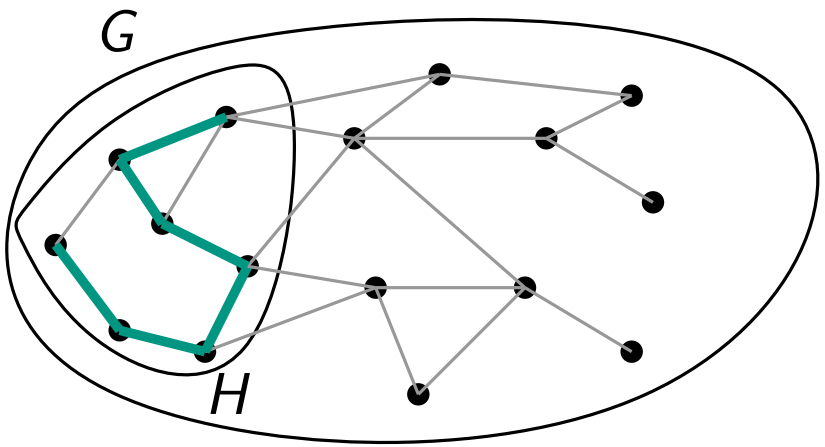
- Erste Überlegung:
  - Angenommen wir haben schon MST für Teilgraph H
  - MST muss alle Knoten erwischen $\Longrightarrow$ wir brauchen mindestens eine Kante zwischen H und V/H



Minimale Spannbäume 4

- Erste Überlegung:
  - Angenommen wir haben schon MST für Teilgraph H
  - MST muss alle Knoten erwischen $\Longrightarrow$ wir brauchen mindestens eine Kante zwischen H und V/H

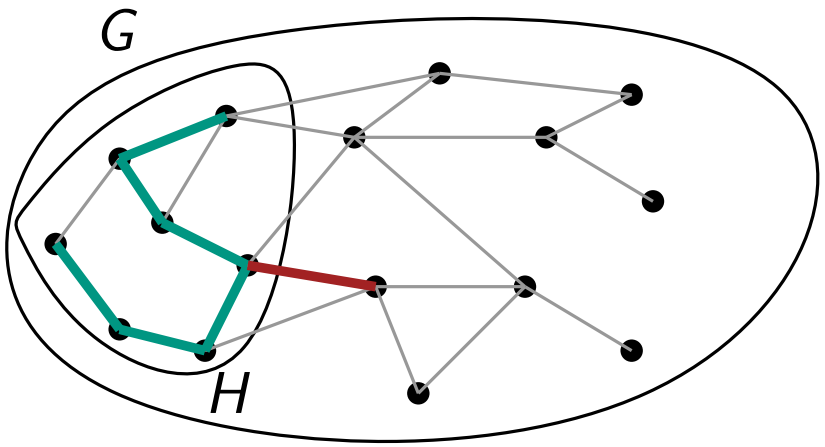
$\Longrightarrow$ Wir wählen die billigste Kante



Minimale Spannbäume 4

- Erste Überlegung:
  - Angenommen wir haben schon MST für Teilgraph H
  - MST muss alle Knoten erwischen $\Longrightarrow$ wir brauchen mindestens eine Kante zwischen H und V/H

$\Longrightarrow$ Wir wählen die billigste Kante

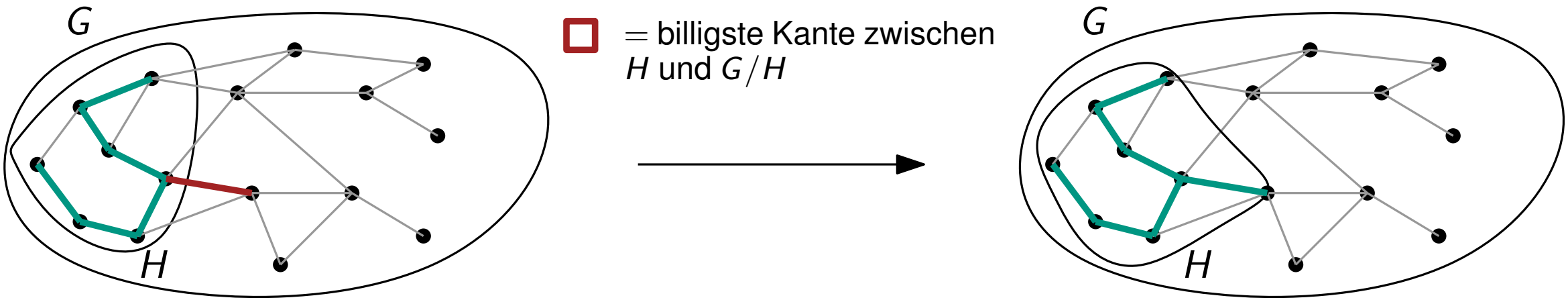


= billigste Kante zwischen H und G/H

Minimale Spannbäume 4

- Erste Überlegung:
  - Angenommen wir haben schon MST für Teilgraph H
  - MST muss alle Knoten erwischen $\Longrightarrow$ wir brauchen mindestens eine Kante zwischen H und V/H

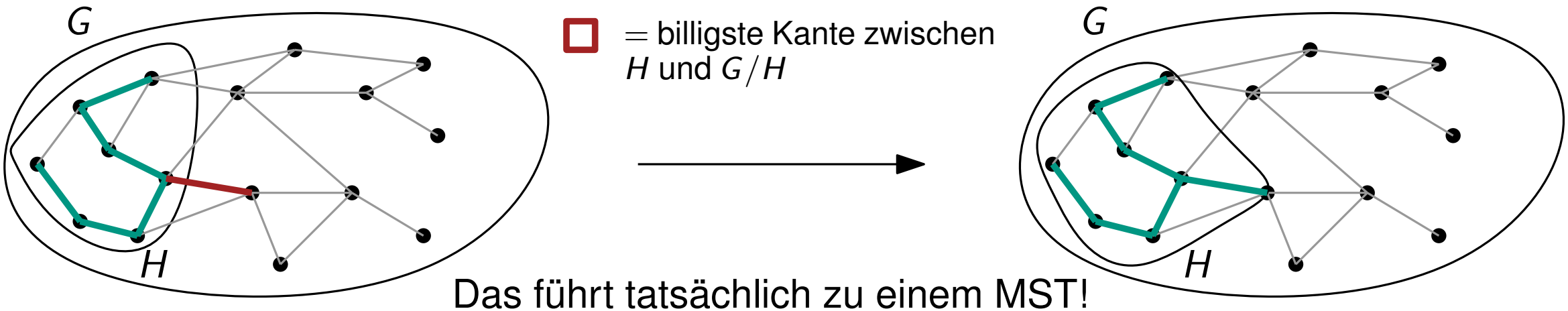
$\Longrightarrow$ Wir wählen die billigste Kante



Minimale Spannbäume 4

- Erste Überlegung:
  - Angenommen wir haben schon MST für Teilgraph H
  - MST muss alle Knoten erwischen $\Longrightarrow$ wir brauchen mindestens eine Kante zwischen H und V/H

$\Longrightarrow$ Wir wählen die billigste Kante



Das führt tatsächlich zu einem MST!

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M
  - Füge die Kante samt Knoten zu M hinzu

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M
  - Füge die Kante samt Knoten zu M hinzu
  - Wiederhole Schritte 2 und 3 bis alle Knoten in MST M

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M
  - Füge die Kante samt Knoten zu M hinzu
  - Wiederhole Schritte 2 und 3 bis alle Knoten in MST M
- Algorithmische Sicht:

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M
  - Füge die Kante samt Knoten zu M hinzu
  - Wiederhole Schritte 2 und 3 bis alle Knoten in MST M
- Algorithmische Sicht:
  - Halte für jeden Knoten nur Elternknoten, Baum ist dann implizit gespeichert

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M
  - Füge die Kante samt Knoten zu M hinzu
  - Wiederhole Schritte 2 und 3 bis alle Knoten in MST M
- Algorithmische Sicht:
  - Halte für jeden Knoten nur Elternknoten, Baum ist dann implizit gespeichert
  - Starte bei beliebigen Knoten v als Wurzel

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M
  - Füge die Kante samt Knoten zu M hinzu
  - Wiederhole Schritte 2 und 3 bis alle Knoten in MST M

- Algorithmische Sicht:
  - Halte für jeden Knoten nur Elternknoten, Baum ist dann implizit gespeichert
  - Starte bei beliebigen Knoten v als Wurzel
  - Suche Knoten mit niedrigestem Abstand zu bisherigem MST

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M
  - Füge die Kante samt Knoten zu M hinzu
  - Wiederhole Schritte 2 und 3 bis alle Knoten in MST M

- Algorithmische Sicht:
  - Halte für jeden Knoten nur Elternknoten, Baum ist dann implizit gespeichert
  - Starte bei beliebigen Knoten v als Wurzel
  - Suche Knoten mit niedrigestem Abstand zu bisherigem MST
  - Füge ihn zum MST hinzu und update neuen Abstand der Nachbarn

Minimale Spannbäume 5

- Unsere Vorgehensweise:
  - Füge beliebigen Knoten v zu MST M hinzu
  - Suche billigste Kante zwischen M und V/M
  - Füge die Kante samt Knoten zu M hinzu
  - Wiederhole Schritte 2 und 3 bis alle Knoten in MST M

- Algorithmische Sicht:
  - Halte für jeden Knoten nur Elternknoten, Baum ist dann implizit gespeichert
  - Starte bei beliebigen Knoten v als Wurzel
  - Suche Knoten mit niedrigestem Abstand zu bisherigem MST
  - Füge ihn zum MST hinzu und update neuen Abstand der Nachbarn
  - Wiederhole bis alle Knoten im MST sind

Minimale Spannbäume 6

- Algorithmische Sicht:
  - Halte für jeden Knoten nur Elternknoten, Baum ist dann implizit gespeichert
  - Starte bei beliebigen Knoten v als Wurzel
  - Suche Knoten mit niedrigestem Abstand zu bisherigem MST
  - Füge ihn zum MST hinzu und update neuen Abstand der Nachbarn
  - Wiederhole bis alle Knoten im MST sind

Minimale Spannbäume 6

- Algorithmische Sicht:
  - Halte für jeden Knoten nur Elternknoten, Baum ist dann implizit gespeichert
  - Starte bei beliebigen Knoten v als Wurzel
  - Suche Knoten mit niedrigestem Abstand zu bisherigem MST
  - Füge ihn zum MST hinzu und update neuen Abstand der Nachbarn
  - Wiederhole bis alle Knoten im MST sind

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

Minimale Spannbäume 6

- Algorithmische Sicht:
  - Halte für jeden Knoten nur Elternknoten, Baum ist dann implizit gespeichert
  - Starte bei beliebigen Knoten v als Wurzel
  - Suche Knoten mit niedrigestem Abstand zu bisherigem MST
  - Füge ihn zum MST hinzu und update neuen Abstand der Nachbarn
  - Wiederhole bis alle Knoten im MST sind

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

Minimale Spannbäume 7

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

- Wie schnell geht das?

Minimale Spannbäume 7

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

- Wie schnell geht das?
  - Jeder Knoten wird ein mal aus der PriorityQueue entfernt
    $\Longrightarrow$ $\Theta(n)$ mal popMIN

Minimale Spannbäume 7

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

- Wie schnell geht das?
  - Jeder Knoten wird ein mal aus der PriorityQueue entfernt
    $\Longrightarrow$ $\Theta(n)$ mal popMIN
  - Für jede Kante wird maximal 1 mal decPrio aufgerufen
    $\Longrightarrow$ $\Theta(m)$ mal decPRIO

Minimale Spannbäume 7

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

- Wie schnell geht das?
  - Jeder Knoten wird ein mal aus der PriorityQueue entfernt
    $\Longrightarrow \Theta(n)$ mal popMIN
  - Für jede Kante wird maximal 1 mal decPrio aufgerufen
    $\Longrightarrow \Theta(m)$ mal decPRIO

$\Longrightarrow \Theta(n \log(n) + m \log(n))$

Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet

Minimale Spannbäume 8

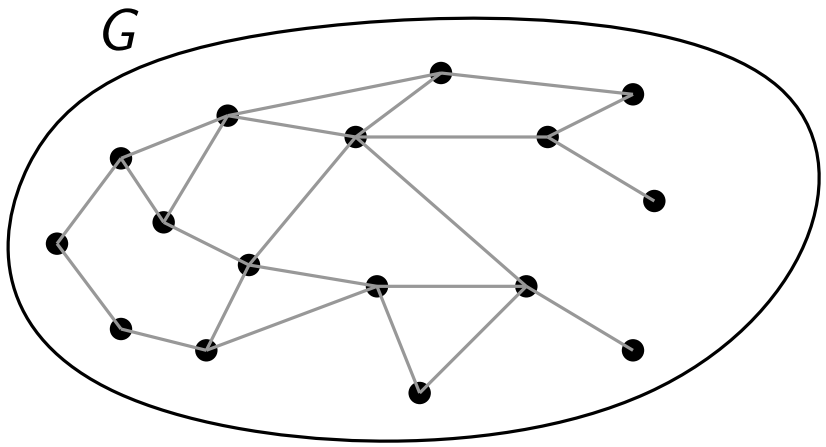
- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben

Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen

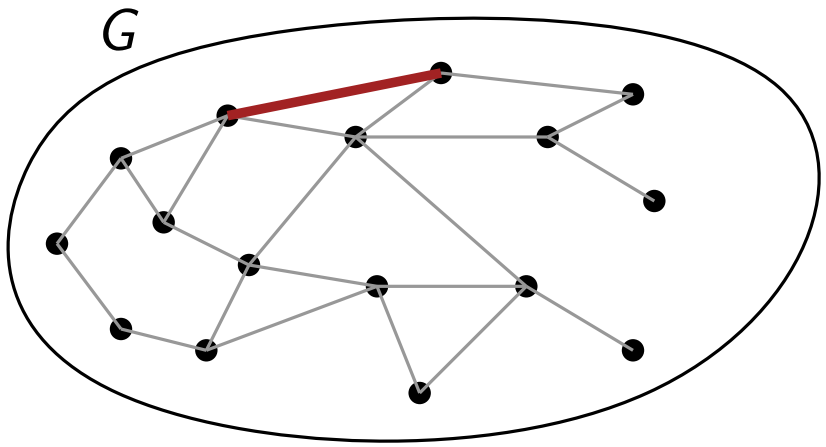
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



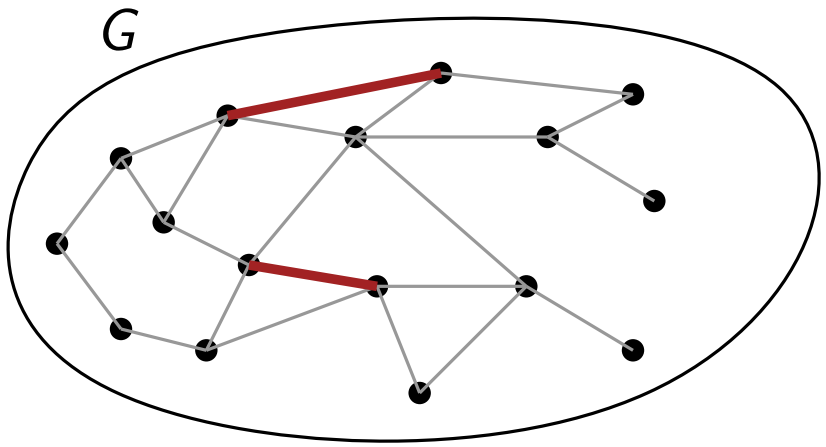
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



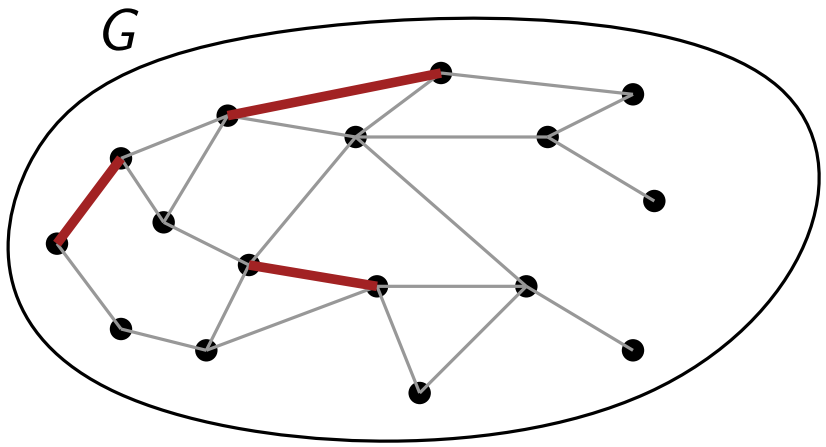
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



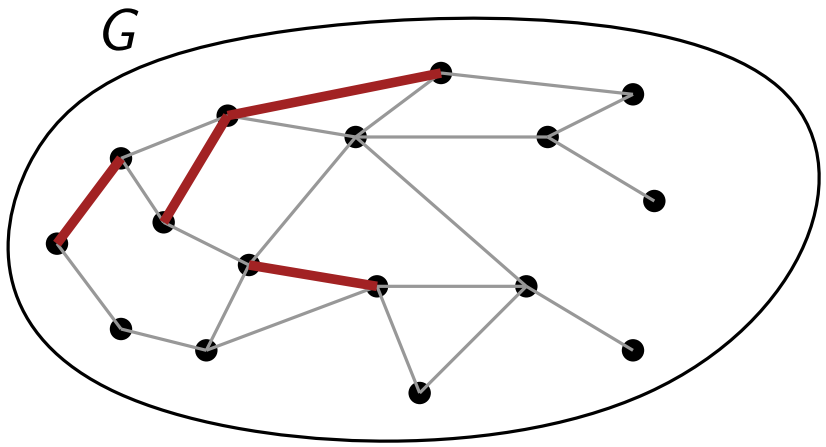
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



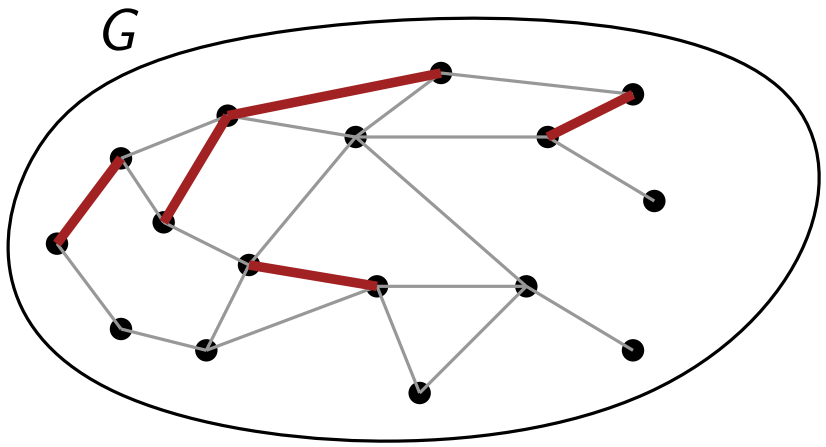
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



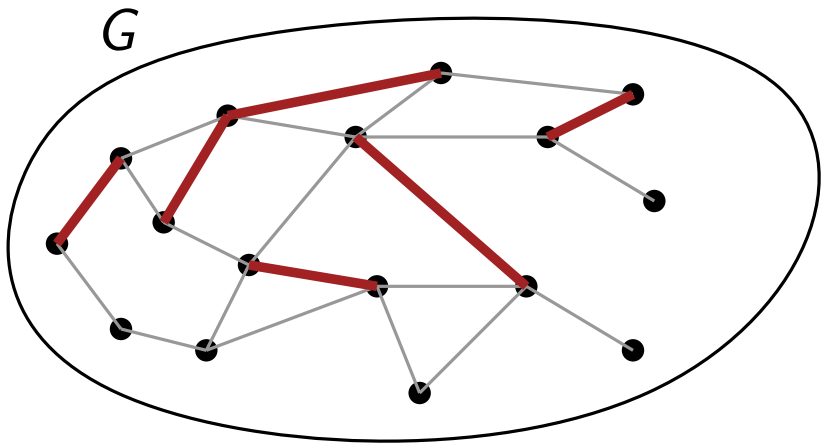
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



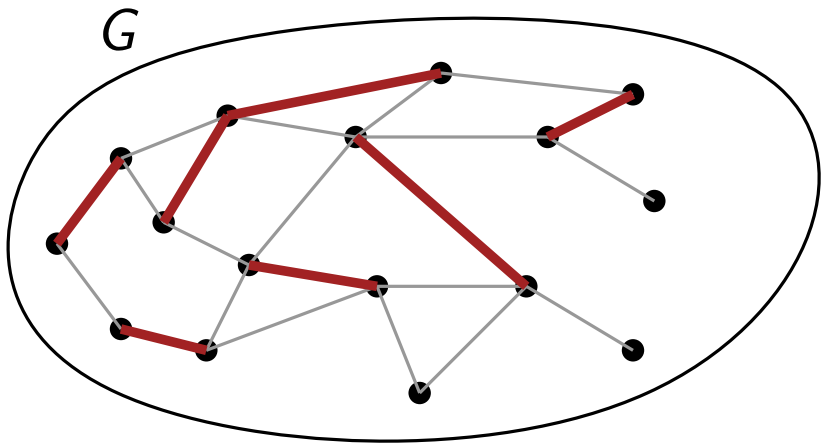
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



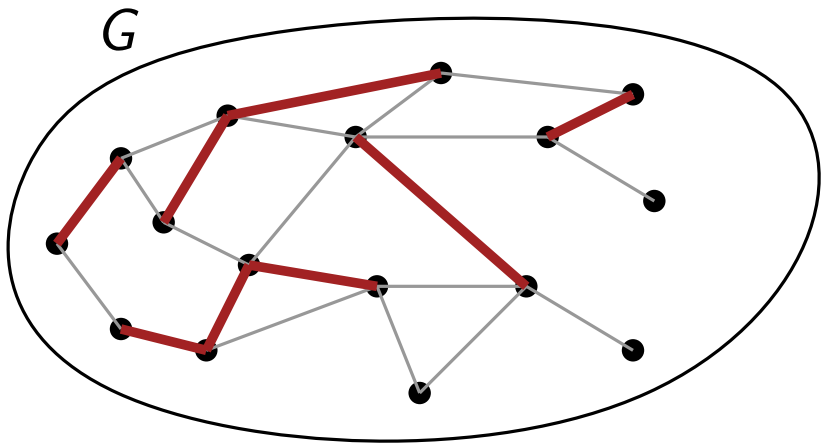
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



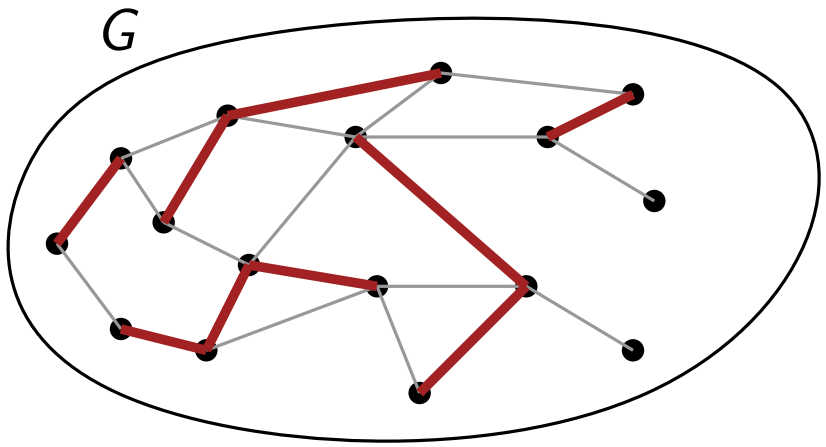
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



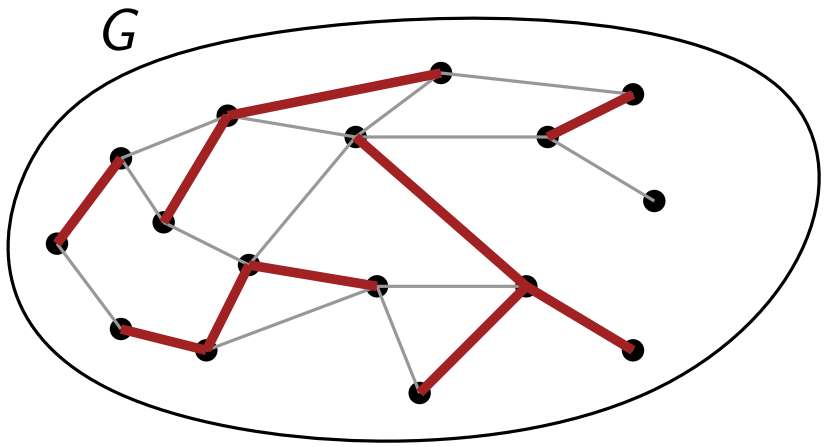
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



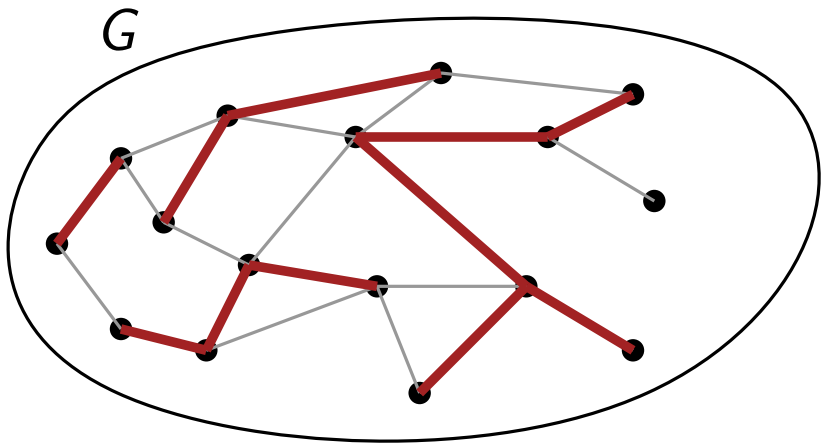
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



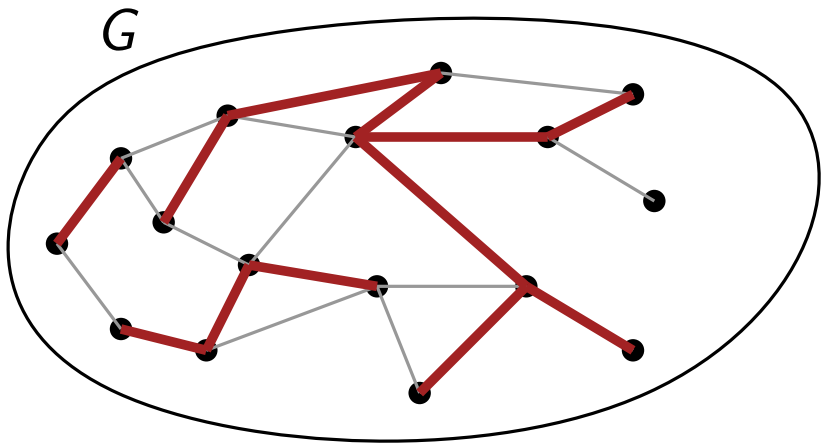
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



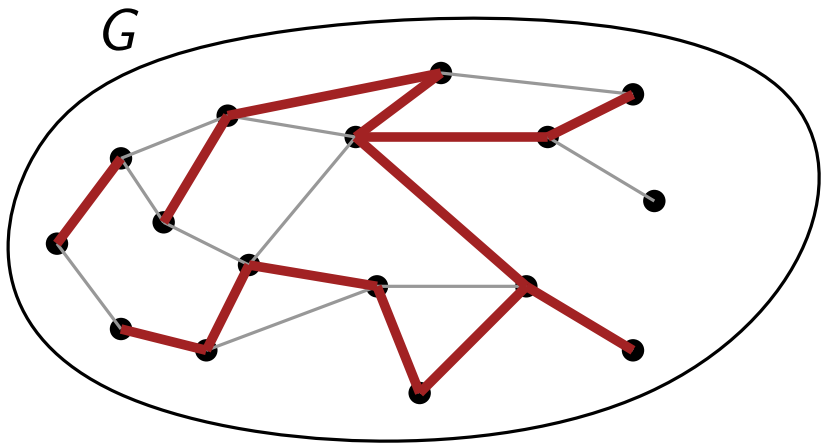
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



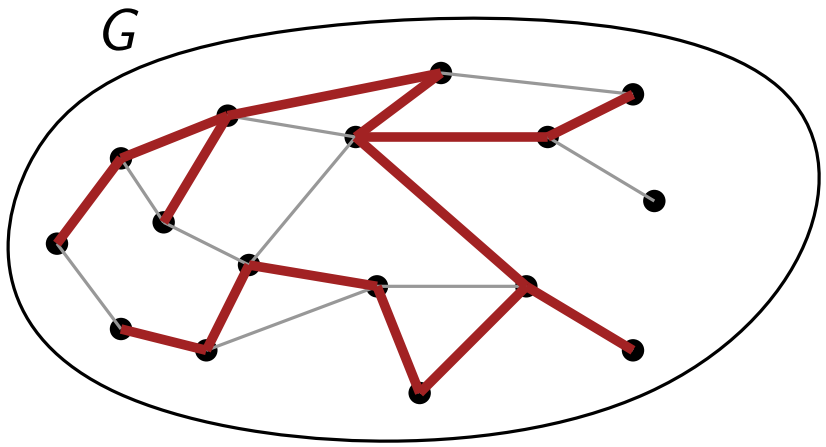
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



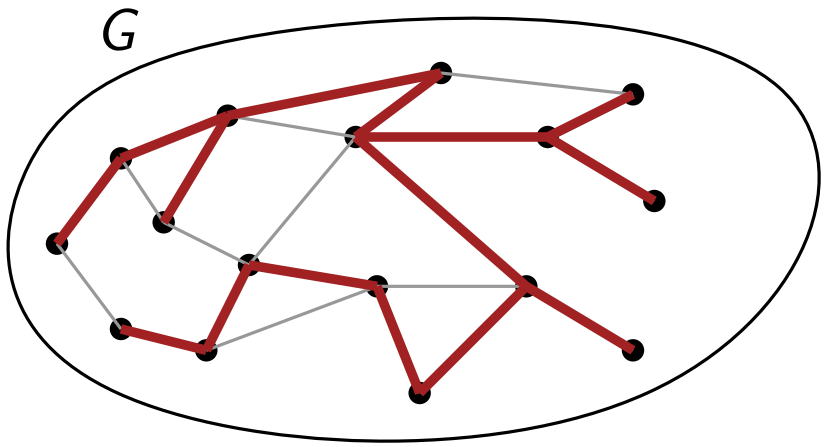
Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



Minimale Spannbäume 8

- Neue Strategie: Wähle aus ganzem Graph immer billigste Kante die keinen Kreis bildet
  - Wir bauen MST nicht von einer Seite auf, sondern können viele Zusammenhangskomponenten haben
  - Führt aber zu MST, da Kanten immer billig gewählt werden und wir nie einen Kreis bilden, also Baum bekommen



Minimale Spannbäume 9

- Konkrete Strategie:

Minimale Spannbäume 9

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt

Minimale Spannbäume 9

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt
  - Falls sie Kreis schließt, schmeiße sie weg, die Kante wird auch später noch einen Kreis schließen

Minimale Spannbäume 9

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt
  - Falls sie Kreis schließt, schmeiße sie weg, die Kante wird auch später noch einen Kreis schließen
  - Falls sie keinen Kreis schließt, füge Kante zu MST hinzu

Minimale Spannbäume 9

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt
  - Falls sie Kreis schließt, schmeiße sie weg, die Kante wird auch später noch einen Kreis schließen
  - Falls sie keinen Kreis schließt, füge Kante zu MST hinzu
  - Wiederhole bis $n - 1$ Kanten gewählt wurden

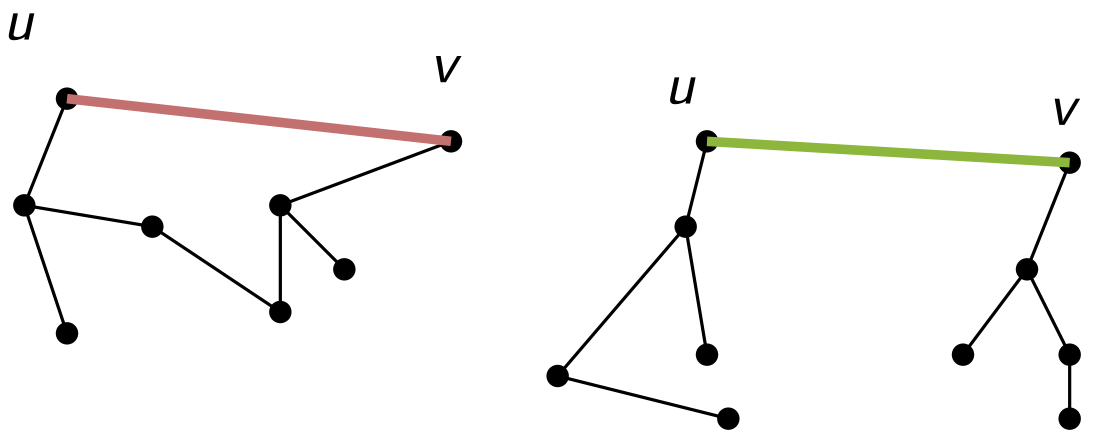
Minimale Spannbäume 9

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt
  - Falls sie Kreis schließt, schmeiße sie weg, die Kante wird auch später noch einen Kreis schließen
  - Falls sie keinen Kreis schließt, füge Kante zu MST hinzu
  - Wiederhole bis $n - 1$ Kanten gewählt wurden
- Wie kann man entscheiden ob Kante einen Kreis schließt?

Minimale Spannbäume 9

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt
  - Falls sie Kreis schließt, schmeiße sie weg, die Kante wird auch später noch einen Kreis schließen
  - Falls sie keinen Kreis schließt, füge Kante zu MST hinzu
  - Wiederhole bis $n - 1$ Kanten gewählt wurden

- Wie kann man entscheiden ob Kante einen Kreis schließt?
  - Falls u und v schon in gleicher Zusammenhangskomponente sind, schließt Kante einen Kreis, ansonsten nicht



Minimale Spannbäume 10

- Überprüfen, ob v und u in der gleichen Zusammenhangskomponente liegen?

Minimale Spannbäume 10

- Überprüfen, ob v und u in der gleichen Zusammenhangskomponente liegen?
  - Das geht doch mit Union-Find!

Minimale Spannbäume 10

- Überprüfen, ob v und u in der gleichen Zusammenhangskomponente liegen?
  - Das geht doch mit Union-Find!
  - Falls v und u in der gleichen Komponente sind, so ist $find(v) = find(u)$

Minimale Spannbäume 10

- Überprüfen, ob v und u in der gleichen Zusammenhangskomponente liegen?
  - Das geht doch mit Union-Find!
  - Falls v und u in der gleichen Komponente sind, so ist $find(v) = find(u)$
  - Falls sie es nicht sind und wir Kante $\{u, v\}$ wählen, so sind sie es danach, also vereinigen wir die beiden Komponenten mit $union(v, u)$

Minimale Spannbäume 11

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt
  - Falls sie Kreis schließt, schmeiße sie weg, die Kante wird auch später noch einen Kreis schließen
  - Falls sie keinen Kreis schließt, füge Kante zu MST hinzu
  - Wiederhole bis $n - 1$ Kanten gewählt wurden

Minimale Spannbäume 11

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt
  - Falls sie Kreis schließt, schmeiße sie weg, die Kante wird auch später noch einen Kreis schließen
  - Falls sie keinen Kreis schließt, füge Kante zu MST hinzu
  - Wiederhole bis $n - 1$ Kanten gewählt wurden

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

Minimale Spannbäume 11

- Konkrete Strategie:
  - Suche immer billigste Kante $\{u, v\}$ und prüfe ob sie Kreis schließt
  - Falls sie Kreis schließt, schmeiße sie weg, die Kante wird auch später noch einen Kreis schließen
  - Falls sie keinen Kreis schließt, füge Kante zu MST hinzu
  - Wiederhole bis $n - 1$ Kanten gewählt wurden

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

Minimale Spannbäume 12

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

- Wie schnell geht das?

Minimale Spannbäume 12

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

- Wie schnell geht das?
  - Maximal für jede Kante wird zweimal FIND aufgerufen
    $\Longrightarrow$ $2m$ mal FIND

Minimale Spannbäume 12

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

- Wie schnell geht das?
  - Maximal für jede Kante wird zweimal FIND aufgerufen
    $\Longrightarrow$ $2m$ mal FIND
  - Maximal $n-1$ mal können Zusammenhangskomponenten verbunden werden bis alle verbunden sind
    $\Longrightarrow$ $n-1$ mal UNION

Minimale Spannbäume 12

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

- Wie schnell geht das?
  - Maximal für jede Kante wird zweimal FIND aufgerufen
    $\Longrightarrow$ $2m$ mal FIND
  - Maximal $n-1$ mal können Zusammenhangskomponenten verbunden werden bis alle verbunden sind
    $\Longrightarrow$ $n-1$ mal UNION
  - Maximal m mal eine Kante poppen
    $\Longrightarrow$ m mal popMIN

$\Longrightarrow \Theta(2m \cdot \log^*(n) + (n-1) \cdot \log^*(n) + m \cdot \log(m)) = \Theta(m \log(m))$