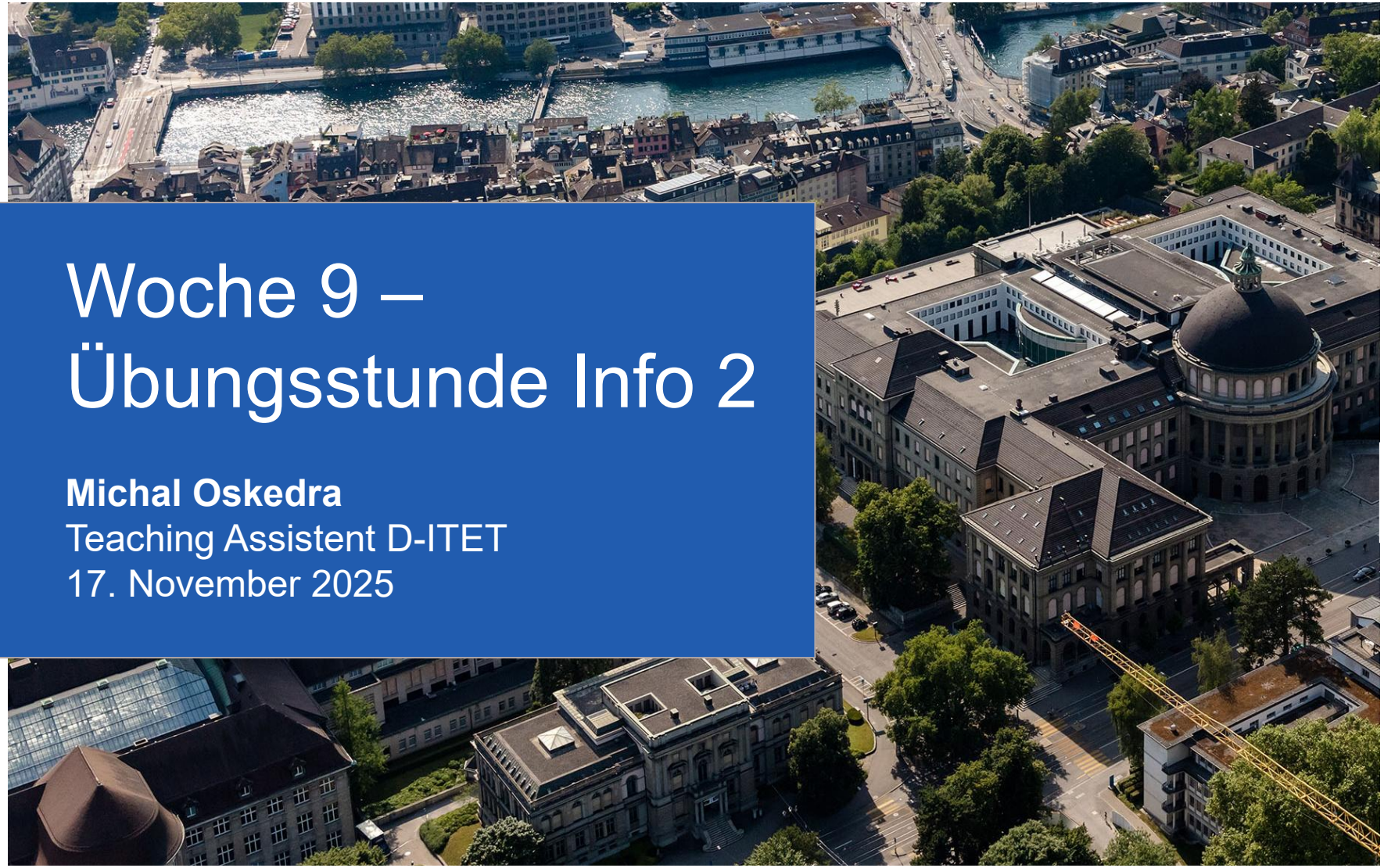
ETH zürich

Info2 - ML J 34.1 14-16

WhatsApp group



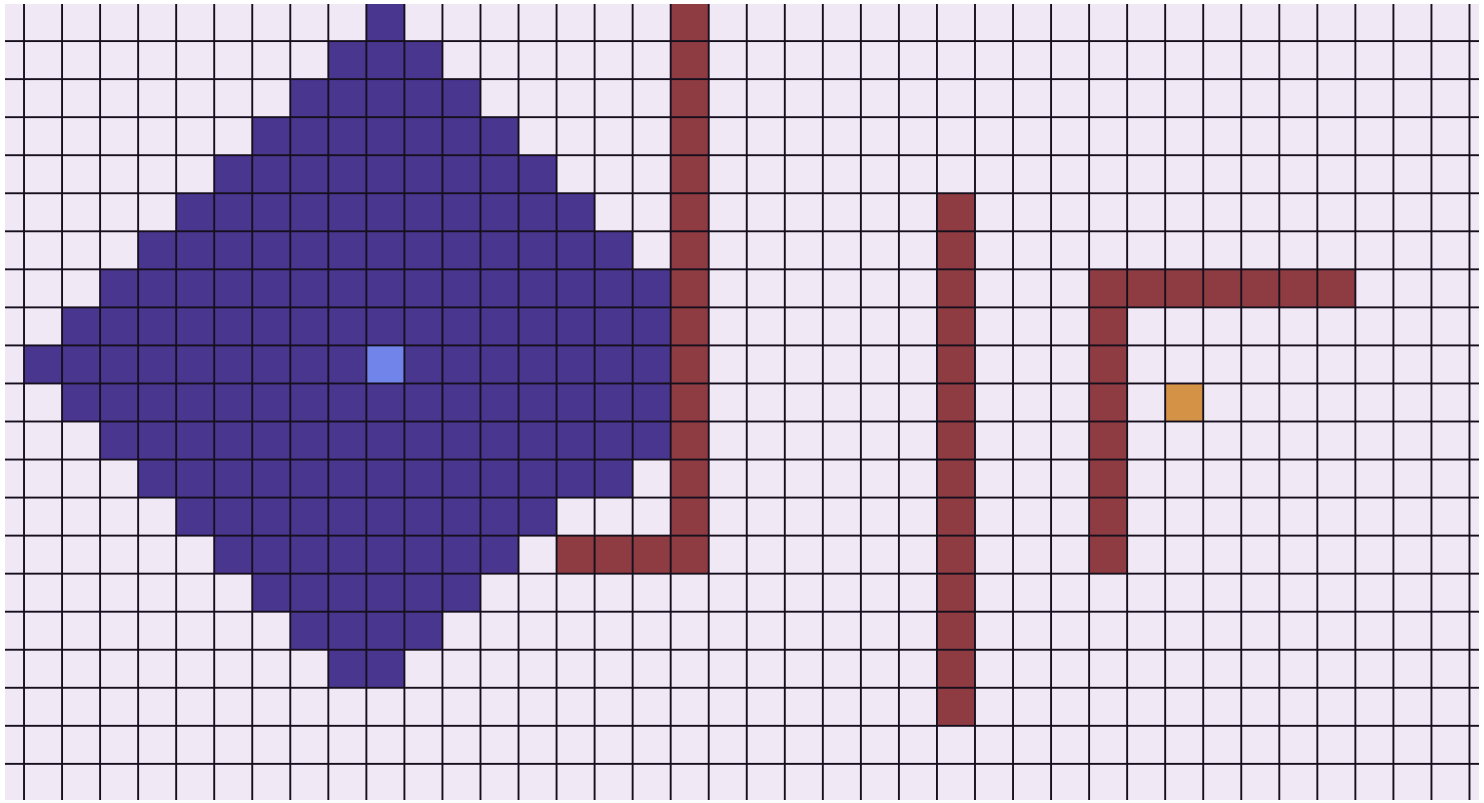
Woche 9 – Übungsstunde Info 2

Michal Oskedra
Teaching Assistent D-ITET
17. November 2025

Webseite:
moskedra-web.ethz.ch

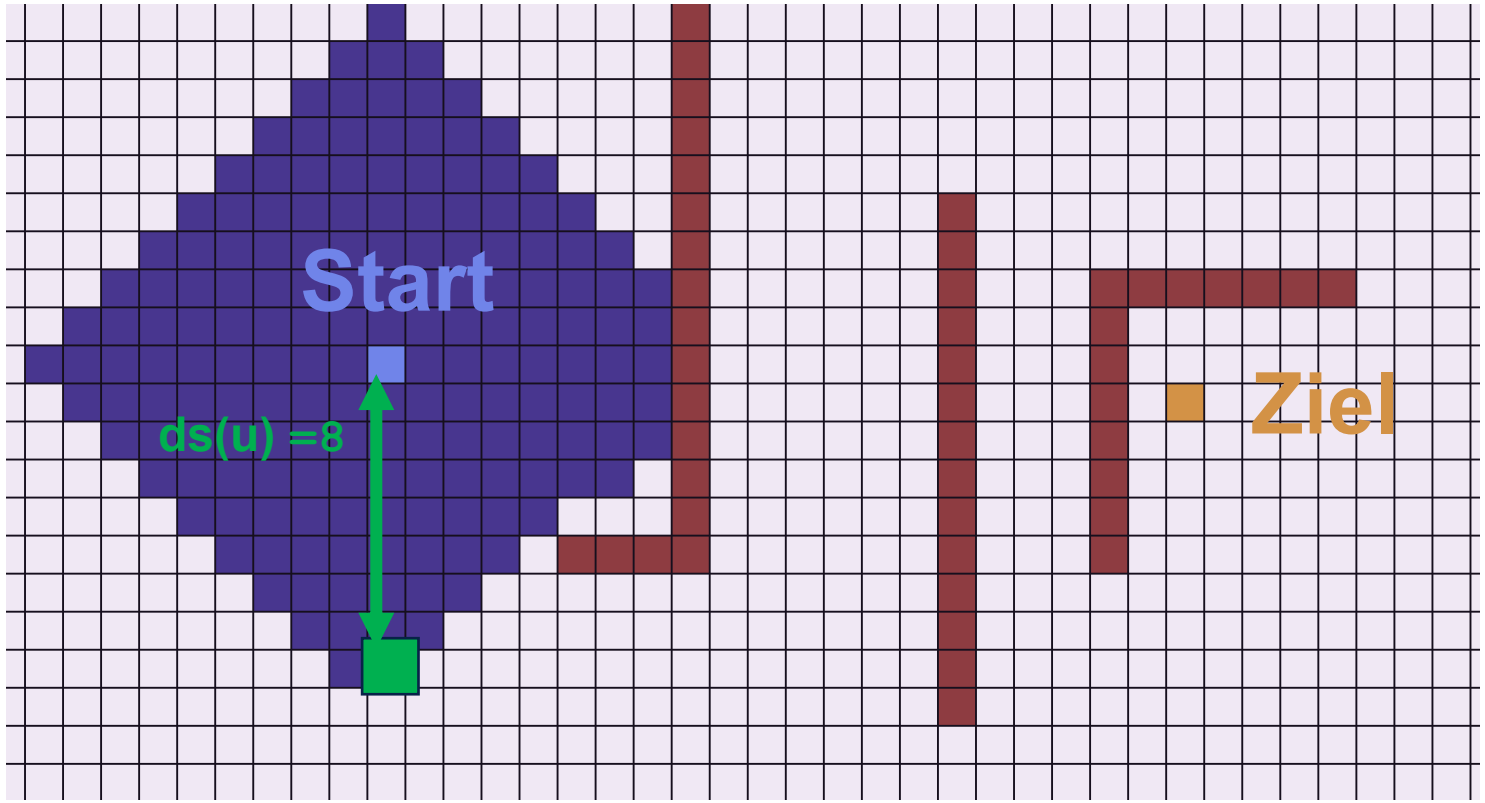
01 – Theorie
01.1 A*
01.2 Kruskal
01.3 Jarnik (Prim)
01.4 Traveling Salesman

Wie kann Dijkstra “gerichteter” vorangehen?



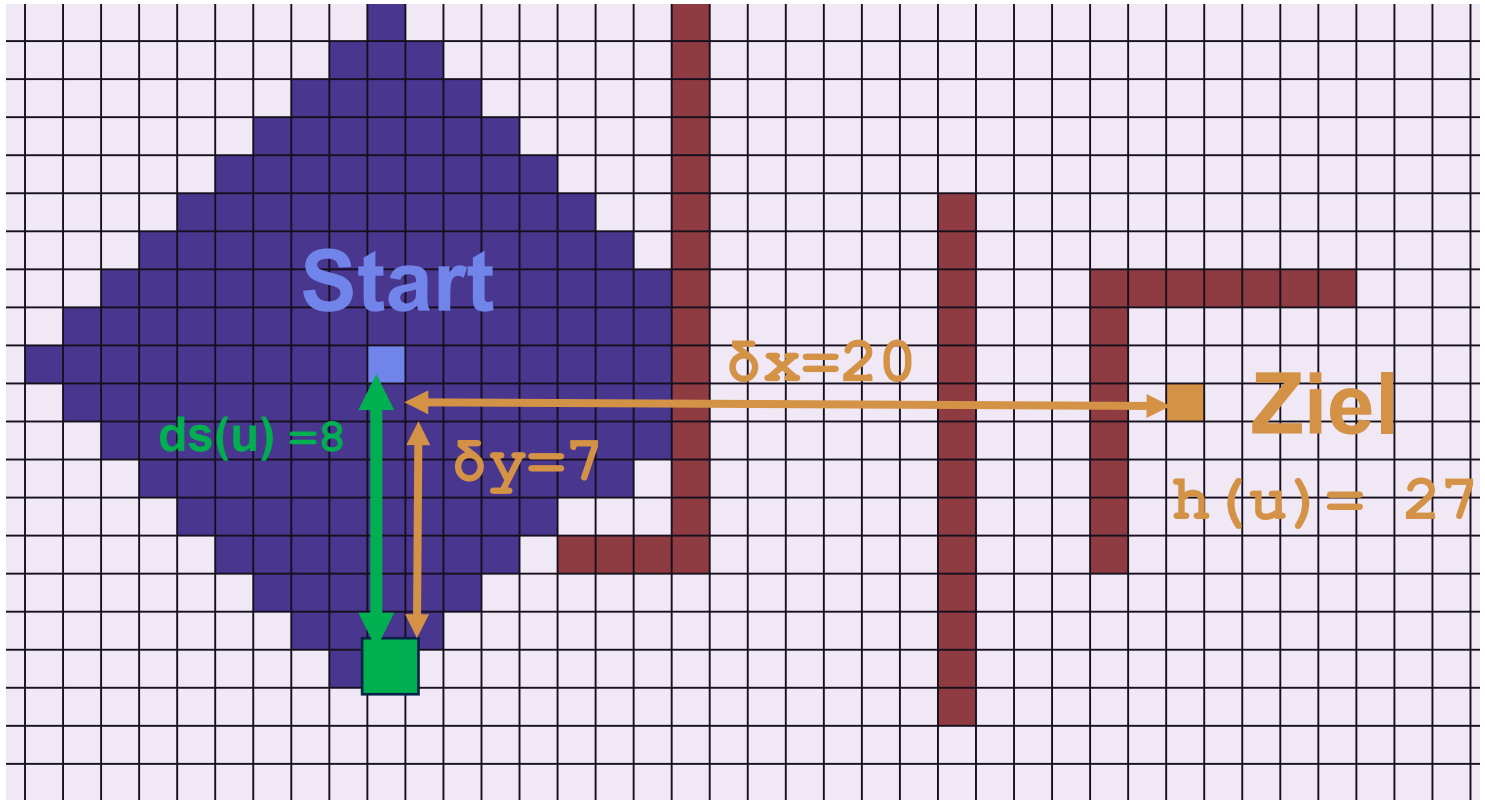
https://pathfinding-visualizer.ivan-sem.com/en

Wie kann Dijkstra “gerichteter” vorangehen?



Dijkstra: Knoten priorisieren nach der Distanz vom Start. Priority Queue: `f(u)=ds(u)`

Wie kann Dijkstra “gerichteter” vorangehen?



Beispiel Heuristik:
`h(u)= δx+δy`

Dijkstra: Knoten priorisieren nach der Distanz vom Start. Priority Queue: `f(u)=ds(u)`

A*: Knoten priorisieren nach der Distanz ds vom Start und einer Heuristik `h`, das die Distanz von Ziel abschätzt. Priority Queue: `f(u) = ds(u) + h(u)`

A* vs Dijkstra

A*-Algorithmus($G, s, t, \hat{h}$)

Input: Positiv gewichteter Graph $G = (V, E, c)$, Startpunkt $s \in V$, Endpunkt $t \in V$, Schätzung $\hat{h}(v) \leq \delta(v, t)$

Output: Existenz und Wert eines kürzesten Weges von s nach t

foreach $u \in V$ **do**
  $d[u] \leftarrow \infty$; $\hat{f}[u] \leftarrow \infty$; $\pi[u] \leftarrow$ null
$d[s] \leftarrow 0$; $\hat{f}[s] \leftarrow \hat{h}(s)$; $U \leftarrow \{s\}$
while $U \neq \emptyset$ **do**
  $u \leftarrow \text{ExtractMin}_{\hat{f}}(U)$
  if $u = t$ **then return** success
  foreach $v \in N^+(u)$ with $d[v] > d[u] + c((u, v))$ **do**
    $d[v] \leftarrow d[u] + c((u, v))$; $\hat{f}[v] \leftarrow d[v] + \hat{h}(v)$; $\pi[v] \leftarrow u$
    $U \leftarrow U \cup \{v\}$
return failure

Algorithmus: Dijkstra(G, s)

Input: Positiv gewichteter Graph $G = (V, E, c)$, Startpunkt $s \in V$

Output: Länge d_s der kürzesten Pfade von s und Vorgänger π_s für jeden Knoten

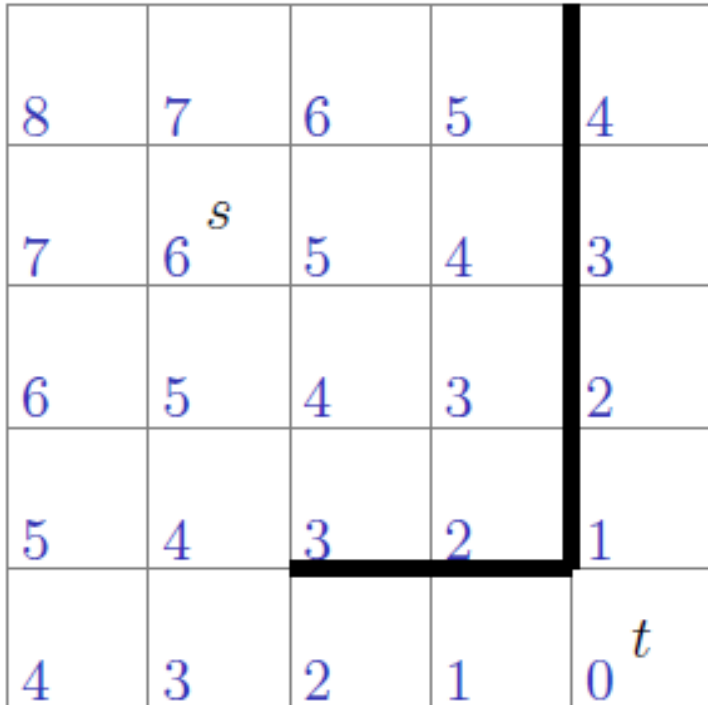
foreach $u \in V$ **do**
  $d_s[u] \leftarrow \infty$; $\pi_s[u] \leftarrow$ **null**
$d_s[s] \leftarrow 0$; $U \leftarrow \{s\}$
while $U \neq \emptyset$ **do**
  $u \leftarrow \arg\min_{u \in U} d_s[u]$; $U \leftarrow U \setminus \{u\}$
  foreach $v \in N^+(u)$ **do**
    if $d_s[u] + c(u, v) < d_s[v]$ **then**
      $d_s[v] \leftarrow d_s[u] + c(u, v)$
      $\pi_s[v] \leftarrow u$
      $U \leftarrow U \cup \{v\}$

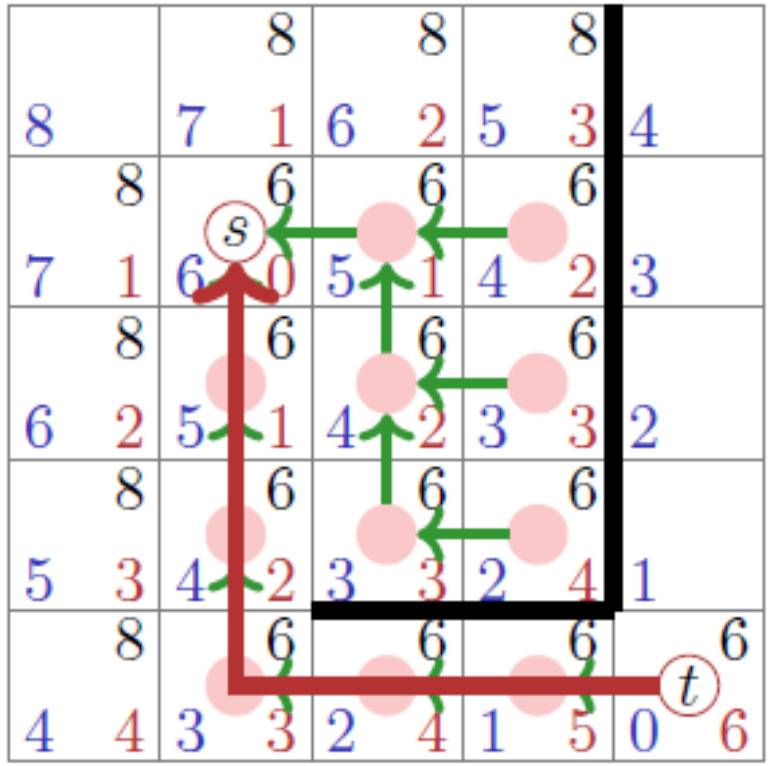
Voraussetzungen für A*

- Positiv gewichteter Graph $\forall Kantengewichte > 0$
- Zusammenhängender Graph für s und t
- Unterschätzung: $h(v) \leq \delta(v, t)$
- Monotonie: $\mathrm{h(u)} \leq \mathrm{c((u, v))} + \mathrm{h(v)}$

Korrekte Heuristik

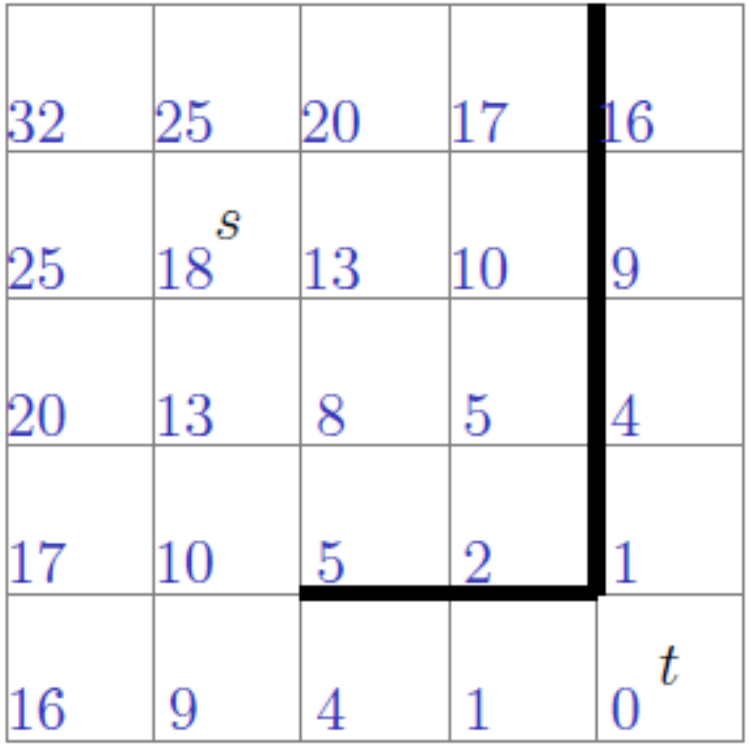
$$\hat{f}(u) = d_s[u] + \hat{h}(u) \qquad (\hat{h} = \delta_x + \delta_y)$$

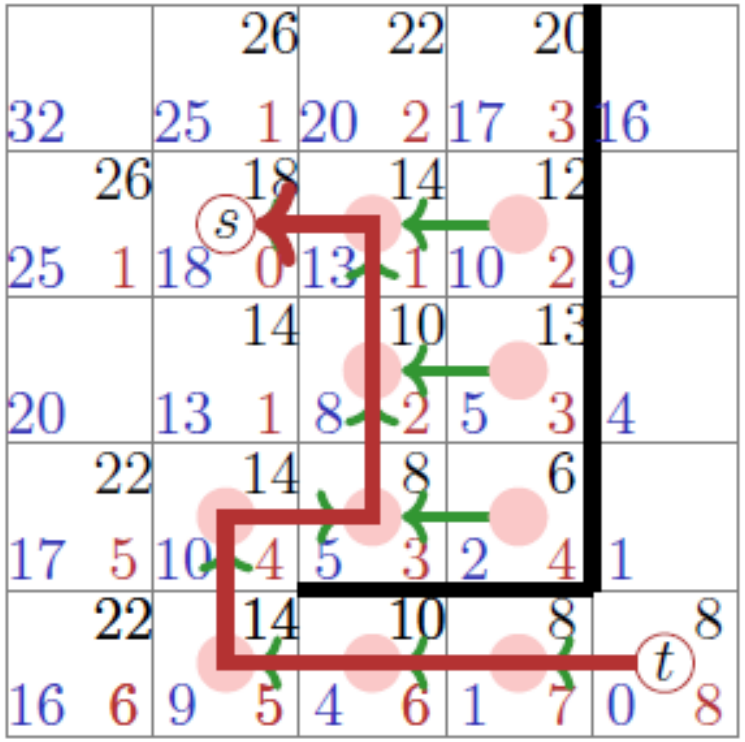




Unterschätzung verletzt

$$\hat{f}(u) = d_s[u] + \hat{h}(u) \quad (\hat{h} = \delta_x^2 + \delta_y^2) \quad \exists v \colon \hat{h}(v) > \delta(v,t)$$

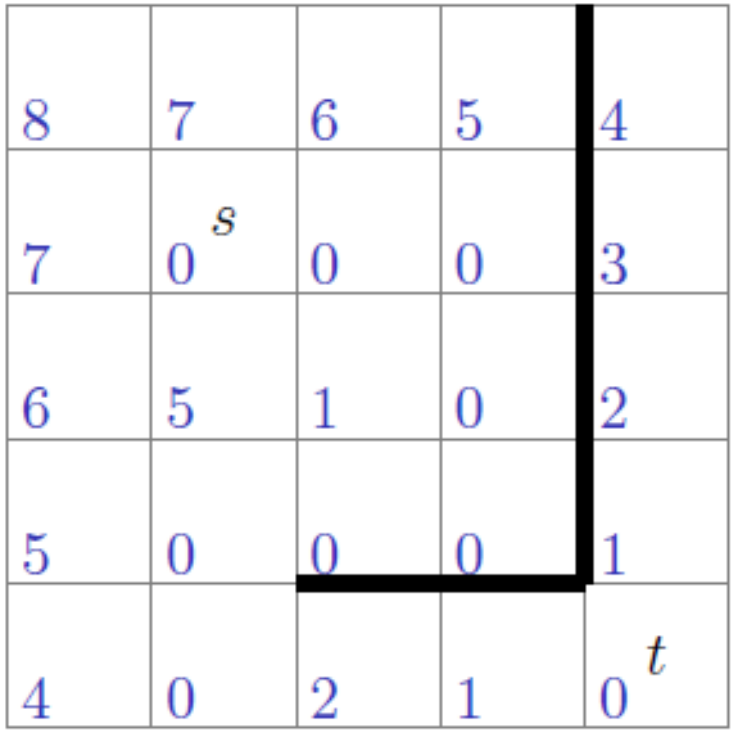


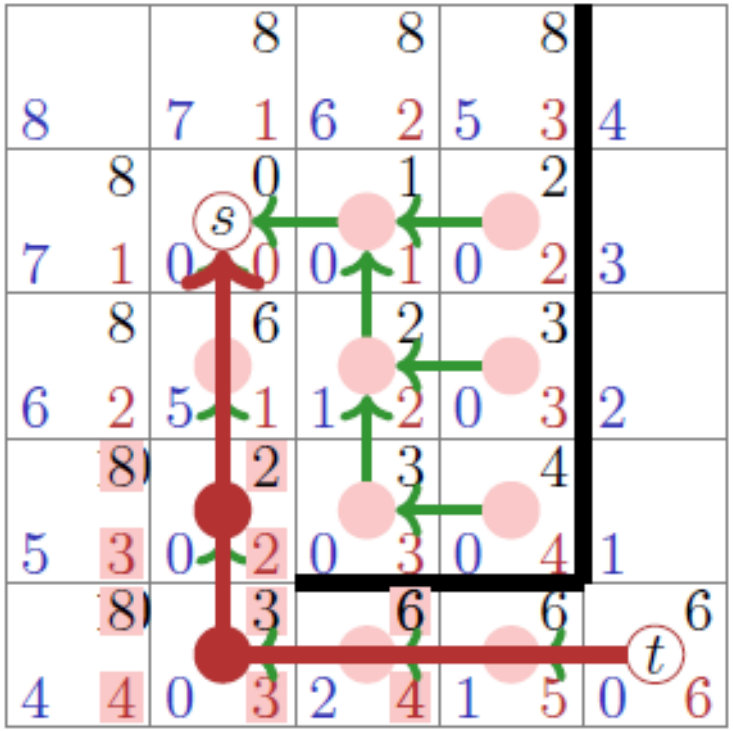


Monotonie verletzt

$$\hat{f}(u) = d_s[u] + \hat{h}(u)$$

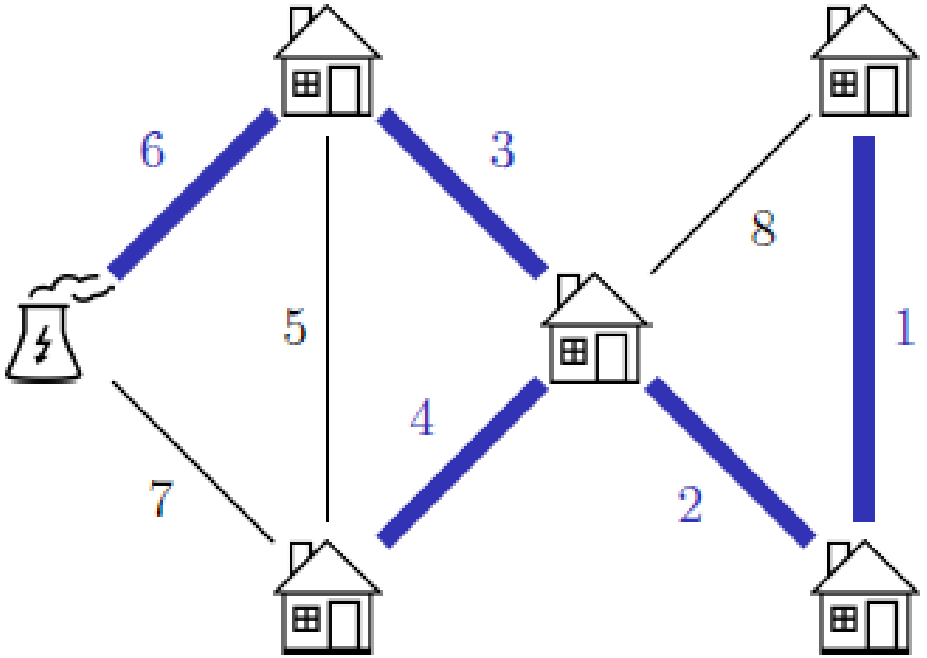
$$\forall (u,v) \in E\colon \hat{h}(u) \geq c((u,v)) + \hat{h}(v)$$



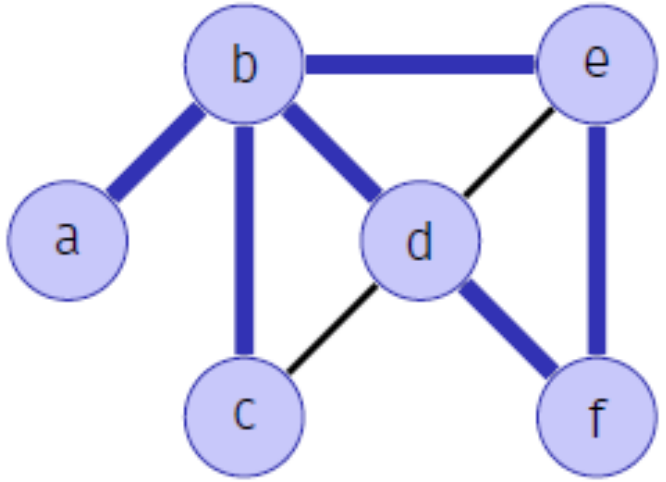


01 – Theorie
01.1 A*
01.2 Kruskal
01.3 Jarnik (Prim)
01.4 Traveling Salesman

Minimale Spannbäume



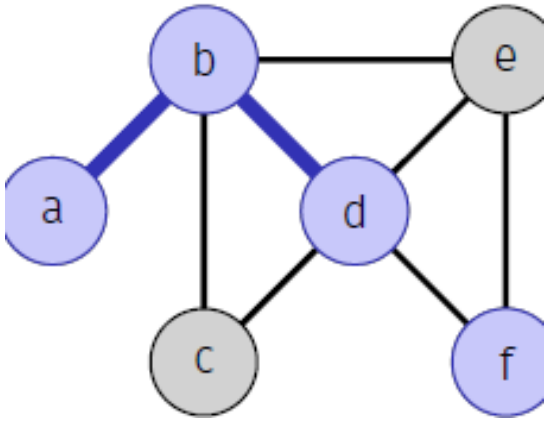
Minimale Spannbäume



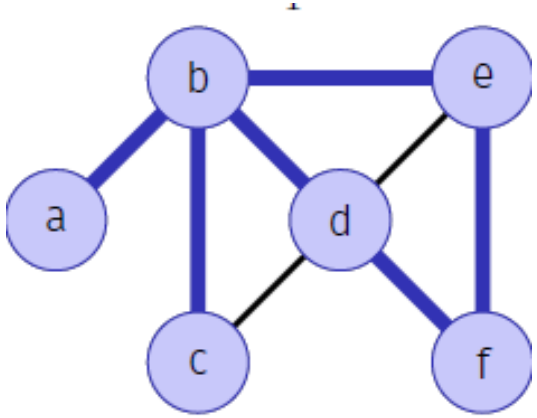
Spannbaum von G**:** Teilgraph $T = (V', E')$ mit $V' \subseteq V, E' \subseteq E$ so dass

- Spann-: $V' = V$ (spannt alle Knoten auf)
- Baum: zusammenhängend

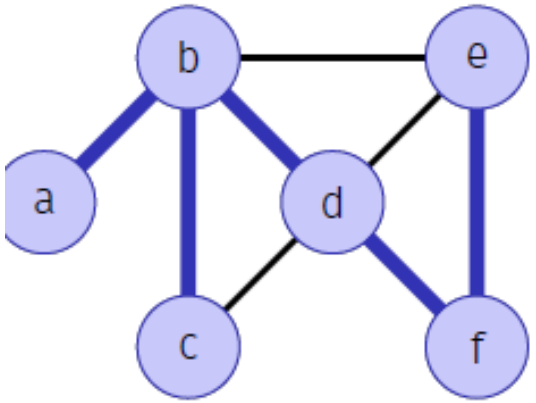
Minimale Spannbäume



Nicht alle Knoten verbunden

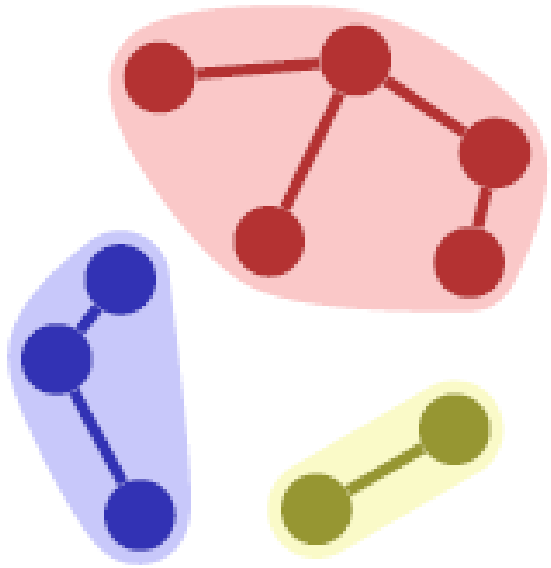


Kreise



Minimaler Spannbaum

Kruskal Algorithmus für Minimale Spannbäume

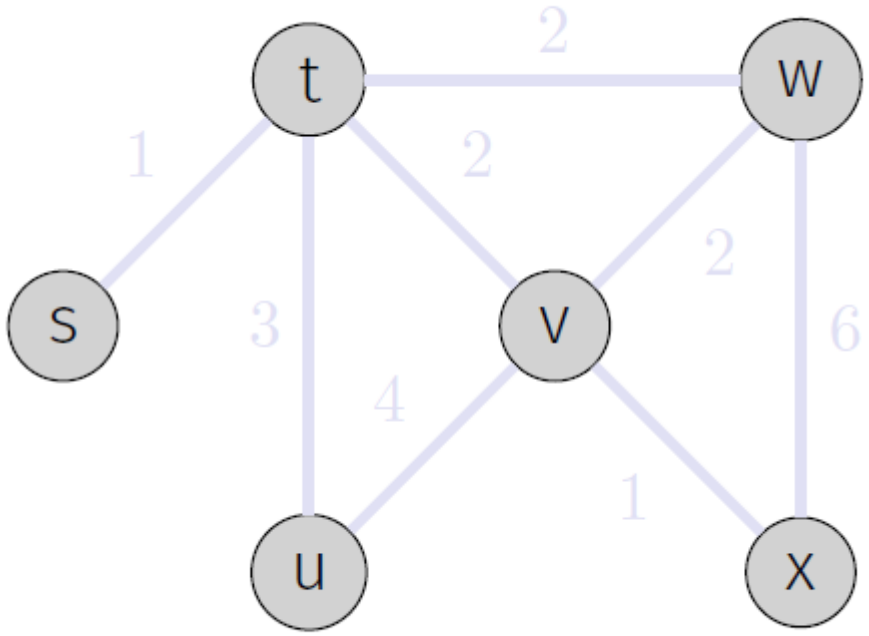


Gegeben eine Menge von zusammenhängenden Minimalen Spannbäumen

Greedy approach: Nimm die nächste Kante mit kleinsten Gewicht:

a) Verbindet die Kante zwei Knoten aus der Gleichen Menge → Ignorieren
b) Verbindet die Kante zwei Knoten aus verschiedenen Mengen → Kante einfügen und Mengen vereinigen

Kruskal Algorithmus



Komponenten:

s

t

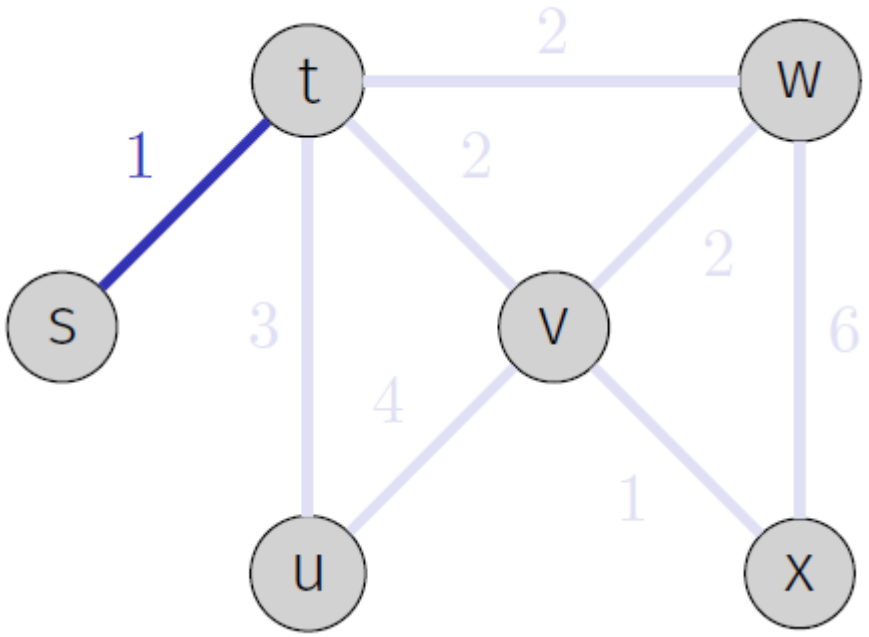
u

v

w

x

Kruskal Algorithmus



Komponenten:

s, t

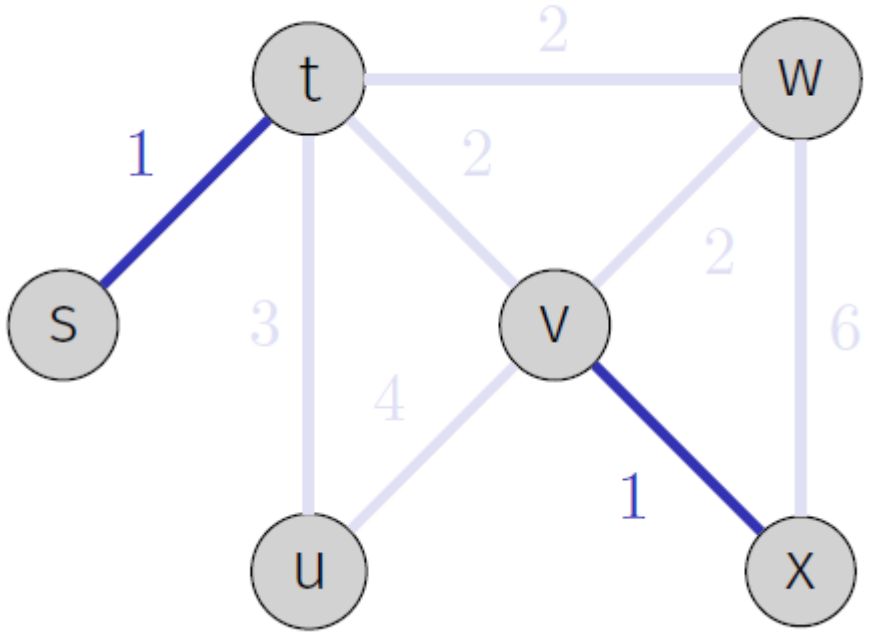
u

v

w

x

Kruskal Algorithmus



Komponenten:

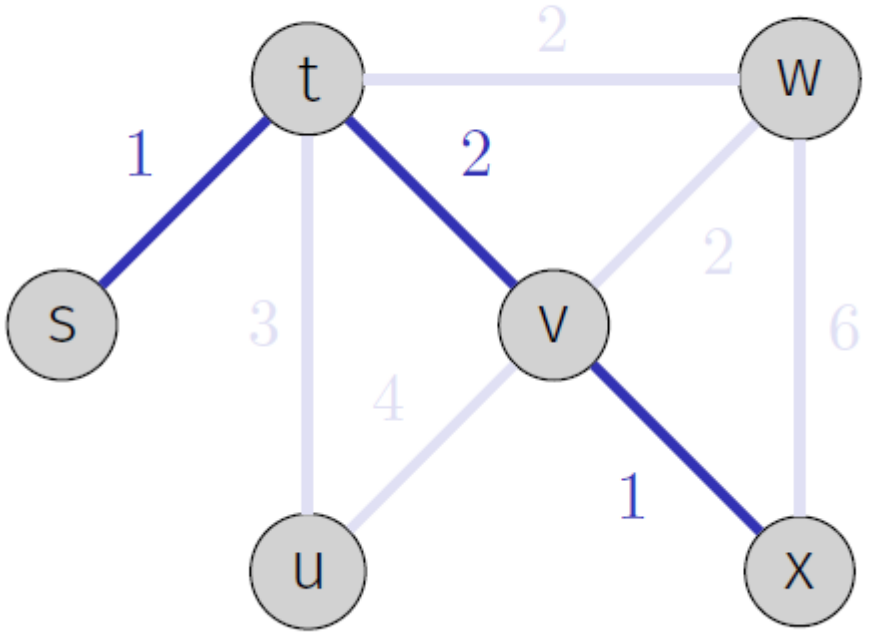
s, t

u

v, x

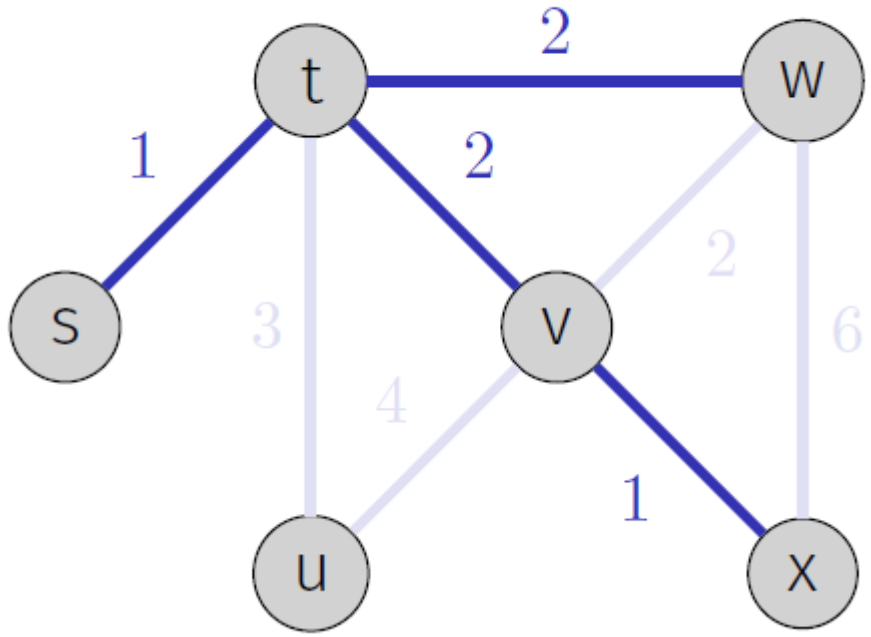
w

Kruskal Algorithmus



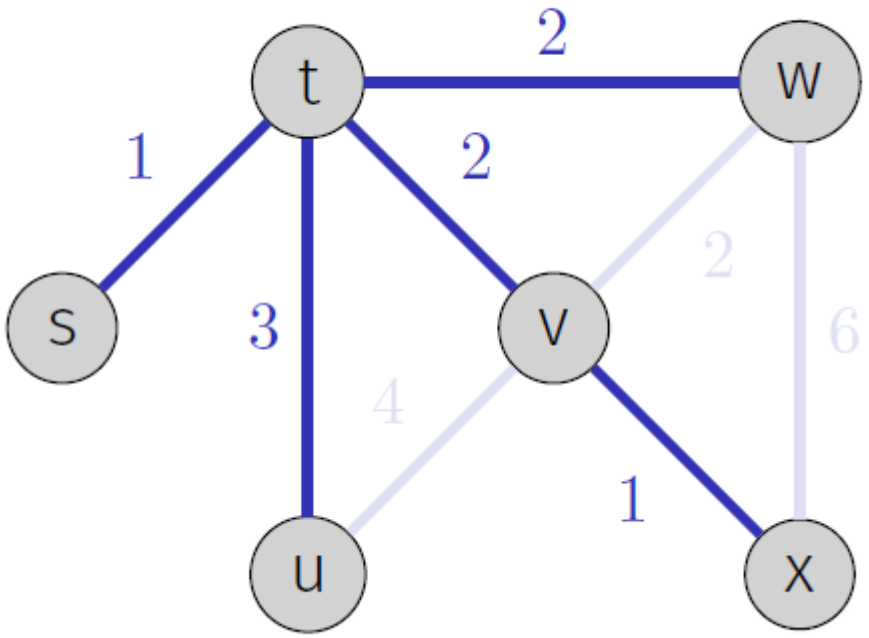
Komponenten:
s, t, v, x
u
w

Kruskal Algorithmus



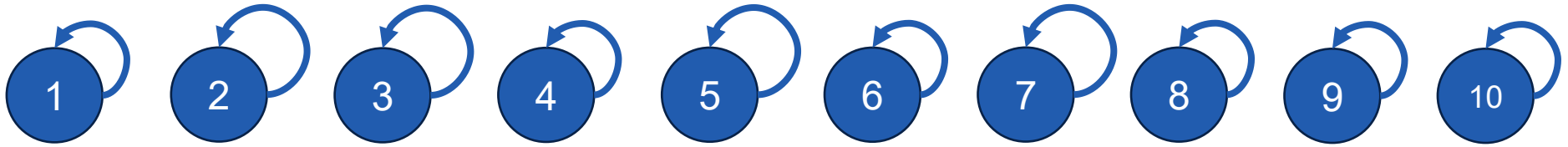
Komponenten:
s, t, v, w, x
u

Kruskal Algorithmus



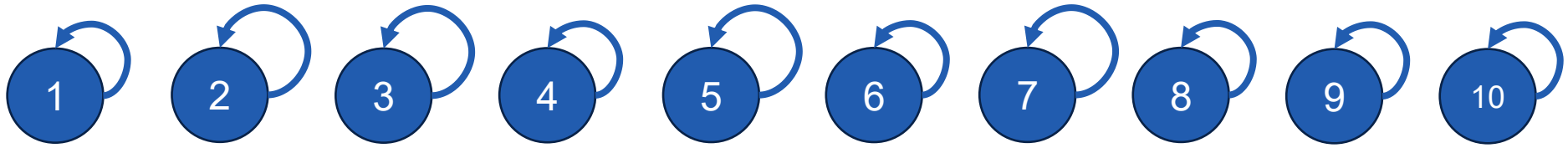
Komponenten:
s, t, u, v, w, x

Wie können wir die Knoten gruppieren?



Parent	1	2	3	4	5	6	7	8	9	10
Index	1	2	3	4	5	6	7	8	9	10

Antwort: Die Gruppen der Knoten werden in einer Tabelle gespeichert. Der Parent repräsentiert die Gruppe. Wird eine Verkettung Knoten am Ende den gleichen Parent haben, gehören sie der gleichen Gruppe

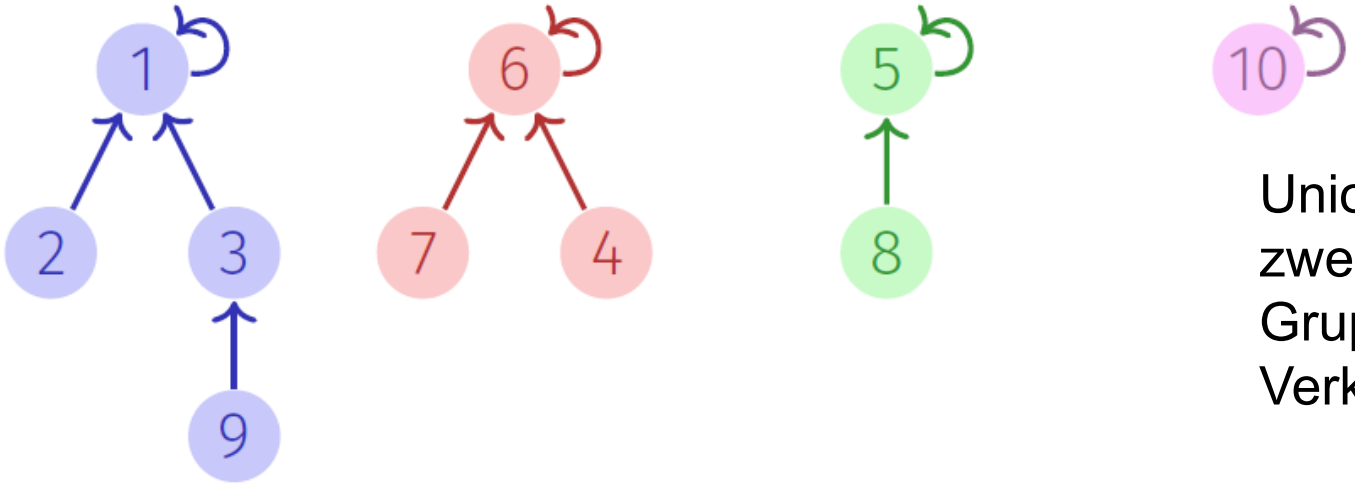


UnionFind(2,1),
UnionFind(3,1),
UnionFind(9,3)

UnionFind(7,6),
UnionFind(4,6)

UnionFind(8,5)

UnionFind(9,1)

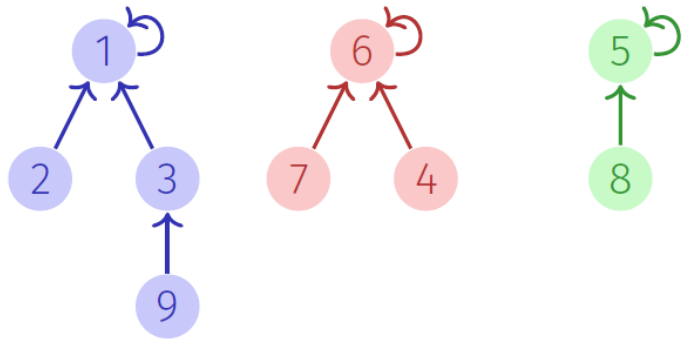


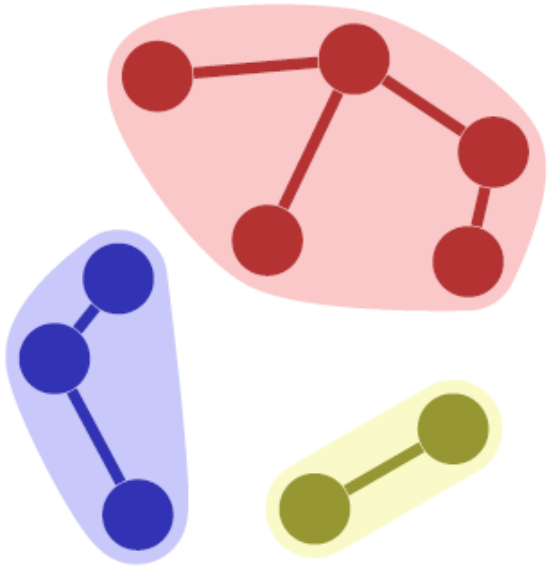
UnionFind(x,y) überprüft ob zwei Knoten der gleichen Gruppe geören, indem es die Verkettung traversiert

Index	1	2	3	4	5	6	7	8	9	10
Parent	1	1	1	6	5	6	6	5	3	10

Kruskal Algorithmus Code

Input: Gewichteter Graph $G = (V, E, c)$
Output: MST von G
Sortiere Kanten nach Gewicht $c(e_1) \leq \ldots \leq c(e_m)$
$M \leftarrow \emptyset$
for $i = 1$ **to** $|V|$ **do**
    `MakeSet`(i)
for $i = 1$ **to** m **do**
    $(u, v) \leftarrow e_i$
    if `Find`$(u) \neq$ `Find`(v) **then**
        `Union`(`Find`(u), `Find`(v))
        $M \leftarrow M \cup \{e_i\}$
    // konzeptuell: else $D \leftarrow D \cup \{e_i\}$
return (V, M, c)

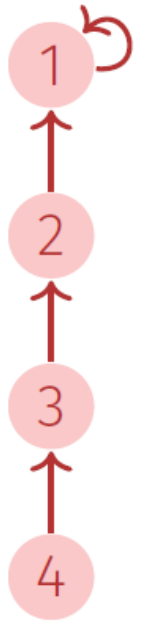




Kruskal Algorithmus Laufzeit



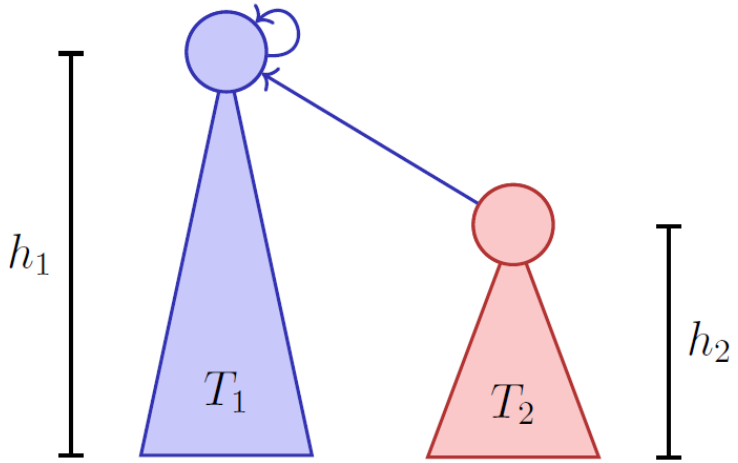
- Sortieren der Kanten: $\Theta(|E| \log |E|) = \Theta(|E| \log |V|)$
- Initialisieren der Union-Find Datenstruktur: $\Theta(|V|)$
- $|E| \times$ `Union`$($`Find`(x), `Find`$(y))$: ?



Kruskal Algorithmus Laufzeit

- Sortieren der Kanten: $\Theta(|E| \log |E|) = \Theta(|E| \log |V|)$
- Initialisieren der Union-Find Datenstruktur: $\Theta(|V|)$
- $|E| \times$ `Union`(`Find`(x), `Find`(y)): $\mathcal{O}(|E| \log |V|)$

Insgesamt $\Theta(|E| \log |V|)$



`Union`(i, j)

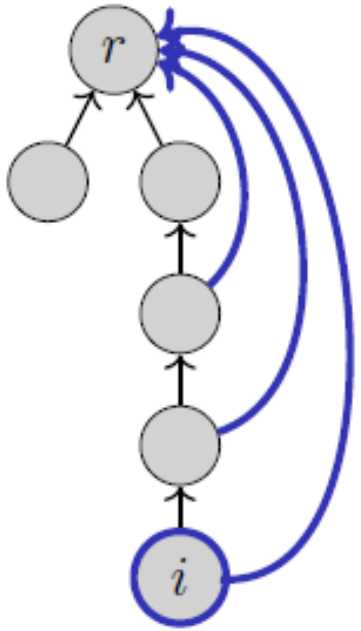
if $h[j] > h[i]$ **then** swap(i, j)
$p[j] \leftarrow i$
if $h[i] = h[j]$ **then** $h[i] \leftarrow h[i] + 1$

Kruskal Algorithmus Laufzeit

- Sortieren der Kanten: $\Theta(|E| \log |E|) = \Theta(|E| \log |V|)$
- Initialisieren der Union-Find Datenstruktur: $\Theta(|V|)$
- $|E| \times$ `Union`(`Find`(x), `Find`(y)): $\Theta(|E|)$

Insgesamt $\Theta(|E| \log |V|)$

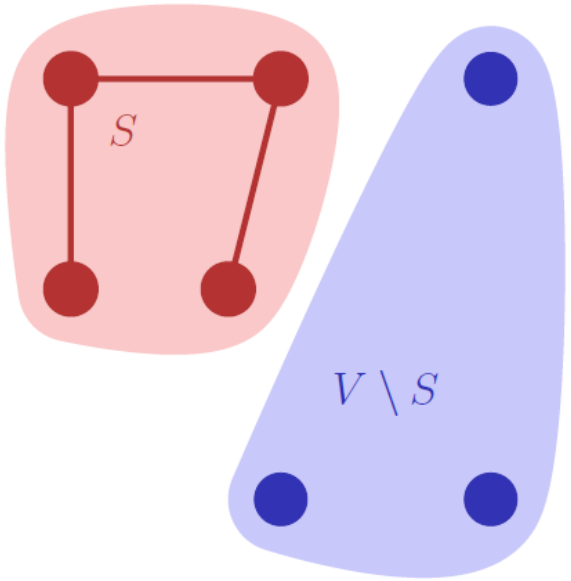
```
ImprovedFind(i):
r ← Find(i)
v ← i
while (v ≠ r) do
    p_v ← p[v]
    p[v] ← r
    v ← p_v
return r
```



01 – Theorie
01.1 A*
01.2 Kruskal
01.3 Jarnik (Prim)
01.4 Traveling Salesman

Jarnik (Prim) Algorithmus

Input: $G = (V, E, c)$
Output: MST von G
$M \leftarrow \emptyset$
$S \leftarrow \{v_0\}$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    Wähle billigste $e = \{u, v\}$ mit $u \in S$, $v \notin S$
    $M \leftarrow M \cup \{e\}$
    $S \leftarrow S \cup \{v\}$
return (V, M, c)



Jarnik (Prim) Algorithmus

Jarnik (Prim): Sortiert nach Kantengewichten
→ MST als Output

A*: Sortiert nach Pfadlängen + Heuristik
→ Kürzeste Pfadlänge as Output

Um A* zu erhalten, erweitere um diese Teile.
Um Prim zu erhalten, entferne diese Teile.

for $u \in V$ **do**
  $d_s[u] \leftarrow \infty$; $\pi[u] \leftarrow$null; $\hat{f}[u] \leftarrow \infty$

$U = \{s\}$; $d_s[s] \leftarrow 0$ /* Put only s in U and keep track */
while $U \neq \emptyset$ **do**
  $u \leftarrow$ extractMin(U) using $d_s[u]$ (use $\hat{f}[u]$ instead) /* Get nearest u */
  for $v \in N(u)$ with $d[u]+c(u,v) < d[v]$ **do**
    $d_s[v] \leftarrow d_s[u]+c(u,v)$ /* Update distance to v */
    $\pi[v] \leftarrow u$ /* Store predecessor */
    $\hat{f}[v] \leftarrow d_s[v] + \hat{h}[v]$ /* Update estimates */
    $U \leftarrow U \cup \{v\}$ /* Queue v */

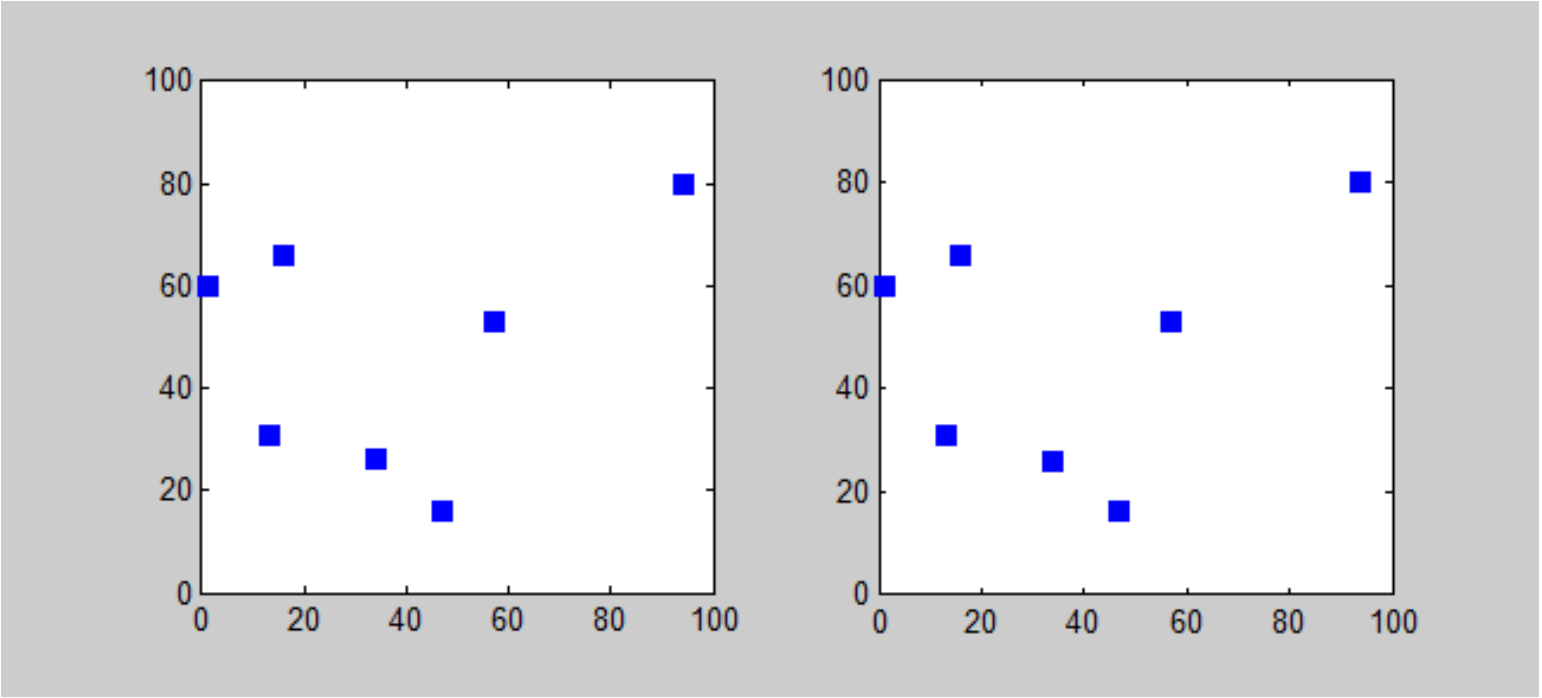
01 – Theorie
01.1 A*
01.2 Kruskal
01.3 Jarnik (Prim)
01.4 Traveling Salesman

Traveling Salesman Problen



Frage: Was ist das kürzeste Weg alle Städte zu besuchen und zum Startpunkt zurückzukommen?

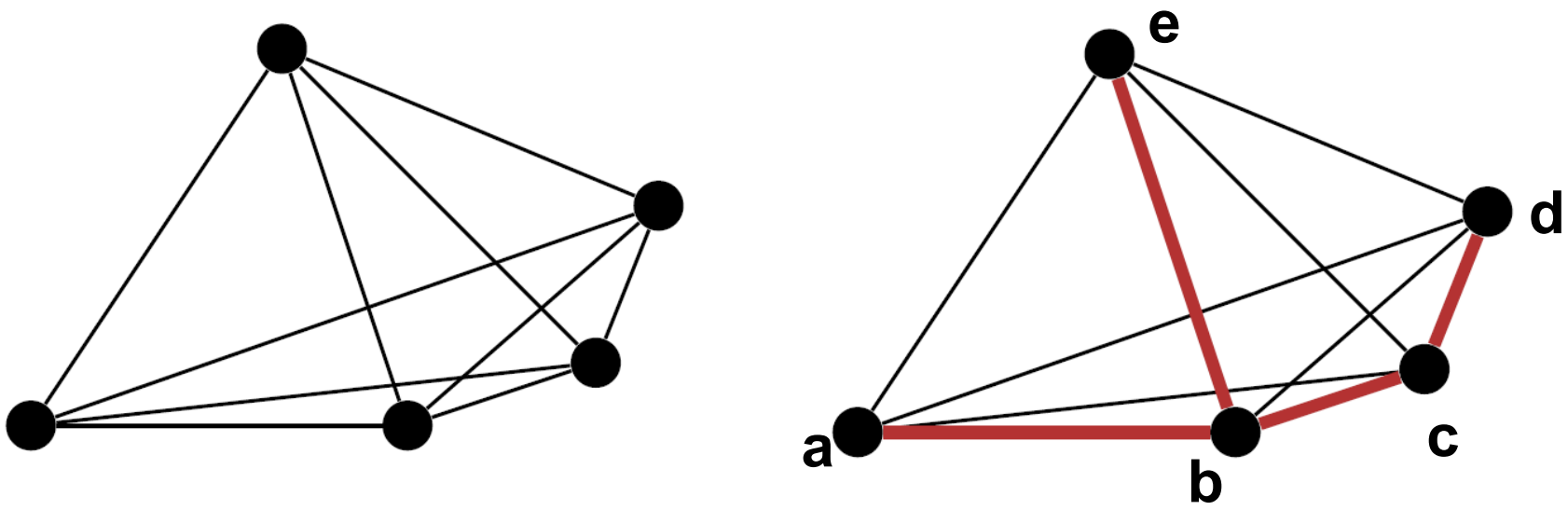
Traveling Salesman Problen



https://upload.wikimedia.org/wikipedia/commons/2/2b/Bruteforce.gif

Antwort: Keine effiziente Lösung bekannt. Naiv alle Wege ausprobieren liefert $\Theta(n!)$ → sehr ineffizient.
Doch verschiedene Näherungen existieren.

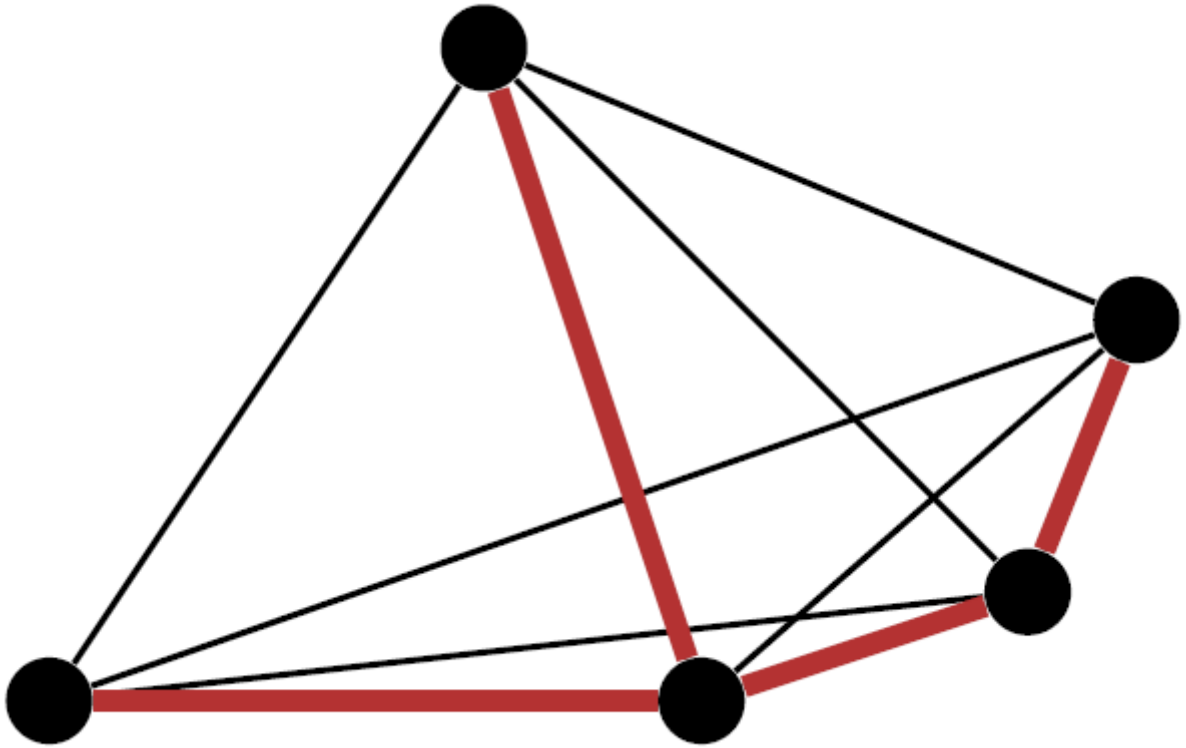
Traveling Salesman Problem Näherung



Schritt 1: Finde MST, markiere die Knoten

Traveling Salesman Problem Näherung

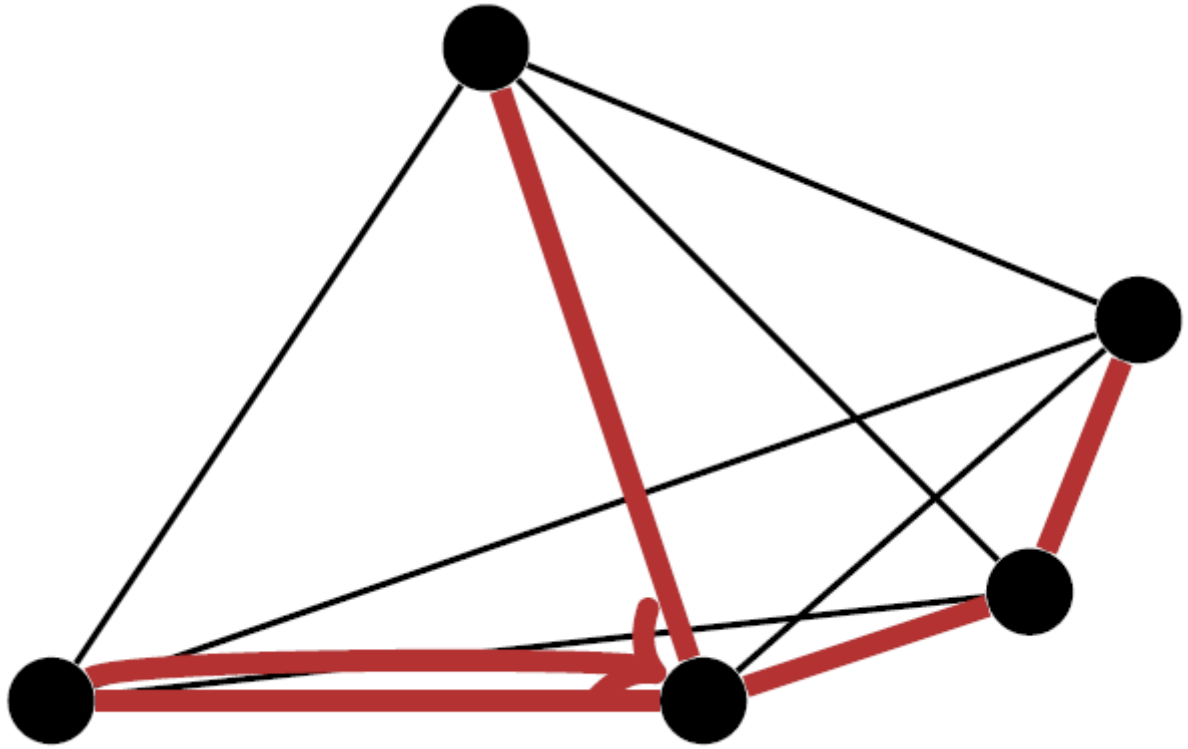
Besuchsliste:
a



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

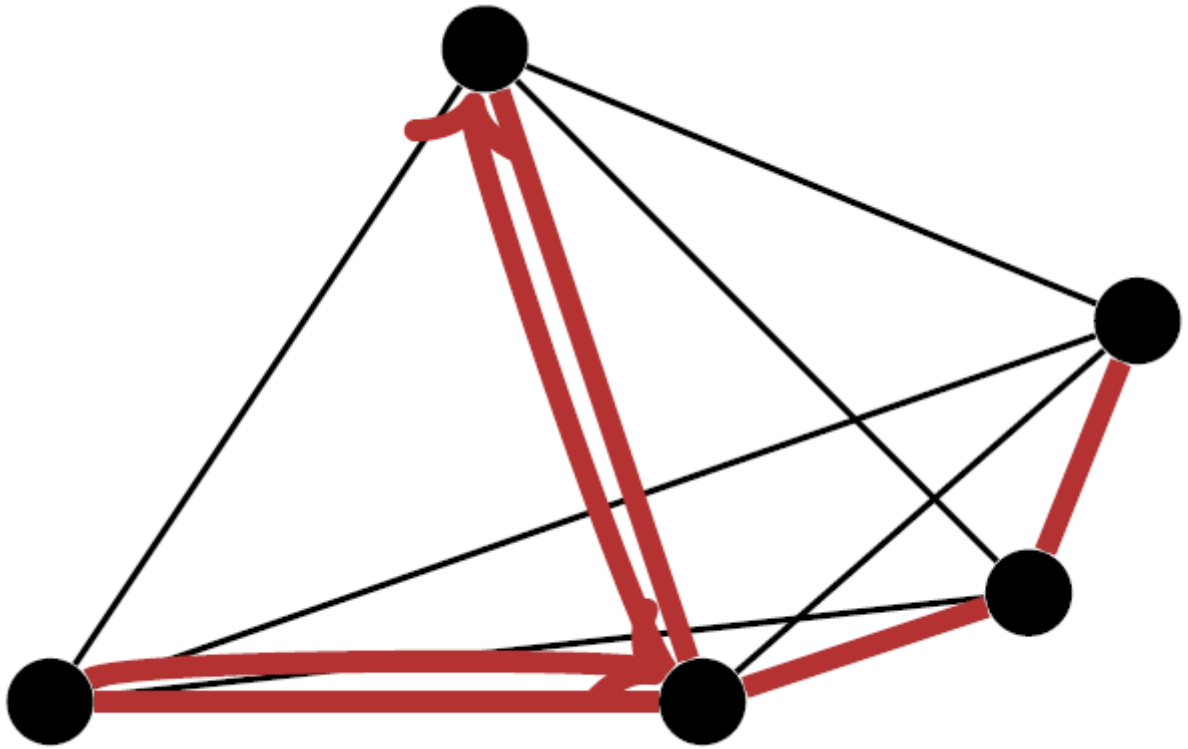
Besuchsliste:
a,b



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

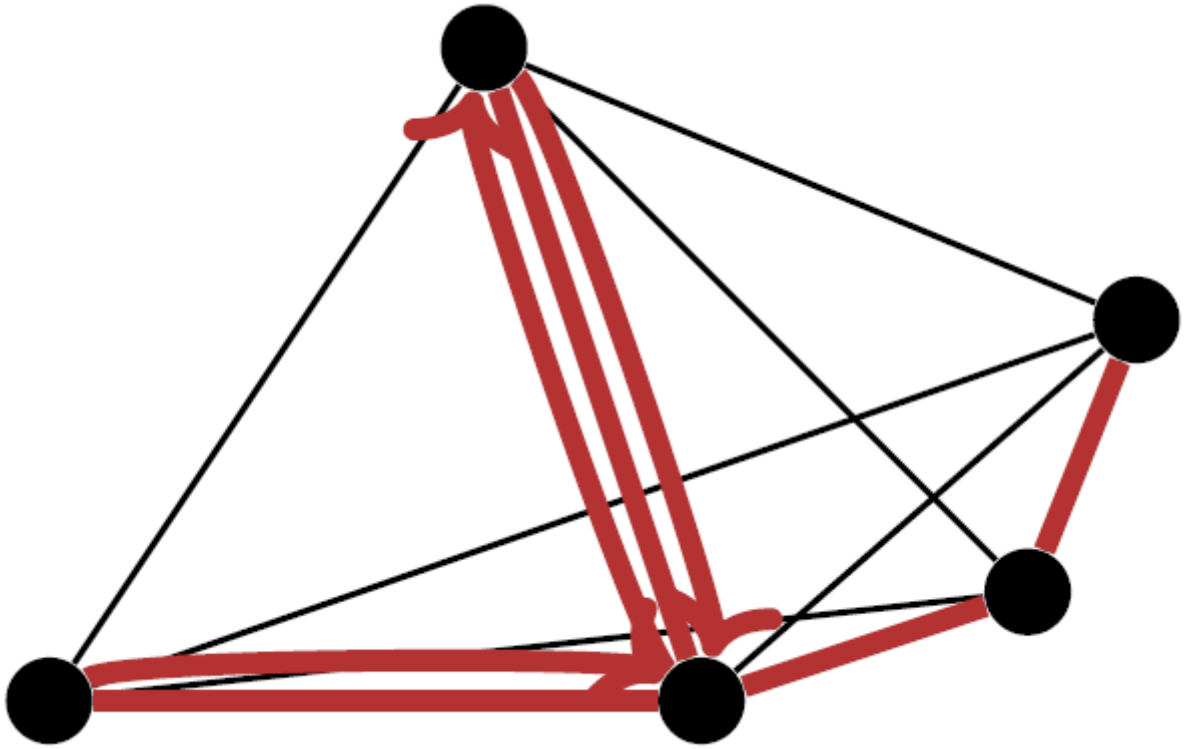
Besuchsliste:
a,b,e



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

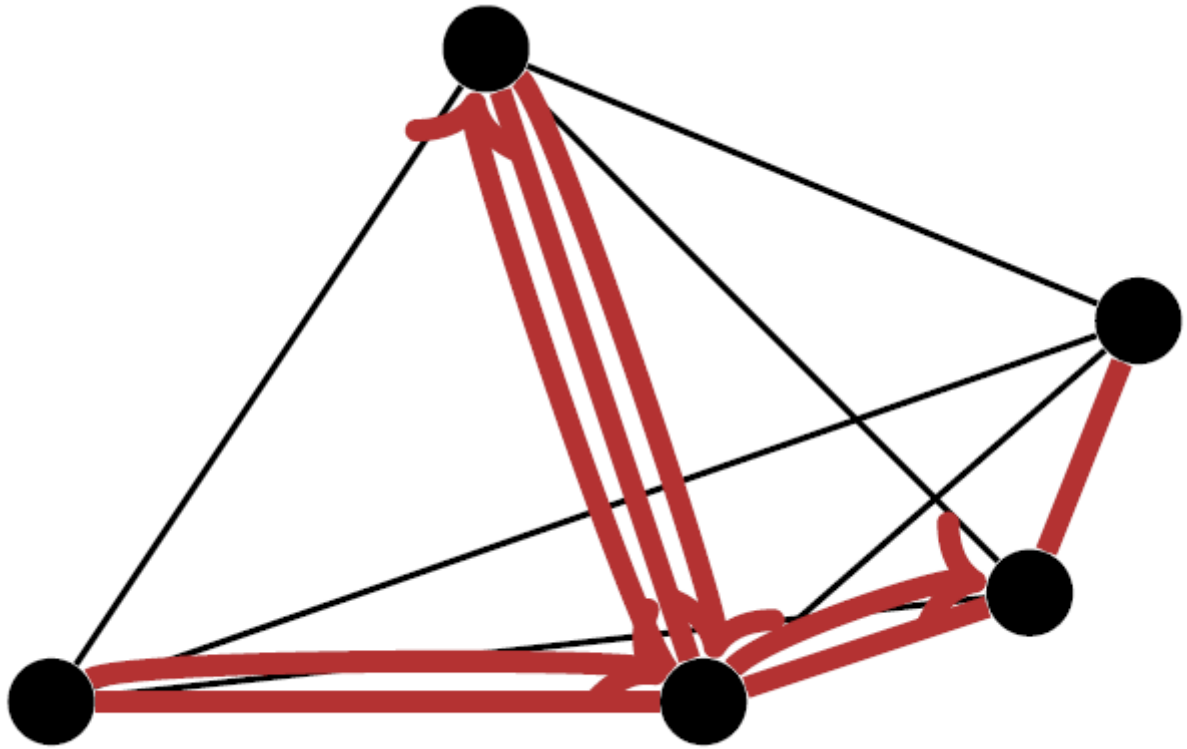
Besuchsliste:
a,b,e



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

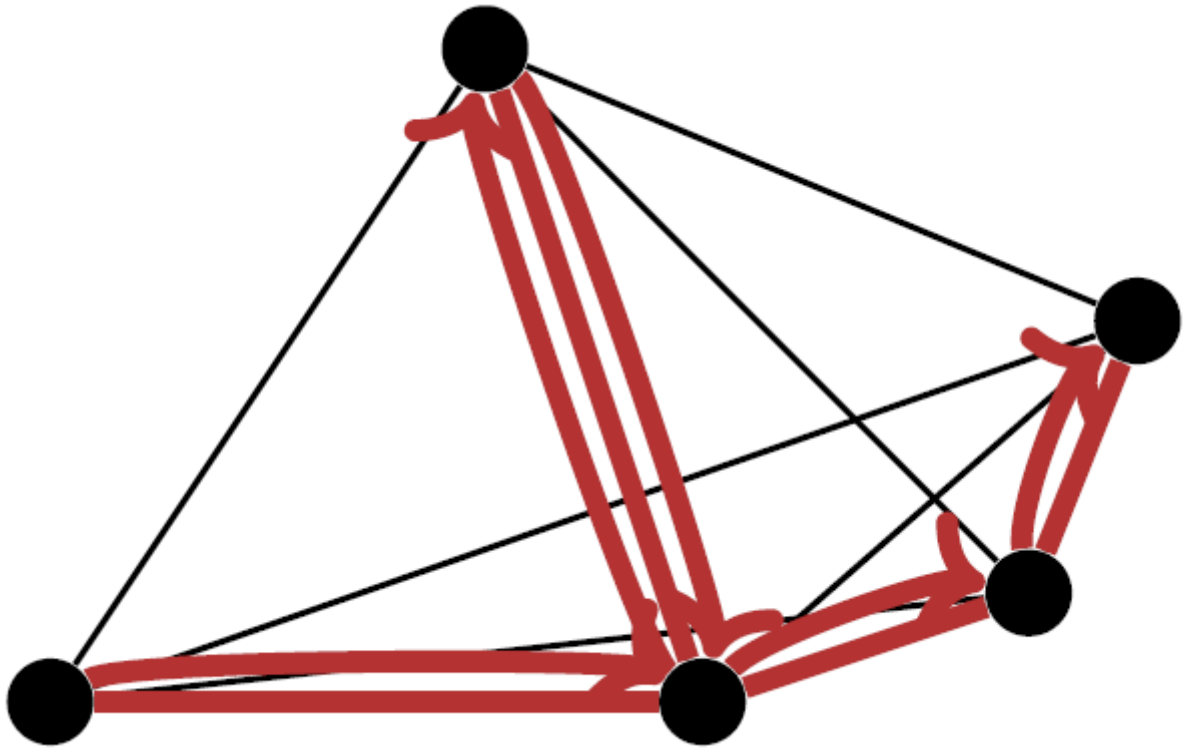
Besuchsliste:
a,b,e,c



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

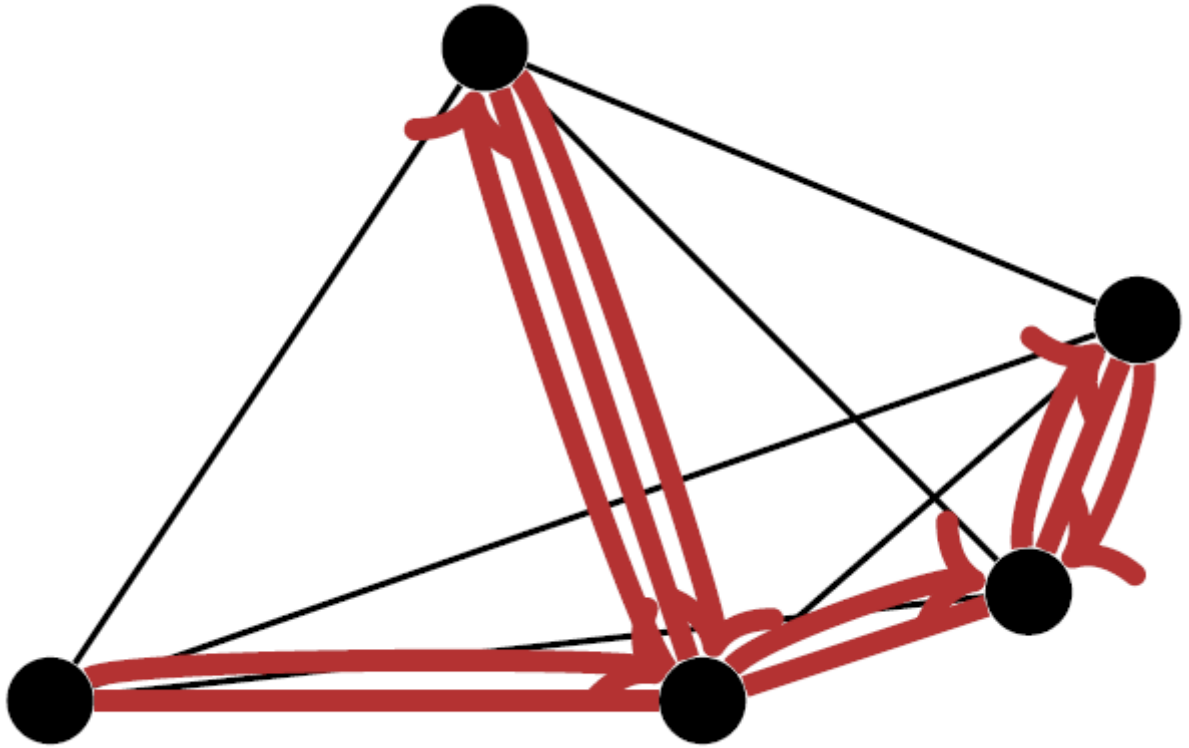
Besuchsliste:
a,b,e,c



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

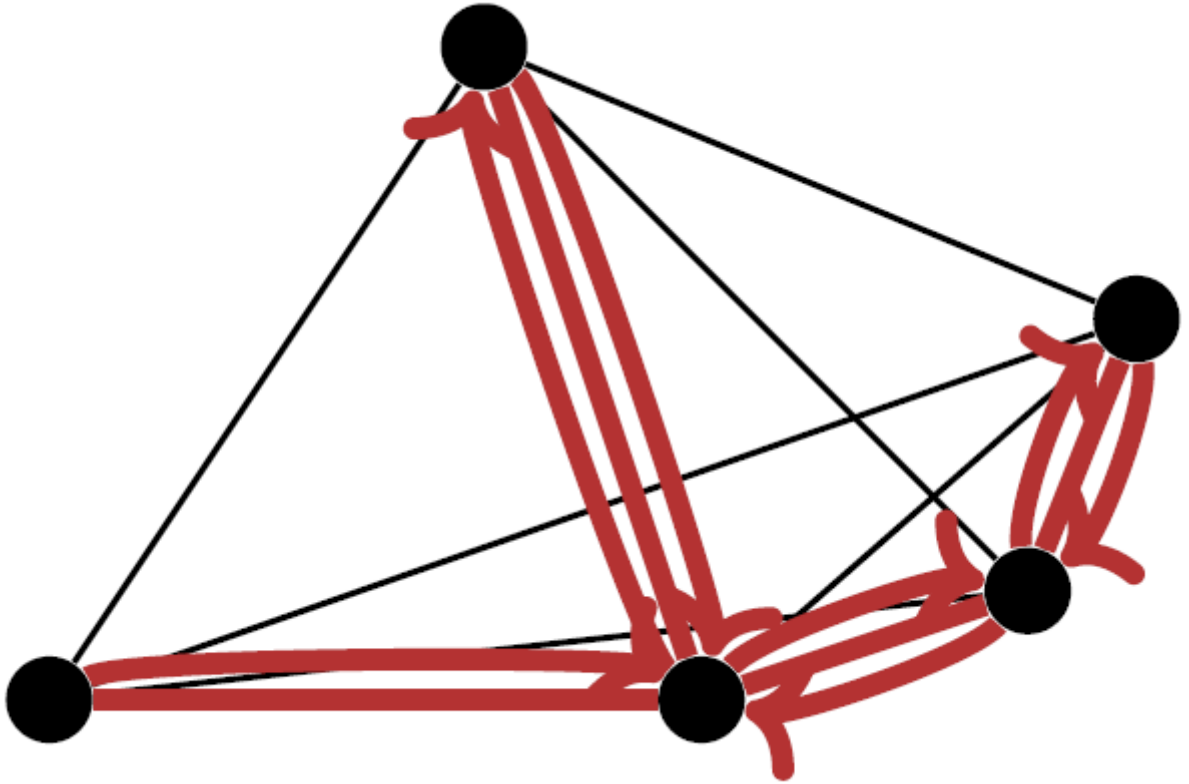
Besuchsliste:
a,b,e,c



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

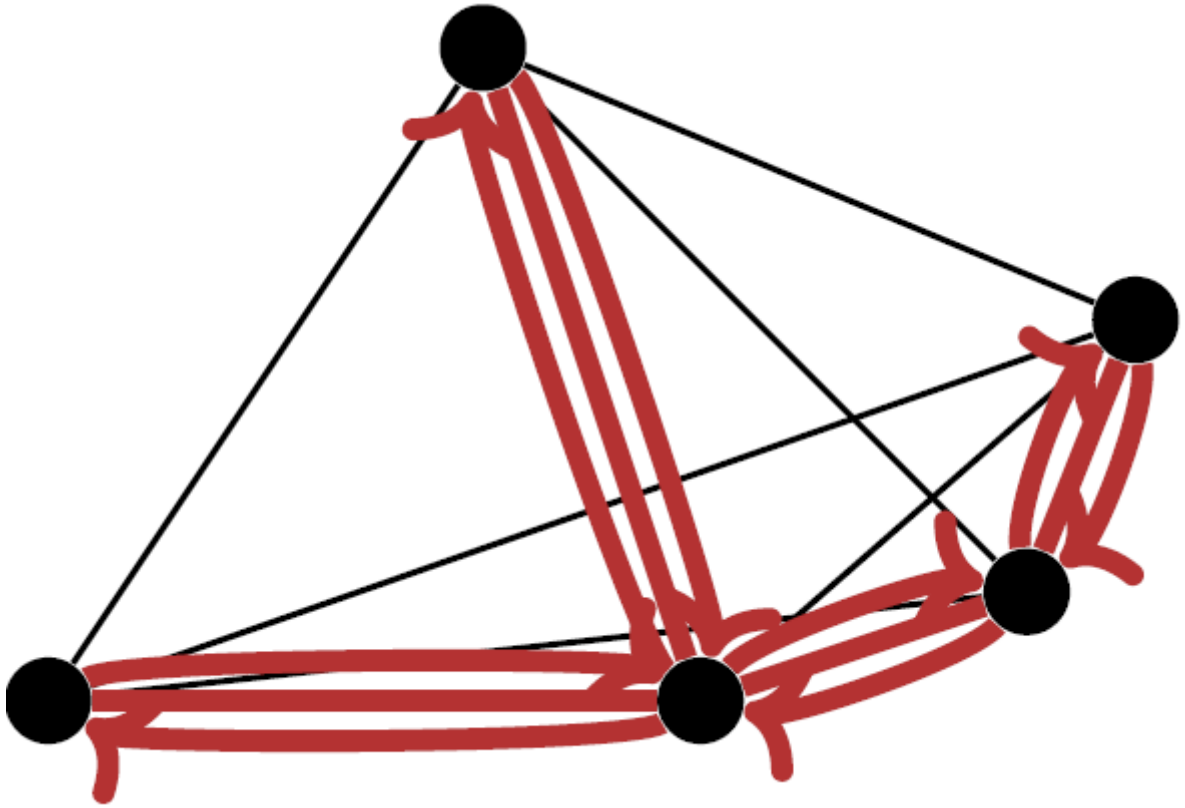
Besuchsliste:
a,b,e,c



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

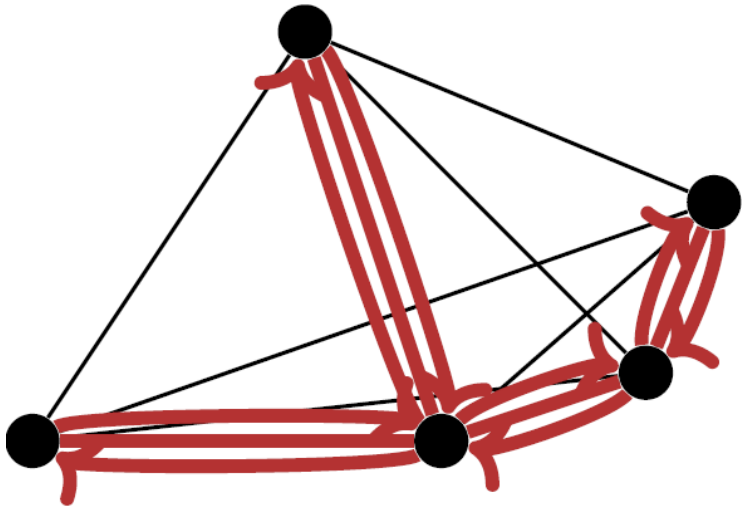
Besuchsliste:
a,b,e,c



Schritt 2: Laufe Depth First Search. Wenn du eine neue Kante begegnest, füge die in die Besuchsliste hinzu

Traveling Salesman Problem Näherung

Optimale Länge: $c(S^*)$
Länge von diesem Kreis $c(S) \leq 2c(S^*)$ →
Serie - Aufgabe
Beste bekannte Algorithmen $c(S) \leq 1.01c(S^*)$



02 – Quiz

Quiz Frage 1

Was beschreibt am ehesten die Heuristik $h(v)$ im A*-Algorithmus?

1. Eine Funktion, die den Kanten des Graphen Gewichte zuweist.
2. Eine Funktion, die die Entfernung zwischen dem aktuellen Knoten und dem Zielknoten schätzt.
3. Eine Funktion, die den nächsten zu erweiternden Knoten auf der Grundlage der niedrigsten Pfadkosten auswählt.
4. Eine Funktion, die überprüft, ob der Zielknoten erreicht wurde.

Quiz Frage 2

Welche Voraussetzung muss eine Heuristik im A*-Algorithmus erfüllen, damit A* korrekt funktioniert?

A. Knotengewicht $\in \mathbb{R}$
B. $h(u) < \delta(u,t)$
C. $h(u) > \delta(u,t)$
D. $h(u) > c((u,v)) + h(v)$

Quiz Frage 3

Welche Heuristik tut die Distanz auf dem zweidimensionalen Grid überschätzen?

A. $h = |\delta x| + |\delta y|$
B. $h = |\delta x^2| + |\delta y^2|$
C. $h = \sqrt{|\delta x^2| + |\delta y^2|}$
D. $h = max(\delta x, \delta y)$

Quiz Frage 4

Wie viele Kanten besitzt ein Spannbaum in einem Graphen mit |V| Knoten?

A. |V|
B. |E| − 1
C. |V| − 1
D. |V| + |E|

Quiz Frage 5

Was ist die beste asymptotische Laufzeit erreicht der Union-Find-Datenstruktur ?

A. O(n^2)
B. O(n)
C. O($log\, n$)
D. O(1)

Quiz Frage 6

In einem gewichteten Graph mit n Knoten und m Kanten, die alle dasselbe Gewicht haben, kann der minimale Spannbaum mit $\Theta(n + m)$ Schritten gefunden werden.

A. Wahr
B. Falsch

Quiz Frage 7

Worin besteht der Unterschied zwischen Jarník/Prim und Kruskal?

A. Prim sortiert Kanten, Kruskal nicht
B. Prim wächst einen Baum von einem Startknoten aus, Kruskal fügt global die kleinsten Kanten hinzu
C. Kruskal benötigt keine Datenstrukturen
D. Prim vermeidet Zyklen nicht

Quiz Frage 8

Welche Operation wird in Dijkstra durchgeführt, aber in Prim nicht?

A. Extract-Min auf einer Priority Queue
B. Berechnung von d[u] + c(u,v)
C. Initialisierung aller Distanzen mit ∞
D. Auswahl der minimalen Kante

Quiz Frage 9

Wie viele mögliche Rundreisen (Touren) müssen beim Brute-Force-TSP für einen vollständigen Graphen mit 7 Knoten untersucht werden, wenn man den Startknoten fixiert?

A. 120
B. 360
C. 720
D. 1024

Quiz Frage 10

Welche asymptotische Laufzeit besitzt der in der Vorlesung vorgestellten TSP Algorithmus auf einem vollständigen Graphen mit n Knoten?

A. $\Theta(n \log n)$
B. $\Theta(n^2)$
C. $\Theta(n^2 \log n)$
D. $\Theta(n!)$

Quiz Frage 1

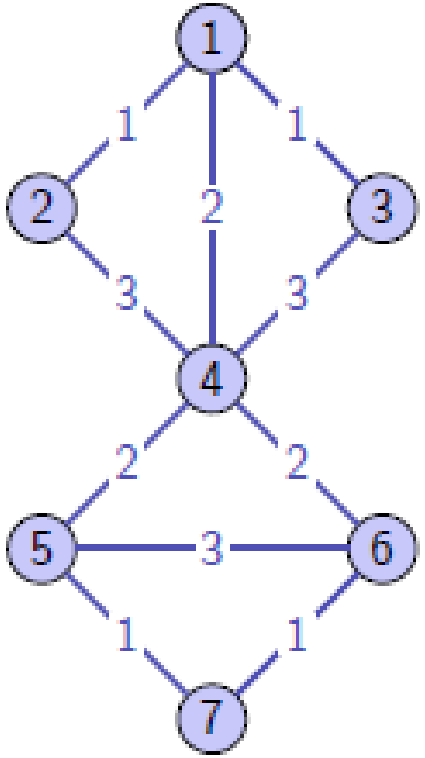
Welche Eigenschaften hat die Lösung von TSP aus der Vorlesung im vergleich zum optimalen Lösung?

A. Die Weglänge ist höchstenst doppelt so lang
B. Die Weglänge ist mindestens doppel so lang
C. Die Anzahl von Operationen ist zumindest zwei mal kleiner
D. Die Grösse von benötigten Speicherplatz ist höchstens doppel so gross

Quiz Frage 1

Was ist das Gewicht von minimalen Spannbaum?

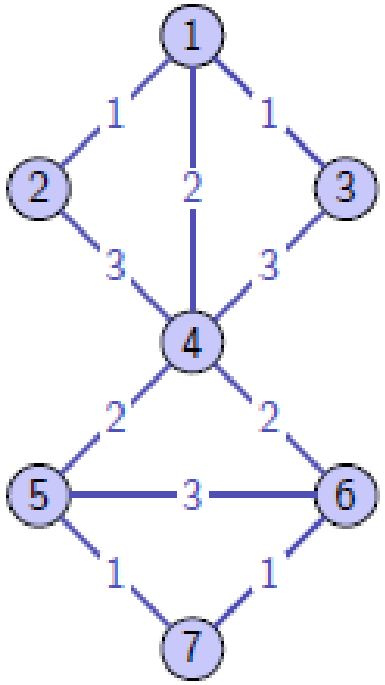
A.4
B.6
C.8
D.15



Quiz Frage 1

Wie viele minimale Spannbäume existieren?

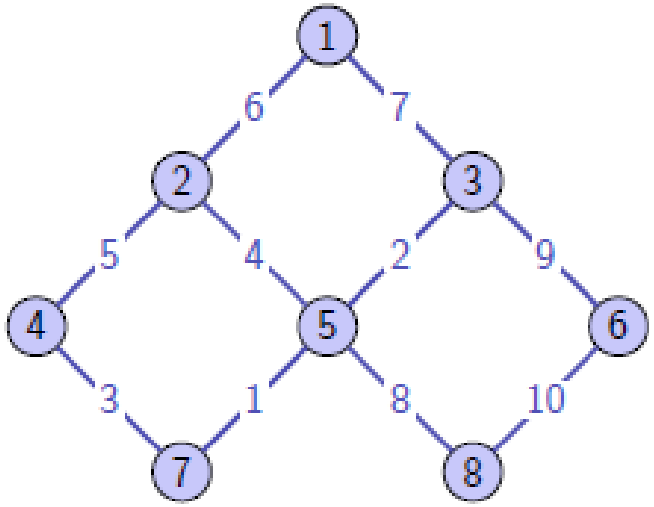
A.1
B.2
C.3
D.4



Quiz Frage 1

Was ist das Gewicht von minimalen Spannbaum?

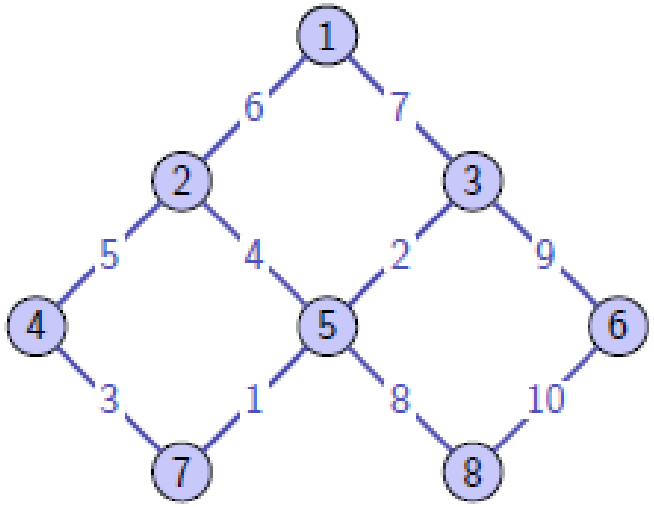
A.10
B.15
C.26
D.33



Quiz Frage 1

Wie viele minimale Spannbäume existieren?

A.1
B.2
C.3
D.4



03 – Programmierübung

Programmierübung: Union Find Experiments

Union Find Experiments - Student attempt

```cpp
#include "union_find.hpp"

//pre: Elements i and j are not required to be the root of a set.
//post: Merge the sets associated with i and j.
bool Union_find::unify(unsigned i, unsigned j){
  i = find(i);
  j = find(j);
  if (i == j){
    // TODO
  }
  // TODO
  return true;
}

// Return the representative of the set containing i.
unsigned Union_find::find(unsigned i){
  while (parents[i] != i){
    // TODO
  }
  // TODO
  return i;
}

```

1. Geh auf CodeExpert → Code Examples → Lazy Deletion
2. Implementiere:
   1. Einfachste mögliche Union-Find Struktur mit `unify` und `find`
   2. «Small below large» Optimalisation
   3. «Consolidate Find» Optimalisation