Universität Trier

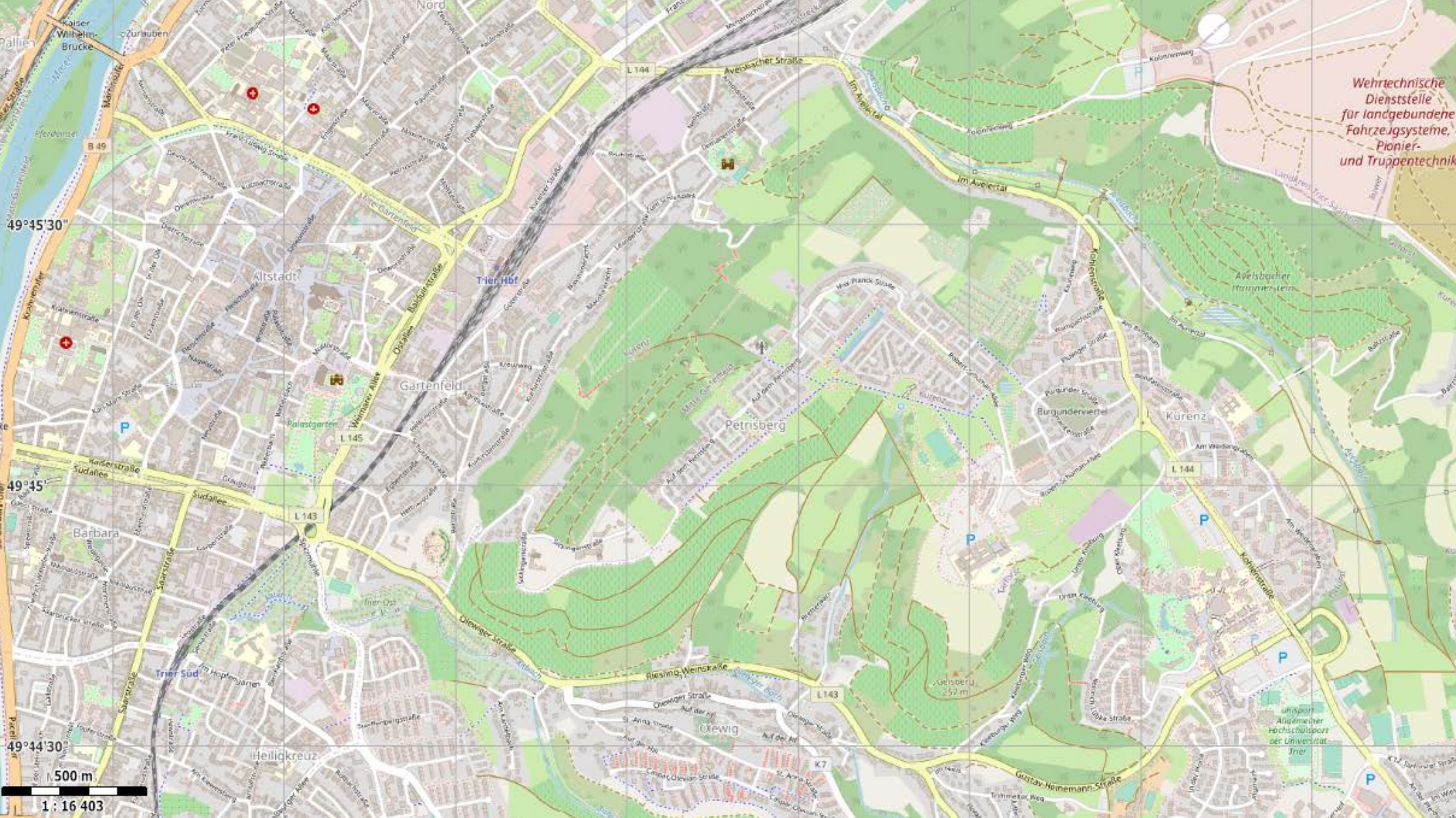
Algorithmen für geographische Informationssysteme

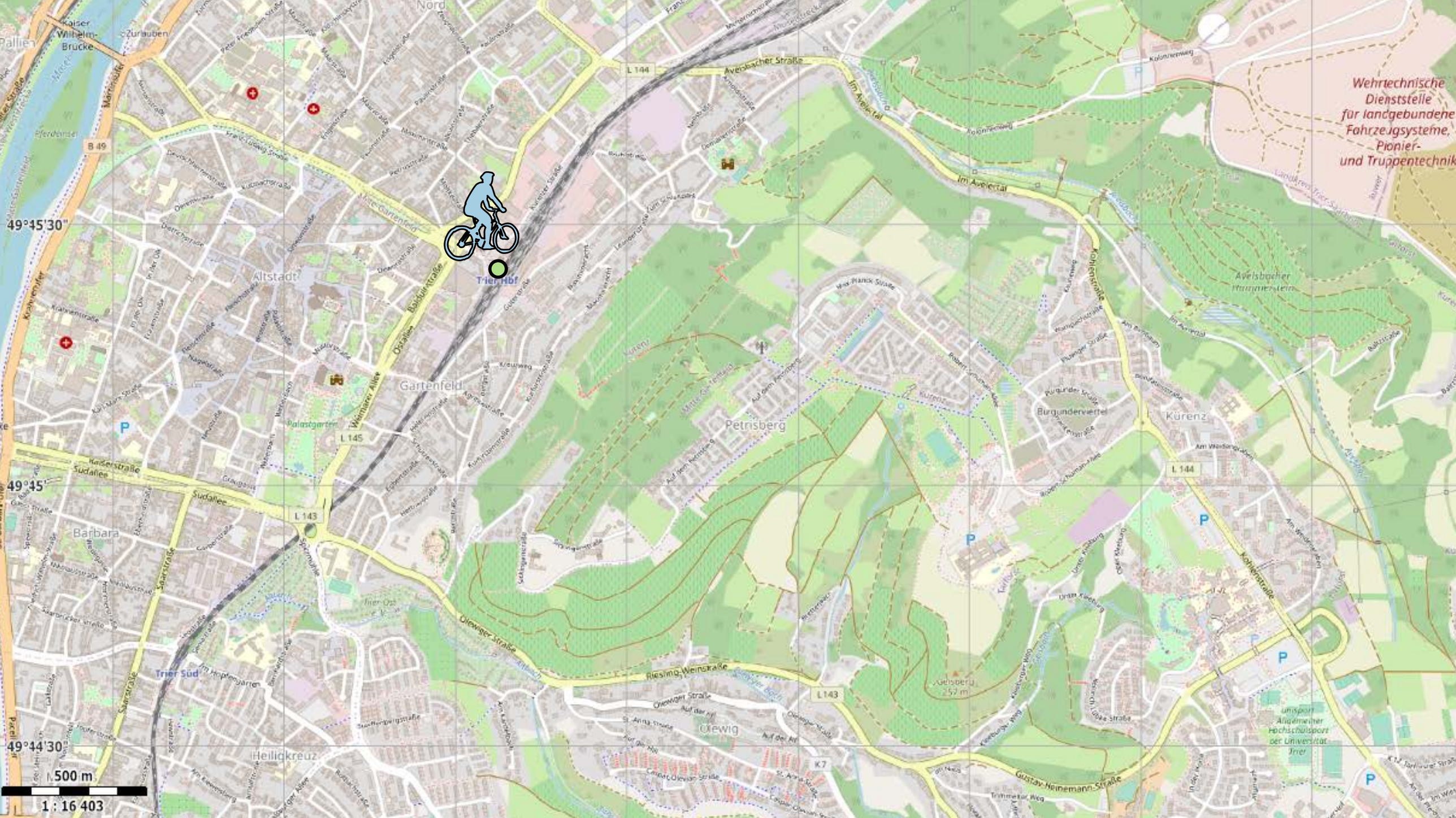
2. Vorlesung
Routenplanung

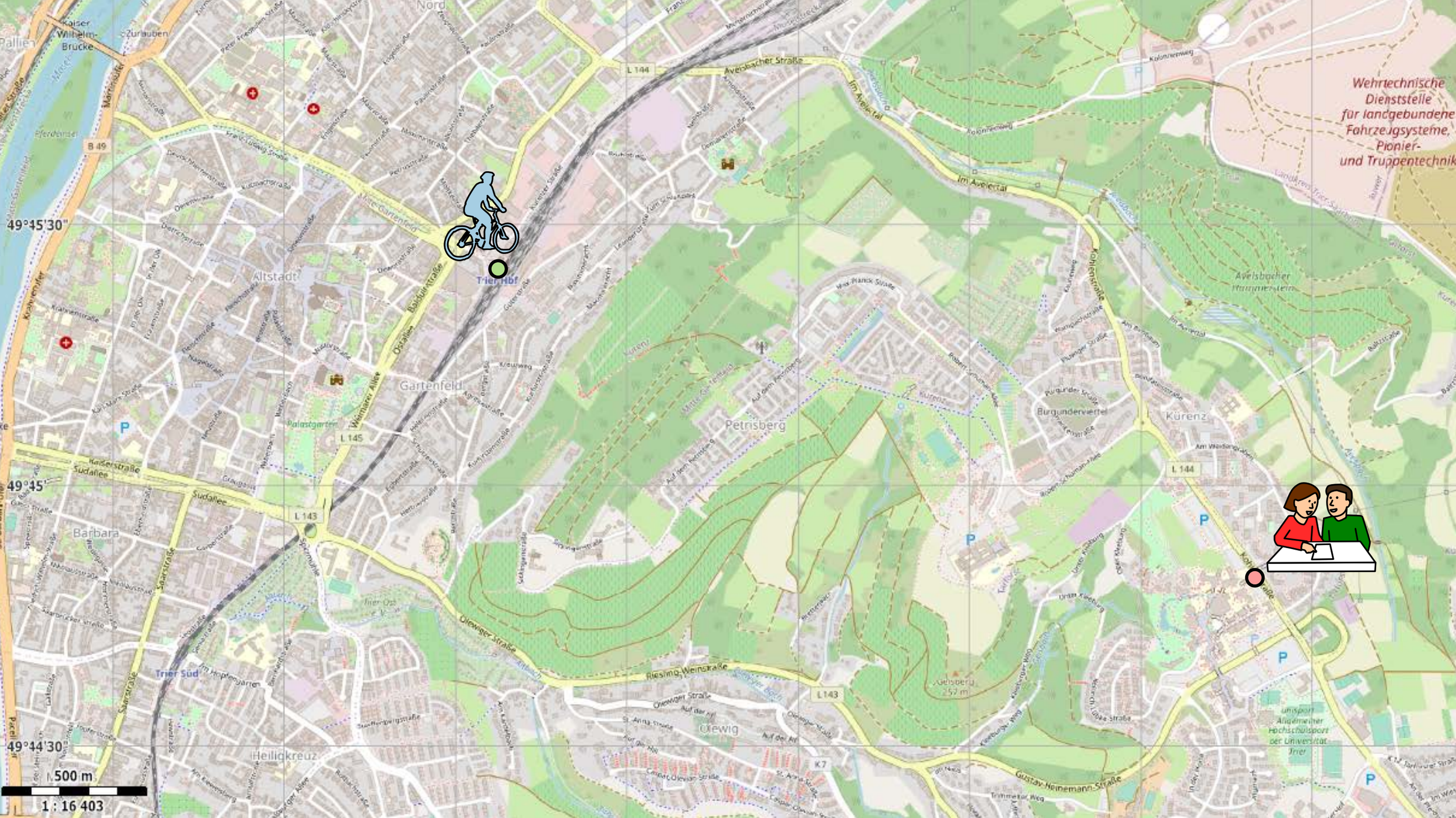
Teil I:
Problemstellung

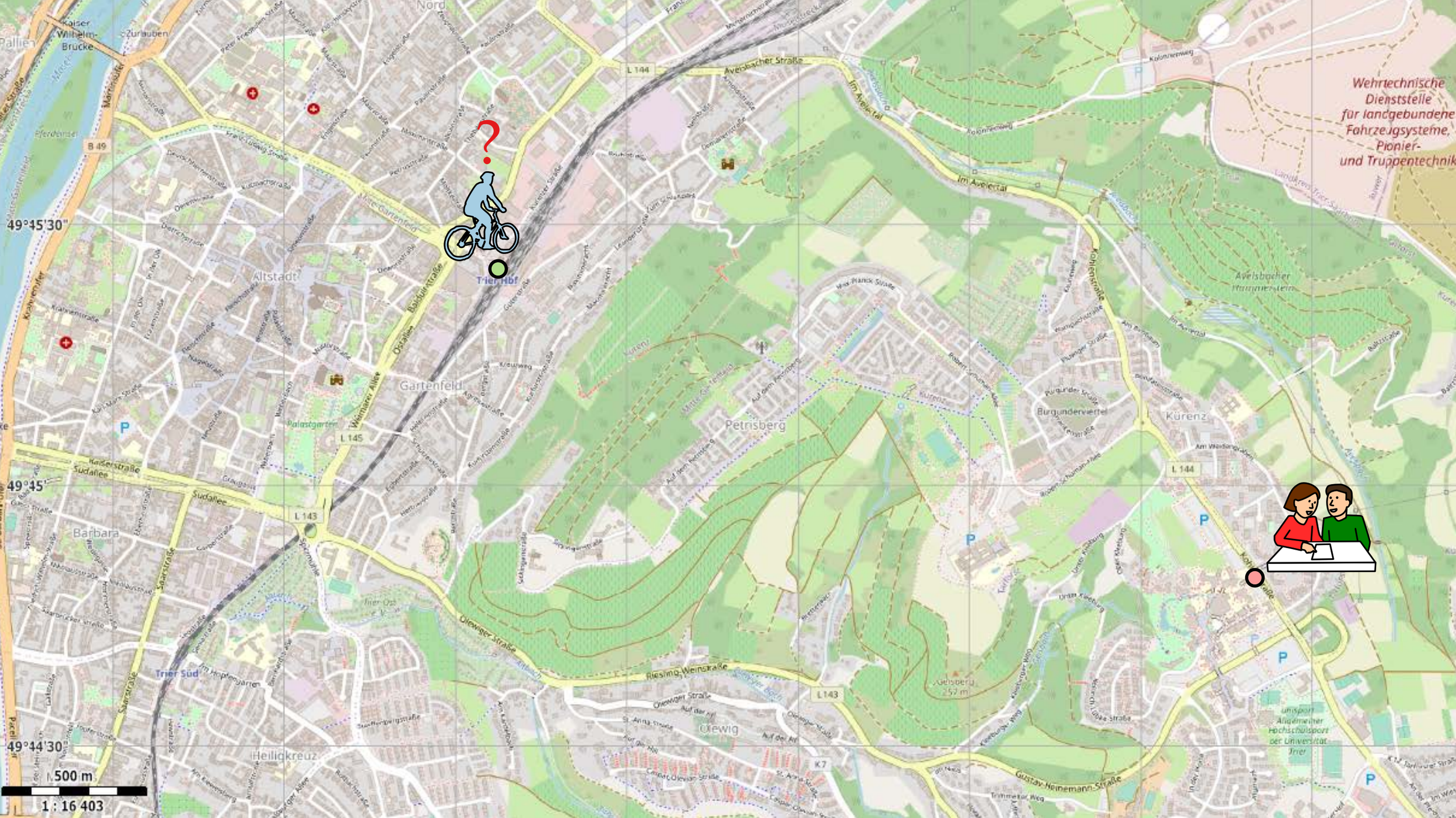
Philipp Kindermann

Wintersemester 2024/25


Wehrtechnische Dienststelle für landgebundene Fahrzeugsysteme, Pionier- und Truppentechnik
Kaiser-Wilhelm-Brücke
Zurlauben
Nord
Pallien
Mosel
B 49
L 144
Aveldacher Straße
Im Aveltal
Kolonnenweg
49°45'30"
Altstadt
Trier Hbf
Gartenfeld
Palastgarten
Weimarer Allee
Ostallee
Baldgirnstraße
Krahnenufer
Kaiserstraße
Südallee
L 145
L 143
49°45'
Barbara
Petrisberg
Kürenz
Burgunderviertel
Kohlenstraße
Am Weidengraben
Trier Süd
Im Hopfengarten
Saarstraße
Olewiger Straße
Riesling-Weinstraße
Olewig
Heiligkreuz
Gelsberg
257 m
Gustav-Heinemann-Straße
49°44'30"
500 m
1 : 16 403

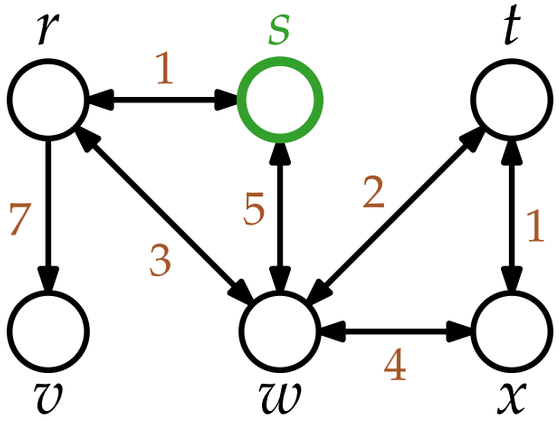

Kaiser-Wilhelm-Brücke
Zurlauben
Nord
Pallien
B 49
L 144
Aveler Straße
Im Aviertal
Wehrtechnische Dienststelle für landgebundene Fahrzeugsysteme, Pionier- und Truppentechnik
49°45'30"
Altstadt
Trier Hbf
Gartenfeld
Palastgarten
L 145
Petrisberg
Burgunderviertel
Kürenz
Avelsbacher Hauptmeilenstein
Kaiserstraße
Südallee
L 143
49°45'
Barbara
Trier Süd
Im Hopfengarten
Olewiger Straße
Riesling-Weinstraße
Olewig
Gelsberg
257 m
Heiligkreuz
49°44'30"
Gustav-Heinemann-Straße
unisport Allgemeiner Hochschulsport der Universität Trier
500 m
1 : 16 403


Kaiser-Wilhelm-Brücke
Zurlauben
Nord
L 144
Avelsbacher Straße
Im Avelertal
Wehrtechnische Dienststelle für landgebundene Fahrzeugsysteme, Pionier- und Truppentechnik
B 49
49°45'30"
Trier Hbf
Altstadt
Gartenfeld
Palastgarten
Weimarer Allee
Ostallee
Balduinstraße
Kohlenstraße
Petrisberg
Kürenz
Burgunderviertel
L 145
L 143
L 144
Kaiserstraße
Südallee
49°45'
Barbara
Saarstraße
Trier Süd
Im Hopfengarten
Olewiger Straße
Riesling-Weinstraße
Olewig
Olewiger Berg
Heiligkreuz
49°44'30"
500 m
1 : 16 403

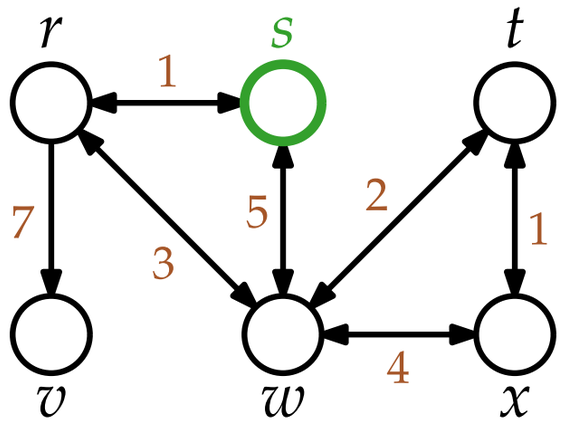

?
Trier Hbf
Altstadt
Gartenfeld
Petrisberg
Kürenz
Barbara
Heiligkreuz
Olewig
Trier Süd
Wehrtechnische Dienststelle für landgebundene Fahrzeugsysteme, Pionier- und Truppentechnik
49°45'30"
49°45'
49°44'30
500 m
1 : 16 403


?
49°45'30"
49°45
49°44'30"
500 m
1 : 16 403

Wiederholung – Dijkstra



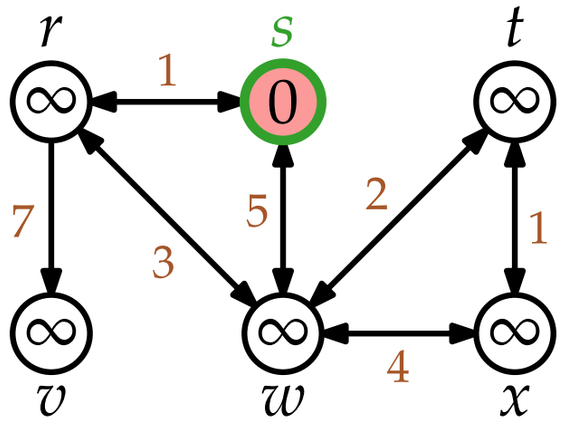
Wiederholung – Dijkstra




```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

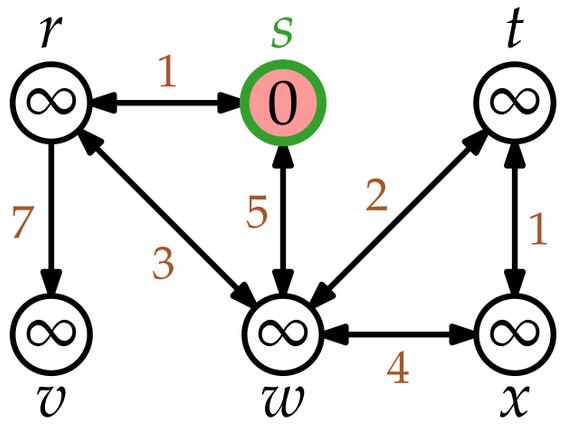
Wiederholung – Dijkstra




```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

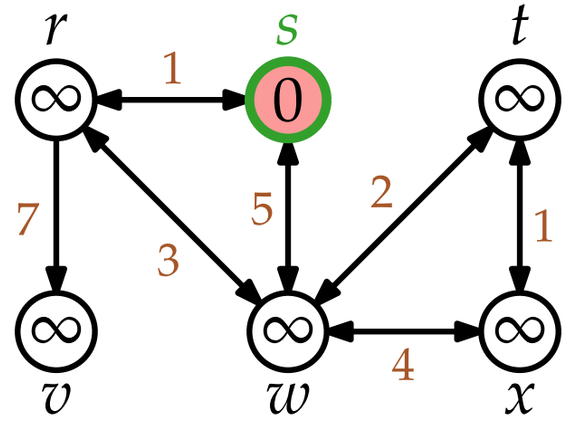
Wiederholung – Dijkstra

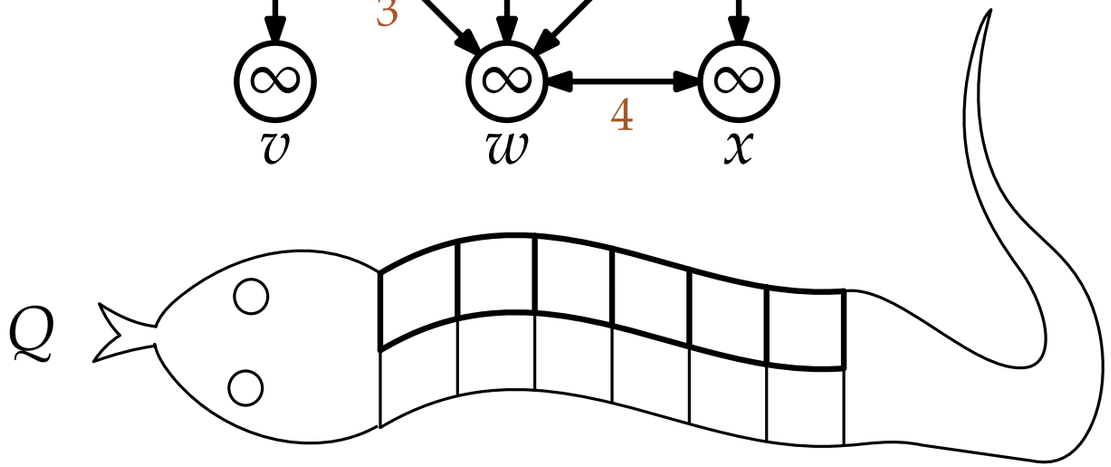



```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra



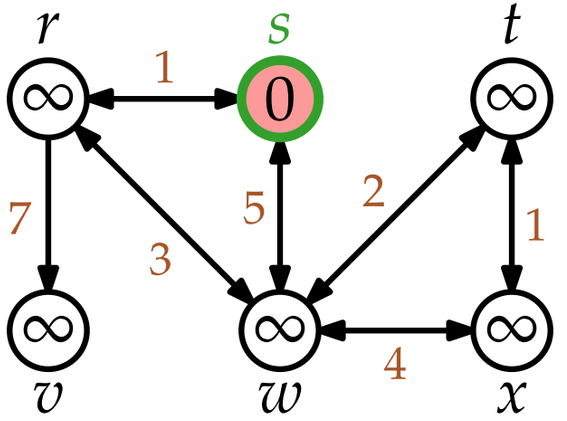


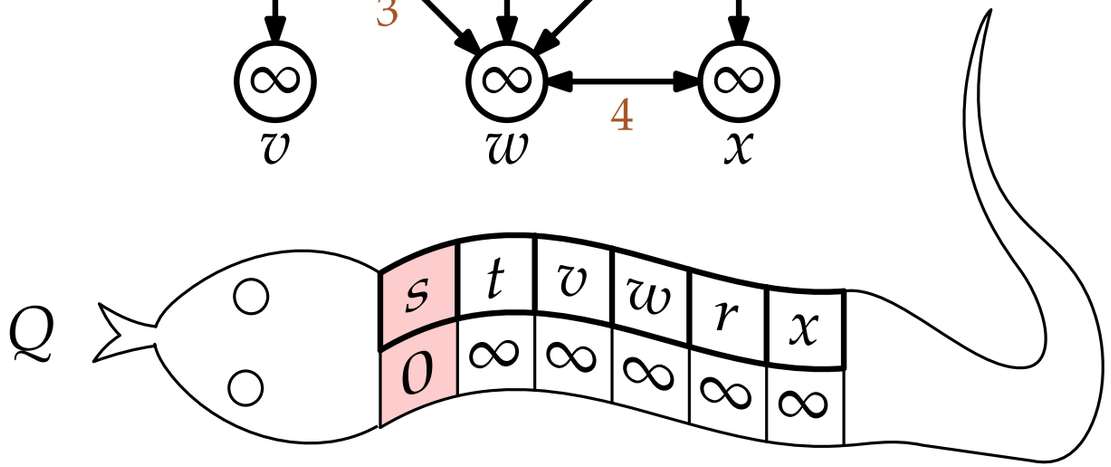

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

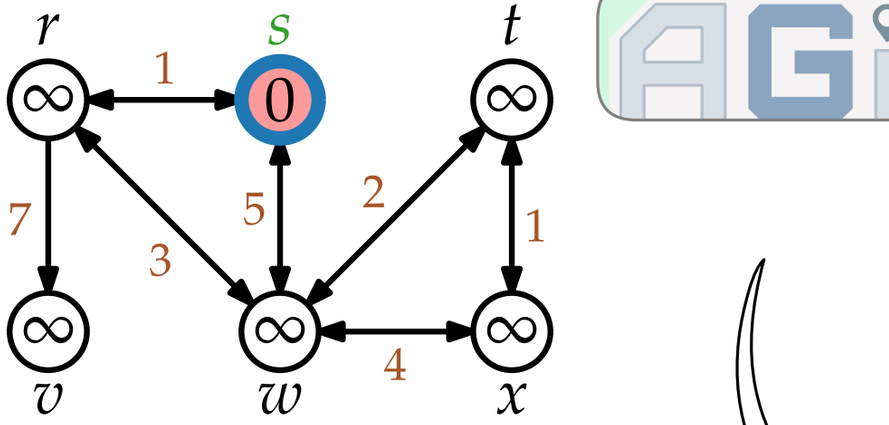


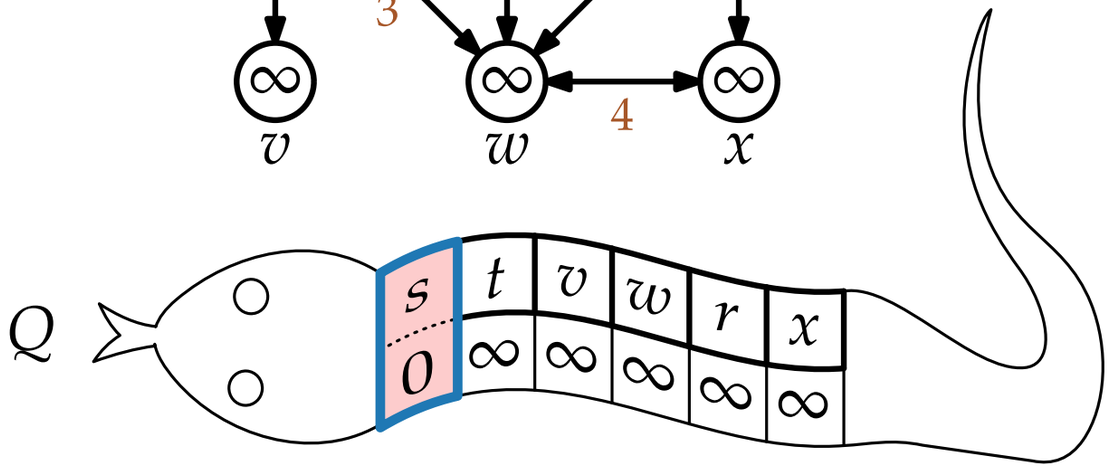



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

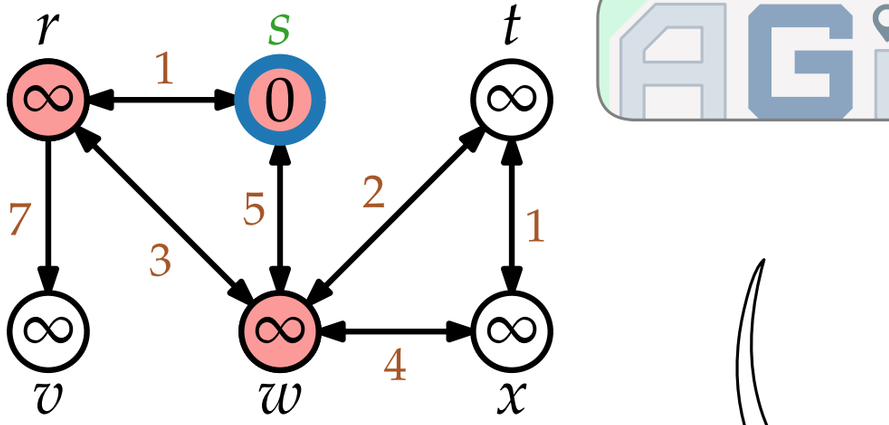


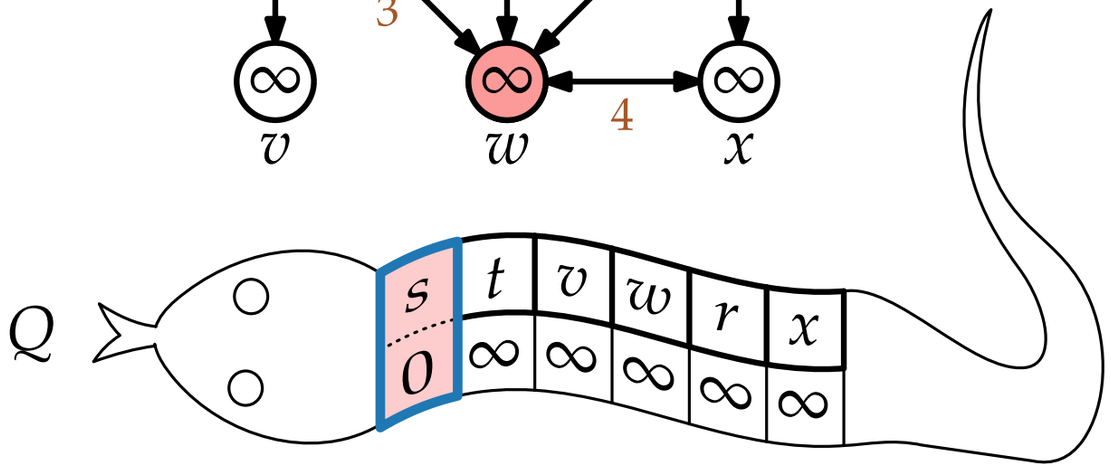



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

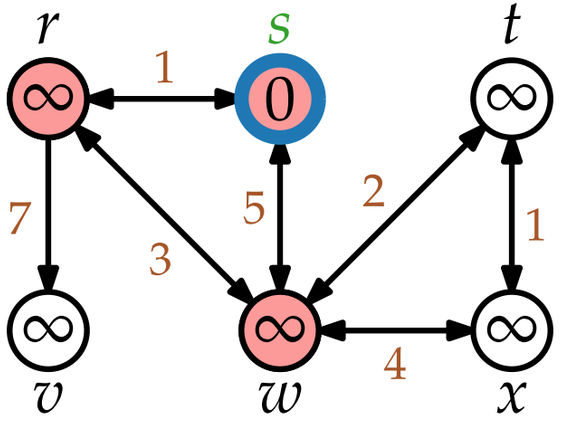


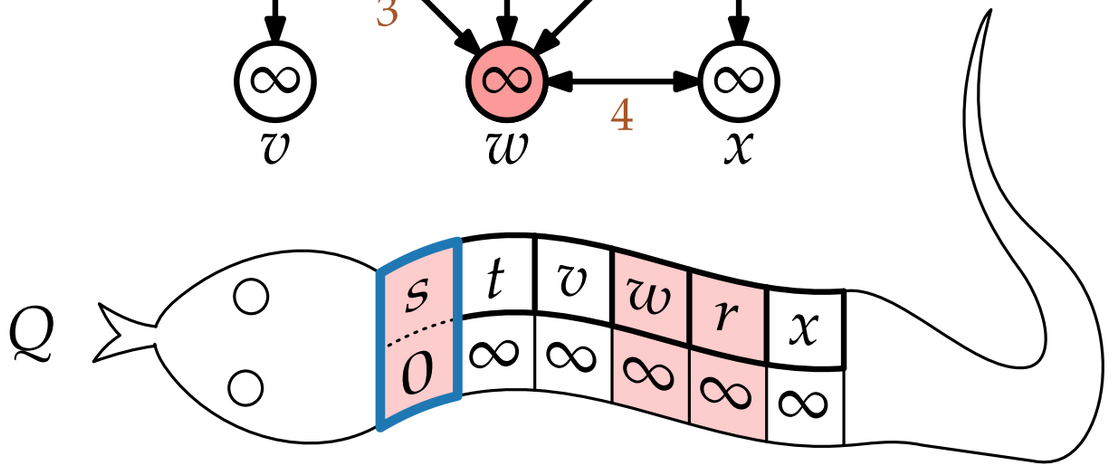



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

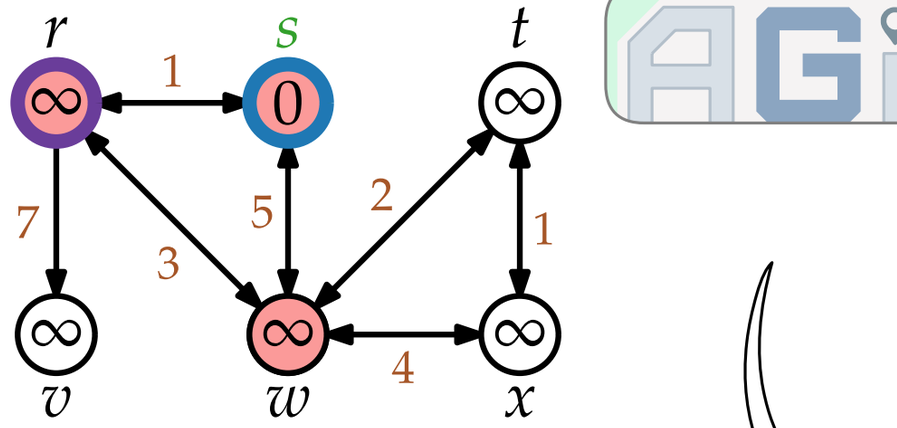


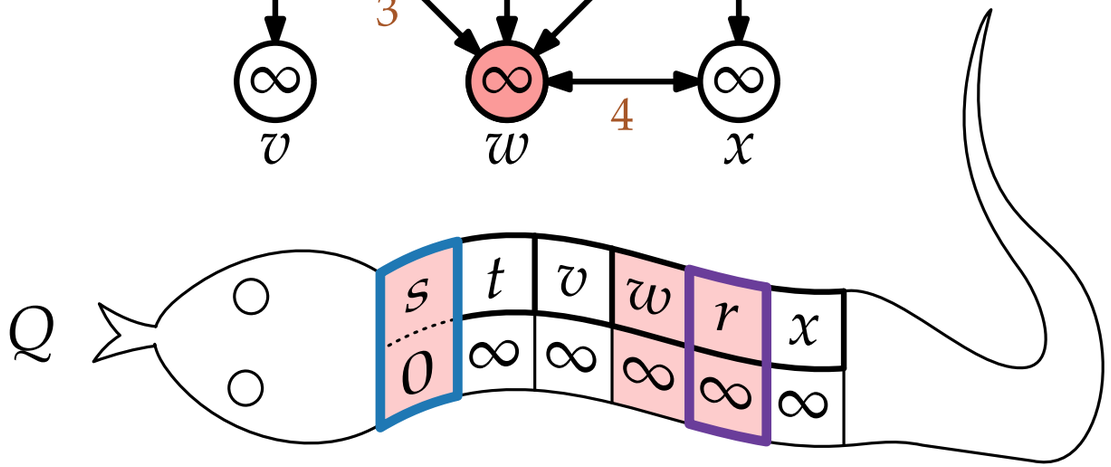



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

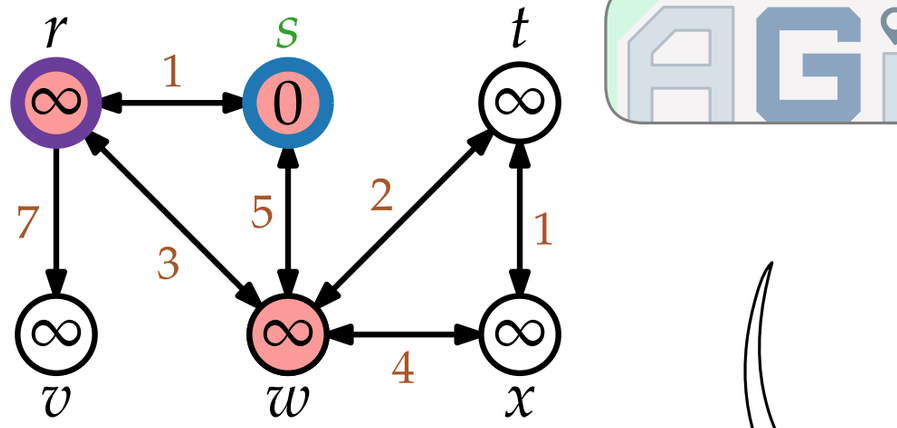


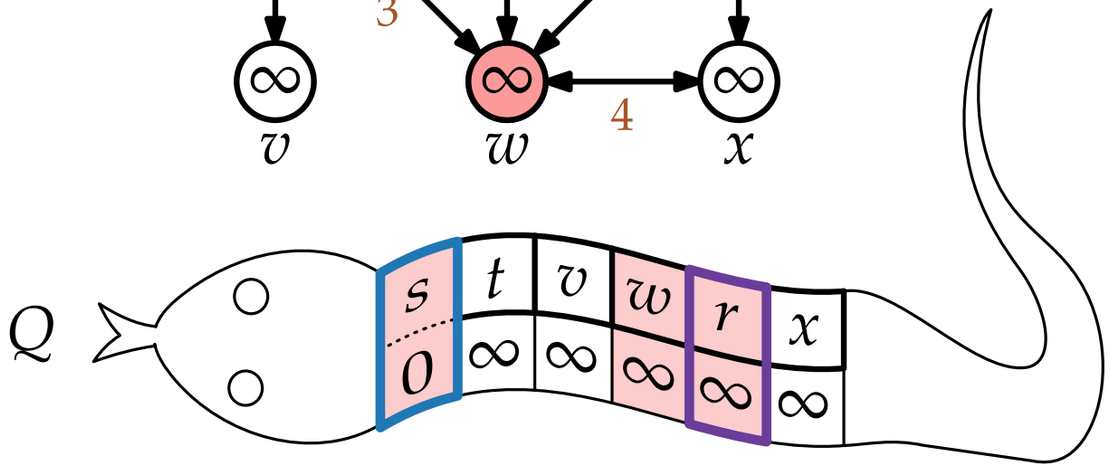



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

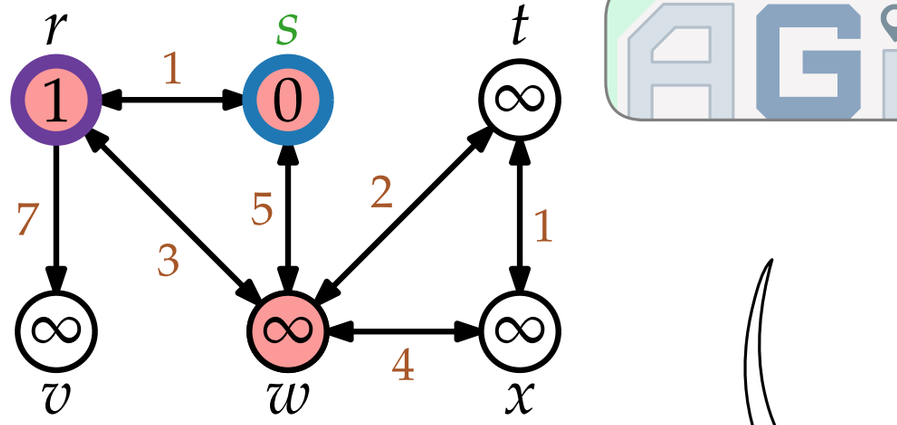


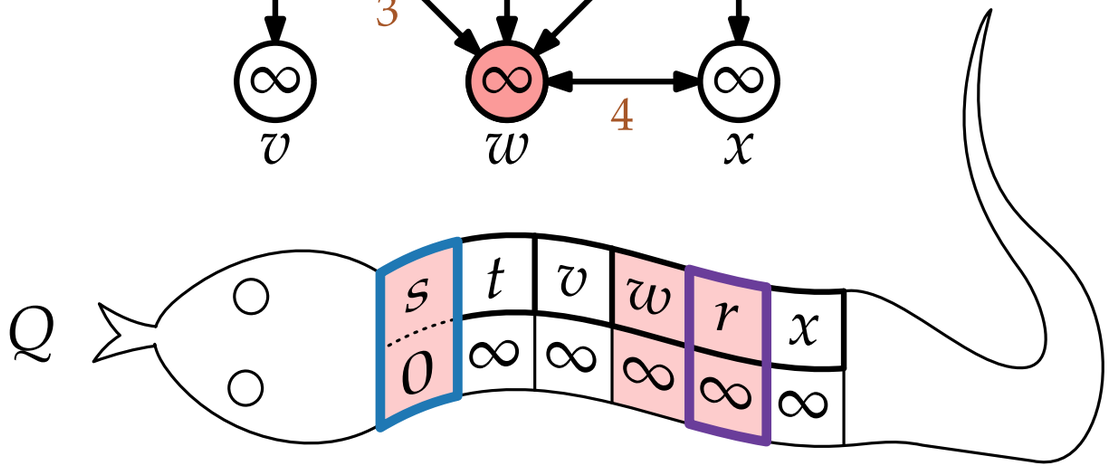



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

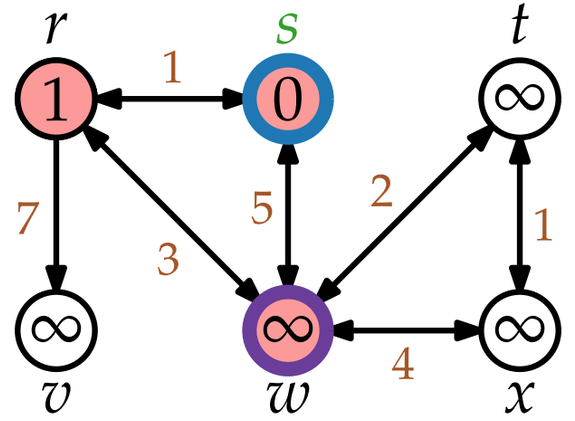


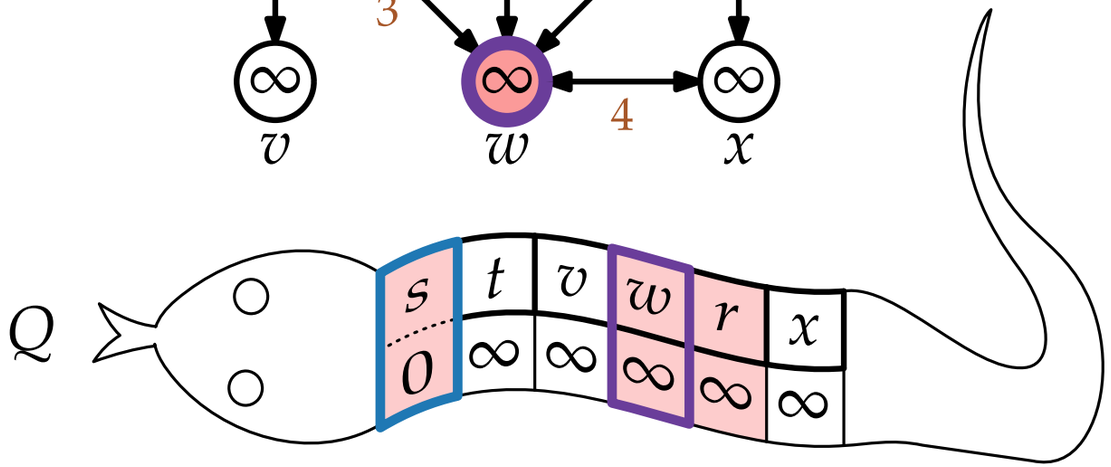



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

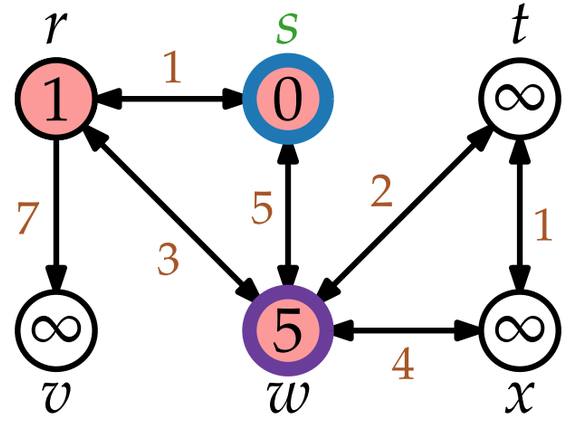


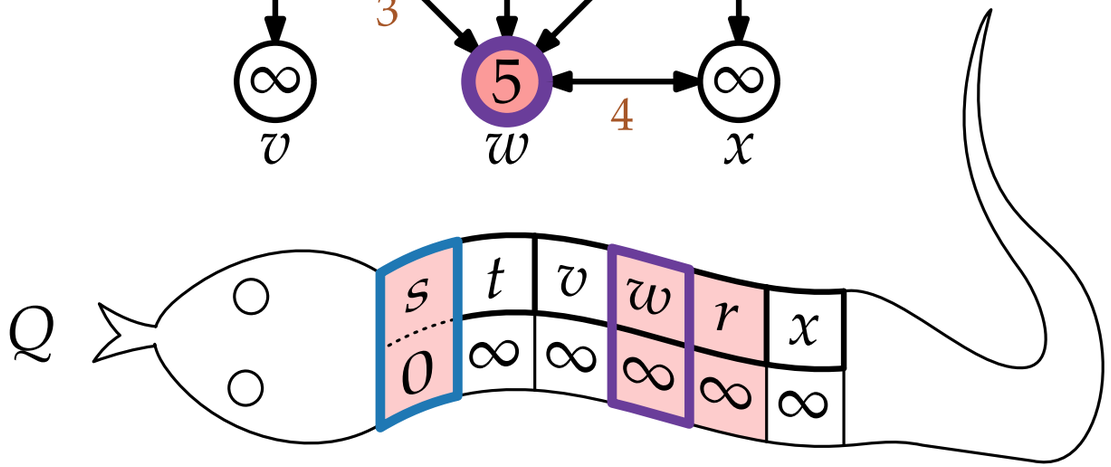



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

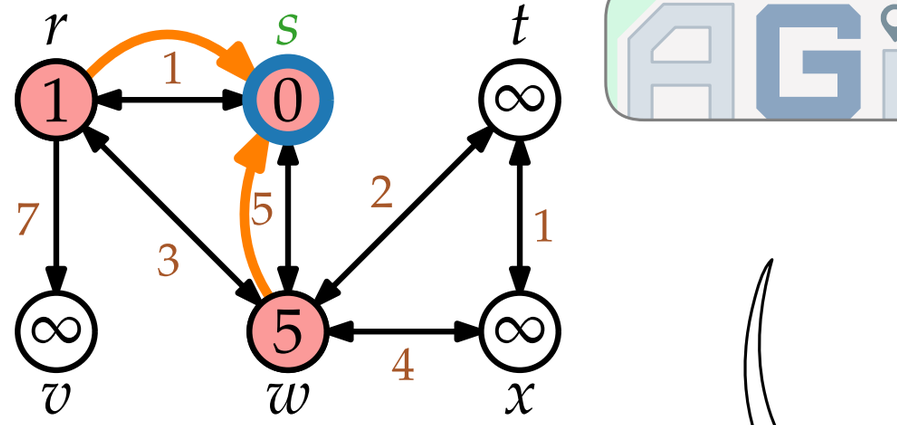


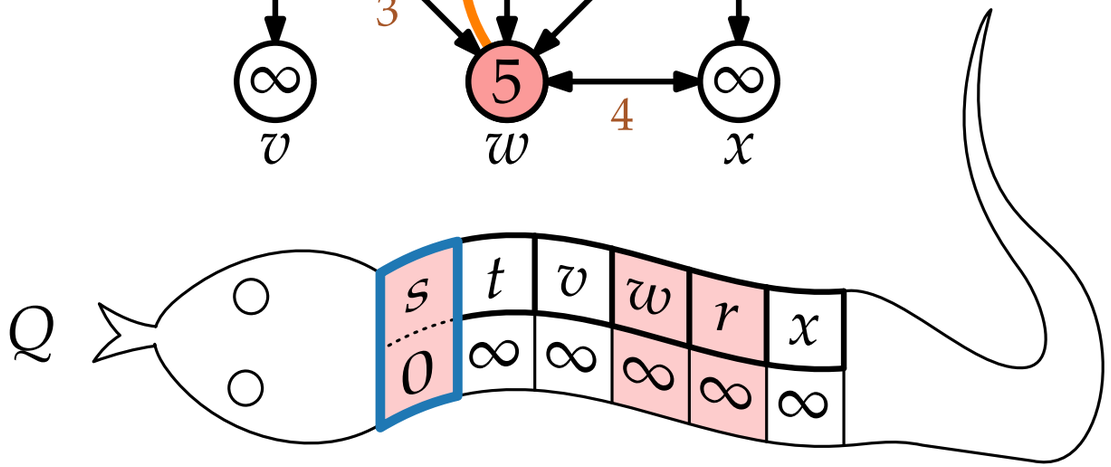



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

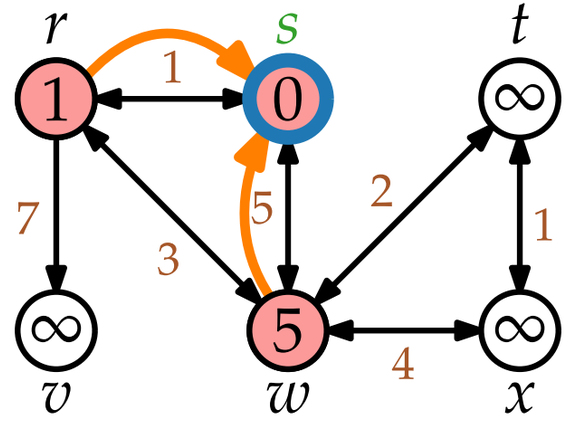


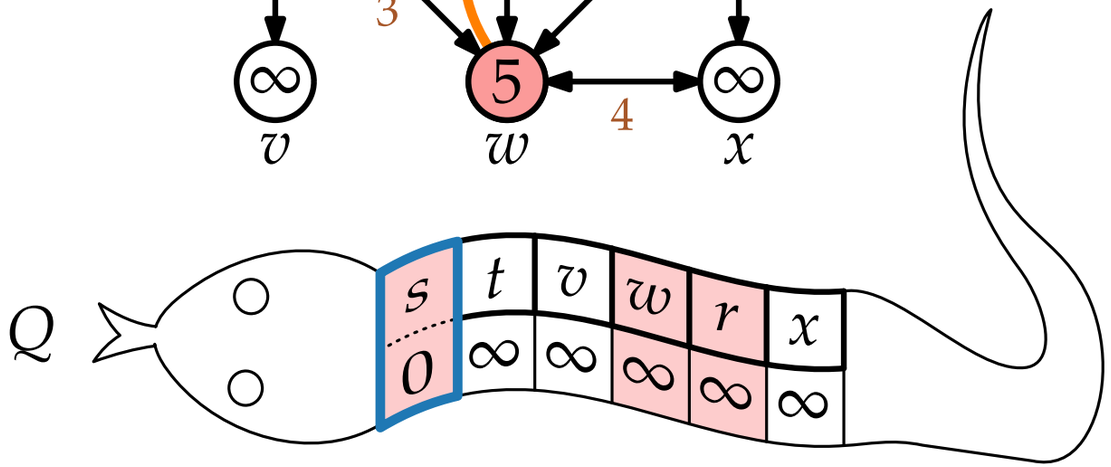



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

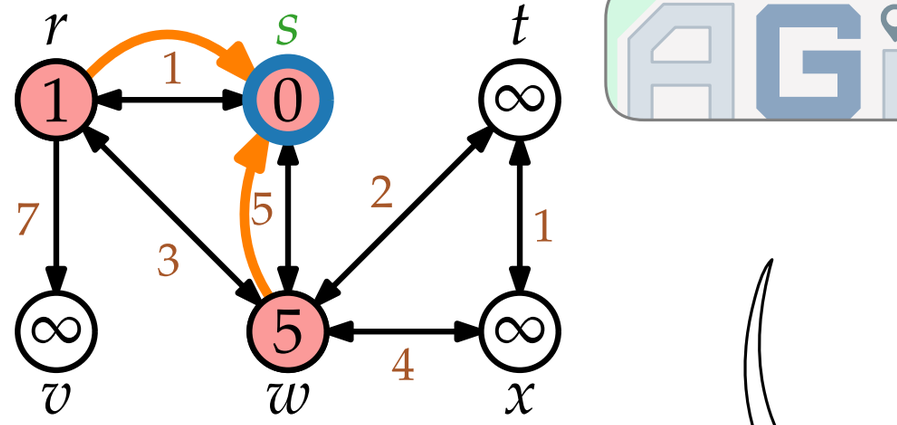


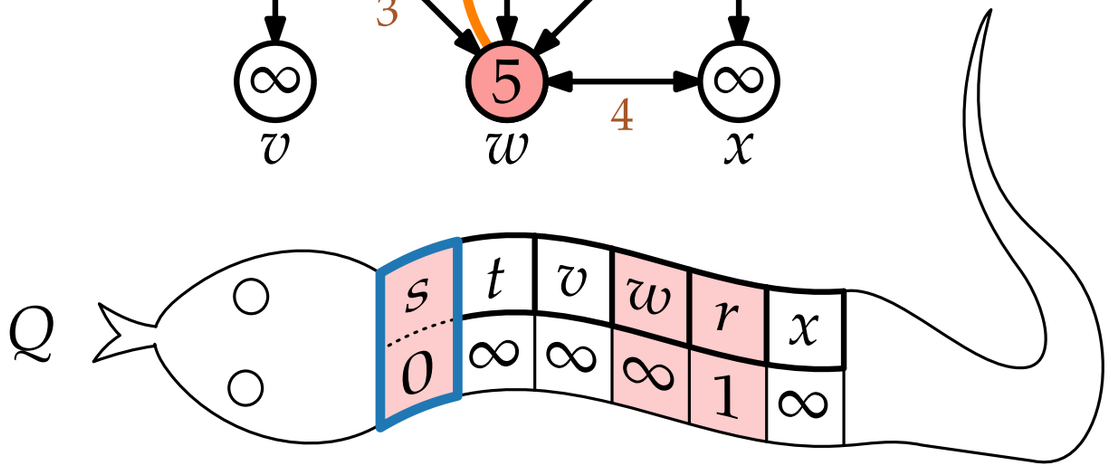



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

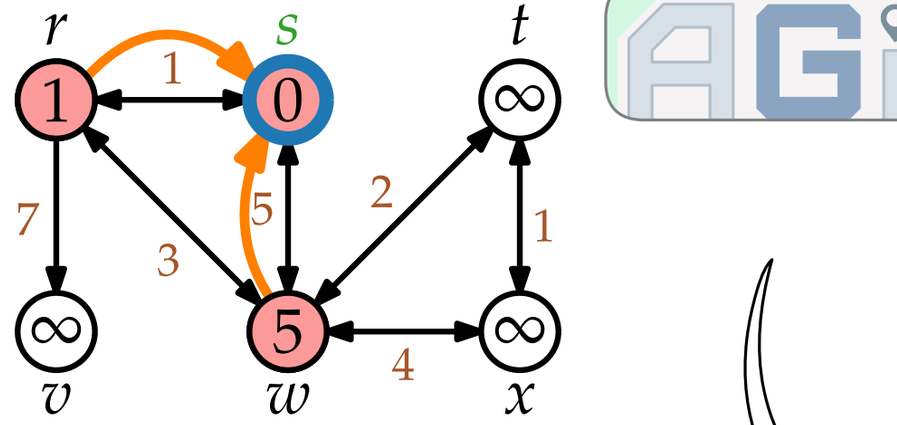


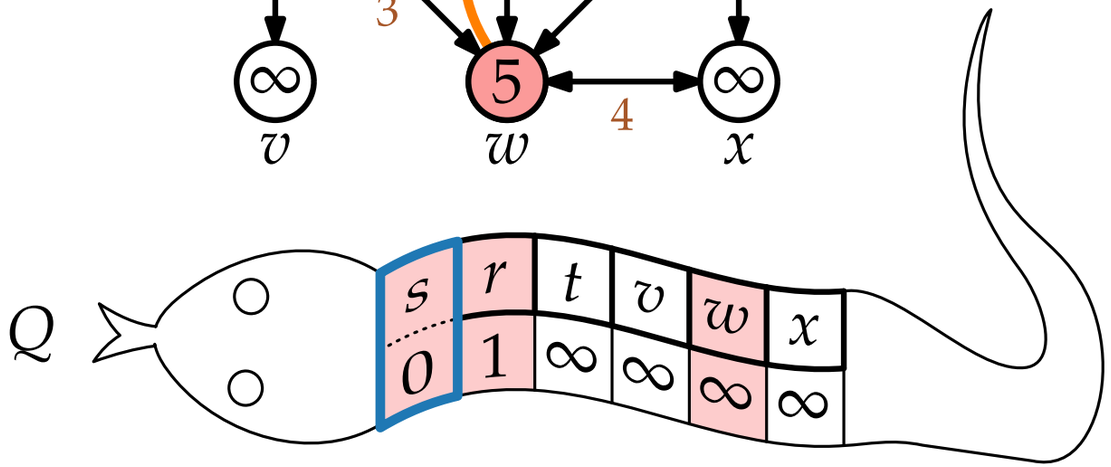



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

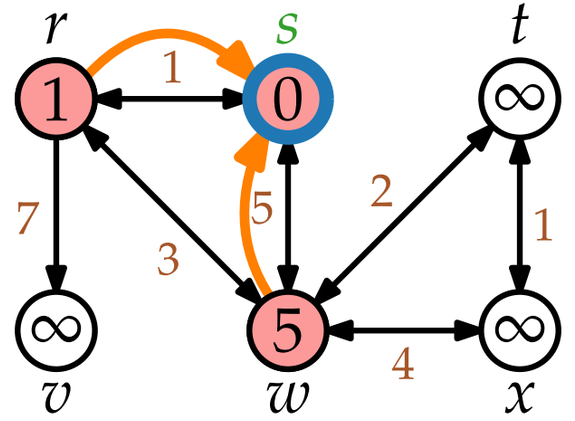


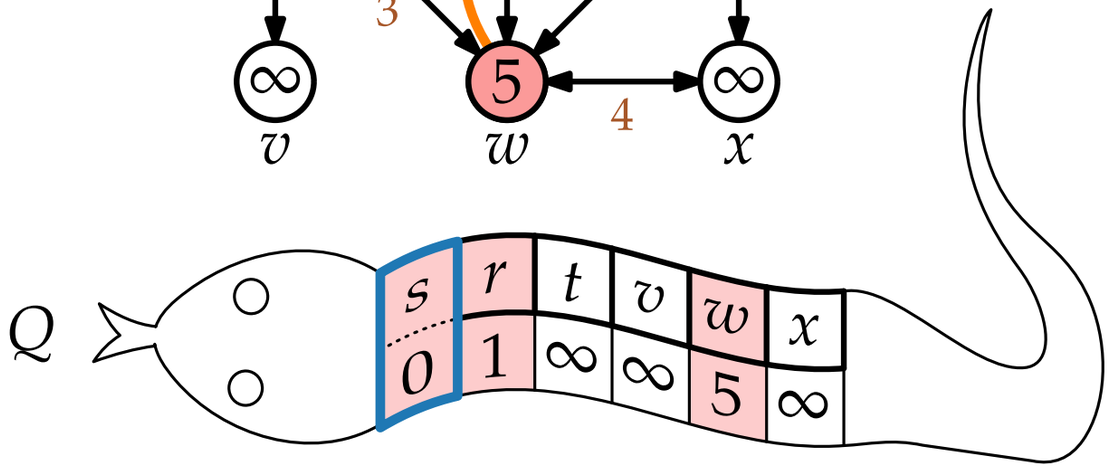



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

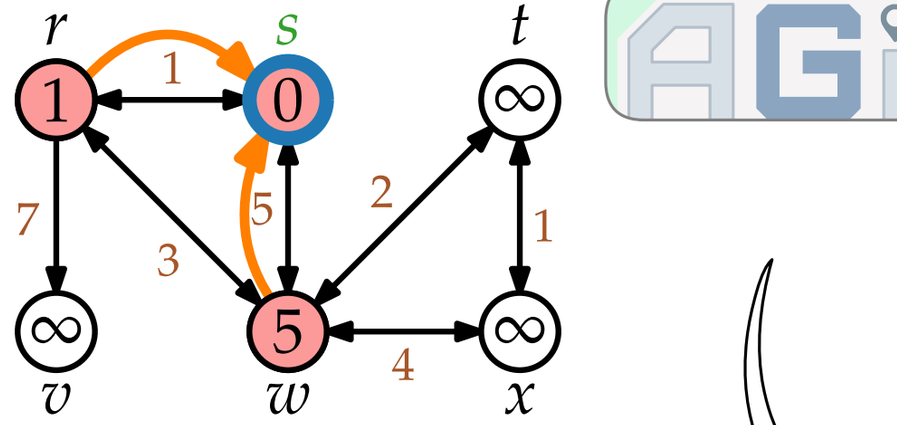


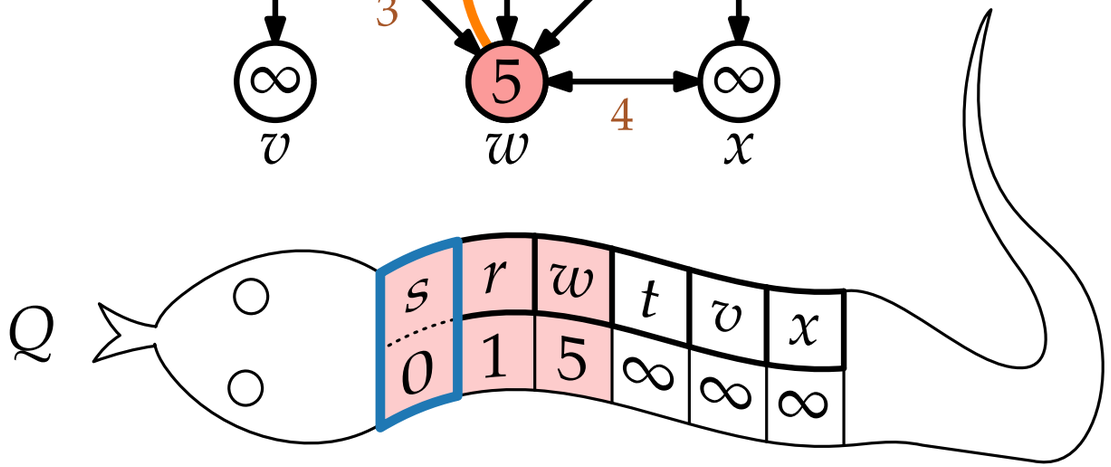



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

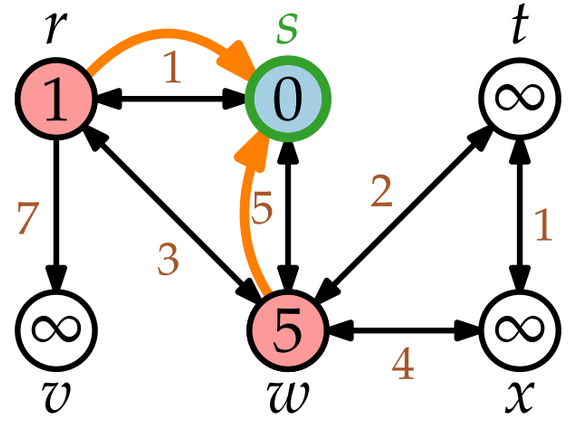


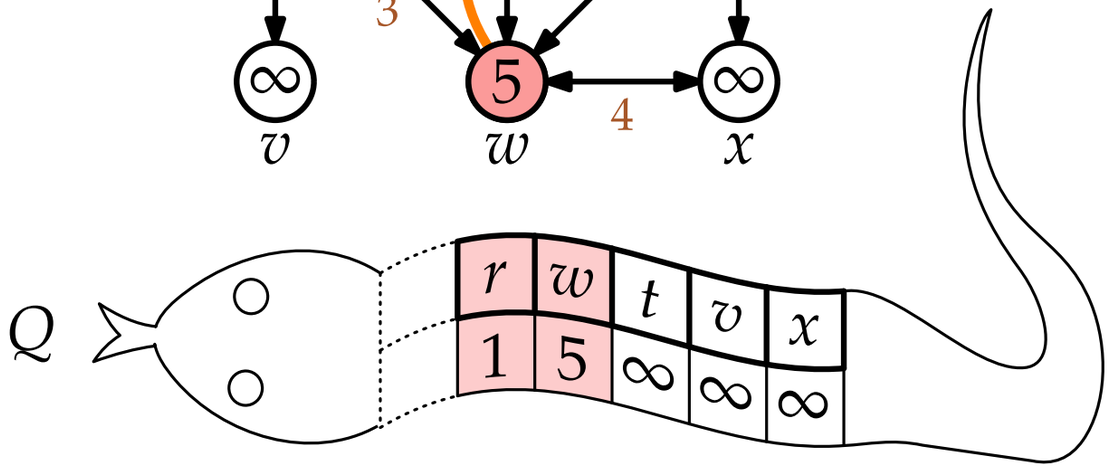



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

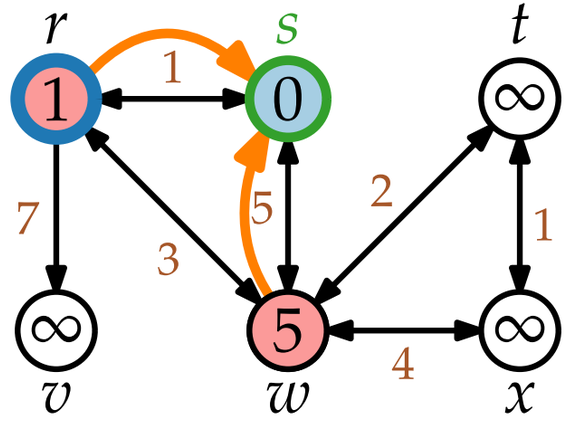


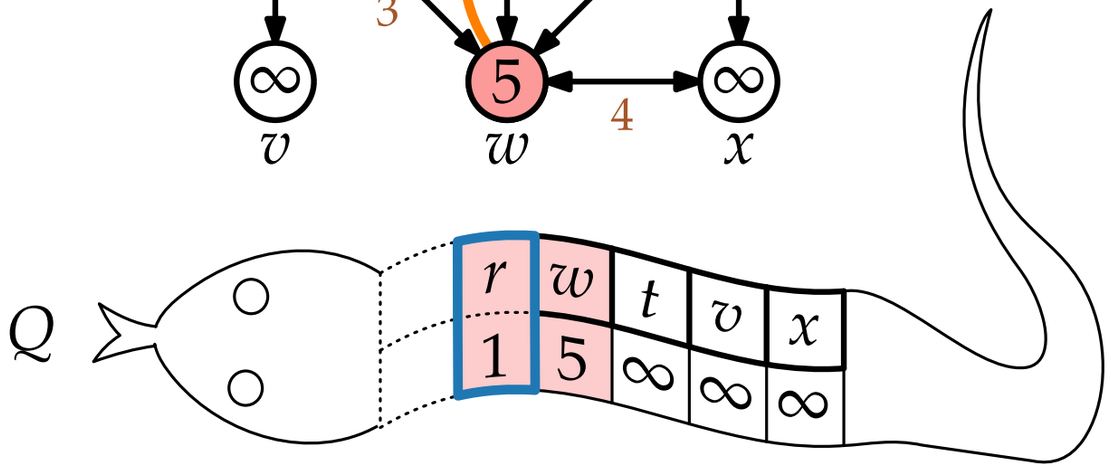



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

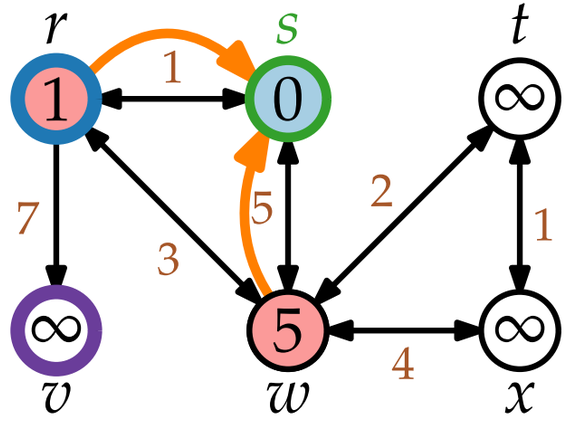


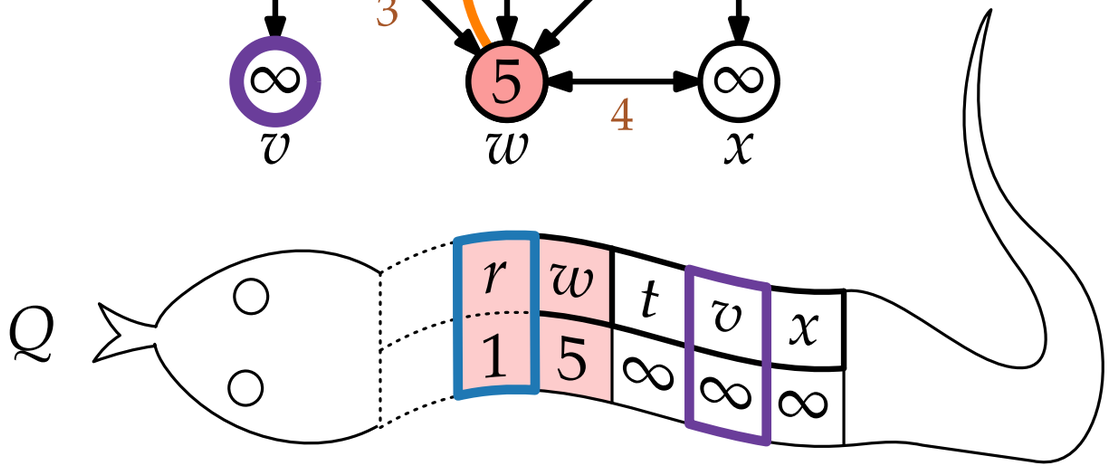



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

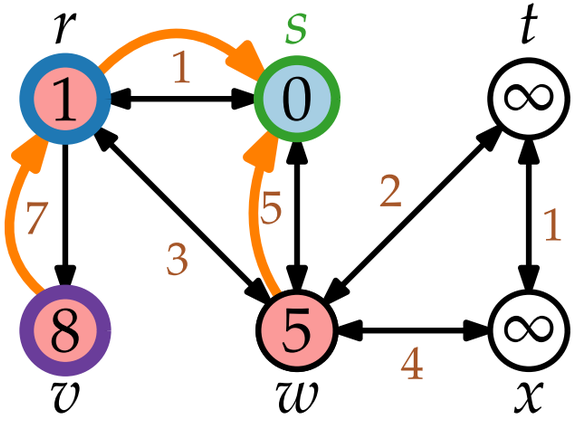


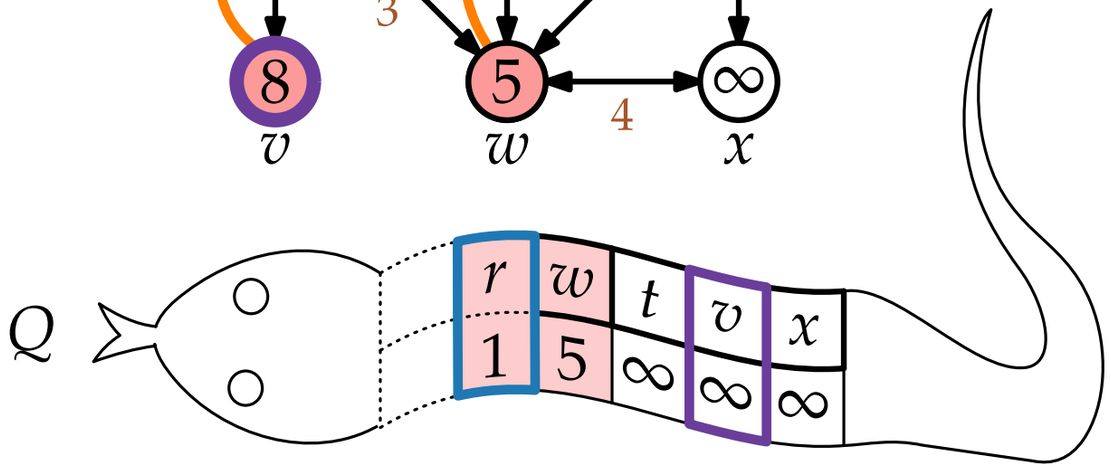



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

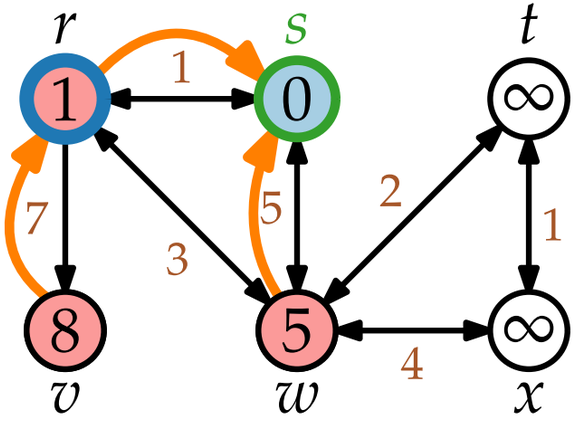


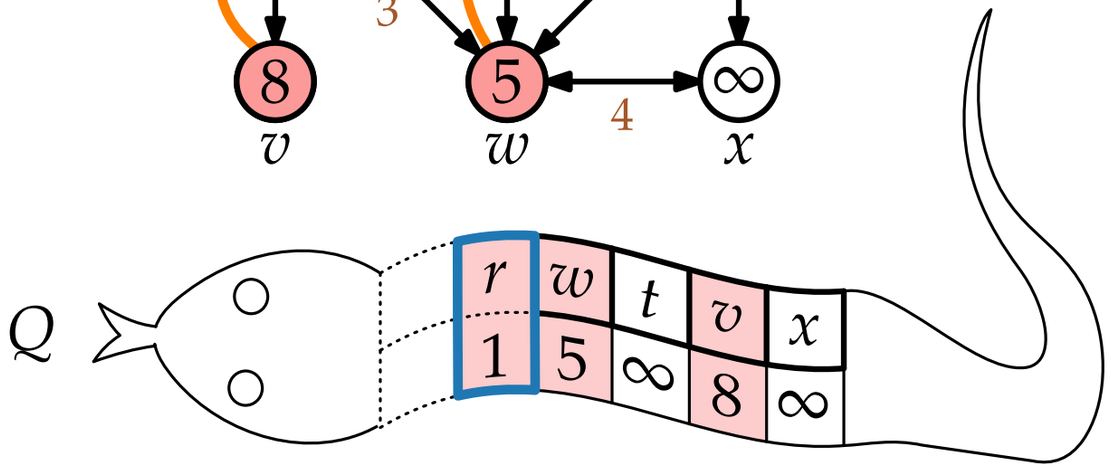



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

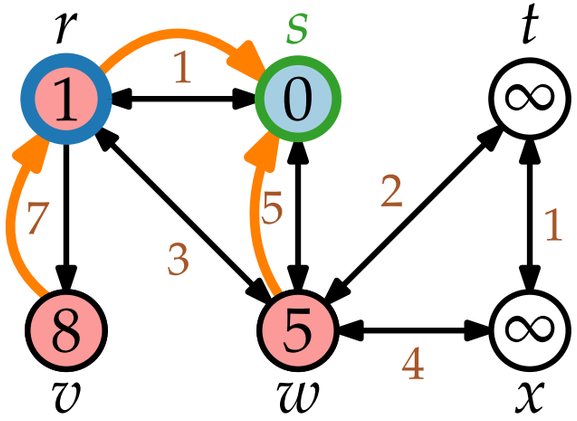


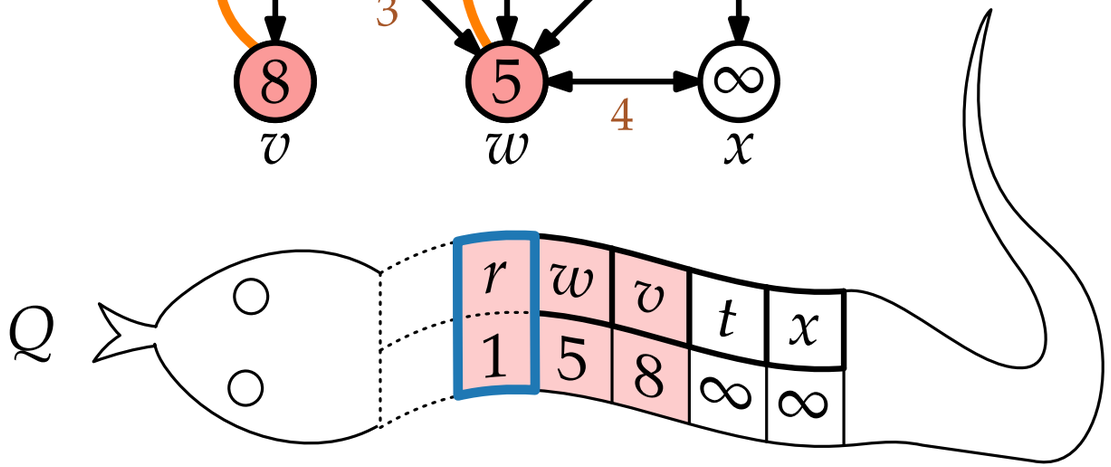



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

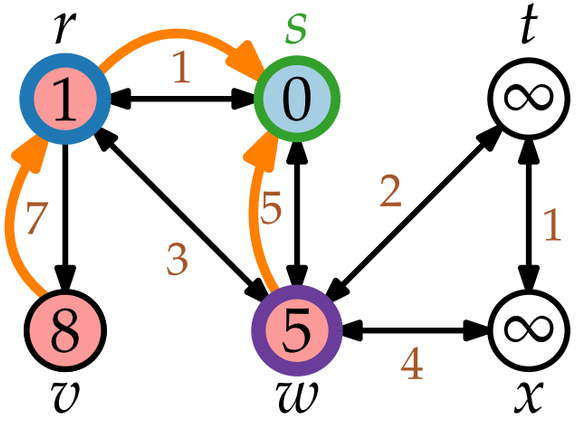


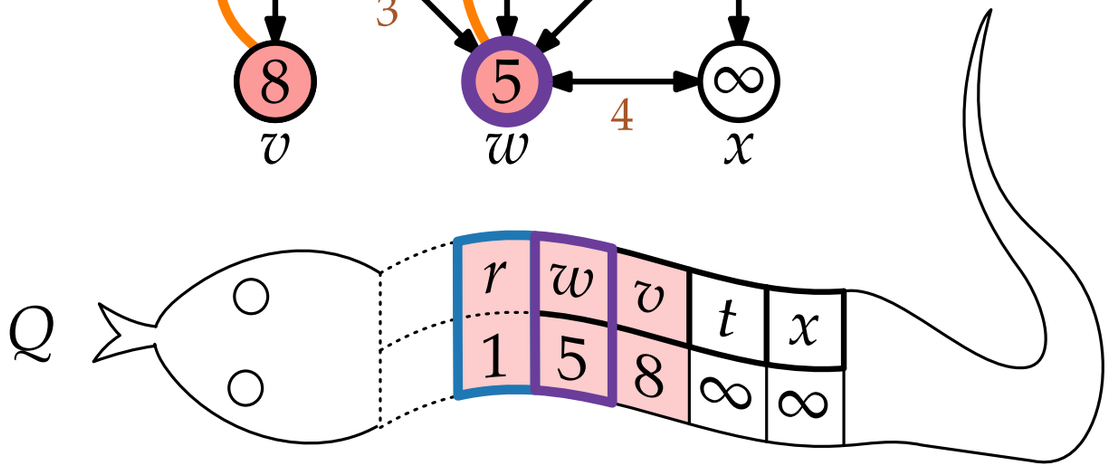



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

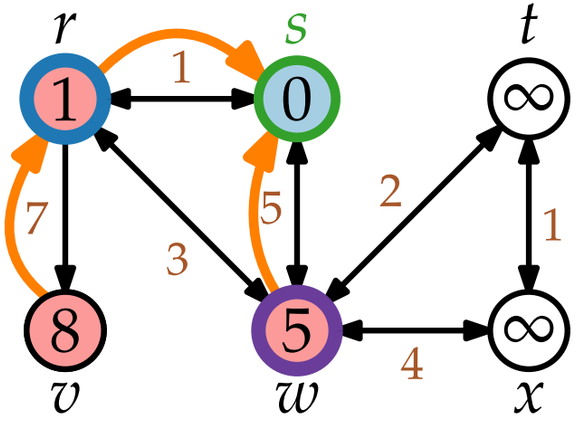


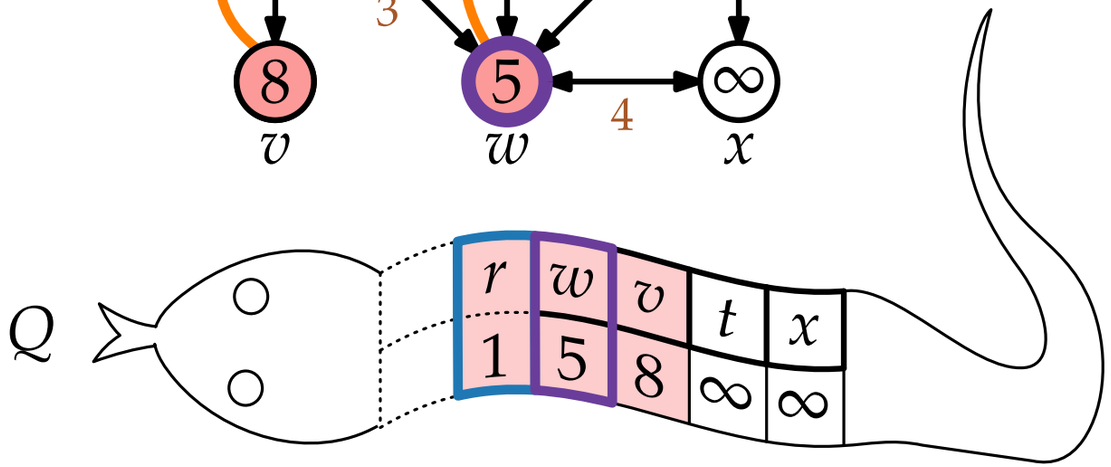



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

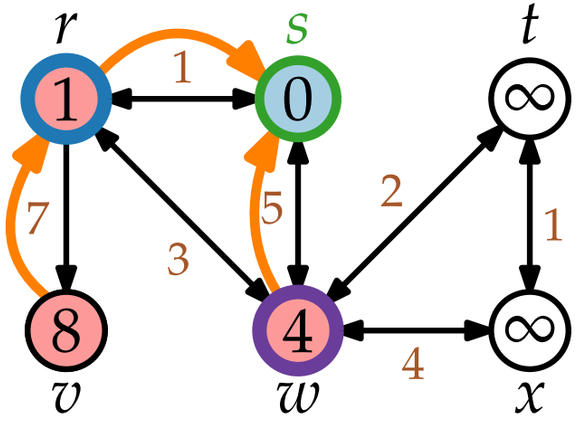


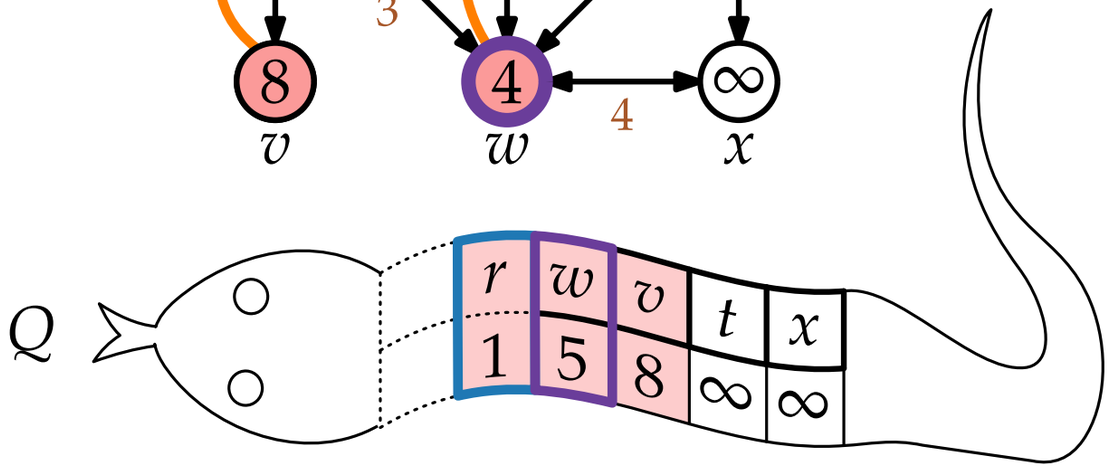



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

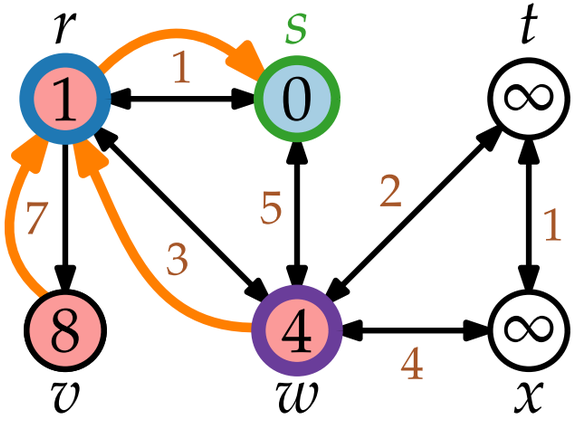


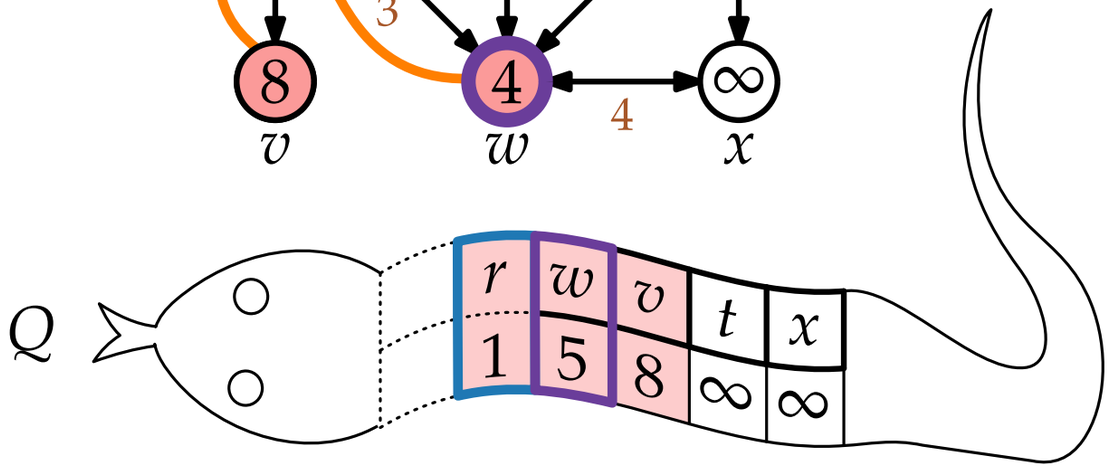



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

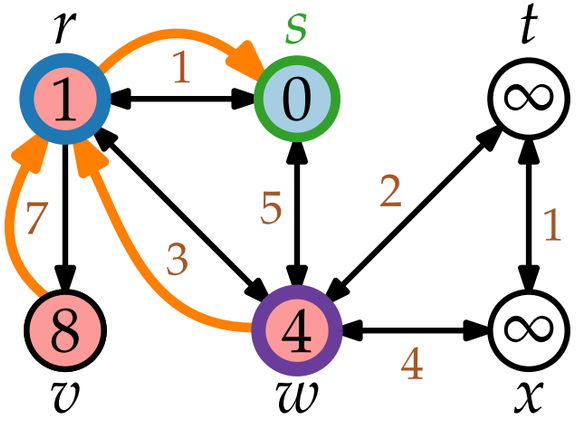


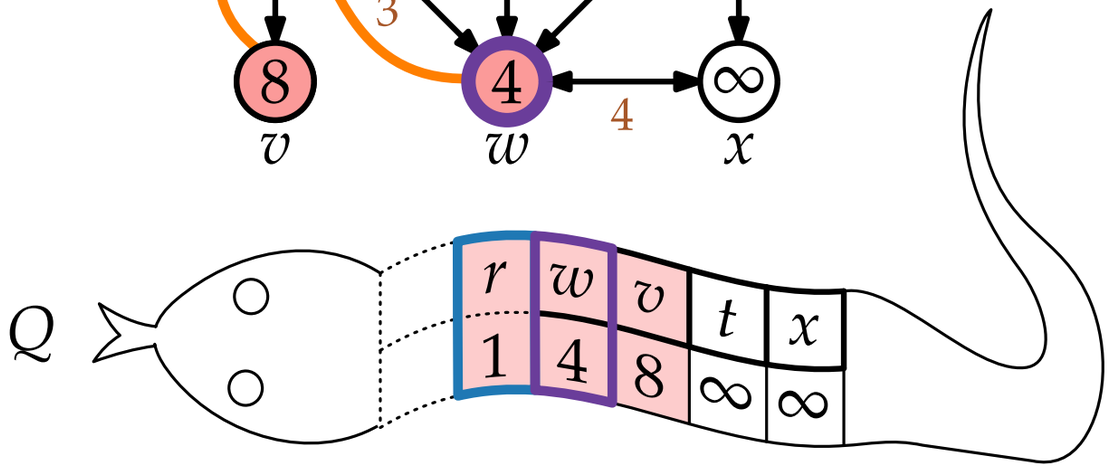



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

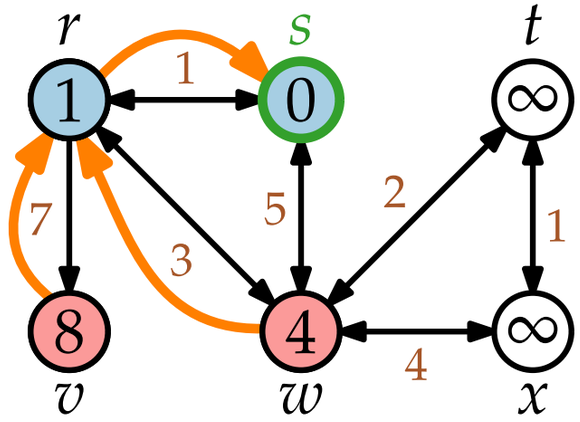


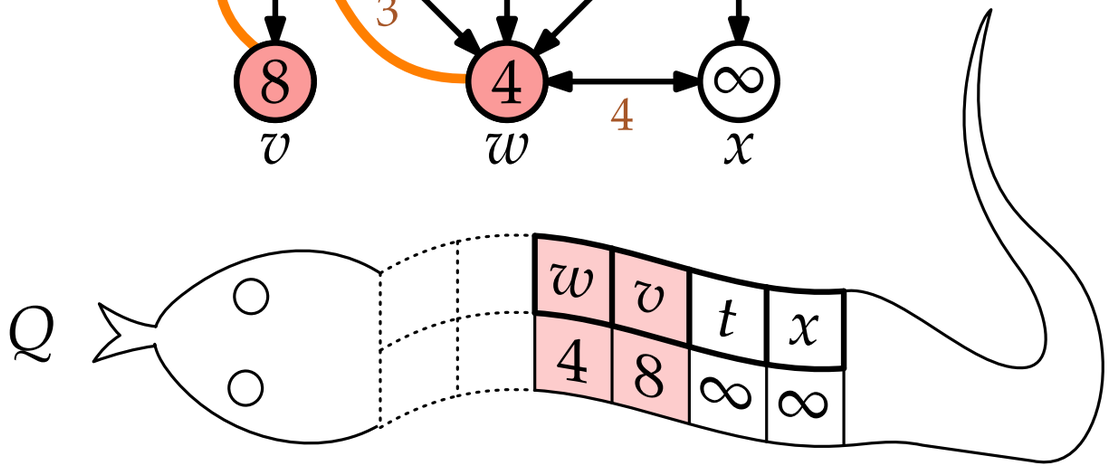



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

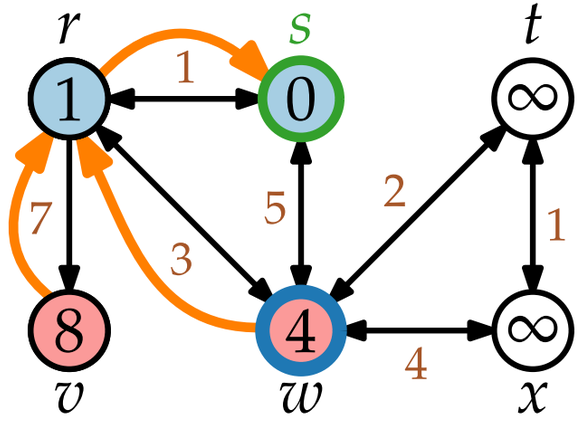


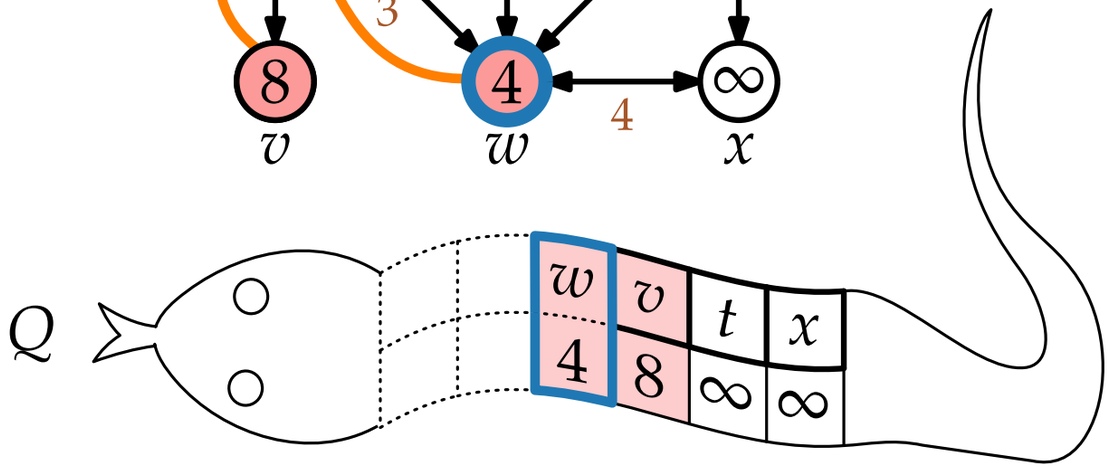



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

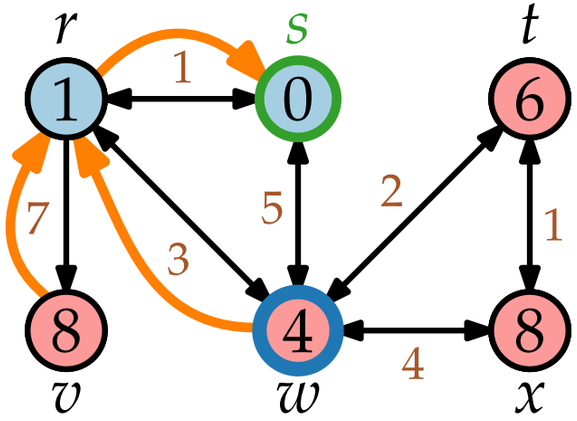


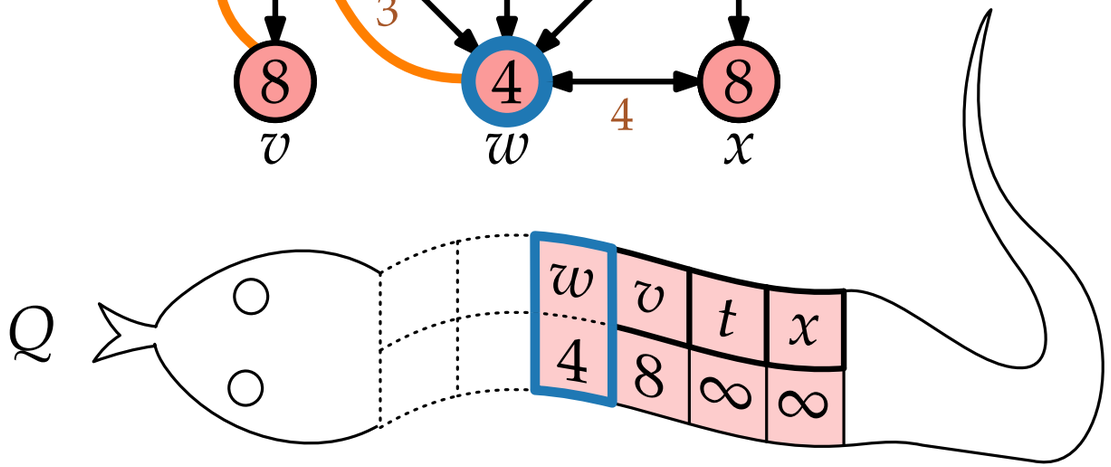



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

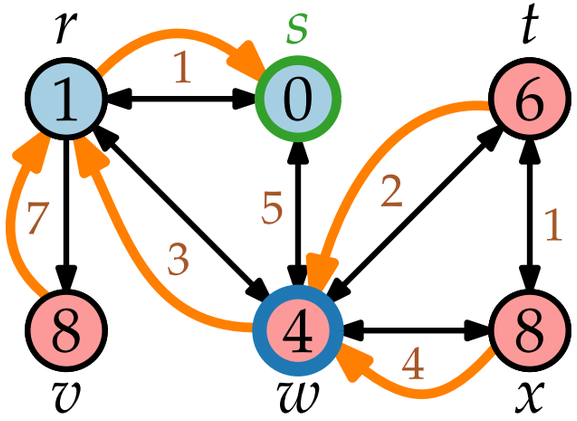


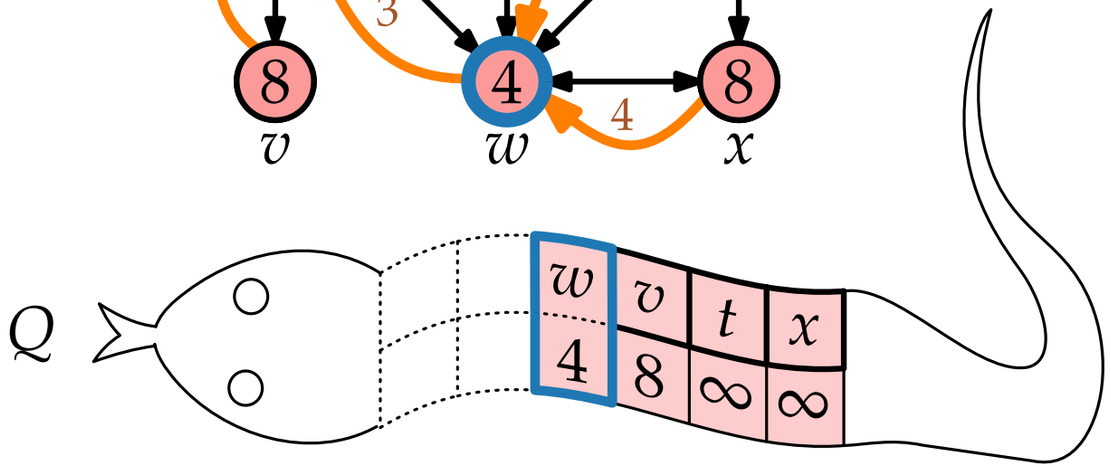



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

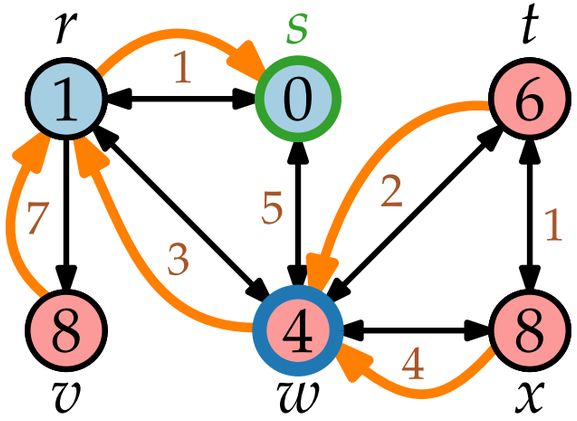


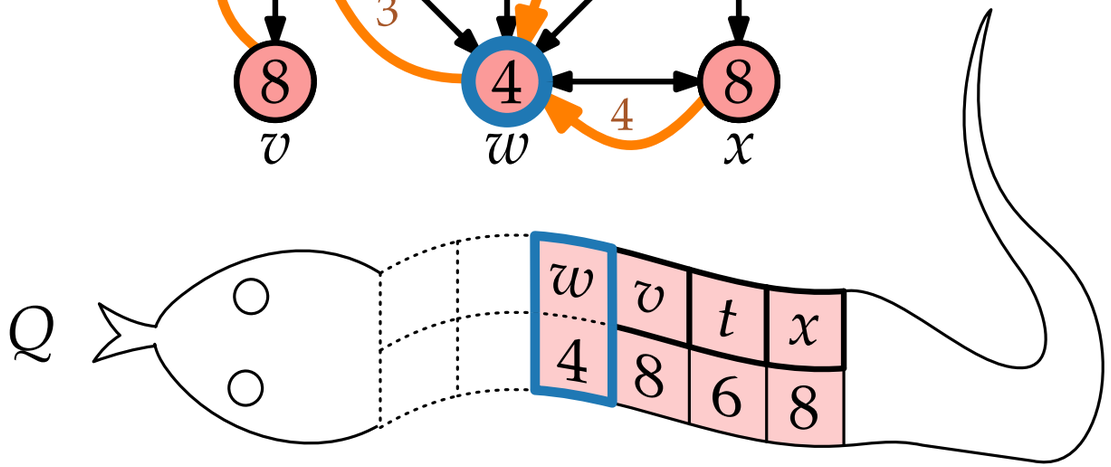



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

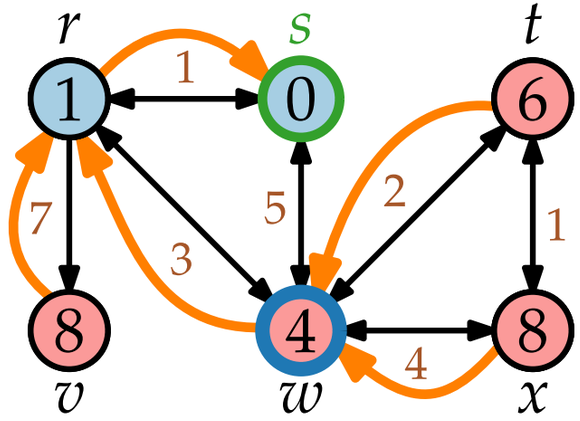


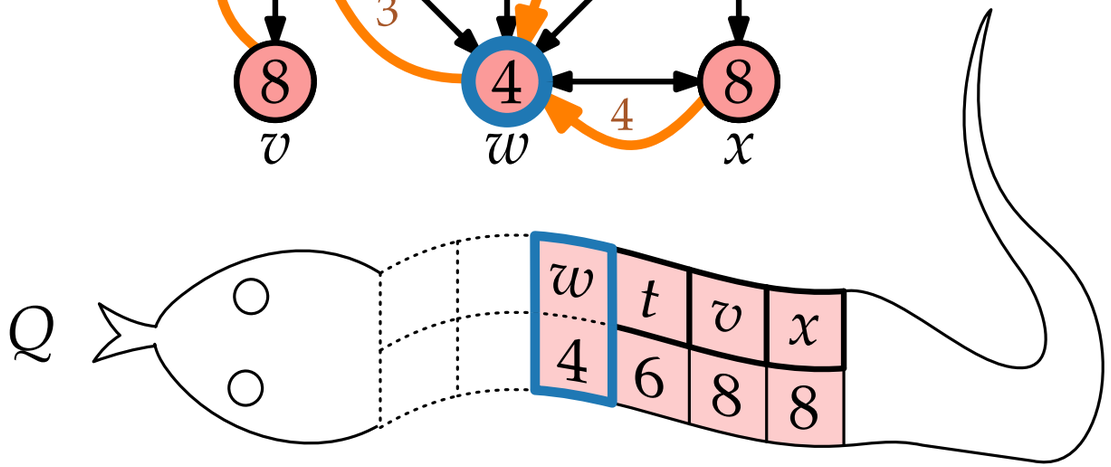



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

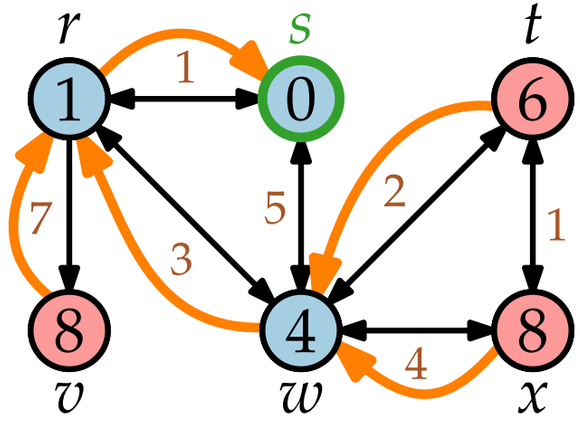


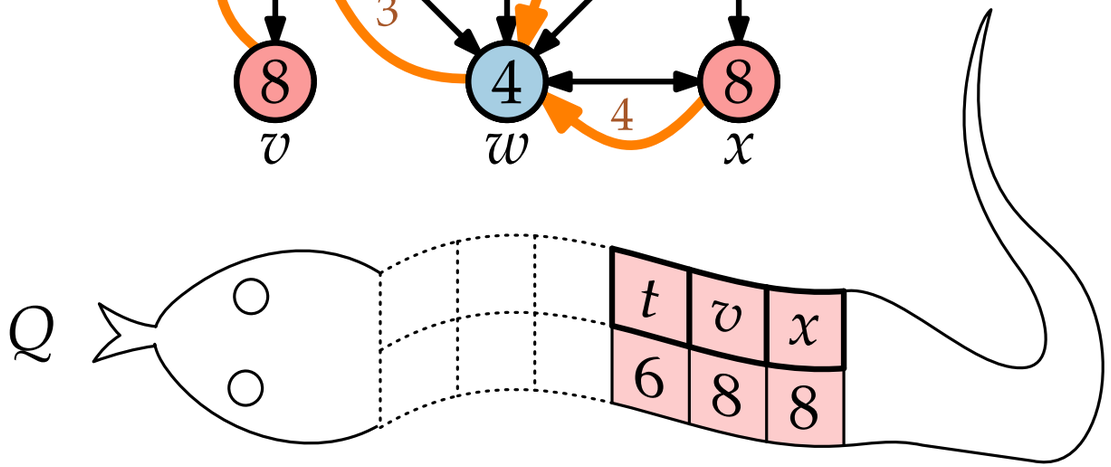



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

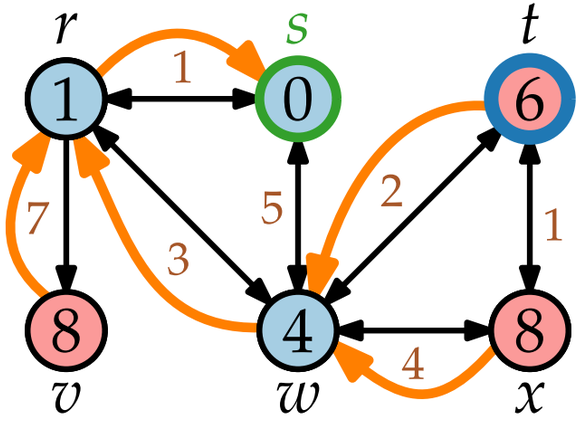


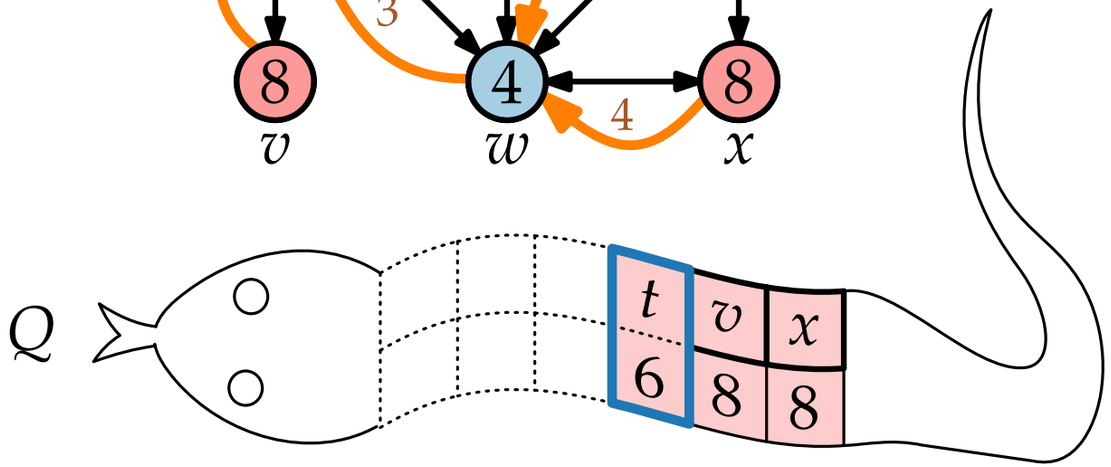



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

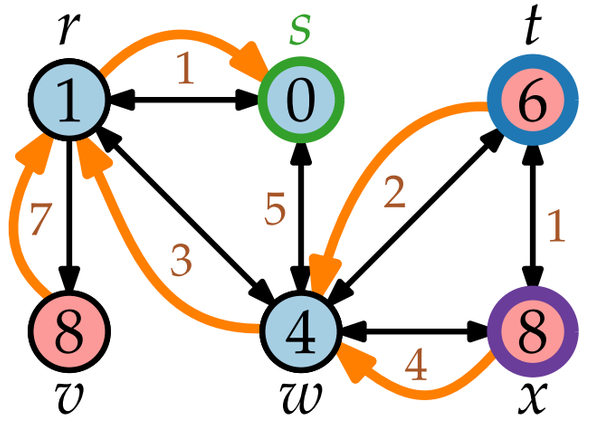


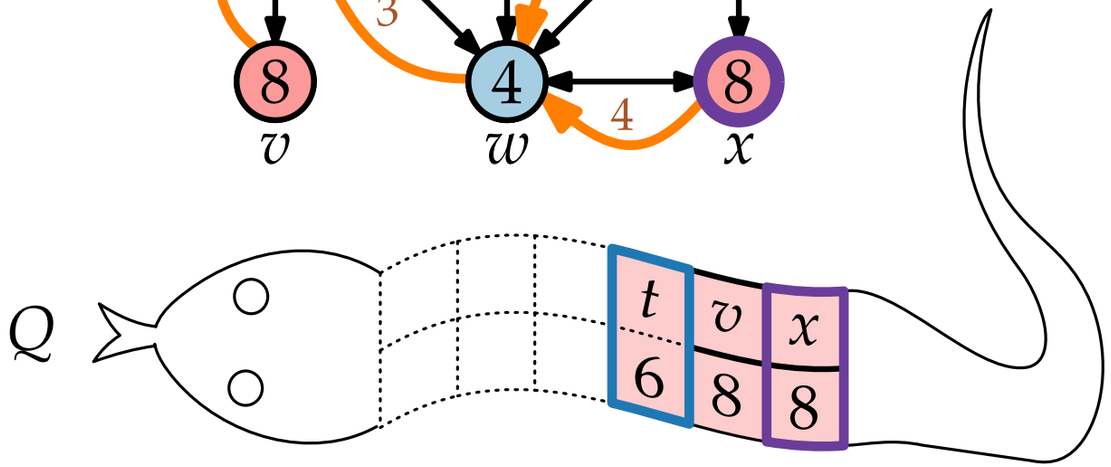



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

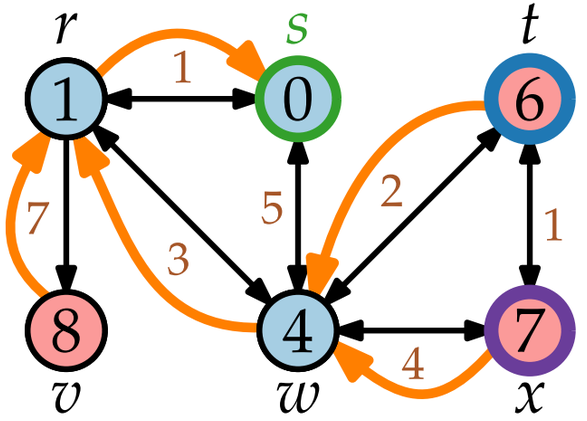


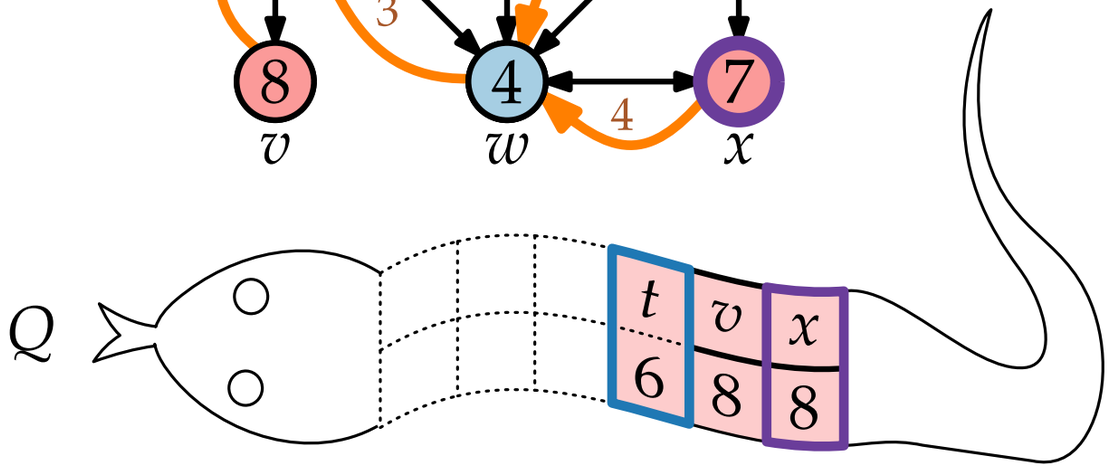



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

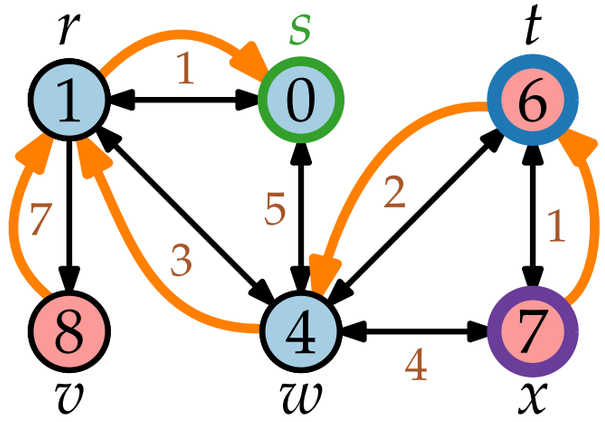


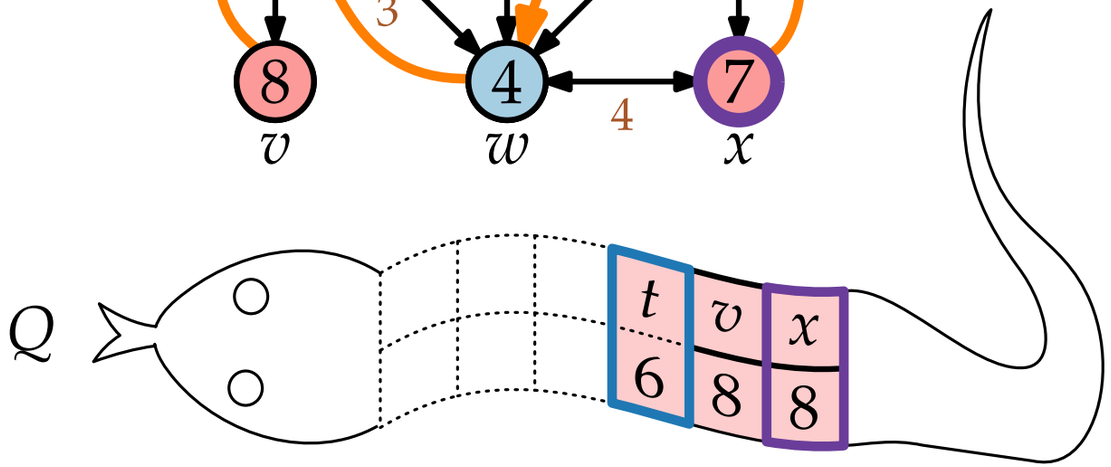



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

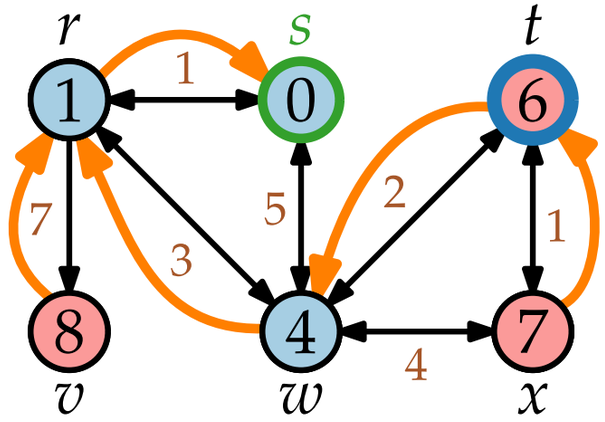


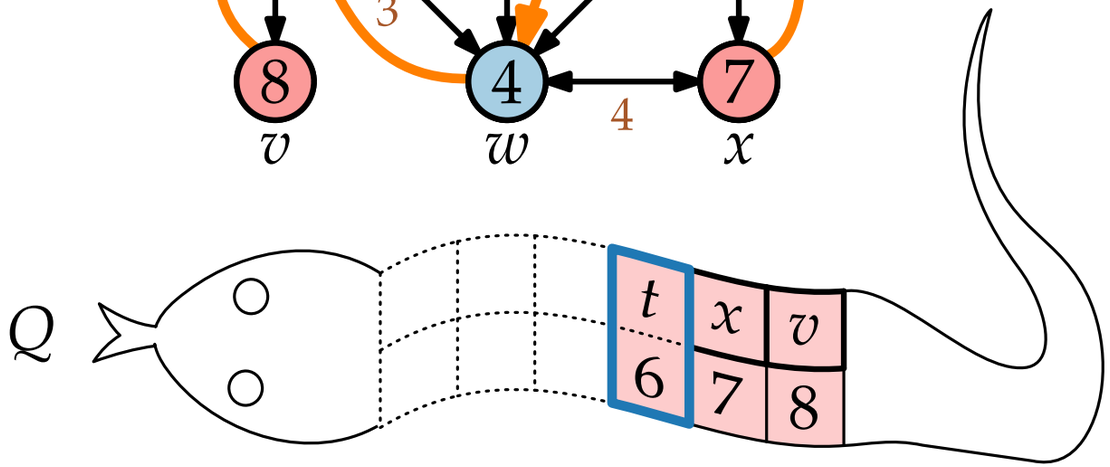



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

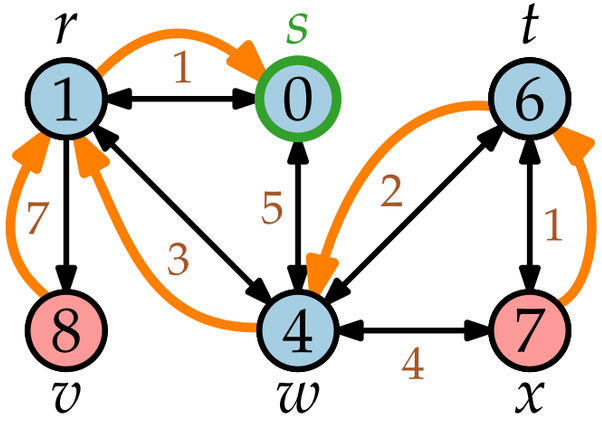


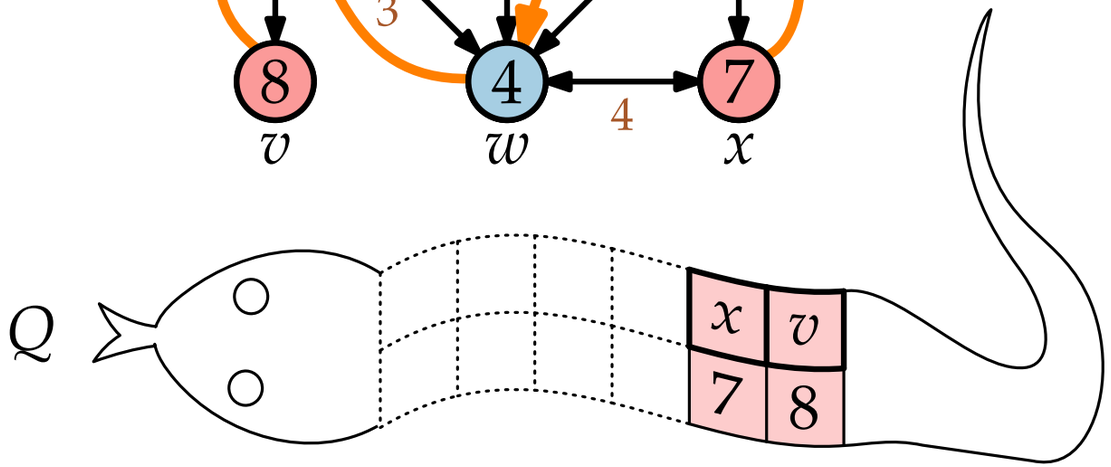



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

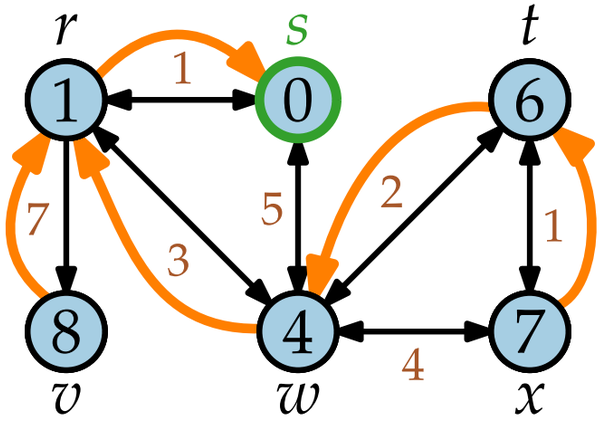
```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

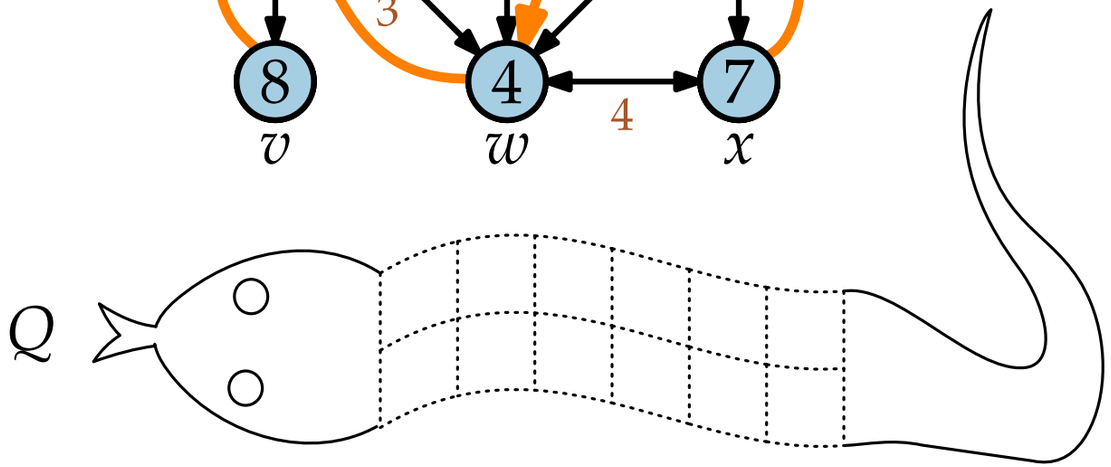





```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

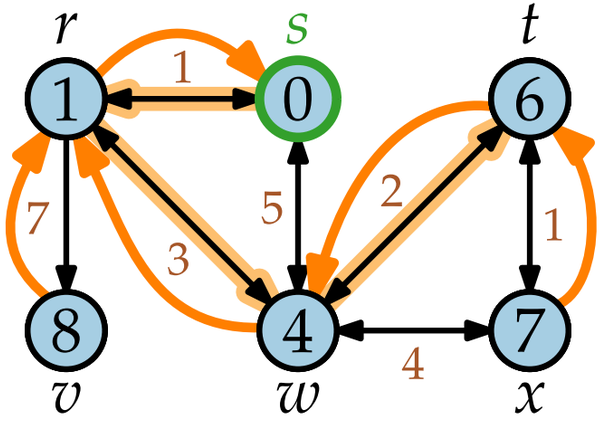


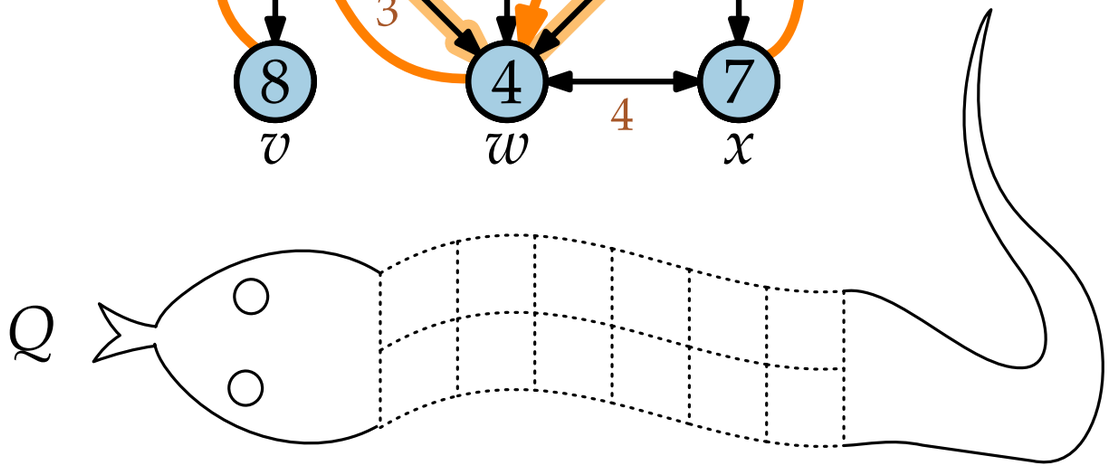



```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra



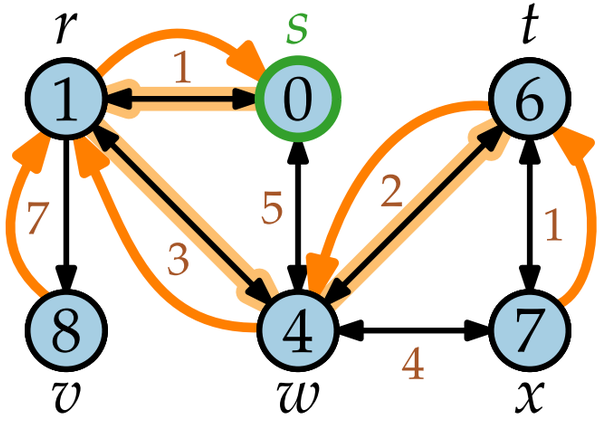


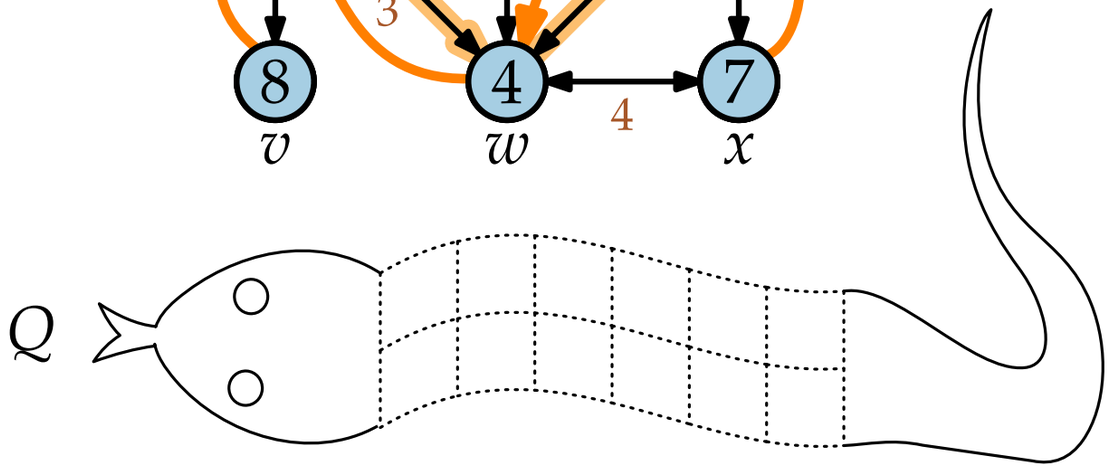

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra

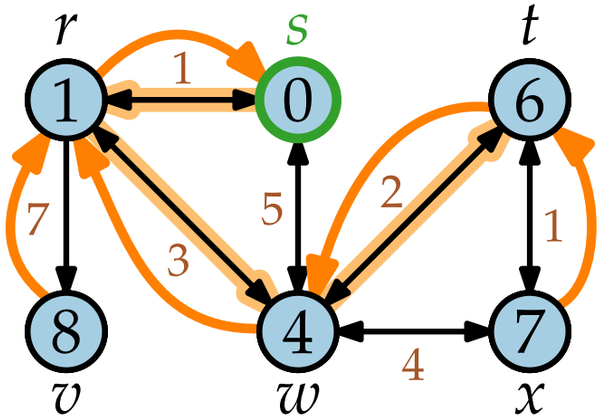
```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

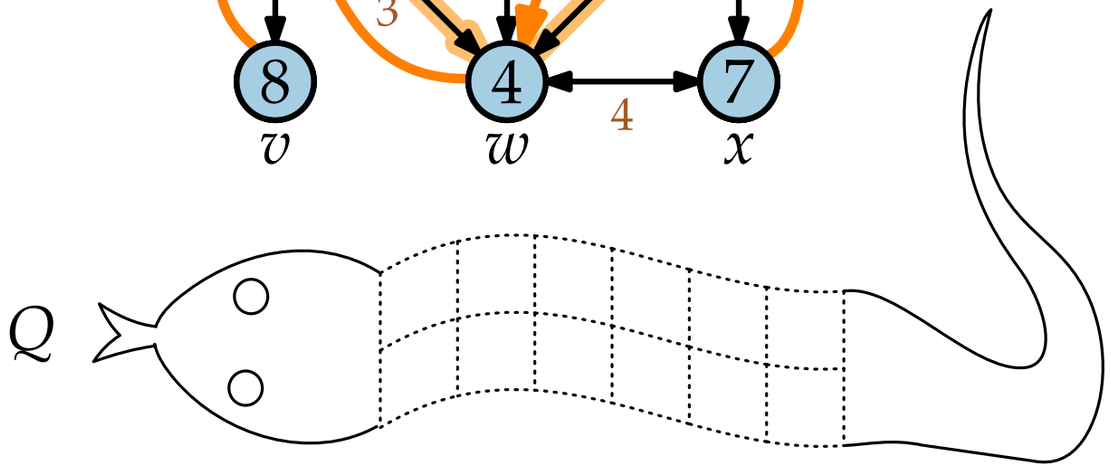





```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Wiederholung – Dijkstra





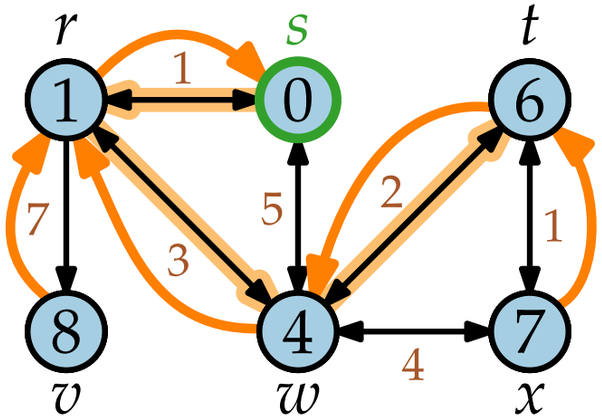

```
Dijkstra(WeightedGraph G, Vertex s)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

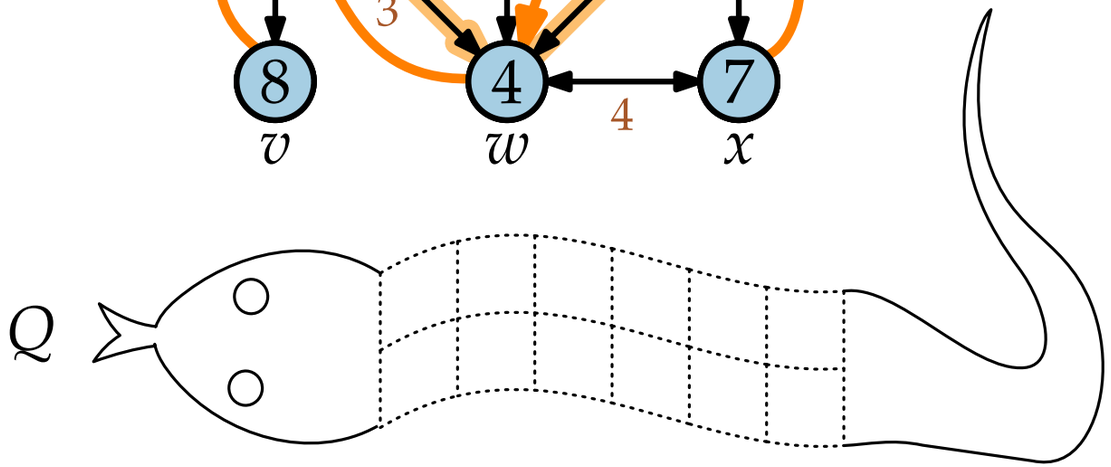
```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Demo.

`https://algo.uni-trier.de/demos/graphtraversal.html`

Wiederholung – Dijkstra





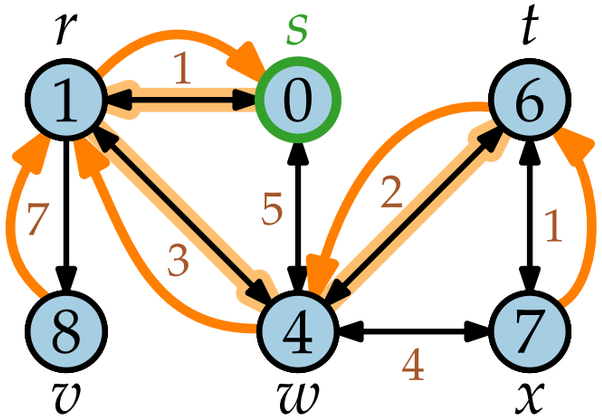

```
Dijkstra(WeightedGraph G, Vertex s, Vertex t)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

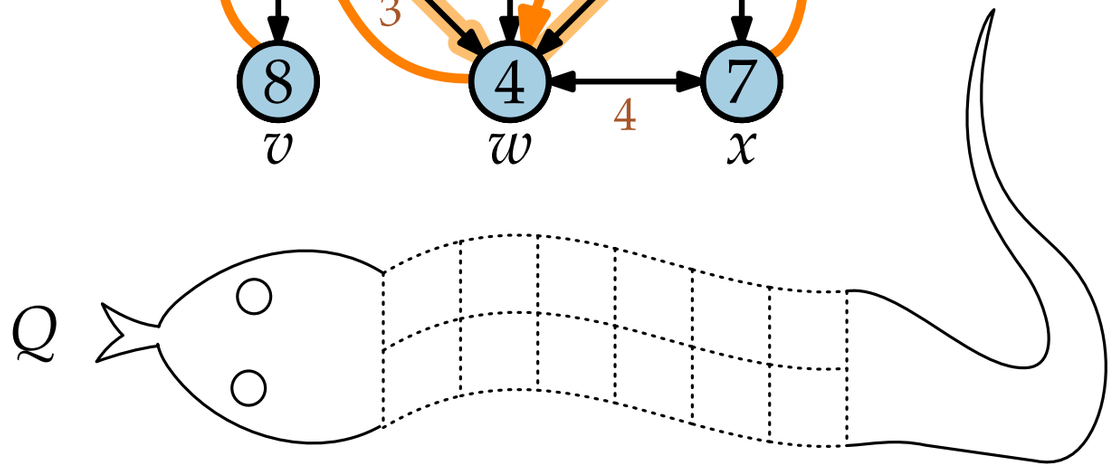
```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Demo.

`https://algo.uni-trier.de/demos/graphtraversal.html`

Wiederholung – Dijkstra





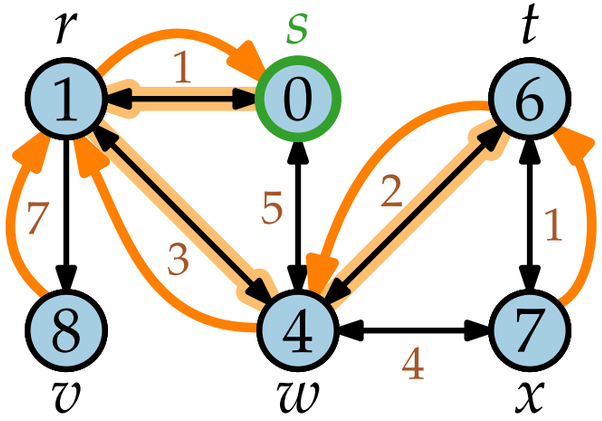

```
Dijkstra(WeightedGraph G, Vertex s, Vertex t)
  Initialize(G, s)
  Q = new PriorityQueue(V, d)
  while not Q.Empty() do
      u = Q.ExtractMin()
      if u == t then return
      foreach v ∈ Adj[u] do
          if v.d > u.d + w(u, v) then
              v.d = u.d + w(u, v)
              v.π = u
              Q.DecreaseKey(v, v.d)
```

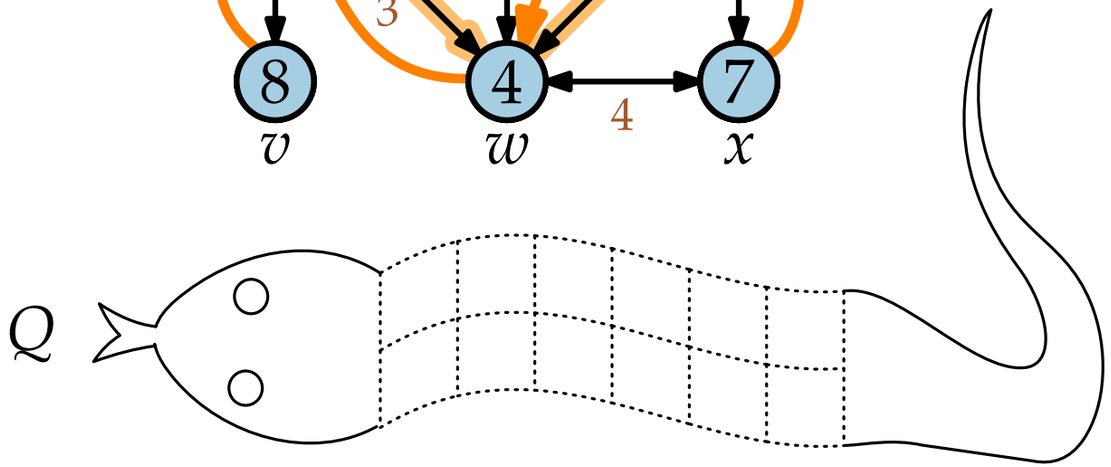
```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.d = ∞
      u.π = nil
  s.d = 0
```

Demo.

`https://algo.uni-trier.de/demos/graphtraversal.html`

Wiederholung – Dijkstra





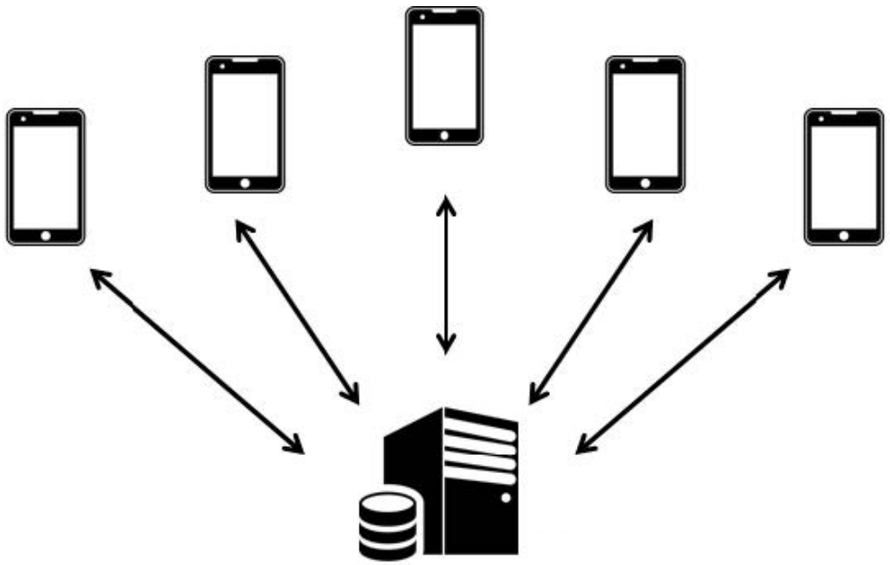

```
Dijkstra(WeightedGraph G, Vertex s, Vertex t)
    Initialize(G, s)
    Q = new PriorityQueue(V, d)
    while not Q.Empty() do
        u = Q.ExtractMin()
        if u == t then return
        foreach v ∈ Adj[u] do
            if v.d > u.d + w(u, v) then
                v.d = u.d + w(u, v)
                v.π = u
                Q.DecreaseKey(v, v.d)
```

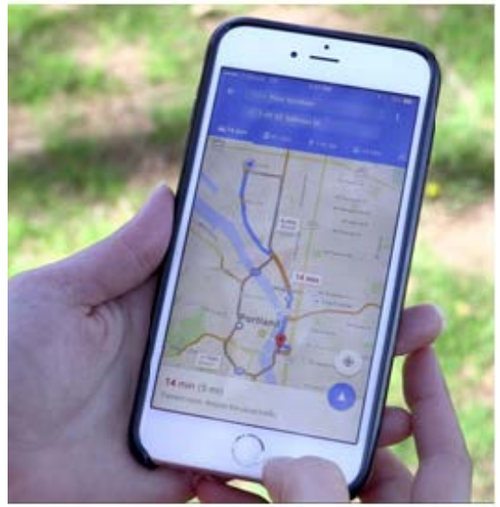
```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.d = ∞
        u.π = nil
    s.d = 0
```

Satz. Dijkstra berechnet in einem Graphen $G = (V, E; w)$ mit $w\colon E \to \mathbb{Q}_{\geq 0}$ in $\mathcal{O}(E + V \log V)$ Zeit den kürzesten Weg zwischen zwei Knoten.

Motivation

„Jeden Tag werden mit den Google Maps 1 Milliarde Kilometer navigiert“
(https://www.googlewatchblog.de/2017/05/offizielle-statistiken-google-nutzerzahlen/)

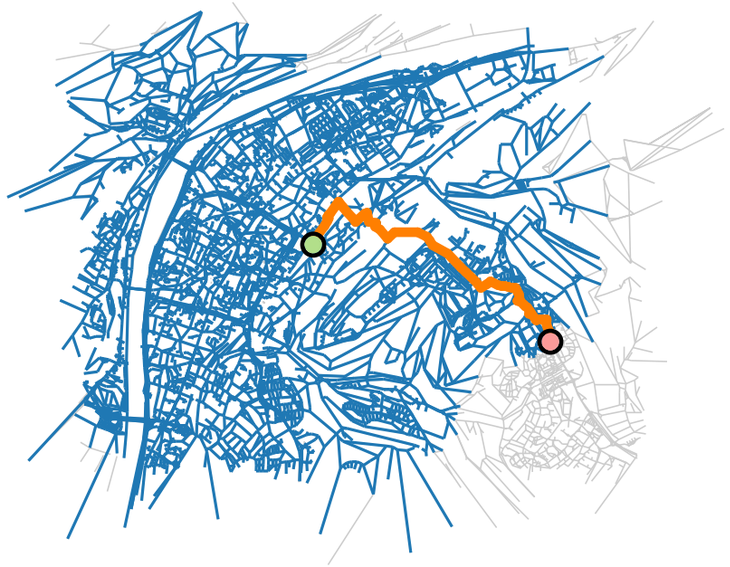




Universität Trier

Algorithmen für geographische Informationssysteme

2. Vorlesung
Routenplanung

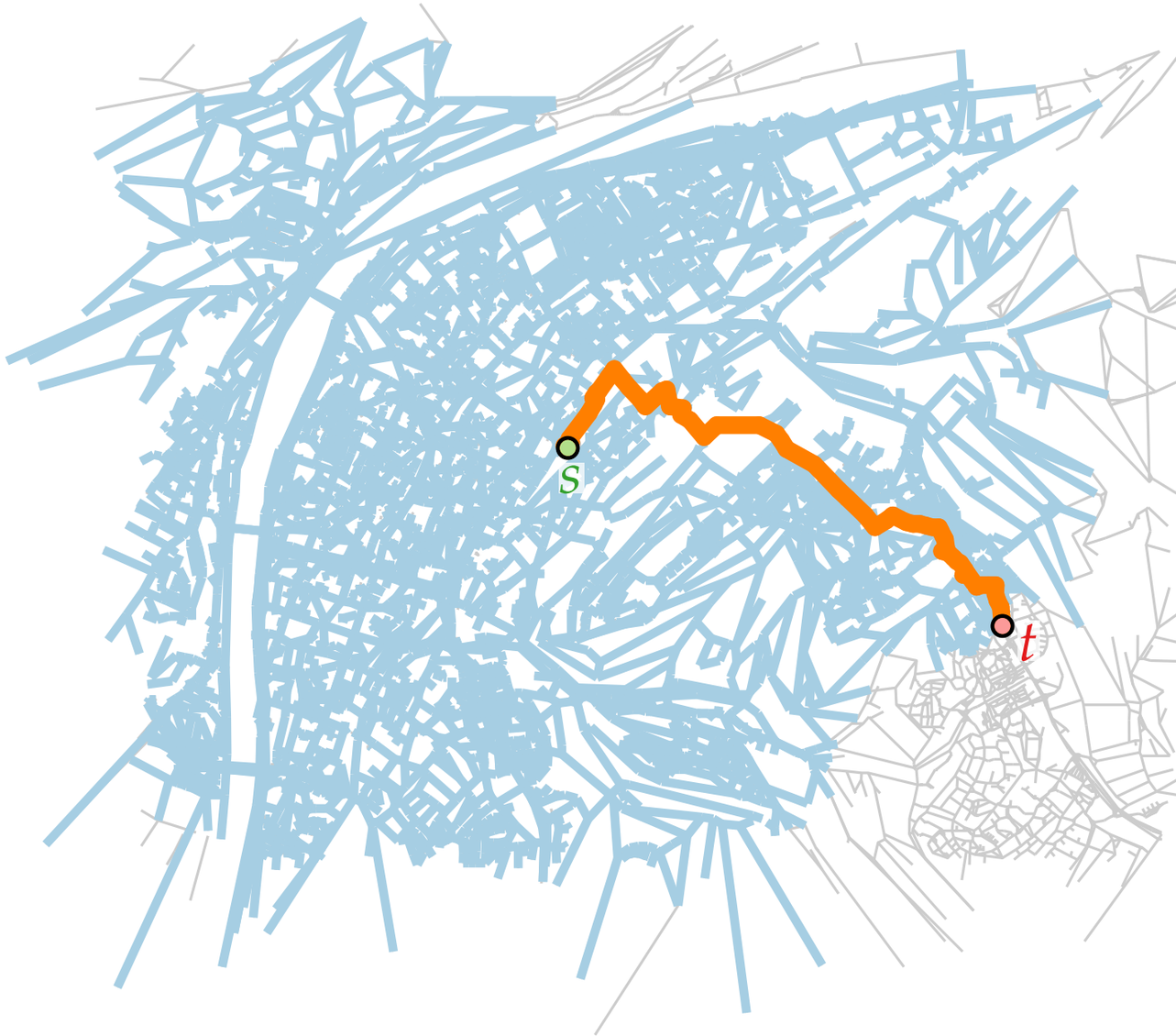


Teil II:
Schätzfunktionen

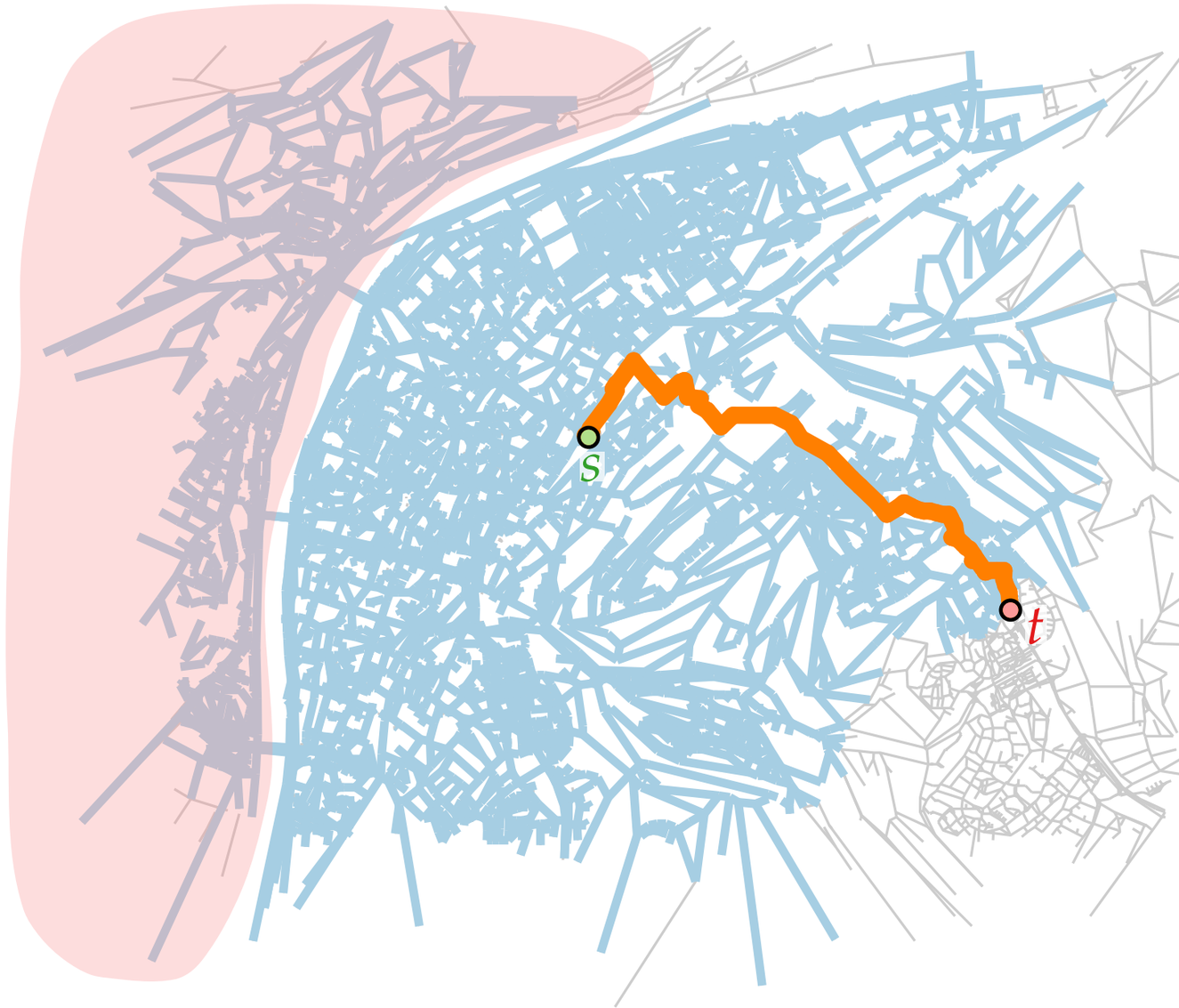
Philipp Kindermann

Wintersemester 2024/25

GreedyBestFirstSearch

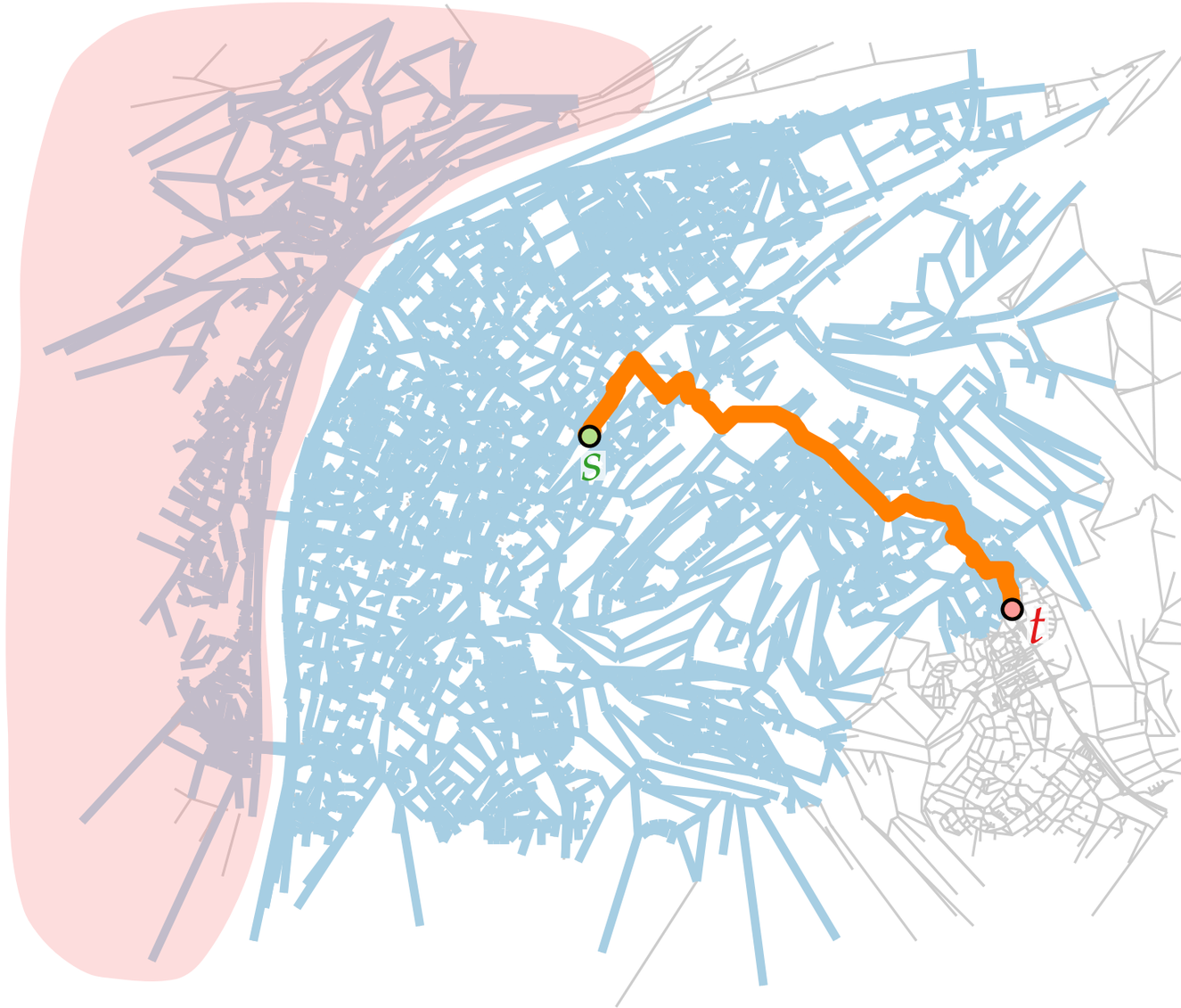


GreedyBestFirstSearch



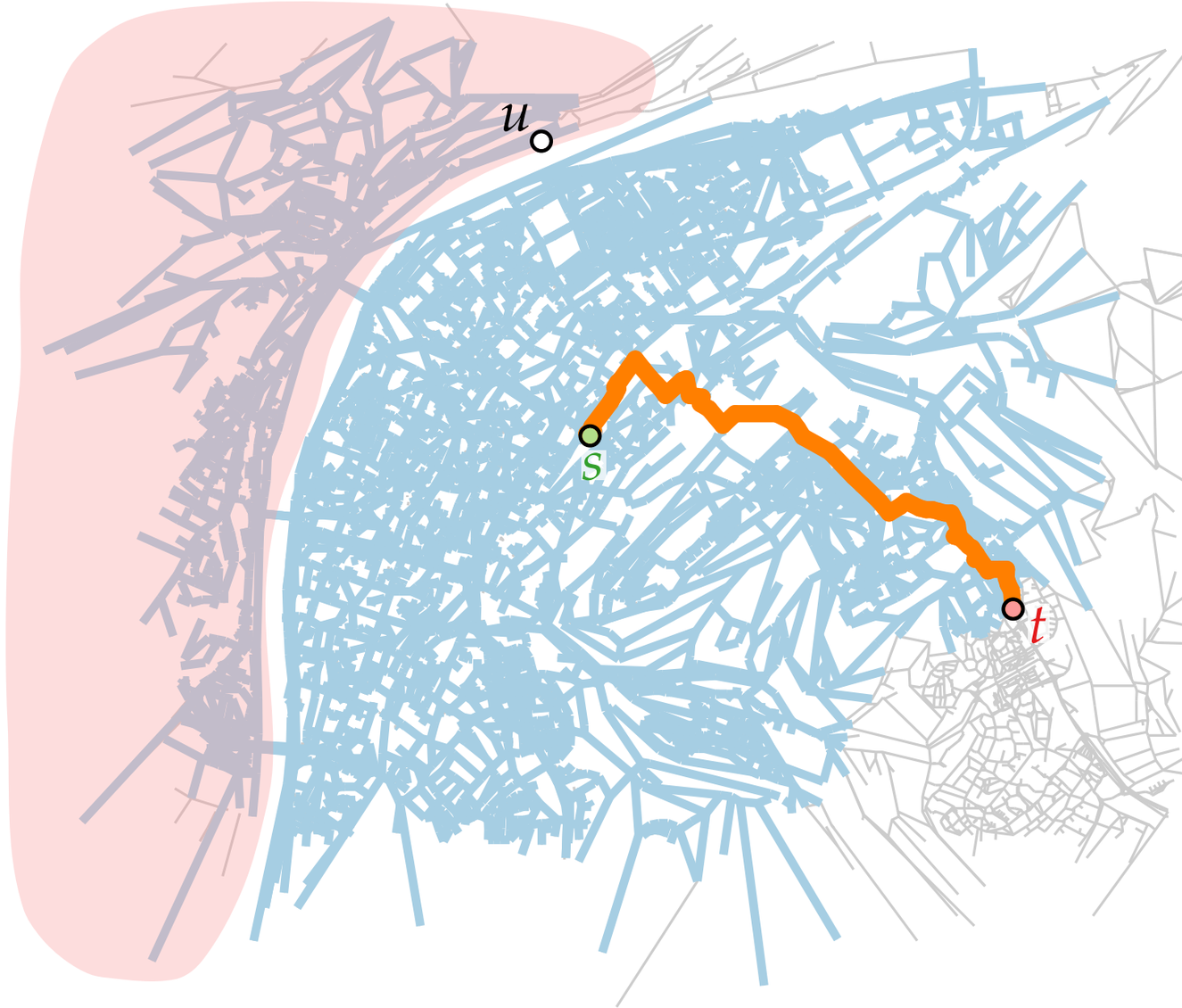
GreedyBestFirstSearch

Ist die Suche hier sinnvoll?



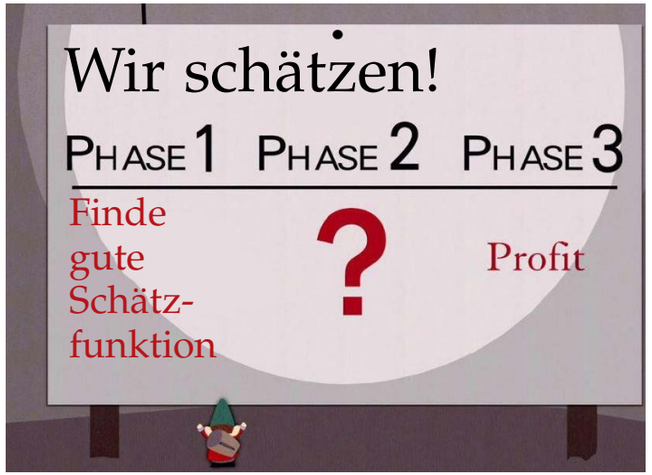
GreedyBestFirstSearch

Ist die Suche hier sinnvoll?

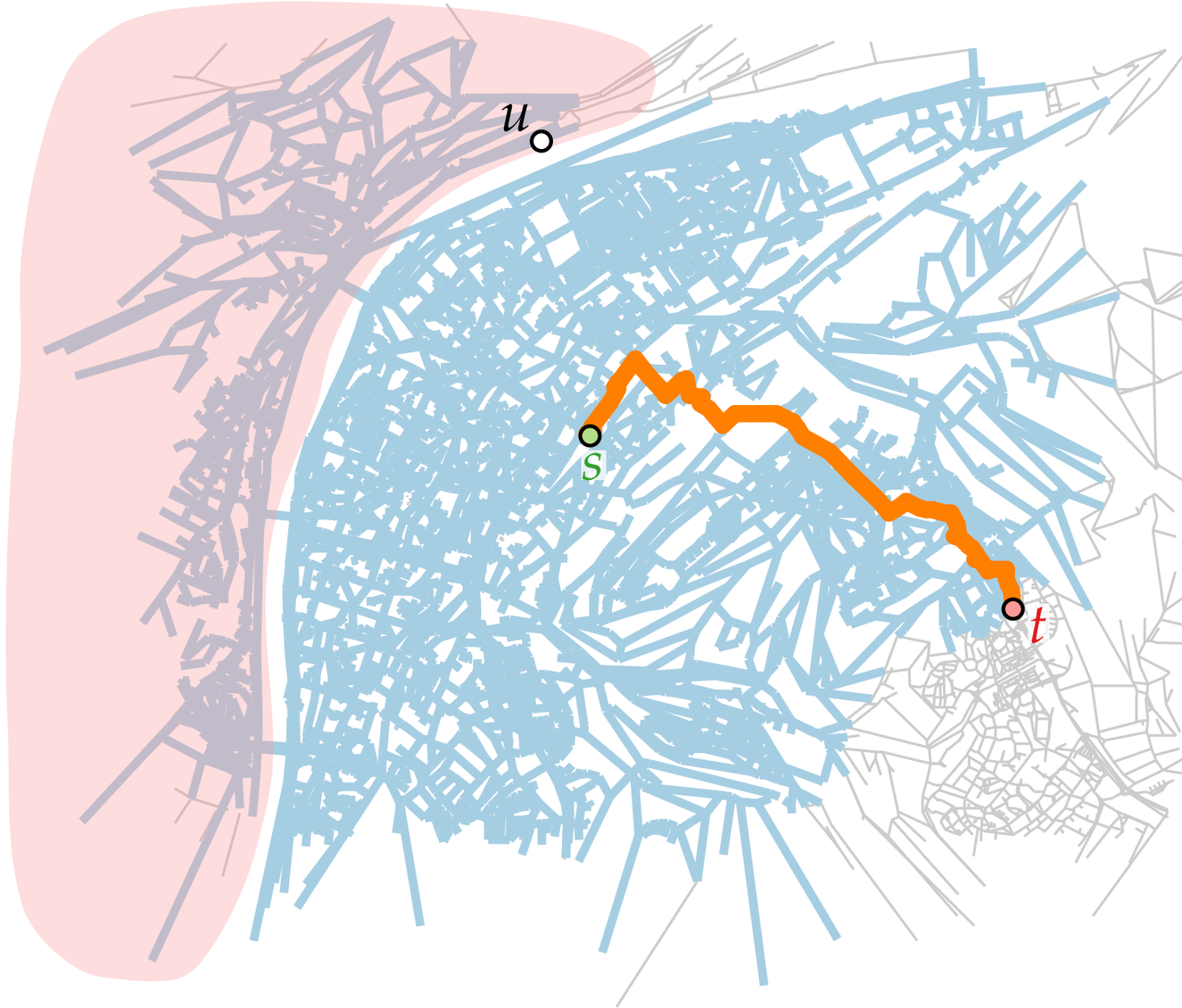


GreedyBestFirstSearch

Idee:

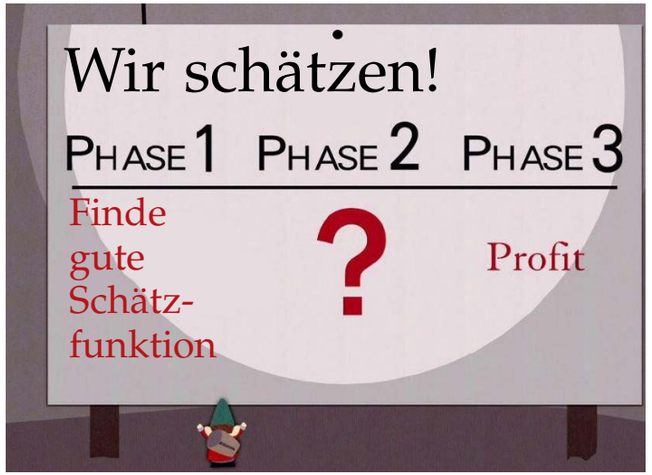


Ist die Suche hier sinnvoll?

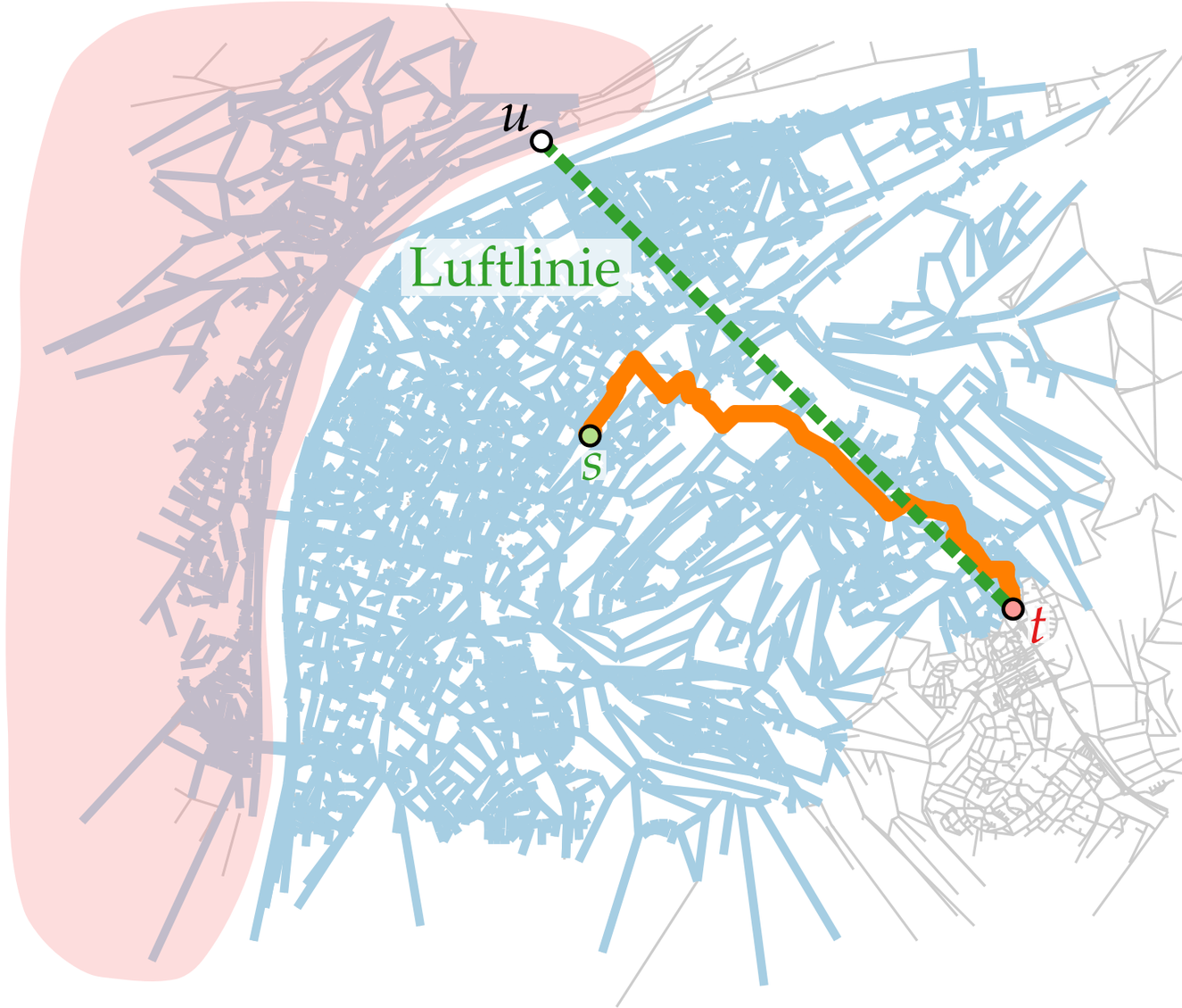


GreedyBestFirstSearch

Idee:

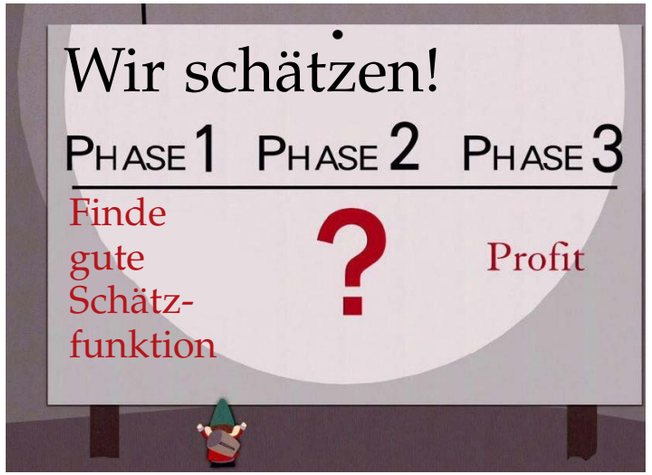


Ist die Suche hier sinnvoll?

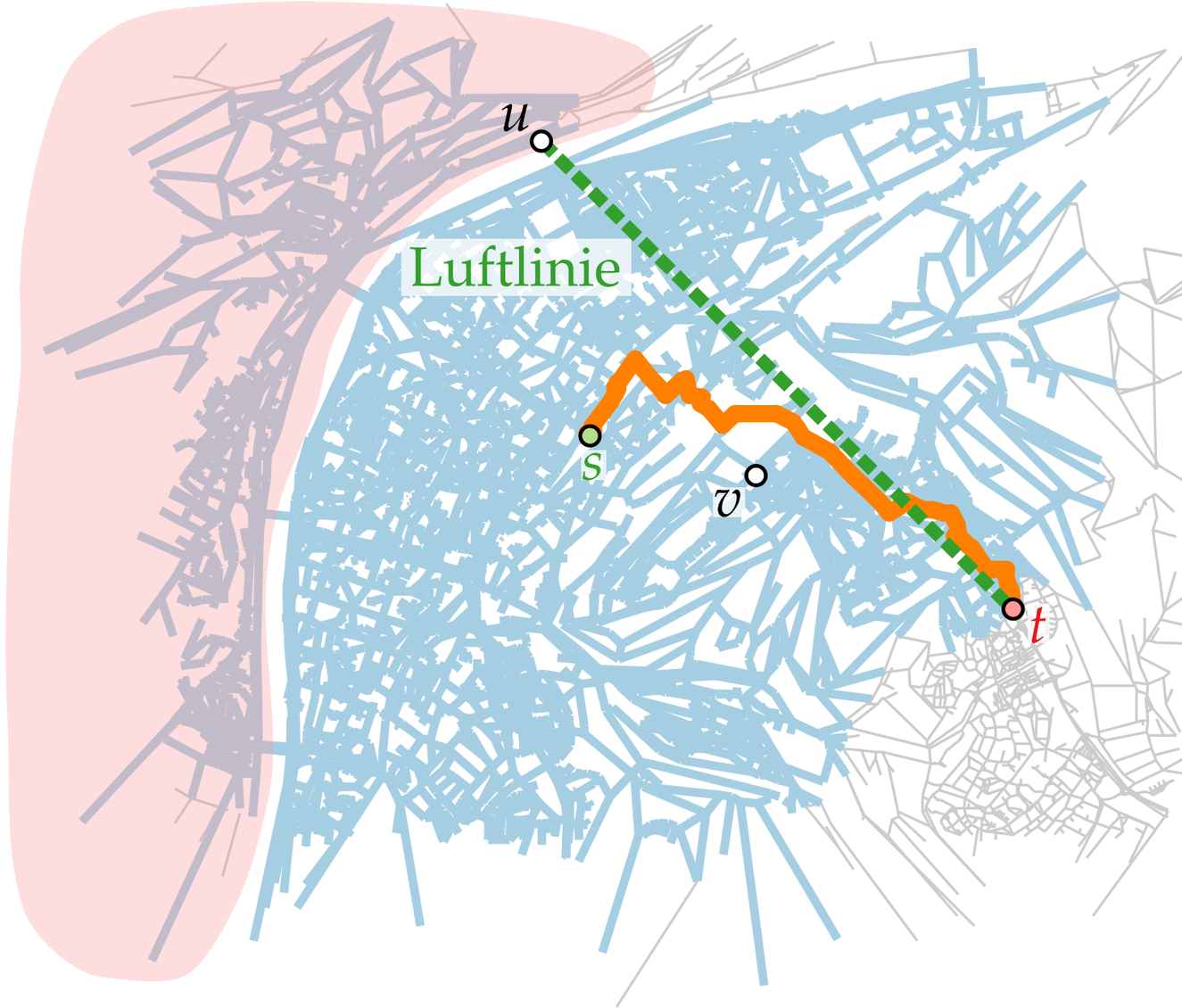


GreedyBestFirstSearch

Idee:

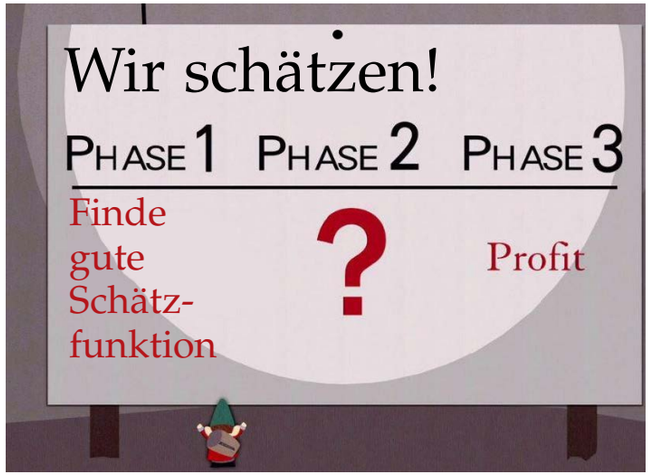


Ist die Suche hier sinnvoll?

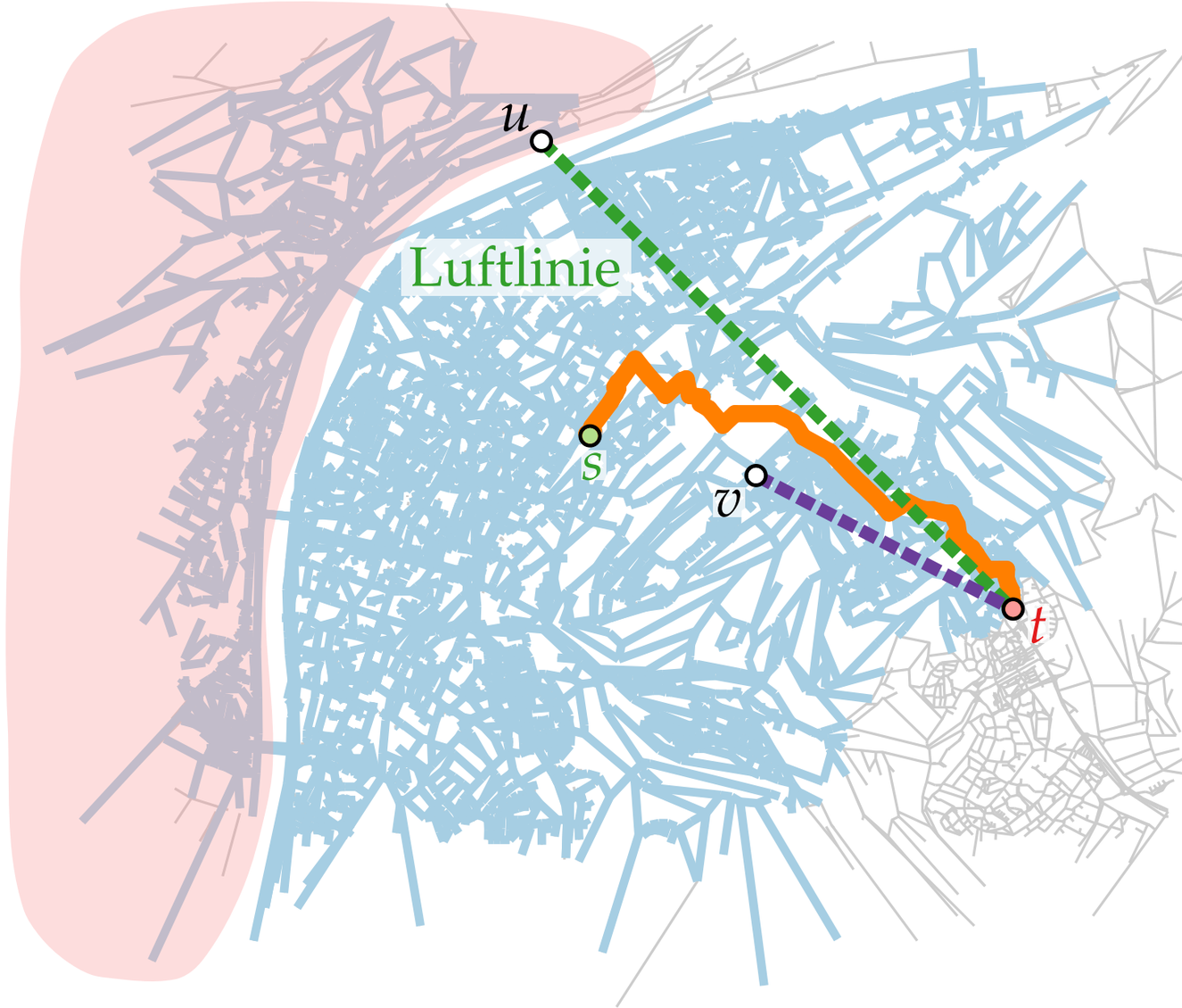


GreedyBestFirstSearch

Idee:

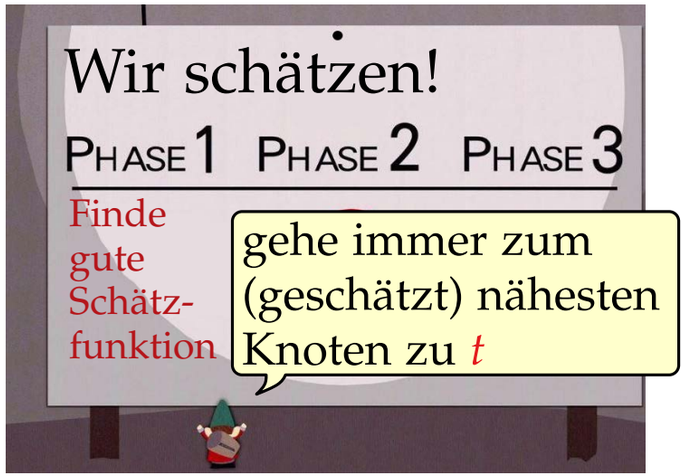


Ist die Suche hier sinnvoll?

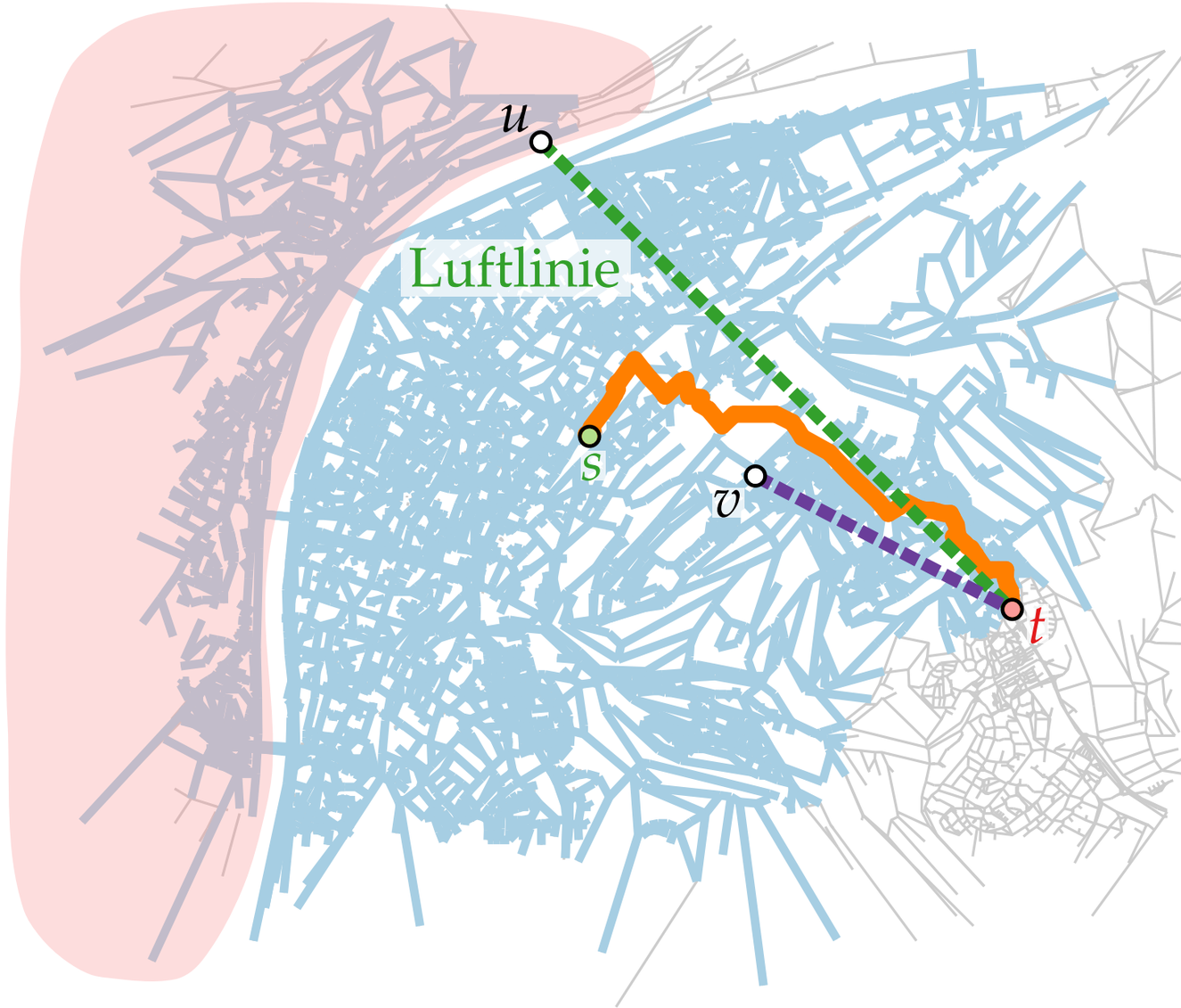


GreedyBestFirstSearch

Idee:

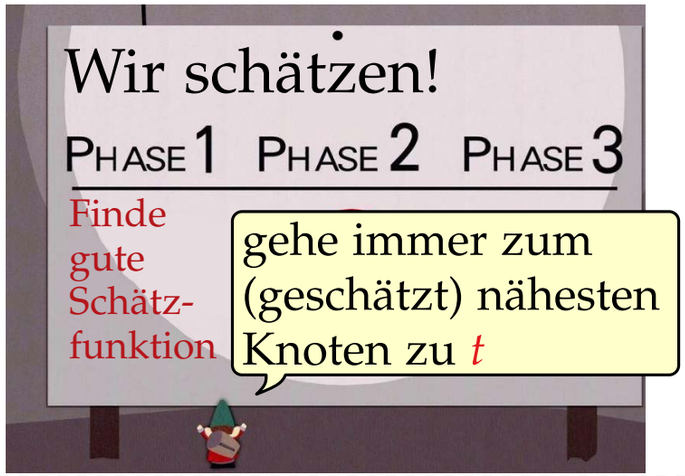


Ist die Suche hier sinnvoll?



GreedyBestFirstSearch

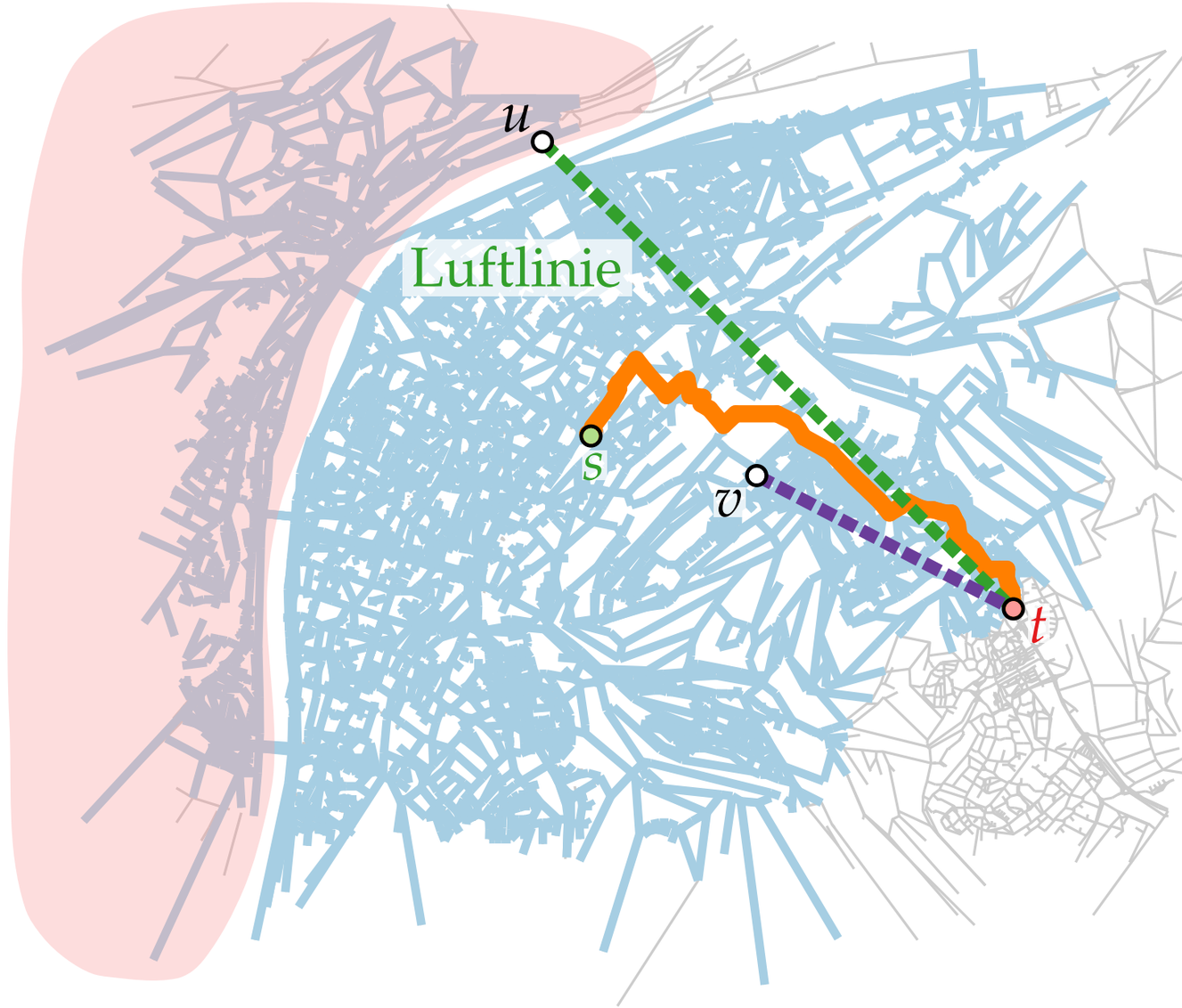
Idee:



Dijkstra(WeightedGraph G, Vertices s, t)

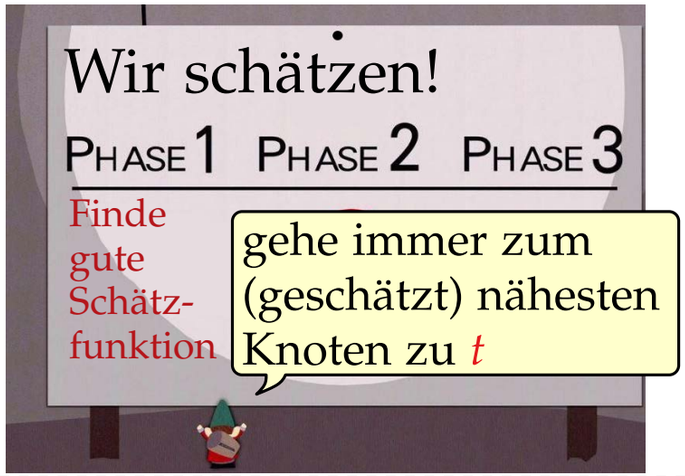
```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d > u.d + w(u,v) then
            v.d = u.d + w(u,v)
            v.π = u
            Q.DecreaseKey(v, v.d)
```

Ist die Suche hier sinnvoll?



GreedyBestFirstSearch

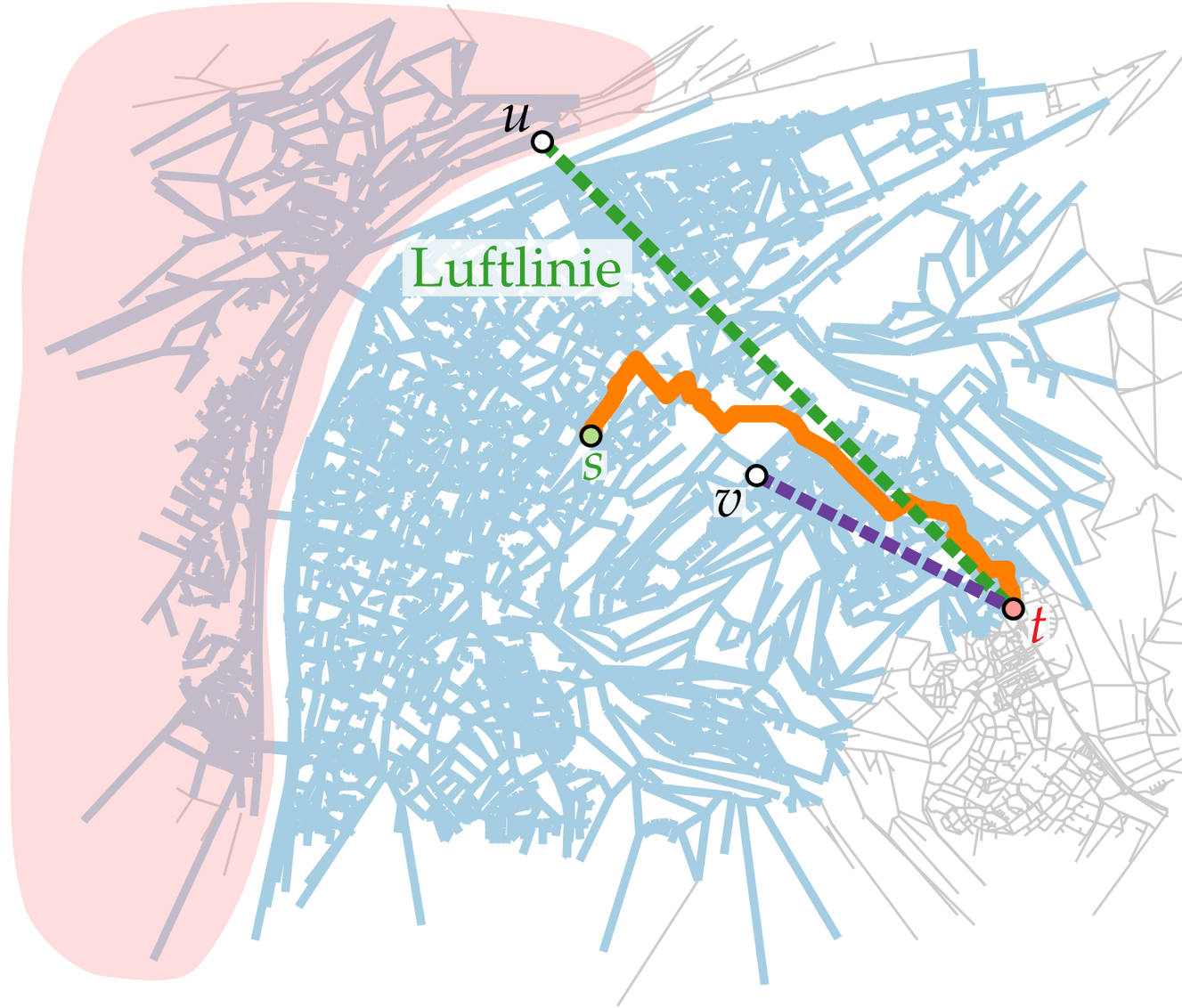
Idee:



GBFS(WeigtedGraph G, Vertices s, t)

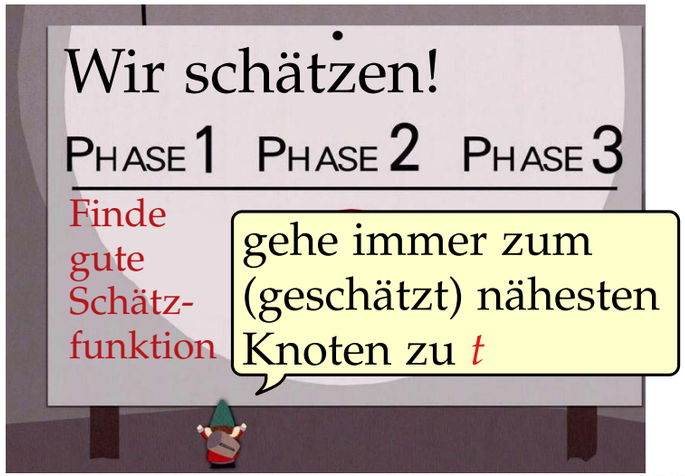
```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d > u.d + w(u, v) then
            v.d = u.d + w(u, v)
            v.π = u
            Q.DecreaseKey(v, v.d)
```

Ist die Suche hier sinnvoll?



GreedyBestFirstSearch

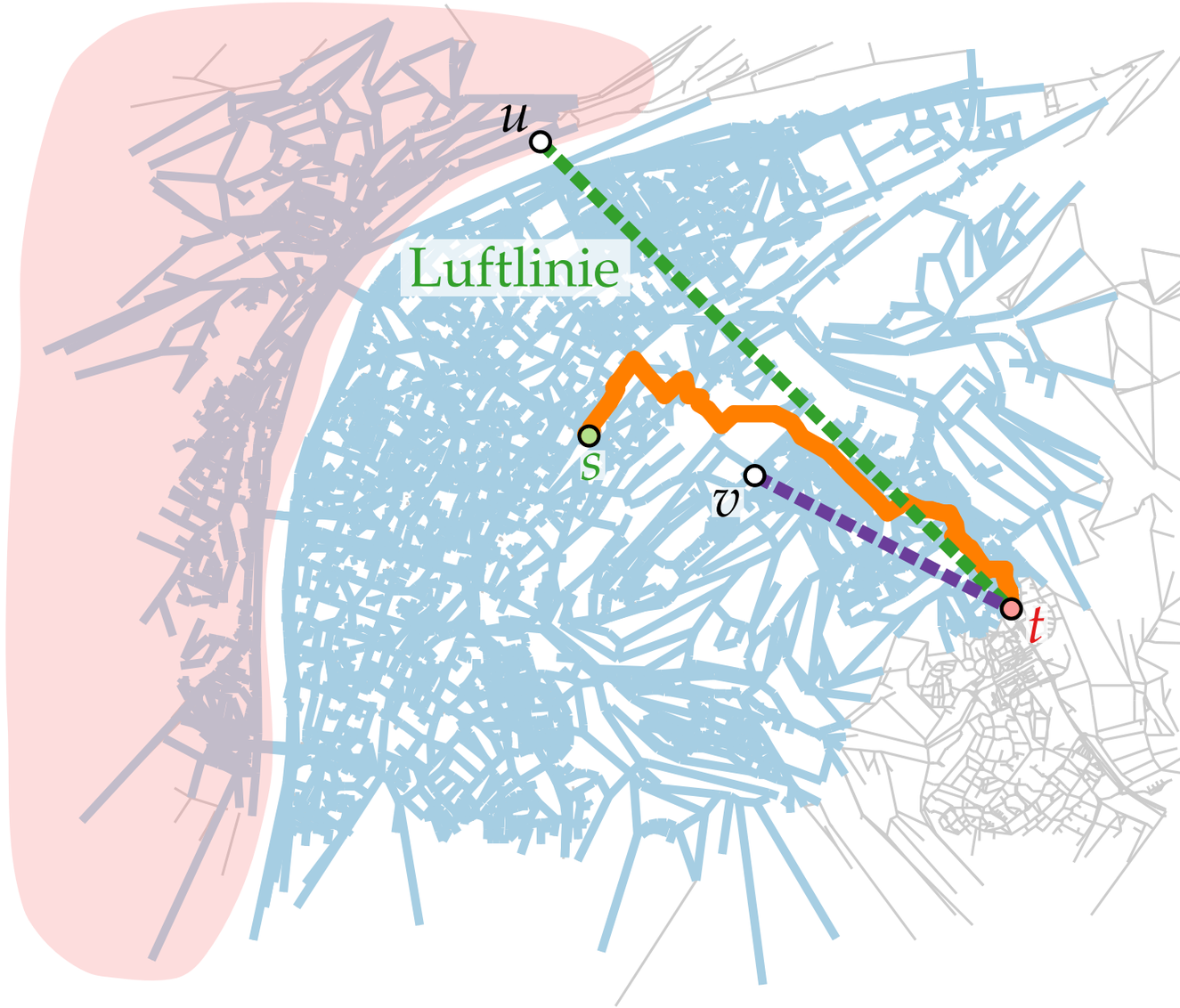
Idee:



GBFS(WeigtedGraph G, Vertices s, t)

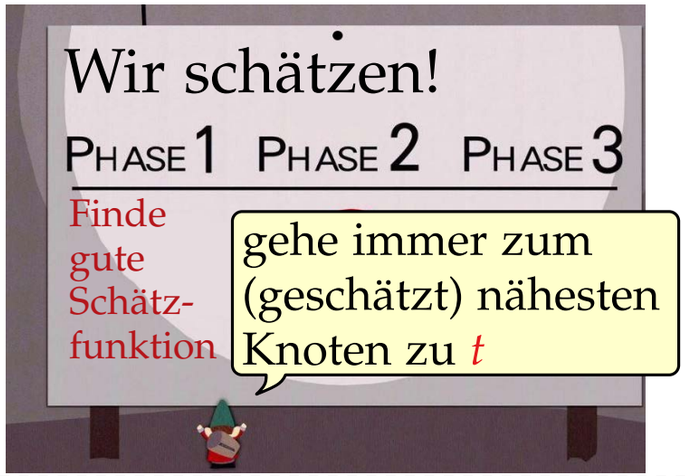
```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d == ∞ then
            v.d = u.d + w(u, v)
            v.π = u
            Q.DecreaseKey(v, v.d)
```

Ist die Suche hier sinnvoll?



GreedyBestFirstSearch

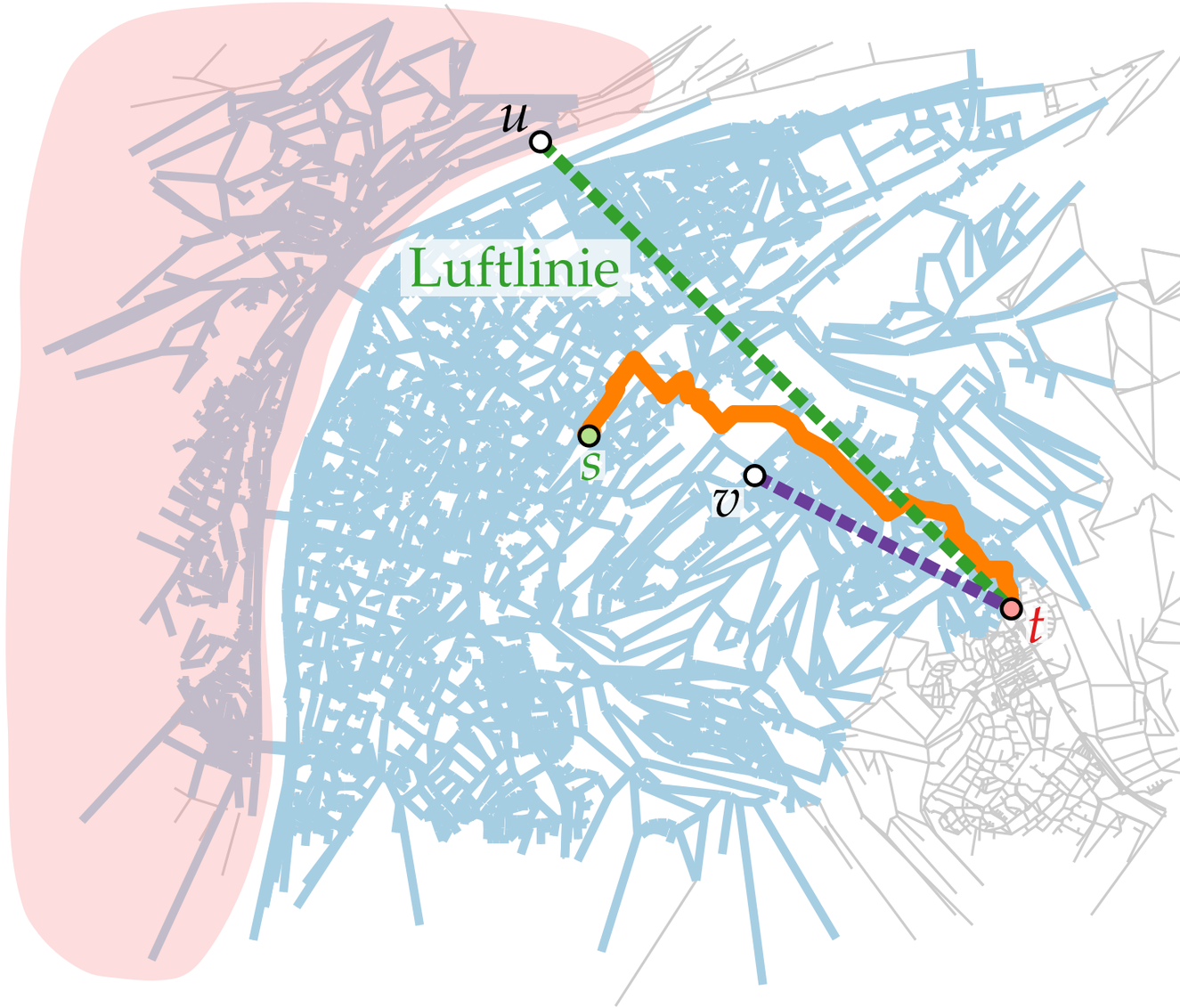
Idee:



GBFS(WeigtedGraph G, Vertices s, t)

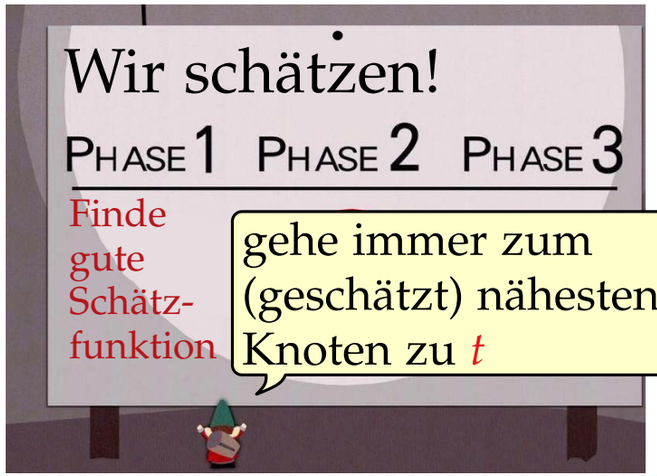
```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d == ∞ then
            v.d = δ*(v, t)
            v.π = u
            Q.DecreaseKey(v, v.d)
```

Ist die Suche hier sinnvoll?



GreedyBestFirstSearch

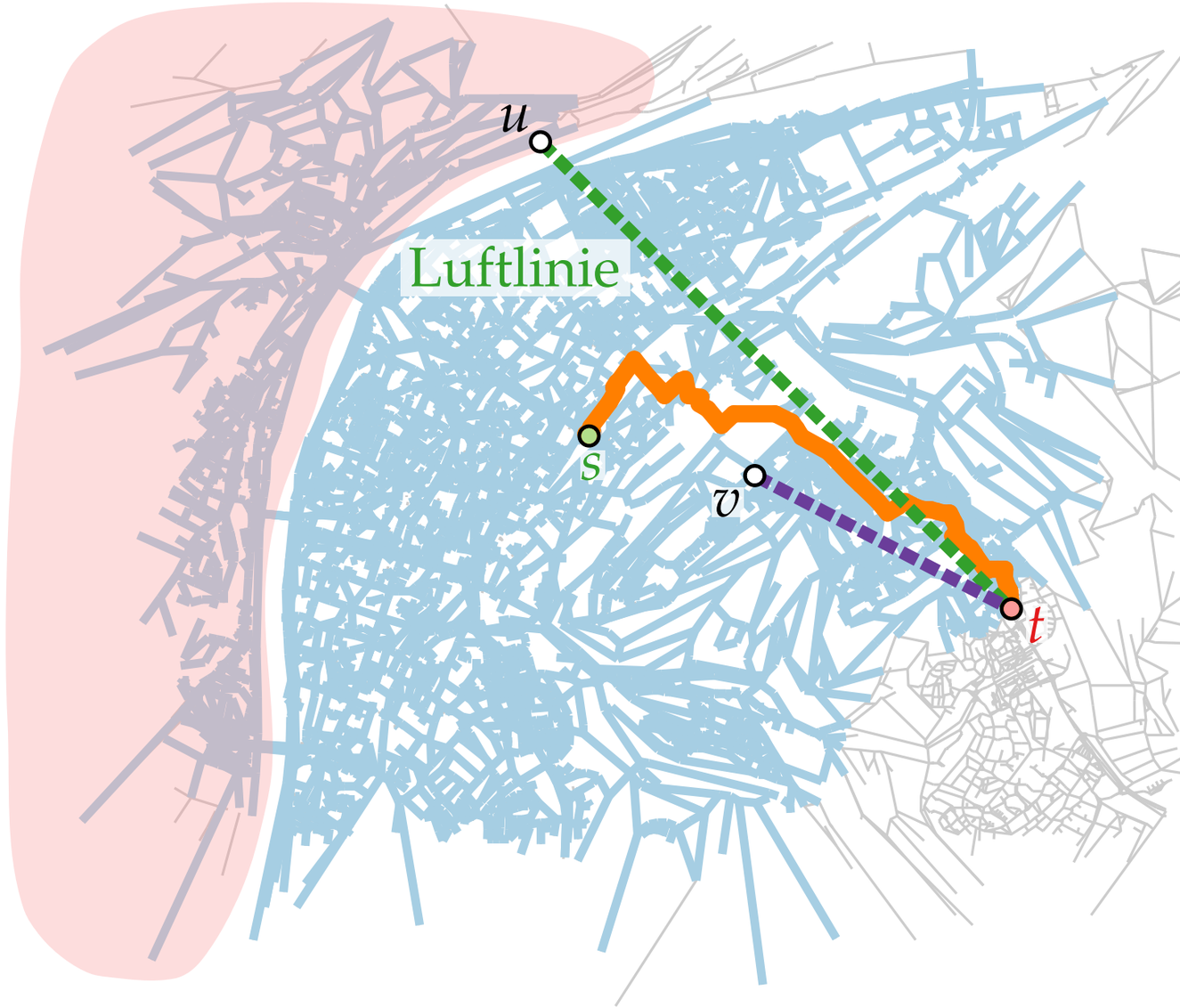
Idee:



GBFS(WeigtedGraph G, Vertices s, t)

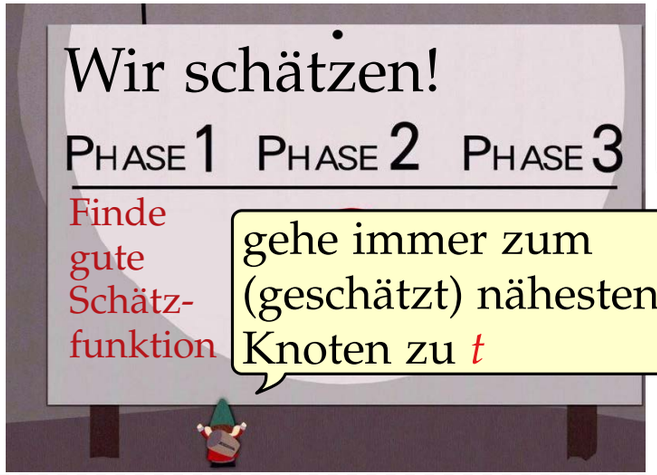
```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d == ∞ then
            v.d = δ*(v, t)        (geschätzte Distanz)
            v.π = u
            Q.DecreaseKey(v, v.d)
```

Ist die Suche hier sinnvoll?



GreedyBestFirstSearch

Idee:

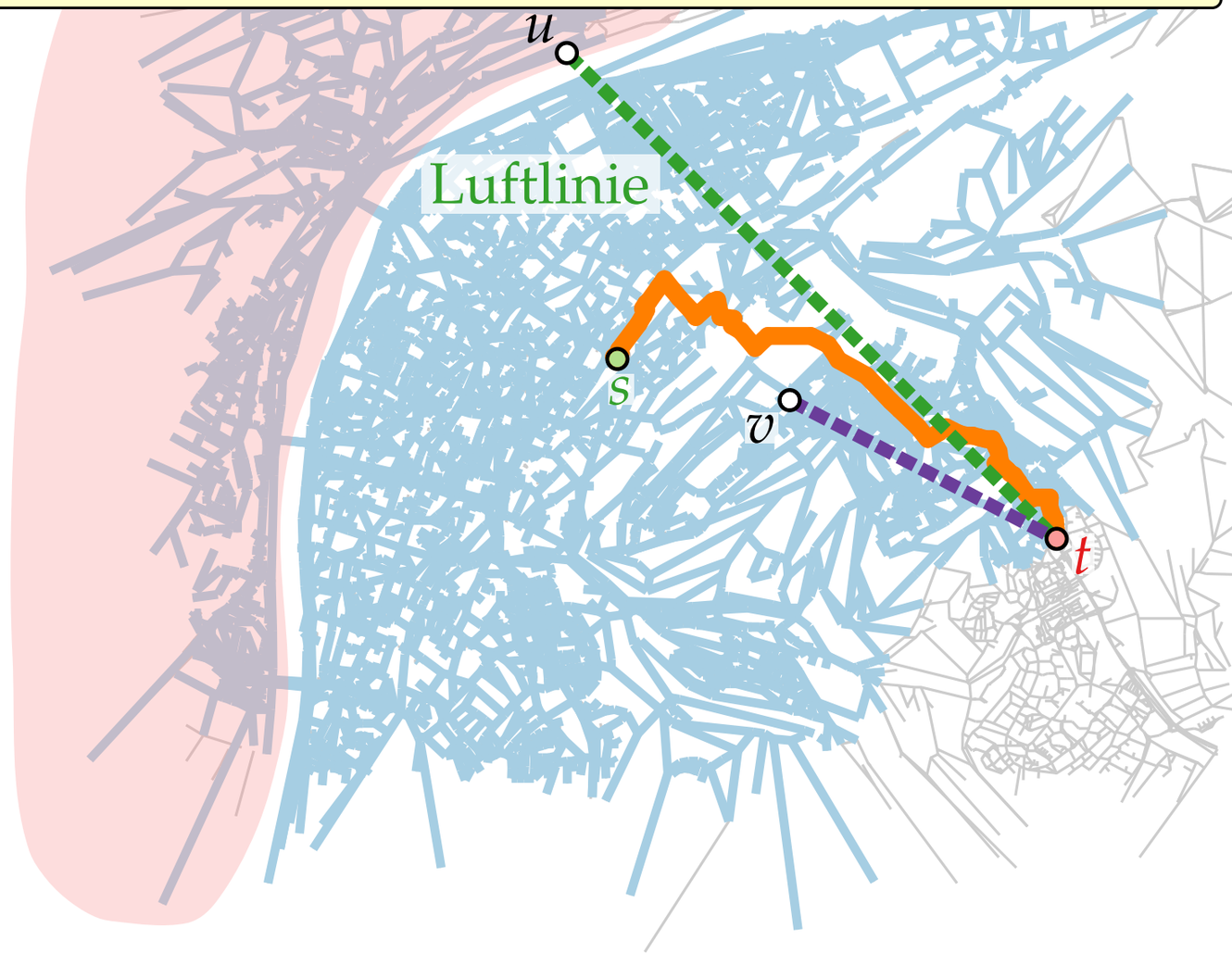


Demo.

`https://algo.uni-trier.de/demos/shortestpath.html`

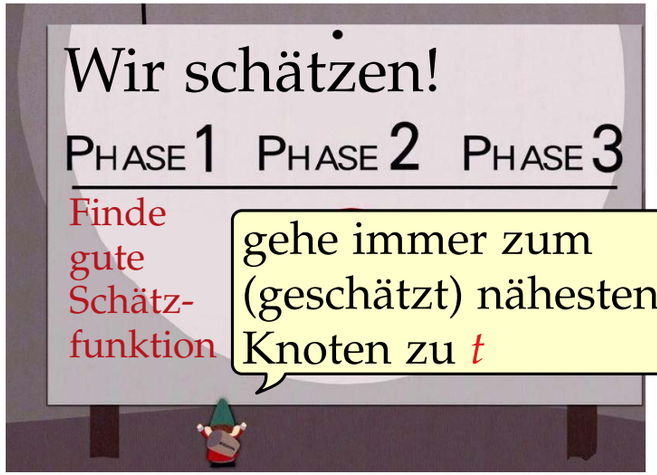
GBFS(WeigtedGraph G, Vertices s, t)

```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d == ∞ then
            v.d = δ*(v, t)        // geschätzte Distanz
            v.π = u
            Q.DecreaseKey(v, v.d)
```



GreedyBestFirstSearch

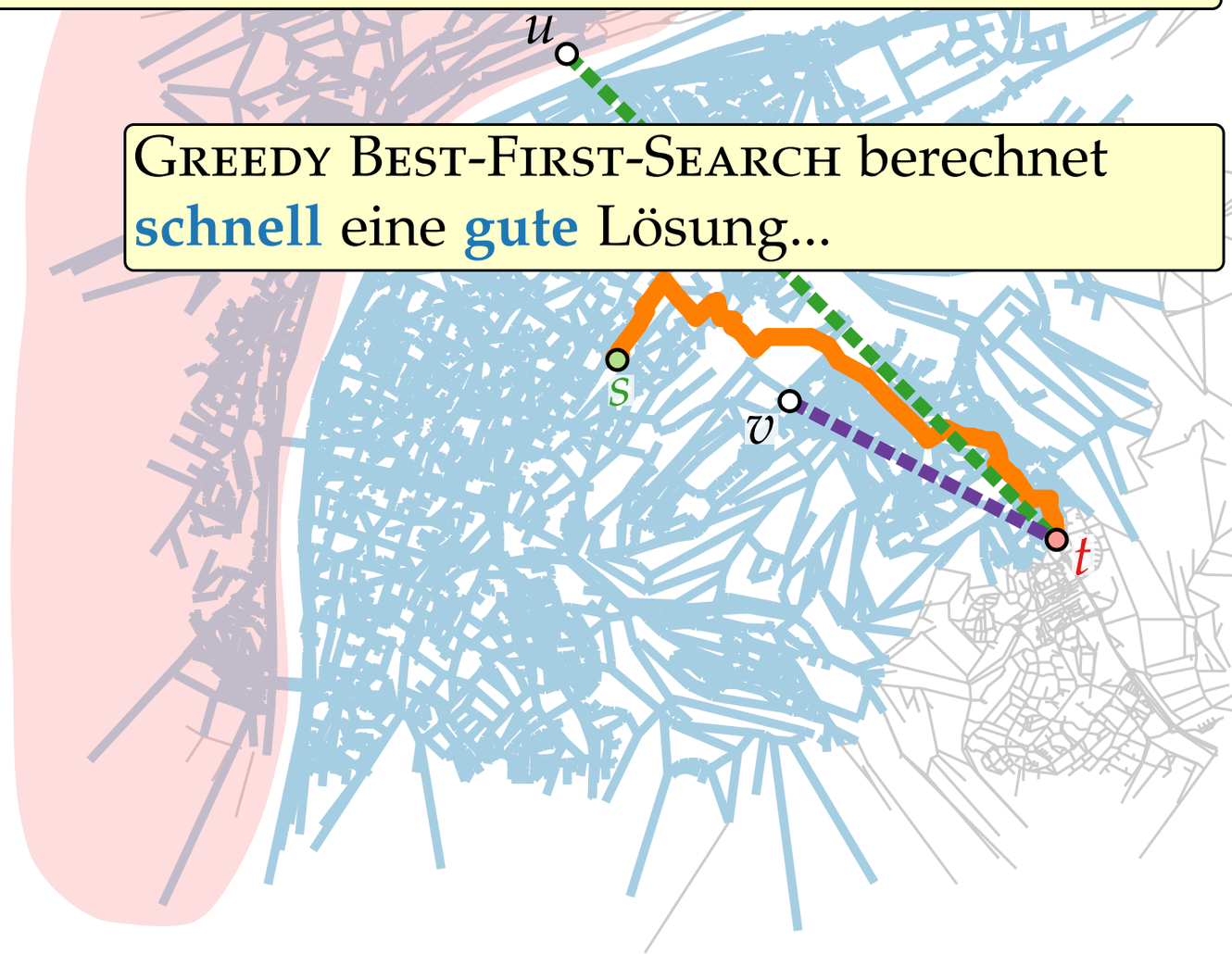
Idee:



Demo.
`https://algo.uni-trier.de/demos/shortestpath.html`

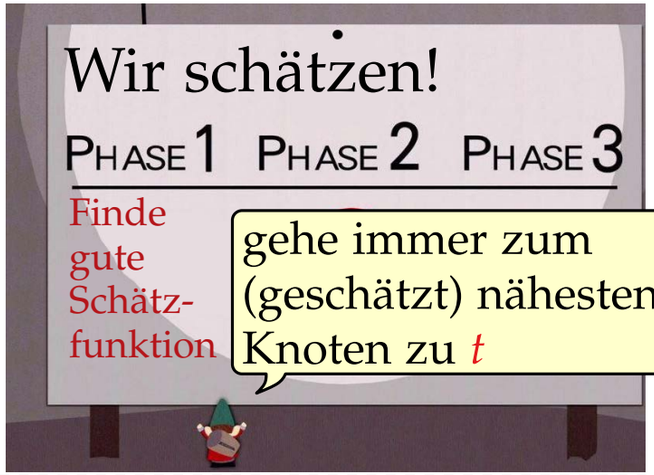
GBFS(WeigtedGraph G, Vertices s, t)

```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d == ∞ then
            v.d = δ*(v, t)        // geschätzte Distanz
            v.π = u
            Q.DecreaseKey(v, v.d)
```

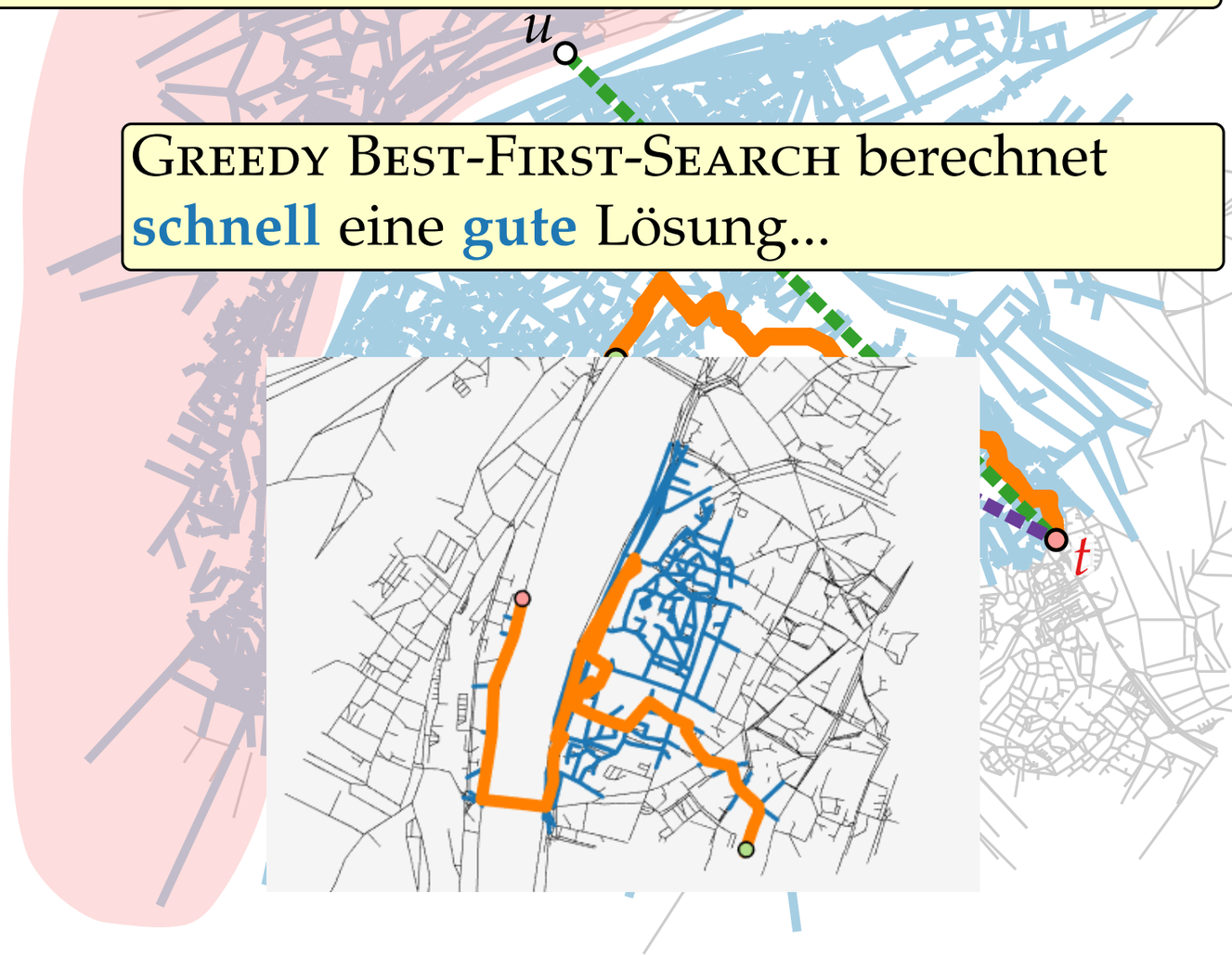


GreedyBestFirstSearch

Idee:



Demo.
`https://algo.uni-trier.de/demos/shortestpath.html`



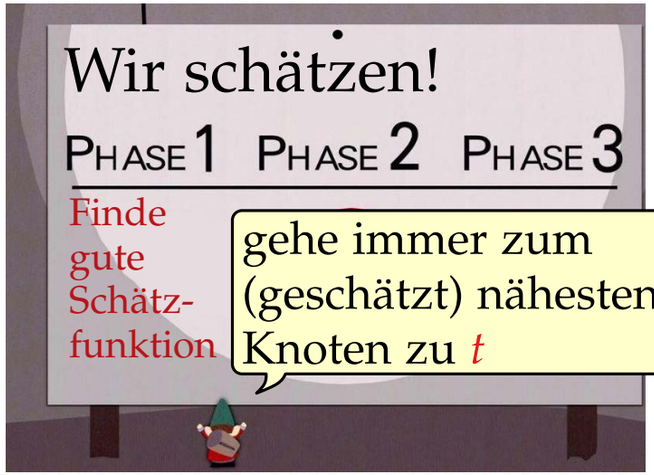
GBFS(WeigtedGraph G, Vertices s, t)

```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d == ∞ then
            v.d = δ*(v, t)        // geschätzte Distanz
            v.π = u
            Q.DecreaseKey(v, v.d)
```

Greedy Best-First-Search berechnet **schnell** eine **gute** Lösung...

GreedyBestFirstSearch

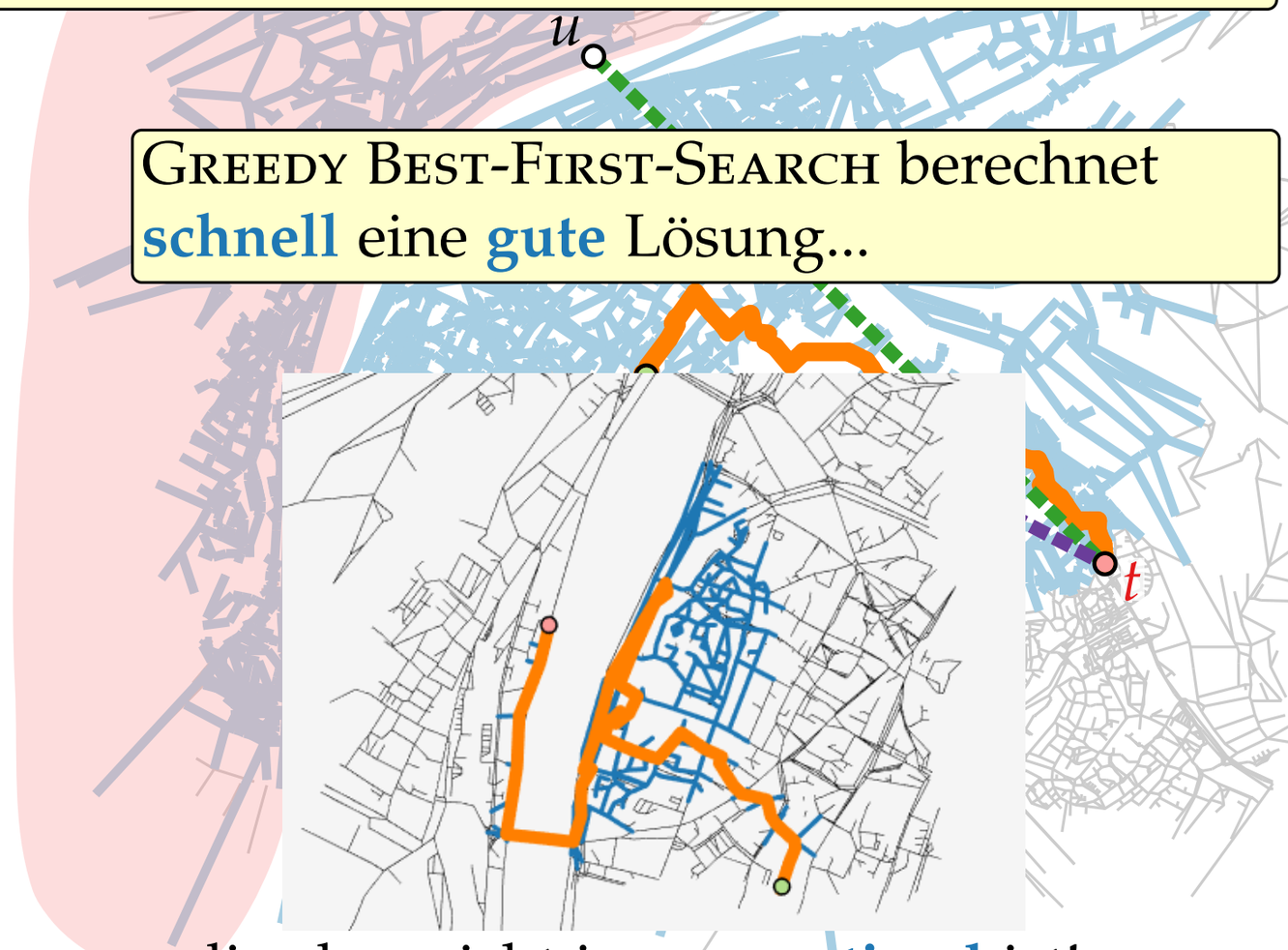
Idee:



Demo.
`https://algo.uni-trier.de/demos/shortestpath.html`

GBFS(WeigtedGraph G, Vertices s, t)

```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d == ∞ then
            v.d = δ*(v, t)        // geschätzte Distanz
            v.π = u
            Q.DecreaseKey(v, v.d)
```

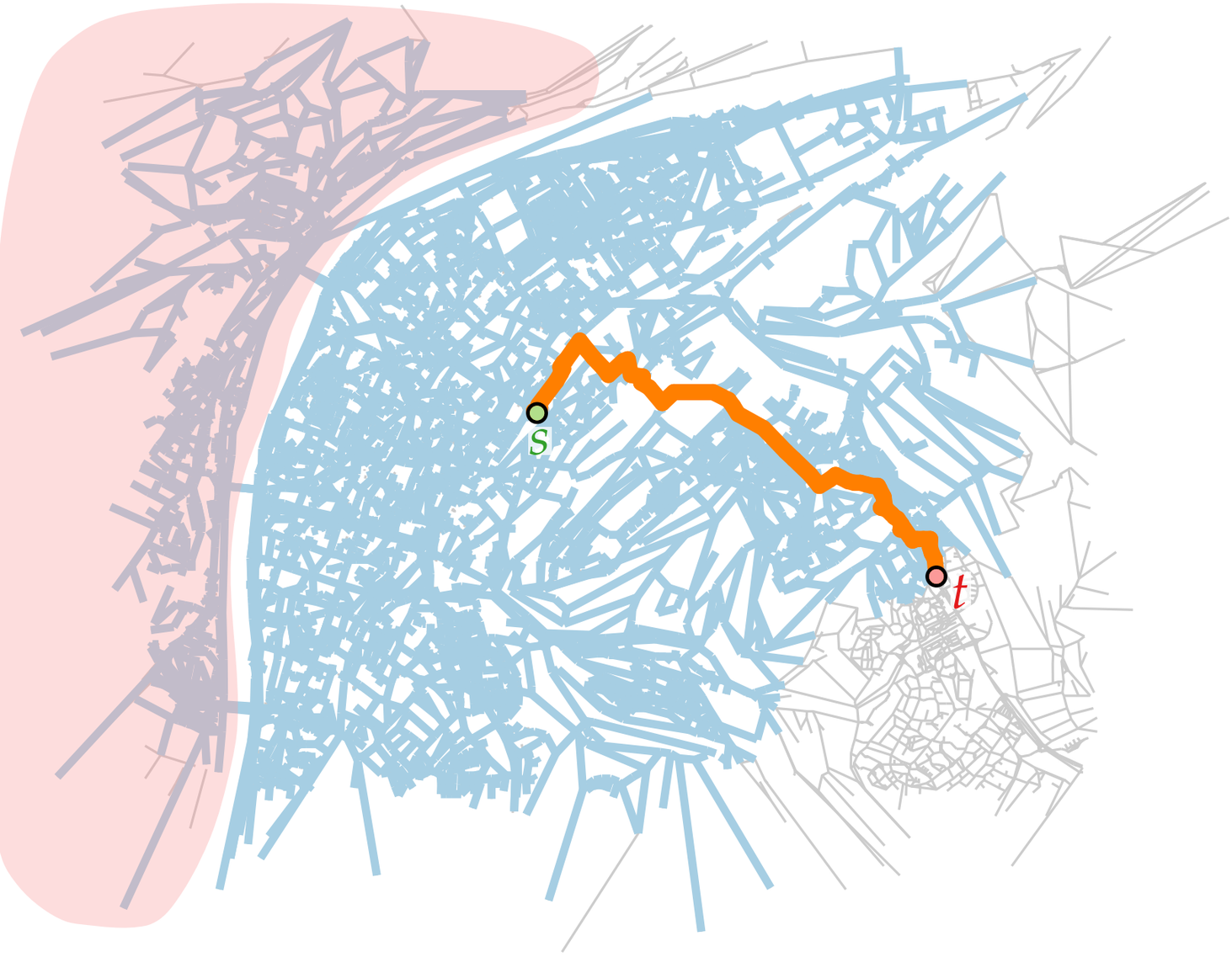


Greedy Best-First-Search berechnet **schnell** eine **gute** Lösung...

...die aber nicht immer **optimal** ist!

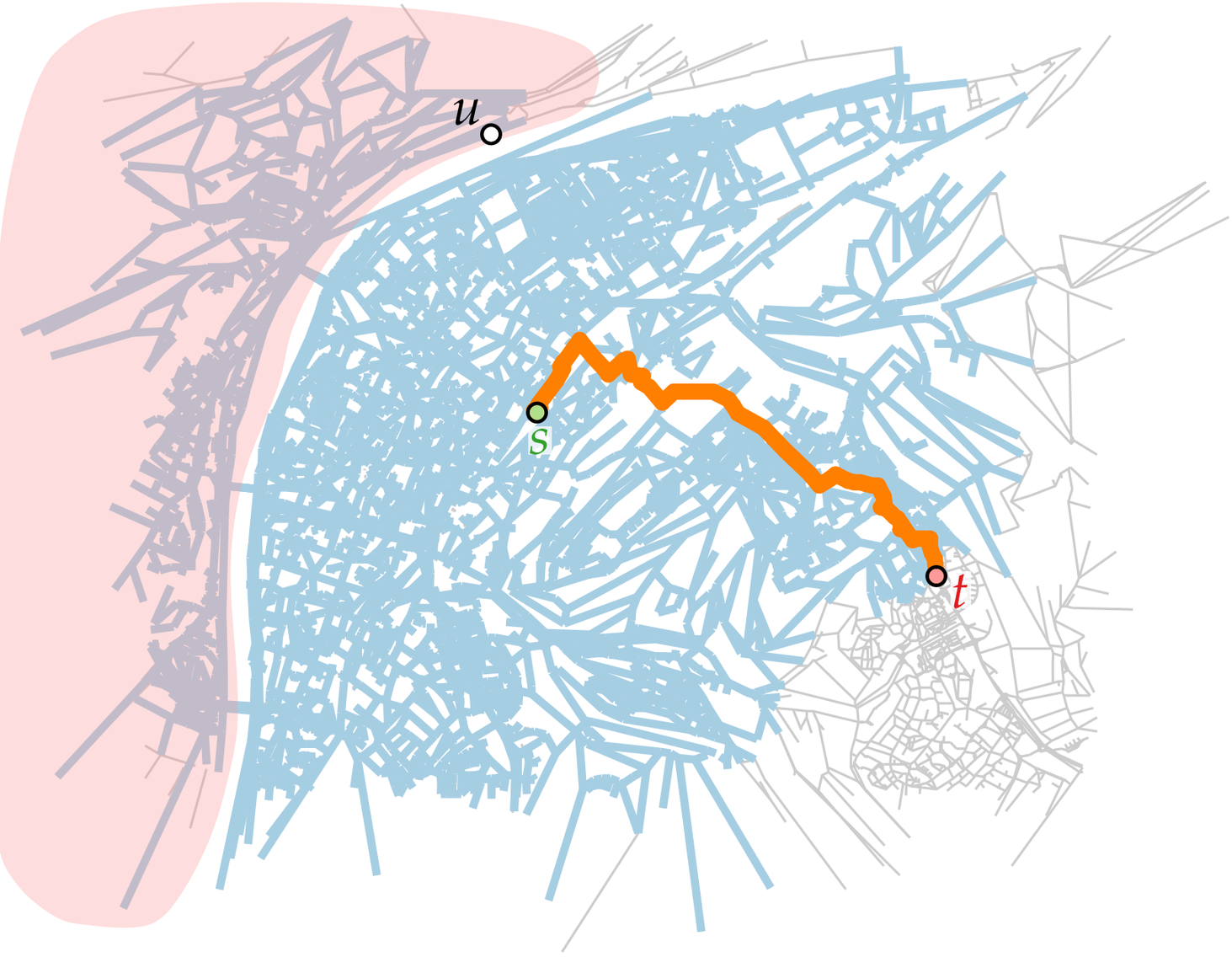
Schätzfunktionen

Ist die Suche hier sinnvoll?



Schätzfunktionen

Ist die Suche hier sinnvoll?

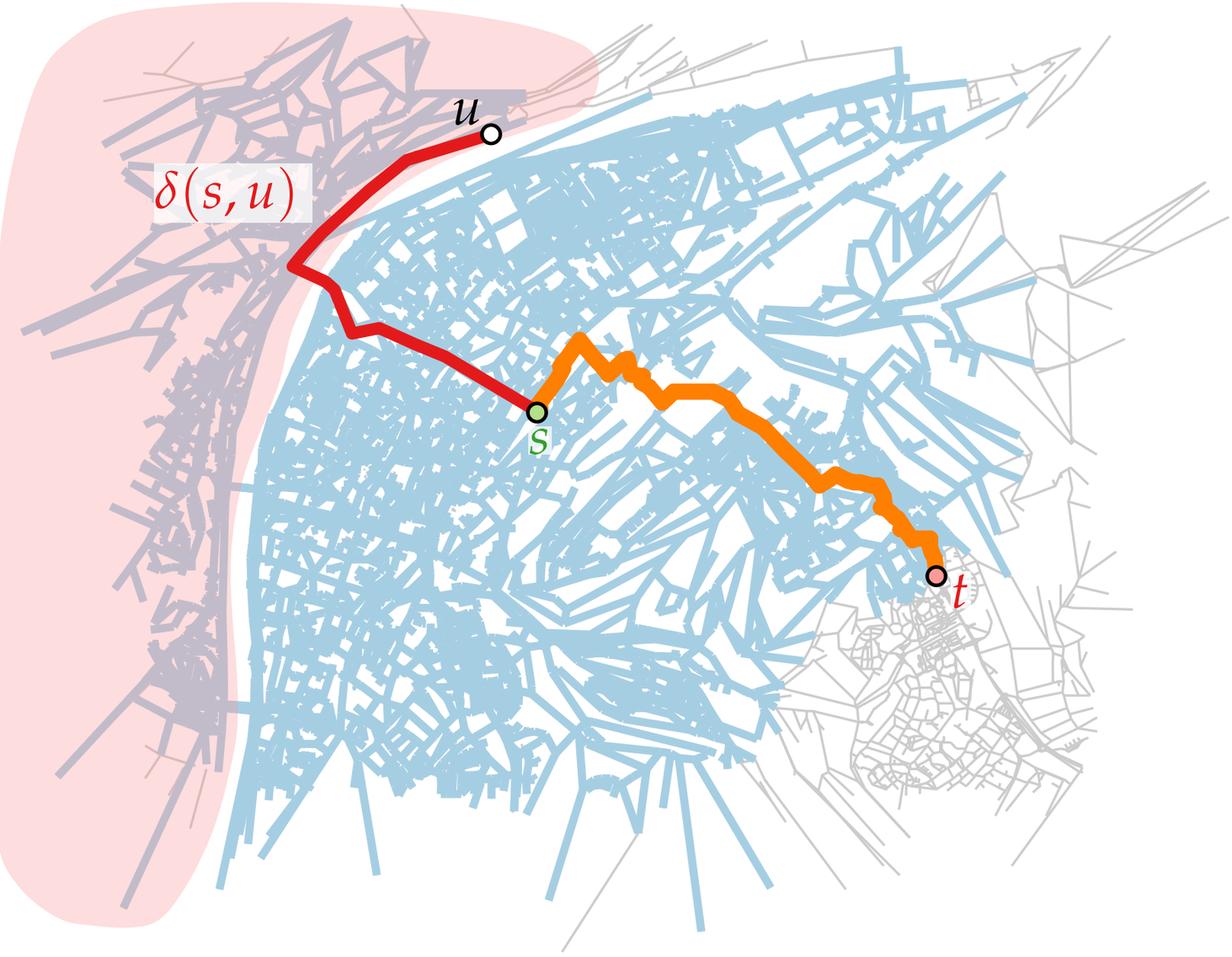


Schätzfunktionen

Ist die Suche hier sinnvoll?

Schätzfunktionen

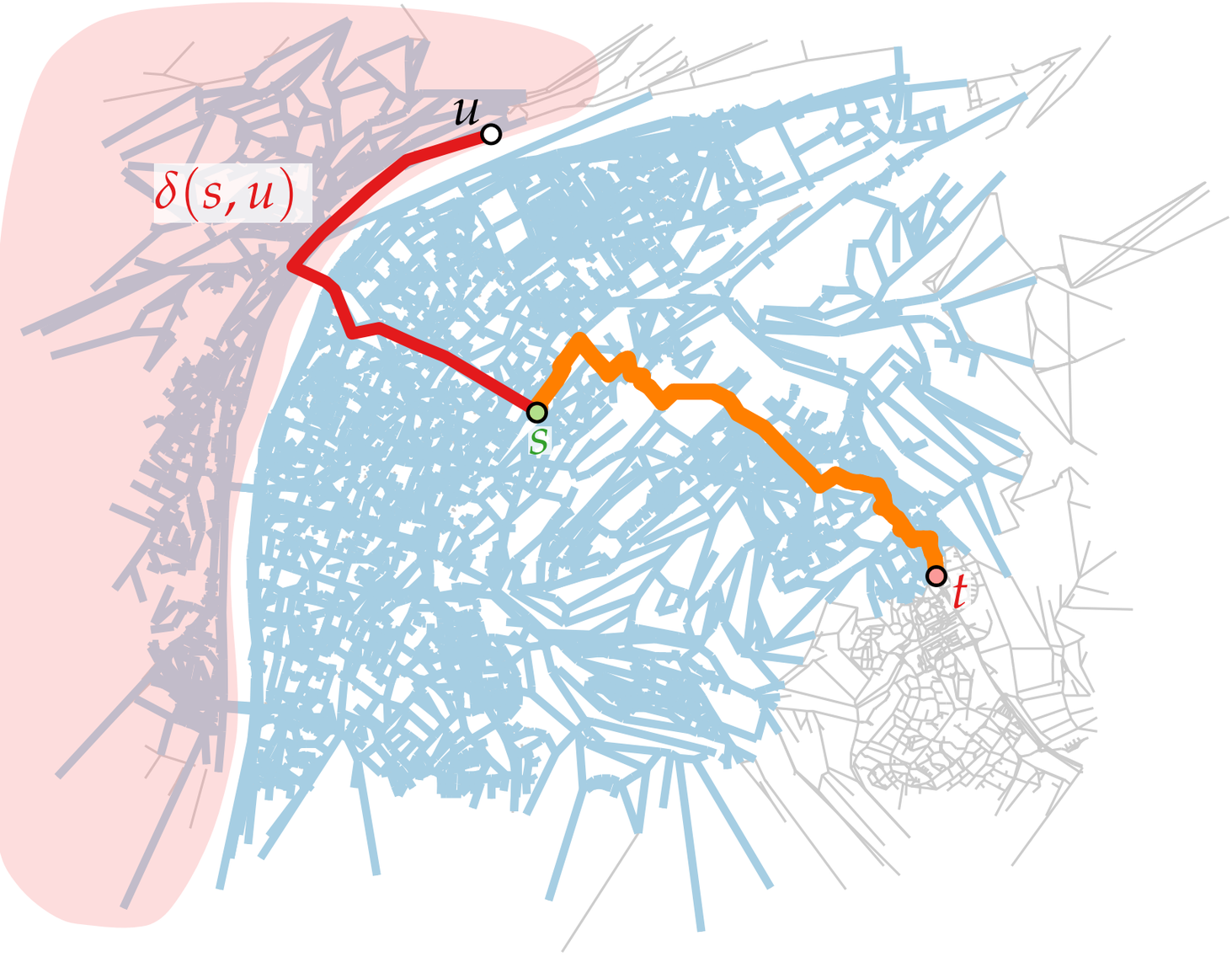
Ist die Suche hier sinnvoll?



Schätzfunktionen

$\delta(s,u) + \delta(u,t)$

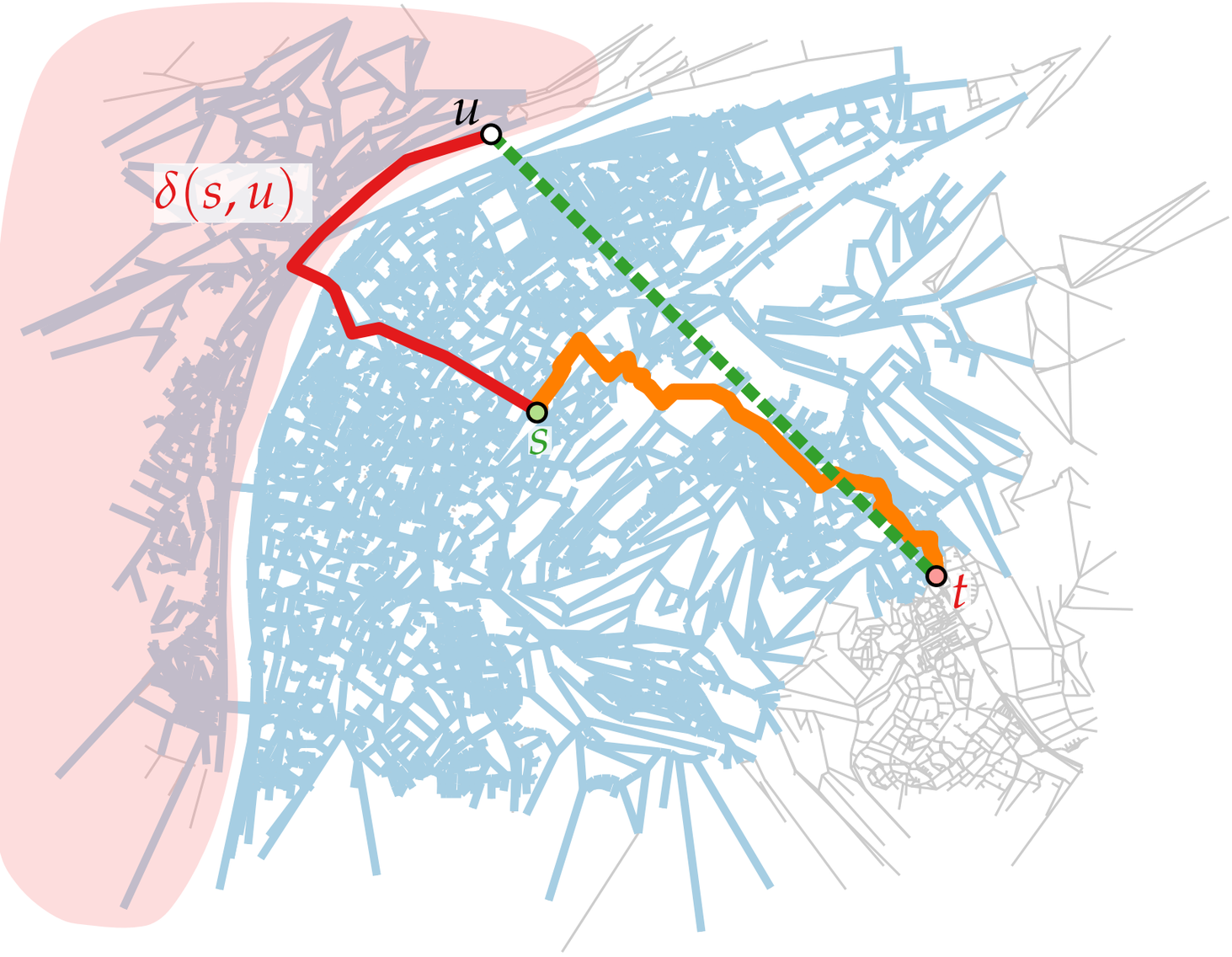
Ist die Suche hier sinnvoll?



Schätzfunktionen

$\delta(s,u) + \delta(u,t)$

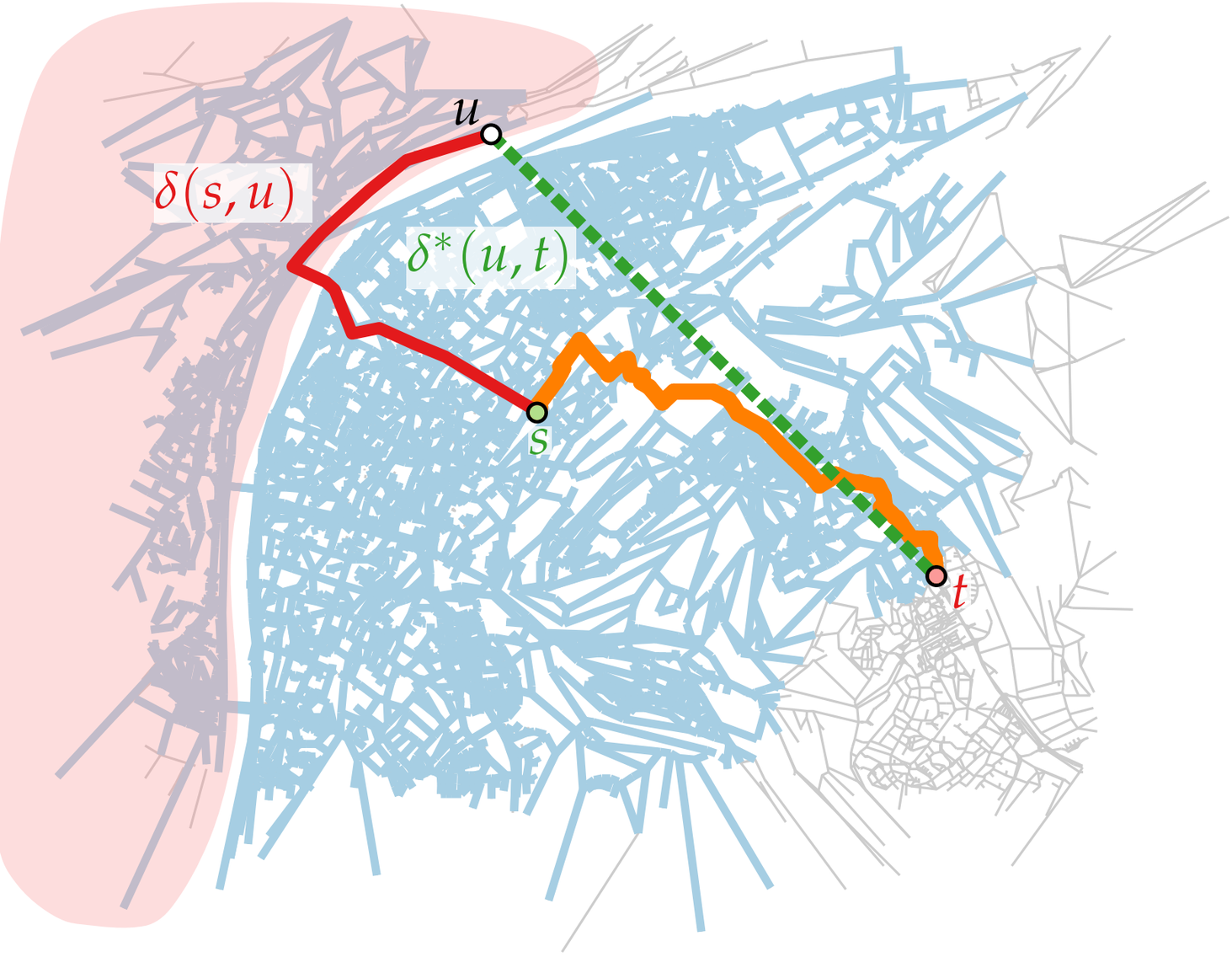
Ist die Suche hier sinnvoll?



Schätzfunktionen

$\delta(s,u) + \delta(u,t)$

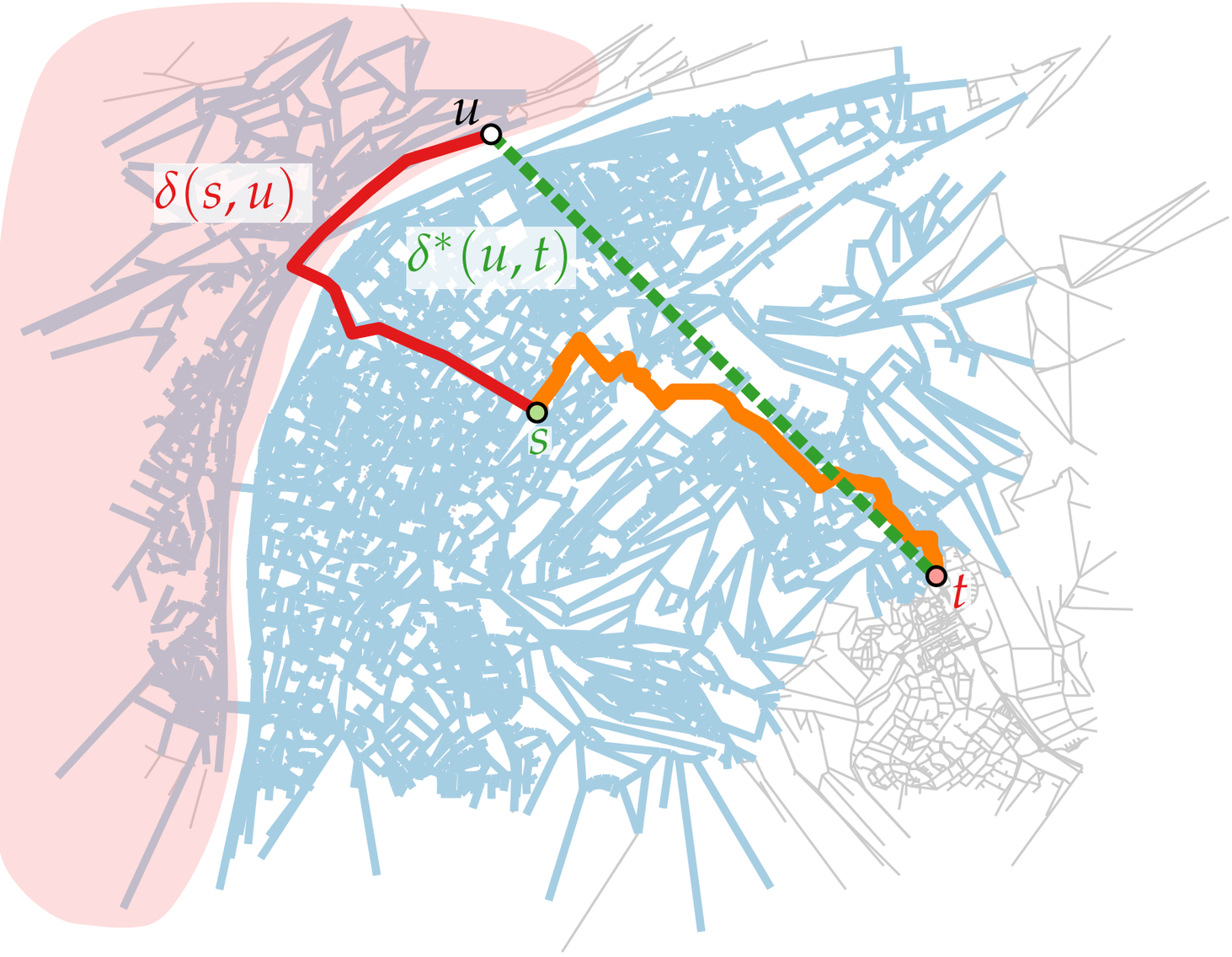
Ist die Suche hier sinnvoll?



Schätzfunktionen

$$\delta(s,u) + \delta(u,t)$$
$$\geq \delta(s,u) + \delta^*(u,t)$$

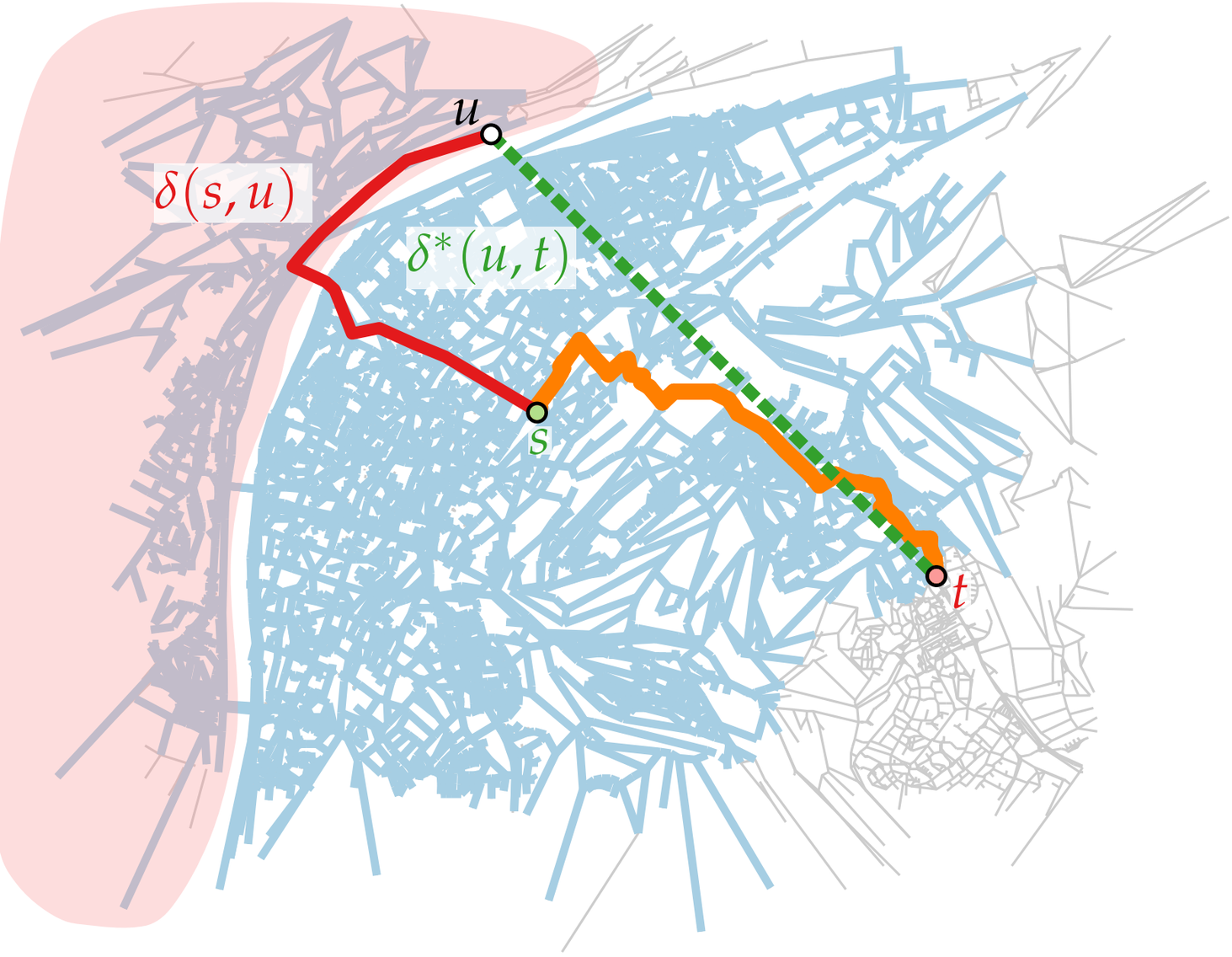
Ist die Suche hier sinnvoll?



Schätzfunktionen

$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

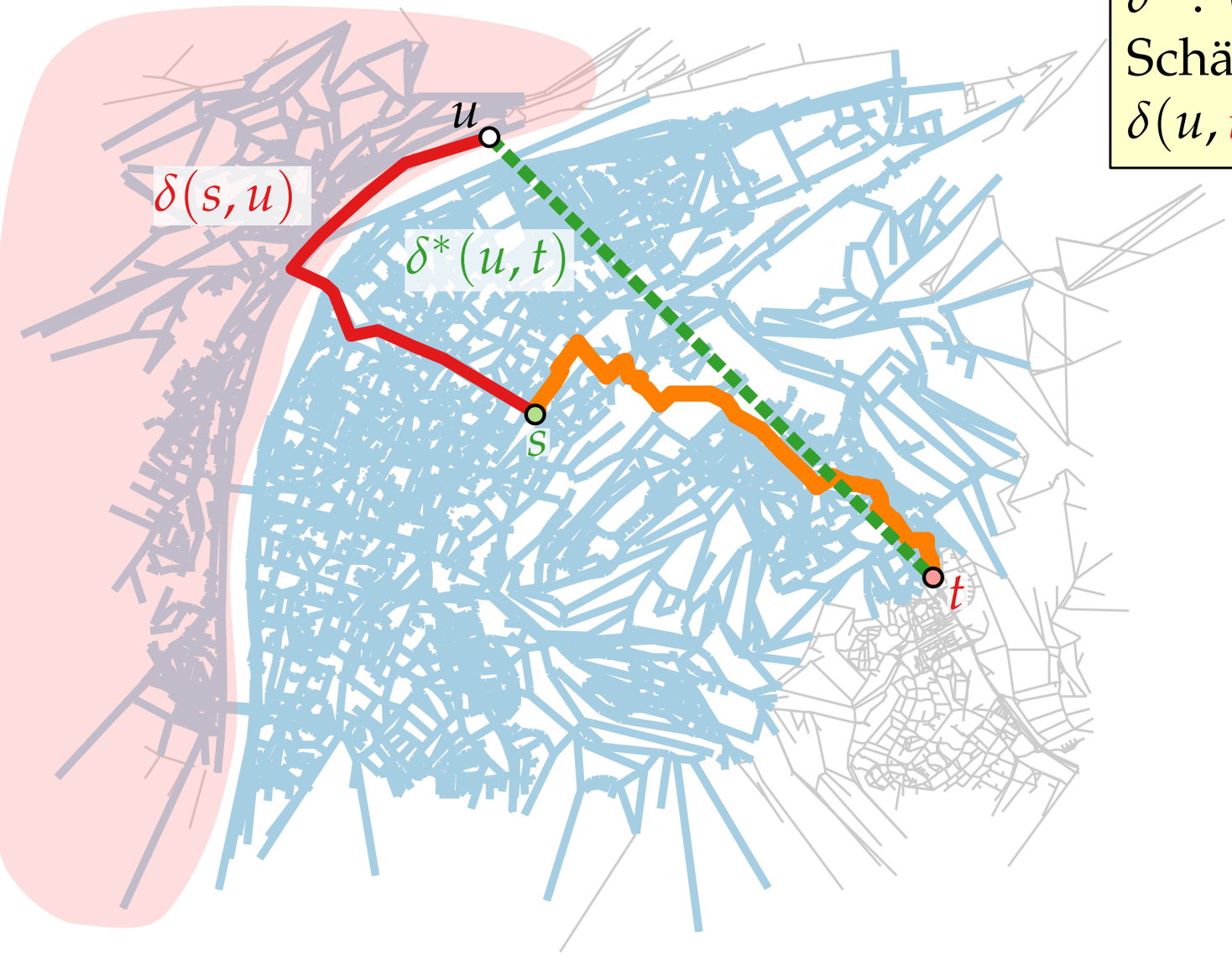
Ist die Suche hier sinnvoll?



Schätzfunktionen

$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

Ist die Suche hier sinnvoll?

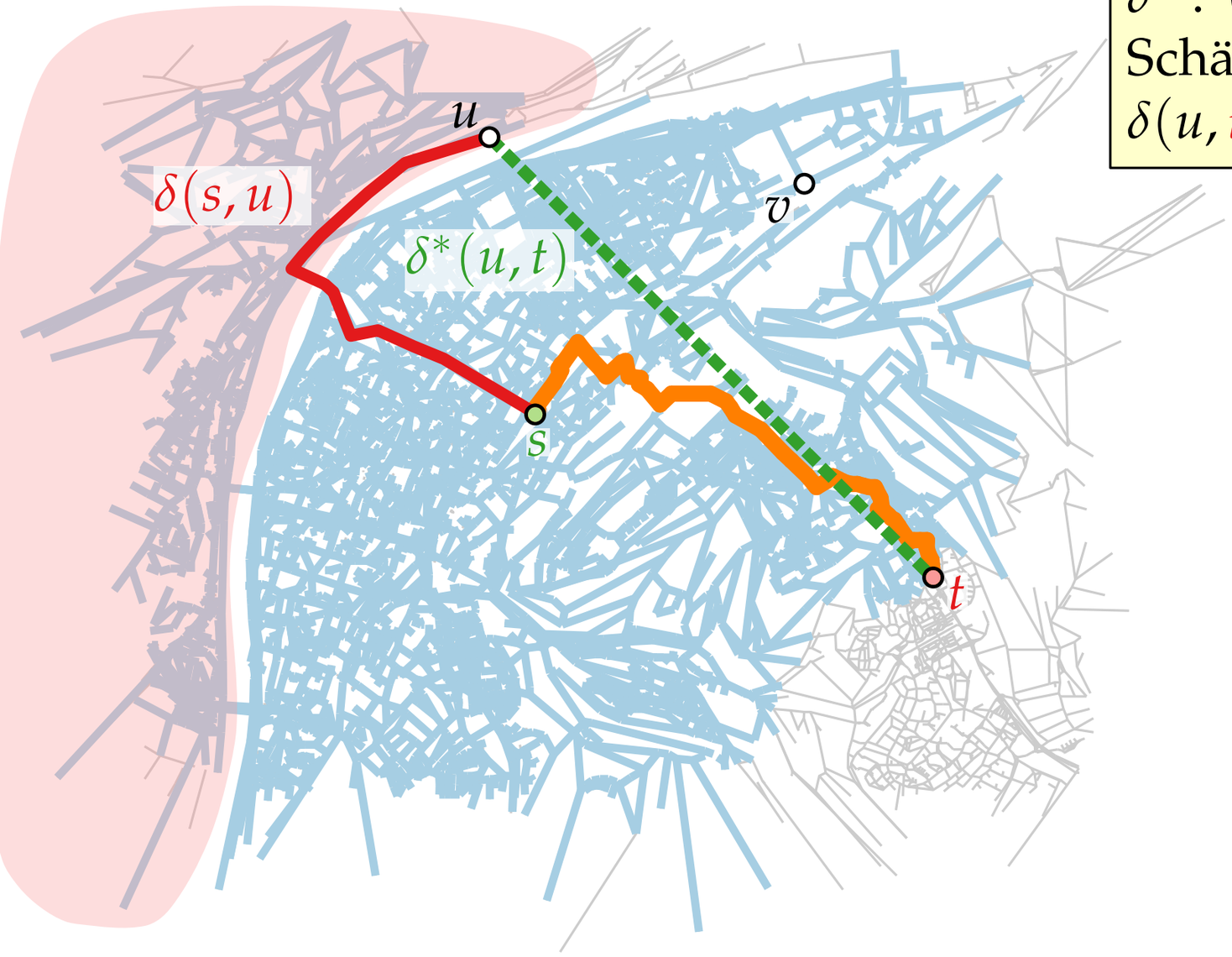


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

Schätzfunktionen

$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

Ist die Suche hier sinnvoll?

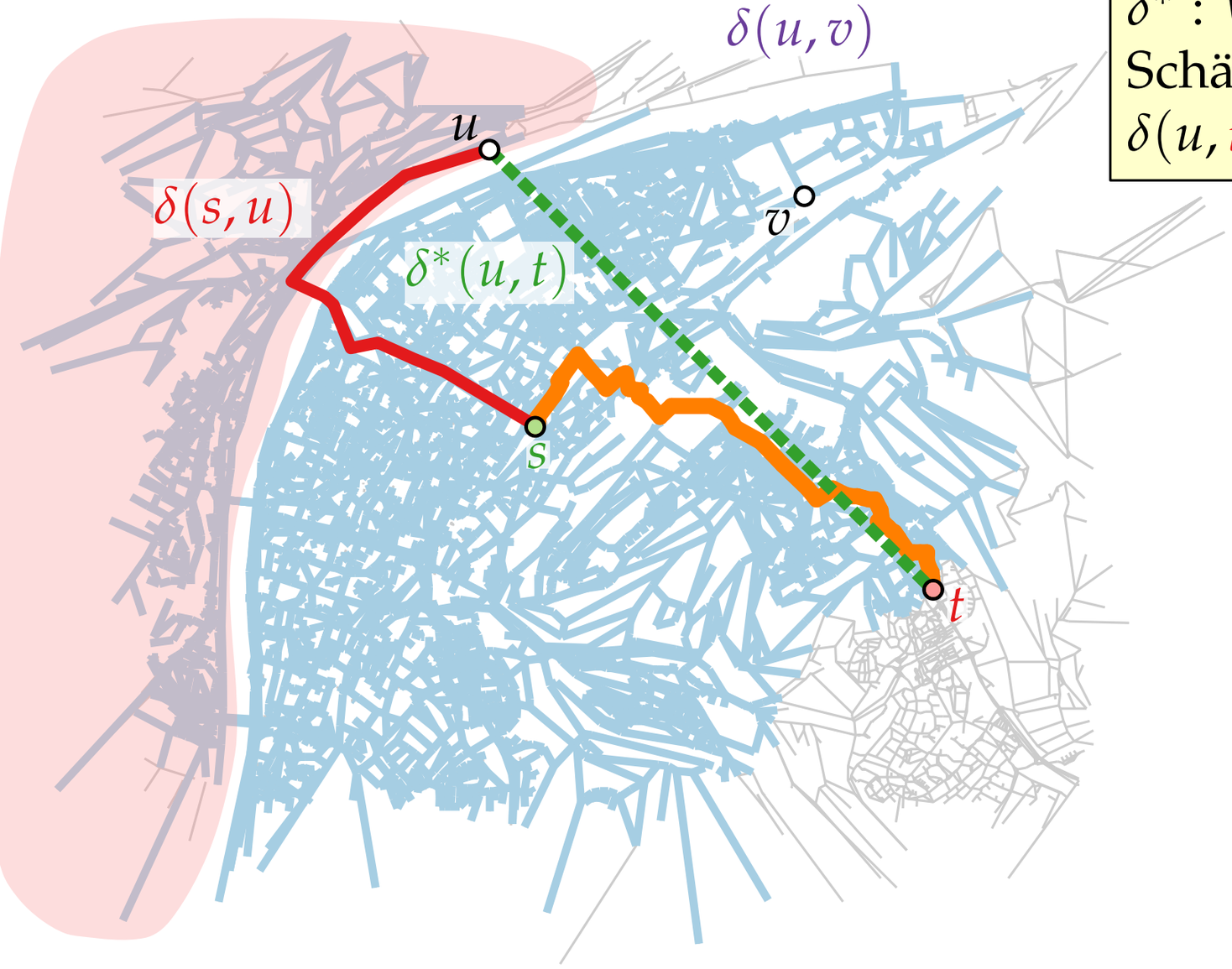


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

Schätzfunktionen

$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

Ist die Suche hier sinnvoll?

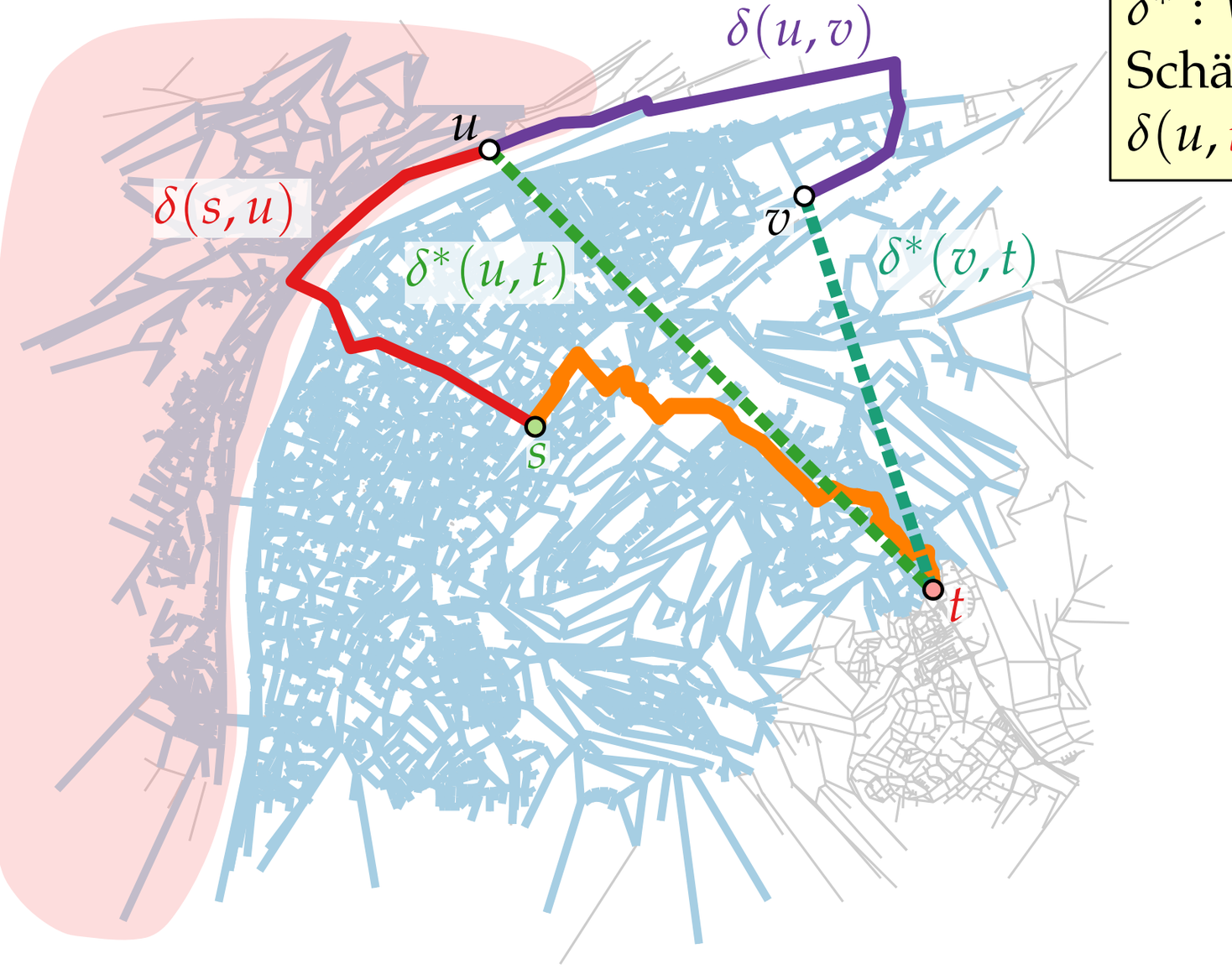


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

Schätzfunktionen

$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

Ist die Suche hier sinnvoll?

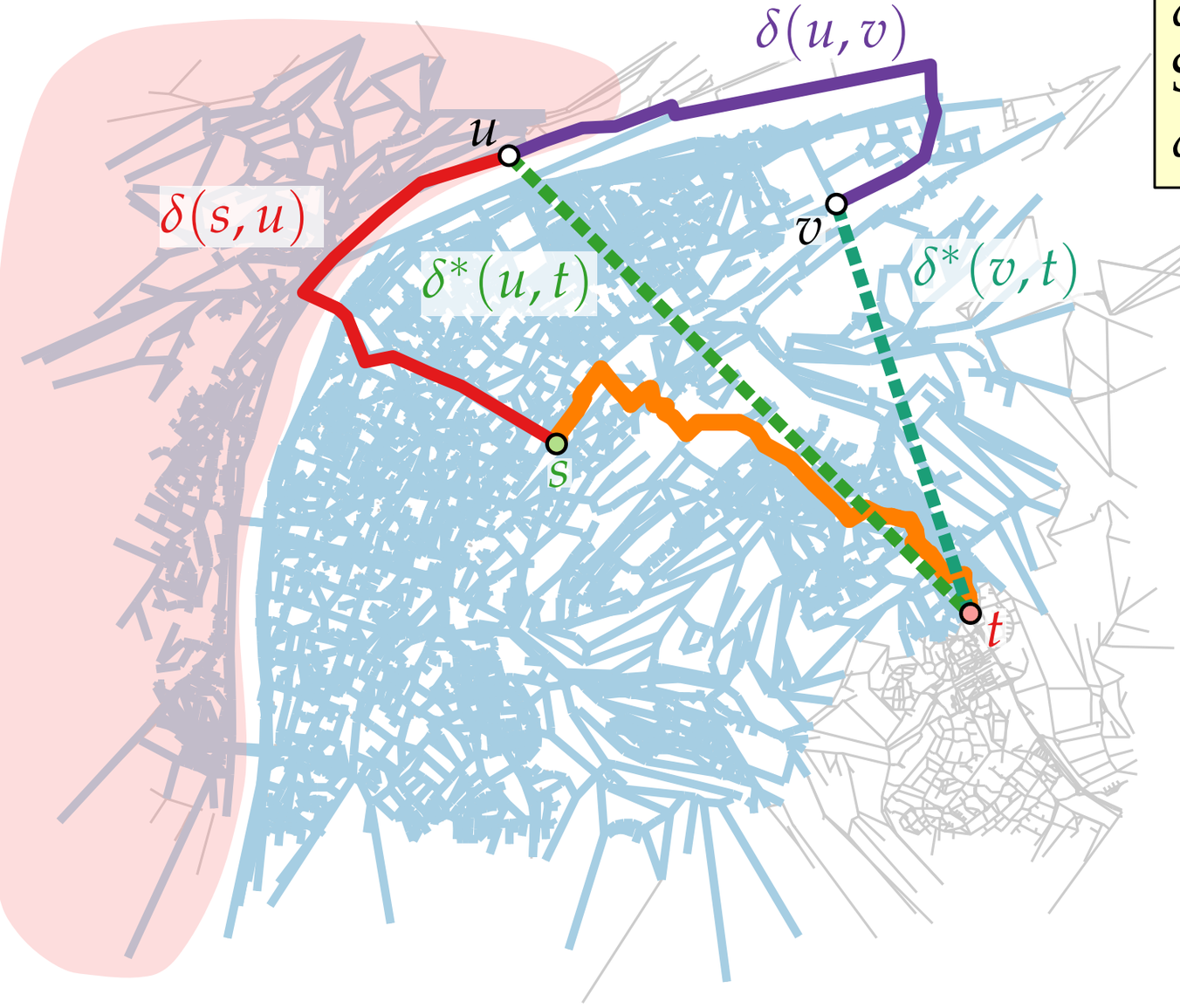


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

Schätzfunktionen

$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

Ist die Suche hier sinnvoll?

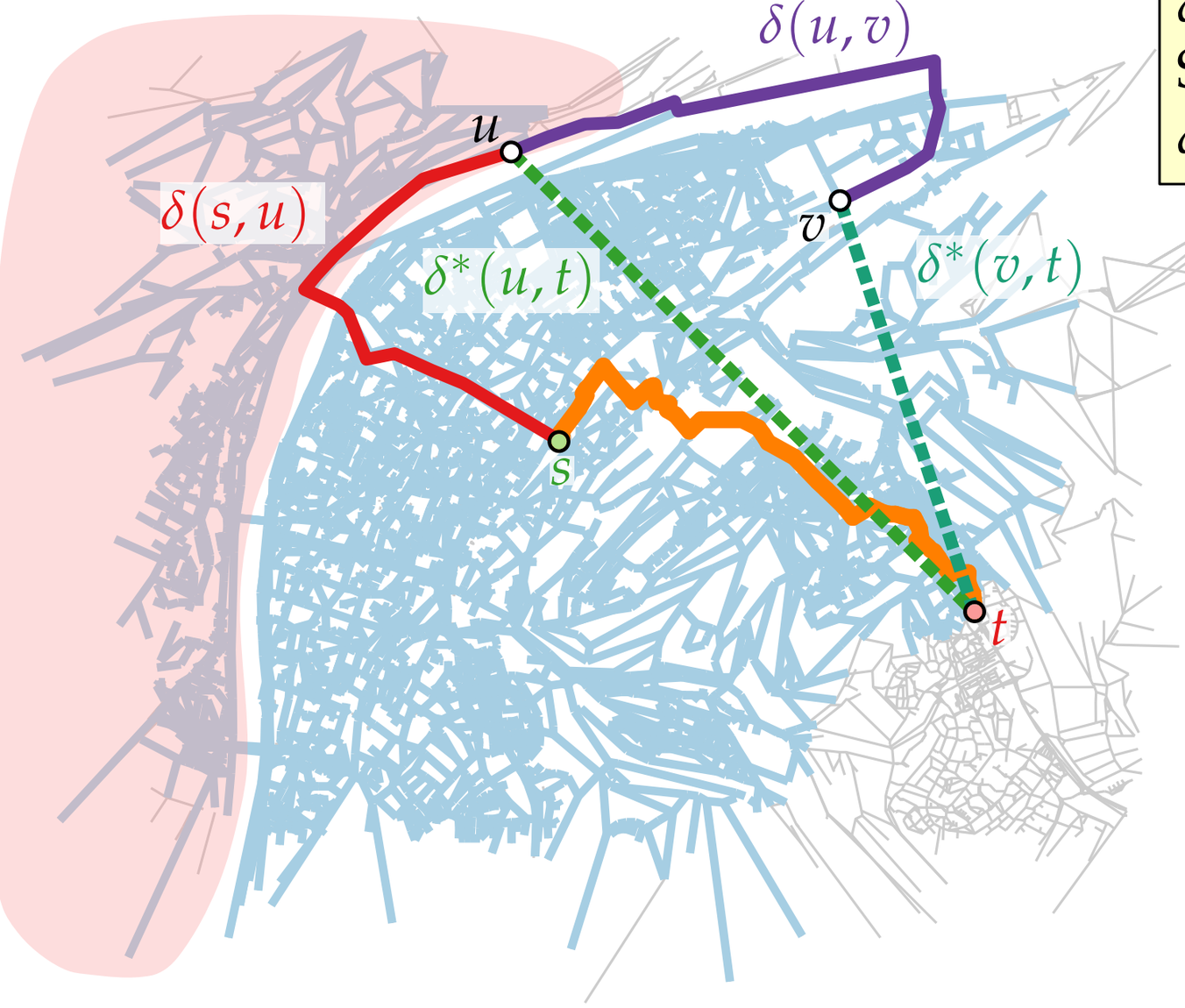


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta(u,v) + \delta^*(v,t)$

Schätzfunktionen

Ist die Suche hier sinnvoll?



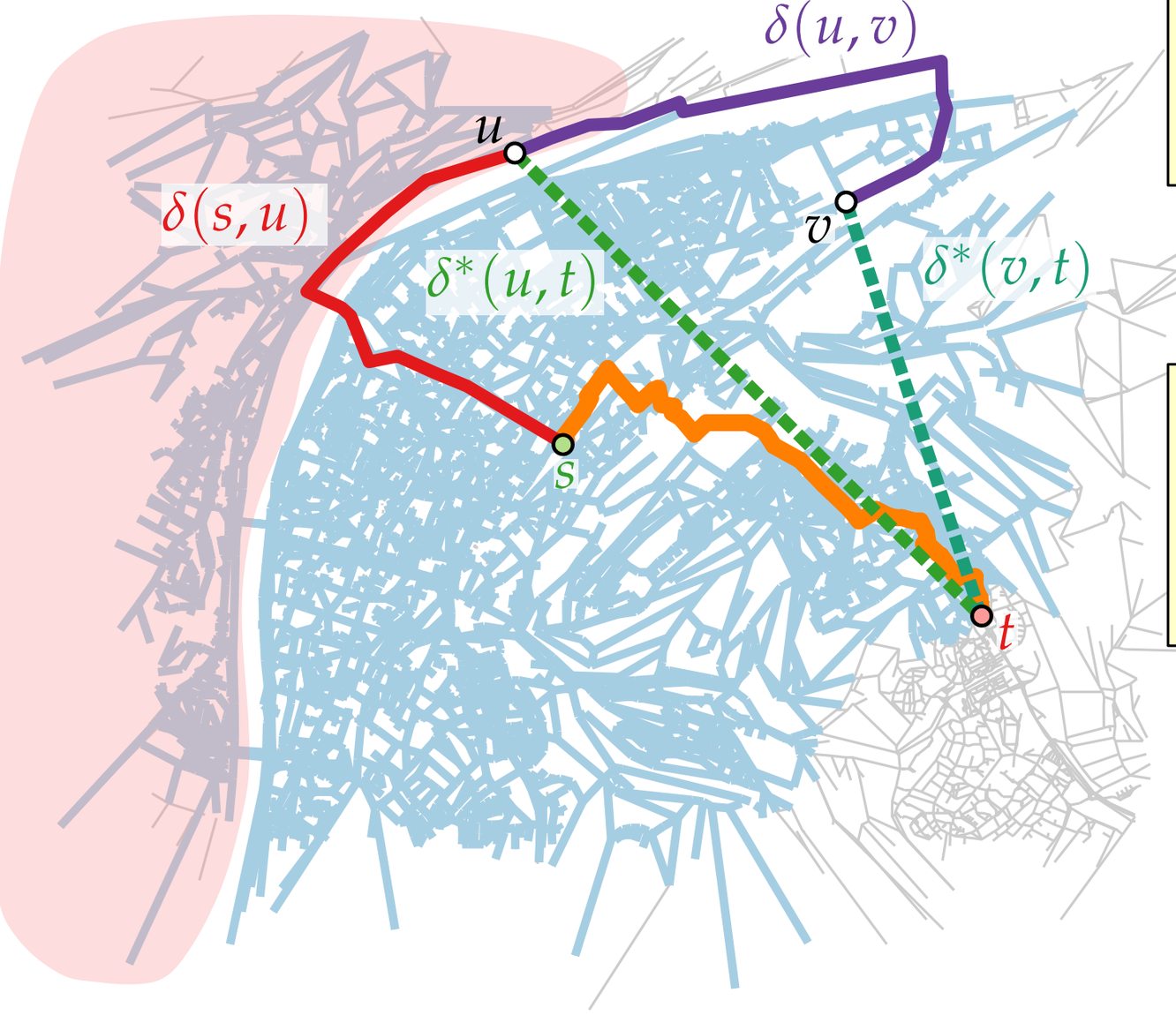
$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$$\delta(u,v) + \delta^*(v,t)$$
$$\geq \delta^*(u,t)$$

Schätzfunktionen

Ist die Suche hier sinnvoll?



$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

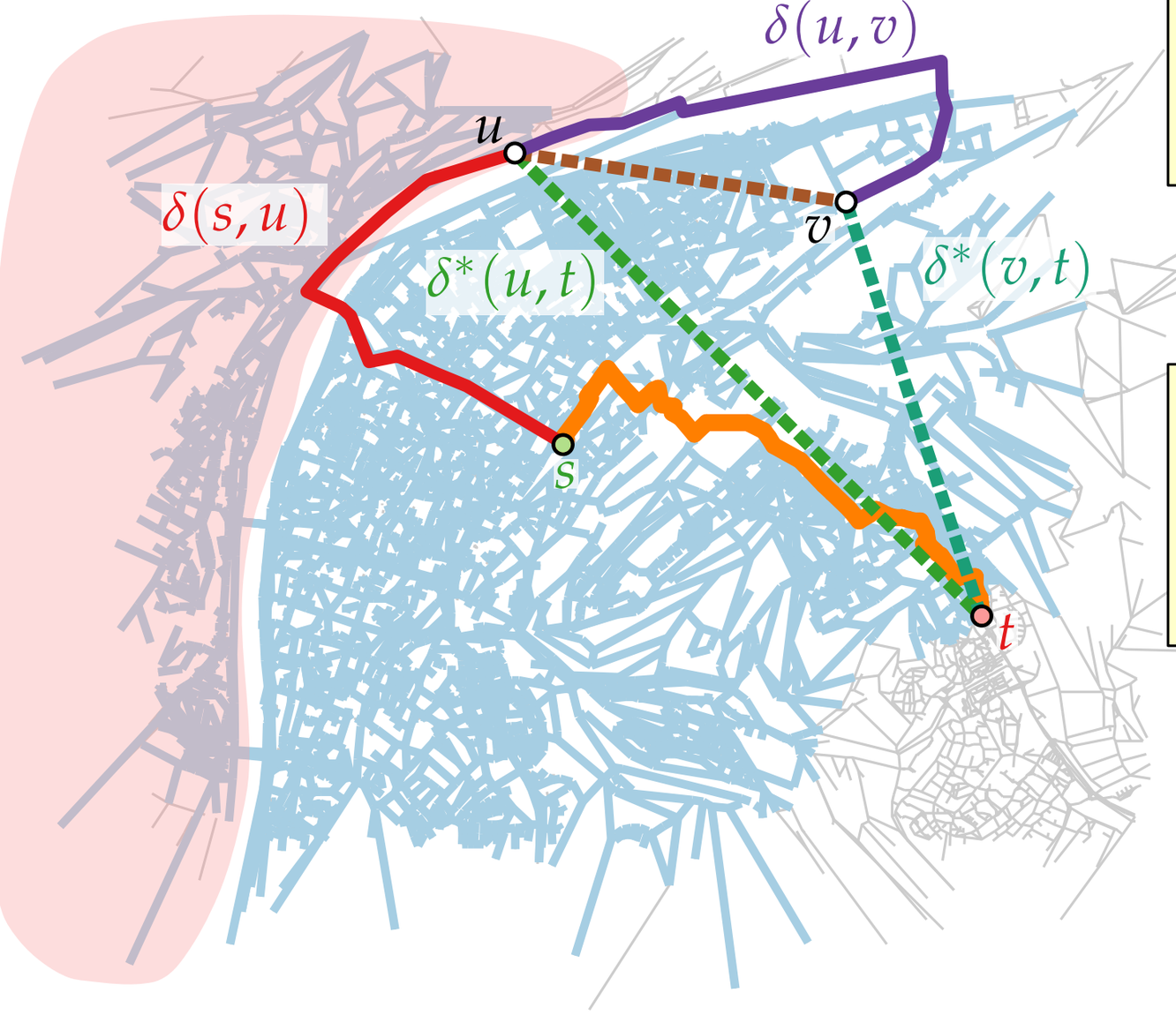
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$$\delta(u,v) + \delta^*(v,t)$$
$$\geq \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Schätzfunktionen

Ist die Suche hier sinnvoll?



$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

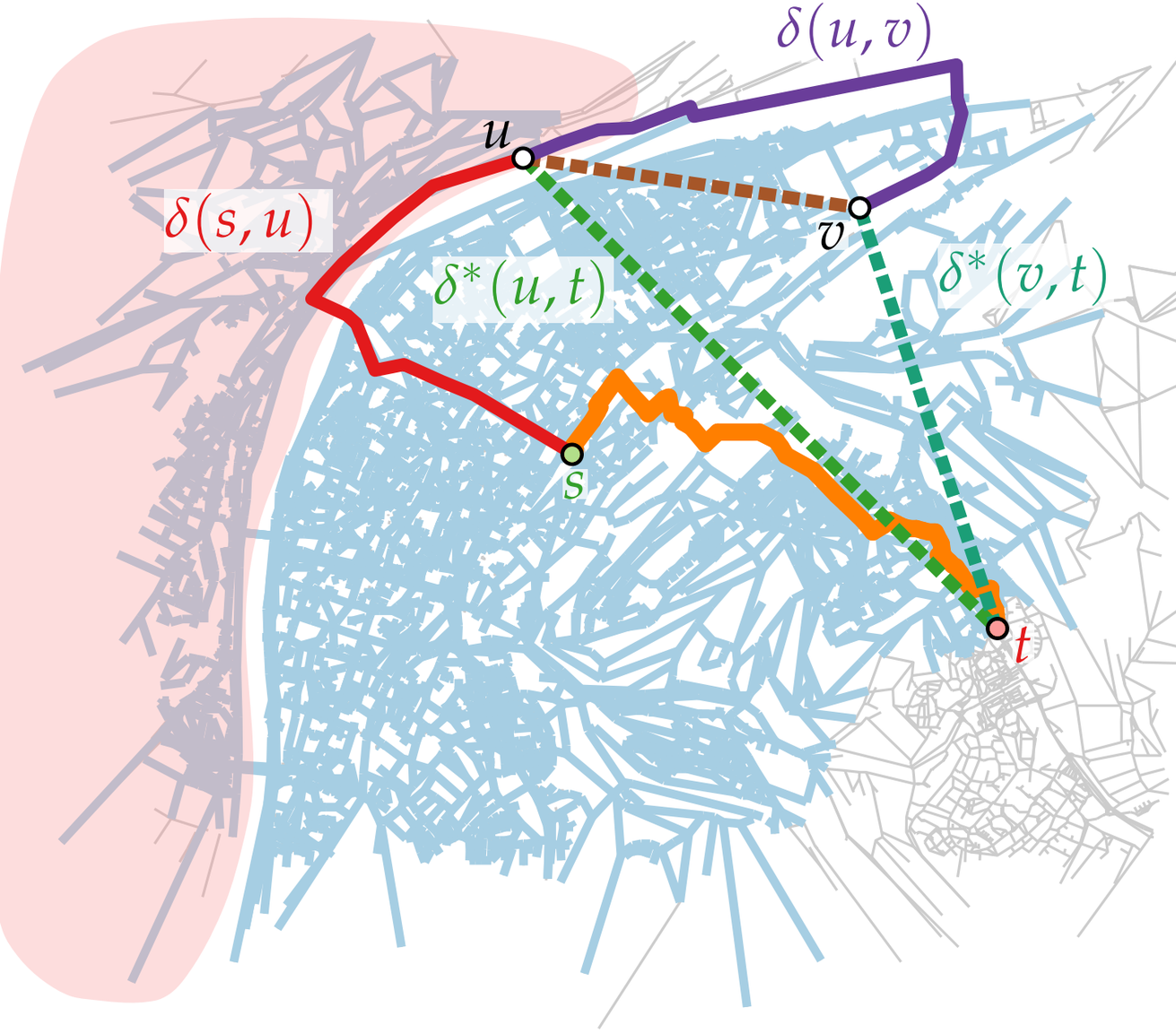
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$$\delta(u,v) + \delta^*(v,t)$$
$$\geq \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Schätzfunktionen

Ist die Suche hier sinnvoll?



$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

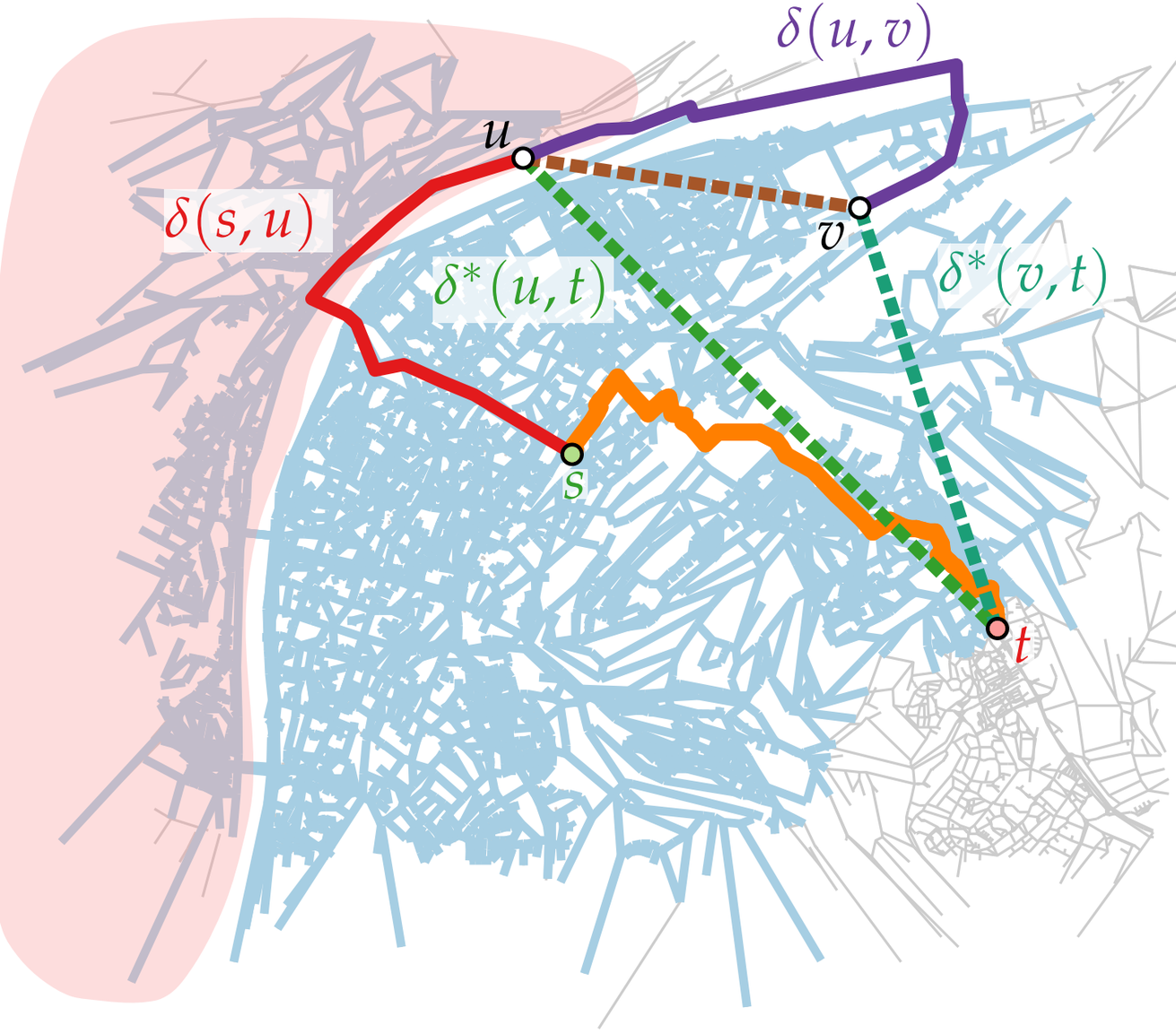
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$$\delta(u,v) + \delta^*(v,t) \quad [\geq \delta^*(u,v) + \delta^*(v,t)]$$
$$\geq \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Schätzfunktionen

Ist die Suche hier sinnvoll?



$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

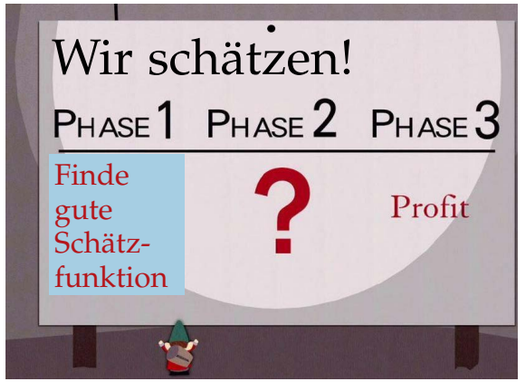
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$$\delta(u,v) + \delta^*(v,t) \quad [\underset{\triangle}{\geq} \delta^*(u,v) + \delta^*(v,t)]$$
$$\geq \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

Idee:



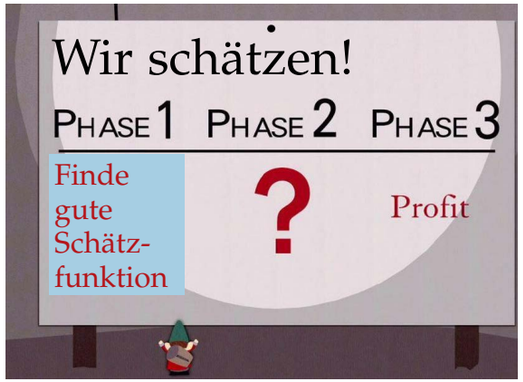
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

1. $\delta^*(u,t) = 0$

Idee:



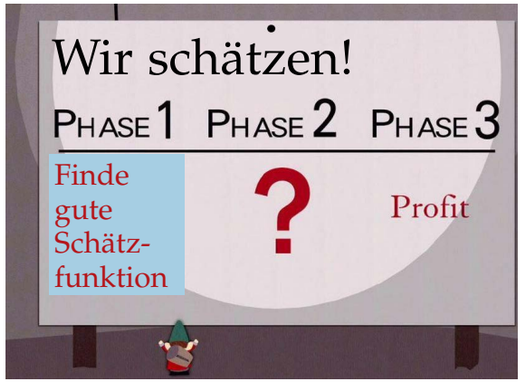
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen

Idee:



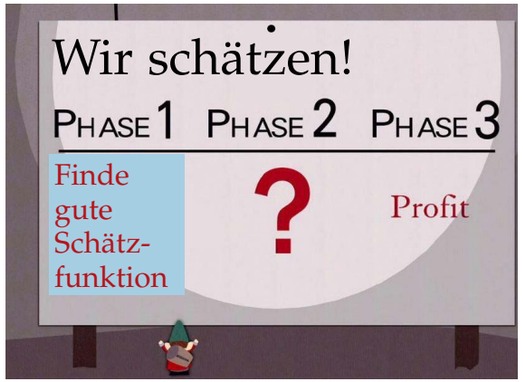
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig

Idee:



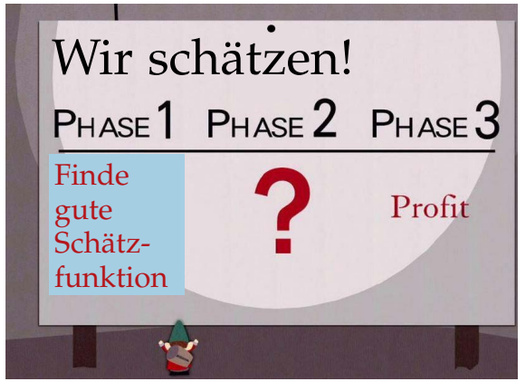
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig
2. $\delta^*(u,t) = \delta(u,t)$

Idee:



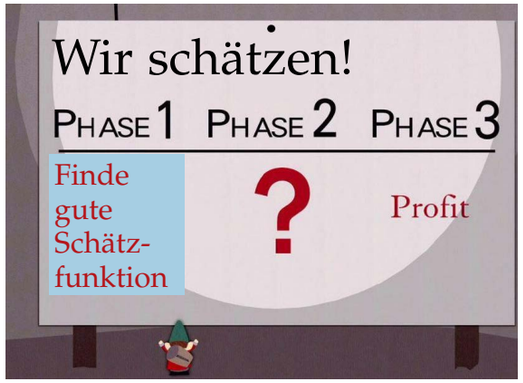
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig
2. $\delta^*(u,t) = \delta(u,t)$
   - ⊕ sehr genau

Idee:



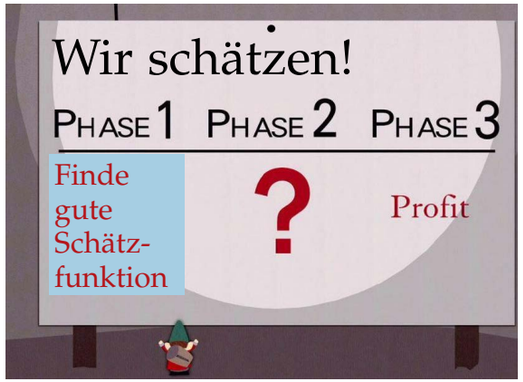
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig
2. $\delta^*(u,t) = \delta(u,t)$
   - ⊕ sehr genau
   - ⊖ zum Ausrechnen müssen wir das Kürzester-Pfad-Problem lösen …

Idee:

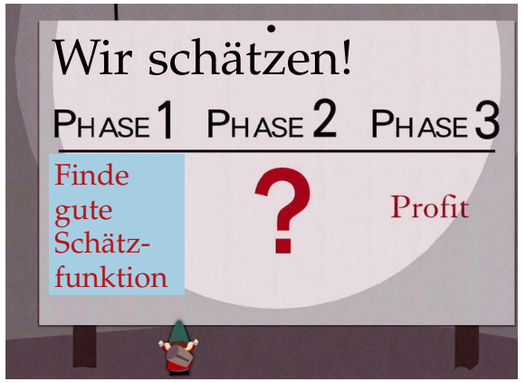


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

Idee:



1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig
2. $\delta^*(u,t) = \delta(u,t)$
   - ⊕ sehr genau
   - ⊖ zum Ausrechnen müssen wir das Kürzester-Pfad-Problem lösen …
3. $\delta^*(u,t) = \delta_{\text{Euklid}}(u,t) = \sqrt{(x(u)-x(t))^2 + (y(u)-y(t))^2}$

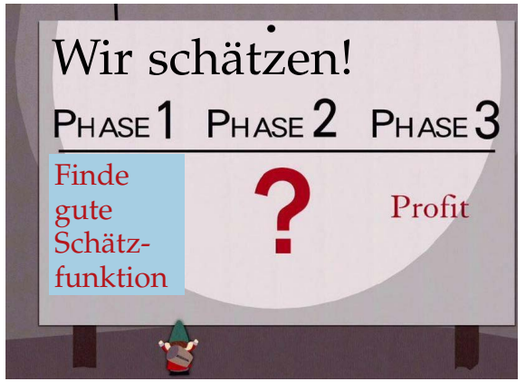
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig
2. $\delta^*(u,t) = \delta(u,t)$
   - ⊕ sehr genau
   - ⊖ zum Ausrechnen müssen wir das Kürzester-Pfad-Problem lösen …
3. $\delta^*(u,t) = \delta_{\text{Euklid}}(u,t) = \sqrt{(x(u)-x(t))^2 + (y(u)-y(t))^2}$
   - ⊕ relativ genau

Idee:

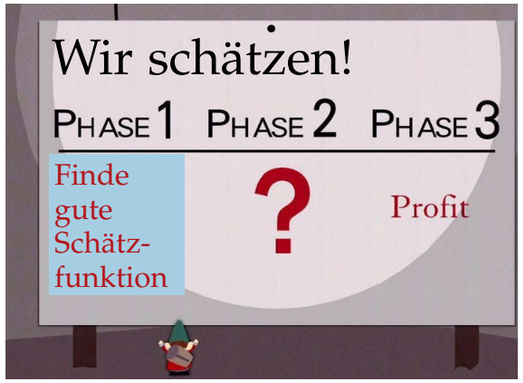


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

Idee:



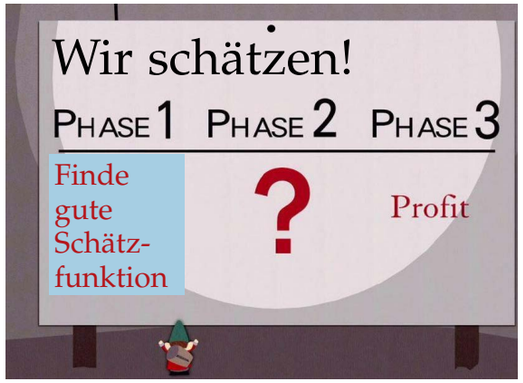
1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig
2. $\delta^*(u,t) = \delta(u,t)$
   - ⊕ sehr genau
   - ⊖ zum Ausrechnen müssen wir das Kürzester-Pfad-Problem lösen …
3. $\delta^*(u,t) = \delta_{\text{Euklid}}(u,t) = \sqrt{(x(u)-x(t))^2 + (y(u)-y(t))^2}$
   - ⊕ relativ genau
   - ⊖ funktioniert nur für Graphen mit geometrischen Informationen

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

Idee:



1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig

2. $\delta^*(u,t) = \delta(u,t)$
   - ⊕ sehr genau
   - ⊖ zum Ausrechnen müssen wir das Kürzester-Pfad-Problem lösen …

3. $\delta^*(u,t) = \delta_{\text{Euklid}}(u,t) = \sqrt{(x(u)-x(t))^2 + (y(u)-y(t))^2}$
   - ⊕ relativ genau
   - ⊖ funktioniert nur für Graphen mit geometrischen Informationen

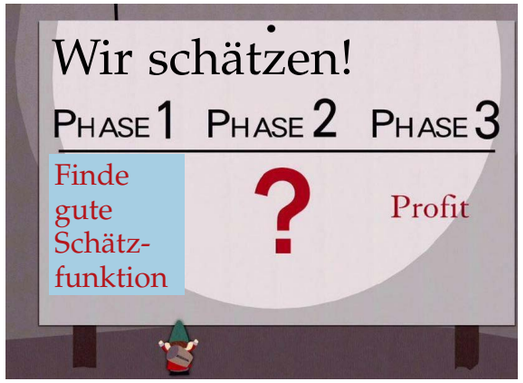
4. $\delta^*(u,t) = \delta^2_{\text{Euklid}}(u,t) = (x(u)-x(t))^2 + (y(u)-y(t))^2$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

Idee:



1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig

2. $\delta^*(u,t) = \delta(u,t)$
   - ⊕ sehr genau
   - ⊖ zum Ausrechnen müssen wir das Kürzester-Pfad-Problem lösen …

3. $\delta^*(u,t) = \delta_{\text{Euklid}}(u,t) = \sqrt{(x(u)-x(t))^2 + (y(u)-y(t))^2}$
   - ⊕ relativ genau
   - ⊖ funktioniert nur für Graphen mit geometrischen Informationen

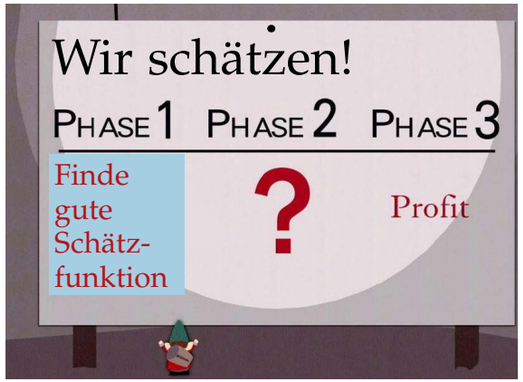
4. $\delta^*(u,t) = \delta^2_{\text{Euklid}}(u,t) = (x(u)-x(t))^2 + (y(u)-y(t))^2$
   - ⊕ relativ genau, keine Wurzel

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Gute Schätzfunktion

Idee:



1. $\delta^*(u,t) = 0$
   - ⊕ leicht zu berechnen
   - ⊖ nicht aussagekräftig

2. $\delta^*(u,t) = \delta(u,t)$
   - ⊕ sehr genau
   - ⊖ zum Ausrechnen müssen wir das Kürzester-Pfad-Problem lösen …

3. $\delta^*(u,t) = \delta_{\text{Euklid}}(u,t) = \sqrt{(x(u)-x(t))^2 + (y(u)-y(t))^2}$
   - ⊕ relativ genau
   - ⊖ funktioniert nur für Graphen mit geometrischen Informationen

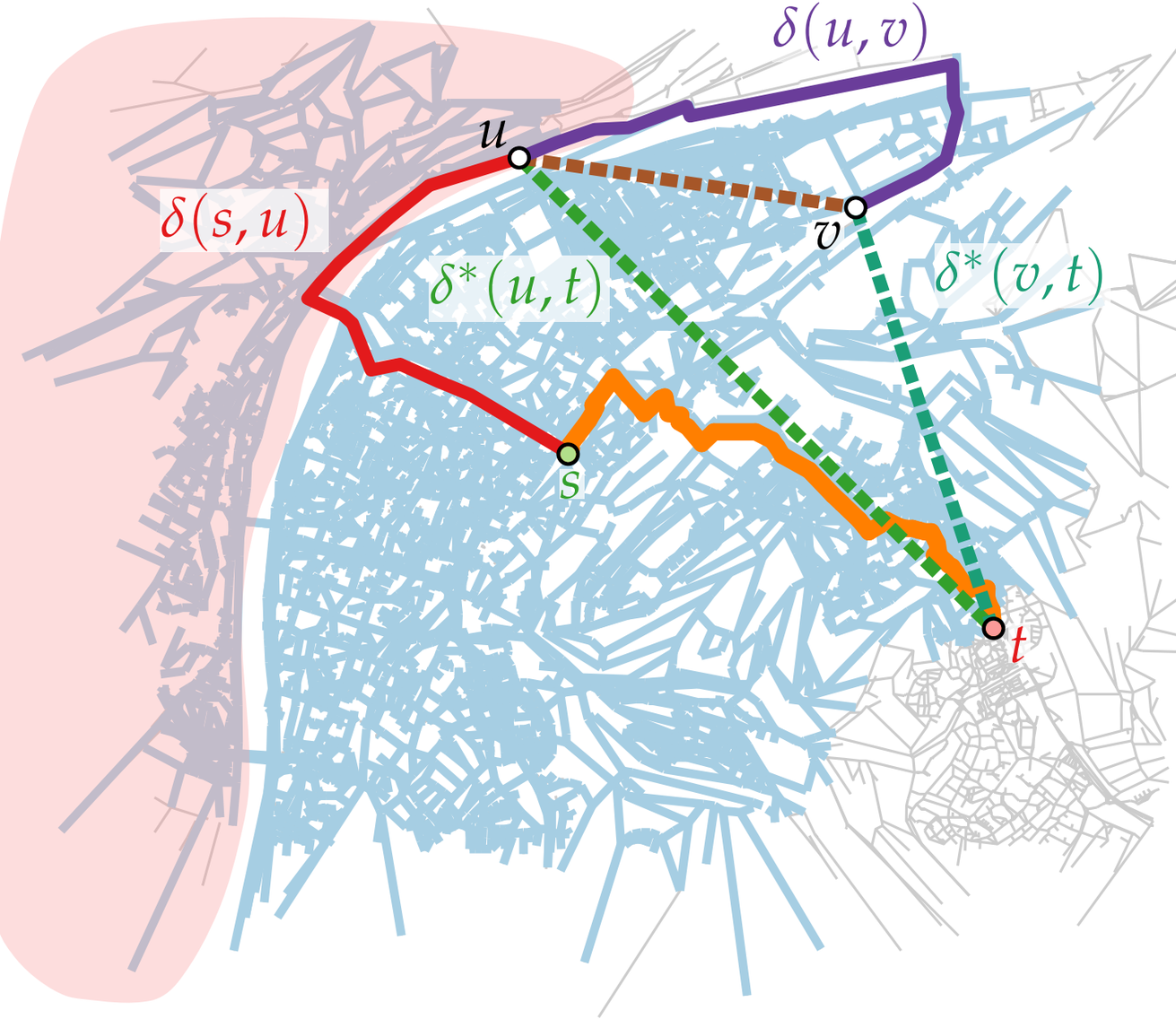
4. $\delta^*(u,t) = \delta^2_{\text{Euklid}}(u,t) = (x(u)-x(t))^2 + (y(u)-y(t))^2$
   - ⊕ relativ genau, keine Wurzel
   - ⊖ nicht **optimistisch**

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Neue Gewichte

Ist die Suche hier sinnvoll?



$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

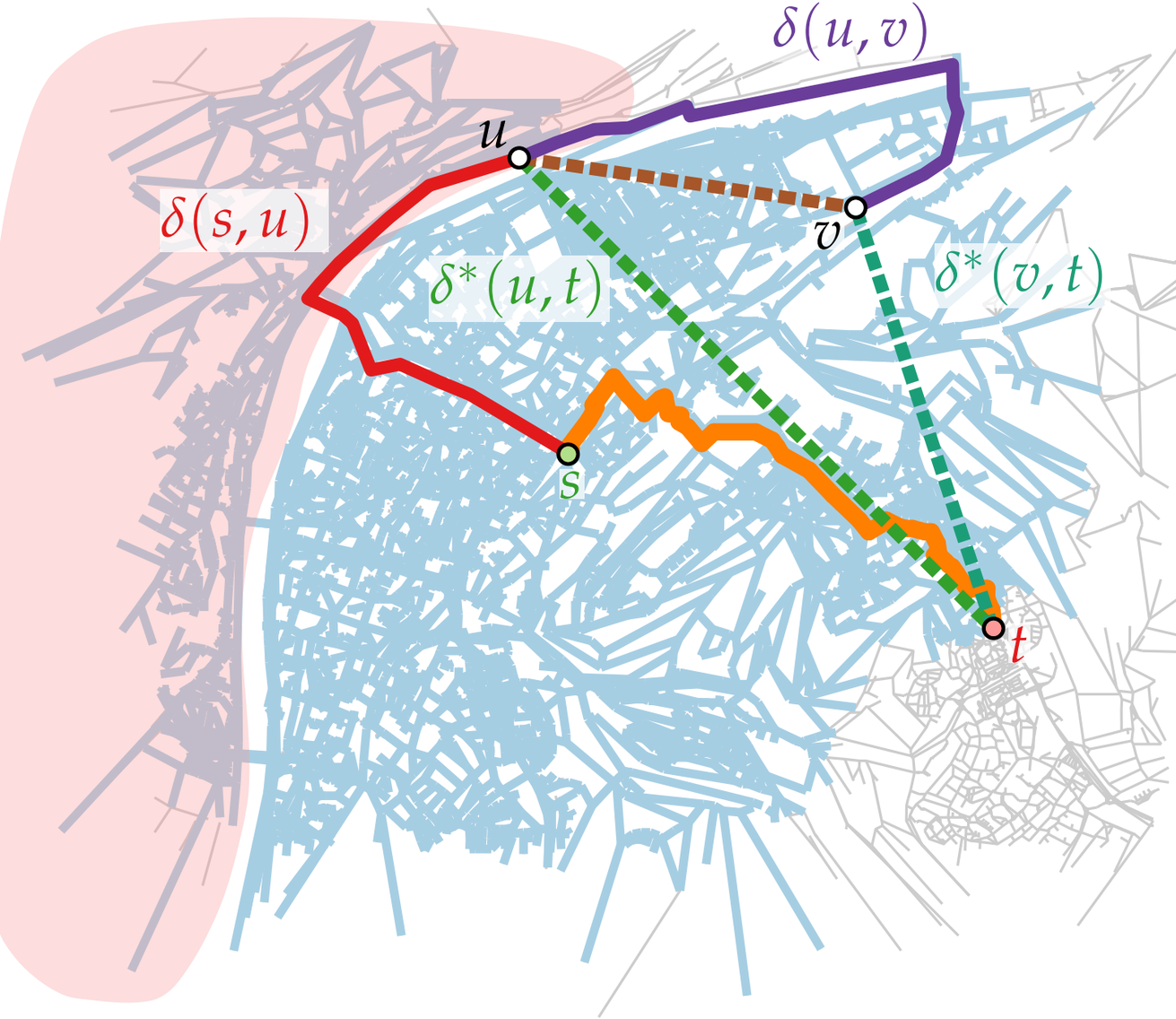
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$$\delta(u,v) + \delta^*(v,t) \quad [\underset{\triangle}{\geq} \delta^*(u,v) + \delta^*(v,t)]$$
$$\geq \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Neue Gewichte

Ist die Suche hier sinnvoll?



$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

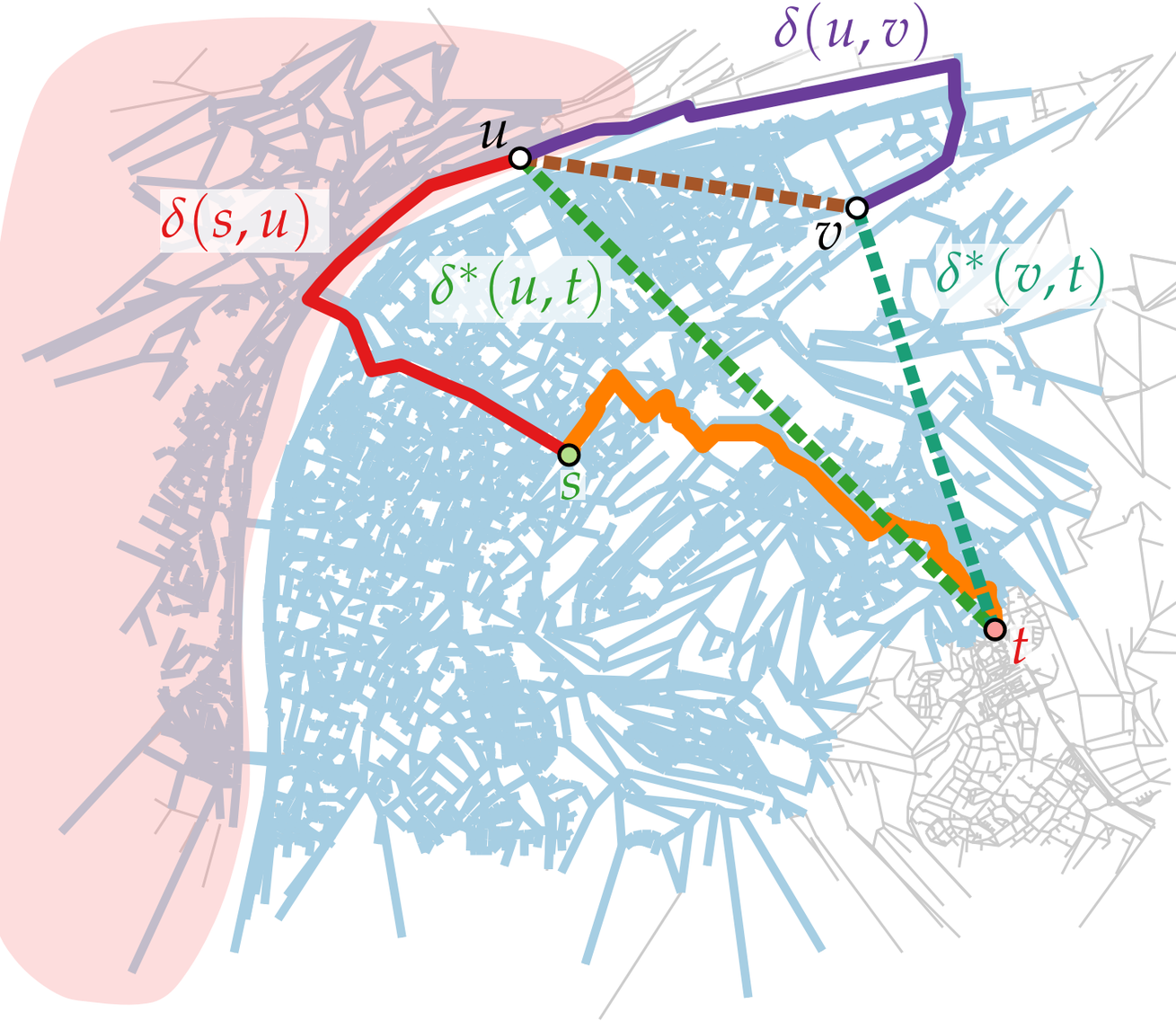
$$\delta(u,v) + \delta^*(v,t) \quad [\underset{\triangle}{\geq} \delta^*(u,v) + \delta^*(v,t)]$$
$$\geq \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Neue Kantengewichte:

Neue Gewichte

Ist die Suche hier sinnvoll?



$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$$\delta(u,v) + \delta^*(v,t) \quad [\underset{\triangle}{\geq} \delta^*(u,v) + \delta^*(v,t)]$$
$$\geq \delta^*(u,t)$$

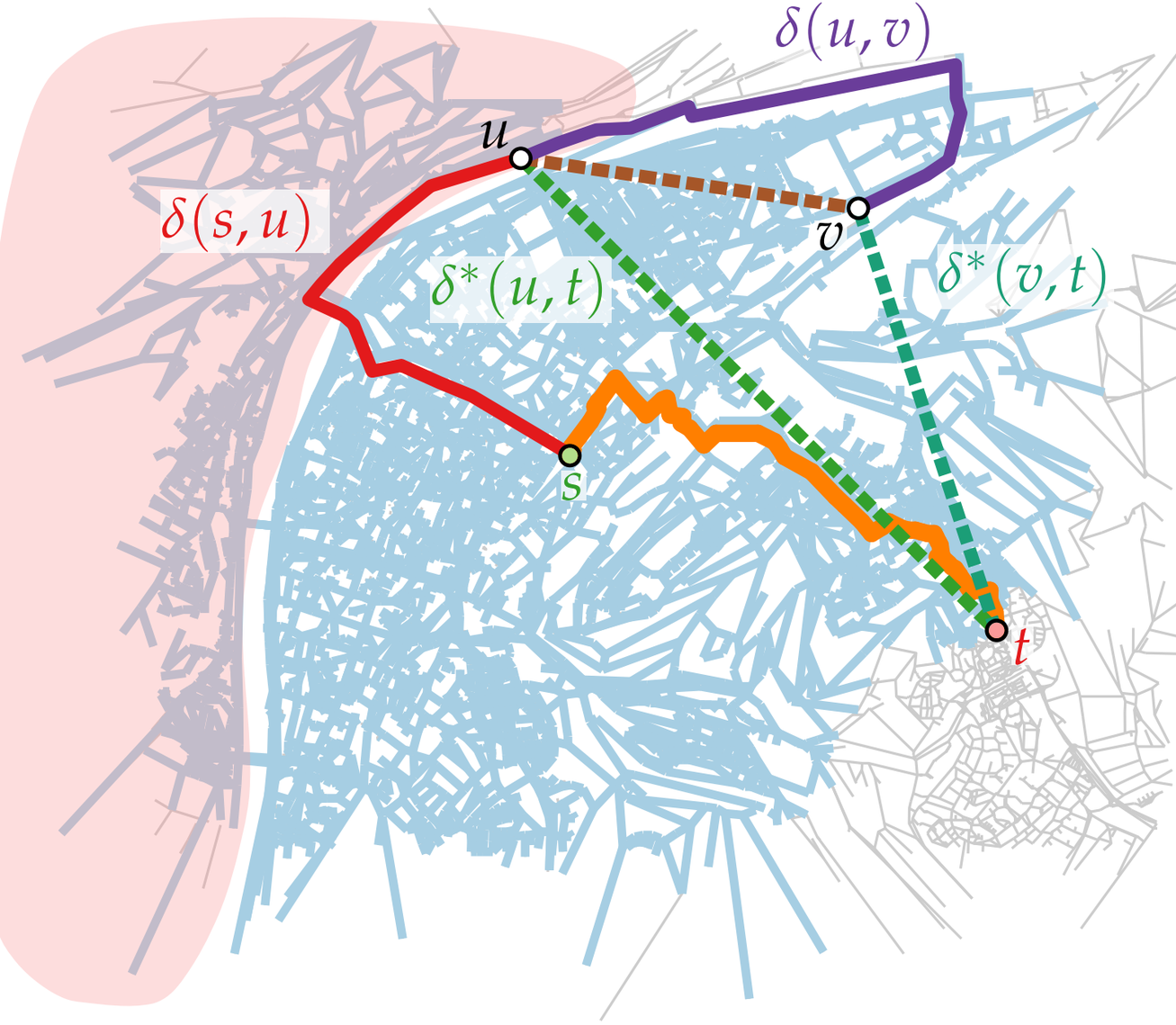
$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Neue Kantengewichte:

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

Neue Gewichte

Ist die Suche hier sinnvoll?



$$\cancel{\delta(s,u)} + \delta(u,t)$$
$$\geq \cancel{\delta(s,u)} + \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$$\delta(u,v) + \delta^*(v,t) \quad [\underset{\triangle}{\geq} \delta^*(u,v) + \delta^*(v,t)]$$
$$\geq \delta^*(u,t)$$

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

Neue Kantengewichte:

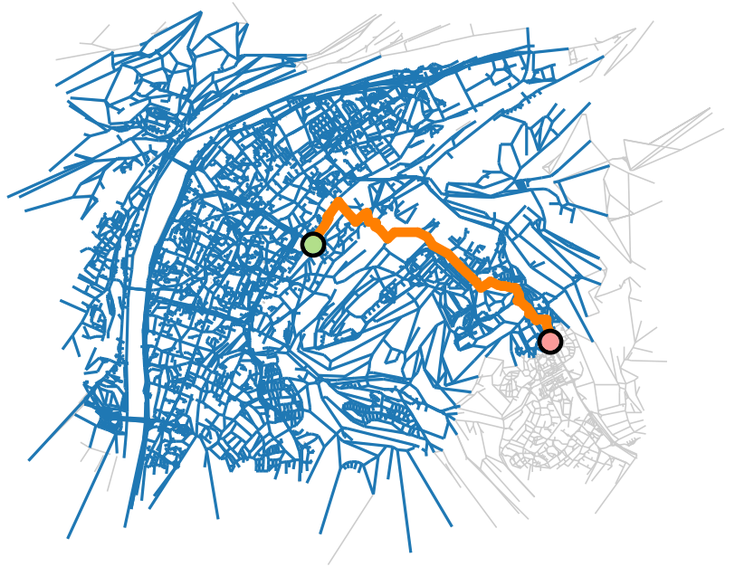
$$w'(u,v) = w(u,v) + \underbrace{\delta^*(v,t) - \delta^*(u,t)}$$

„wie viel näher kommen wir zu t?“

Universität Trier

Algorithmen für geographische Informationssysteme

2. Vorlesung
Routenplanung



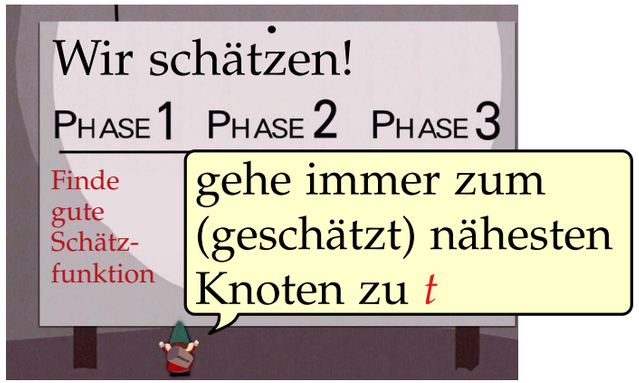
Teil III:
A*-Algorithmus

Philipp Kindermann

Wintersemester 2024/25

A*-Algorithmus

Idee:

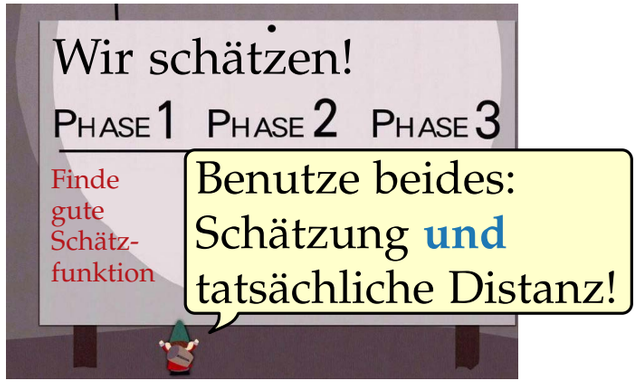


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

A*-Algorithmus

Idee:

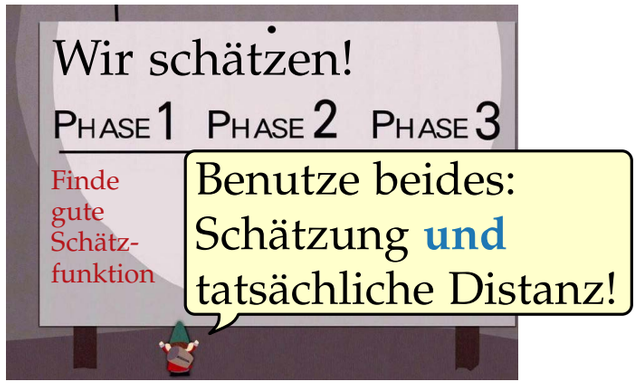


$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

A*-Algorithmus

Idee:




```
Dijkstra(WeightedGraph G, Vertices s, t)
  Initialize(G, s)
  Q = new PriorityQueue(V, d)
  while not Q.Empty() do
      u = Q.ExtractMin()
      if u == t then return
      foreach v ∈ Adj[u] do
          if v.d > u.d + w(u, v) then
              v.d = u.d + w(u, v)
              v.π = u
              Q.DecreaseKey(v, v.d)
```

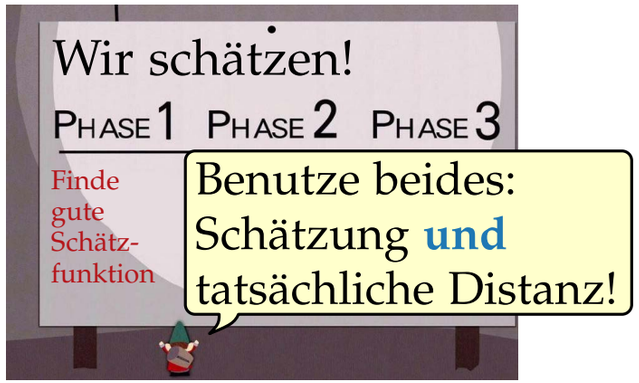
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.d = ∞
      u.π = nil
  s.d = 0
```

A*-Algorithmus

Idee:




```
A*(WeightedGraph G, Vertices s, t)
  Initialize(G, s)
  Q = new PriorityQueue(V, d)
  while not Q.Empty() do
      u = Q.ExtractMin()
      if u == t then return
      foreach v ∈ Adj[u] do
          if v.d > u.d + w(u, v) then
              v.d = u.d + w(u, v)
              v.π = u
              Q.DecreaseKey(v, v.d)
```

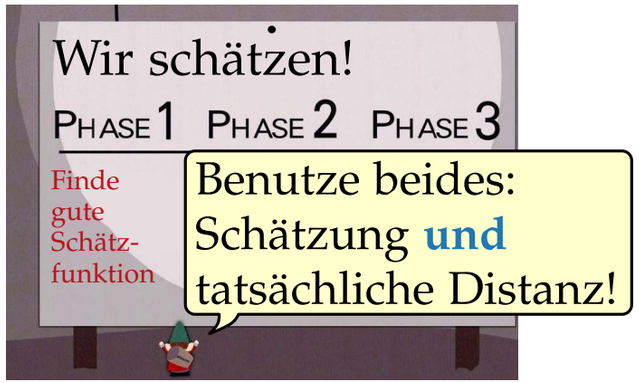
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.d = ∞
      u.π = nil
  s.d = 0
```

A*-Algorithmus

Idee:



$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

A*(WeightedGraph G, Vertices s, t)
- Initialize(G, s)
- Q = **new** PriorityQueue(V, d)
- **while not** Q.Empty() **do**
  - $u = Q$.ExtractMin()
  - **if** $u == t$ **then return**
  - **foreach** $v \in$ Adj$[u]$ **do**
    - **if** $v.d > u.d + w(u,v)$ **then**
      - $v.d = u.d + w(u,v)$
      - $v.\pi = u$
      - Q.DecreaseKey$(v, v.d)$

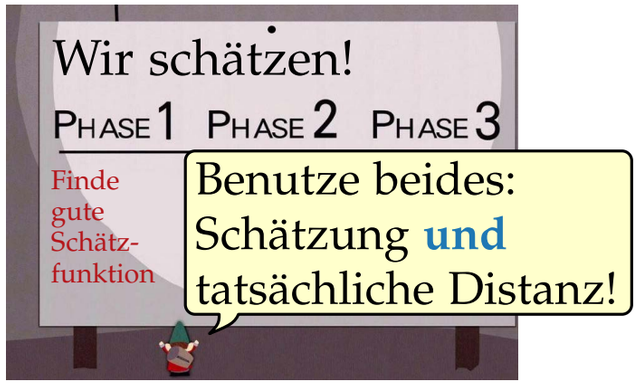
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

Initialize(Graph G, Vertex s)
- **foreach** $u \in V$ **do**
  - $u.d = \infty$
  - $u.\pi = nil$
- $s.d = 0$

A*-Algorithmus

Idee:



$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

A*(WeightedGraph G, Vertices s, t)

```
Initialize(G, s)
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    if u == t then return
    foreach v ∈ Adj[u] do
        if v.d > u.d + w'(u,v) then
            v.d = u.d + w'(u,v)
            v.π = u
            Q.DecreaseKey(v, v.d)
```

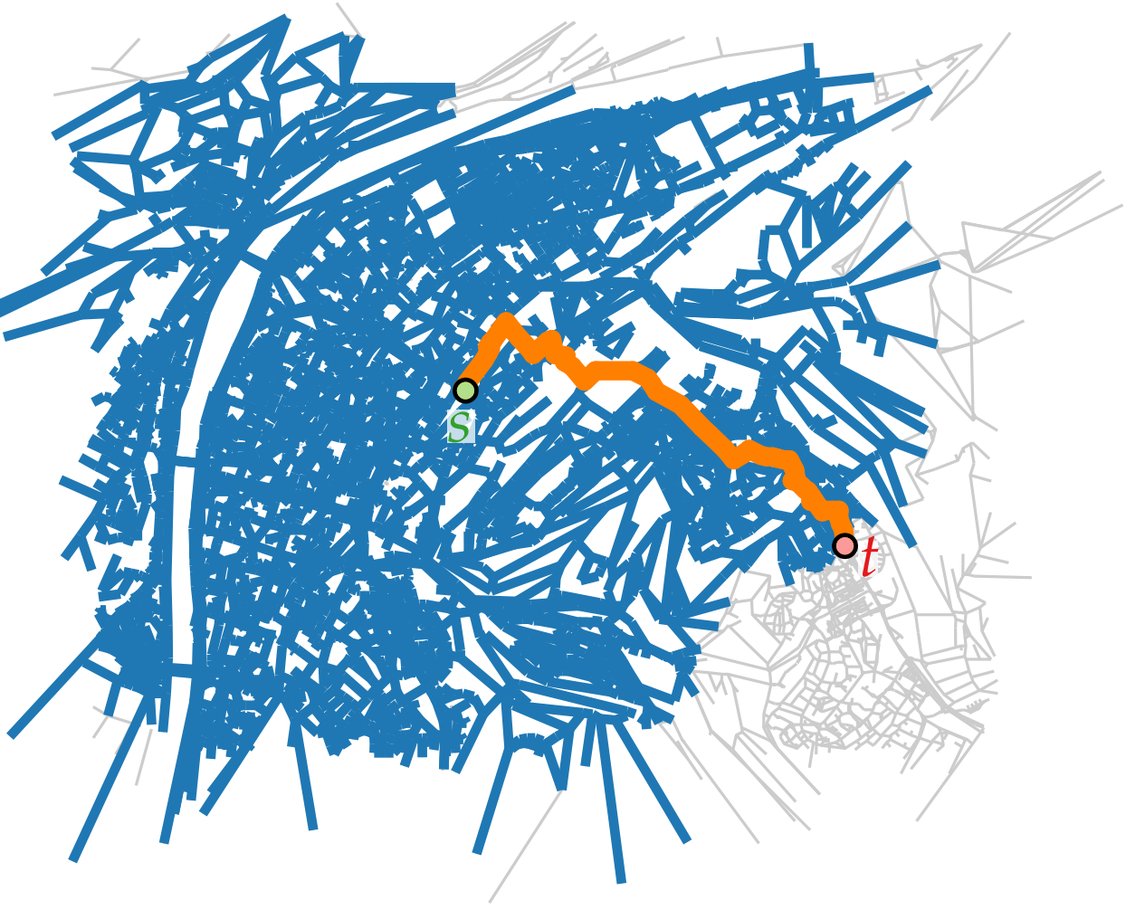
$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

Initialize(Graph G, Vertex s)

```
foreach u ∈ V do
    u.d = ∞
    u.π = nil
s.d = 0
```

Vergleich Dijkstra – A*



Dijkstra

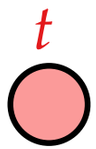
A*

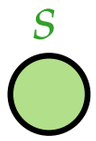
Demo.

`https://algo.uni-trier.de/demos/shortestpath.html`

A* – Korrektheit

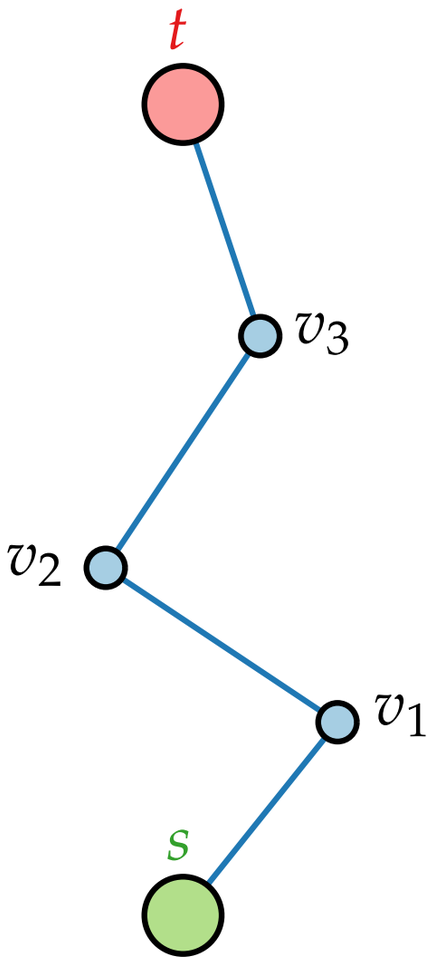
$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$





A* – Korrektheit

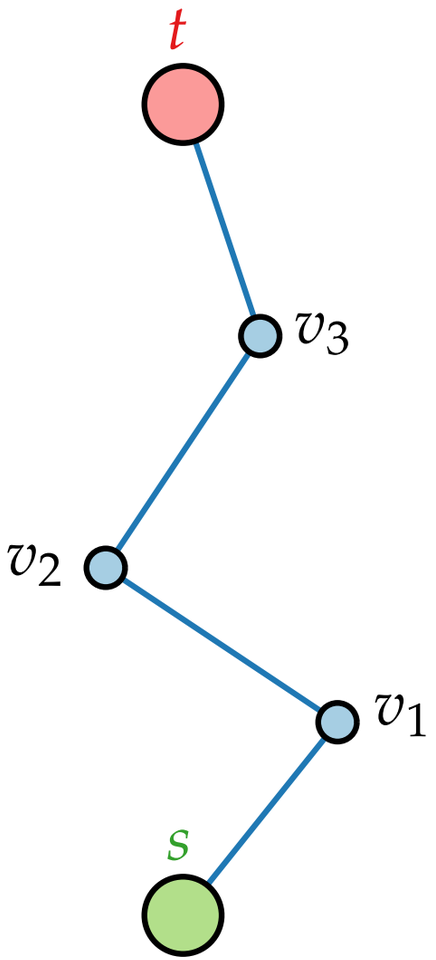
$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$



kürzester s–t–Pfad P

A* – Korrektheit

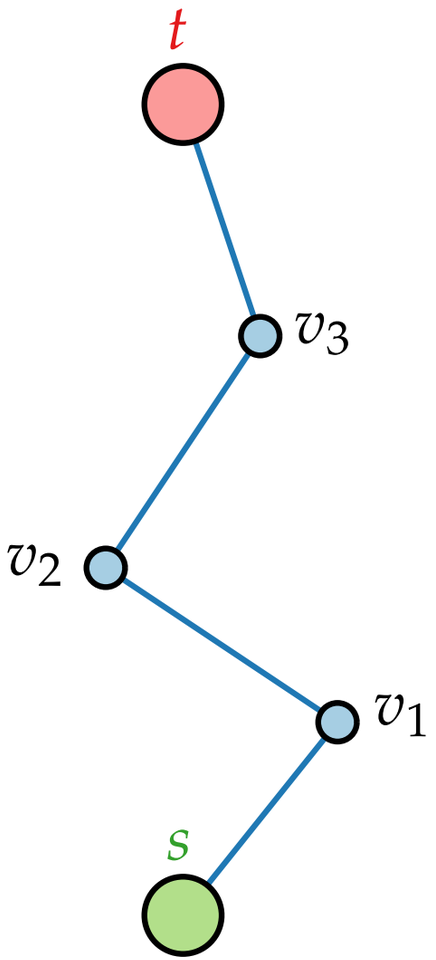
$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$



kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

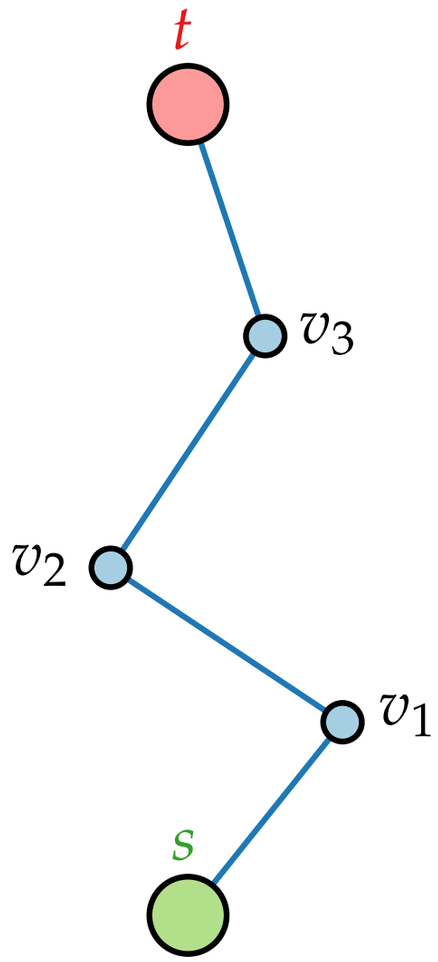


$$w'(P) = w'(v_3,t)$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

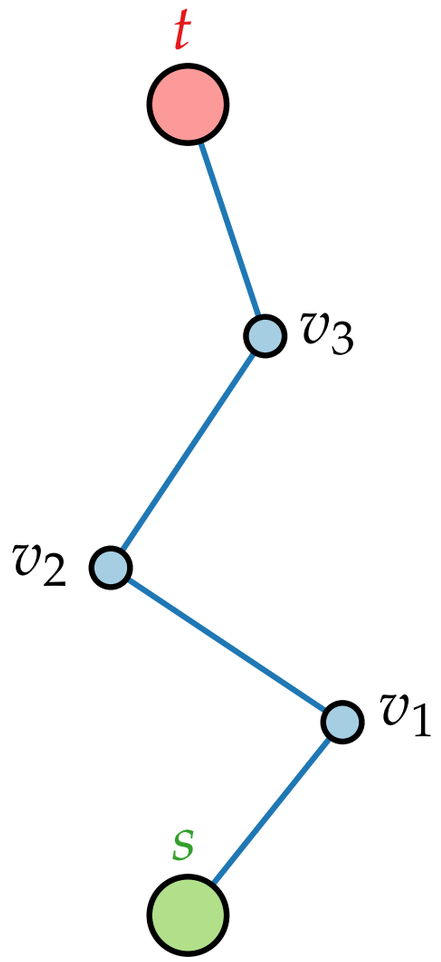


$$w'(P) = w'(v_3,t) + w'(v_2,v_3)$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

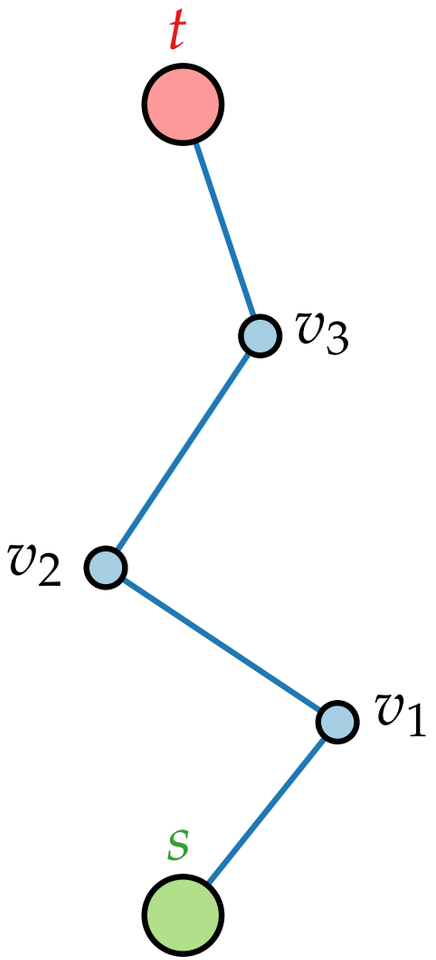


$$w'(P) = w'(v_3,t) + w'(v_2,v_3) + w'(v_1,v_2)$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

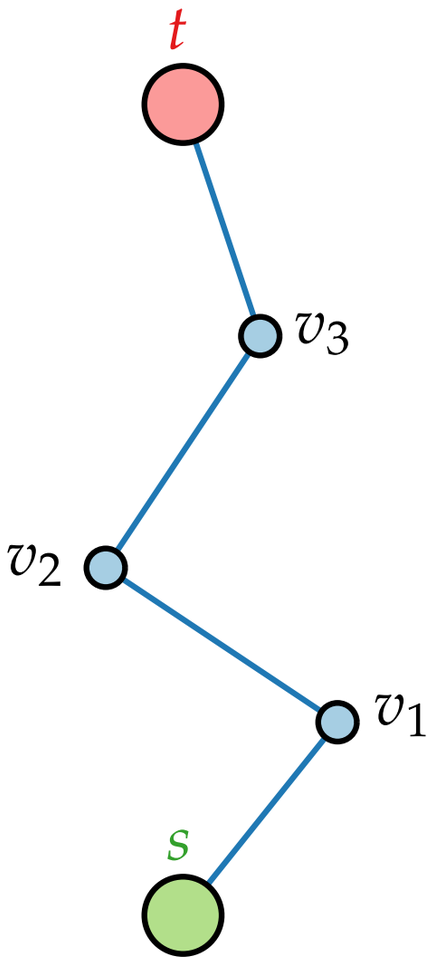


$$w'(P) = w'(v_3,t) + w'(v_2,v_3) + w'(v_1,v_2) + w'(s,v_1)$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

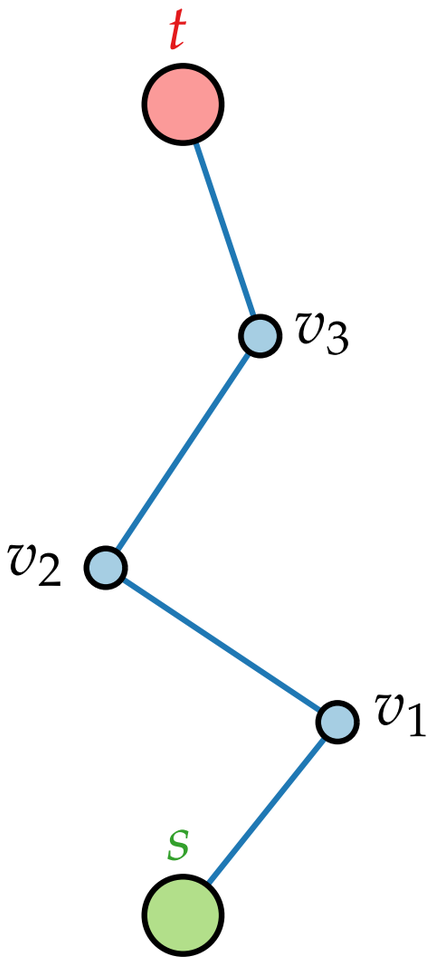


$$\begin{aligned} w'(P) &= \\ & w'(v_3,t) \\ & + \\ & w'(v_2,v_3) \\ & + \\ & w'(v_1,v_2) \\ & + \\ & w'(s,v_1) \end{aligned} \quad \begin{aligned} &= w(s,v_1) \\ &+\delta^*(v_1,t) \\ &-\delta^*(s,t) \end{aligned}$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

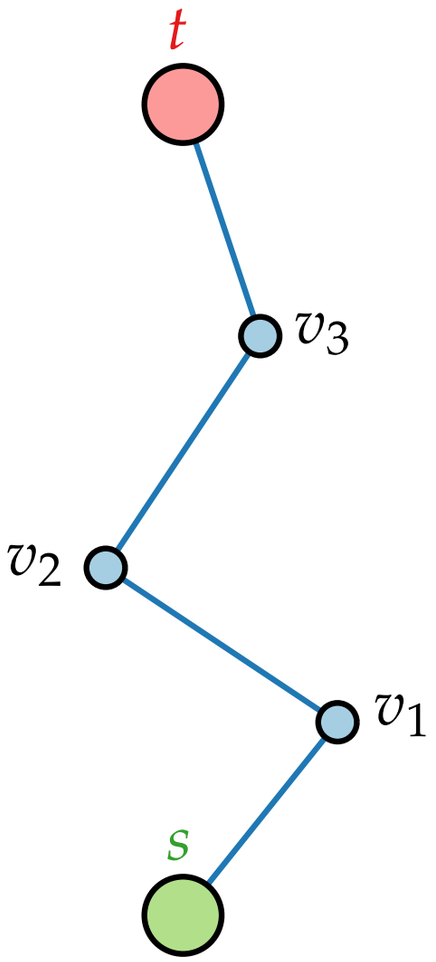


$$
\begin{aligned}
w'(P) &= \\
w'(v_3,t) &+ \\
w'(v_2,v_3) &+ \\
w'(v_1,v_2) &= w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t) \\
w'(s,v_1) &= w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

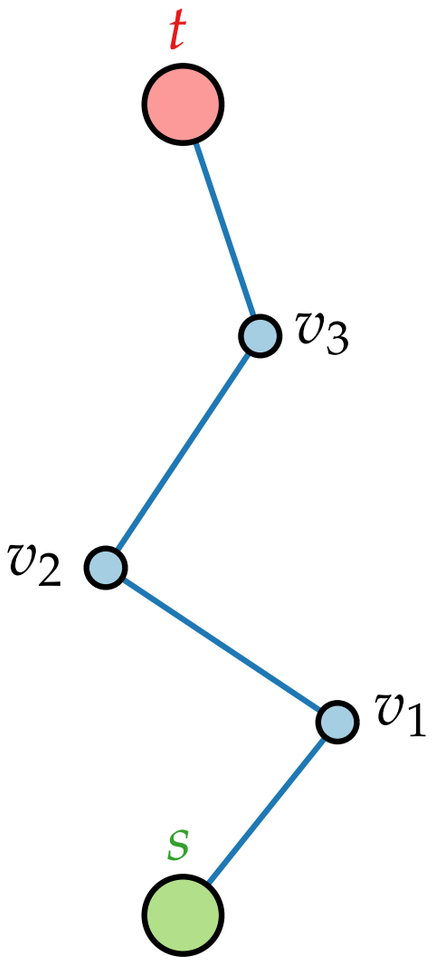


kürzester s–t–Pfad P

$$
\begin{aligned}
& w'(P) \\
&\quad = \\
& w'(v_3,t) \\
&\quad + \\
& w'(v_2,v_3) = w(v_2,v_3) + \delta^*(v_3,t) - \delta^*(v_2,t) \\
&\quad + \\
& w'(v_1,v_2) = w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t) \\
&\quad + \\
& w'(s,v_1) = w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

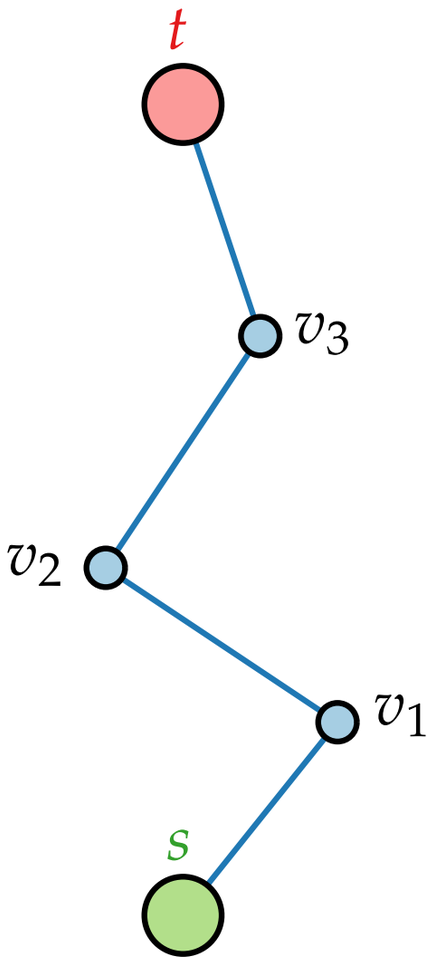


kürzester s–t–Pfad P

$$
\begin{aligned}
w'(P) &\\
&= \\
w'(v_3,t) &= w(v_3,t) + \delta^*(t,t) - \delta^*(v_3,t)\\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) + \delta^*(v_3,t) - \delta^*(v_2,t)\\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t)\\
&+ \\
w'(s,v_1) &= w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

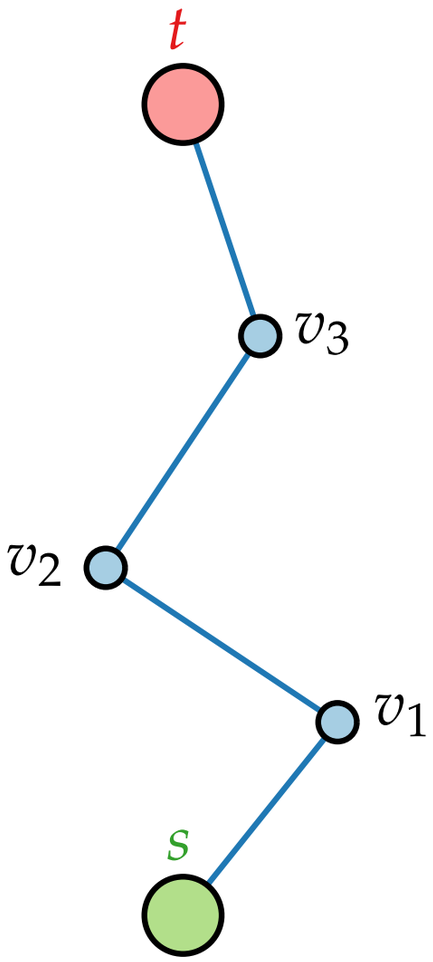


kürzester s–t–Pfad P

$$
\begin{aligned}
w'(P) &\\
&= \\
w'(v_3,t) &= w(v_3,t) + \delta^*(t,t) - \delta^*(v_3,t) \\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) + \delta^*(v_3,t) - \delta^*(v_2,t) \\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t) \\
&+ \\
w'(s,v_1) &= w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

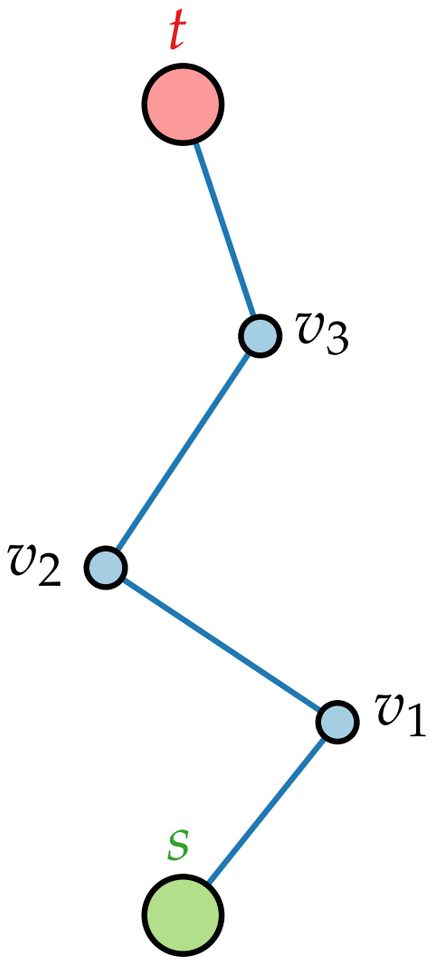


$$
\begin{aligned}
w'(P) &\\
&= \\
w'(v_3,t) &= w(v_3,t) + \delta^*(t,t) - \delta^*(v_3,t) \\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) + \delta^*(v_3,t) - \delta^*(v_2,t) \\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t) \\
&+ \\
w'(s,v_1) &= w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

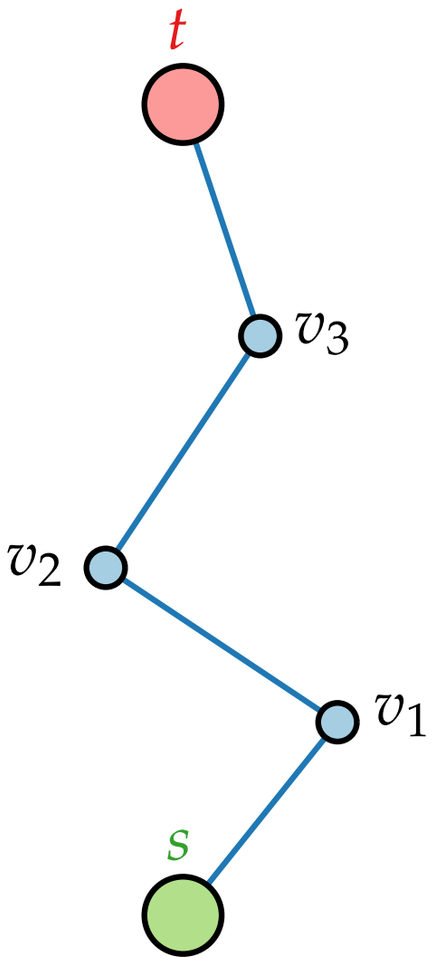


kürzester s–t–Pfad P

$$
\begin{aligned}
w'(P) &\\
&= \\
w'(v_3,t) &= w(v_3,t) + \delta^*(t,t) - \delta^*(v_3,t) \\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) + \delta^*(v_3,t) - \delta^*(v_2,t) \\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t) \\
&+ \\
w'(s,v_1) &= w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

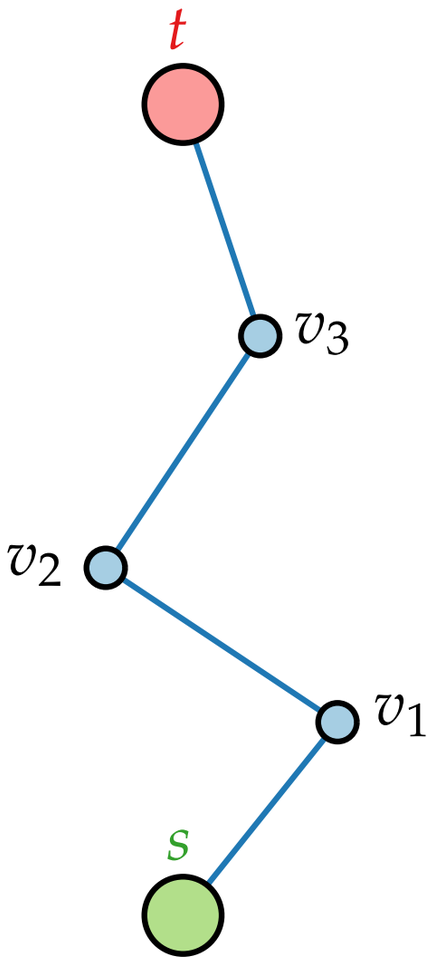


$$
\begin{aligned}
w'(P) &= w(P) \\
&= \\
w'(v_3,t) &= w(v_3,t) + \delta^*(t,t) - \delta^*(v_3,t) \\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) + \delta^*(v_3,t) - \delta^*(v_2,t) \\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t) \\
&+ \\
w'(s,v_1) &= w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

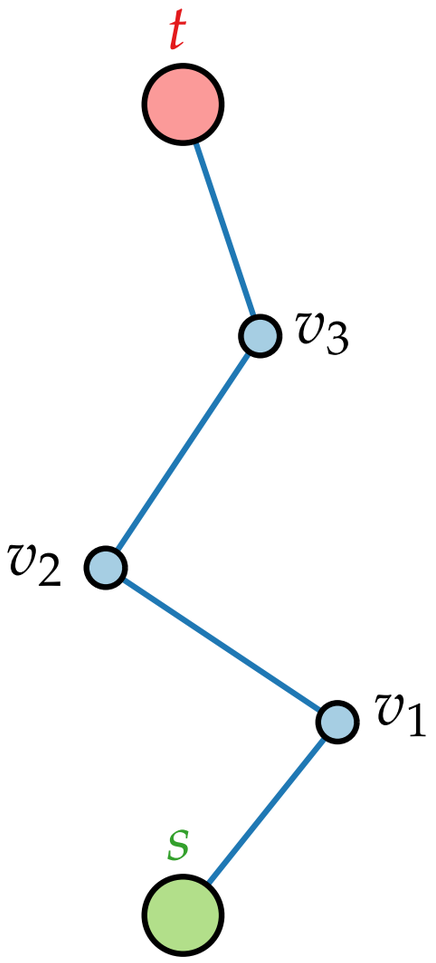


$$
\begin{aligned}
w'(P) &= w(P) + \delta^*(t,t) \\
&= \\
w'(v_3,t) &= w(v_3,t) + \delta^*(t,t) - \delta^*(v_3,t) \\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) + \delta^*(v_3,t) - \delta^*(v_2,t) \\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t) \\
&+ \\
w'(s,v_1) &= w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

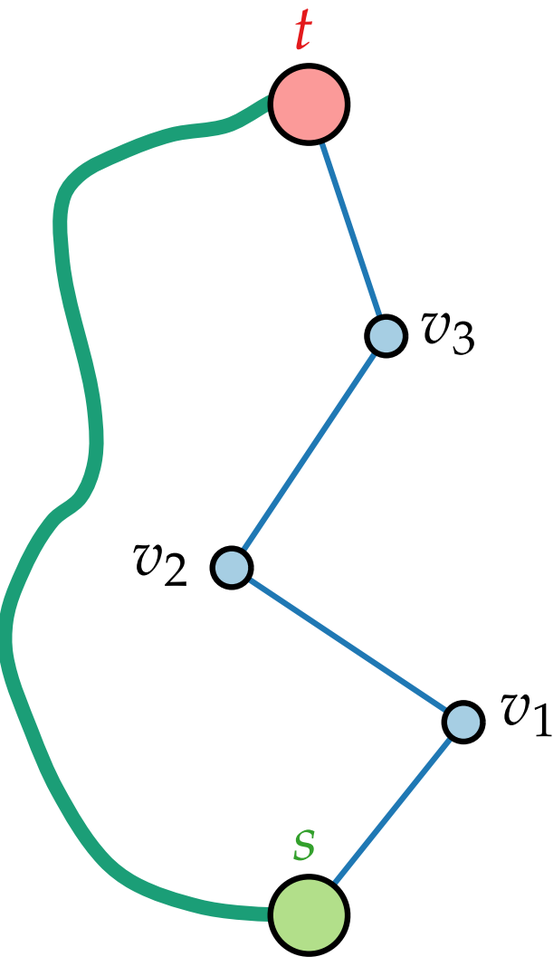


$$
\begin{aligned}
w'(P) &= w(P) + \delta^*(t,t) - \delta^*(s,t) \\
&= \\
w'(v_3,t) &= w(v_3,t) + \delta^*(t,t) - \delta^*(v_3,t) \\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) + \delta^*(v_3,t) - \delta^*(v_2,t) \\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) + \delta^*(v_2,t) - \delta^*(v_1,t) \\
&+ \\
w'(s,v_1) &= w(s,v_1) + \delta^*(v_1,t) - \delta^*(s,t)
\end{aligned}
$$

kürzester s–t–Pfad P

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$



$$
\begin{aligned}
w'(P) &= w(P) && +\delta^*(t,t) && -\delta^*(s,t) \\
&= \\
w'(v_3,t) &= w(v_3,t) && +\delta^*(t,t) && -\delta^*(v_3,t) \\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) && +\delta^*(v_3,t) && -\delta^*(v_2,t) \\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) && +\delta^*(v_2,t) && -\delta^*(v_1,t) \\
&+ \\
w'(s,v_1) &= w(s,v_1) && +\delta^*(v_1,t) && -\delta^*(s,t)
\end{aligned}
$$

kürzester s–t–Pfad P

anderer s–t–Pfad P'

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

$$w'(P') = w(P') + \delta^*(t,t) - \delta^*(s,t)$$

$$w'(P) = w(P) + \delta^*(t,t) - \delta^*(s,t)$$

$$=$$

$$w'(v_3,t) = w(v_3,t) + \delta^*(t,t) - \cancel{\delta^*(v_3,t)}$$

$$+$$

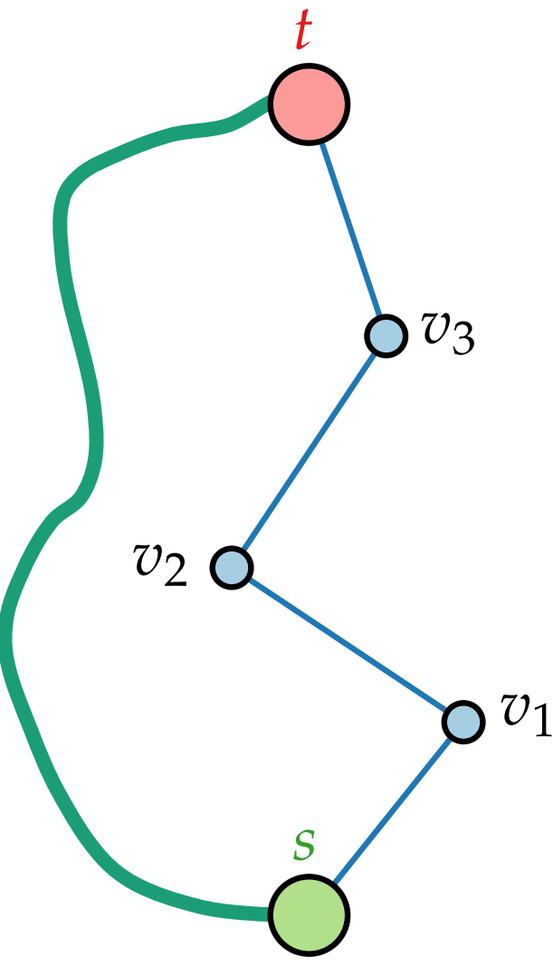
$$w'(v_2,v_3) = w(v_2,v_3) + \cancel{\delta^*(v_3,t)} - \cancel{\delta^*(v_2,t)}$$

$$+$$

$$w'(v_1,v_2) = w(v_1,v_2) + \cancel{\delta^*(v_2,t)} - \cancel{\delta^*(v_1,t)}$$

$$+$$

$$w'(s,v_1) = w(s,v_1) + \cancel{\delta^*(v_1,t)} - \delta^*(s,t)$$

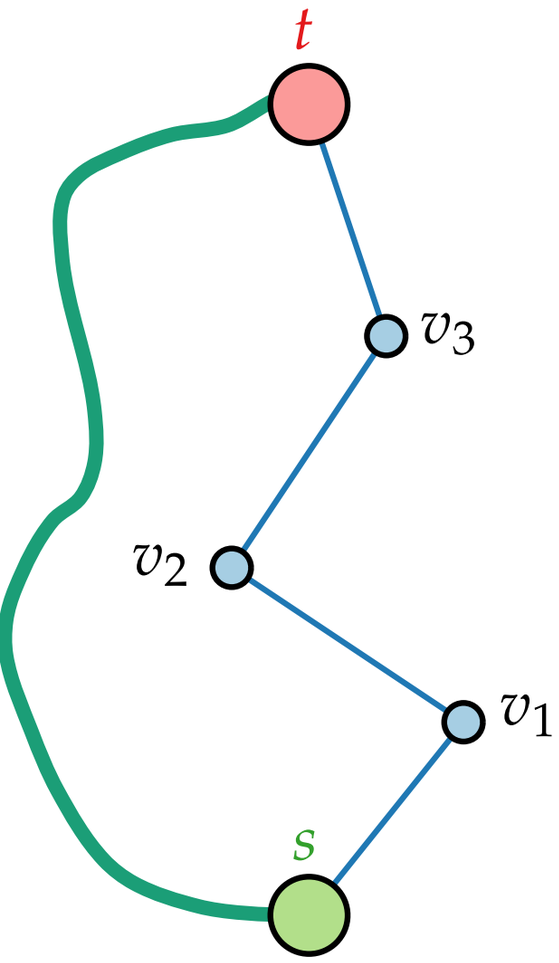


kürzester s–t–Pfad P

anderer s–t–Pfad P'

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$



$$
\begin{aligned}
w'(P') &= w(P') &&+\delta^*(t,t) &&-\delta^*(s,t)\\
w'(P) &= w(P) &&+\delta^*(t,t) &&-\delta^*(s,t)\\
&= \\
w'(v_3,t) &= w(v_3,t) &&+\delta^*(t,t) &&-\delta^*(v_3,t)\\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) &&+\delta^*(v_3,t) &&-\delta^*(v_2,t)\\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) &&+\delta^*(v_2,t) &&-\delta^*(v_1,t)\\
&+ \\
w'(s,v_1) &= w(s,v_1) &&+\delta^*(v_1,t) &&-\delta^*(s,t)
\end{aligned}
$$

kürzester s–t–Pfad P

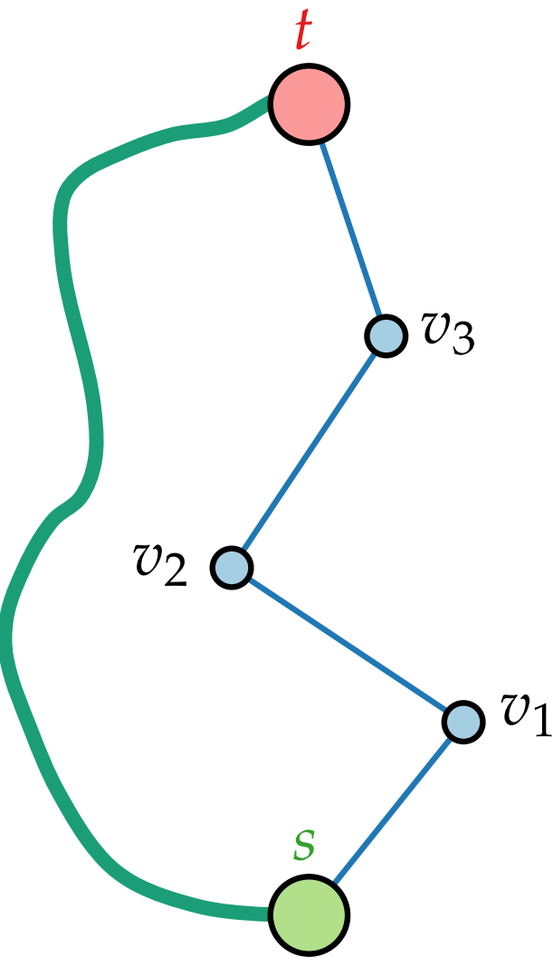
anderer s–t–Pfad P'

Lemma. Ein Pfad P ist optimal bzgl. $w' \Leftrightarrow P$ ist optimal bzgl. w

A* – Korrektheit

$$w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$$

$$\begin{aligned}
w'(P') &= w(P') &&+\delta^*(t,t) &&-\delta^*(s,t)\\
w'(P) &= w(P) &&+\delta^*(t,t) &&-\delta^*(s,t)\\
&= \\
w'(v_3,t) &= w(v_3,t) &&+\delta^*(t,t) &&-\delta^*(v_3,t)\\
&+ \\
w'(v_2,v_3) &= w(v_2,v_3) &&+\delta^*(v_3,t) &&-\delta^*(v_2,t)\\
&+ \\
w'(v_1,v_2) &= w(v_1,v_2) &&+\delta^*(v_2,t) &&-\delta^*(v_1,t)\\
&+ \\
w' &
\end{aligned}$$



kürzester s–t–Pfad P

anderer s–t–Pfad P'

Satz. Dijkstra berechnet in einem Graphen $G = (V, E; w)$ mit $w\colon E \to \mathbb{Q}_{\geq 0}$ in $\mathcal{O}(E + V \log V)$ Zeit den kürzesten Weg zwischen zwei Knoten.

Lemma. Ein Pfad P ist optimal bzgl. $w' \Leftrightarrow P$ ist optimal bzgl. w

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Beweis. Betrachte beliebigen Knoten $u \in V$.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Beweis. Betrachte beliebigen Knoten $u \in V$.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Beweis. Betrachte beliebigen Knoten $u \in V$.

Wähle $v = t$.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Beweis. Betrachte beliebigen Knoten $u \in V$.

Wähle $v = t$.

Dann gilt $\delta(u,t) + \delta^*(t,t) \geq \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Beweis. Betrachte beliebigen Knoten $u \in V$.

Wähle $v = t$.

Dann gilt $\delta(u,t) + \delta^*(t,t) \geq \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Beweis. Betrachte beliebigen Knoten $u \in V$.

Wähle $v = t$.

Dann gilt $\delta(u,t) + \delta^*(t,t) \geq \delta^*(u,t)$

$\Rightarrow \delta(u,t) \geq \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Beweis. Betrachte beliebigen Knoten $u \in V$.

Wähle $v = t$.

Dann gilt $\delta(u,t) + \delta^*(t,t) \geq \delta^*(u,t)$

$\Rightarrow \delta(u,t) \geq \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Beweis. Betrachte beliebigen Knoten $u \in V$.

Wähle $v = t$.

Dann gilt $\delta(u,t) + \delta^*(t,t) \geq \delta^*(u,t)$

$\Rightarrow \delta(u,t) \geq \delta^*(u,t)$ □

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.

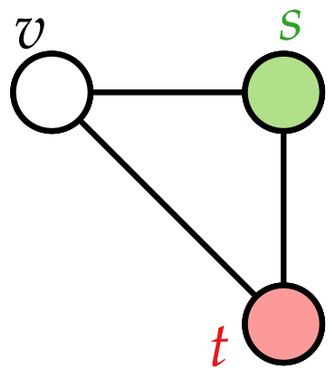
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



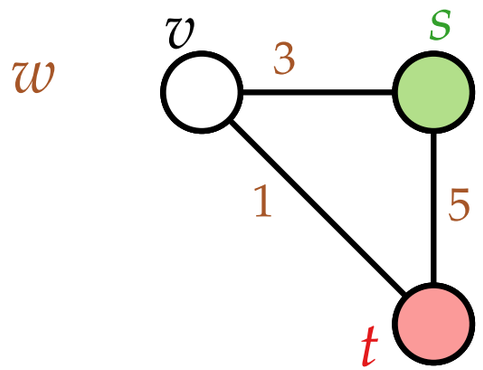
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



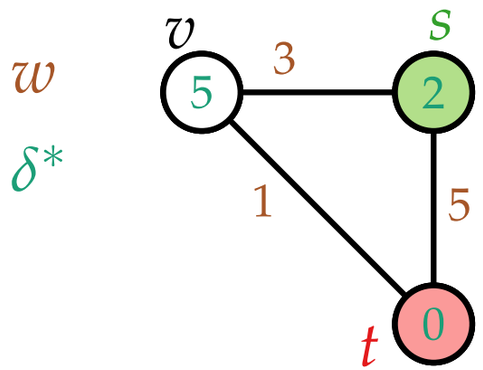
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



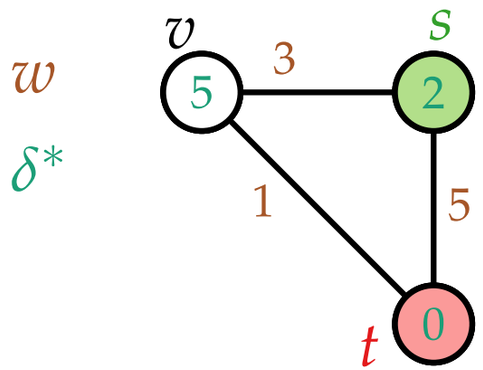
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t)$ $\delta^*(v,t)$

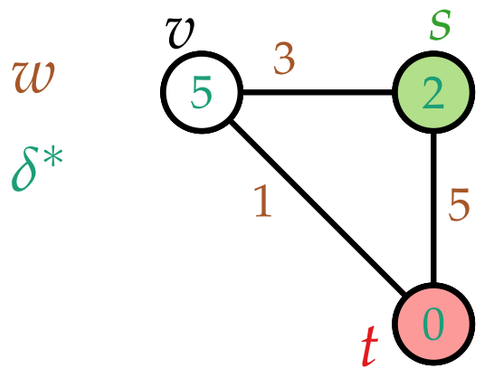
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t)$ $\quad$ $\delta^*(v,t)$

1

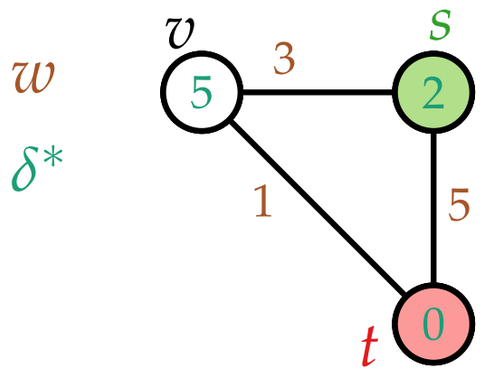
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t)$ $\quad$ $\delta^*(v,t)$

1 $\quad$ 0

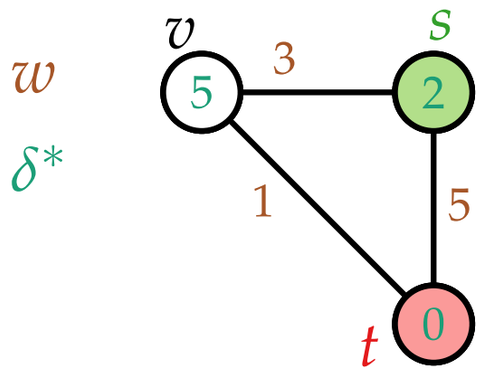
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t)$ $\quad$ $\delta^*(v,t)$

1 $\quad$ 0 $\quad$ 5

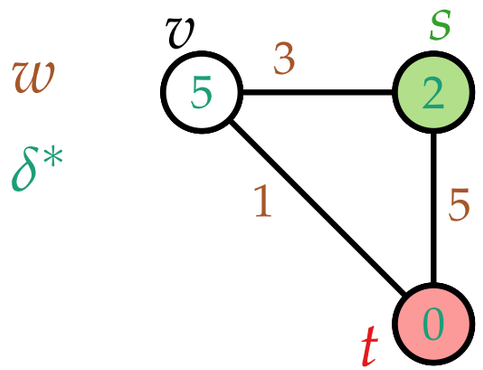
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$$
$$1 \qquad\quad 0 \qquad\quad 5$$

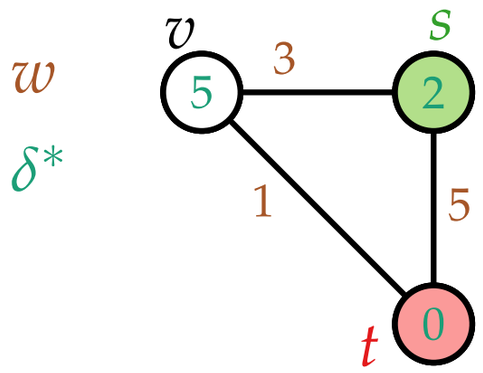
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$$
$$1 \qquad\quad 0 \qquad\quad 5$$

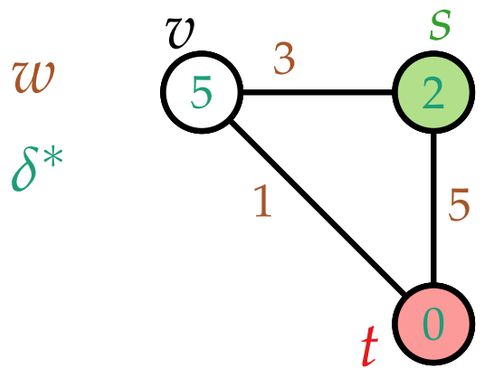
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

$1 \qquad 0 \qquad 5$

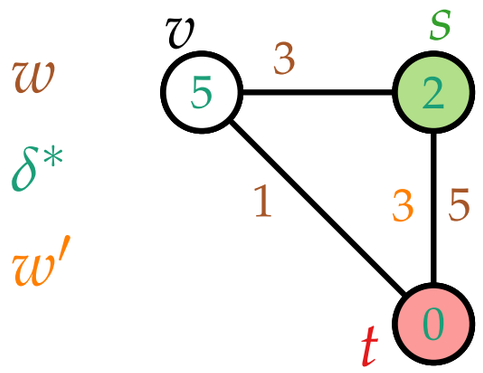
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

$1 \qquad 0 \qquad 5$

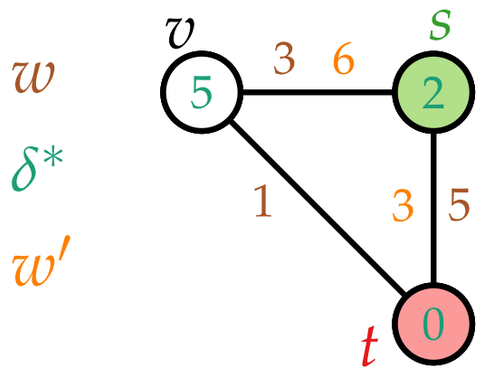
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

$1 \qquad 0 \qquad 5$

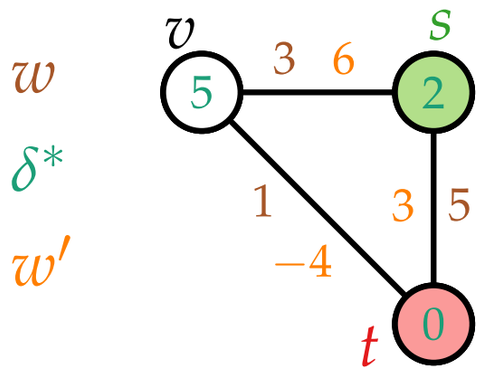
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

$1 \qquad 0 \qquad 5$

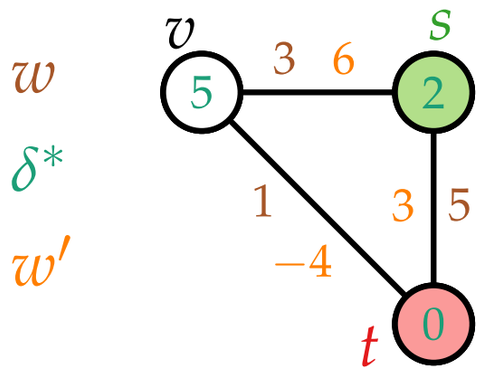
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

$1 \qquad 0 \qquad 5$

Lösung A*:

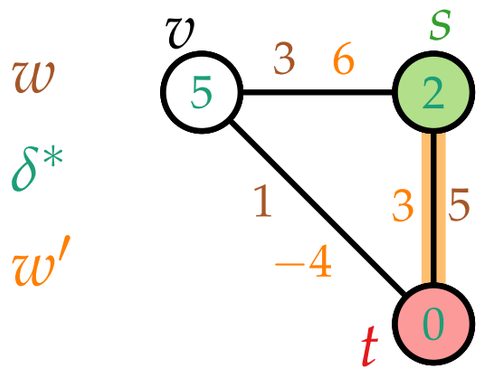
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

$1 \qquad 0 \qquad 5$

Lösung A*:

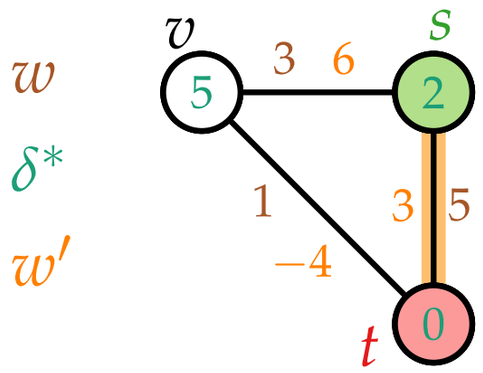
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

$1 \qquad 0 \qquad 5$

Lösung A*: Länge 5

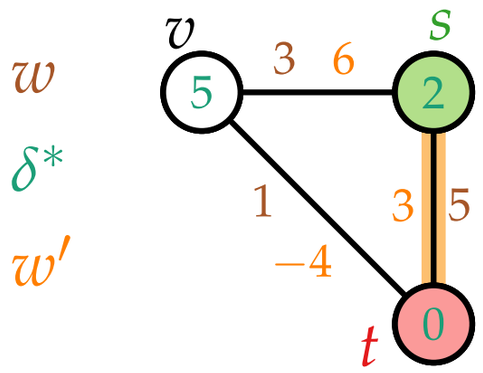
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

1 0 5

Lösung A*: Länge 5

Kürzester Pfad:

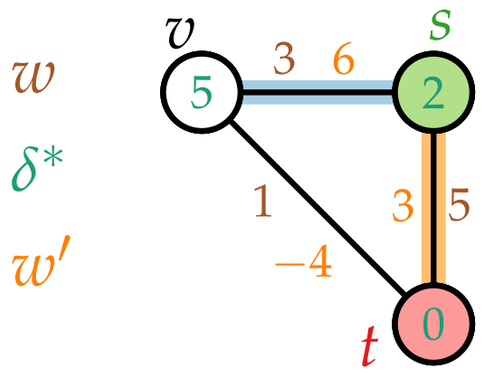
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

1 0 5

Lösung A*: Länge 5

Kürzester Pfad:

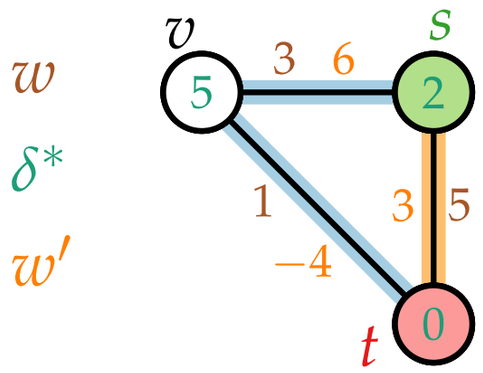
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

1 0 5

Lösung A*: Länge 5

Kürzester Pfad:

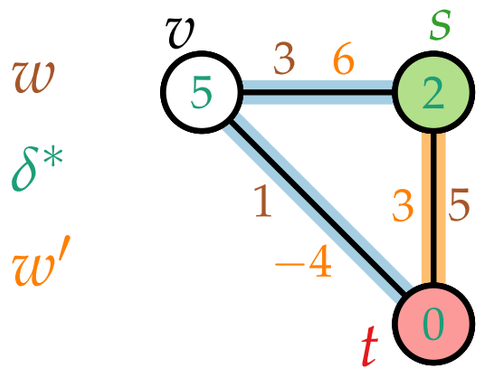
Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u, v \in V$.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Beweis.



$\delta(v,t) + \delta^*(t,t) < \delta^*(v,t)$ also nicht **monoton**

1 0 5

Lösung A*: Länge 5

Kürzester Pfad: Länge 4

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Beweis.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Beweis. $w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Beweis. $w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Beweis. $w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t) \geq \delta(u,v) + \delta^*(v,t) - \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Beweis. $w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t) \geq \delta(u,v) + \delta^*(v,t) - \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Beweis. $w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t) \geq \delta(u,v) + \delta^*(v,t) - \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Beweis. $w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t) \geq \delta(u,v) + \delta^*(v,t) - \delta^*(u,t)$

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Beweis. $w'(u,v) = w(u,v) + \delta^*(v,t) - \delta^*(u,t) \geq \delta(u,v) + \delta^*(v,t) - \delta^*(u,t) \geq 0$ □

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Ein Pfad P ist optimal bzgl. w' $\Leftrightarrow$ P ist optimal bzgl. w

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Ein Pfad P ist optimal bzgl. w' $\Leftrightarrow$ P ist optimal bzgl. w

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Satz. Wenn δ^* **monoton** ist, dann berechnet A* in einem Graphen $G = (V, E; w)$ mit $w\colon E \to \mathbb{Q}_{\geq 0}$ in $\mathcal{O}(E + V \log V)$ Zeit den kürzesten Weg zwischen zwei Knoten.

Monotone Schätzfunktion

$\delta^* : V \times V \to \mathbb{R}$ ist eine **optimistische** Schätzfunktion genau dann wenn $\delta(u,t) \geq \delta^*(u,t)$ für alle $u \in V$.

$\delta^* : V \times V \to \mathbb{R}$ ist eine **monotone** Schätzfunktion genau dann wenn $\delta(u,v) + \delta^*(v,t) \geq \delta^*(u,t)$ für alle $u,v \in V$.

Lemma. Ein Pfad P ist optimal bzgl. w' $\Leftrightarrow$ P ist optimal bzgl. w

Lemma. Jede **monotone** Schätzfunktion mit $\delta^*(t,t) = 0$ ist **optimistisch**.

Lemma. Wenn δ^* **nicht monoton** ist, dann kann es sein, dass A* nicht den kürzesten Pfad findet.

Lemma. Wenn δ^* **monoton** ist, dann sind die Kantengewichte w' nicht-negativ.

Satz. Wenn δ^* **monoton** ist, dann berechnet A* in einem Graphen $G = (V, E; w)$ mit $w\colon E \to \mathbb{Q}_{\geq 0}$ in $\mathcal{O}(E + V \log V)$ Zeit den kürzesten Weg zwischen zwei Knoten.

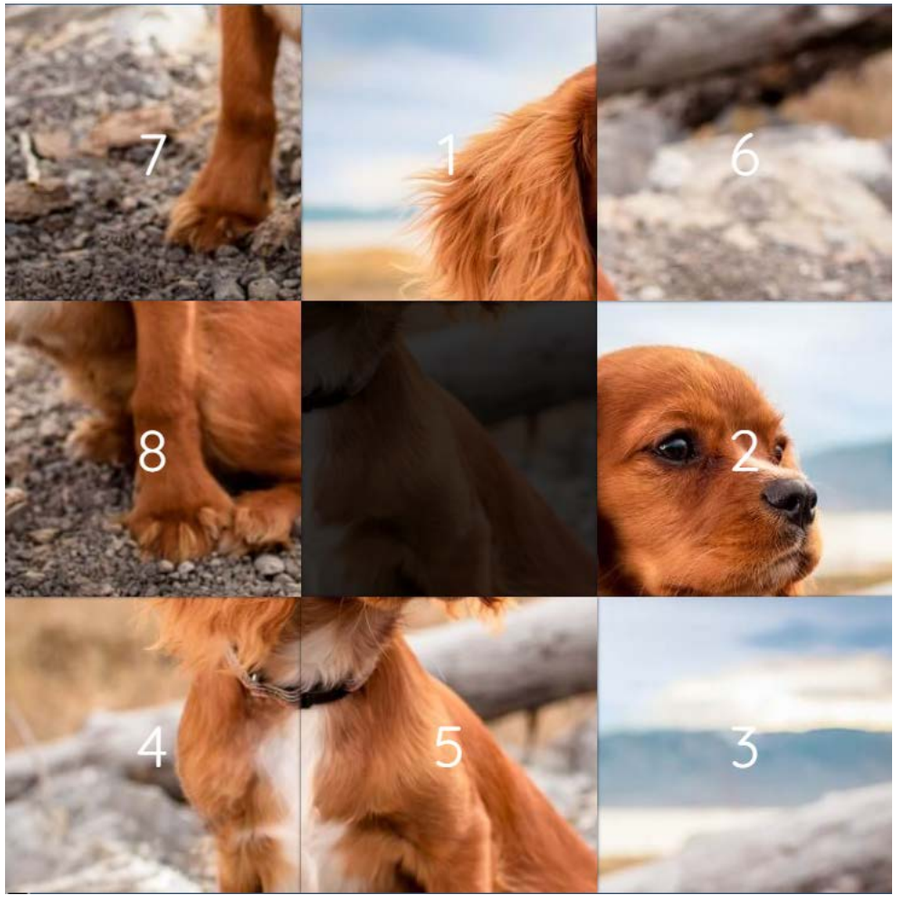
(gilt sogar, wenn δ^* „nur“ **optimistisch** ist.)

Nicht-geometrische Probleme: Schiebepuzzle

Quelle: `https://murhafsousli.github.io/8puzzle`

Nicht-geometrische Probleme: Schiebepuzzle

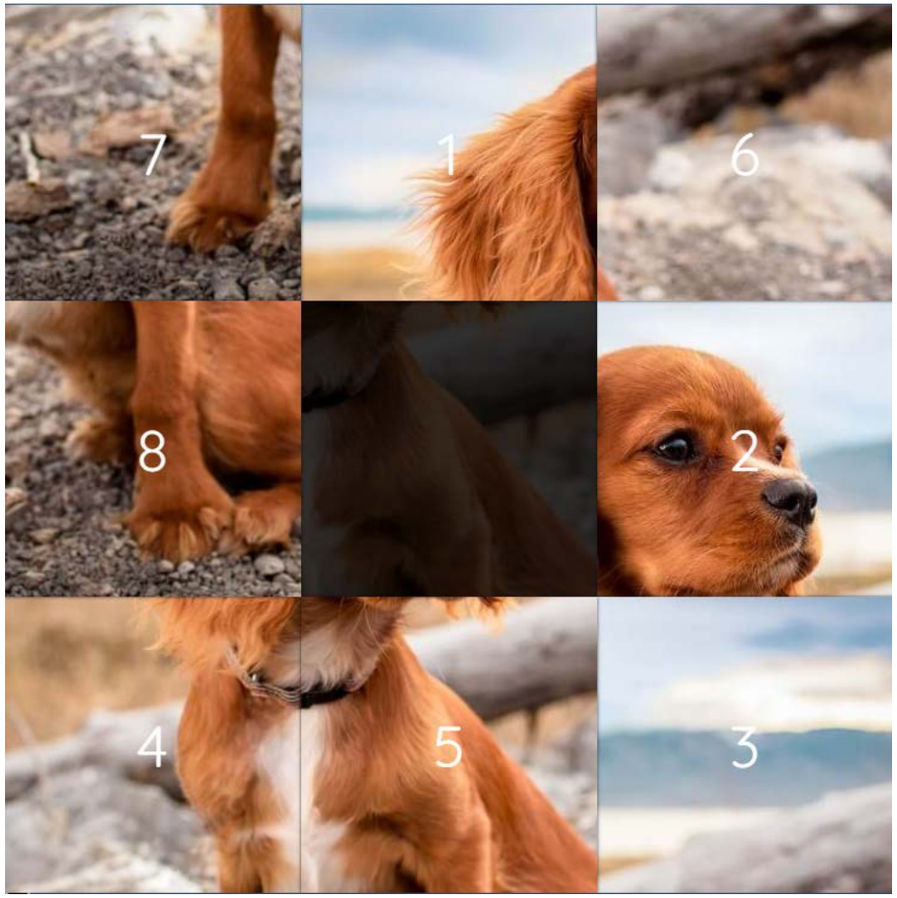
Quelle: `https://murhafsousli.github.io/8puzzle`



Nicht-geometrische Probleme: Schiebepuzzle

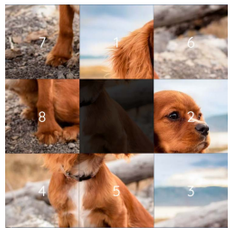
Modellierung als Graph:

Quelle: `https://murhafsousli.github.io/8puzzle`



Nicht-geometrische Probleme: Schiebepuzzle

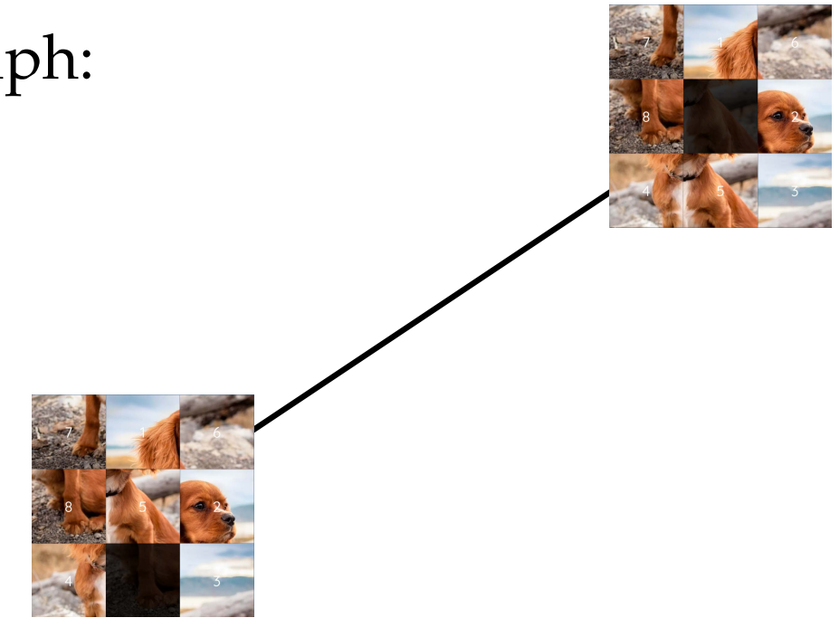
Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

Nicht-geometrische Probleme: Schiebepuzzle

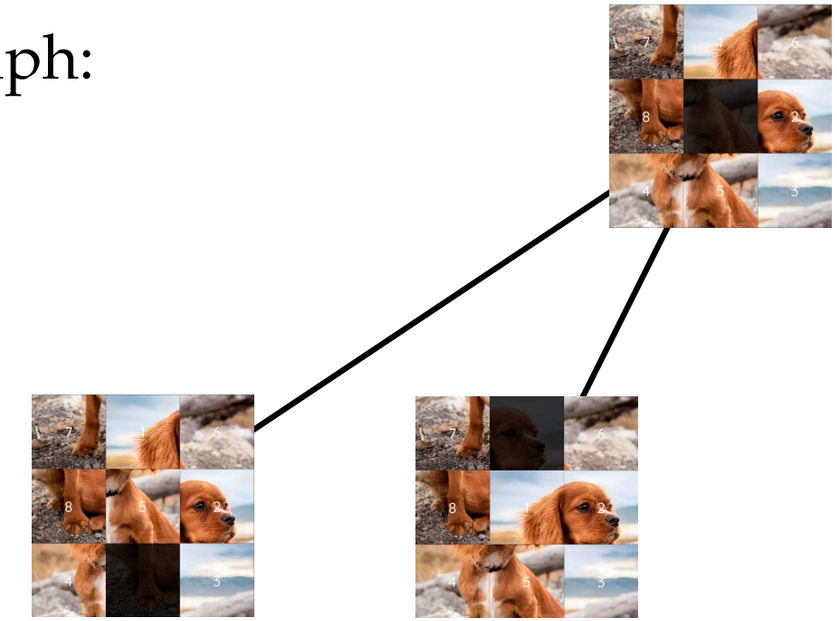
Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

Nicht-geometrische Probleme: Schiebepuzzle

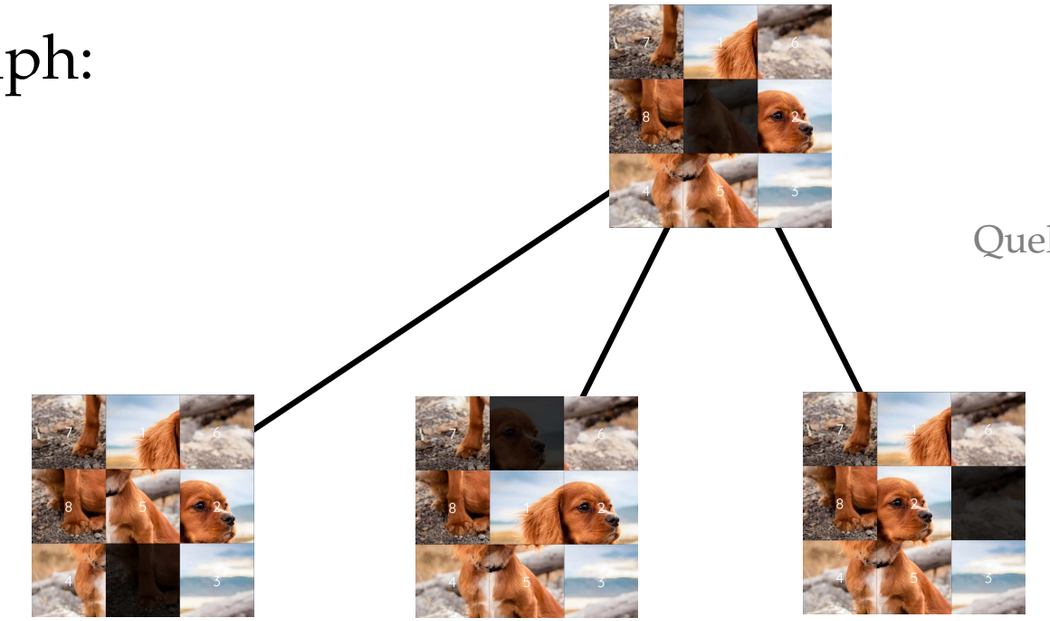
Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

Nicht-geometrische Probleme: Schiebepuzzle

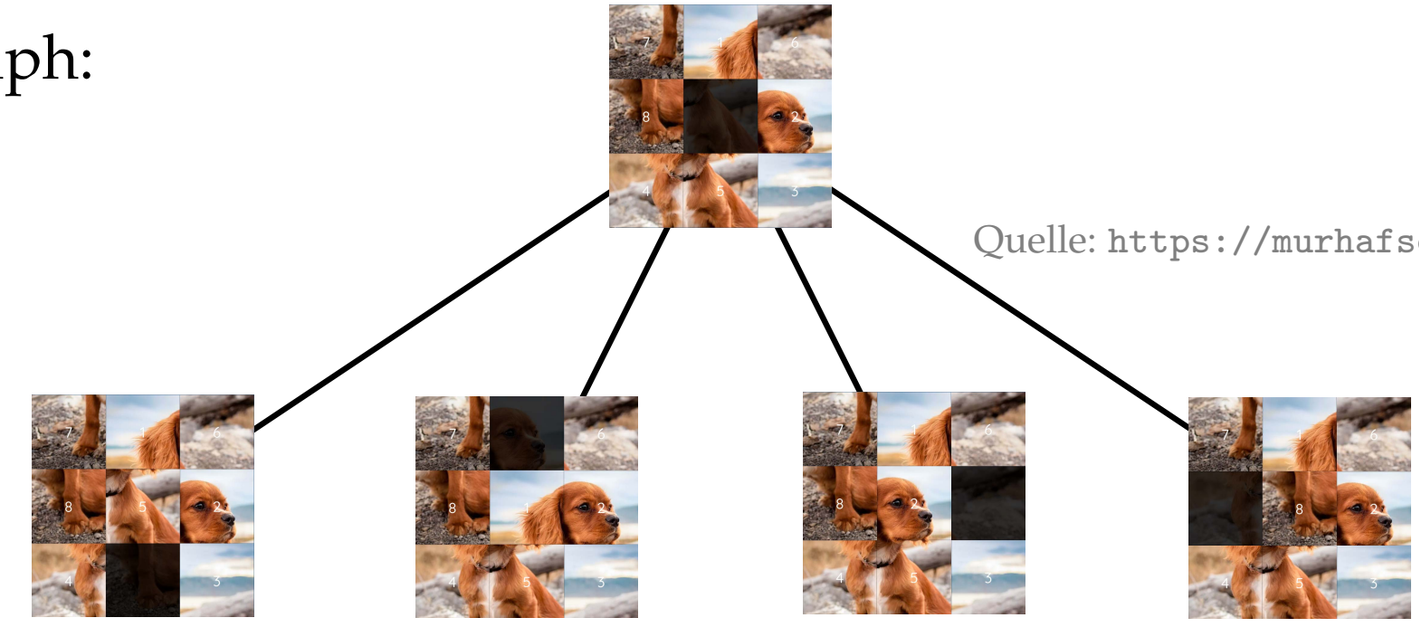
Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

Nicht-geometrische Probleme: Schiebepuzzle

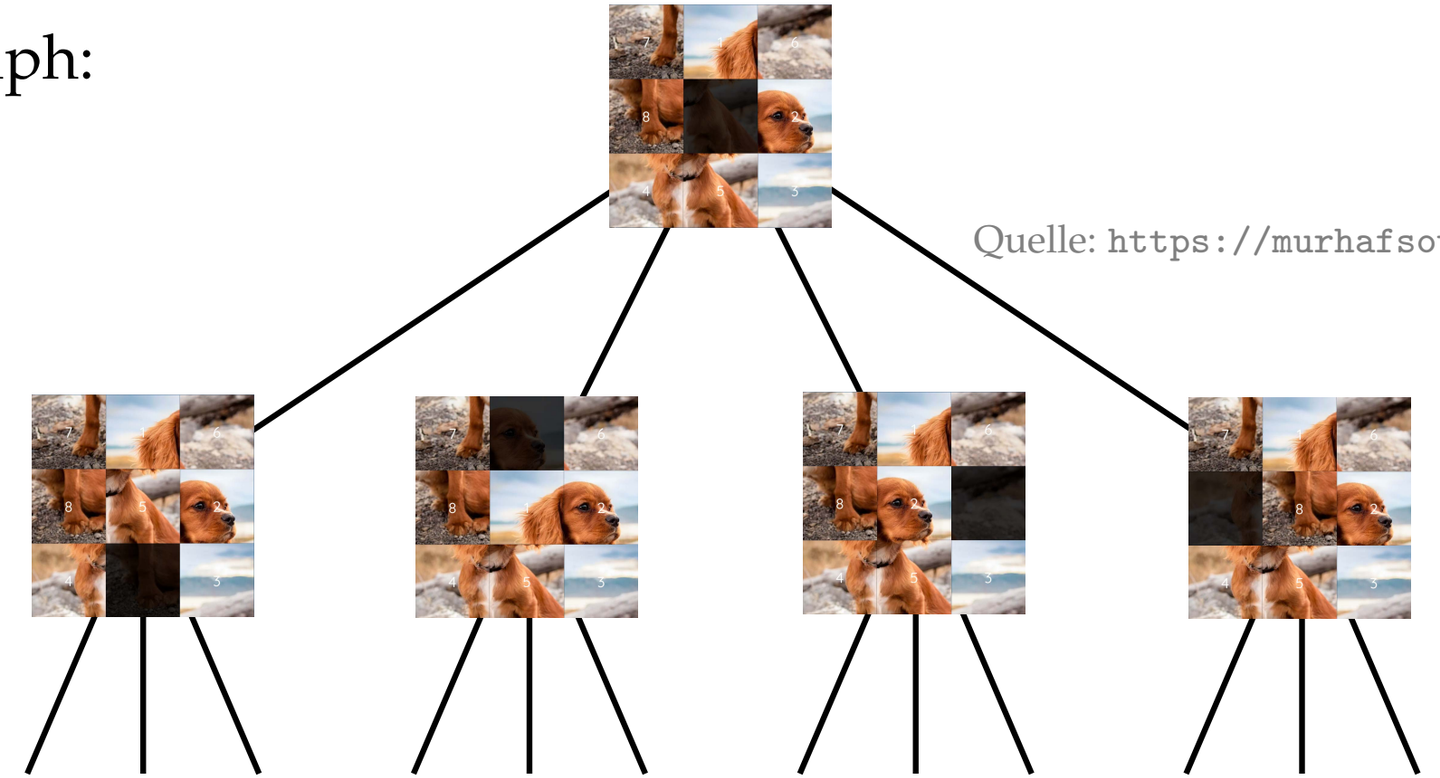
Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

Nicht-geometrische Probleme: Schiebepuzzle

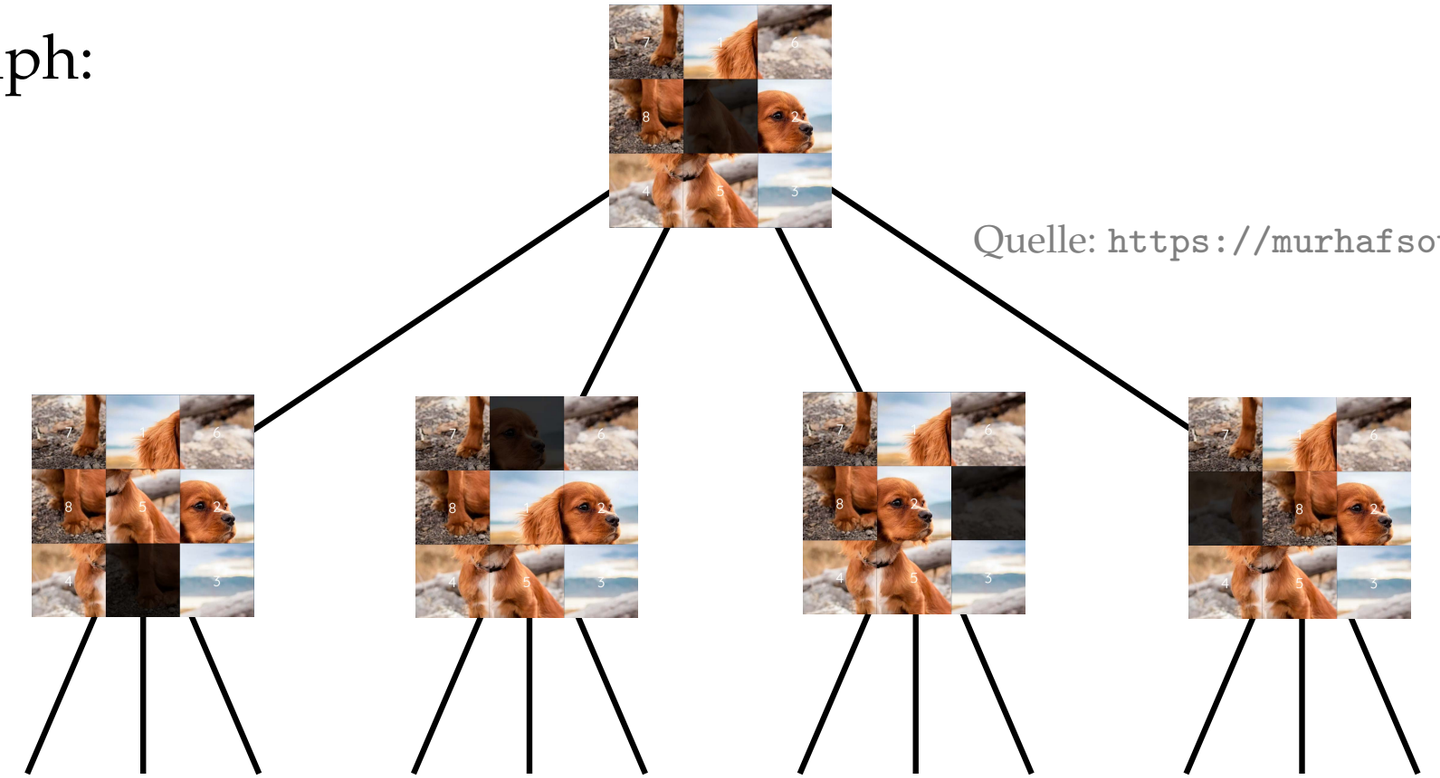
Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:

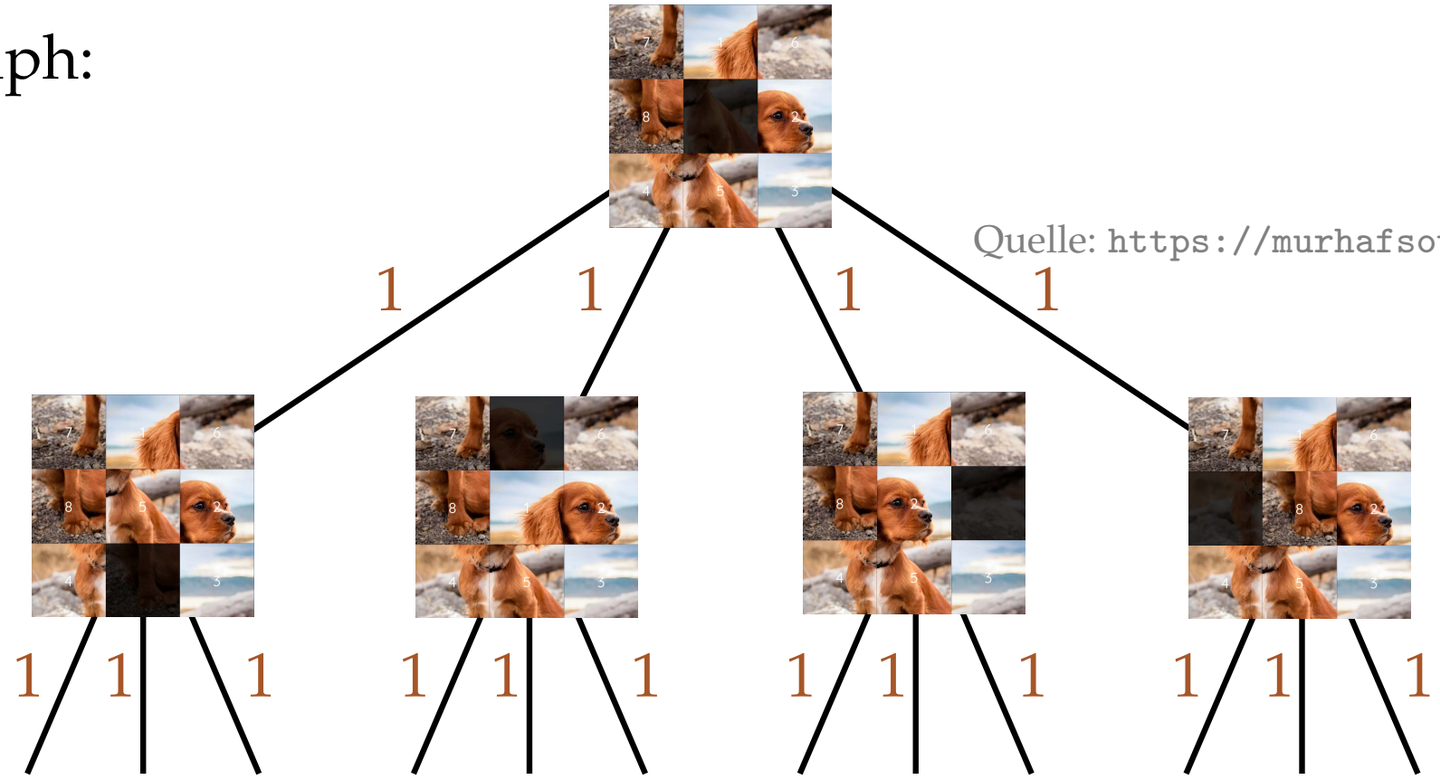


Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:

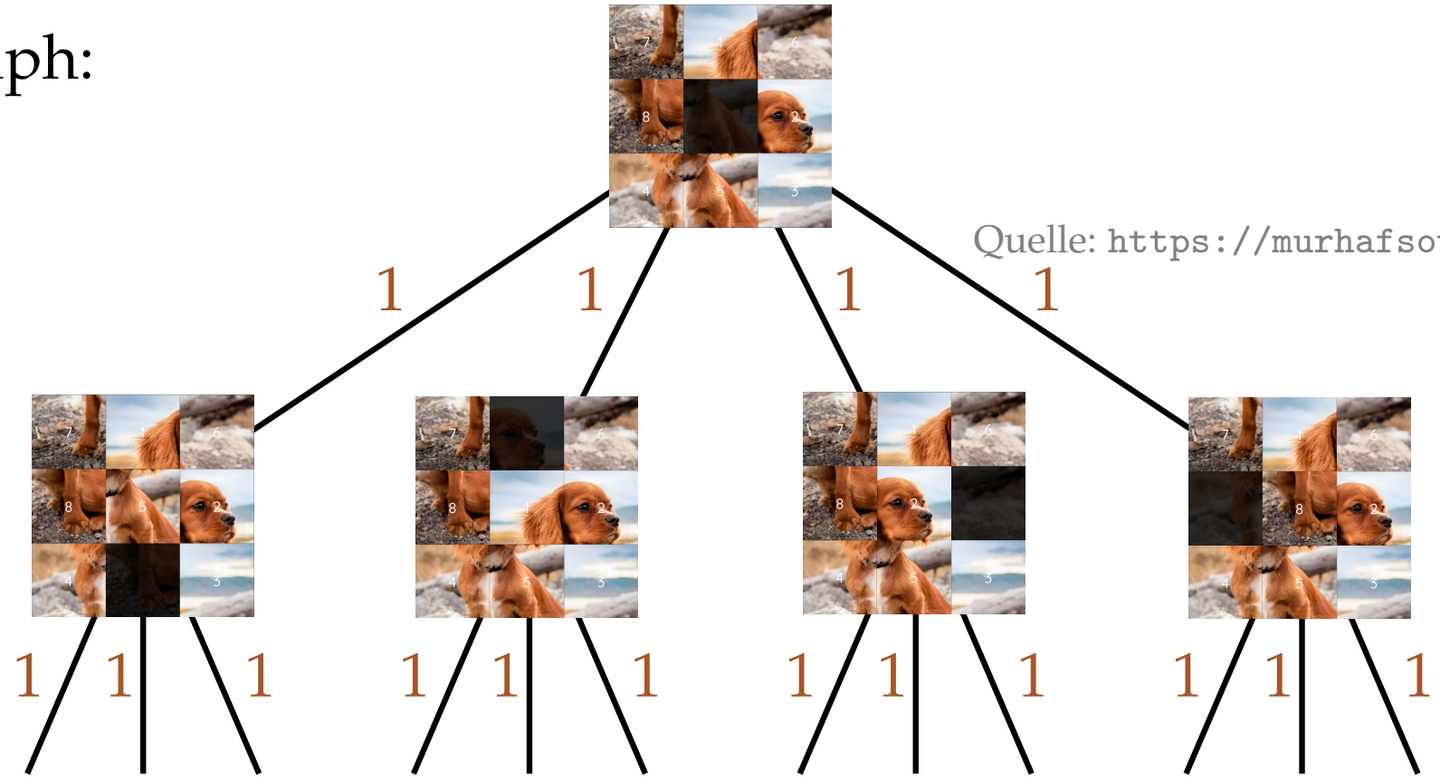


Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



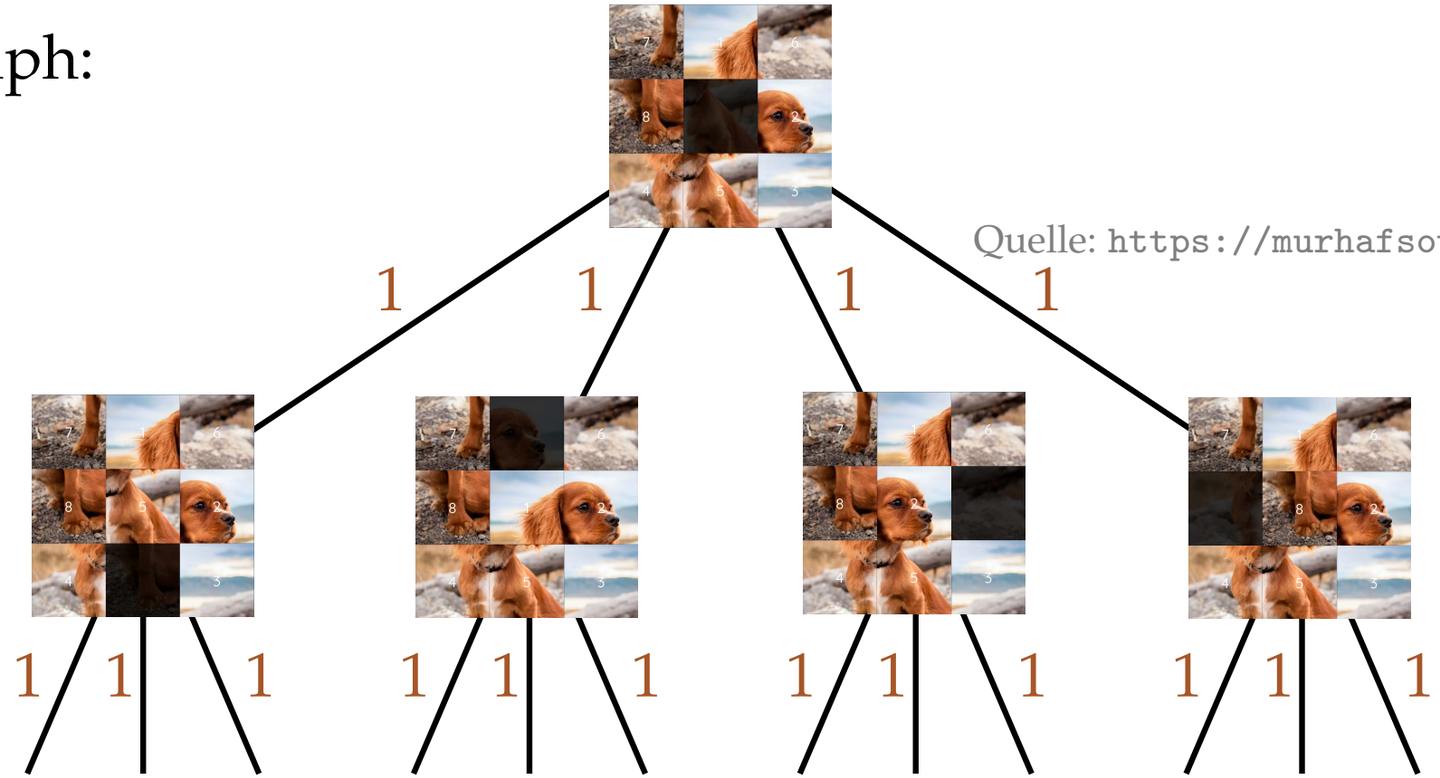
Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



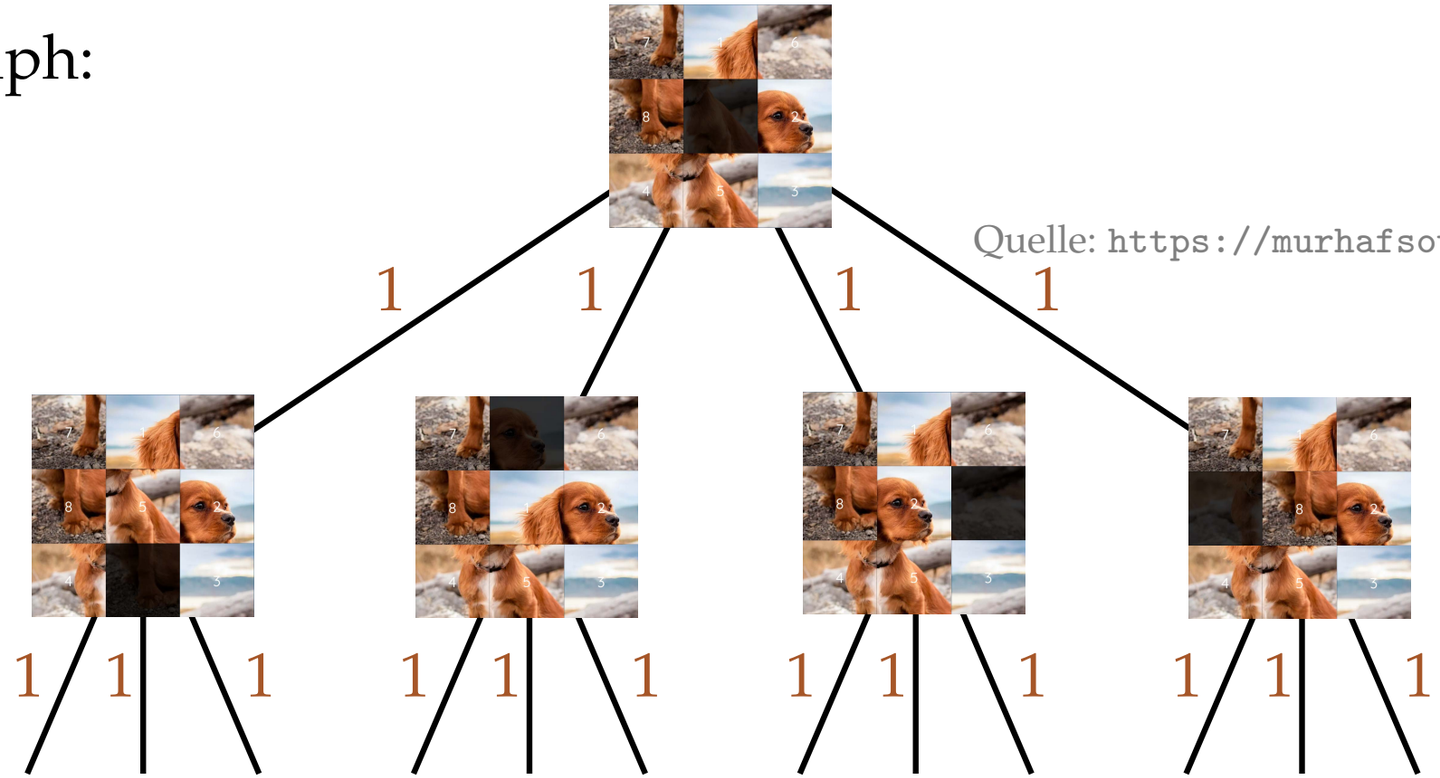
Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra			

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



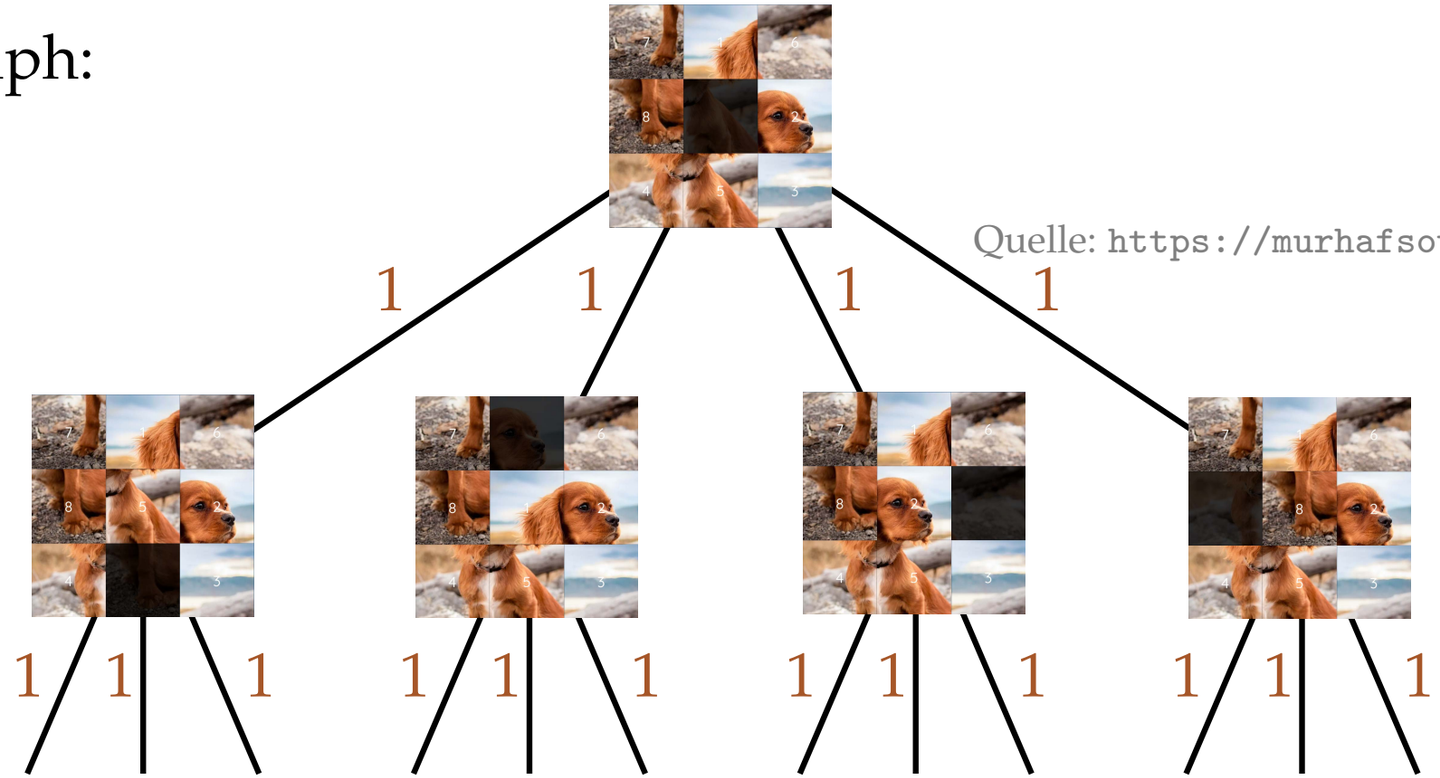
Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112		

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



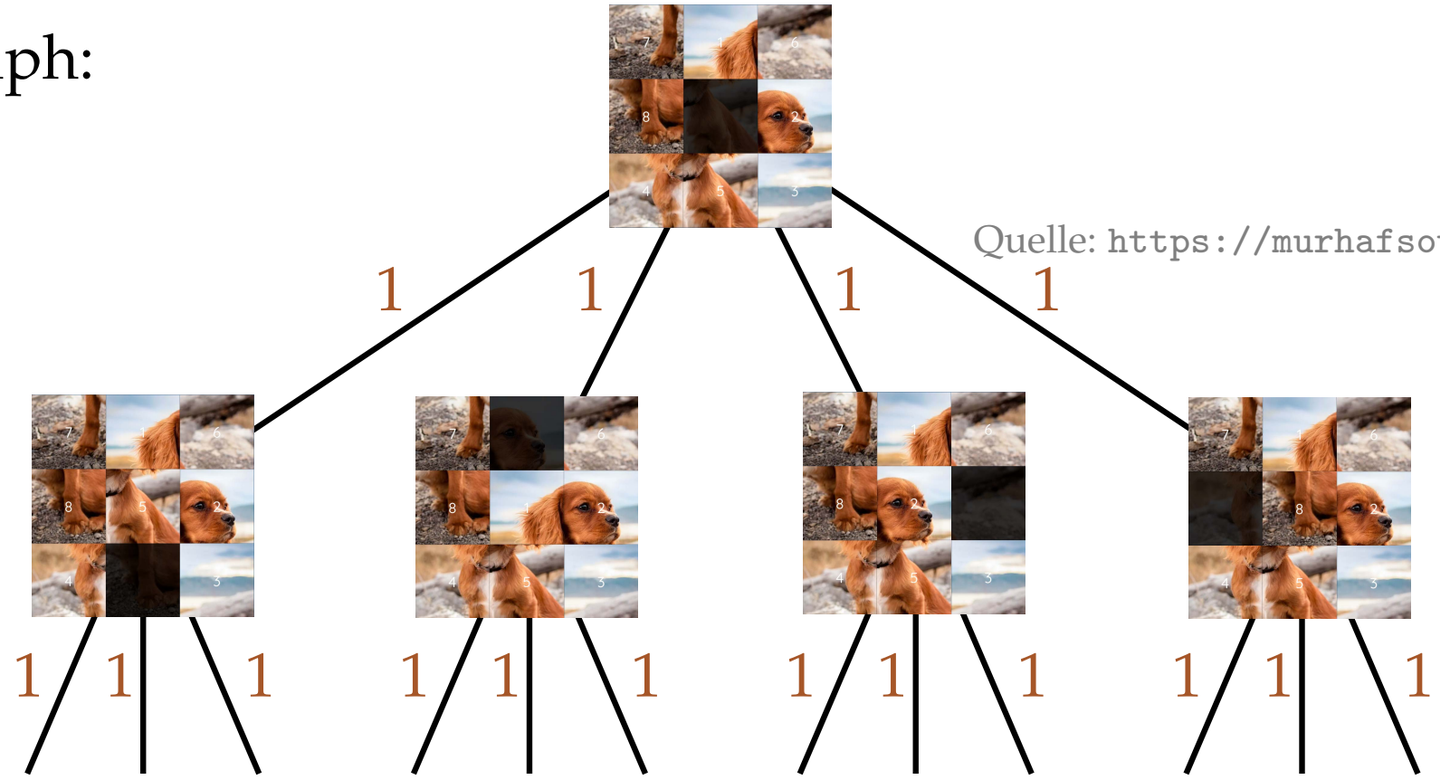
Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112	6 300	

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



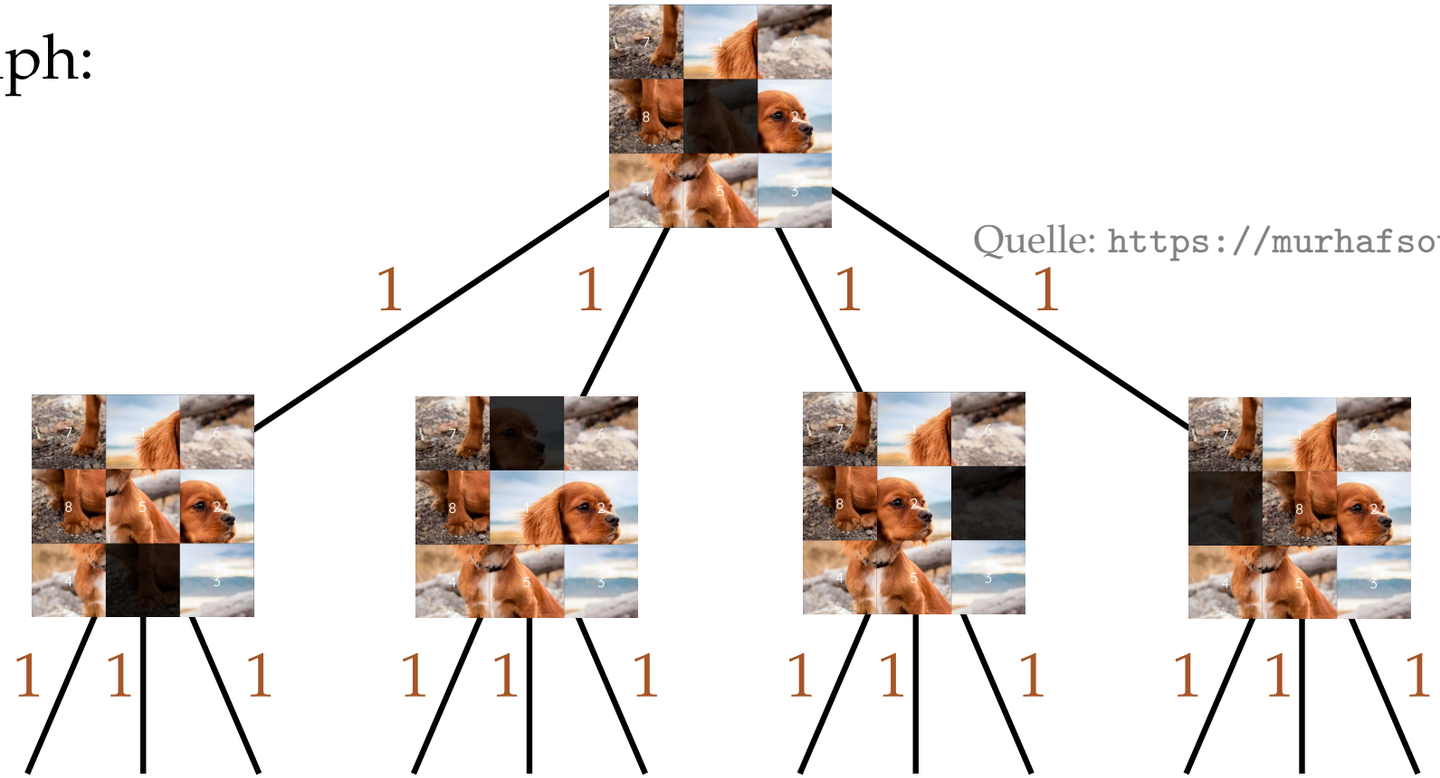
Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112	6 300	3 600 000

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



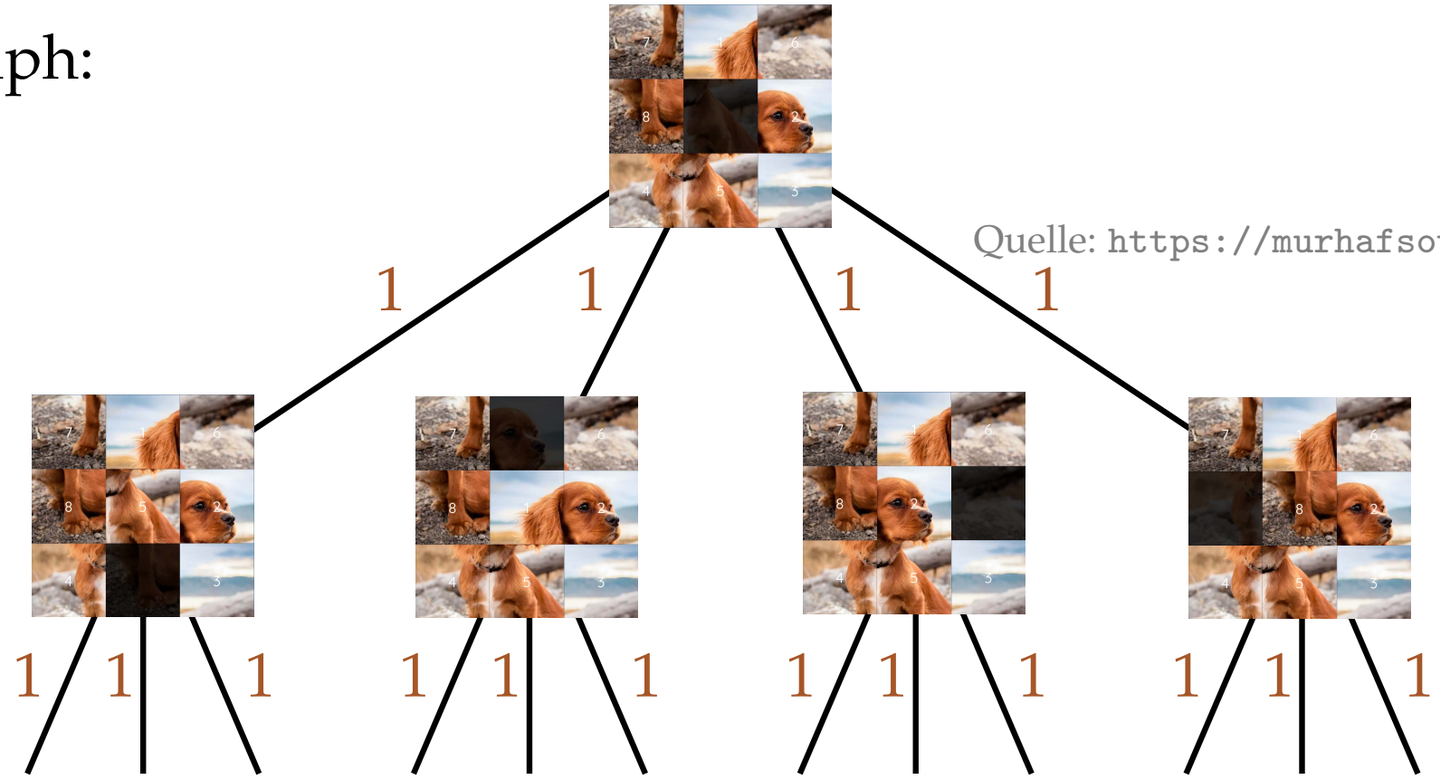
Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112	6 300	3 600 000
A*			

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

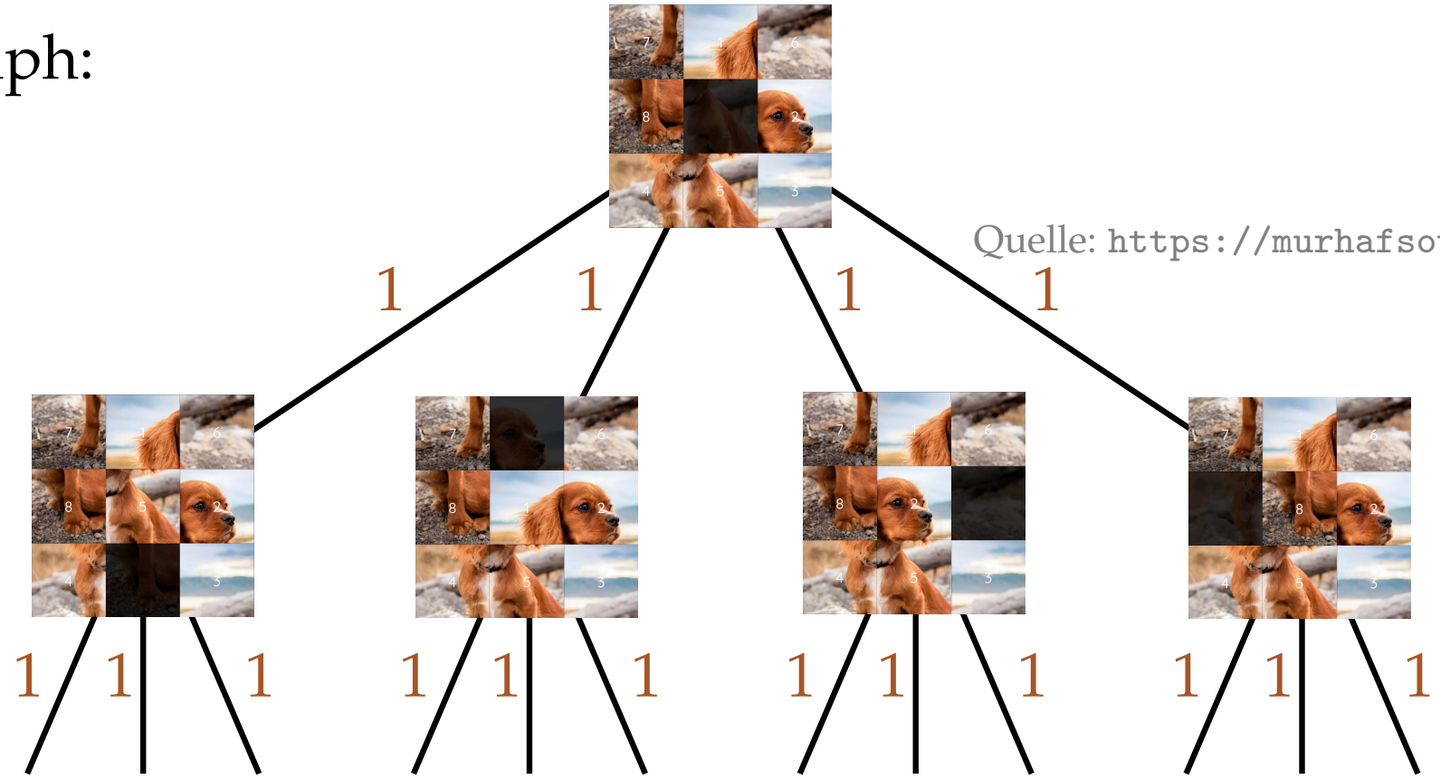
Lösung: kürzester Weg von  nach 

Schätzfunktion?

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112	6 300	3 600 000
A*			

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

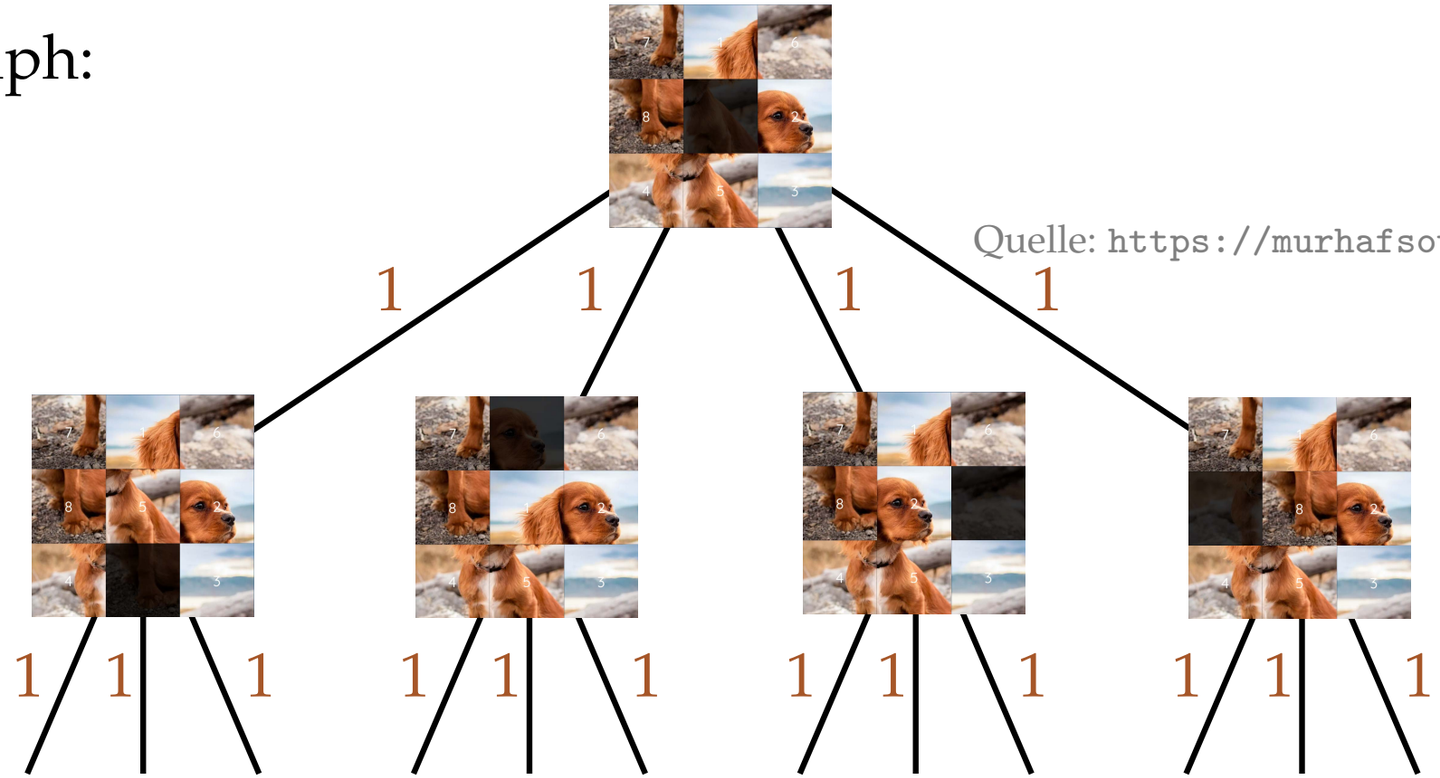
Lösung: kürzester Weg von  nach 

Schätzfunktion?
#Kacheln an falscher Position

	Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:		
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112	6 300	3 600 000
A*			

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

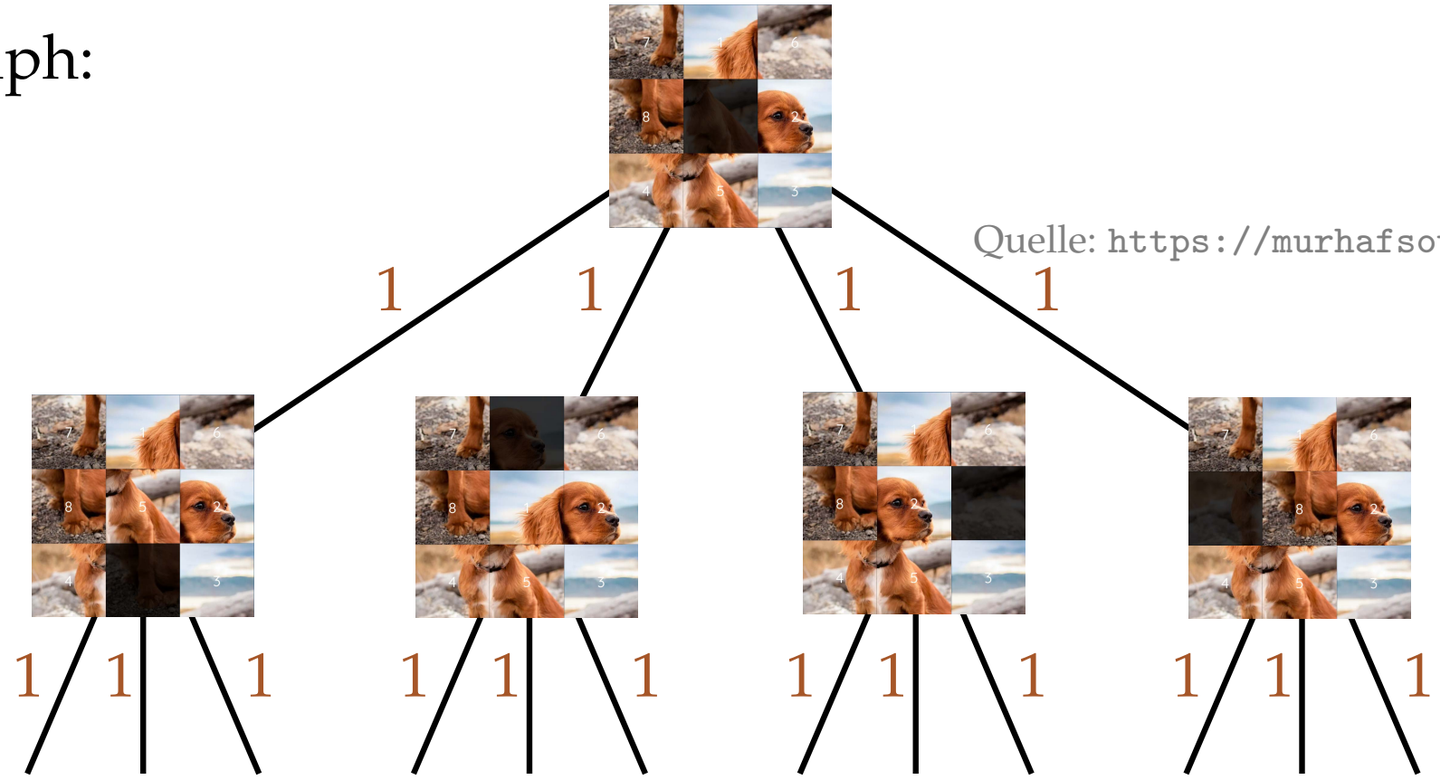
Lösung: kürzester Weg von  nach 

Schätzfunktion?
#Kacheln an falscher Position

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112	6 300	3 600 000
A*	13		

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

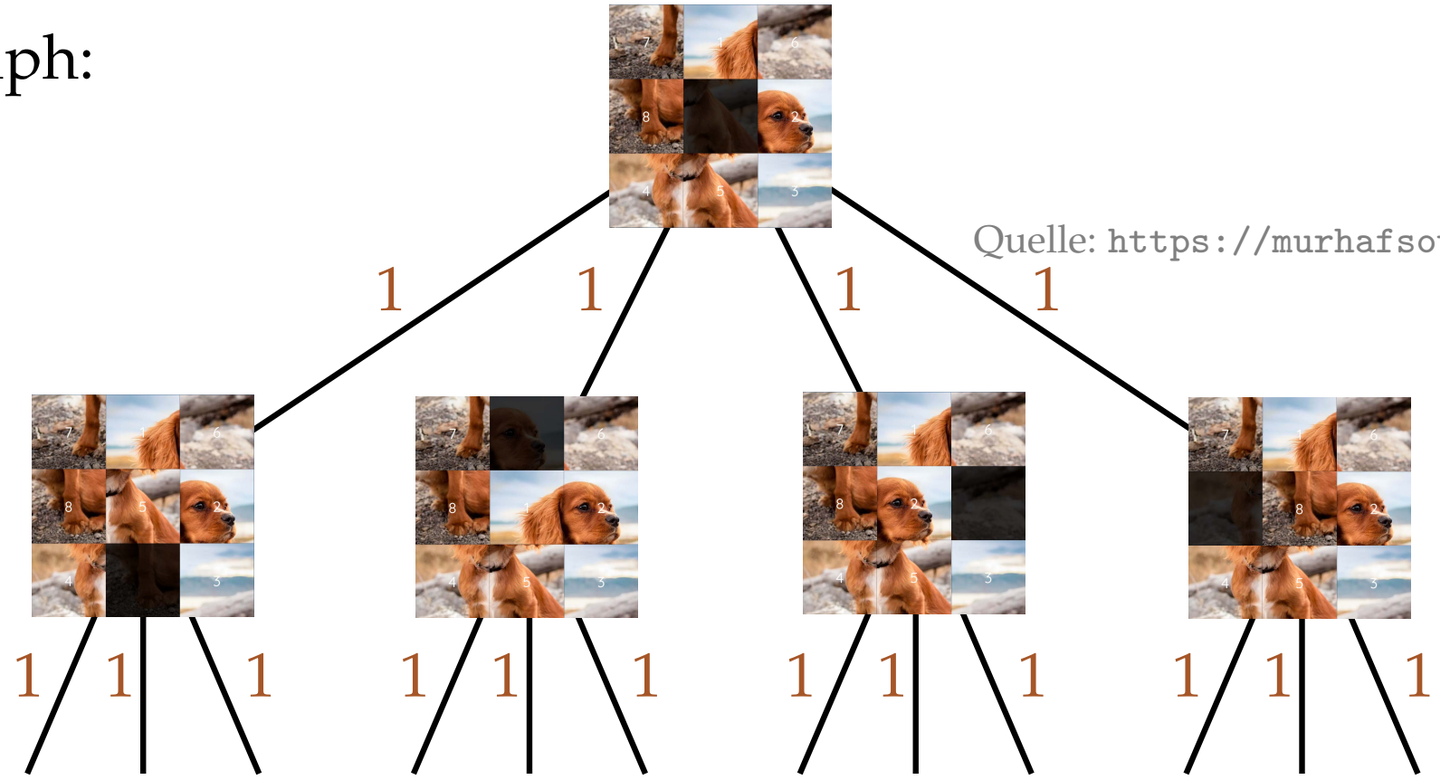
Lösung: kürzester Weg von  nach 

Schätzfunktion?
#Kacheln an falscher Position

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112	6 300	3 600 000
A*	13	39	

Nicht-geometrische Probleme: Schiebepuzzle

Modellierung als Graph:



Quelle: `https://murhafsousli.github.io/8puzzle`

Lösung: kürzester Weg von  nach 

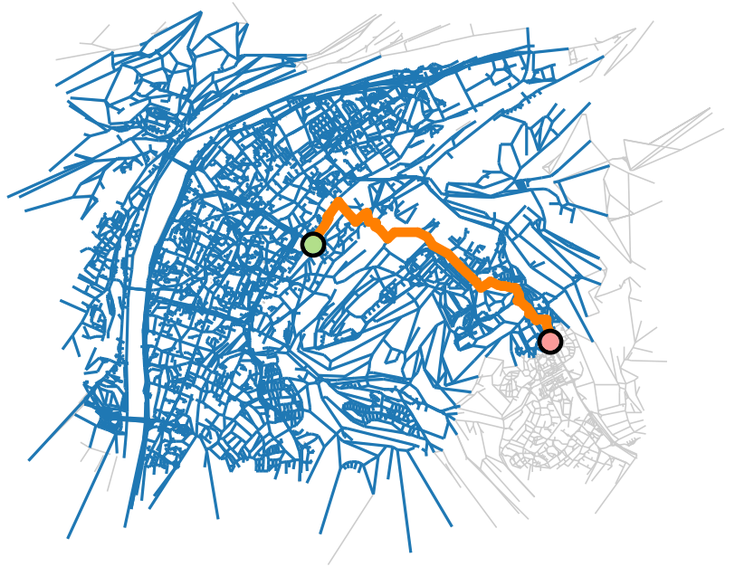
Schätzfunktion?
#Kacheln an falscher Position

Durchschnittliche Anzahl besuchter Knoten wenn kürzester Pfad Länge x hat:			
	$x = 4$	$x = 8$	$x = 12$
Dijkstra	112	6 300	3 600 000
A*	13	39	227

Universität Trier

Algorithmen für geographische Informationssysteme

2. Vorlesung
Routenplanung



Teil IV:
Contraction Hierarchies

Philipp Kindermann

Wintersemester 2024/25

Contraction Hierarchies – Idee

- Dijkstra zu langsam für Server Queries

Geisenberger, Robert et al. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Karlsruhe: Department of Informatics, KIT

Contraction Hierarchies – Idee

- Dijkstra zu langsam für Server Queries
- A* funktioniert nicht für beliebige Kostenfunktionen

Geisenberger, Robert et al. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Karlsruhe: Department of Informatics, KIT

Contraction Hierarchies – Idee

- Dijkstra zu langsam für Server Queries
- A* funktioniert nicht für beliebige Kostenfunktionen
- **Idee:** Informationen vorberechnen um Queries zu beschleunigen

Geisenberger, Robert et al. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Karlsruhe: Department of Informatics, KIT

Contraction Hierarchies – Idee

- Dijkstra zu langsam für Server Queries
- A* funktioniert nicht für beliebige Kostenfunktionen
- **Idee:** Informationen vorberechnen um Queries zu beschleunigen
- Time–Space Tradeoff

Geisenberger, Robert et al. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Karlsruhe: Department of Informatics, KIT

Contraction Hierarchies – Idee

- Dijkstra zu langsam für Server Queries
- A* funktioniert nicht für beliebige Kostenfunktionen
- **Idee:** Informationen vorberechnen um Queries zu beschleunigen
- Time–Space Tradeoff

- Es gibt eine Hierarchie:

Geisenberger, Robert et al. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Karlsruhe: Department of Informatics, KIT

Contraction Hierarchies – Idee

- Dijkstra zu langsam für Server Queries
- A* funktioniert nicht für beliebige Kostenfunktionen
- **Idee:** Informationen vorberechnen um Queries zu beschleunigen
- Time–Space Tradeoff

- Es gibt eine Hierarchie:
  Autobahn ➡ Bundesstraße ➡ Landstraße
  ➡ Kreisstraße ➡ Gemeindestraße ➡ Feldwege

Geisenberger, Robert et al. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Karlsruhe: Department of Informatics, KIT

Contraction Hierarchies – Idee

- Dijkstra zu langsam für Server Queries
- A* funktioniert nicht für beliebige Kostenfunktionen
- **Idee:** Informationen vorberechnen um Queries zu beschleunigen
- Time–Space Tradeoff

- Es gibt eine Hierarchie:
  Autobahn ➡ Bundesstraße ➡ Landstraße
  ➡ Kreisstraße ➡ Gemeindestraße ➡ Feldwege
- Entferne nach und nach unwichtigste Knoten

Geisenberger, Robert et al. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Karlsruhe: Department of Informatics, KIT

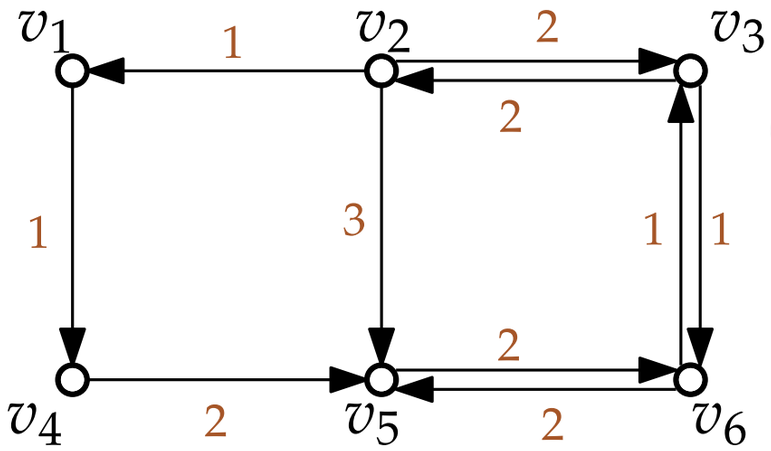
Contraction Hierarchies – Idee

- Dijkstra zu langsam für Server Queries
- A* funktioniert nicht für beliebige Kostenfunktionen
- **Idee:** Informationen vorberechnen um Queries zu beschleunigen
- Time–Space Tradeoff

- Es gibt eine Hierarchie:
  Autobahn ➡ Bundesstraße ➡ Landstraße
  ➡ Kreisstraße ➡ Gemeindestraße ➡ Feldwege
- Entferne nach und nach unwichtigste Knoten
- Füge Abkürzungen in das Straßennetz ein, um kürzeste Wege aufrechtzuerhalten

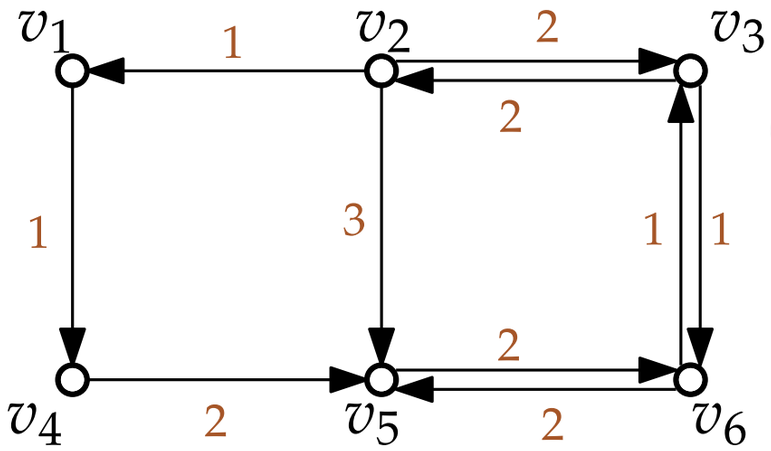
Geisenberger, Robert et al. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. Karlsruhe: Department of Informatics, KIT

Kontraktion



$G = (V, E)$

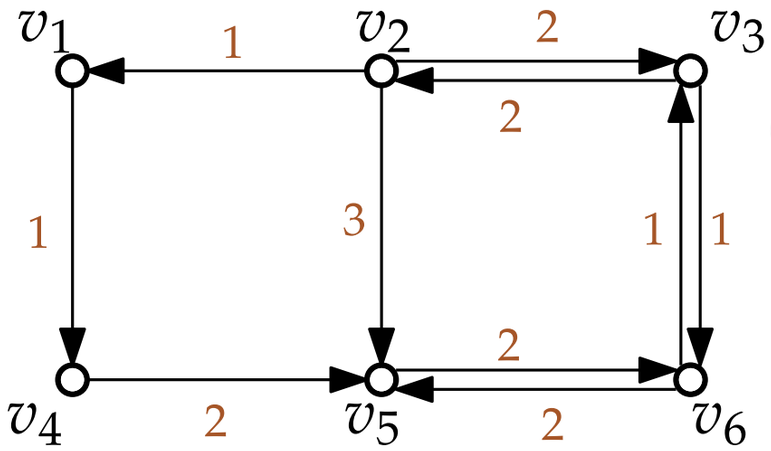
Kontraktion



$G = (V, E)$

Kontraktion eines Knotens $v \in V$, z.B. v_3:

Kontraktion

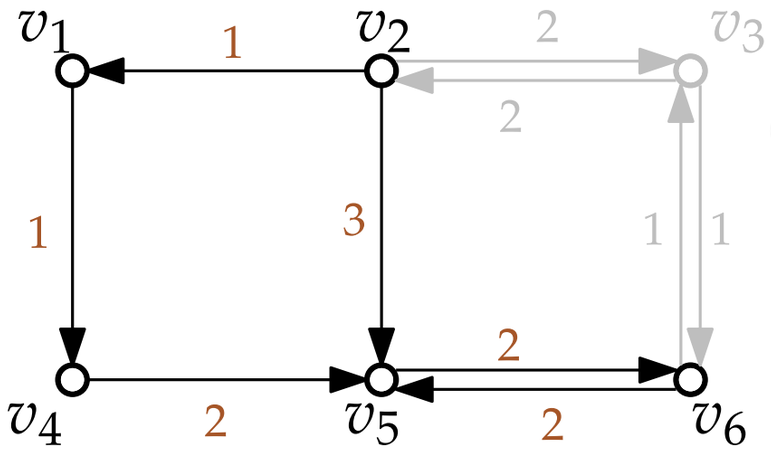


$G = (V, E)$

Kontraktion eines Knotens $v \in V$, z.B. v_3:

- v und inzidente Kanten werden aus G entfernt (*deaktiviert*)

Kontraktion

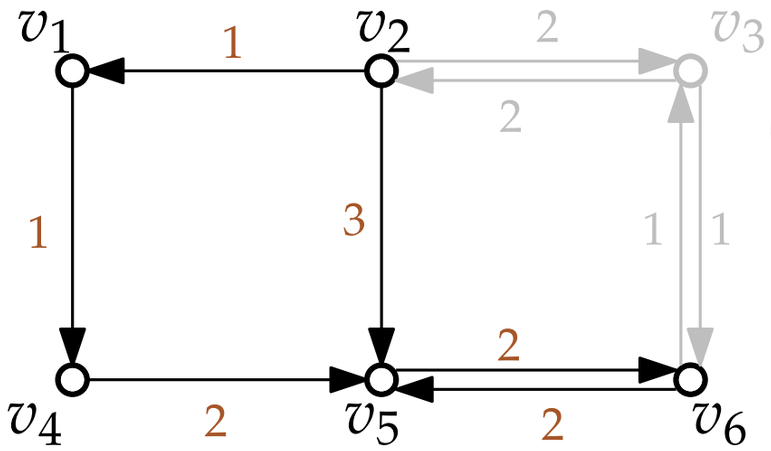


$G = (V, E)$

Kontraktion eines Knotens $v \in V$, z.B. v_3:

- v und inzidente Kanten werden aus G entfernt (*deaktiviert*)

Kontraktion

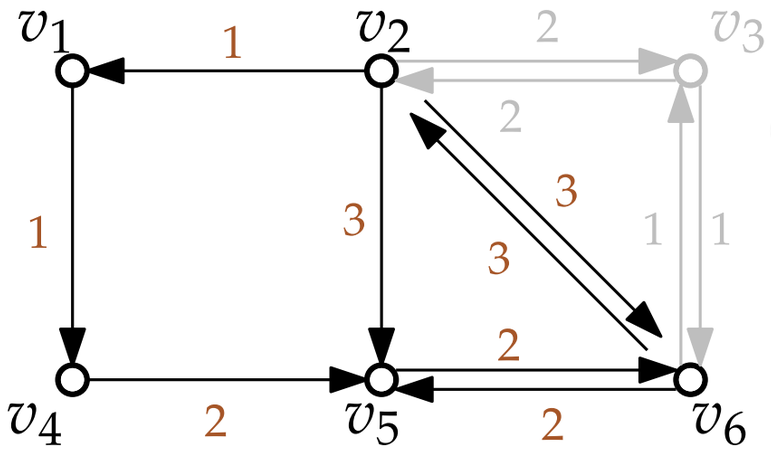


$G = (V, E)$

Kontraktion eines Knotens $v \in V$, z.B. v_3:

- v und inzidente Kanten werden aus G entfernt (*deaktiviert*)
- Für jeden kürzesten Pfad uvw wird eine neue Kante uw mit entsprechendem Gewicht eingefügt.

Kontraktion

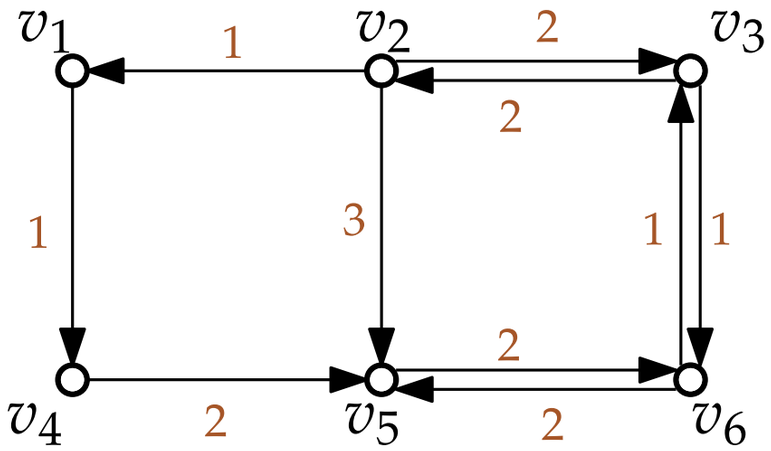


$G = (V, E)$

Kontraktion eines Knotens $v \in V$, z.B. v_3:

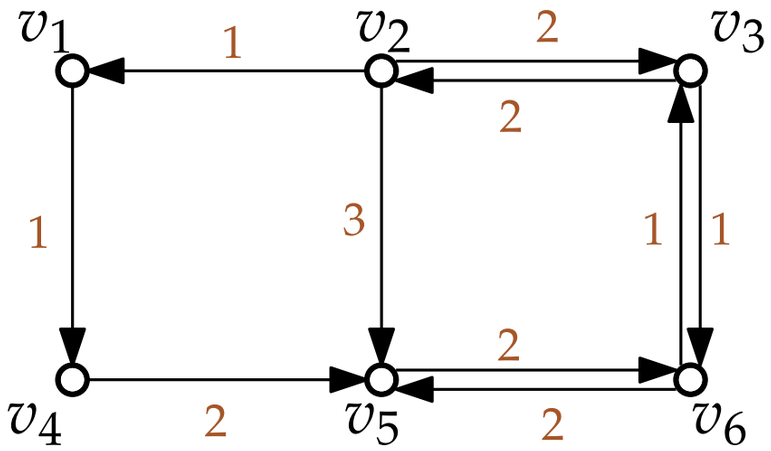
- v und inzidente Kanten werden aus G entfernt (*deaktiviert*)
- Für jeden kürzesten Pfad uvw wird eine neue Kante uw mit entsprechendem Gewicht eingefügt.

Contraction Hierarchies – Definition



Contraction Hierarchy:

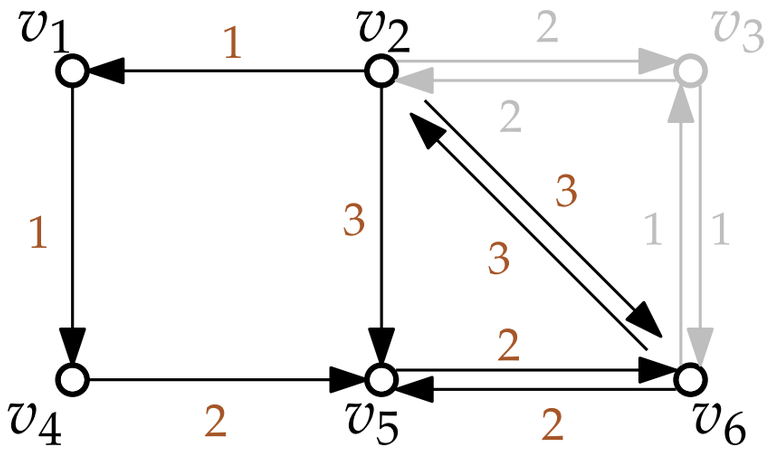
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.

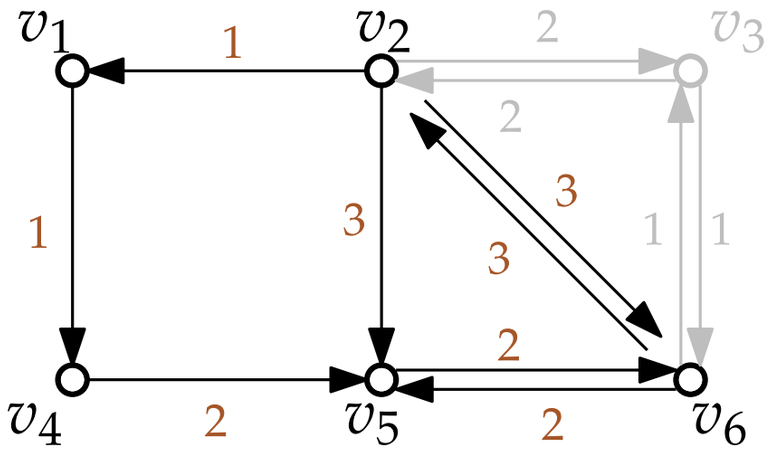
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.

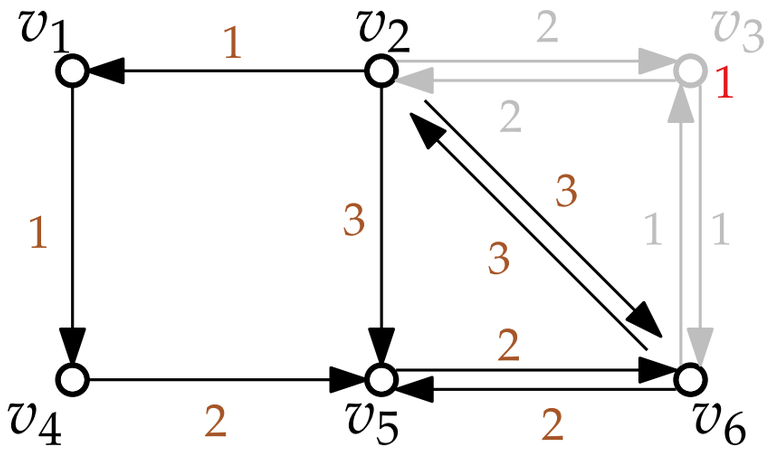
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.

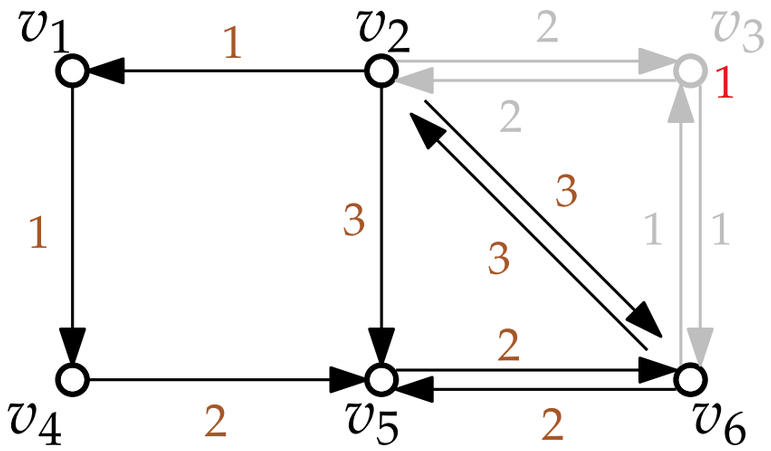
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.

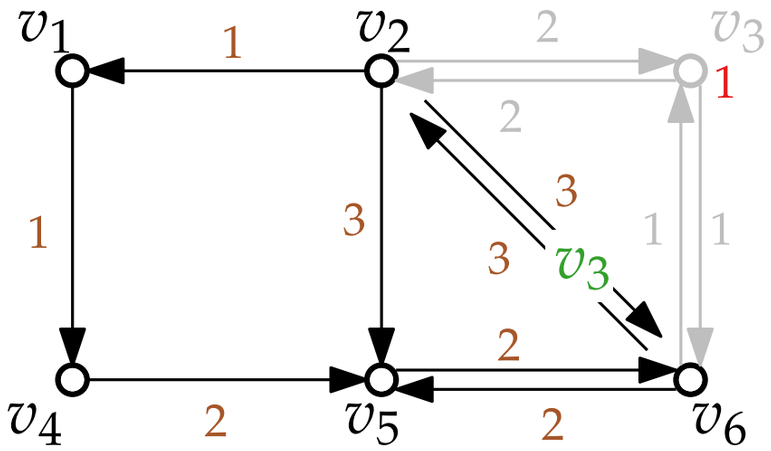
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.
- Für jede neue Kante speichern wir, **welche Knoten** übersprungen wurden.

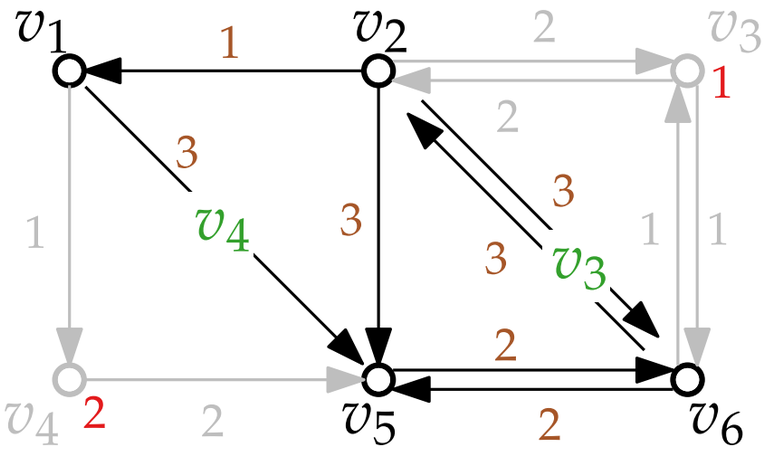
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.
- Für jede neue Kante speichern wir, **welche Knoten** übersprungen wurden.

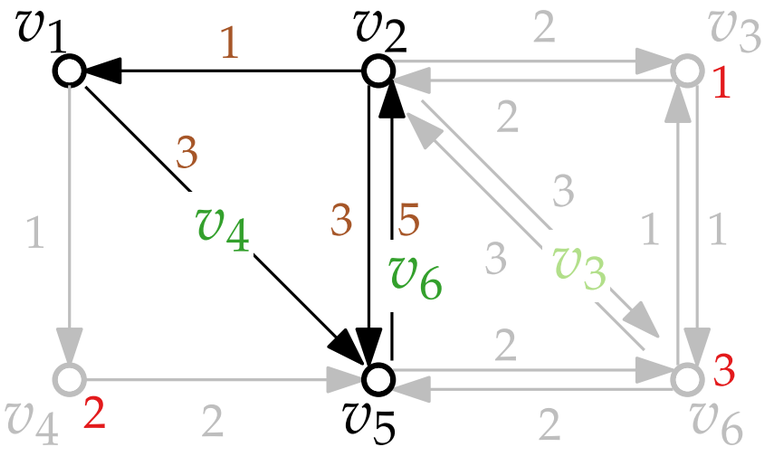
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.
- Für jede neue Kante speichern wir, **welche Knoten** übersprungen wurden.

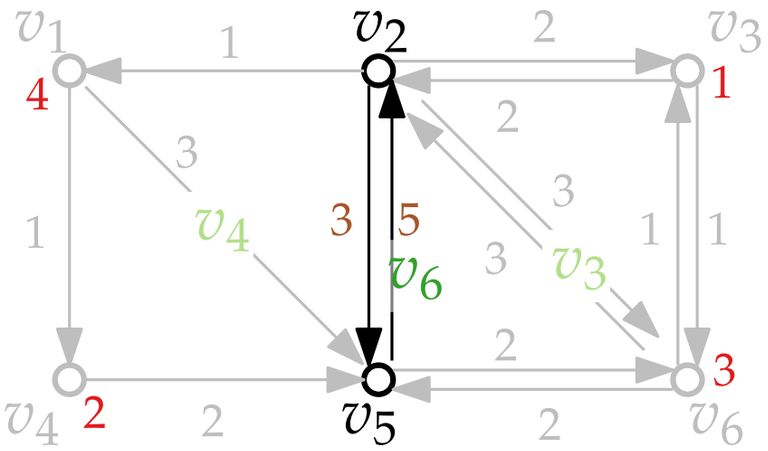
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.
- Für jede neue Kante speichern wir, **welche Knoten** übersprungen wurden.

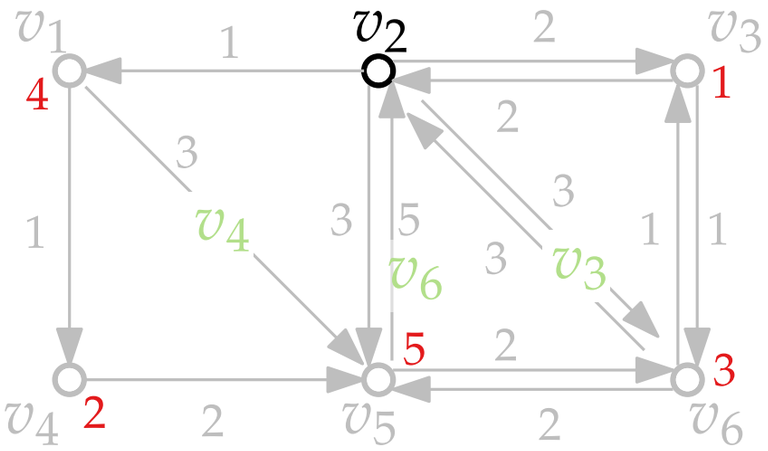
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.
- Für jede neue Kante speichern wir, **welche Knoten** übersprungen wurden.

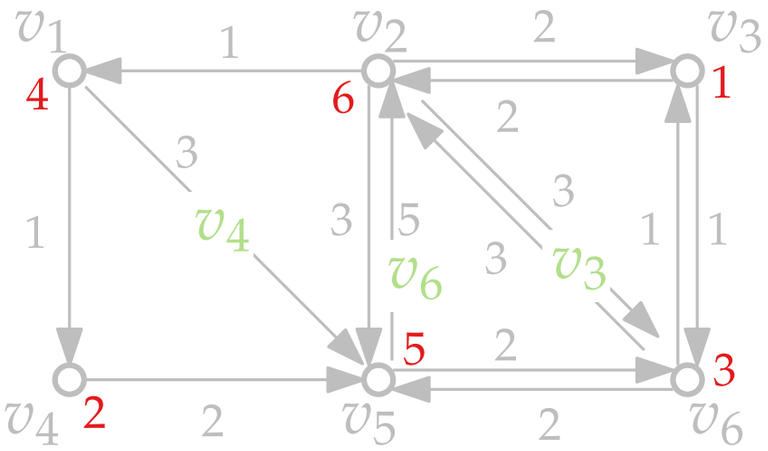
Contraction Hierarchies – Definition



Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.
- Für jede neue Kante speichern wir, **welche Knoten** übersprungen wurden.

Contraction Hierarchies – Definition

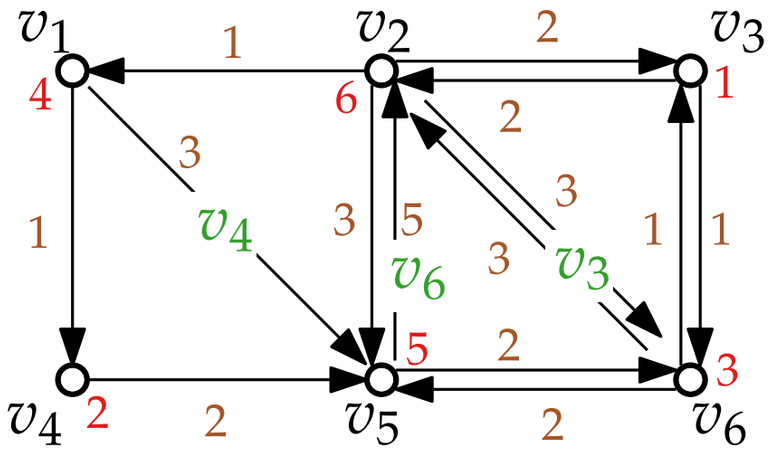


Contraction Hierarchy:

- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.
- Für jede neue Kante speichern wir, **welche Knoten** übersprungen wurden.

Contraction Hierarchies – Definition

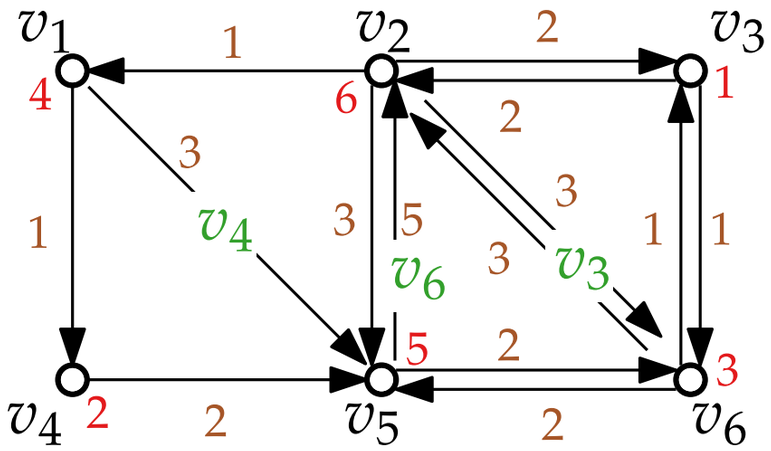
$G' = (V, E')$



- Knoten werden in bestimmter Reihenfolge kontrahiert, z.B. $v_3, v_4, v_6, v_1, v_5, v_2$.
- Für jeden kontrahierten Knoten speichern wir den **Rang**, d.h., **wann** der Knoten kontrahiert wurde.
- Für jede neue Kante speichern wir, **welche Knoten** übersprungen wurden.

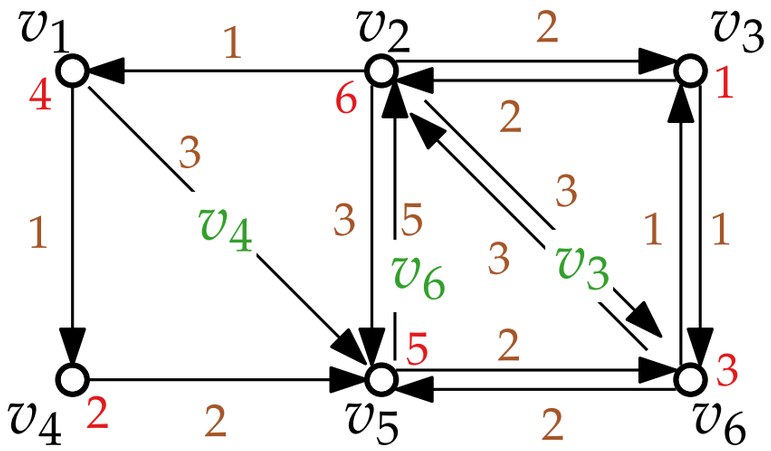
Contraction Hierarchies – Eigenschaften

$G' = (V, E')$



Contraction Hierarchies – Eigenschaften

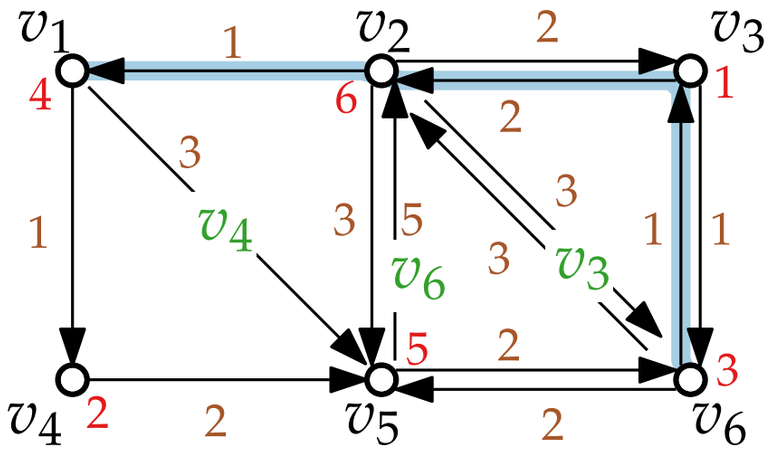
$G' = (V, E')$



Für zwei Knoten $u, v \in V$, sei P_{uv} ein kürzester u-v-Pfad in G.

Contraction Hierarchies – Eigenschaften

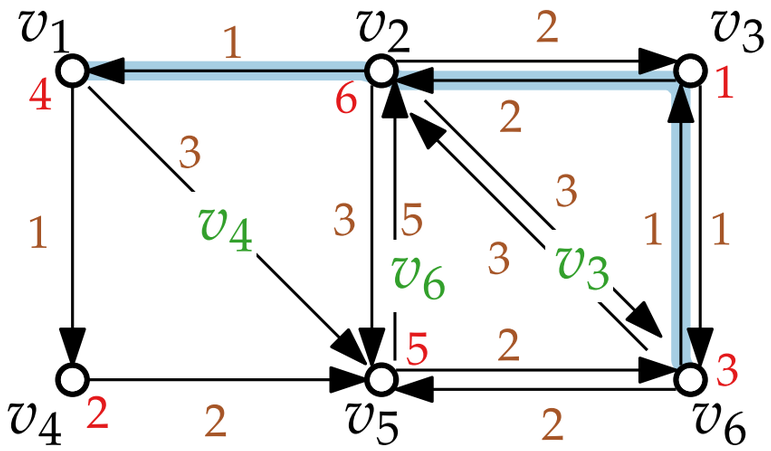
$G' = (V, E')$



Für zwei Knoten $u, v \in V$, sei P_{uv} ein kürzester u-v-Pfad in G.

Contraction Hierarchies – Eigenschaften

$G' = (V, E')$

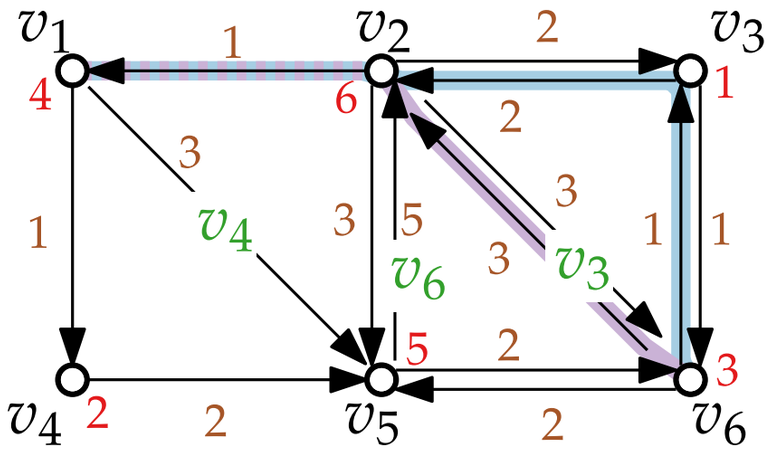


Für zwei Knoten $u, v \in V$, sei P_{uv} ein kürzester u-v-Pfad in G.

Es gibt einen kürzesten u-v Pfad P'_{uv} in G', so dass:

Contraction Hierarchies – Eigenschaften

$G' = (V, E')$

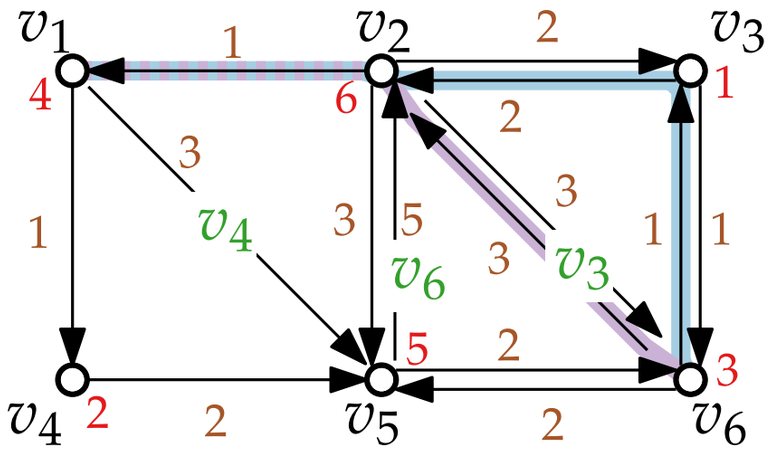


Für zwei Knoten $u, v \in V$, sei P_{uv} ein kürzester u-v-Pfad in G.

Es gibt einen kürzesten u-v Pfad P'_{uv} in G', so dass:

Contraction Hierarchies – Eigenschaften

$G' = (V, E')$



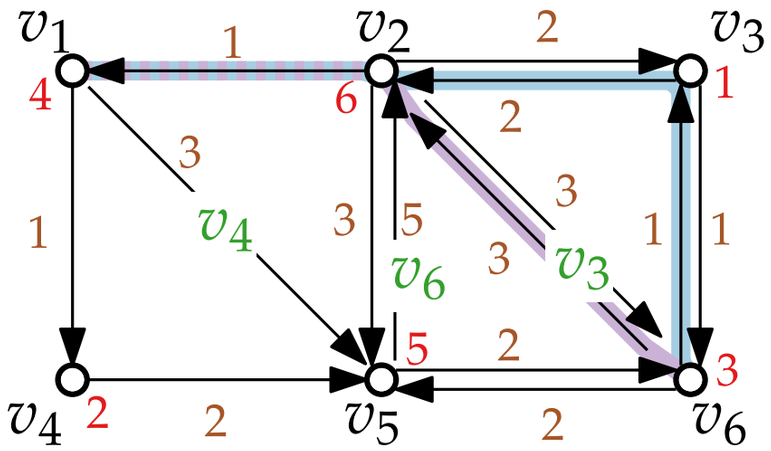
Für zwei Knoten $u, v \in V$, sei P_{uv} ein kürzester u-v-Pfad in G.

Es gibt einen kürzesten u-v Pfad P'_{uv} in G', so dass:

- die **Ränge** der Knoten steigen zuerst und sinken dann,

Contraction Hierarchies – Eigenschaften

$G' = (V, E')$



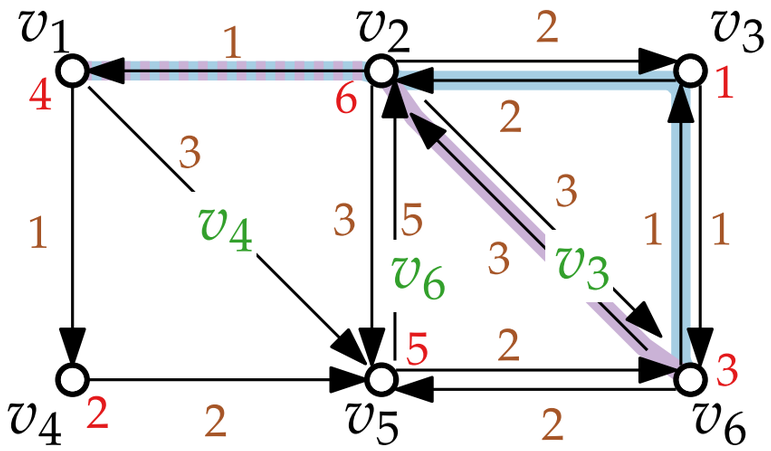
Für zwei Knoten $u, v \in V$, sei P_{uv} ein kürzester u-v-Pfad in G.

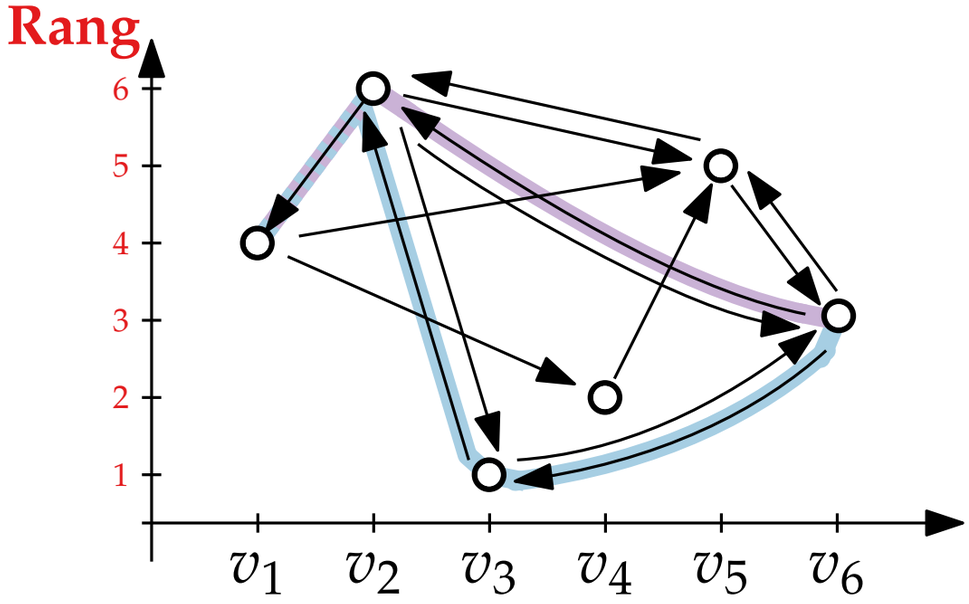
Es gibt einen kürzesten u-v Pfad P'_{uv} in G', so dass:

- die **Ränge** der Knoten steigen zuerst und sinken dann,
- man erhält den Pfad P_{uv} aus P'_{uv}, indem man nach und nach die gespeicherten Abkürzungen auflöst.

Contraction Hierarchies – Eigenschaften

$G' = (V, E')$





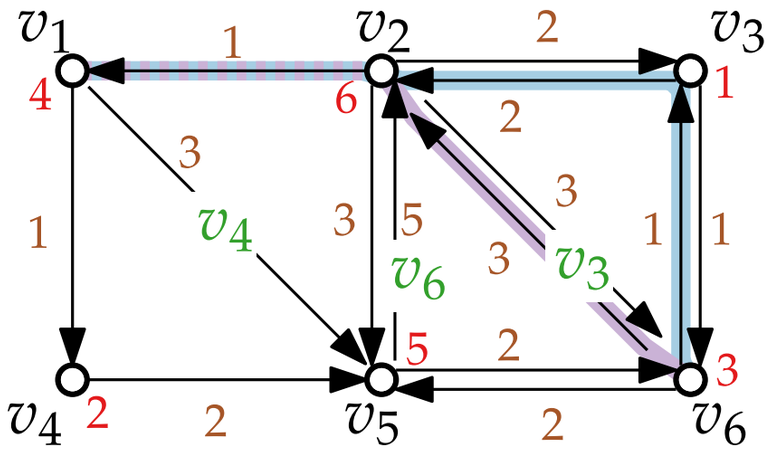
Für zwei Knoten $u, v \in V$, sei P_{uv} ein kürzester u-v-Pfad in G.

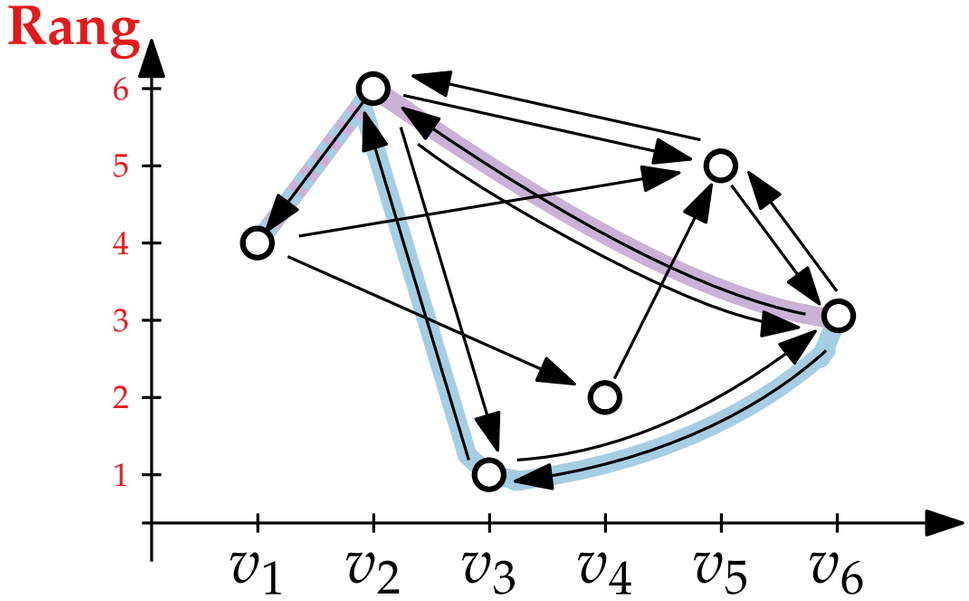
Es gibt einen kürzesten u-v Pfad P'_{uv} in G', so dass:

- die **Ränge** der Knoten steigen zuerst und sinken dann,
- man erhält den Pfad P_{uv} aus P'_{uv}, indem man nach und nach die gespeicherten Abkürzungen auflöst.

Contraction Hierarchies – Benutzung

$G' = (V, E')$

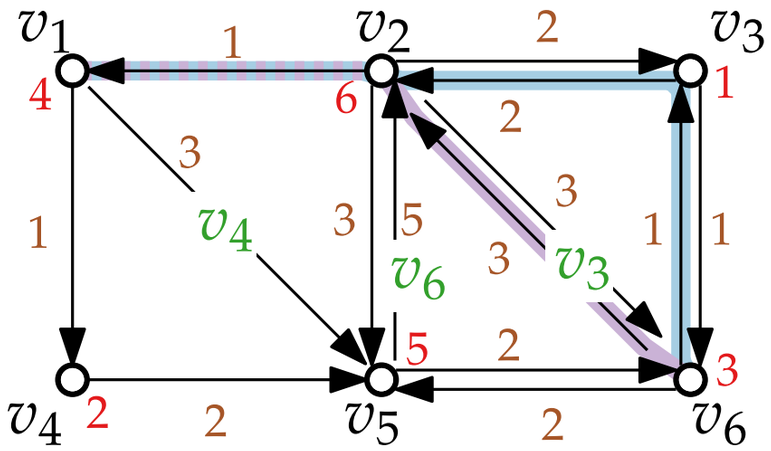


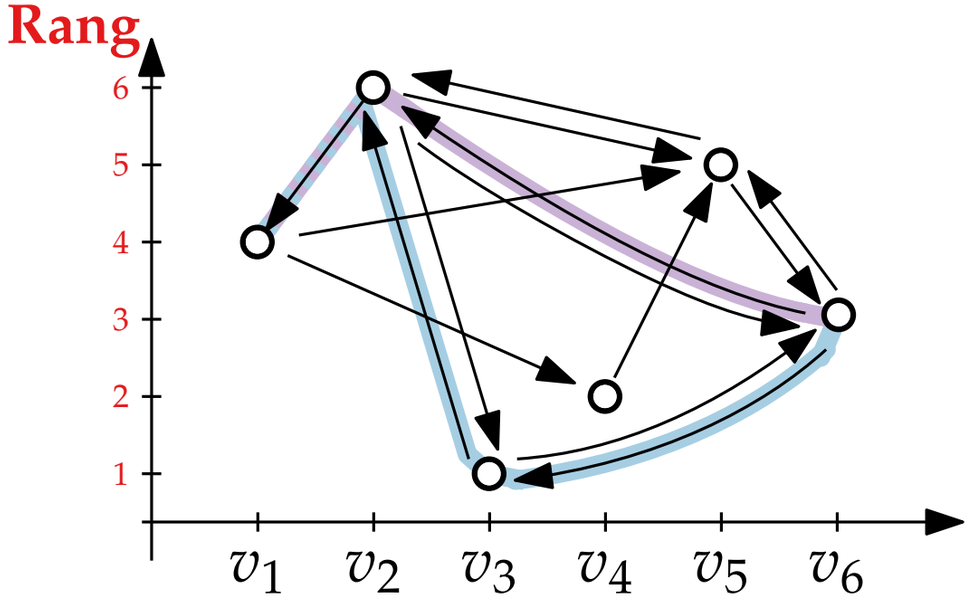


P_{uv} berechnen:

Contraction Hierarchies – Benutzung

$G' = (V, E')$



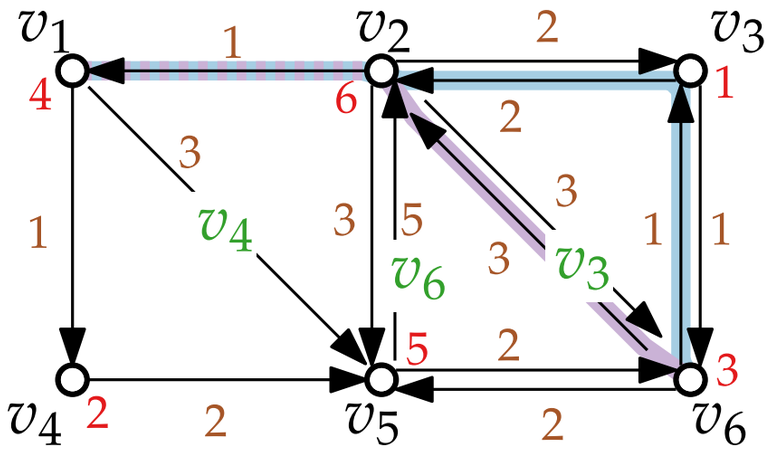


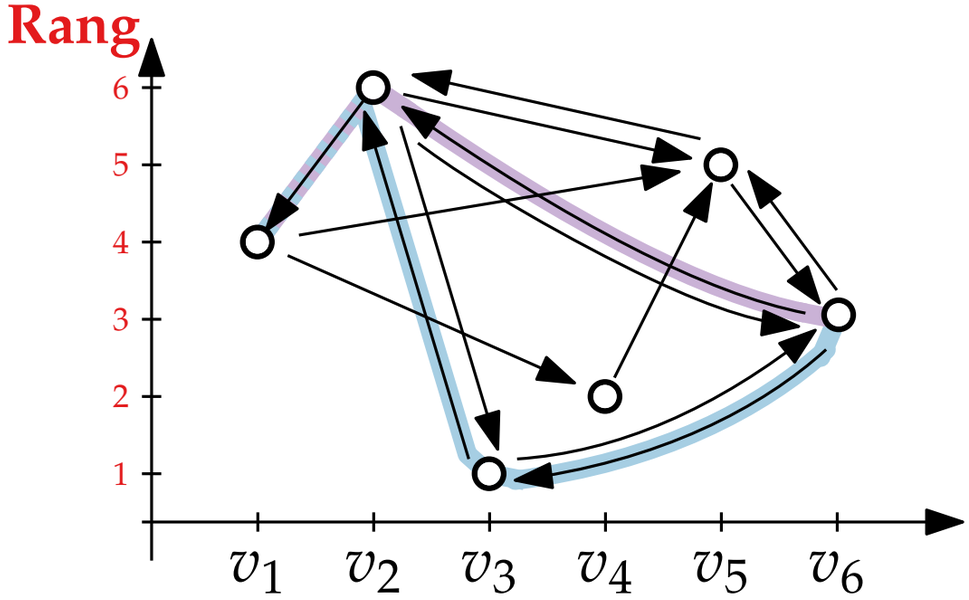
P_{uv} berechnen:

1. Berechne P'_{uv}.

Contraction Hierarchies – Benutzung

$G' = (V, E')$



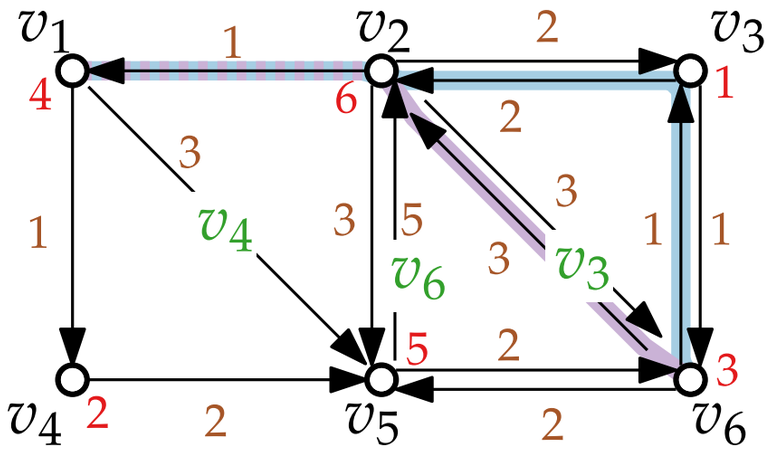


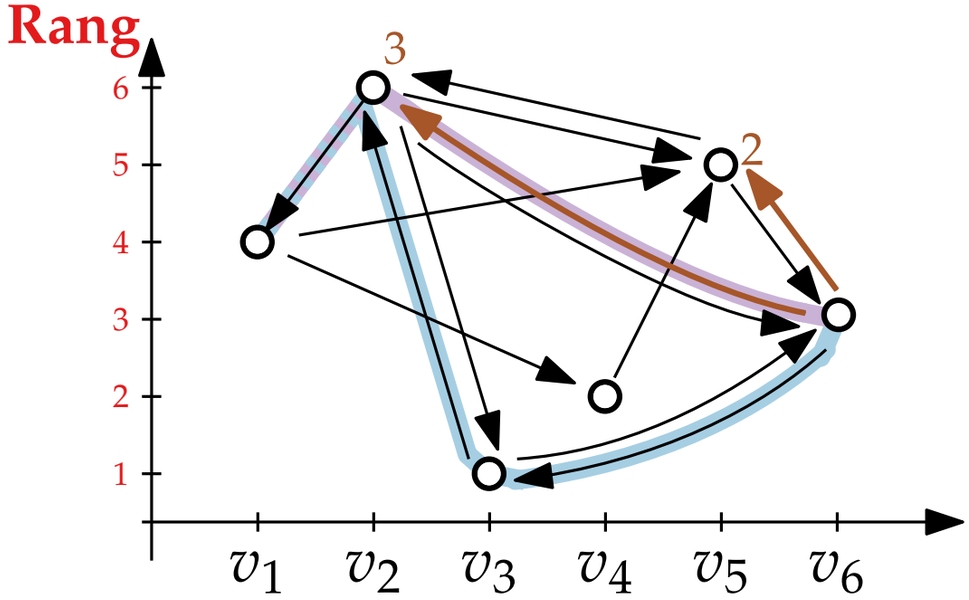
P_{uv} berechnen:

1. Berechne P'_{uv}.
2. Transformiere P'_{uv} zu P_{uv}.

Contraction Hierarchies – Benutzung

$G' = (V, E')$





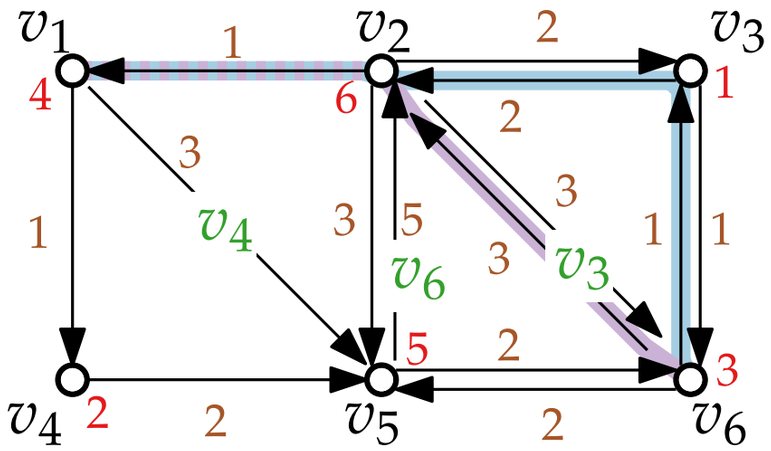
P_{uv} berechnen:

1. Berechne P'_{uv}.
2. Transformiere P'_{uv} zu P_{uv}.

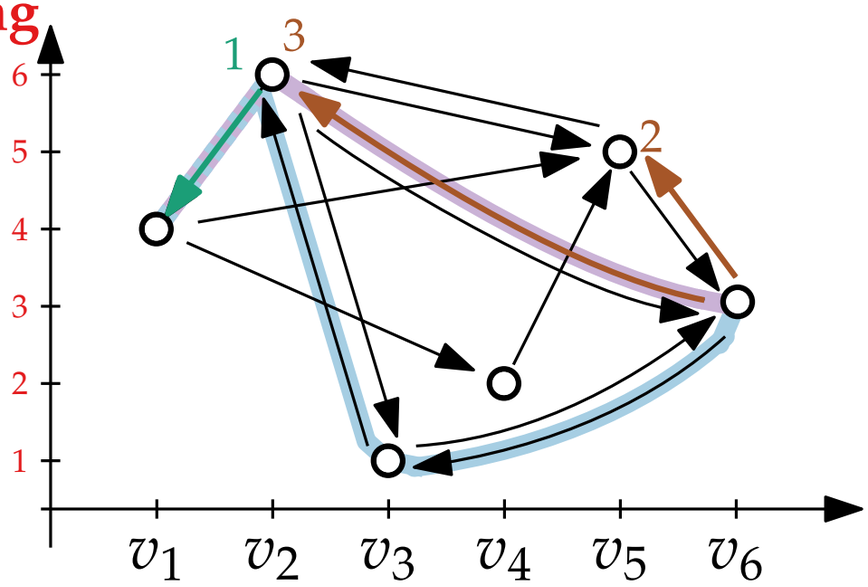
1. Benutze DIJKSTRA von u aus mit nur aufsteigenden Kanten

Contraction Hierarchies – Benutzung

$G' = (V, E')$



Rang



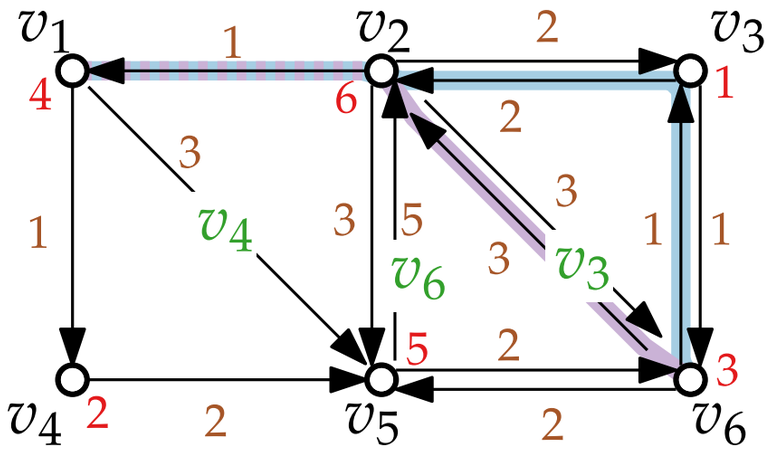
P_{uv} berechnen:

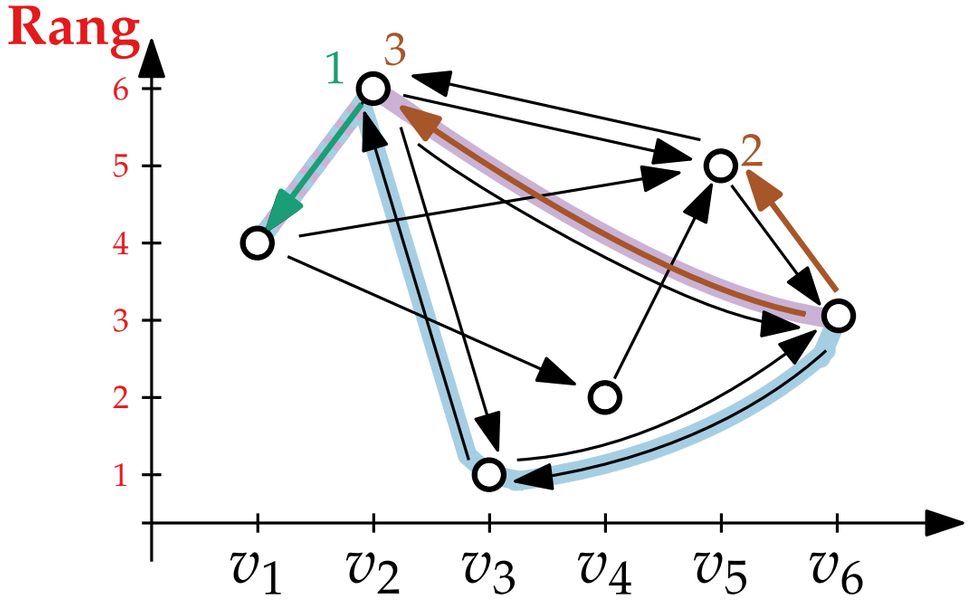
1. Berechne P'_{uv}.
2. Transformiere P'_{uv} zu P_{uv}.

1. Benutze DIJKSTRA von u aus mit nur aufsteigenden Kanten
2. Benutze DIJKSTRA von v aus mit nur absteigenden Rückwärtskanten

Contraction Hierarchies – Benutzung

$G' = (V, E')$





P_{uv} berechnen:

1. Berechne P'_{uv}.
2. Transformiere P'_{uv} zu P_{uv}.

1. Benutze Dijkstra von u aus mit nur aufsteigenden Kanten
2. Benutze Dijkstra von v aus mit nur absteigenden Rückwärtskanten
3. „Wendepunkt“ hat niedrigste kumulative Distanz ($1 + 3 = 4$).

Contraction Hierarchies – Anmerkungen

- Die beiden DIJKSTRA-Durchläufe können gleichzeitig anstelle von nacheinander durchgeführt werden, indem eine *bidirektionale Suche* verwendet wird. Dadurch können noch mehr Knoten unerforscht bleiben.

Contraction Hierarchies – Anmerkungen

- Die beiden DIJKSTRA-Durchläufe können gleichzeitig anstelle von nacheinander durchgeführt werden, indem eine *bidirektionale Suche* verwendet wird. Dadurch können noch mehr Knoten unerforscht bleiben.

- Die Methode funktioniert für jede Kontraktionssequenz, aber die Wahl der Knotenreihenfolge beeinflusst die Laufzeit erheblich. Es existieren mehrere Methoden zur Optimierung der Reihenfolge.