

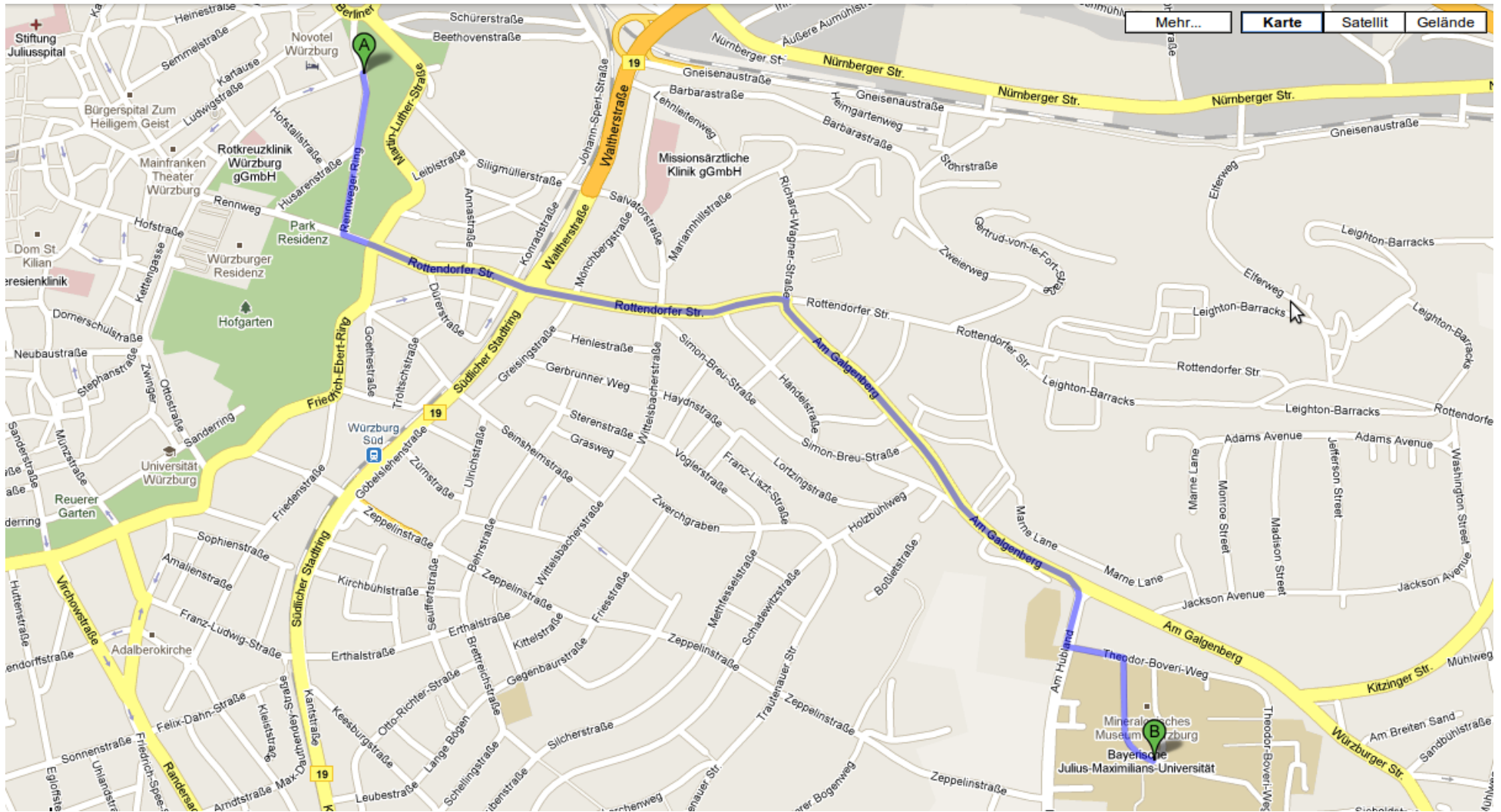
Algorithmen und Datenstrukturen

Wintersemester 2019/20

19. Vorlesung

Kürzeste Wege & Dijkstras Algorithmus

Wozu kürzeste Wege?



Modellierung des Problems Routenplanung

Straßenkreuzung \rightarrow

Straßenabschnitt \rightarrow

Einbahnstraßenabschnitt \rightarrow

Fahrtzeit für Abschnitt e \rightarrow

Straßennetz \rightarrow

Start \rightarrow

Ziel \rightarrow

Start-Ziel-Route \rightarrow

Modellierung des Problems Routenplanung

Straßenkreuzung \rightarrow Knoten

Straßenabschnitt \rightarrow

Einbahnstraßenabschnitt \rightarrow

Fahrtzeit für Abschnitt e \rightarrow

Straßennetz \rightarrow

Start \rightarrow

Ziel \rightarrow

Start-Ziel-Route \rightarrow

Modellierung des Problems Routenplanung

Straßenkreuzung \rightarrow Knoten

Straßenabschnitt \rightarrow zwei entgegengerichtete Kanten

Einbahnstraßenabschnitt \rightarrow

Fahrtzeit für Abschnitt e \rightarrow

Straßennetz \rightarrow

Start \rightarrow

Ziel \rightarrow

Start-Ziel-Route \rightarrow

Modellierung des Problems Routenplanung

Straßenkreuzung \rightarrow Knoten

Straßenabschnitt \rightarrow zwei entgegengerichtete Kanten

Einbahnstraßenabschnitt \rightarrow in Fahrtrichtung gerichtete Kante

Fahrtzeit für Abschnitt $e \rightarrow$

Straßennetz \rightarrow

Start \rightarrow

Ziel \rightarrow

Start-Ziel-Route \rightarrow

Modellierung des Problems Routenplanung

Straßenkreuzung \rightarrow Knoten

Straßenabschnitt \rightarrow zwei entgegengerichtete Kanten

Einbahnstraßenabschnitt \rightarrow in Fahrtrichtung gerichtete Kante

Fahrtzeit für Abschnitt $e \rightarrow$ Kantengewicht $w(e) \geq 0$

Straßennetz \rightarrow

Start \rightarrow

Ziel \rightarrow

Start-Ziel-Route \rightarrow

Modellierung des Problems Routenplanung

Straßenkreuzung \rightarrow Knoten

Straßenabschnitt \rightarrow zwei entgegengerichtete Kanten

Einbahnstraßenabschnitt \rightarrow in Fahrtrichtung gerichtete Kante

Fahrtzeit für Abschnitt $e \rightarrow$ Kantengewicht $w(e) \geq 0$

Straßennetz \rightarrow gerichteter, gewichteter und zusammenhängender Graph $G = (V, E)$

Start \rightarrow

Ziel \rightarrow

Start-Ziel-Route \rightarrow

Modellierung des Problems Routenplanung

Straßenkreuzung \rightarrow Knoten

Straßenabschnitt \rightarrow zwei entgegengerichtete Kanten

Einbahnstraßenabschnitt \rightarrow in Fahrtrichtung gerichtete Kante

Fahrtzeit für Abschnitt $e \rightarrow$ Kantengewicht $w(e) \geq 0$

Straßennetz \rightarrow gerichteter, gewichteter und zusammenhängender Graph $G = (V, E)$

Start \rightarrow Knoten $s \in V$

Ziel \rightarrow Knoten $t \in V$

Start-Ziel-Route \rightarrow

Modellierung des Problems Routenplanung

Straßenkreuzung \rightarrow Knoten

Straßenabschnitt \rightarrow zwei entgegengerichtete Kanten

Einbahnstraßenabschnitt \rightarrow in Fahrtrichtung gerichtete Kante

Fahrtzeit für Abschnitt $e \rightarrow$ Kantengewicht $w(e) \geq 0$

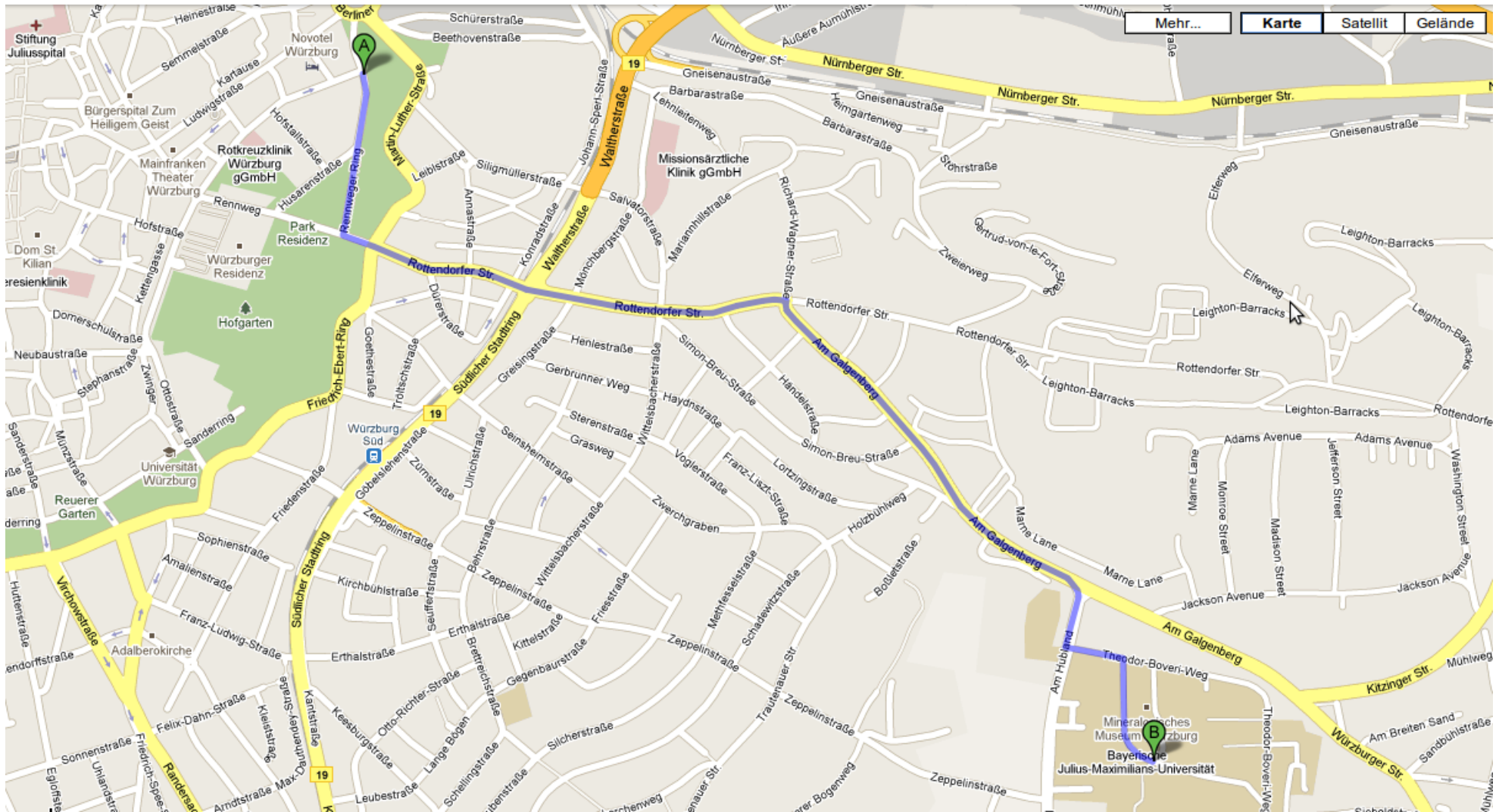
Straßennetz \rightarrow gerichteter, gewichteter und zusammenhängender Graph $G = (V, E)$

Start \rightarrow Knoten $s \in V$

Ziel \rightarrow Knoten $t \in V$

Start-Ziel-Route \rightarrow s - t -Weg, d.h. Folge von Kanten $(s, v_1), (v_1, v_2), \dots, (v_k, t)$ in G

Wozu kürzeste Wege?



Wozu kürzeste Wege?



Wozu kürzeste Wege?



Wozu kürzeste Wege? (II)



Kontakt | Hilfe | Sitemap a a+ a++

Frage oder Suchbegriff eingeben ...

Suchen

Startseite | Angebotsberatung | **Fahrplan & Buchung** | Services | BahnCard | Urlaub

Meine Bahn

Suche
 Auswahl
 Ticket&Reservierung
 Zahlung
 Buchung
 Bestätigung

[-> Neue Anfrage](#)

Reisedaten 1 Erwachsener, 2. Klasse

Hinfahrt von Abfahrt Ankunft
 nach

[-> weitere Angaben ändern](#) **Aktualisieren**

Kurzfristige Fahrplanänderungen

Informieren Sie sich hier über aktuelle Verkehrsmeldungen.

[-> Weitere Informationen](#)

Ihre Hinfahrtmöglichkeiten - sortiert nach

Druckansicht

| Bahnhof/Haltestelle | Datum | Zeit | Dauer | Umst. | Produkte | Normalpreis | |
|---|--------------|----------|-------|-------|----------|-----------------------------|--|
| | | ↑ Früher | | | | | |
| <input type="checkbox"/> Würzburg Busbahnhof Mathematisches Institut, Würzburg | Di, 12.01.10 | ab 10:29 | 0:16 | 0 | Bus | Preisauskunft nicht möglich | -> Rückfahrt hinzufügen |
| | Di, 12.01.10 | an 10:45 | | | | | |
| <input type="checkbox"/> Würzburg Busbahnhof Mathematisches Institut, Würzburg | Di, 12.01.10 | ab 10:49 | 0:16 | 0 | Bus | Preisauskunft nicht möglich | -> Rückfahrt hinzufügen |
| | Di, 12.01.10 | an 11:05 | | | | | |
| <input type="checkbox"/> Würzburg Busbahnhof Mathematisches Institut, Würzburg | Di, 12.01.10 | ab 11:09 | 0:16 | 0 | Bus | Preisauskunft nicht möglich | -> Rückfahrt hinzufügen |
| | Di, 12.01.10 | an 11:25 | | | | | |
| <input type="checkbox"/> Details für alle anzeigen | | ↓ Später | | | | | |

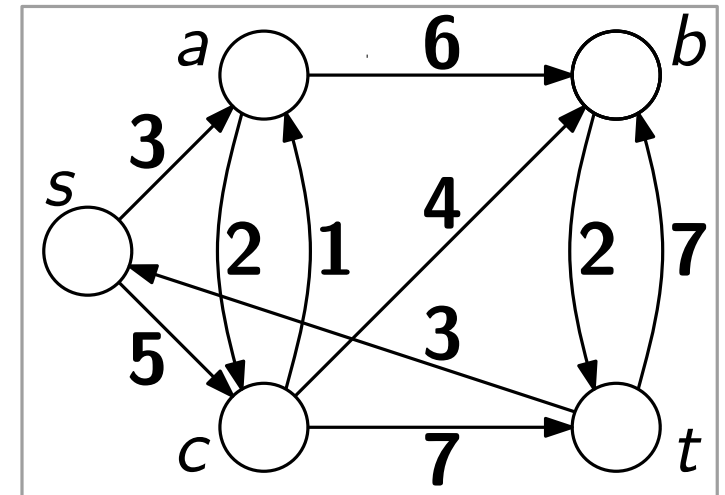
MobilCheck
 UmweltMobilCheck

Zurück

Was ist das Problem?

Eingabe:

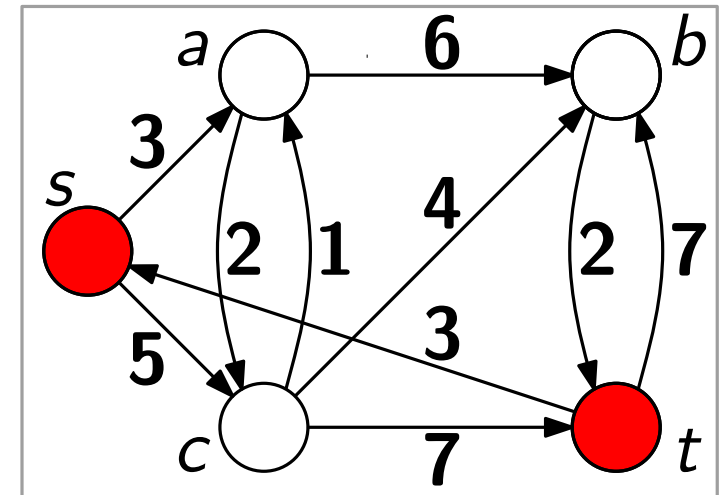
- gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen **Kantengewichten** $w: E \rightarrow \mathbb{Q}_0^+$,



Was ist das Problem?

Eingabe:

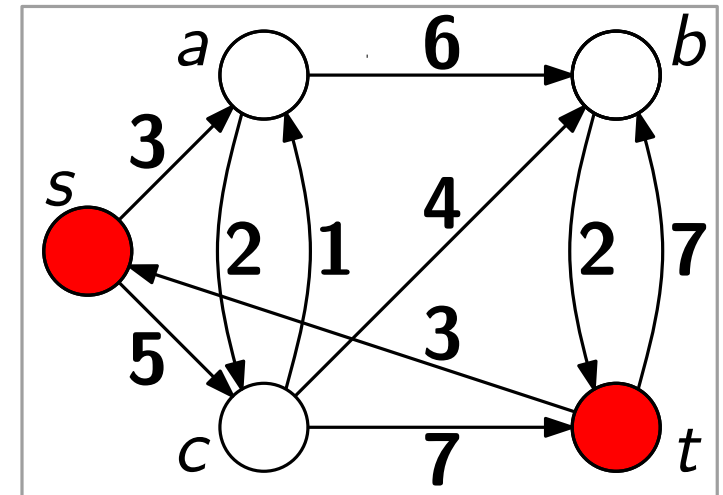
- gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen **Kantengewichten** $w: E \rightarrow \mathbb{Q}_0^+$,
- Knoten s und t



Was ist das Problem?

Eingabe:

- gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen **Kantengewichten** $w: E \rightarrow \mathbb{Q}_0^+$,
- Knoten s und t



Ausgabe:

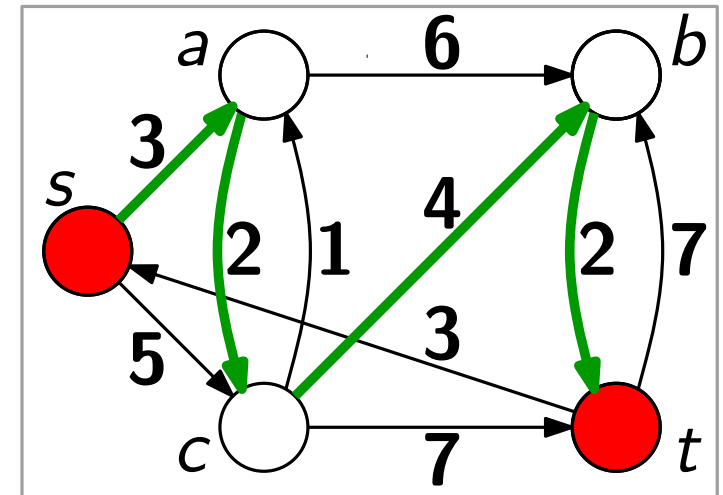
- kürzester s - t -Weg** W in G , d.h. $\sum_{e \in W} w(e)$ minimal.

Darstellung durch Vorgänger-Zeiger π : für jeden Knoten v sei $\pi(v) \in V \cup \{nil\}$ Vorgänger von v auf kürzestem s - v -Weg.

Was ist das Problem?

Eingabe:

- gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen **Kantengewichten** $w: E \rightarrow \mathbb{Q}_0^+$,
- Knoten s und t



Ausgabe:

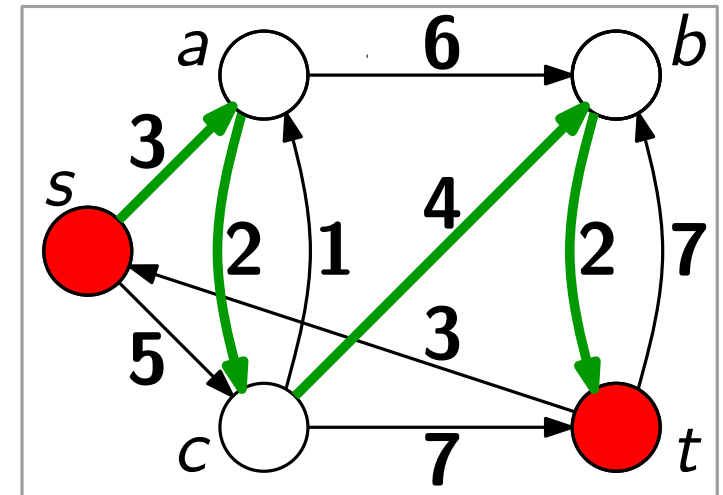
- kürzester s - t -Weg** W in G , d.h. $\sum_{e \in W} w(e)$ minimal.

Darstellung durch Vorgänger-Zeiger π : für jeden Knoten v sei $\pi(v) \in V \cup \{nil\}$ Vorgänger von v auf kürzestem s - v -Weg.

Was ist das Problem?

Eingabe:

- gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen **Kantengewichten** $w: E \rightarrow \mathbb{Q}_0^+$,
- Knoten s und t



Ausgabe:

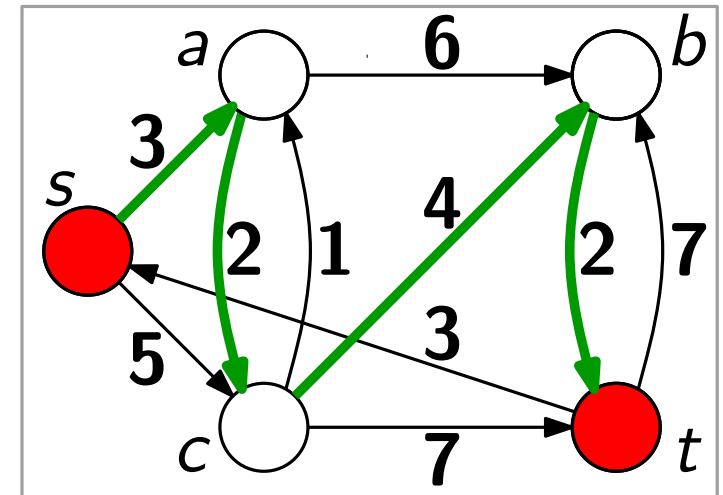
- kürzester** s - t -Weg W_t in G , $\underbrace{\text{für alle } t \in V}$ d.h. $\sum_{e \in W} w(e)$ minimal.

Darstellung durch Vorgänger-Zeiger π : für jeden Knoten v sei $\pi(v) \in V \cup \{nil\}$ Vorgänger von v auf kürzestem s - v -Weg.

Was ist das Problem?

Eingabe:

- gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen **Kantengewichten** $w: E \rightarrow \mathbb{Q}_0^+$,
- Knoten s und t



Ausgabe:

- kürzester/ s-t-Wege** W_t in G , $\underbrace{\text{für alle } t \in V}$ d.h. $\sum_{e \in W} w(e)$ minimal.

Darstellung durch Vorgänger-Zeiger π : für jeden Knoten v sei $\pi(v) \in V \cup \{nil\}$ Vorgänger von v auf kürzestem s - v -Weg.

Nebenbemerkung: Analoge Berechnungsverfahren?

Dijkstra – BFS mit Gewichten

Dijkstra(WeightedGraph $G = (V, E; w)$, Vertex s)



Dijkstra – BFS mit Gewichten

Dijkstra(WeightedGraph $G = (V, E; w)$, Vertex s)

BFS(Graph G , Vertex s)

Initialize(G, s)

$Q = \mathbf{new}$ Queue()

$Q.Enqueue(s)$

while not $Q.Empty()$ **do**

$u = Q.Dequeue()$

foreach $v \in Adj[u]$ **do**

if $v.color == white$ **then**

$v.color = gray$

$v.d = u.d + 1$

$v.\pi = u$

$Q.Enqueue(v)$

$u.color = black$

Dijkstra – BFS mit Gewichten

Dijkstra(WeightedGraph $G = (V, E; w)$, Vertex s)

Initialize(G, s)

// Gewichtung

$Q = \mathbf{new}$ PriorityQueue(V, d)

while not $Q.Empty()$ **do**

$u = Q.ExtractMin()$

foreach $v \in Adj[u]$ **do**

 Relax($u, v; w$)

$u.color = black$

BFS(Graph G , Vertex s)

Initialize(G, s)

$Q = \mathbf{new}$ Queue()

$Q.Enqueue(s)$

while not $Q.Empty()$ **do**

$u = Q.Dequeue()$

foreach $v \in Adj[u]$ **do**

if $v.color == white$ **then**

$v.color = gray$

$v.d = u.d + 1$

$v.\pi = u$

$Q.Enqueue(v)$

$u.color = black$

Dijkstra – BFS mit Gewichten

Dijkstra(WeightedGraph $G = (V, E; w)$, Vertex s)

Initialize(G, s)

// Gewichtung

$Q = \mathbf{new}$ PriorityQueue(V, d)

while not $Q.Empty()$ **do**

$u = Q.ExtractMin()$

foreach $v \in Adj[u]$ **do**

 Relax($u, v; w$)

$u.color = black$

BFS(Graph G , Vertex s)

Initialize(G, s)

$Q = \mathbf{new}$ Queue()

$Q.Enqueue(s)$

while not $Q.Empty()$ **do**

$u = Q.Dequeue()$

foreach $v \in Adj[u]$ **do**

if $v.color == white$ **then**

$v.color = gray$

$v.d = u.d + 1$

$v.\pi = u$

$Q.Enqueue(v)$

$u.color = black$

Dijkstra – BFS mit Gewichten

Dijkstra(WeightedGraph $G = (V, E; w)$, Vertex s)

Initialize(G, s)

// Gewichtung

$Q = \mathbf{new}$ PriorityQueue(V, d)

while not $Q.Empty()$ **do**

$u = Q.ExtractMin()$

foreach $v \in Adj[u]$ **do**

 Relax($u, v; w$)

$u.color = black$

Relax($u, v; w$)

Was muss in dieser
Unterroutine passieren?

Schreiben Sie's auf!

BFS(Graph G , Vertex s)

Initialize(G, s)

$Q = \mathbf{new}$ Queue()

$Q.Enqueue(s)$

while not $Q.Empty()$ **do**

$u = Q.Dequeue()$

foreach $v \in Adj[u]$ **do**

if $v.color == white$ **then**

$v.color = gray$

$v.d = u.d + 1$

$v.\pi = u$

$Q.Enqueue(v)$

$u.color = black$

Dijkstra – BFS mit Gewichten

Dijkstra(WeightedGraph $G = (V, E; w)$, Vertex s)

Initialize(G, s)

// Gewichtung

$Q = \mathbf{new}$ PriorityQueue(V, d)

while not $Q.$ Empty() **do**

$u = Q.$ ExtractMin()

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.$ color = *black*

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.$ color = *gray*

$v.d = u.d + w(u, v)$

$v.$ $\pi = u$

$Q.$ DecreaseKey($v, v.d$)

BFS(Graph G , Vertex s)

Initialize(G, s)

$Q = \mathbf{new}$ Queue()

$Q.$ Enqueue(s)

while not $Q.$ Empty() **do**

$u = Q.$ Dequeue()

foreach $v \in \text{Adj}[u]$ **do**

if $v.$ color == *white* **then**

$v.$ color = *gray*

$v.d = u.d + 1$

$v.$ $\pi = u$

$Q.$ Enqueue(v)

$u.$ color = *black*

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \mathbf{new}$ PriorityQueue(V , d)

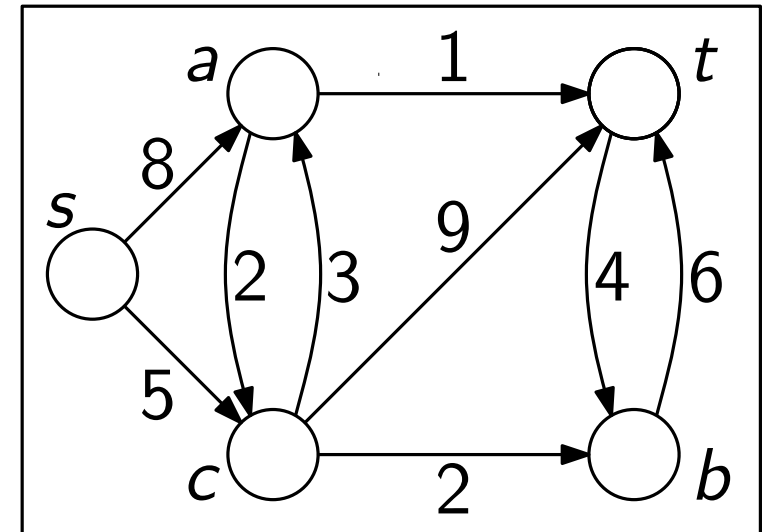
while not Q .Empty() **do**

$u = Q$.ExtractMin()

foreach $v \in \text{Adj}[u]$ **do**

 Relax(u , v ; w)

u .color = *black*



Relax(u , v ; w)

if $v.d > u.d + w(u, v)$ **then**

v .color = *gray*

$v.d = u.d + w(u, v)$

$v.\pi = u$

Q .DecreaseKey(v , $v.d$)

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

u .color = *white*

$u.d = \infty$

$u.\pi = \mathit{nil}$

s .color = *gray*

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

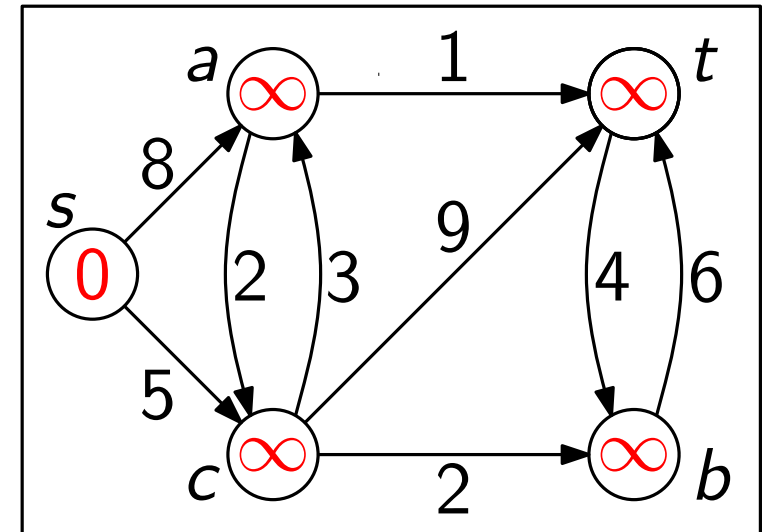
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



$\text{Relax}(u, v; w)$

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

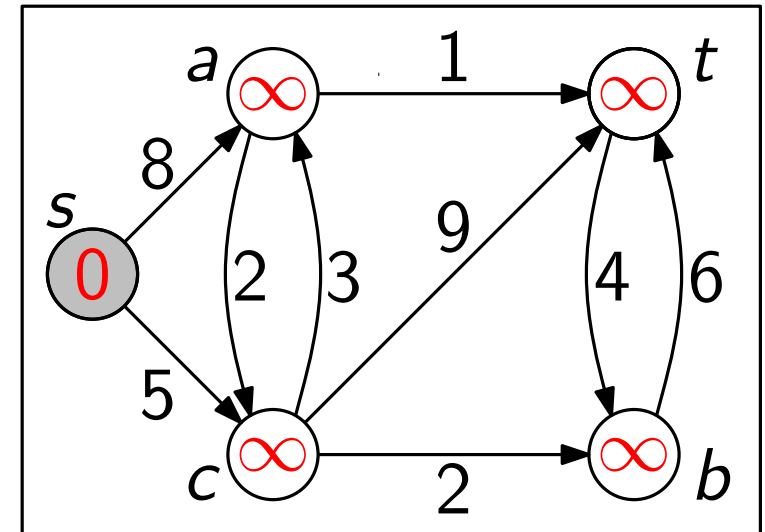
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



$\text{Relax}(u, v; w)$

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

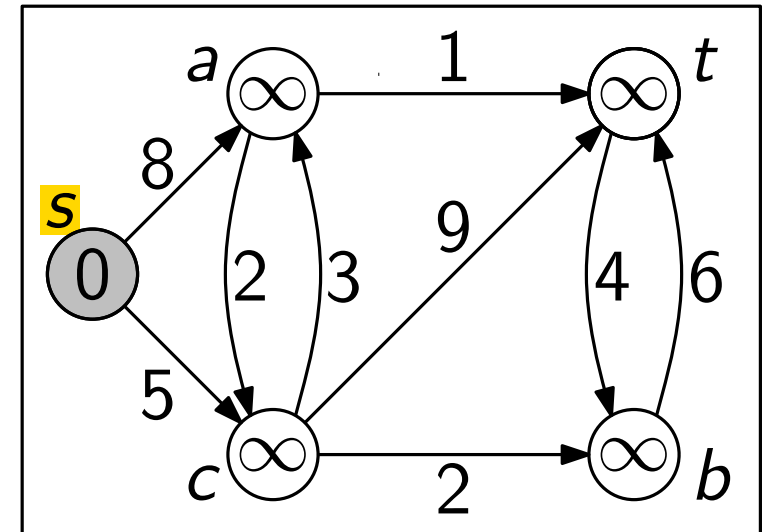
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

└ Relax($u, v; w$)

└ $u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

└ $v.\text{color} = \text{gray}$

└ $v.d = u.d + w(u, v)$

└ $v.\pi = u$

└ $Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

└ $u.\text{color} = \text{white}$

└ $u.d = \infty$

└ $u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

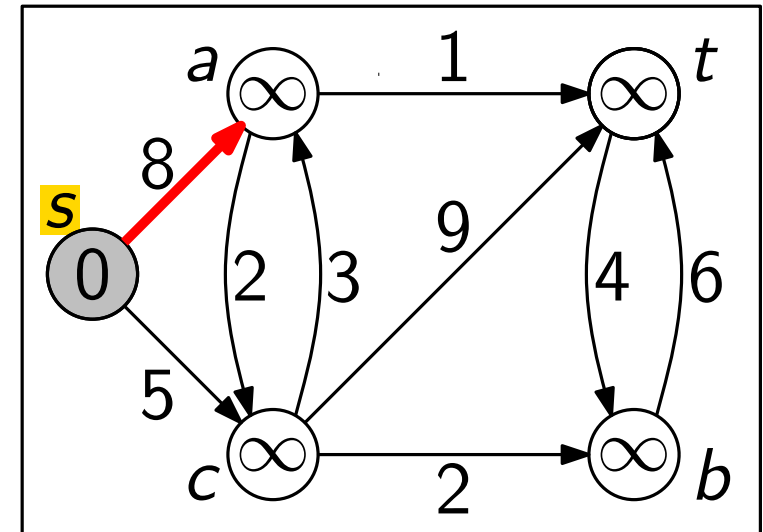
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

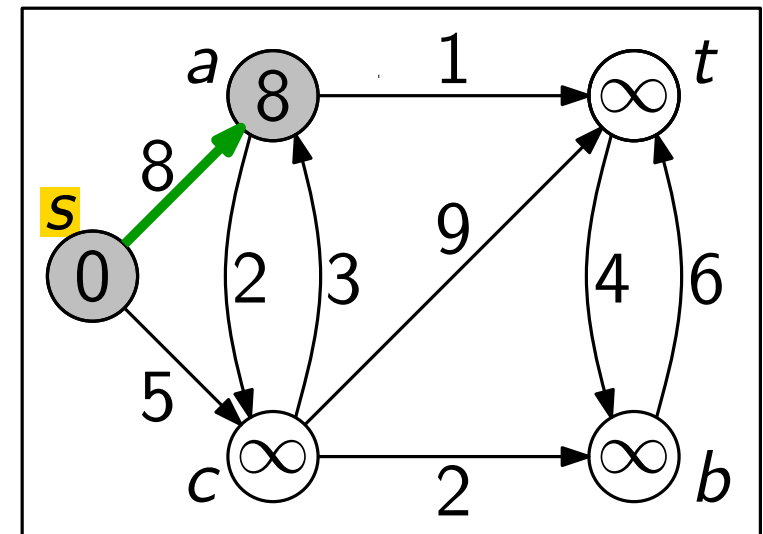
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

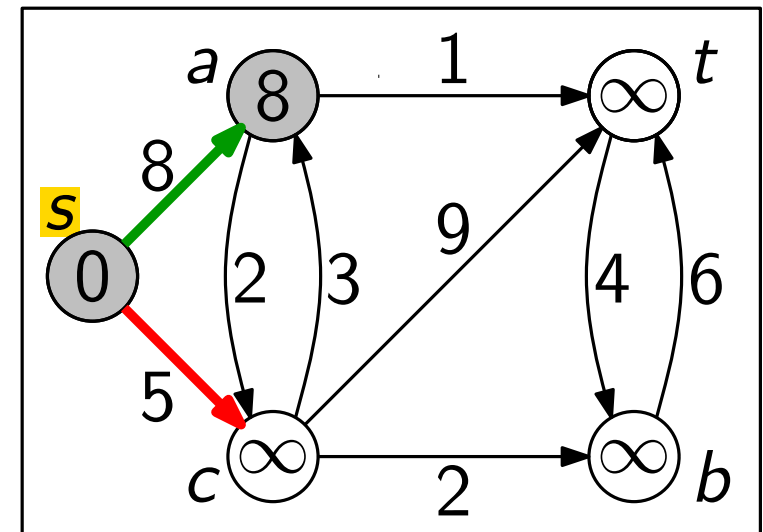
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

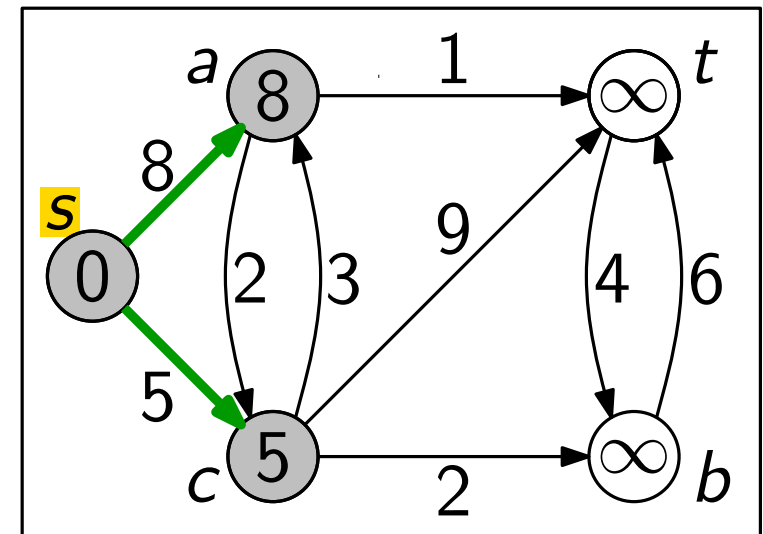
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



Relax(u , v ; w)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

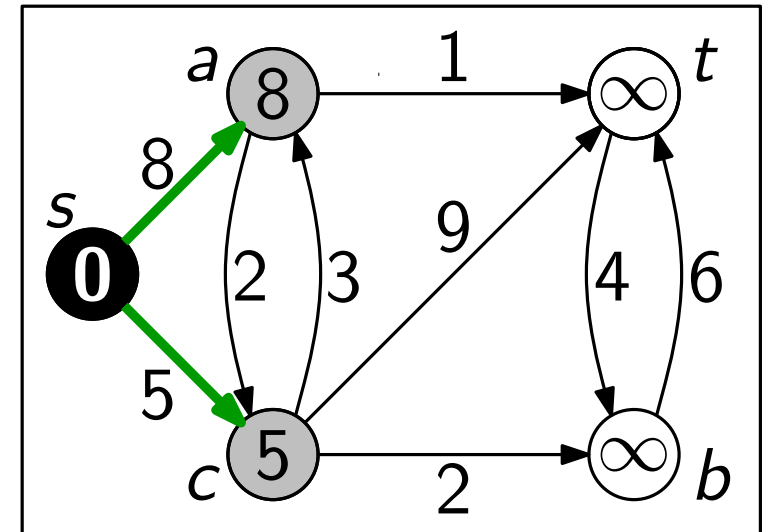
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

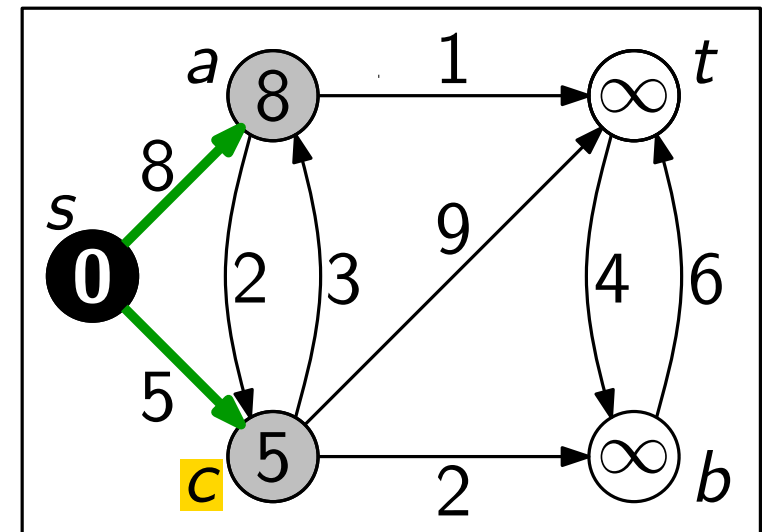
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

└ Relax($u, v; w$)

└ $u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

└ $v.\text{color} = \text{gray}$

└ $v.d = u.d + w(u, v)$

└ $v.\pi = u$

└ $Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

└ $u.\text{color} = \text{white}$

└ $u.d = \infty$

└ $u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

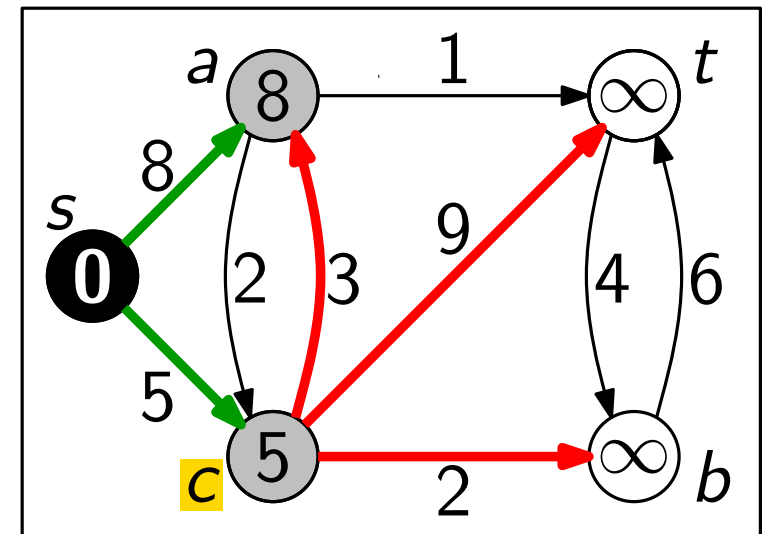
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

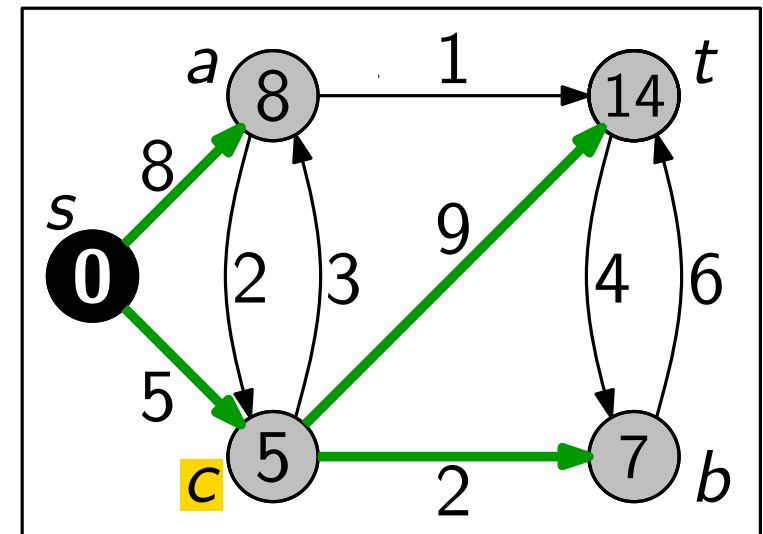
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

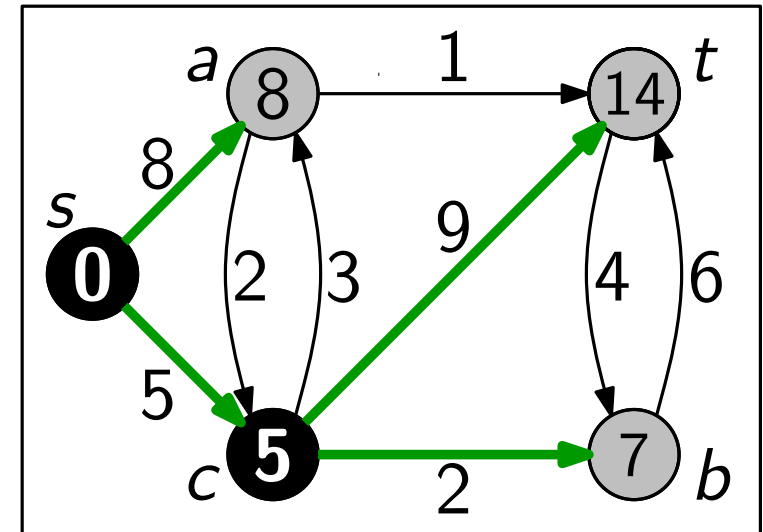
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

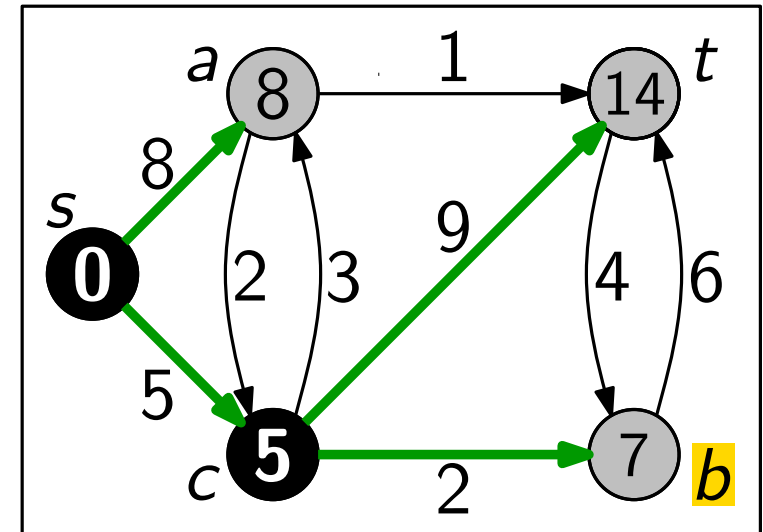
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

└ Relax($u, v; w$)

└ $u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

└ $v.\text{color} = \text{gray}$

└ $v.d = u.d + w(u, v)$

└ $v.\pi = u$

└ $Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

└ $u.\text{color} = \text{white}$

└ $u.d = \infty$

└ $u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

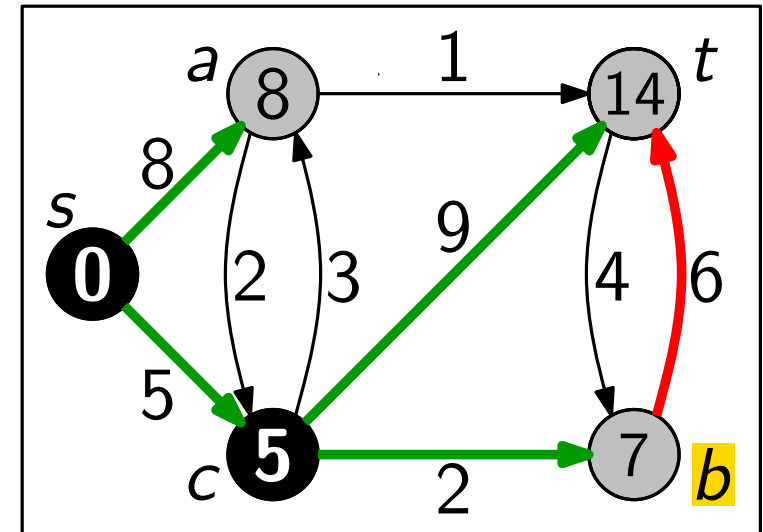
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

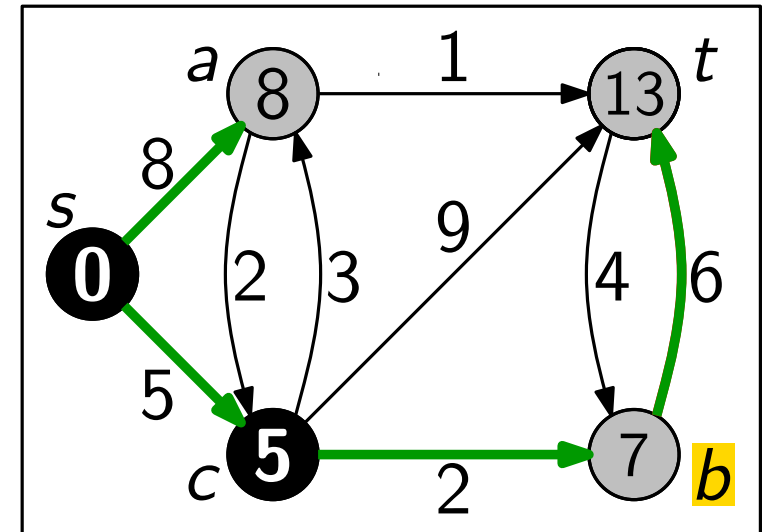
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

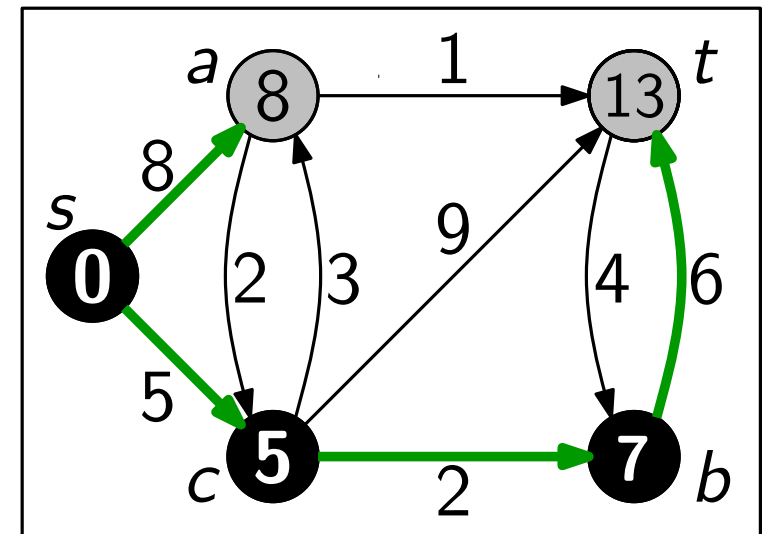
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

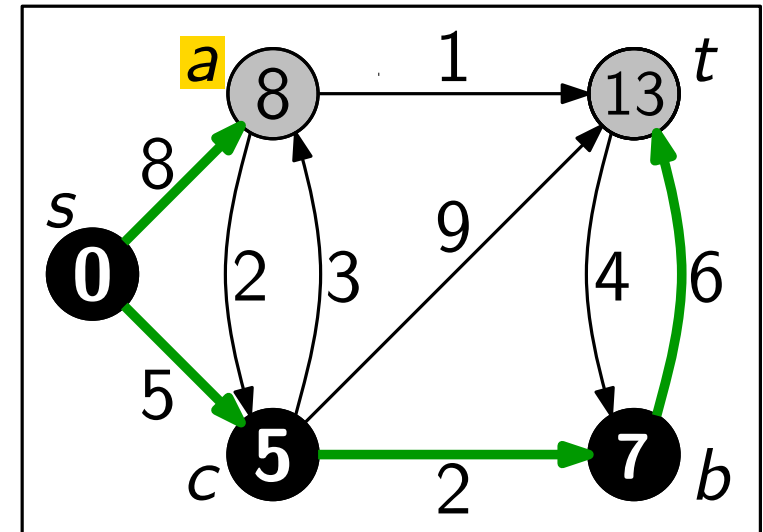
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

└ Relax($u, v; w$)

└ $u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

└ $v.\text{color} = \text{gray}$

└ $v.d = u.d + w(u, v)$

└ $v.\pi = u$

└ $Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

└ $u.\text{color} = \text{white}$

└ $u.d = \infty$

└ $u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

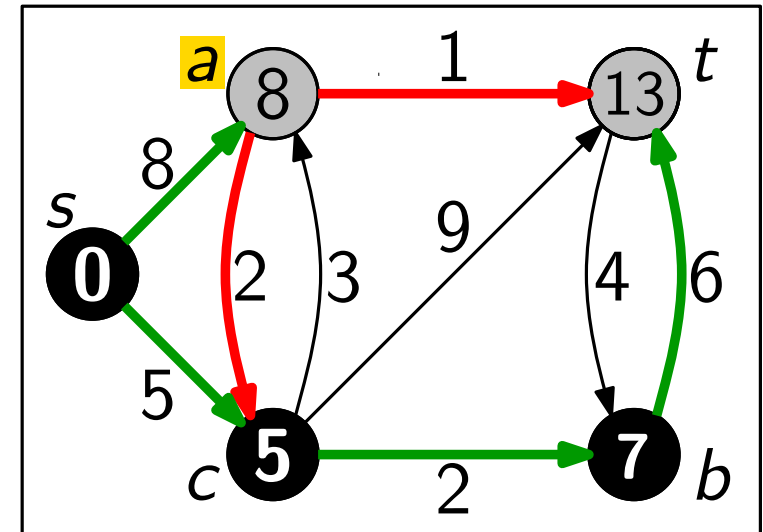
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

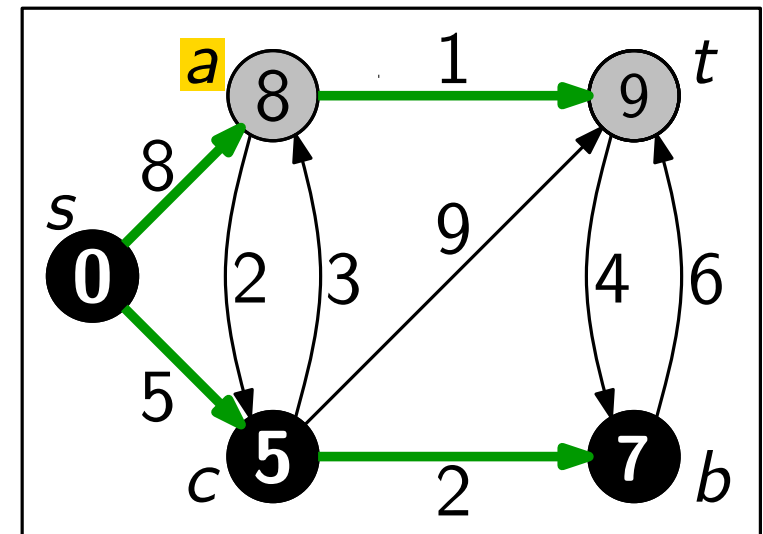
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

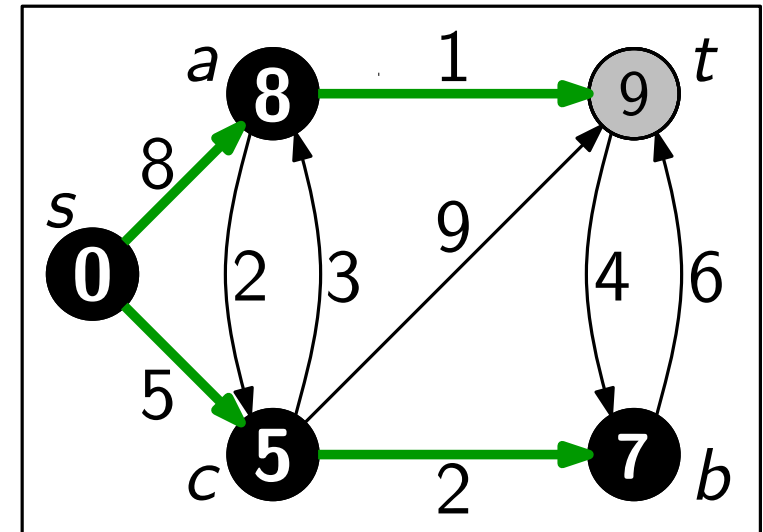
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

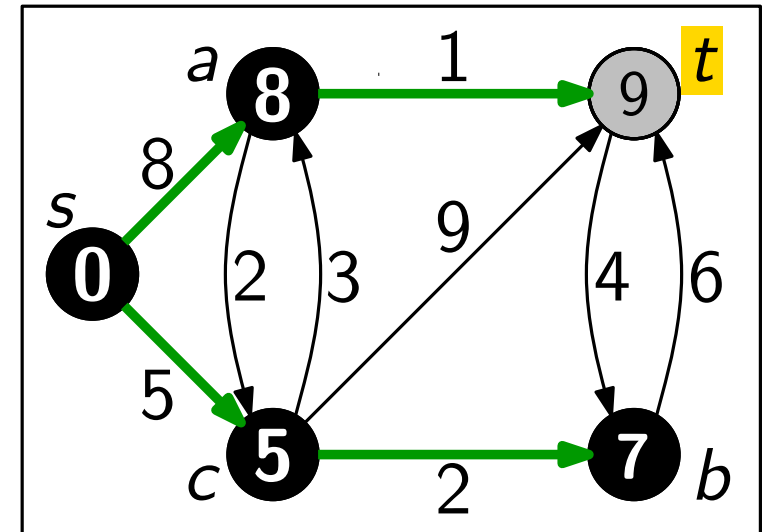
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

└ Relax($u, v; w$)

└ $u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

└ $v.\text{color} = \text{gray}$

└ $v.d = u.d + w(u, v)$

└ $v.\pi = u$

└ $Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

└ $u.\text{color} = \text{white}$

└ $u.d = \infty$

└ $u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

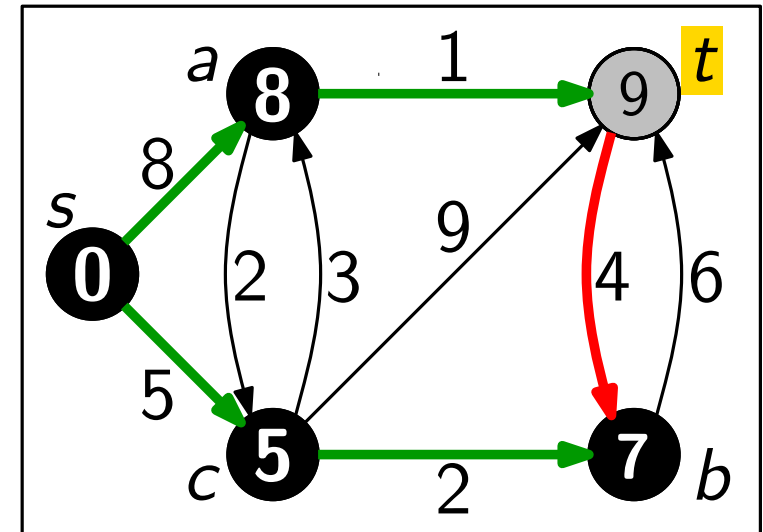
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

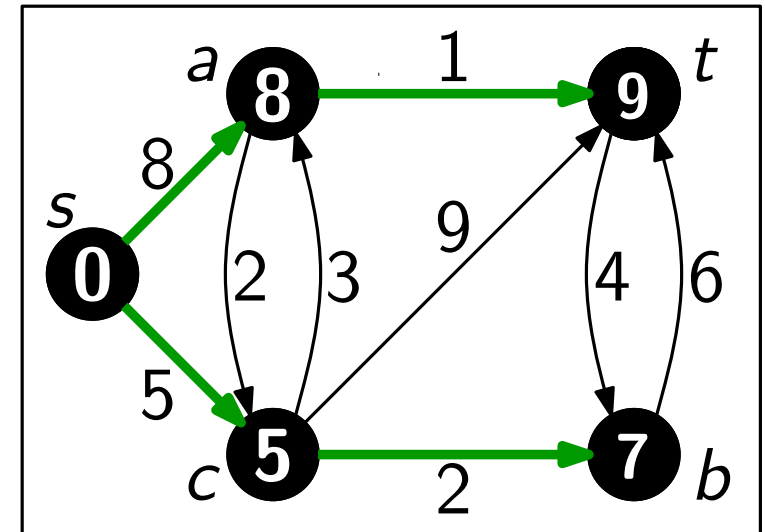
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

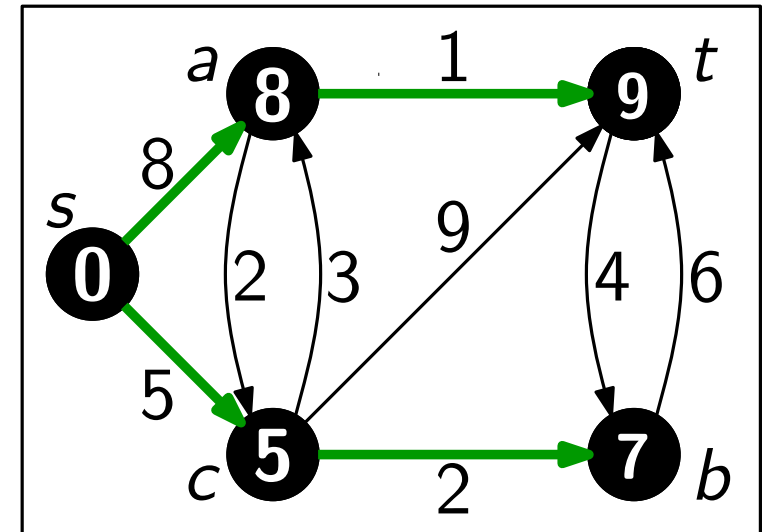
while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$



Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

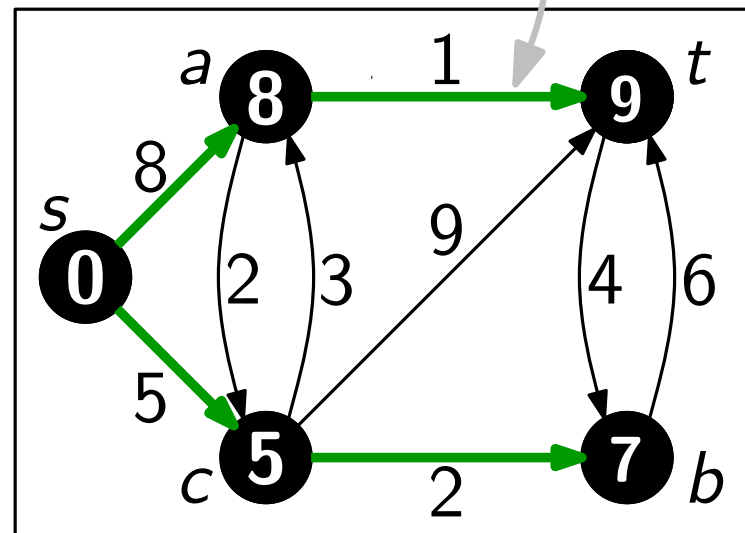
$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Kürzester-Wege-Baum
mit Wurzel s



Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \mathbf{new}$ PriorityQueue(V, d)

while not Q .Empty() **do**

$u = Q$.ExtractMin()

foreach $v \in \text{Adj}[u]$ **do**

 └ Relax($u, v; w$)

 └ $u.color = black$

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.color = gray$

$v.d = u.d + w(u, v)$

$v.\pi = u$

 └ Q .DecreaseKey($v, v.d$)

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

\perp Relax($u, v; w$)

$u.\text{color} = \text{black}$

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

\perp Relax($u, v; w$)

$u.\text{color} = \text{black}$

$O(\quad)$ Zeit

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

\perp Relax($u, v; w$)

$u.\text{color} = \text{black}$

$O(V)$ Zeit

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

$\lfloor \text{Relax}(u, v; w)$

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 └ Relax($u, v; w$)

 └ $u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

 └ $v.\text{color} = \text{gray}$

 └ $v.d = u.d + w(u, v)$

 └ $v.\pi = u$

 └ $Q.\text{DecreaseKey}(v, v.d)$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

Für jeden Knoten $u \in V$
genau $|\text{Adj}[u]|$ mal,

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Für jeden Knoten $u \in V$
genau $|\text{Adj}[u]| = \text{deg } u$ mal,
(out-)

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Für jeden Knoten $u \in V$
genau $|\text{Adj}[u]| = \text{deg } u$ mal,
(out-)
also insg. $\Theta(E)$ mal.

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

Für jeden Knoten $u \in V$
genau $|\text{Adj}[u]| = \text{deg } u$ mal,
^(out-)
also insg. $\Theta(E)$ mal.

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Also wird DecreaseKey
mal aufgerufen.

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Für jeden Knoten $u \in V$
genau $|\text{Adj}[u]| = \text{deg } u$ mal,
^(out-)
also insg. $\Theta(E)$ mal.

Also wird DecreaseKey
 $O(E)$ mal aufgerufen.

Dijkstra – die Laufzeit

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G, s)

$Q = \text{new PriorityQueue}(V, d)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{ExtractMin}()$

foreach $v \in \text{Adj}[u]$ **do**

 Relax($u, v; w$)

$u.\text{color} = \text{black}$

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

Relax($u, v; w$)

if $v.d > u.d + w(u, v)$ **then**

$v.\text{color} = \text{gray}$

$v.d = u.d + w(u, v)$

$v.\pi = u$

$Q.\text{DecreaseKey}(v, v.d)$

Für jeden Knoten $u \in V$
genau $|\text{Adj}[u]| = \text{deg } u$ mal,
(out-)
also insg. $\Theta(E)$ mal.

Also wird DecreaseKey
 $O(E)$ mal aufgerufen.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|--|----------------------------|-----------------------------|---------------------------------|
| | | | |

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

$n = (\text{max.})$ Anzahl der Elemente in der PriorityQueue

Implementierung
einer PriorityQueue

$T_{\text{ExtractMin}}(n)$

$T_{\text{DecreaseKey}}(n)$

$T_{\text{Dijkstra}}(|V|, |E|)$

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|--|----------------------------|-----------------------------|---------------------------------|
| | | | |

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | | | |

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | | |

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | |

*) Das geht, weil

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | |

*) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (→ Direktzugriff)

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |

*) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (→ Direktzugriff)

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | | | |

*) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | | |

*) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (→ Direktzugriff)

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | |

^{*}) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

^{**}) Das geht, obwohl

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | |

^{*}) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

^{**}) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E + V) \log V)$ |

^{*}) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

^{**}) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|----------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E + V) \log V)$ |
| als Fibonacci-Heap | | | |

^{*}) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

^{**}) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|-----------------------------------|-----------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E + V) \log V)$ |
| als Fibonacci-Heap | $O(\log n)$ <i>amortisiert</i> | | |

^{*}) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

^{**}) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|-----------------------------------|------------------------------|---------------------------------|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E + V) \log V)$ |
| als Fibonacci-Heap | $O(\log n)$ <i>amortisiert</i> | $O(1)$ <i>amortisiert</i> | |

^{*}) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

^{**}) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|-----------------------------------|------------------------------|--|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E+V) \log V)$ |
| als Fibonacci-Heap | $O(\log n)$ <i>amortisiert</i> | $O(1)$ <i>amortisiert</i> | $O(E + V \log V)$ <i>im Worst-Case!</i> |

^{*}) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

^{**}) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|-----------------------------------|------------------------------|--|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E+V) \log V)$ |
| als Fibonacci-Heap | $O(\log n)$ <i>amortisiert</i> | $O(1)$ <i>amortisiert</i> | $O(E + V \log V)$ <i>im Worst-Case!</i> |

*) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

**) Das geht, obwohl die Heap-Operation nicht suchbar ist. Wir markieren uns ständig jeden Knoten, wo er im Heap steht.

siehe Seminar *Entwurf und Analyse von Datenstrukturen!*

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|-----------------------------------|------------------------------|--|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E+V) \log V)$ |
| als Fibonacci-Heap | $O(\log n)$ <i>amortisiert</i> | $O(1)$ <i>amortisiert</i> | $O(E + V \log V)$ <i>im Worst-Case!</i> |

^{*}) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

^{**}) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|-----------------------------------|------------------------------|--|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E + V) \log V)$ |
| als Fibonacci-Heap | $O(\log n)$ <i>amortisiert</i> | $O(1)$ <i>amortisiert</i> | $O(E + V \log V)$ <i>im Worst-Case!</i> |

*) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

**) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Korollar. In einem Graphen $G = (V, E; w)$ mit $w: E \rightarrow \mathbb{Q}_{\geq 0}$ kann man in $O(E + V \log V)$ Zeit die kürzesten Wege von einem zu allen Knoten berechnen (SSSP-Problem).

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstra's Alg. in $O(V \cdot T_{\text{ExtractMin}}(|V|) + E \cdot T_{\text{DecreaseKey}}(|V|))$ Zeit.

| Implementierung einer PriorityQueue | $T_{\text{ExtractMin}}(n)$ | $T_{\text{DecreaseKey}}(n)$ | $T_{\text{Dijkstra}}(V , E)$ |
|-------------------------------------|-----------------------------------|------------------------------|--|
| als unsortiertes Feld | $O(n)$ | $O(1)^*$ | $O(V^2 + E)$ |
| als Heap | $O(\log n)$ | $O(\log n)^{**}$ | $O((E + V) \log V)$ |
| als Fibonacci-Heap | $O(\log n)$ <i>amortisiert</i> | $O(1)$ <i>amortisiert</i> | $O(E + V \log V)$ <i>im Worst-Case!</i> |

*) Das geht, weil wir bei ExtractMin Lücken im Feld lassen; daher bleiben die Schlüssel an ihrem Platz (\rightarrow Direktzugriff)

***) Das geht, obwohl wir im Heap nicht suchen können (!). Wir merken uns ständig für jeden Knoten, wo er im Heap steht.

Korollar. In einem Graphen $G = (V, E; w)$ mit $w: E \rightarrow \mathbb{Q}_{\geq 0}$ kann man in $O(E + V \log V)$ Zeit die kürzesten Wege von einem zu allen Knoten berechnen (SSSP-Problem).

Dijkstra – die Korrektheit

Dijkstra – die Korrektheit

siehe [CLRS], Kapitel 24.3., Satz 24.6:
Korrektheitsbeweis mittels Schleifeninvariante.

Dijkstra – die Korrektheit

siehe [CLRS], Kapitel 24.3., Satz 24.6:
Korrektheitsbeweis mittels Schleifeninvariante.

oder

Dijkstra – die Korrektheit

siehe [CLRS], Kapitel 24.3., Satz 24.6:
Korrektheitsbeweis mittels Schleifeninvariante.

oder

MIT-Vorlesungsmitschnitt von Erik Demaine:
http://videlectures.net/mit6046jf05_demaine_lec17

Wozu kürzeste Wege? (III) – SMSen

GHI

MNO

DEF

MNO

PQRS

MNO

ABC

TUV

GHI

JKL



Wozu kürzeste Wege? (III) – SMSen

GHI

MNO

DEF

MNO

PQRS

MNO

ABC

TUV

GHI

JKL



Wozu kürzeste Wege? (III) – SMSen

GHI

MNO

DEF

MNO

PQRS

MNO

ABC

TUV

GHI

JKL

10:21 für T9



Modellierung – SMSen

└

J
K
L

T
U
V

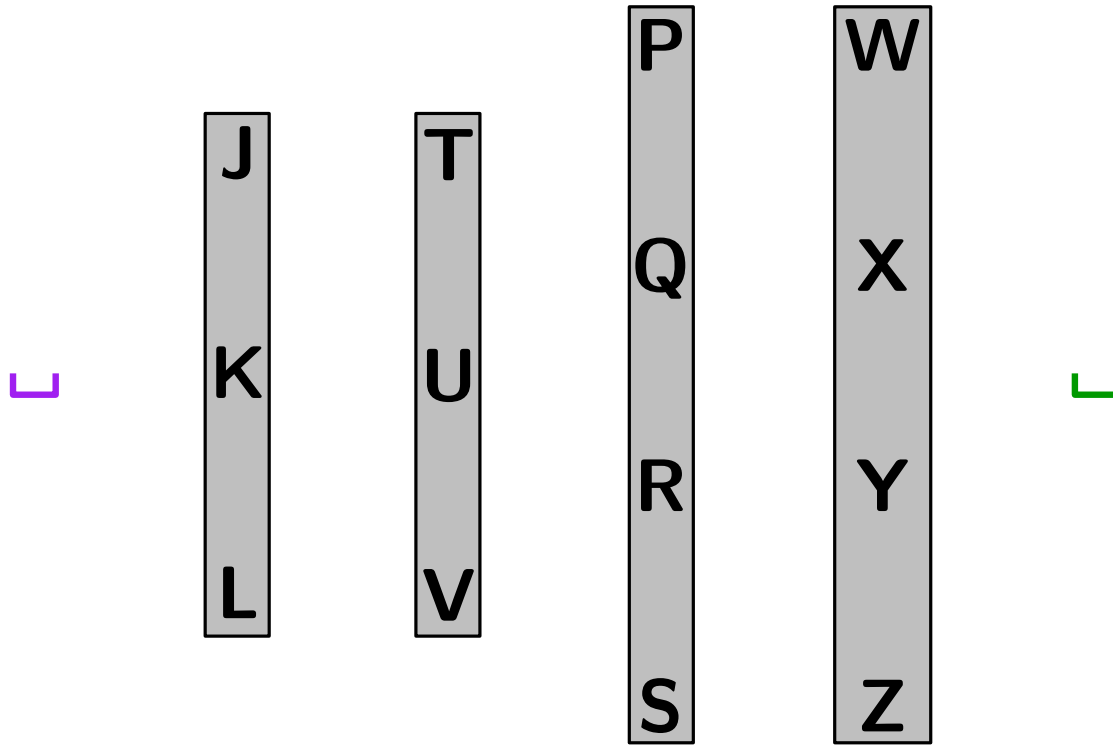
P
Q
R
S

W
X
Y
Z

└

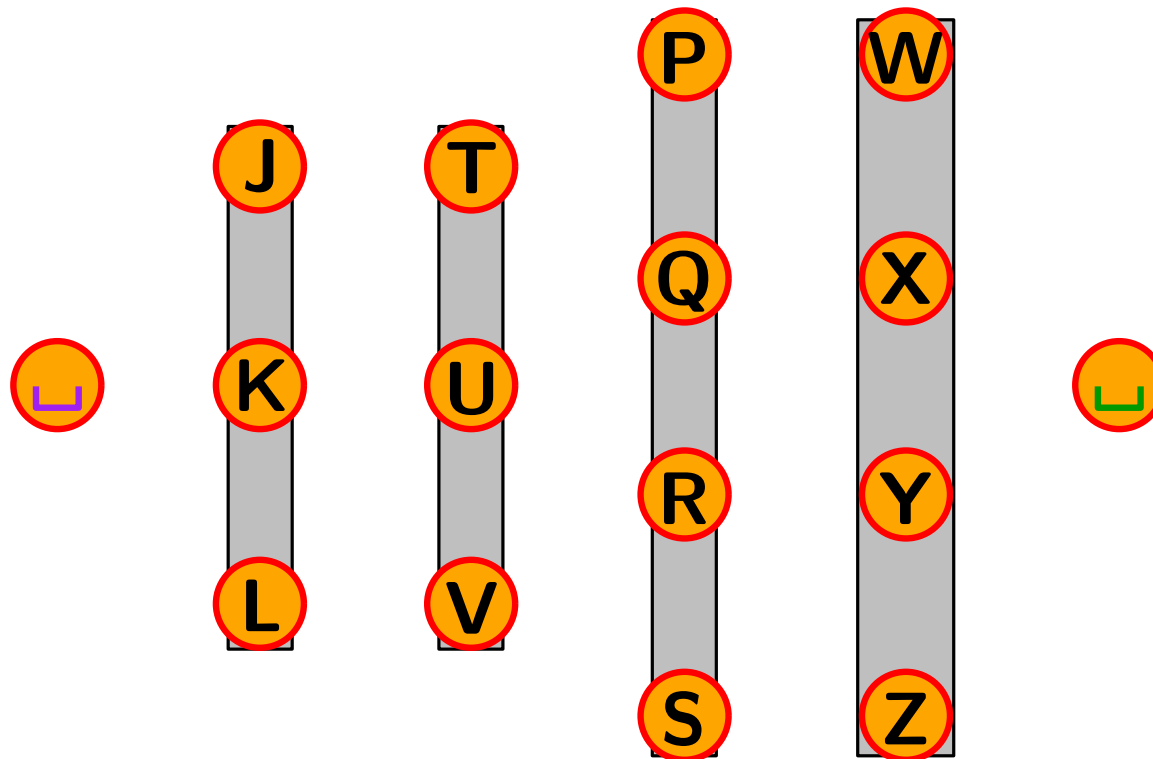


Modellierung – SMSen



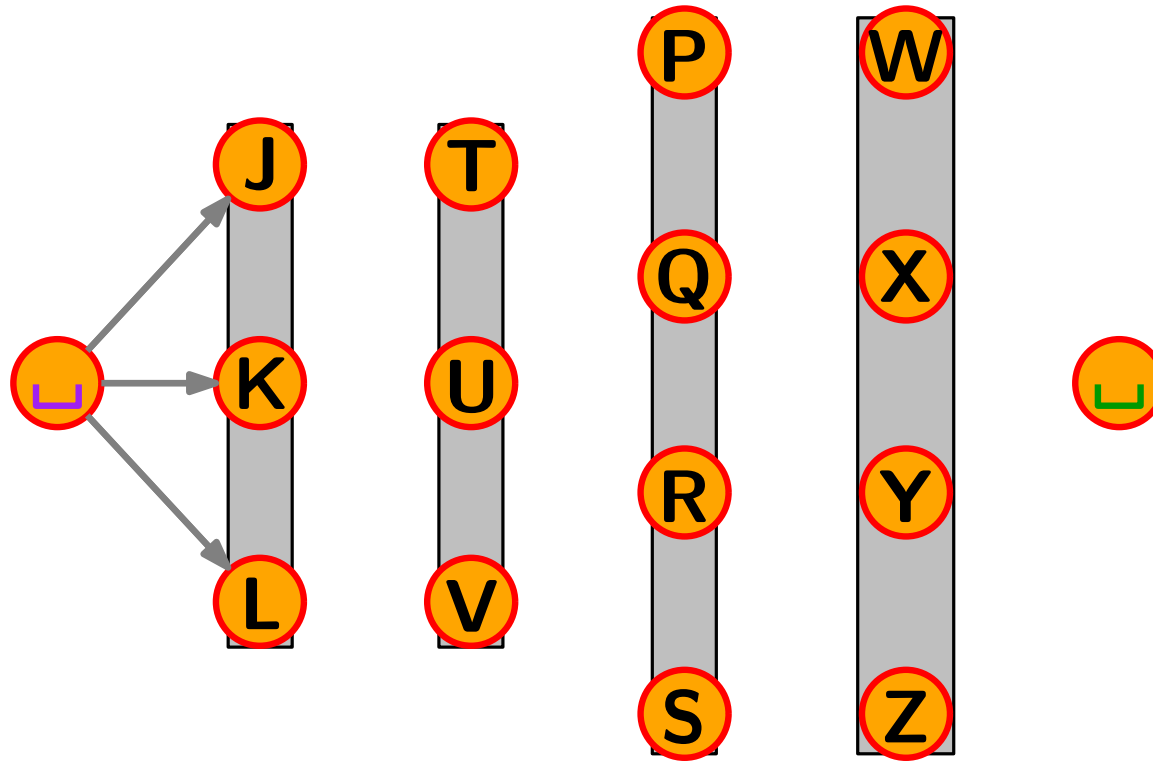
Graph:

Modellierung – SMSen



Graph: Knoten $\hat{=}$ Buchstaben

Modellierung – SMSen

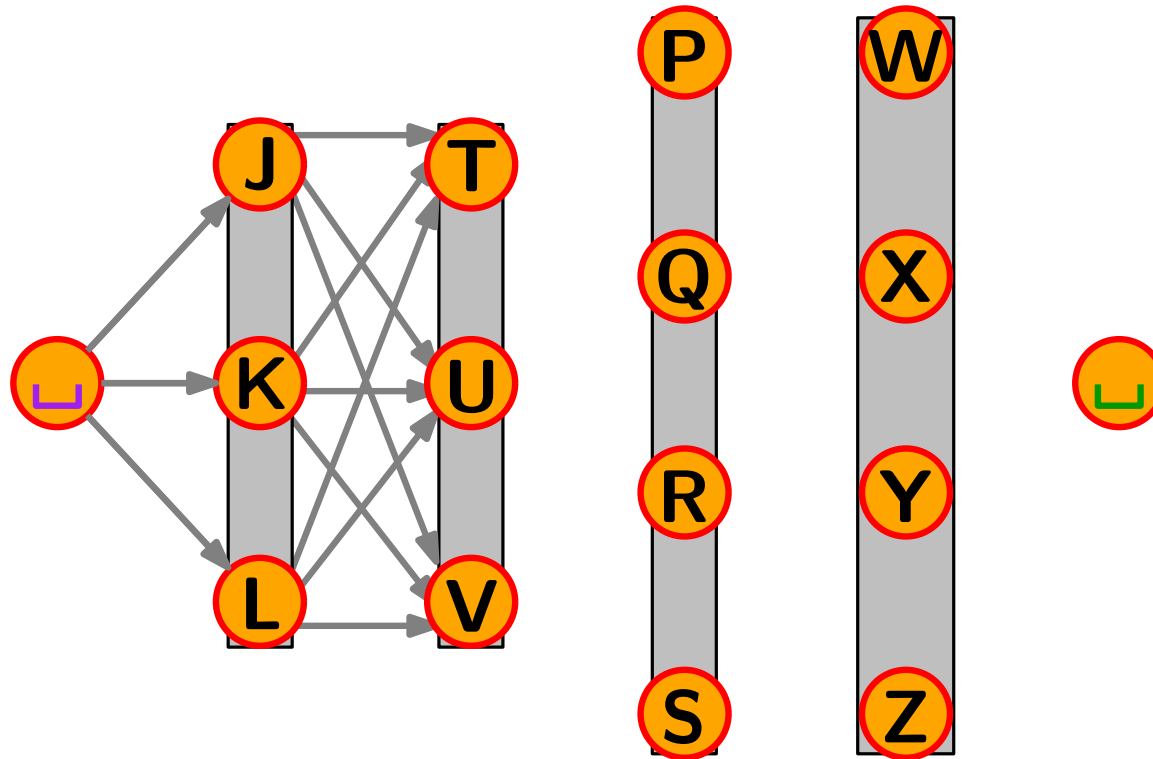


Graph:

Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Modellierung – SMSen

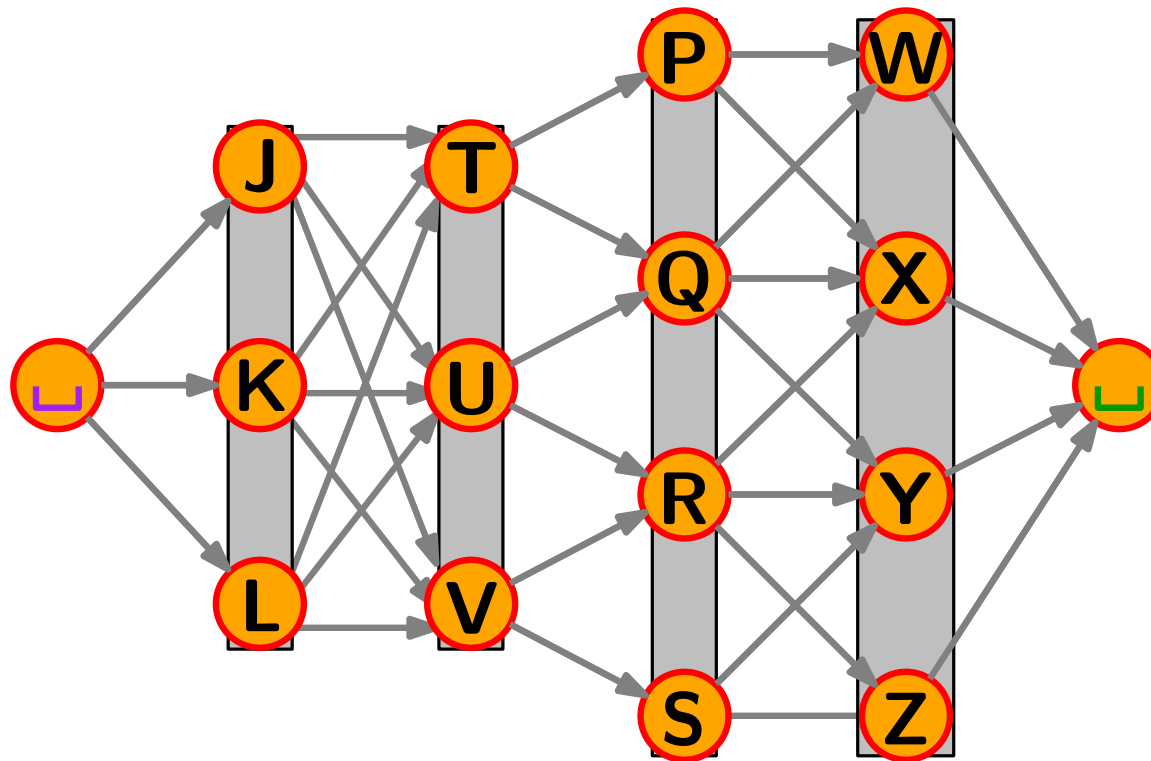


Graph:

Knoten $\hat{=}$ Buchstaben

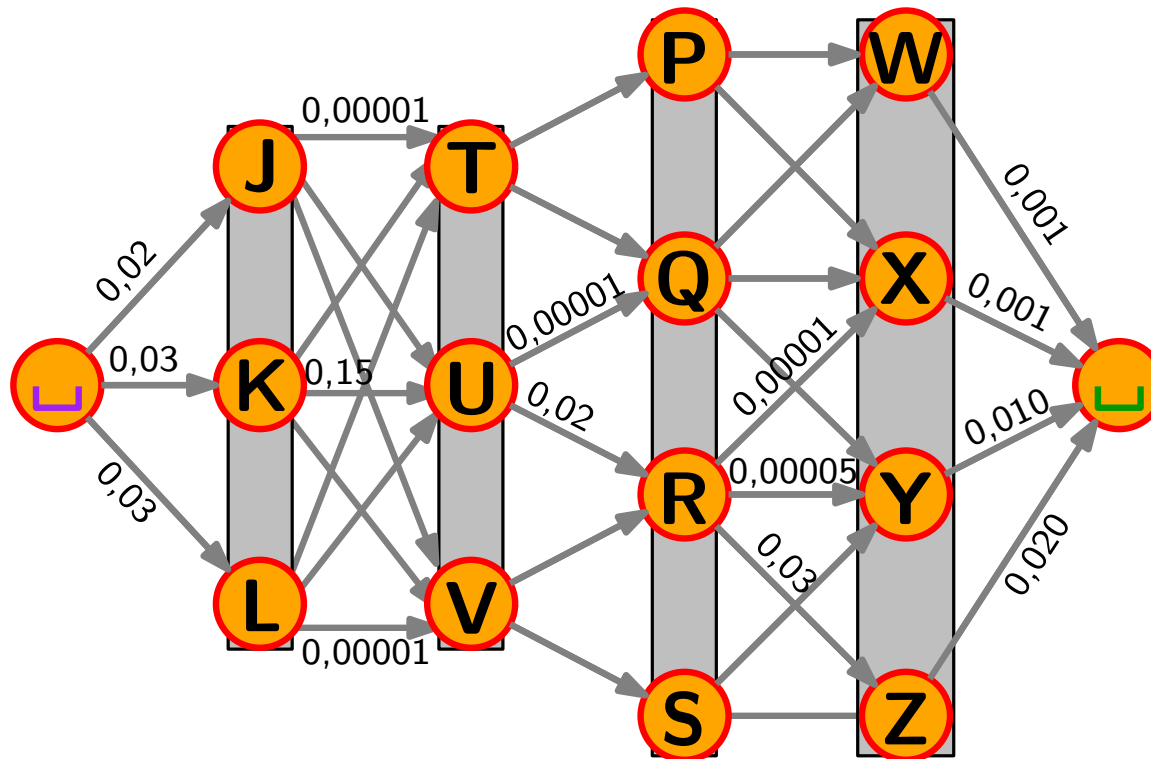
Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Modellierung – SMSen



Graph: Knoten $\hat{=}$ Buchstaben
 Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Modellierung – SMSen



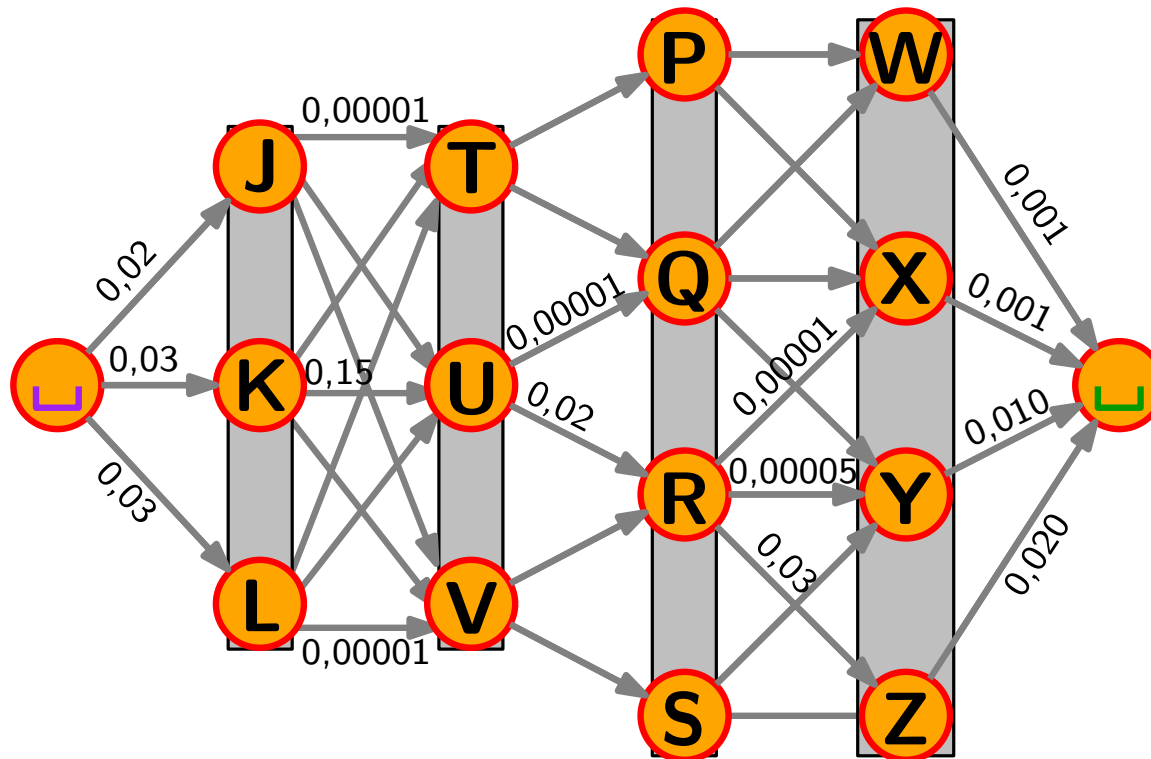
Graph:

Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Gewichte $\hat{=}$ Wahrscheinlichkeiten w

Modellierung – SMSen



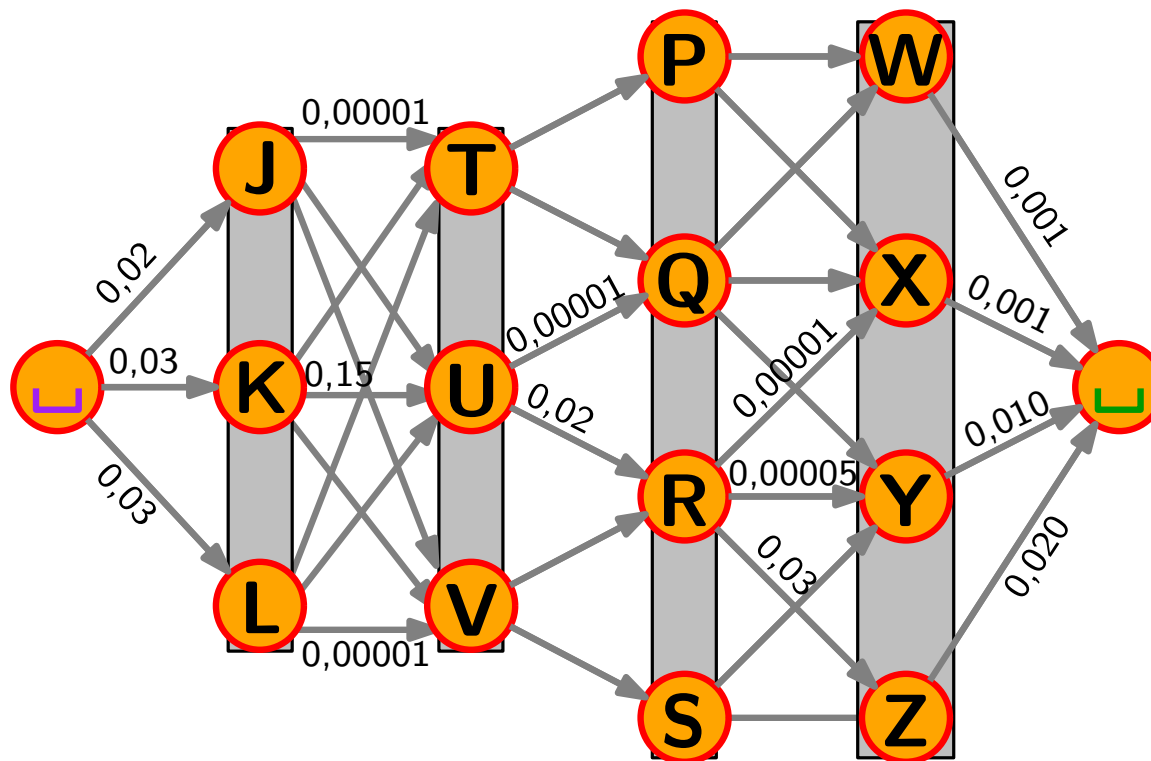
Graph:

Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Gewichte $\hat{=}$ Wahrscheinlichkeiten w / Häufigkeiten

Modellierung – SMSen



Graph:

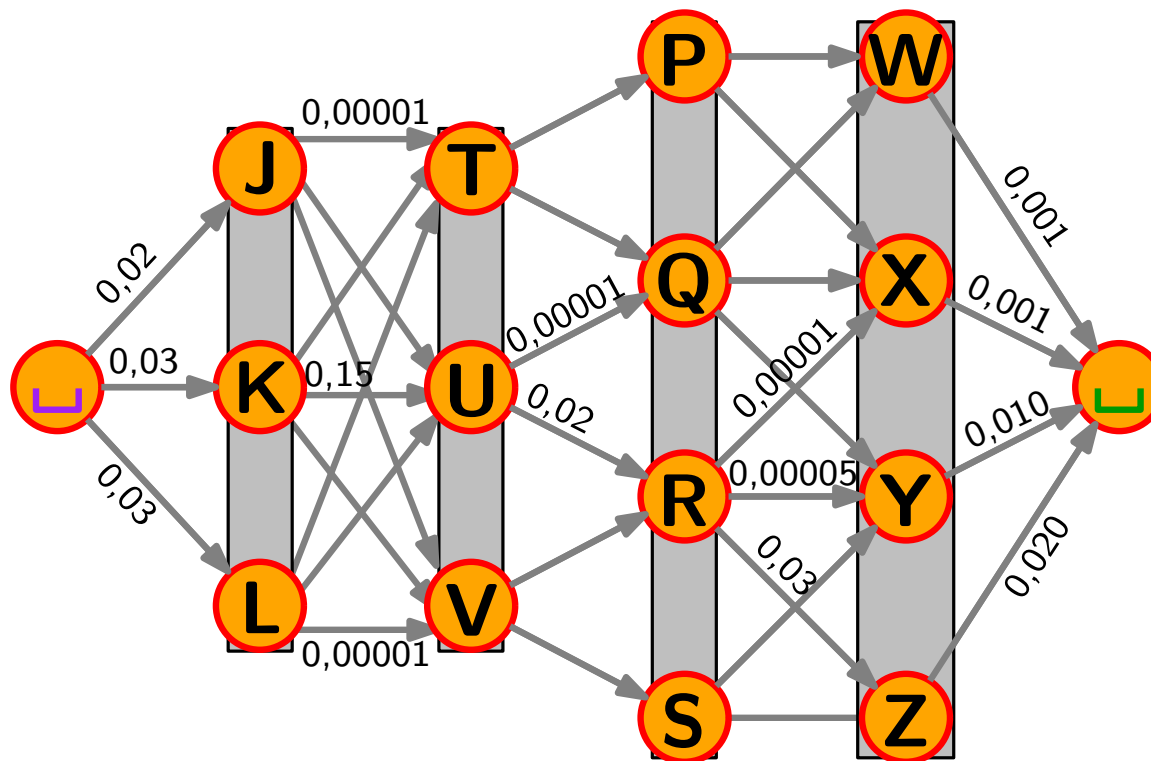
Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Gewichte $\hat{=}$ Wahrscheinlichkeiten w / Häufigkeiten

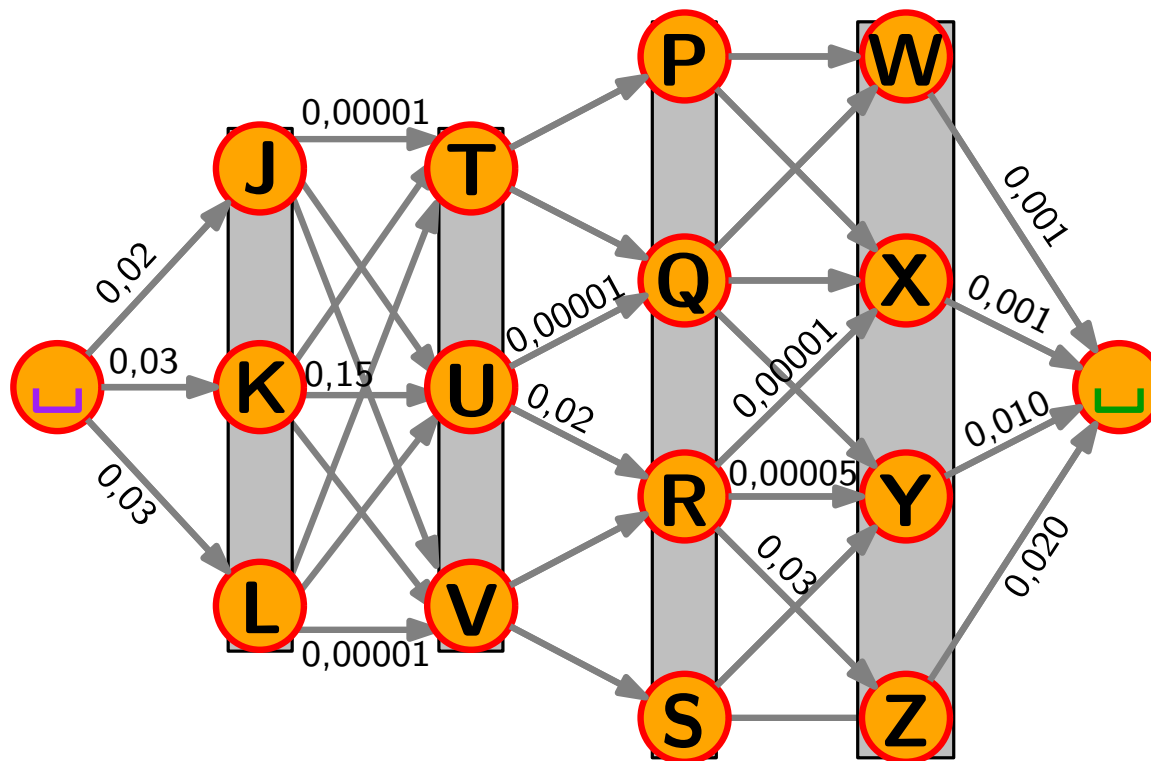
Gesucht:

Modellierung – SMSen



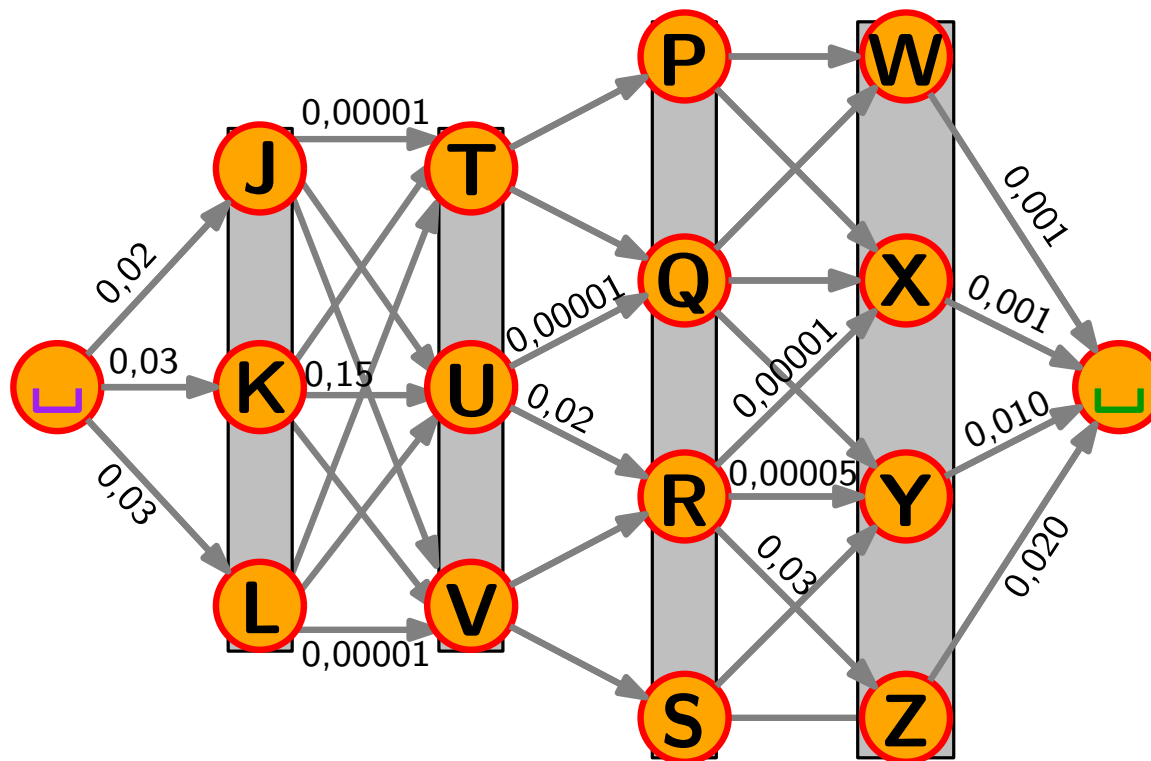
- Graph:** Knoten $\hat{=}$ Buchstaben
 Kanten $\hat{=}$ aufeinanderfolgende Buchst.
 Gewichte $\hat{=}$ Wahrscheinlichkeiten w / Häufigkeiten
- Gesucht:** Weg P von \square nach \square mit *größter* WK

Modellierung – SMSen



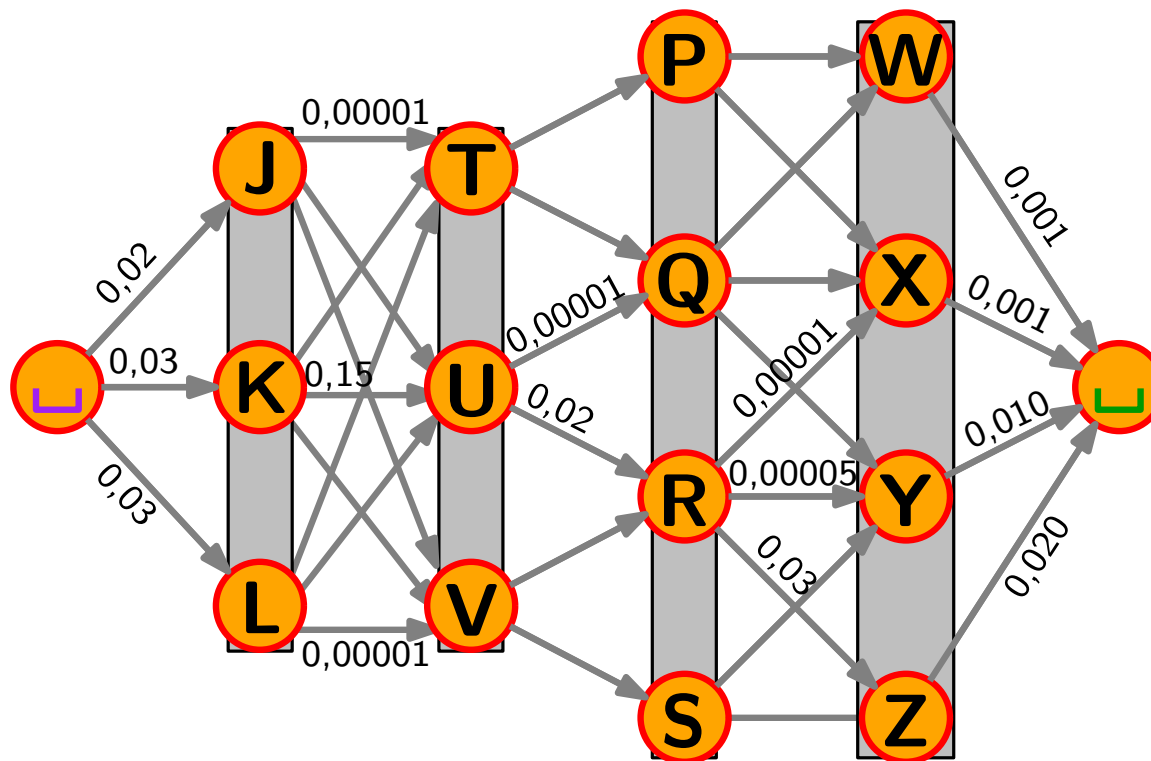
- Graph:** Knoten $\hat{=}$ Buchstaben
 Kanten $\hat{=}$ aufeinanderfolgende Buchst.
 Gewichte $\hat{=}$ Wahrscheinlichkeiten w / Häufigkeiten
- Gesucht:** Weg P von \square nach \square mit *größter* WK ($= \prod_{e \in P} w(e)$)

Modellierung – SMSen



- Graph:** Knoten $\hat{=}$ Buchstaben
 Kanten $\hat{=}$ aufeinanderfolgende Buchst.
 Gewichte $\hat{=}$ Wahrscheinlichkeiten w / Häufigkeiten
- Gesucht:** Weg P von L nach L mit *größter* WK ($= \prod_{e \in P} w(e)$)
- Lösung:** *dynamisches Programmieren...*

Modellierung – SMSen



- Graph:** Knoten $\hat{=}$ Buchstaben
 Kanten $\hat{=}$ aufeinanderfolgende Buchst.
 Gewichte $\hat{=}$ Wahrscheinlichkeiten w / Häufigkeiten
- Gesucht:** Weg P von L nach L mit *größter* WK ($= \prod_{e \in P} w(e)$)
- Lösung:** *dynamisches Programmieren...* [kommt noch!]

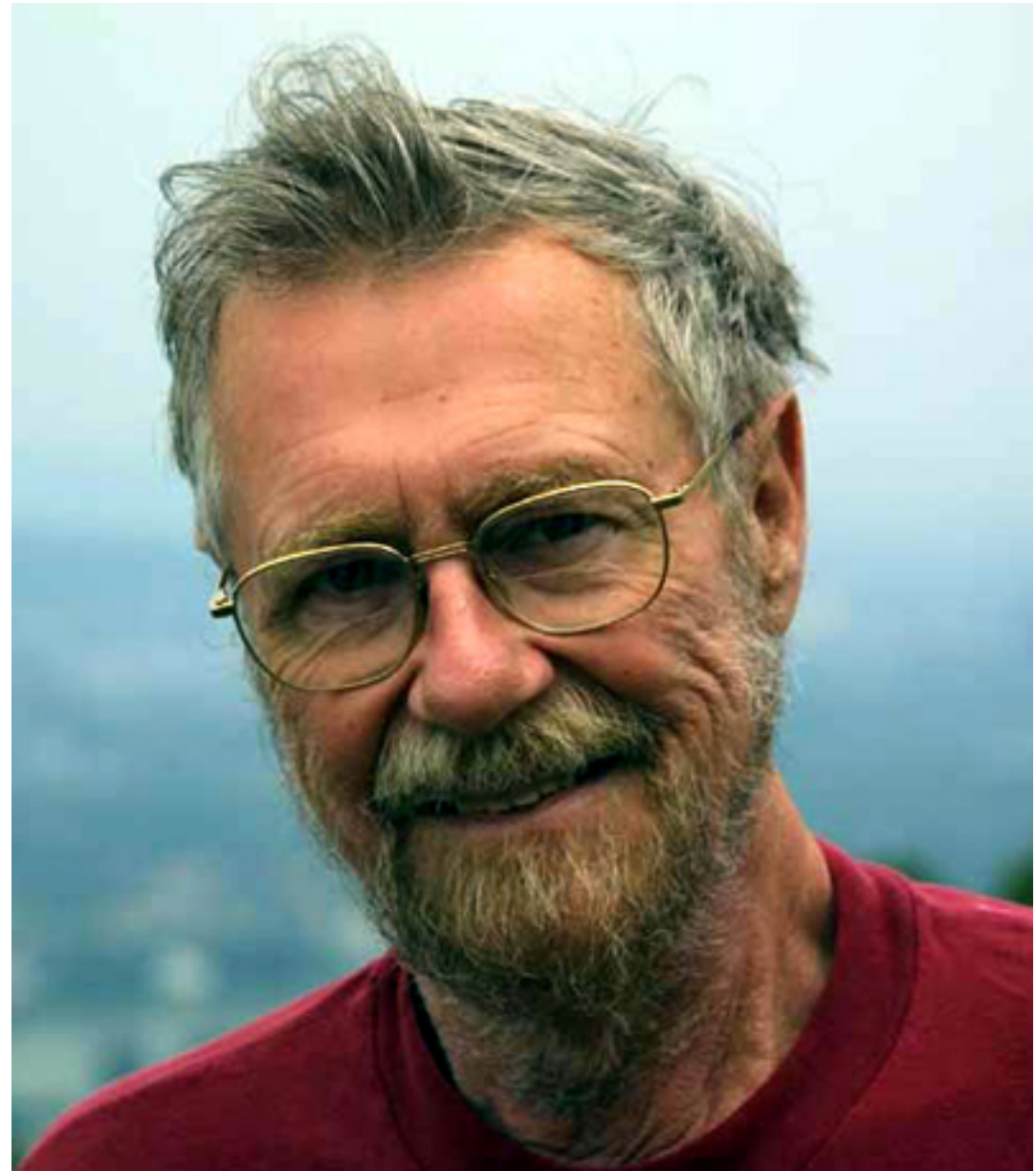
Literatur

- **A note on two problems in connexion with graphs.**

Edsger Wybe Dijkstra:

Numerische Mathematik (1)

1959, S. 269–271.



Edsger Wybe Dijkstra

* 1930 in Rotterdam

† 2002 in Nuenen, Niederlande

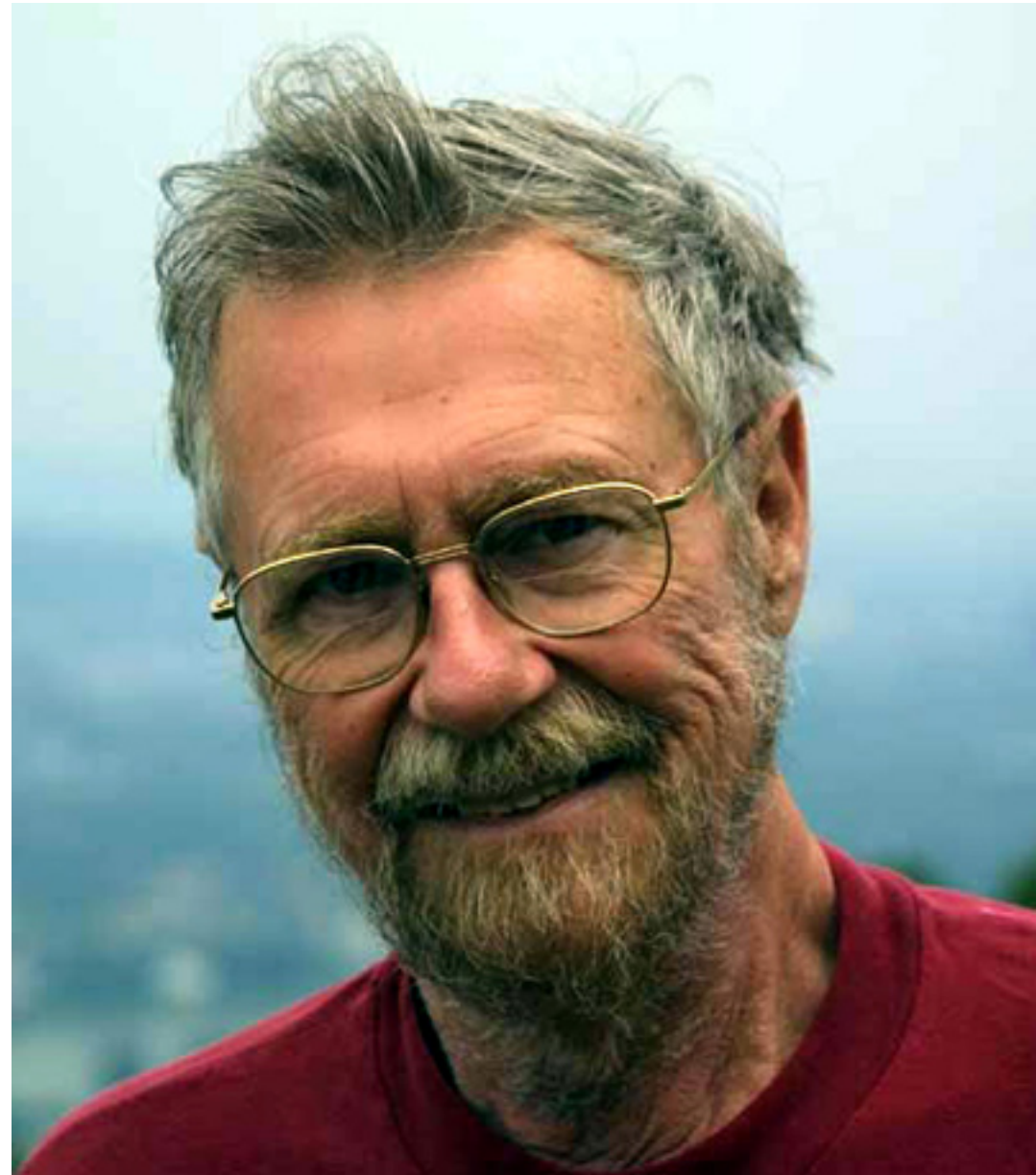
Literatur

- **A note on two problems in connexion with graphs.**

Edsger Wybe Dijkstra:
Numerische Mathematik (1)
1959, S. 269–271.

- **Das Geheimnis des kürzesten Weges.**

Ein mathematisches Abenteuer.
Peter Gritzmann und
René Brandenberg:
Springer-Verlag, 3. Aufl., 2005.



Edsger Wybe Dijkstra
* 1930 in Rotterdam
† 2002 in Nuenen, Niederlande

Literatur

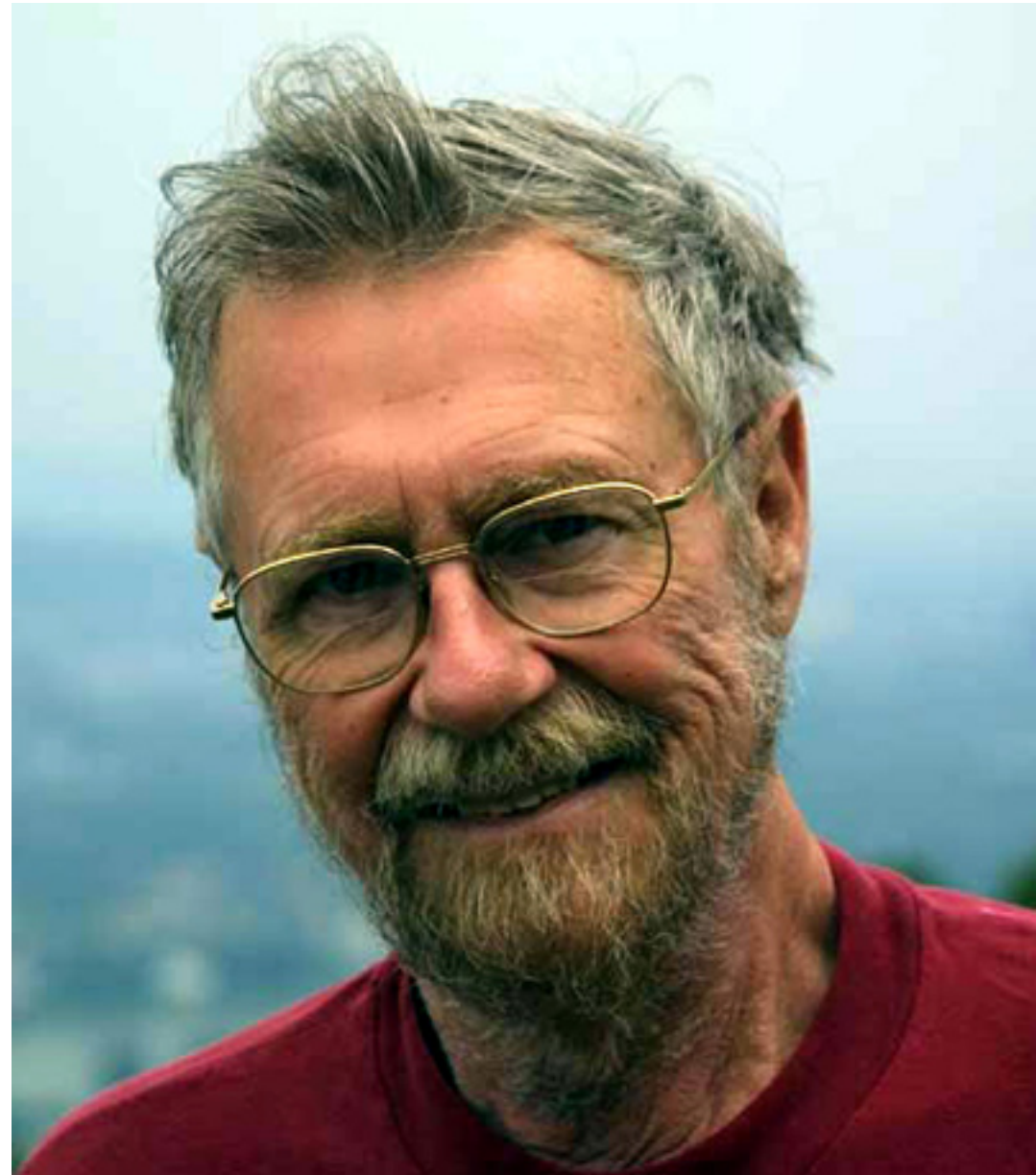
- **A note on two problems in connexion with graphs.**

Edsger Wybe Dijkstra:
Numerische Mathematik (1)
1959, S. 269–271.

- **Das Geheimnis des kürzesten Weges.**

Ein mathematisches Abenteuer.
Peter Gritzmann und
René Brandenberg:
Springer-Verlag, 3. Aufl., 2005.

Beide Werke sind über die UB
frei zugänglich und über unsere
WueCampus-Seite verlinkt!



Edsger Wybe Dijkstra
* 1930 in Rotterdam
† 2002 in Nuenen, Niederlande

Literatur

- **A note on two problems in connexion with graphs.**

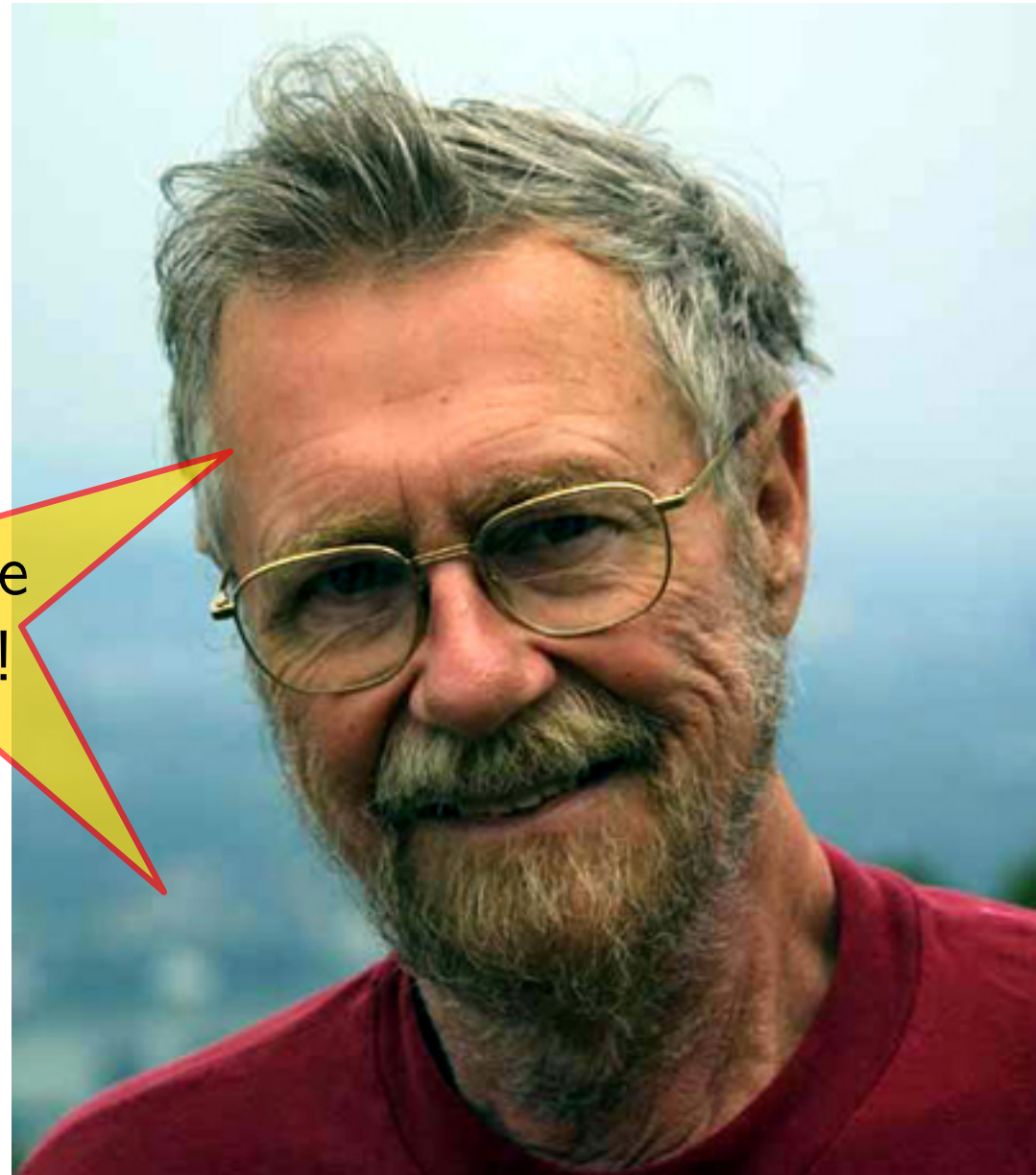
Edsger Wybe Dijkstra:
Numerische Mathematik (1)
 1959, S. 269–271.

Lesen Sie
 mal rein!

- **Das Geheimnis des kürzesten Weges.**

Ein mathematisches Abenteuer.
 Peter Gritzmann und
 René Brandenberg:
Springer-Verlag, 3. Aufl., 2005.

Beide Werke sind über die UB
frei zugänglich und über unsere
 WueCampus-Seite verlinkt!



Edsger Wybe Dijkstra
 * 1930 in Rotterdam
 † 2002 in Nuenen, Niederlande

Kürzeste Wege nach Dijkstra

| | | |
|------------------------------|---------------------|-----------------------|
| nicht-neg. Kantengew. | Dijkstra | $O(E + V \log V)$ ✓ |
| ungewichteter Graph | Breitensuche | $O(E + V)$ ✓ |
| azyklischer Graph | topol. Sortieren | $O(E + V)$ |
| negative Kantengew. | Bellman-Ford | $O(EV)$ |
| für alle Knotenpaare | $V \times$ Dijkstra | $O(V(E + V \log V))$ |
| + negative Kantengew. | Floyd-Warshall | $O(V^3)$ |
| | Johnson | $O(V(E + V \log V))$ |
| k kürzeste s - t -Wege | Eppstein | $O(k + E + V \log V)$ |

Kürzeste Wege nach Dijkstra

| | | |
|--------------------------|---------------------|----------------------------|
| nicht-neg. Kantengew. | Dijkstra | $O(E + V \log V)$ ✓ |
| ungewichteter Graph | Breitensuche | $O(E + V)$ ✓ |
| azyklischer Graph | topol. Sortieren | $O(E + V)$ Nächstes Mal! ✓ |
| negative Kantengew. | Bellman-Ford | $O(EV)$ |
| für alle Knotenpaare | $V \times$ Dijkstra | $O(V(E + V \log V))$ |
| + negative Kantengew. | Floyd-Warshall | $O(V^3)$ |
| | Johnson | $O(V(E + V \log V))$ |
| k kürzeste $s-t$ -Wege | Eppstein | $O(k + E + V \log V)$ |

Kürzeste Wege nach Dijkstra

| | | |
|--------------------------|---------------------|---|
| nicht-neg. Kantengew. | Dijkstra | $O(E + V \log V)$ ✓ |
| ungewichteter Graph | Breitensuche | $O(E + V)$ ✓ |
| azyklischer Graph | topol. Sortieren | $O(E + V)$ Nächstes Mal! ✓ |
| negative Kantengew. | Bellman-Ford | $O(EV)$ Vorlesung Alg. Graphentheorie (?) |
| für alle Knotenpaare | $V \times$ Dijkstra | $O(V(E + V \log V))$ |
| + negative Kantengew. | Floyd-Warshall | $O(V^3)$ |
| | Johnson | $O(V(E + V \log V))$ |
| k kürzeste $s-t$ -Wege | Eppstein | $O(k + E + V \log V)$ |

Kürzeste Wege nach Dijkstra

| | | |
|---|---------------------|---|
| nicht-neg. Kantengew. | Dijkstra | $O(E + V \log V)$ ✓ |
| ungewichteter Graph | Breitensuche | $O(E + V)$ ✓ |
| azyklischer Graph | topol. Sortieren | $O(E + V)$ Nächstes Mal! ✓ |
| negative Kantengew. | Bellman-Ford | $O(EV)$ Vorlesung Alg. Graphentheorie (?) |
| für alle Knotenpaare + negative Kantengew. | $V \times$ Dijkstra | $O(V(E + V \log V))$ ✓ |
| | Floyd-Warshall | $O(V^3)$ |
| | Johnson | $O(V(E + V \log V))$ |
| k kürzeste s - t -Wege | Eppstein | $O(k + E + V \log V)$ |