

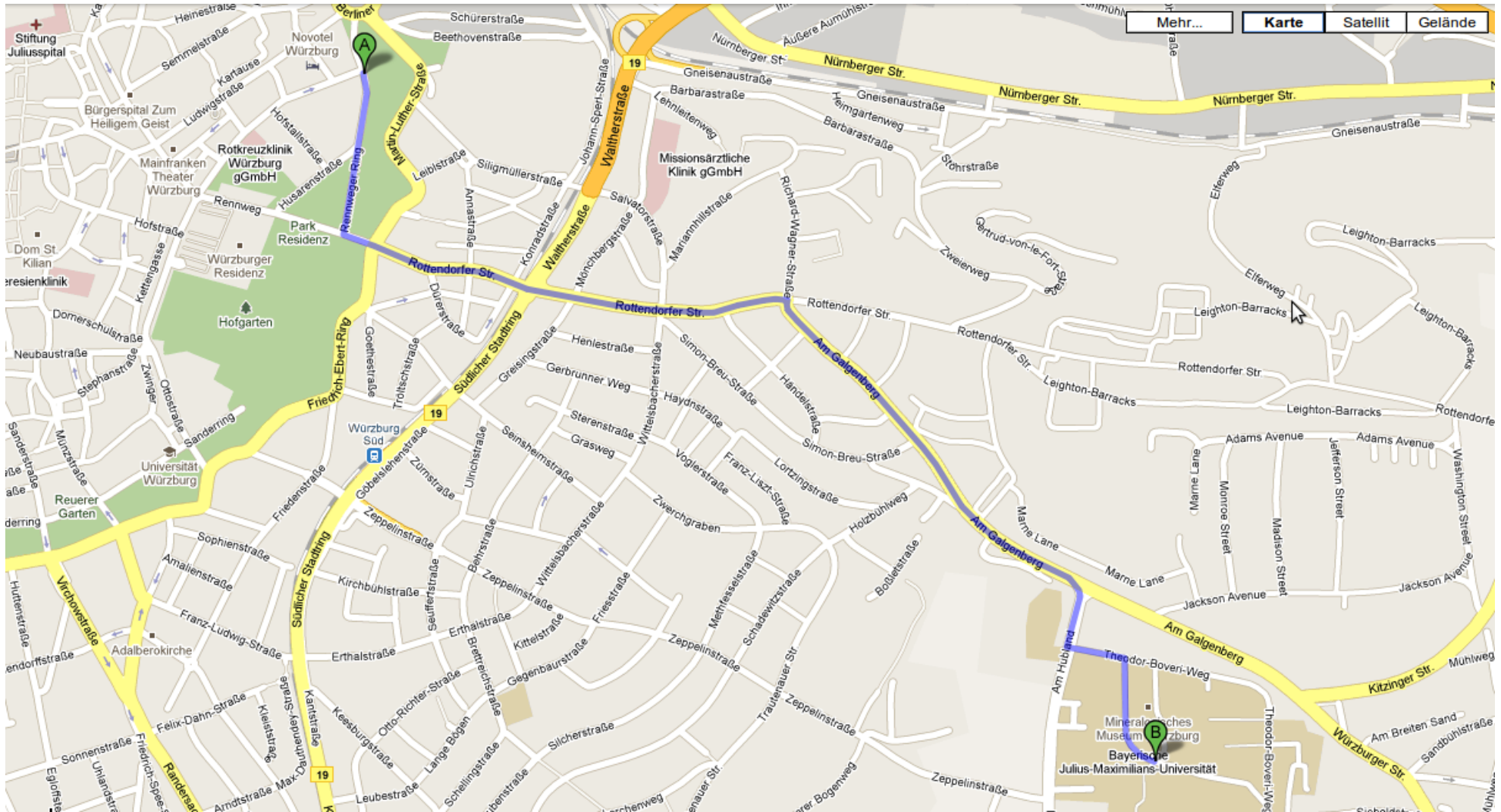
Algorithmen und Datenstrukturen

Wintersemester 2020/21

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 $w(e) \geq 0$

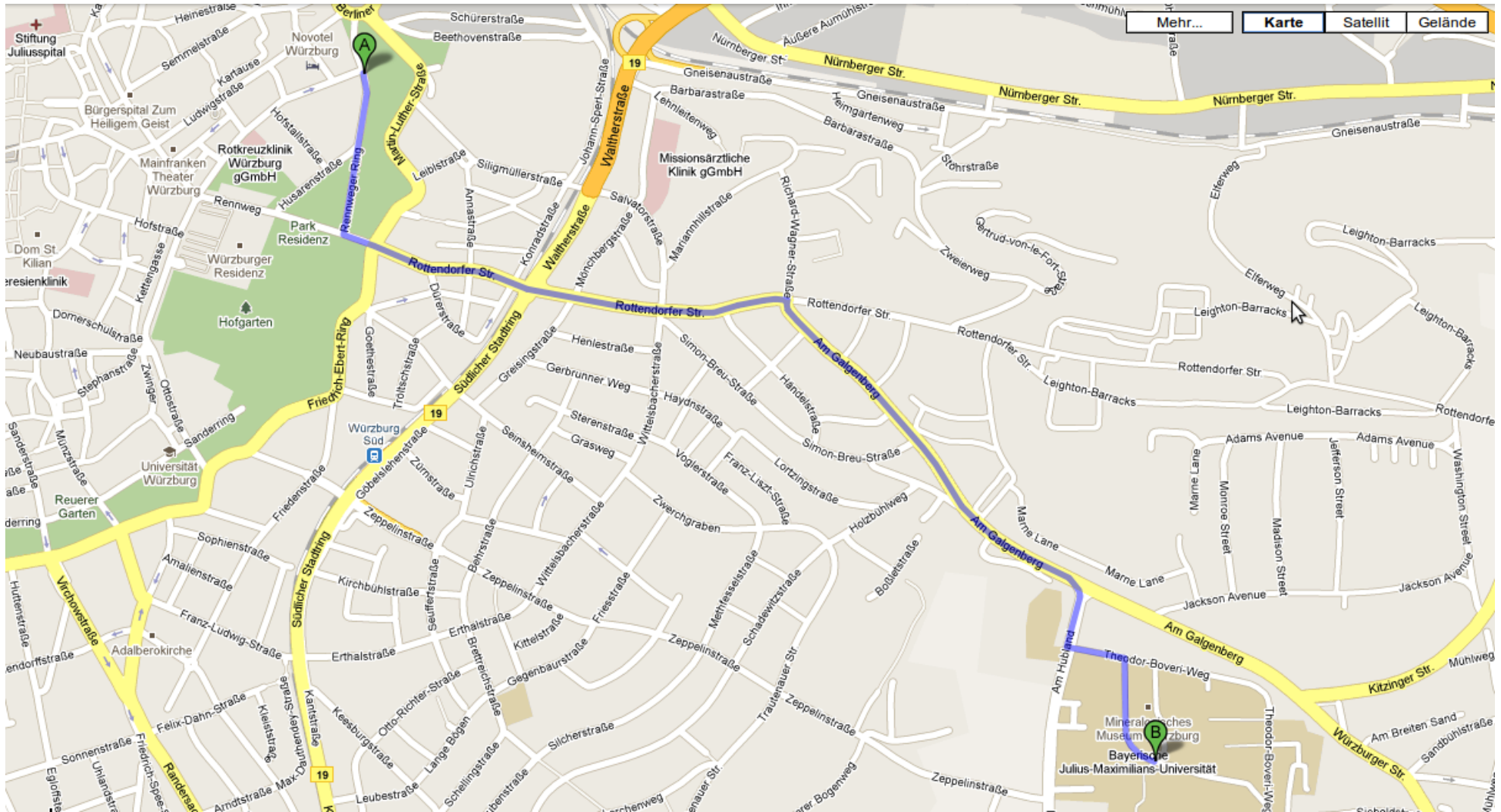
Straßennetz $\rightarrow$ gerichteter, gewichteter und zusammenhängender Graph $G = (V, E)$

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? (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

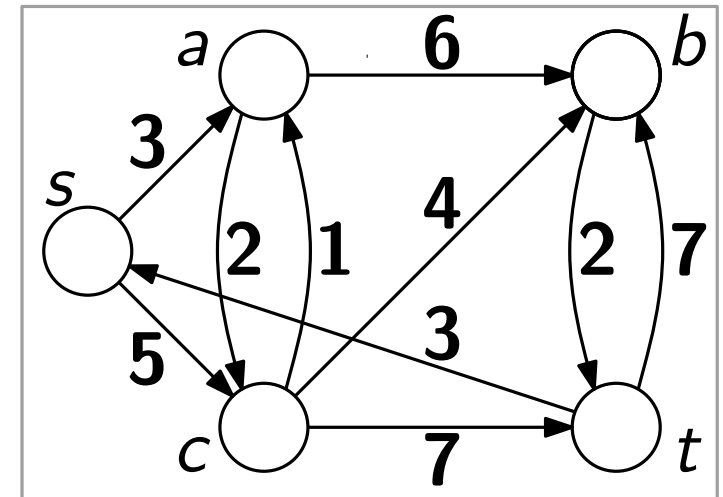
Bahnhof/Haltestelle	Datum	Zeit	Dauer	Umst.	Produkte	Normalpreis	
		↑ Früher					
 Würzburg Busbahnhof Mathematisches Institut, Würzburg	Di, 12.01.10	ab 10:29	0:16	0	Bus	Preisauskunft nicht möglich 	→ Rückfahrt hinzufügen
	Di, 12.01.10	an 10:45					
 Würzburg Busbahnhof Mathematisches Institut, Würzburg	Di, 12.01.10	ab 10:49	0:16	0	Bus	Preisauskunft nicht möglich 	→ Rückfahrt hinzufügen
	Di, 12.01.10	an 11:05					
 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					
 Details für alle anzeigen		↓ Später					
 MobilCheck	 UmweltMobilCheck						

Zurück

Was ist das Problem?

Eingabe:

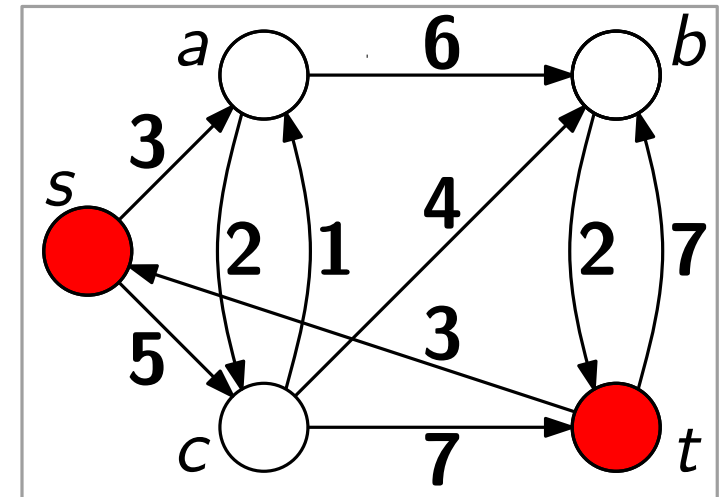
- gerichteter, zusammenhängender Graph $G = (V, E)$
mit nicht-negativen **Kantengewichten** $w: E \rightarrow \mathbb{Q}_0^+$,



Was ist das Problem?

Eingabe:

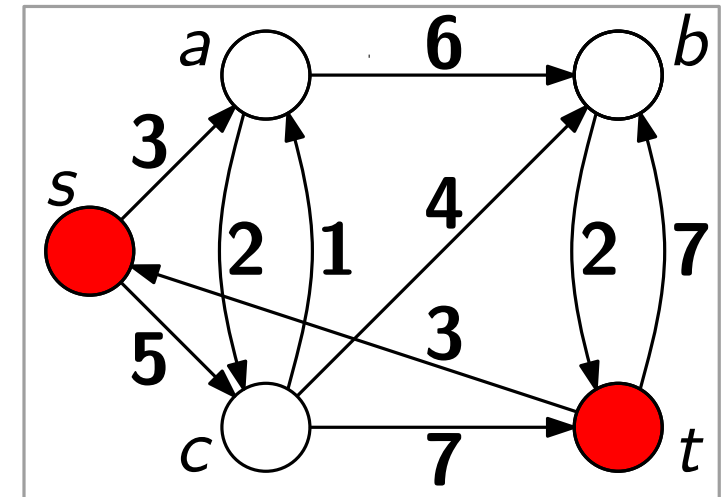
- gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen **Kantengewichten** $w: E \rightarrow \mathbb{Q}_0^+$,
- Knoten s und t



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:

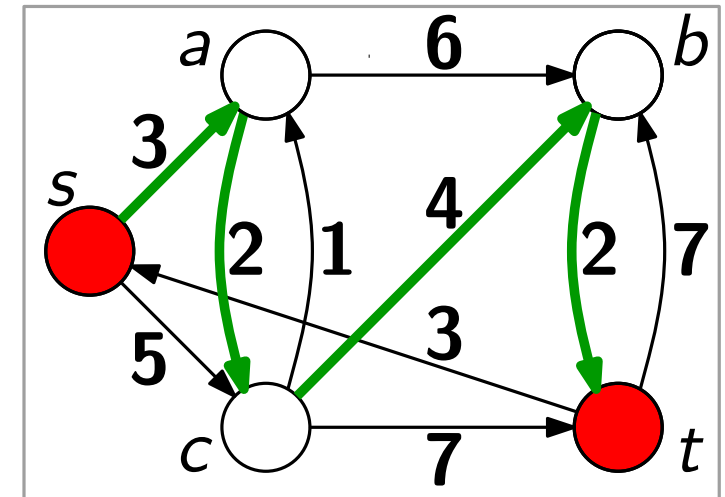
- kürzester s - t -Weg** W in G , d.h. $\sum_{e \in W} w(e)$ minimal.

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:

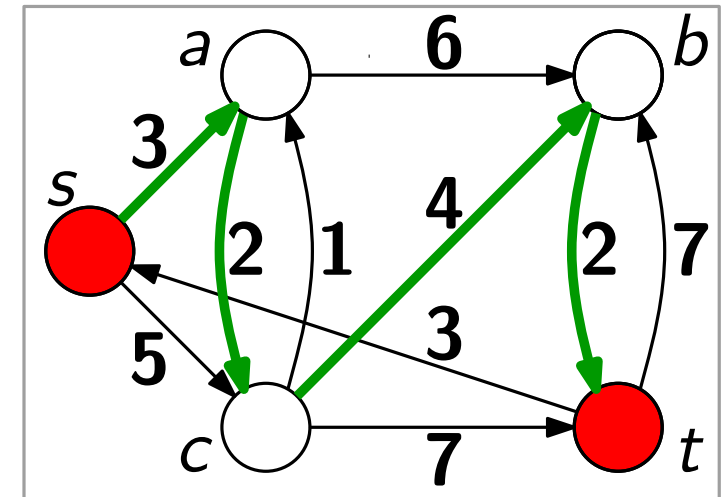
- **kürzester s - t -Weg** W in G , d.h. $\sum_{e \in W} w(e)$ minimal.

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:

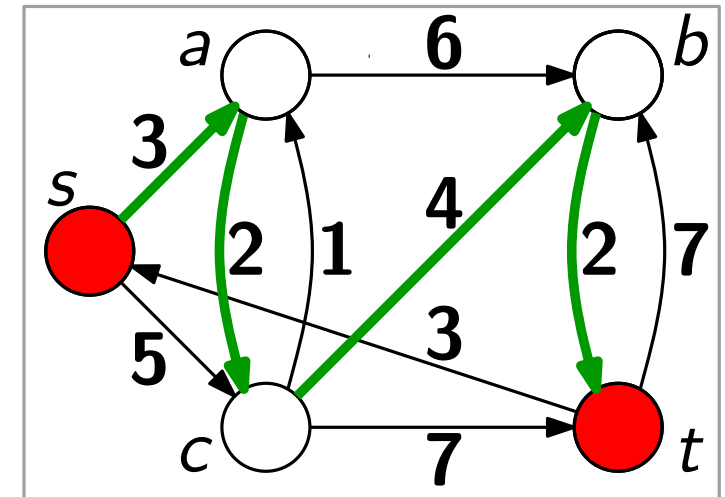
- ~~kürzester~~ s - t -Weg W_t in G , $\underbrace{\text{für alle } t \in V}_{\text{für alle } t \in V}$ d.h. $\sum_{e \in W} w(e)$ minimal.

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

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)

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

$Q.\text{Enqueue}(s)$

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

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

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

if $v.\text{color} == \text{white}$ **then**

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

$v.d = u.d + 1$

$v.\pi = u$

$Q.\text{Enqueue}(v)$

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

Dijkstra – BFS mit Gewichten

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

Initialize(G, s)

// Gewichtung

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

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

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

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

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

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

BFS(Graph G , Vertex s)

Initialize(G, s)

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

$Q.\text{Enqueue}(s)$

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

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

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

if $v.\text{color} == \text{white}$ **then**

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

$v.d = u.d + 1$

$v.\pi = u$

$Q.\text{Enqueue}(v)$

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

Dijkstra – BFS mit Gewichten

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

Initialize(G, s)

// Gewichtung

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

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

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

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

 Relax($u, v; w$)

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

BFS(Graph G , Vertex s)

Initialize(G, s)

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

$Q.\text{Enqueue}(s)$

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

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

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

if $v.\text{color} == \text{white}$ then

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

$v.d = u.d + 1$

$v.\pi = u$

$Q.\text{Enqueue}(v)$

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

Dijkstra – BFS mit Gewichten

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

Initialize(G, s)

// Gewichtung

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

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

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

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

 Relax($u, v; w$)

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

BFS(Graph G , Vertex s)

Initialize(G, s)

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

$Q.\text{Enqueue}(s)$

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

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

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

if $v.\text{color} == \text{white}$ **then**

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

$v.d = u.d + 1$

$v.\pi = u$

$Q.\text{Enqueue}(v)$

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

Dijkstra – BFS mit Gewichten

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

Initialize(G, s)

// Gewichtung

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

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

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

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

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

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

BFS(Graph G , Vertex s)

Initialize(G, s)

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

$Q.\text{Enqueue}(s)$

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

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

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

if $v.\text{color} == \text{white}$ **then**

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

$v.d = u.d + 1$

$v.\pi = u$

$Q.\text{Enqueue}(v)$

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

Dijkstra – BFS mit Gewichten

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

Initialize(G, s)

// Gewichtung

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

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

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

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

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

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

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

Was muss in dieser
Unterroutine passieren?

Schreiben Sie's auf!

BFS(Graph G , Vertex s)

Initialize(G, s)

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

$Q.\text{Enqueue}(s)$

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

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

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

if $v.\text{color} == \text{white}$ **then**

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

$v.d = u.d + 1$

$v.\pi = u$

$Q.\text{Enqueue}(v)$

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

Dijkstra – BFS mit Gewichten

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

Initialize(G, s)

// Gewichtung

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

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**

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

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

$v.\pi = u$

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

BFS(Graph G , Vertex s)

Initialize(G, s)

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

$Q.\text{Enqueue}(s)$

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

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

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

if $v.\text{color} == \text{white}$ **then**

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

$v.d = u.d + 1$

$v.\pi = u$

$Q.\text{Enqueue}(v)$

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

Dijkstra – ein Beispiel

Dijkstra(WeightedGraph G , Vertex s)

Initialize(G , s)

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

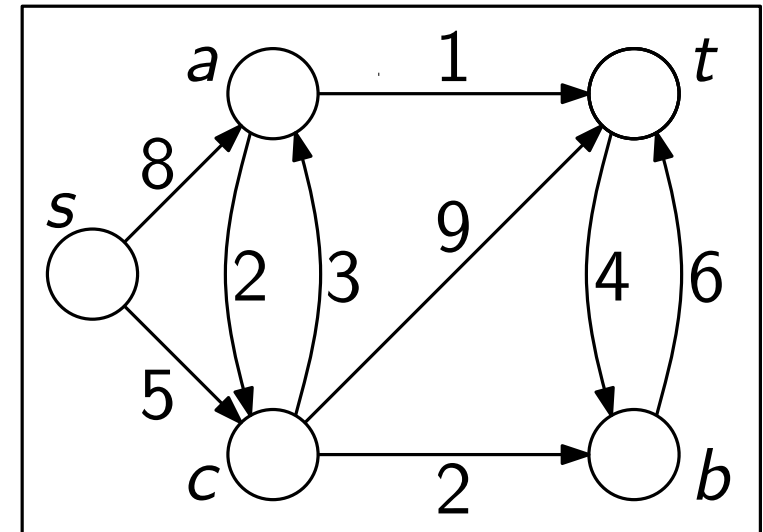
while not $Q.\text{Empty}()$ **do**

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

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)

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

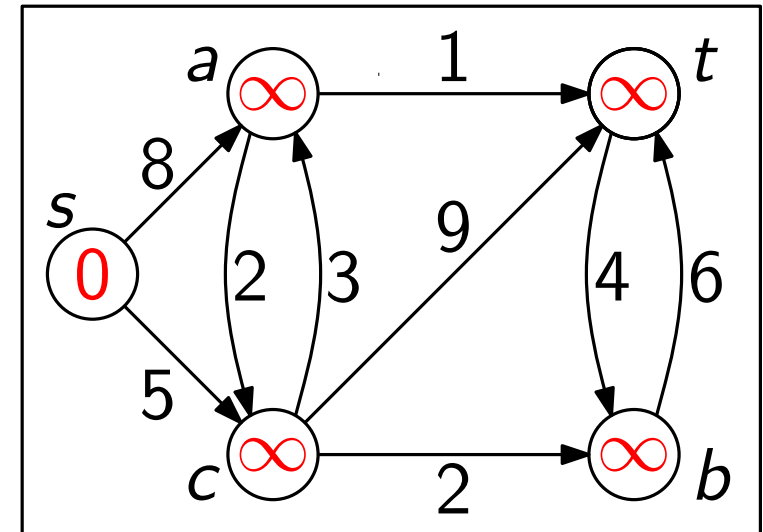
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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

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)

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

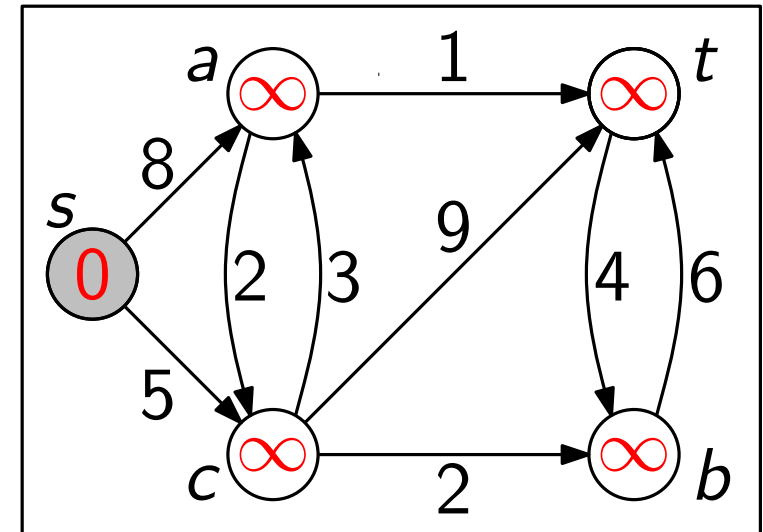
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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

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)

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

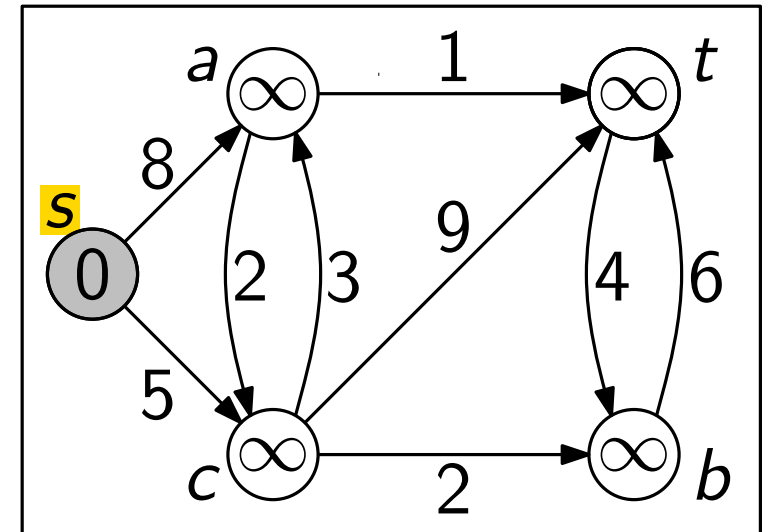
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

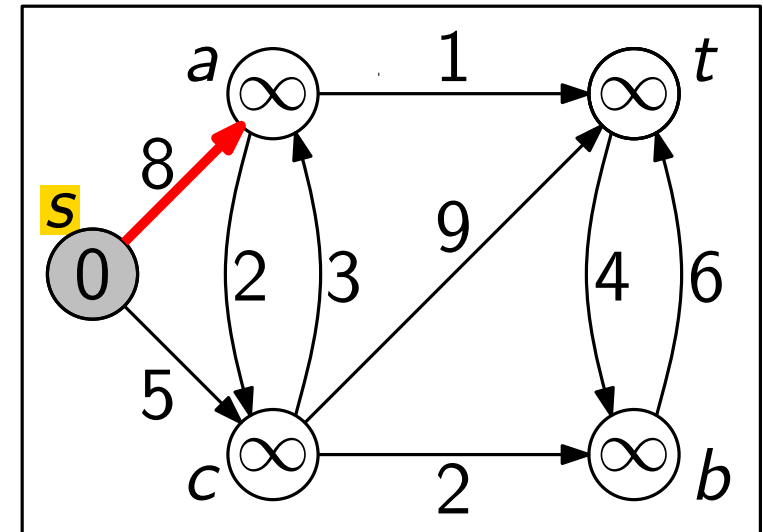
while not $Q.\text{Empty}()$ **do**

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

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)

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

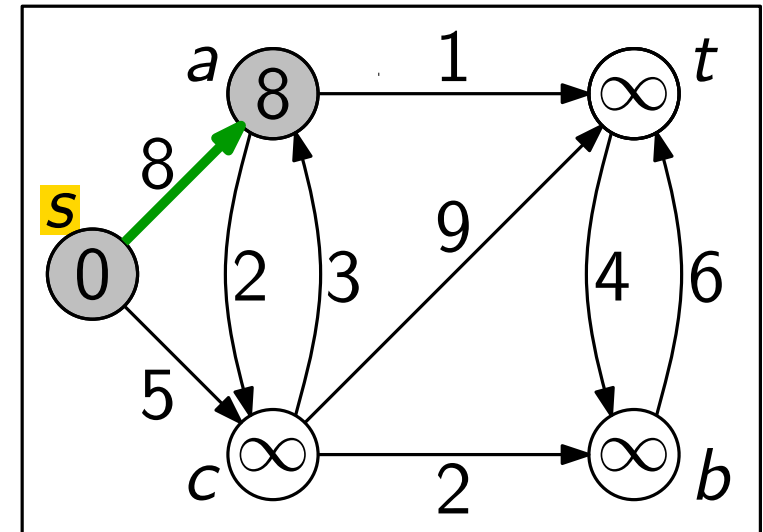
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

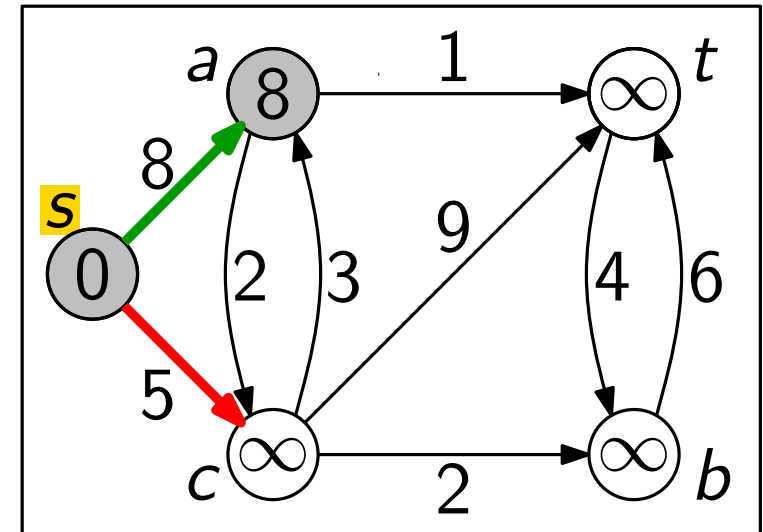
while not $Q.\text{Empty}()$ **do**

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

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)

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

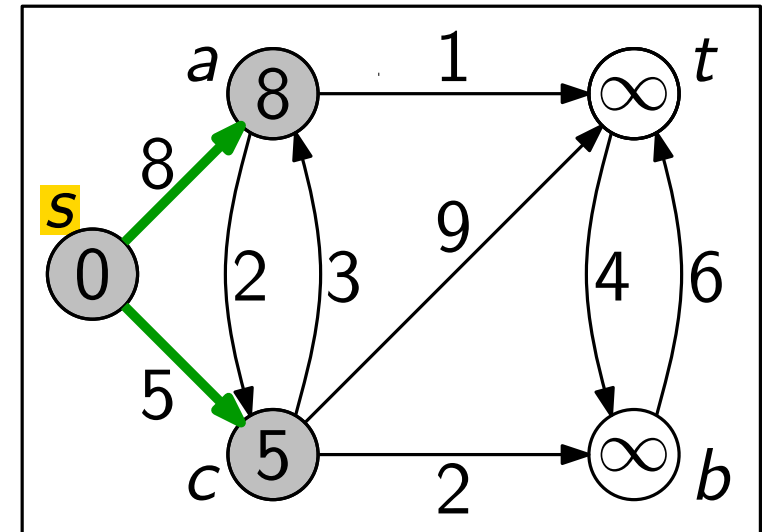
while not $Q.\text{Empty}()$ **do**

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

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)

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

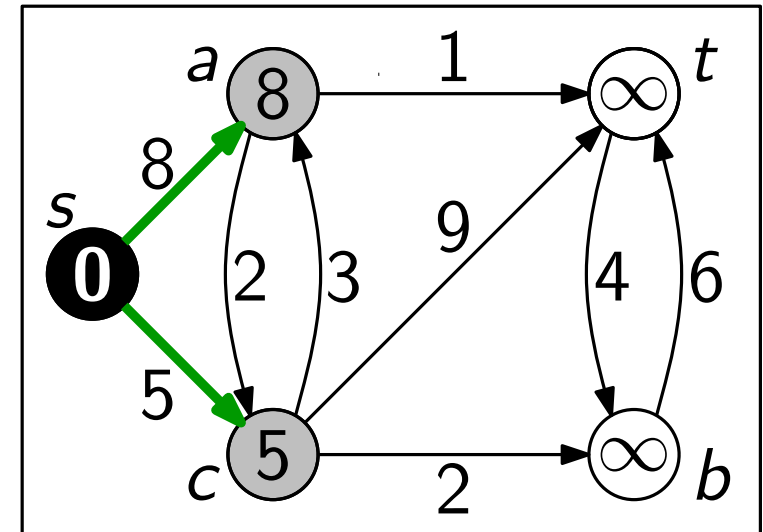
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

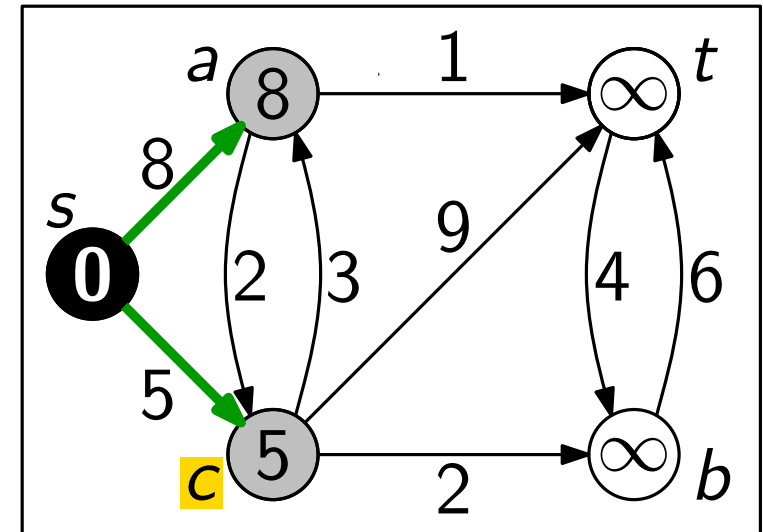
while not $Q.\text{Empty}()$ **do**

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

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)

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

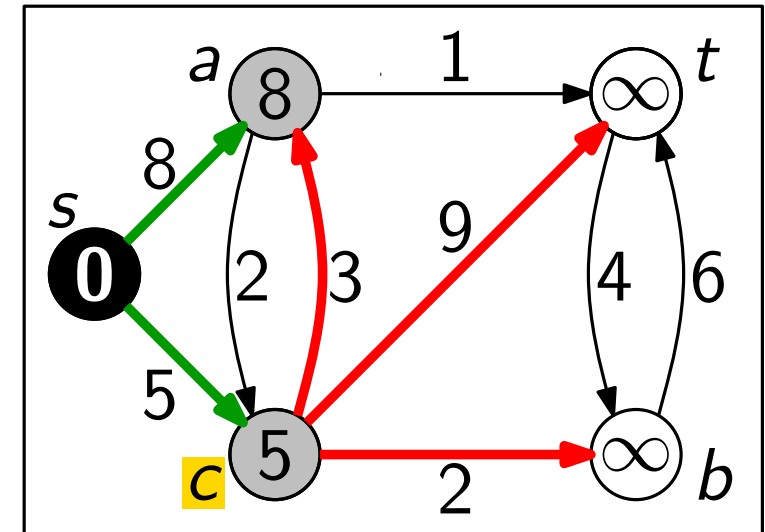
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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

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)

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

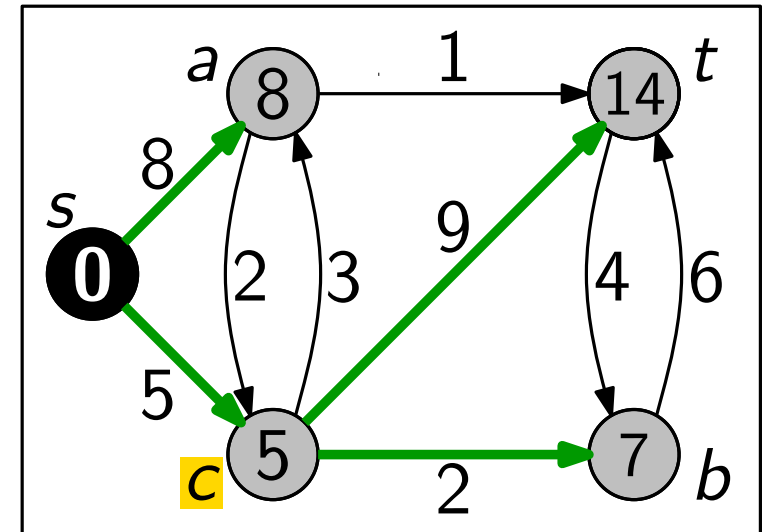
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

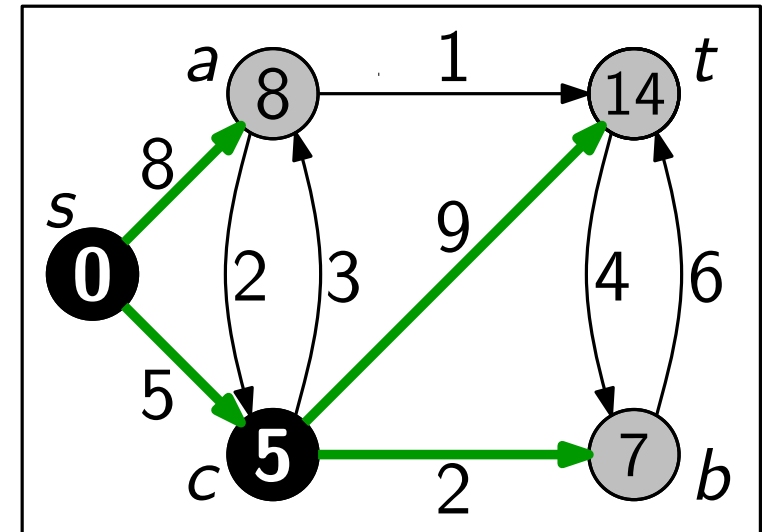
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

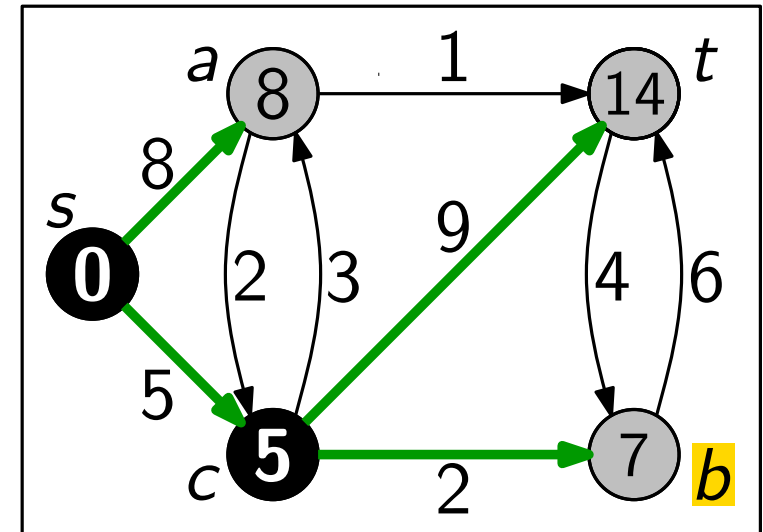
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

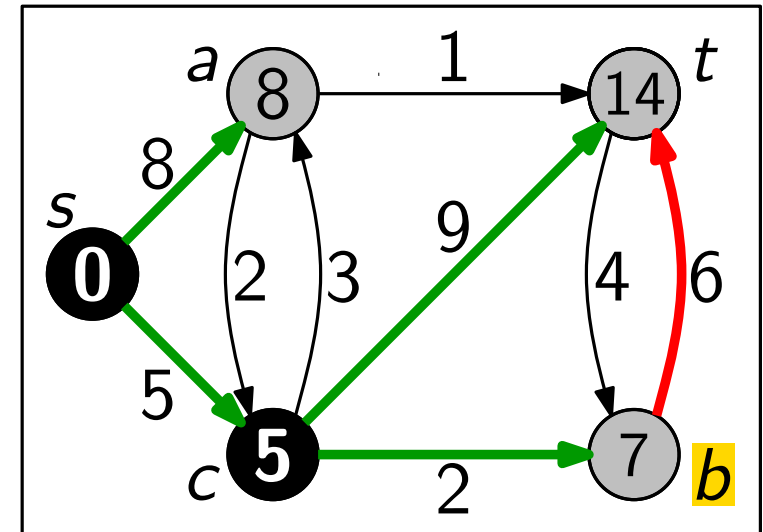
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

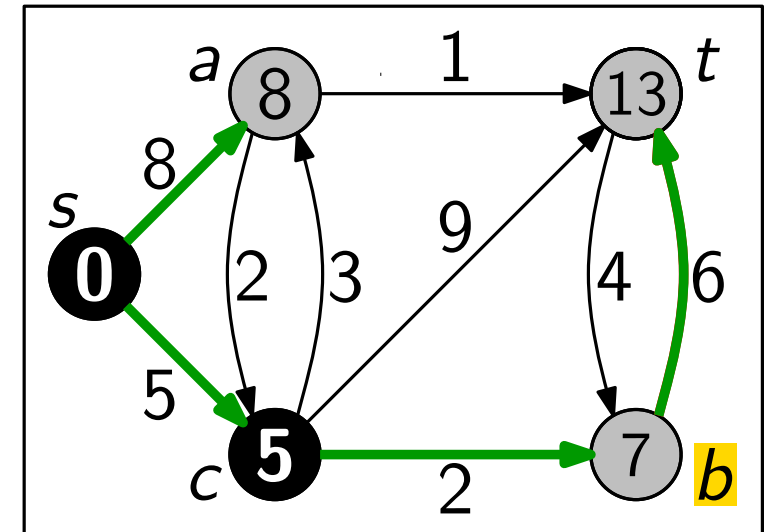
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

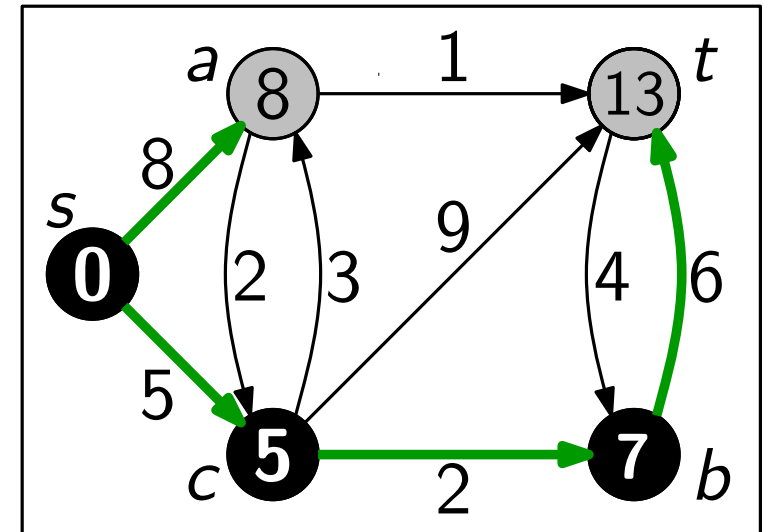
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

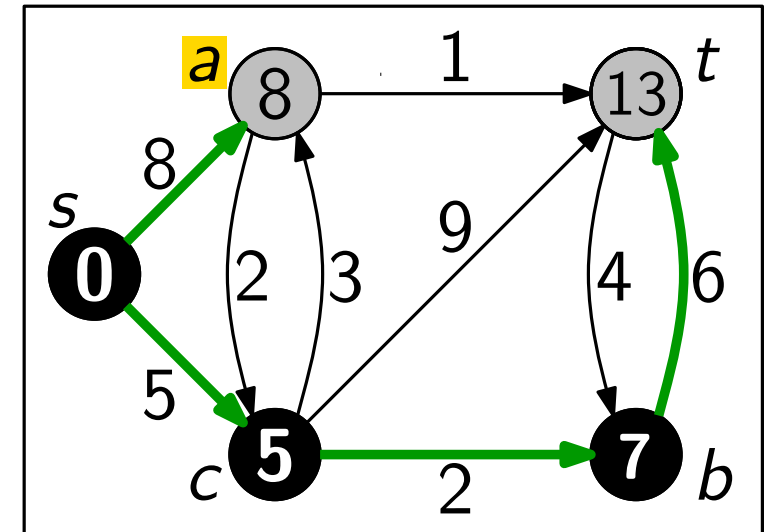
while not $Q.\text{Empty}()$ **do**

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

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)

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

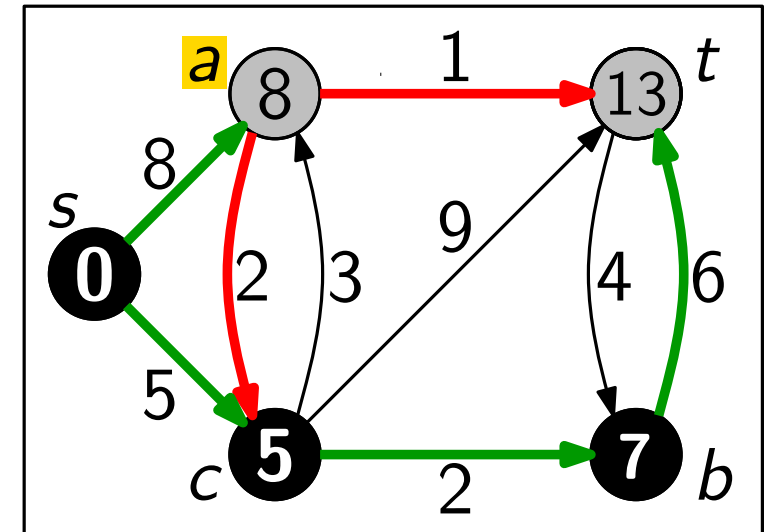
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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

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)

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

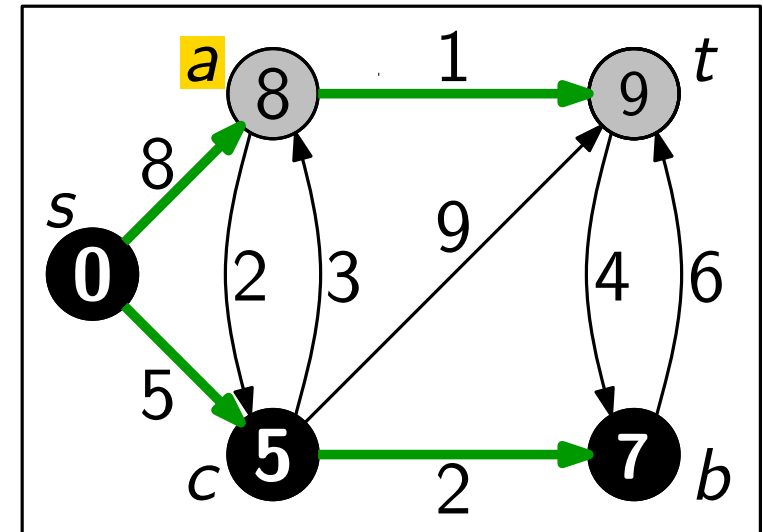
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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

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)

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

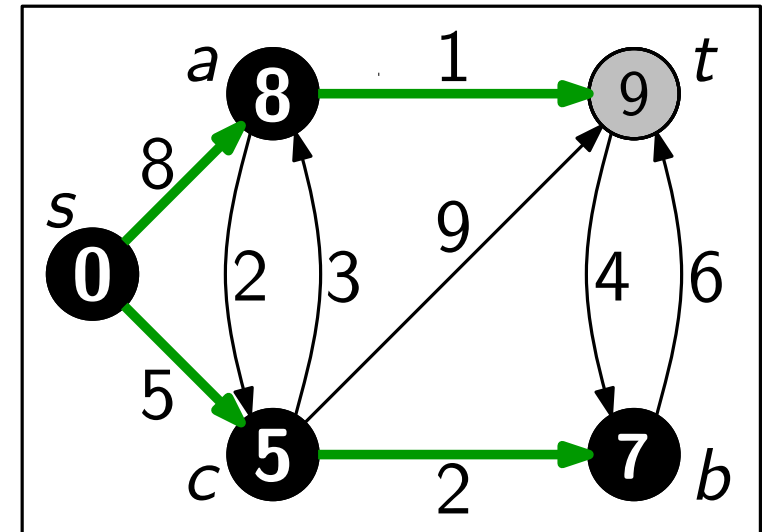
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

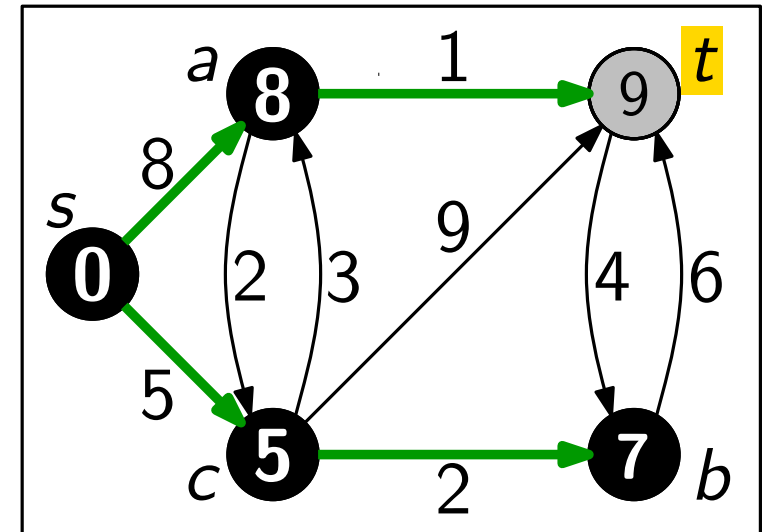
while not $Q.\text{Empty}()$ **do**

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

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)

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

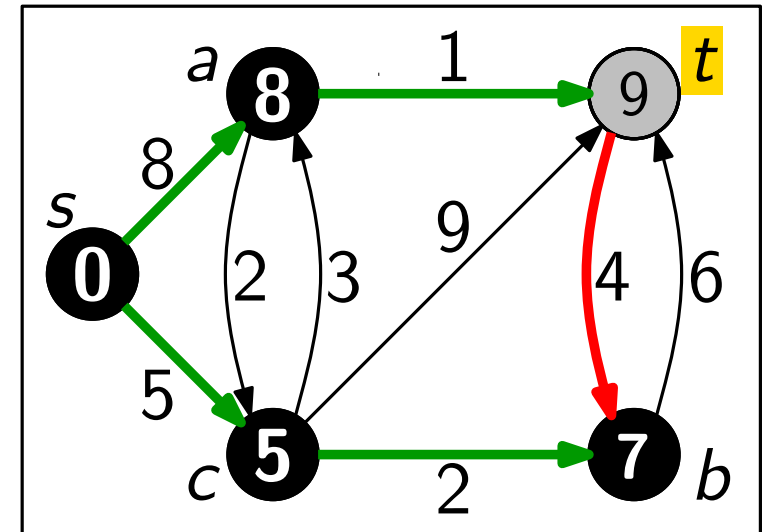
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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

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)

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

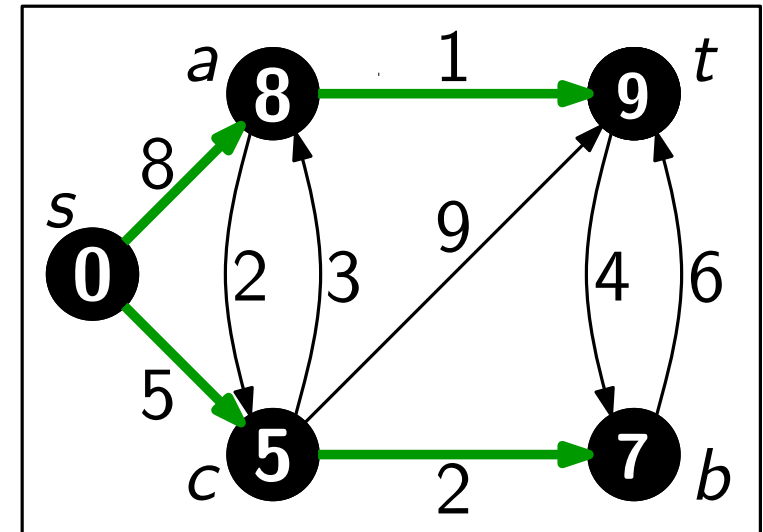
while not $Q.\text{Empty}()$ **do**

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

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

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

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



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)

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

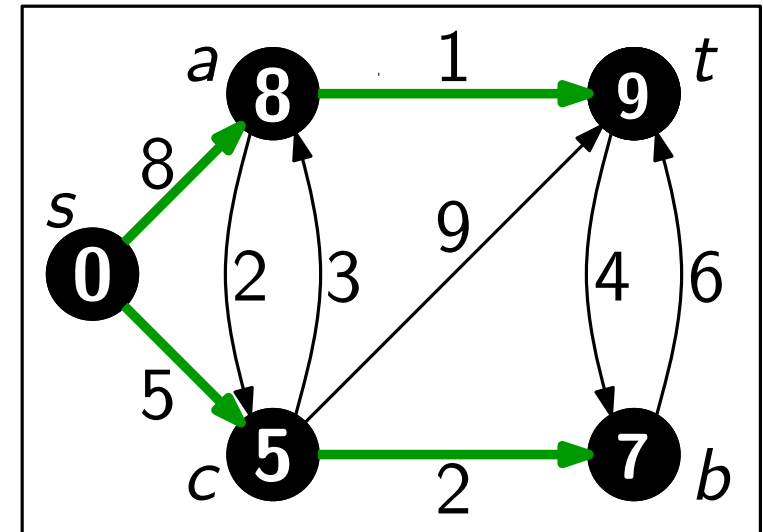
while not $Q.\text{Empty}()$ **do**

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

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**

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

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

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

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**

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

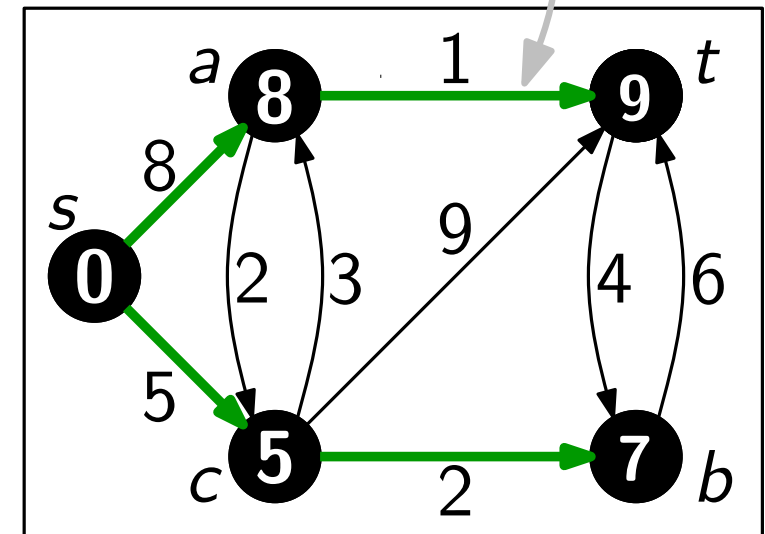
$u.d = \infty$

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

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

$s.d = 0$

Kürzester-Wege-Baum
mit Wurzel s



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**

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

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

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($u, v; w$)

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

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**

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

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

$O(\quad)$ 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)

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

$O(V)$ Zeit

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

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

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

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

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

Relax($u, v; w$)

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

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

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

$v.\pi = u$

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

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($u, v; w$)

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

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**

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

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

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**

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

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

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**

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

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

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**

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

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

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 $|\text{Adj}[u]|$ 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**

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

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

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 $|\text{Adj}[u]| = \text{deg}^{(\text{out-})} u$ 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**

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

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

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 $|\text{Adj}[u]| = \deg^{(\text{out-})} u$ mal,
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**

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

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

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

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

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

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

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

$v.\pi = u$

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

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

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**

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

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

Abk. für $O(|V|)$

$O(V)$ Zeit

genau $|V|$ mal

Wie oft wird Relax aufgerufen?

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

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

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

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

$v.\pi = u$

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

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

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

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

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

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

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 ($\rightarrow$ Direktzugriff)

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstras 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 ($\rightarrow$ Direktzugriff)

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstras 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 Dijkstras 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 ($\rightarrow$ Direktzugriff)

Dijkstra – die Laufzeit

Satz. Gegeben ein Graph $G = (V, E)$, läuft Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 Dijkstras 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 ein Heap nicht suchbar ist (!). Wir suchen 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 Dijkstras 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 Dijkstras 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 Dijkstras 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://videolectures.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

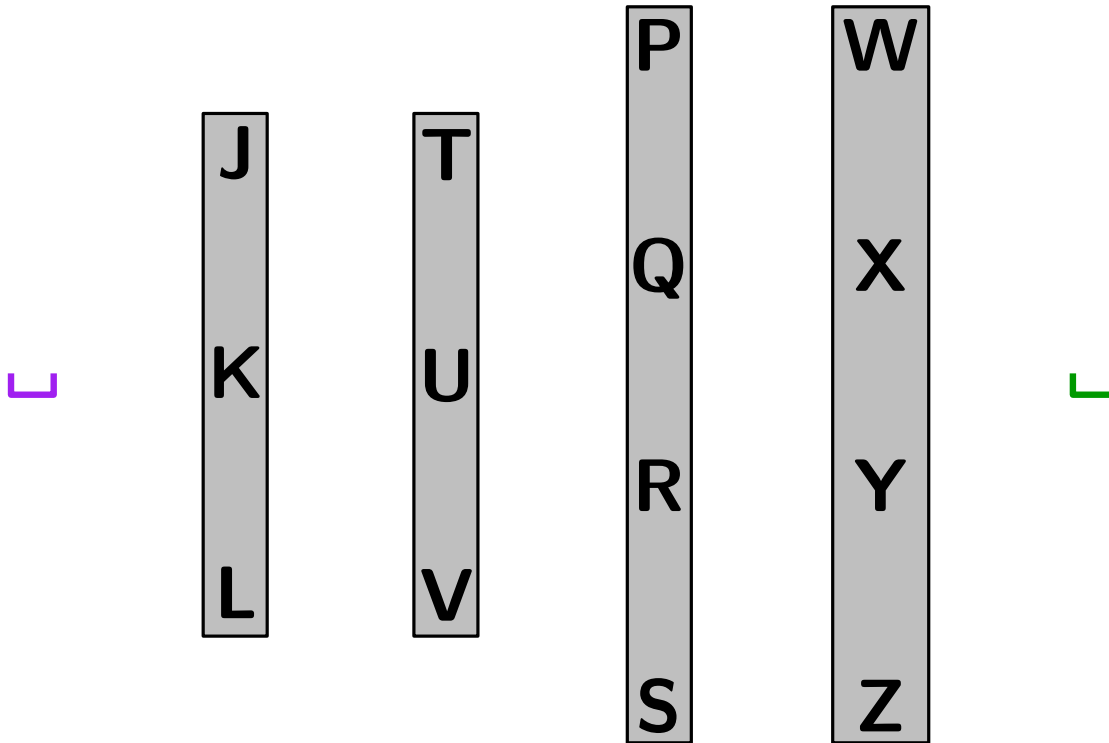
P
Q
R
S

W
X
Y
Z

└



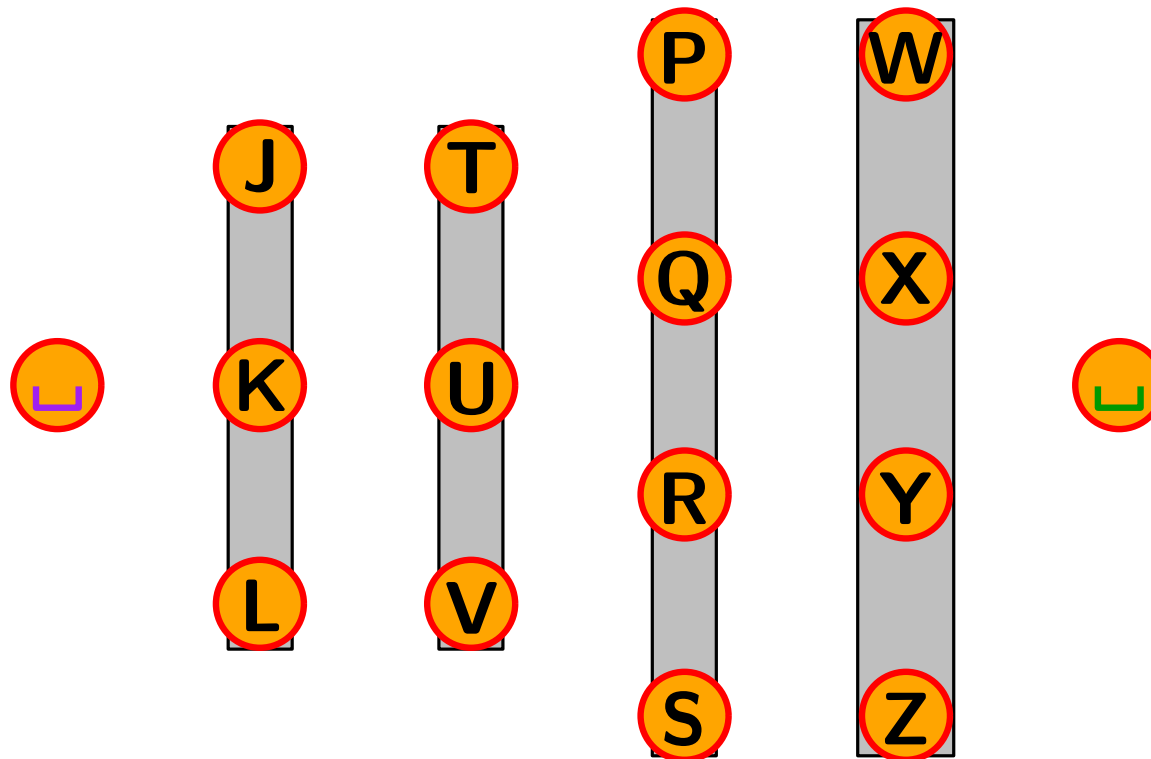
Modellierung – SMSen



Graph:

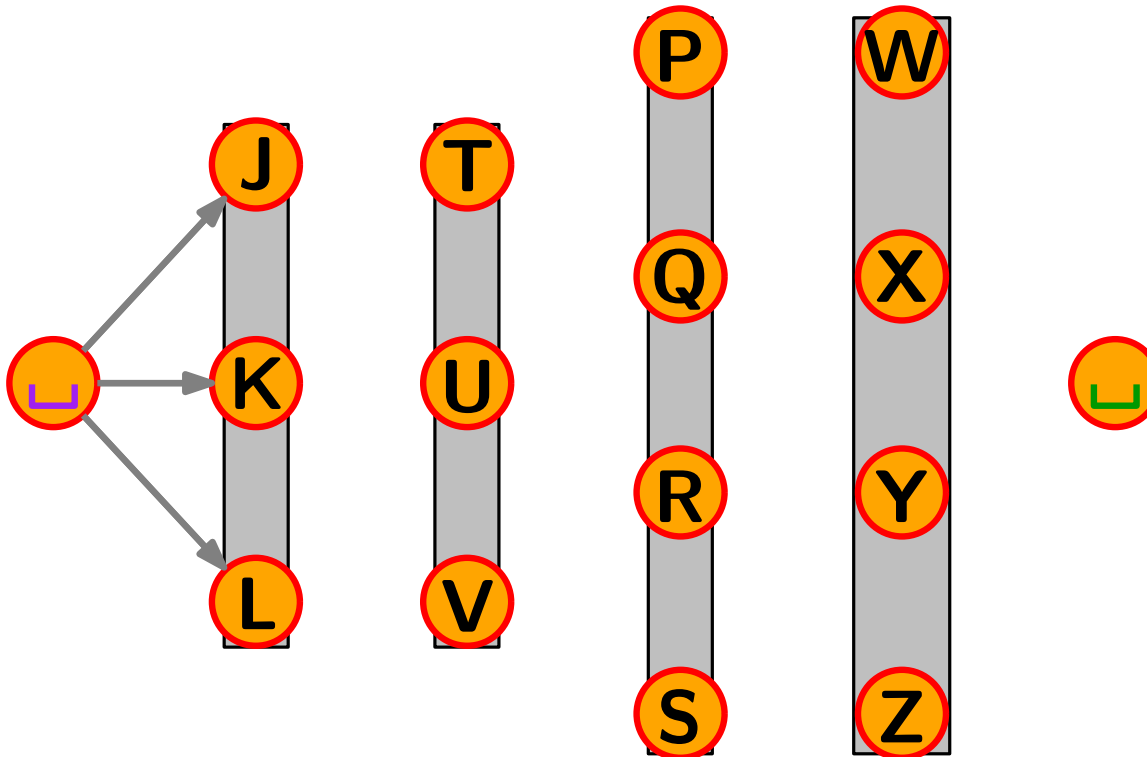


Modellierung – SMSen



Graph: Knoten $\hat{=}$ Buchstaben

Modellierung – SMSen

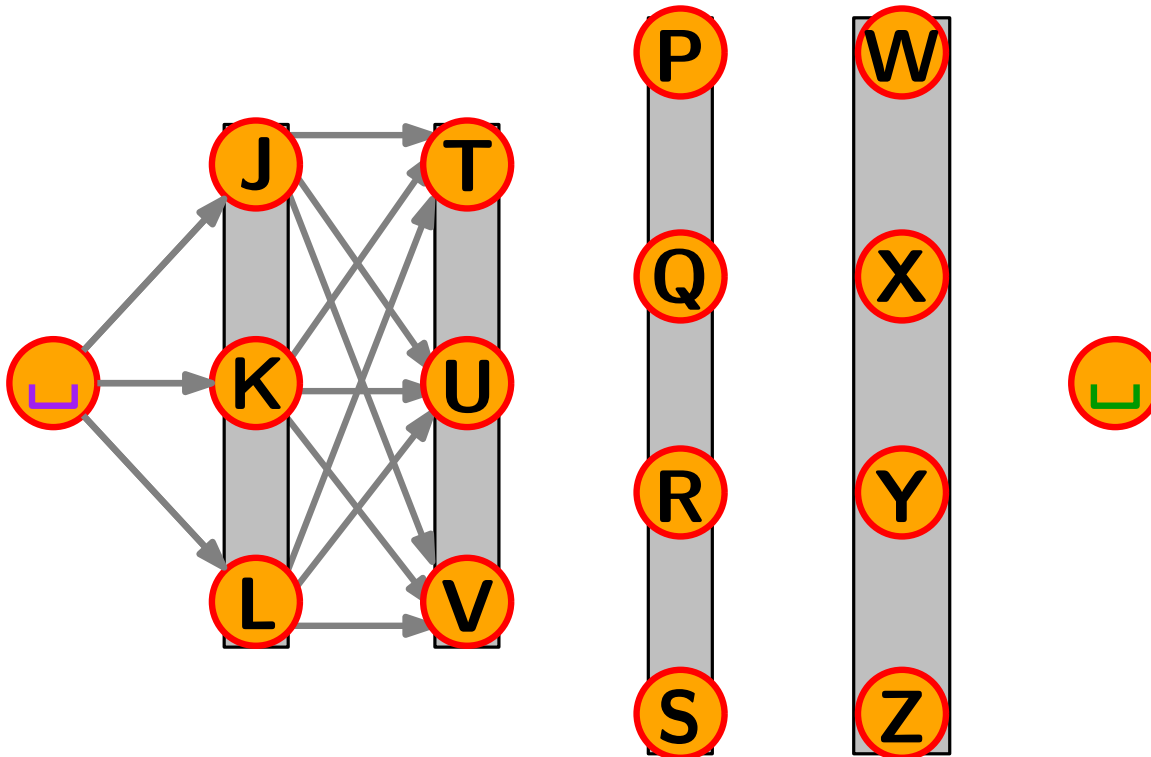


Graph:

Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Modellierung – SMSen

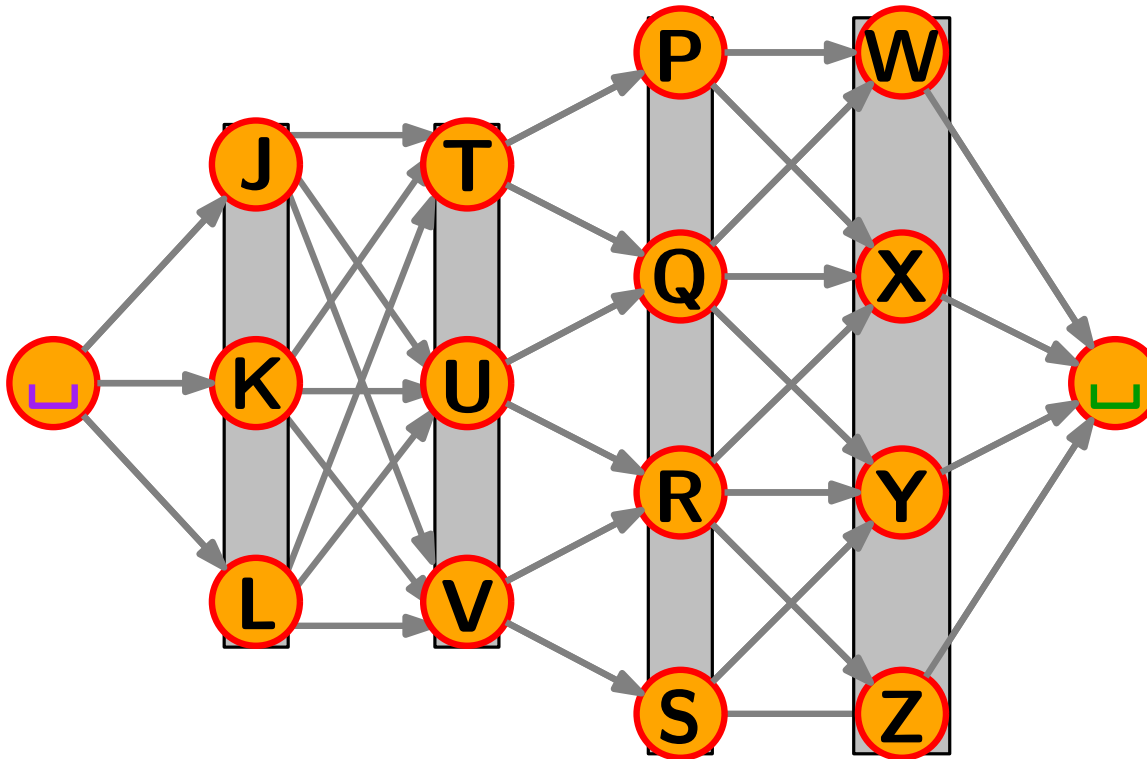


Graph:

Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Modellierung – SMSen

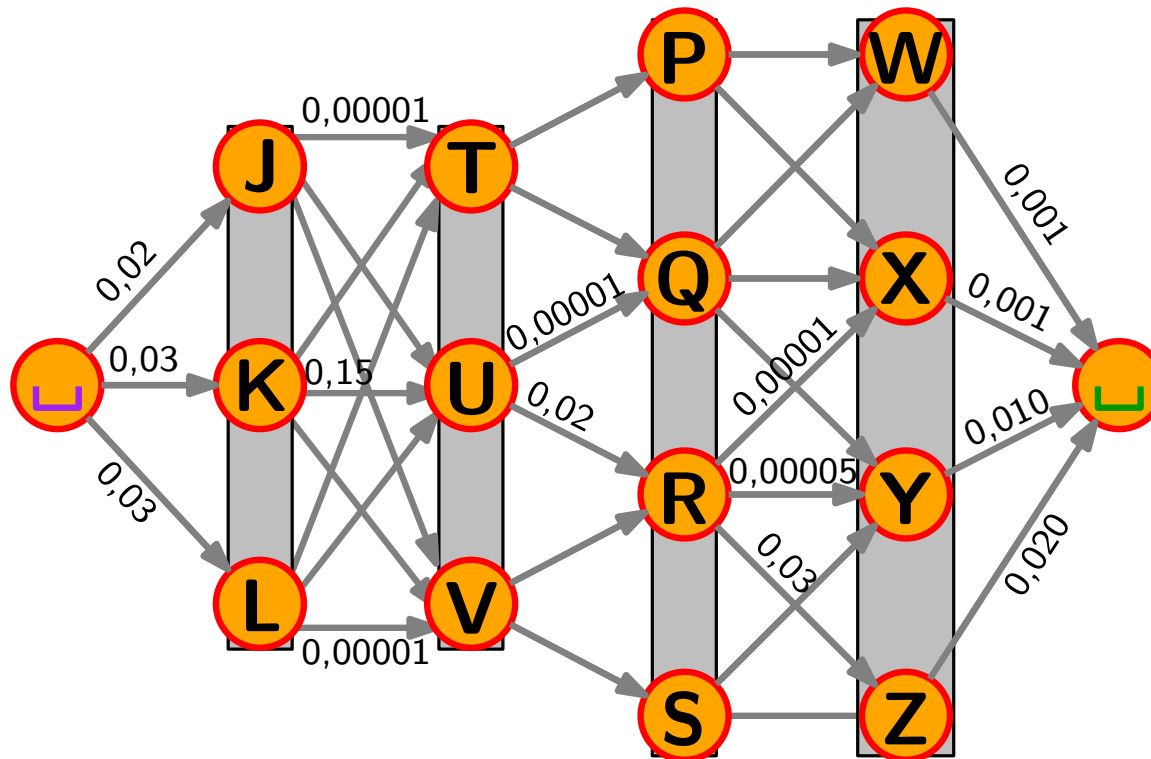


Graph:

Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Modellierung – SMSen



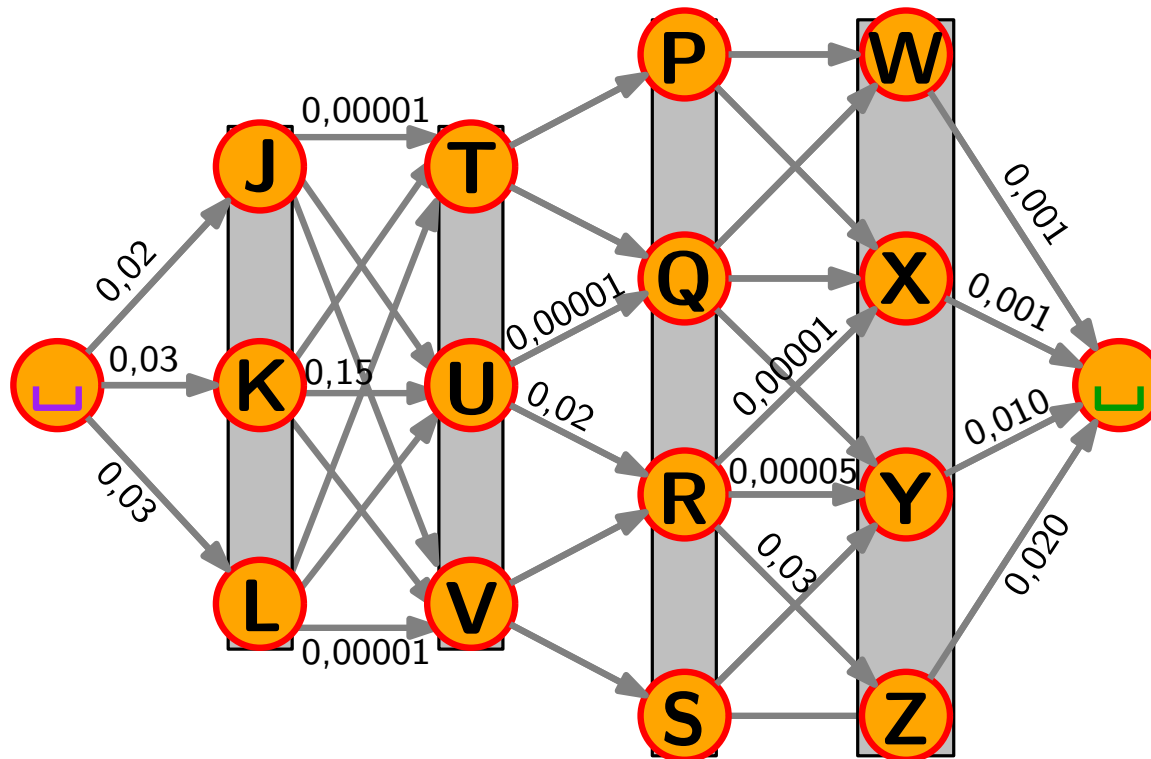
Graph:

Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

Gewichte $\hat{=}$ Wahrscheinlichkeiten w

Modellierung – SMSen



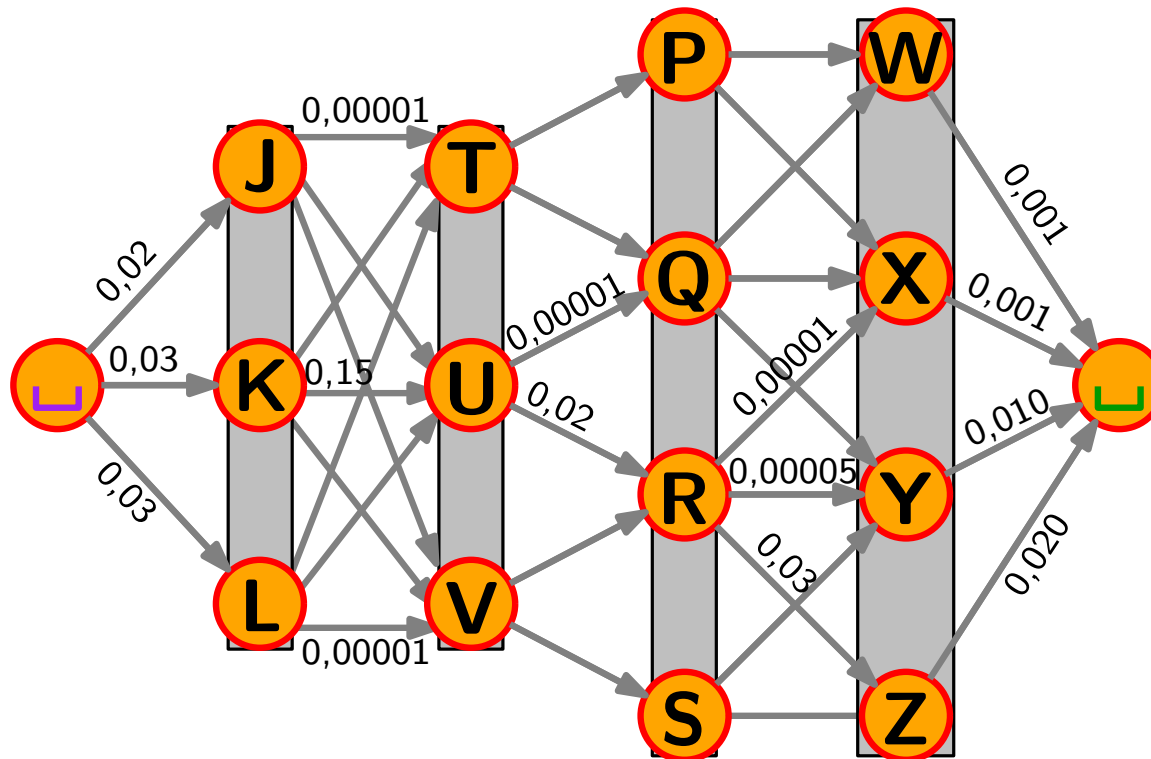
Graph:

Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

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

Modellierung – SMSen



Graph:

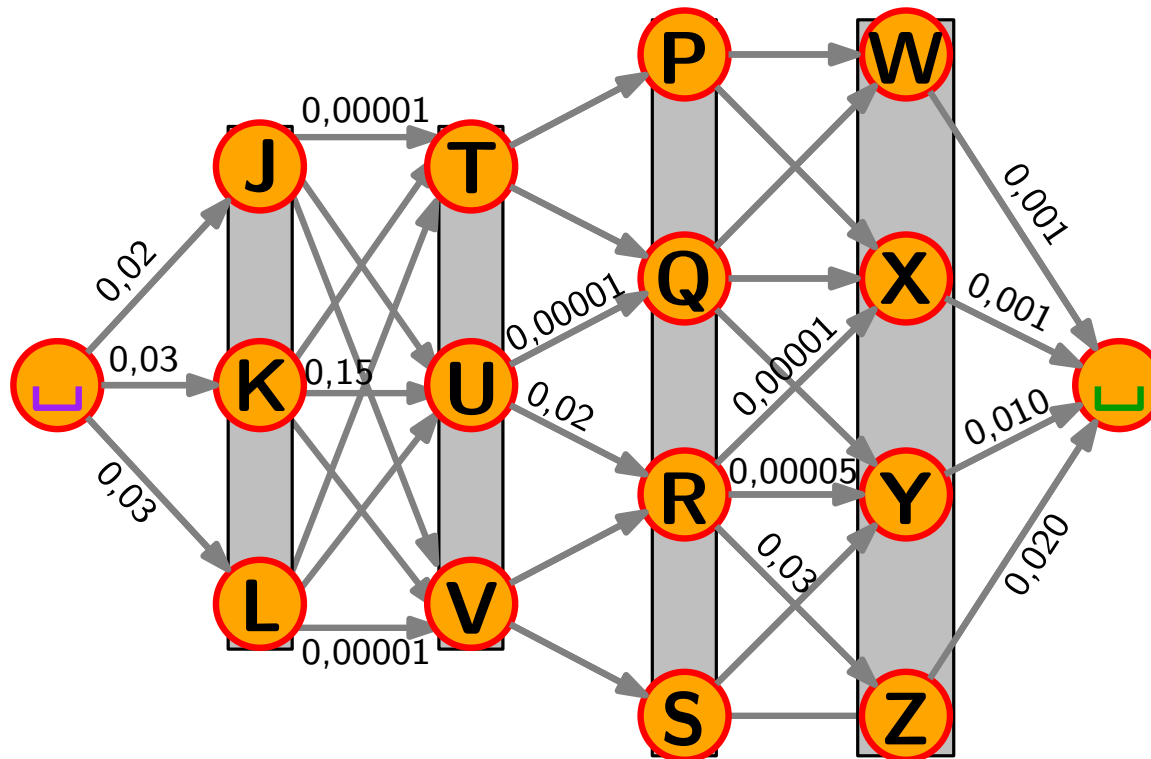
Knoten $\hat{=}$ Buchstaben

Kanten $\hat{=}$ aufeinanderfolgende Buchst.

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

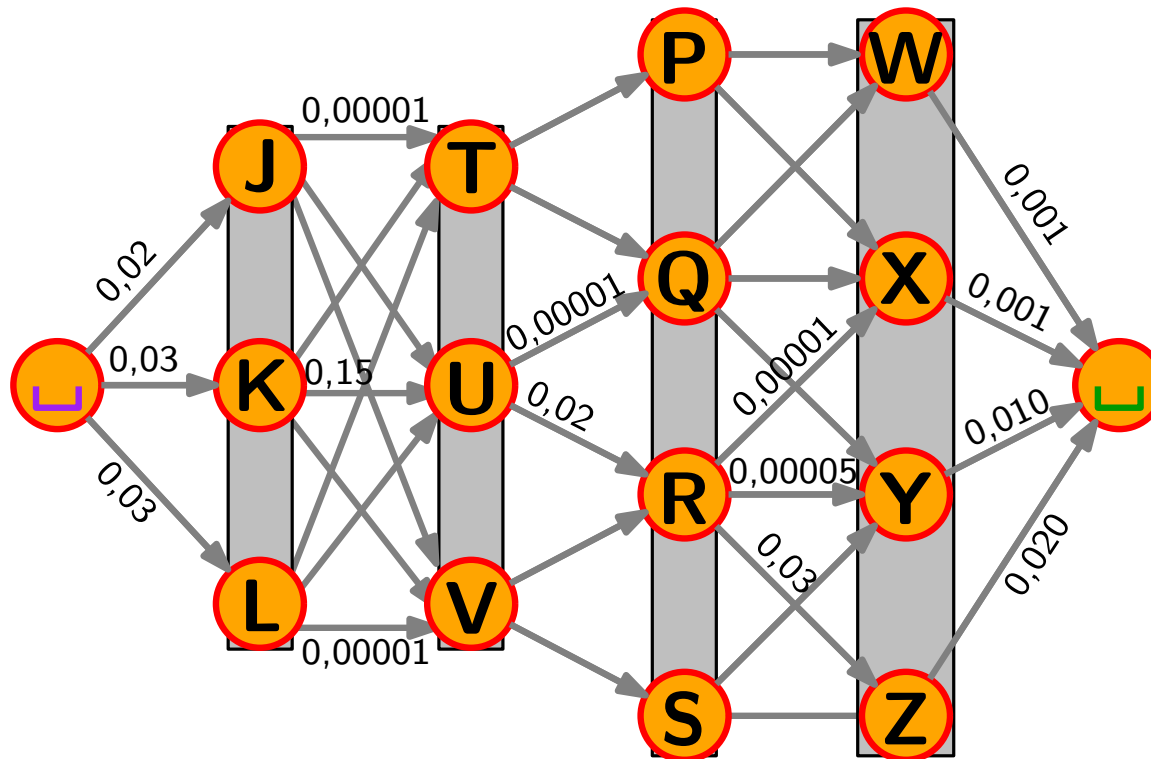
Gesucht:

Modellierung – SMSen



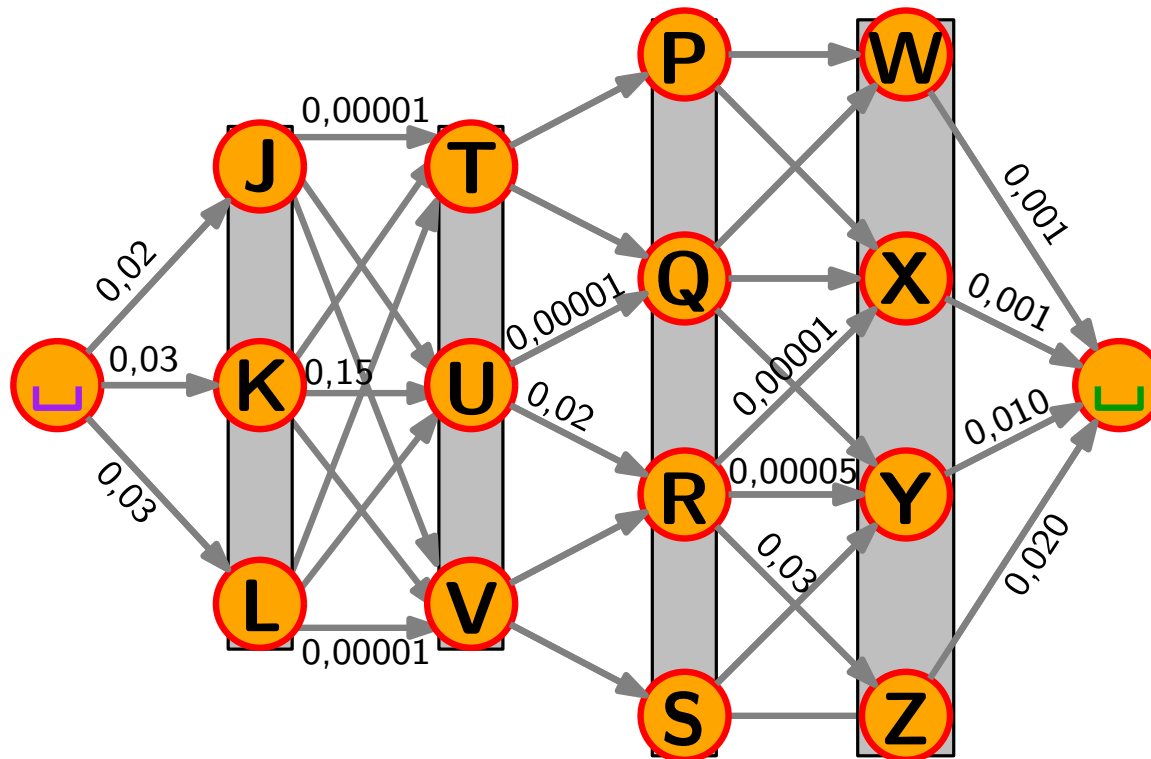
- Graph:** Knoten $\hat{=}$ Buchstaben
 Kanten $\hat{=}$ aufeinanderfolgende Buchst.
 Gewichte $\hat{=}$ Wahrscheinlichkeiten w / Häufigkeiten
- Gesucht:** Weg P von L nach L mit *größter* WK

Modellierung – SMSen



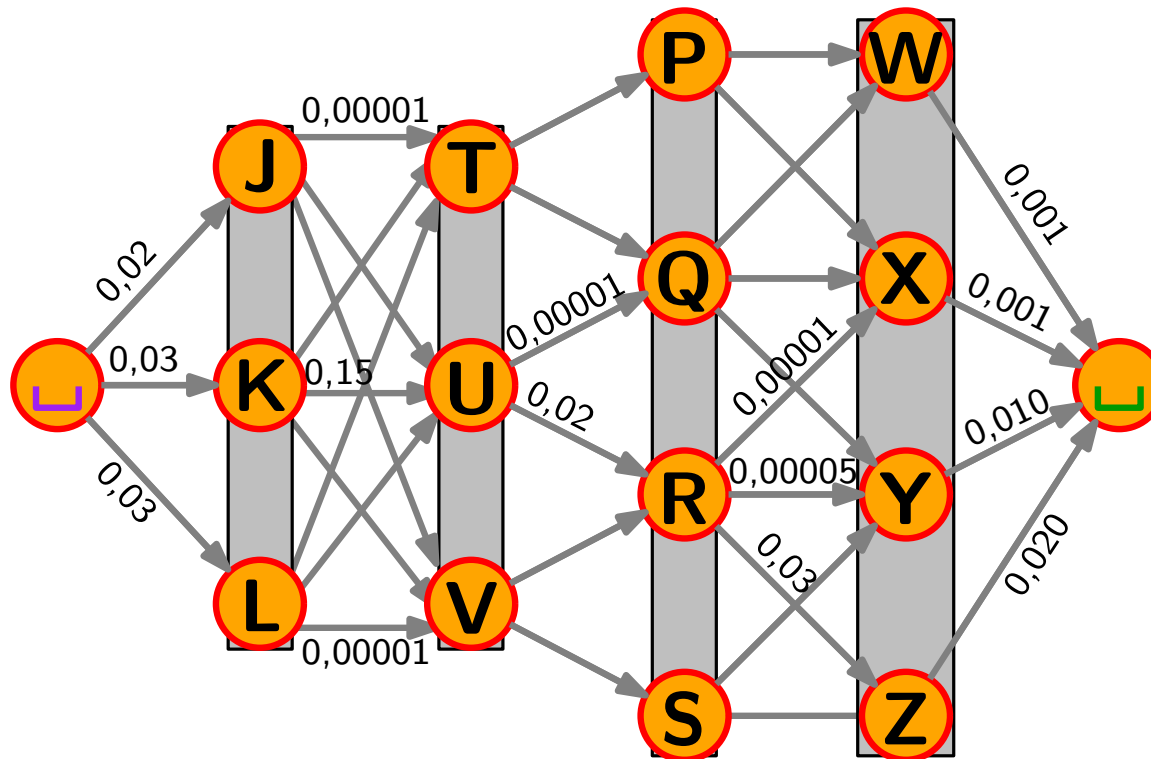
- Graph:** Knoten $\hat{=}$ Buchstaben
 Kanten $\hat{=}$ aufeinanderfolgende Buchst.
 Gewichte $\hat{=}$ Wahrscheinlichkeiten w / Häufigkeiten
- Gesucht:** Weg P von L nach L mit *größter* WK ($= \prod_{e \in P} w(e)$)

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,

Band 1, S. 269–271, 1959.



The solution given above is to be preferred to the solution by L. R. FORD [3] as described by C. BERGE [4], for, irrespective of the number of branches, we need not store the data for all branches simultaneously but only those for the branches in sets I and II, and this number is always less than n . Furthermore, the amount of work to be done seems to be considerably less.

Literatur

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

Edsger Wybe Dijkstra:

Numerische Mathematik,

Band 1, S. 269–271, 1959.



The solution given above is to be preferred to the solution by L. R. FORD [3] as described by C. BERGE [4], for, irrespective of the number of branches, we need not store the data for all branches simultaneously but only those for the branches in sets I and II, and this number is always less than n . Furthermore, the amount of work to be done seems to be considerably less.

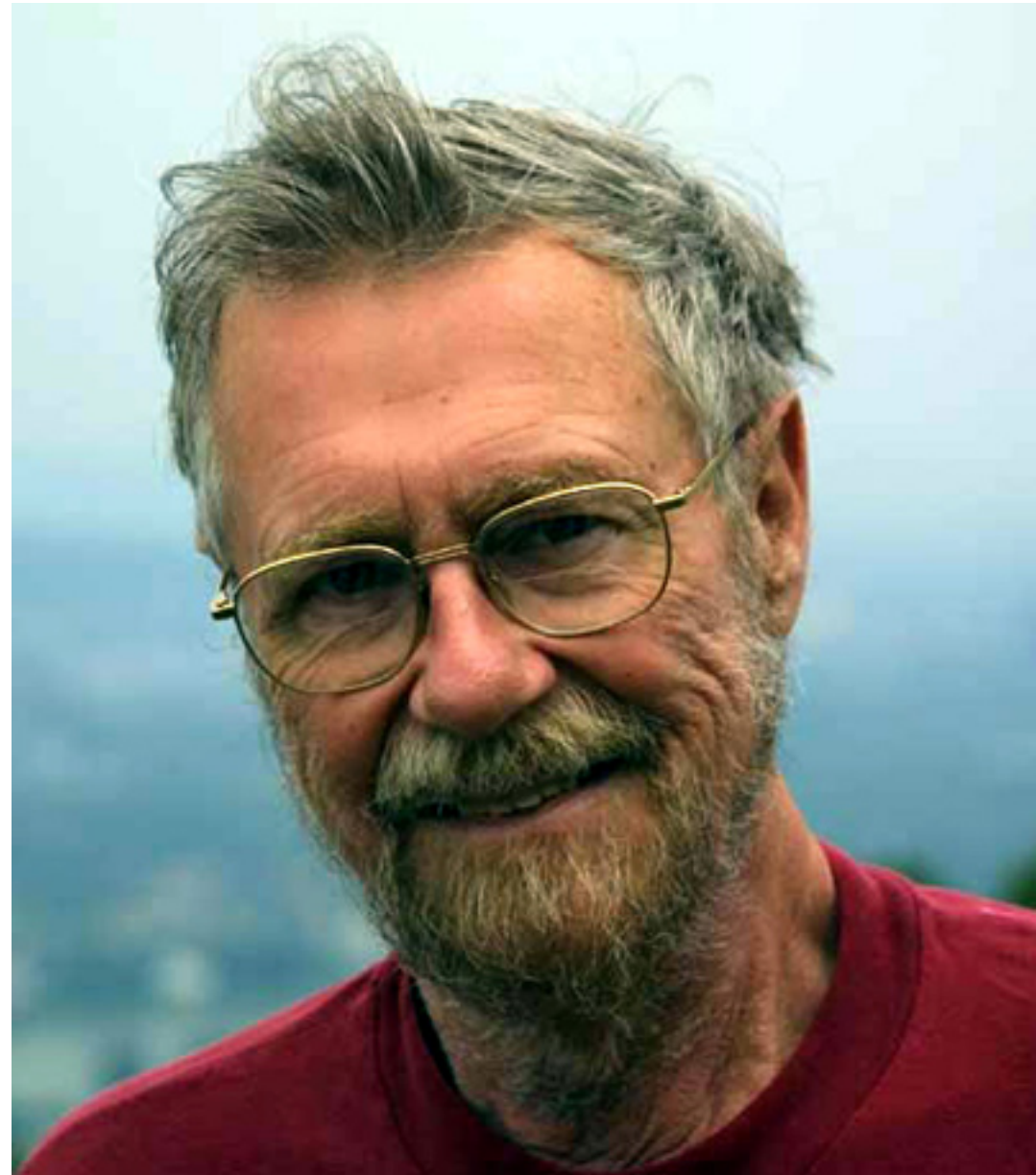
Literatur

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

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

- **Das Geheimnis des kürzesten Weges.**

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



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

Literatur

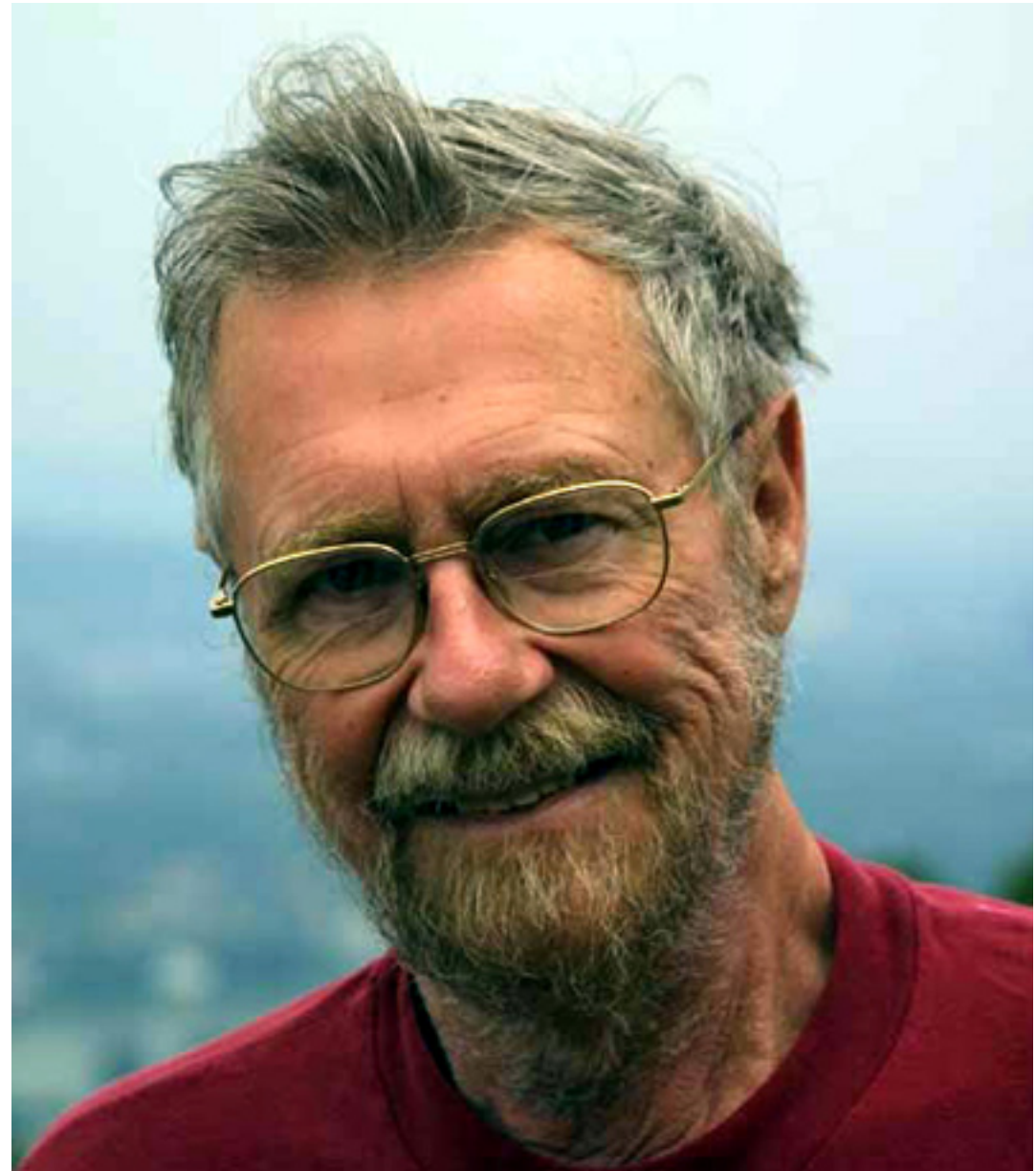
- **A note on two problems in connexion with graphs.**

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

- **Das Geheimnis des kürzesten Weges.**

Ein mathematisches Abenteuer.
Peter Gritzmann und
René Brandenburg:
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,
Band 1, S. 269–271, 1959.

- **Das Geheimnis des kürzesten Weges.**

Ein mathematisches Abenteuer.

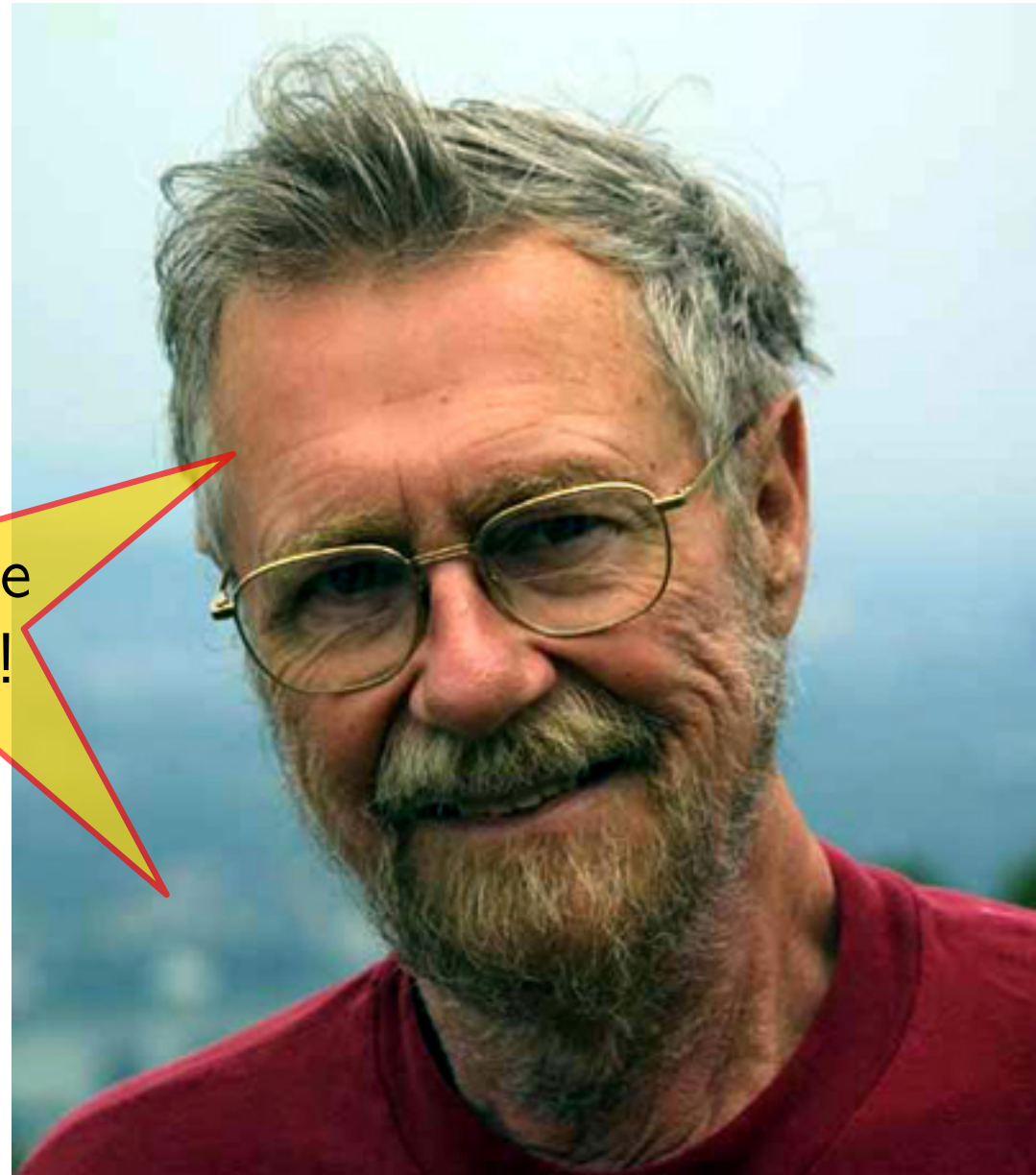
Peter Gritzmann und

René Brandenburg:

Springer-Verlag, 3. Aufl., 2005.

Lesen Sie
mal rein!

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

<i>Eingabe</i>	<i>Algorithmus</i>	<i>Laufzeit</i>
ungewichteter Graph	Breitensuche	$O(E + V)$ <i>letztes Mal</i> ✓
nicht-neg. Kantengew.	Dijkstra	$O(E + V \log V)$
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

<i>Eingabe</i>	<i>Algorithmus</i>	<i>Laufzeit</i>	
ungewichteter Graph	Breitensuche	$O(E + V)$	letztes Mal ✓
nicht-neg. Kantengew.	Dijkstra	$O(E + V \log V)$	✓
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

<i>Eingabe</i>	<i>Algorithmus</i>	<i>Laufzeit</i>	
ungewichteter Graph	Breitensuche	$O(E + V)$	letztes Mal ✓
nicht-neg. Kantengew.	Dijkstra	$O(E + V \log V)$	✓
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

<i>Eingabe</i>	<i>Algorithmus</i>	<i>Laufzeit</i>	
ungewichteter Graph	Breitensuche	$O(E + V)$	letztes Mal ✓
nicht-neg. Kantengew.	Dijkstra	$O(E + V \log V)$	✓
azyklischer Graph	topol. Sortieren	$O(E + V)$	nächstes Mal! ✓
negative Kantengew.	Bellman-Ford	$O(EV)$	Vorlesung Adv. Algorithms (M.Sc.)
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

<i>Eingabe</i>	<i>Algorithmus</i>	<i>Laufzeit</i>	
ungewichteter Graph	Breitensuche	$O(E + V)$	letztes Mal ✓
nicht-neg. Kantengew.	Dijkstra	$O(E + V \log V)$	✓
azyklischer Graph	topol. Sortieren	$O(E + V)$	nächstes Mal! ✓
negative Kantengew.	Bellman-Ford	$O(EV)$	Vorlesung Adv. Algorithms (M.Sc.)
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	$O(k + E + V \log V)$	