# Algo -Tutorium 6

26.11.2021

# Check-In

# Update & Feedback

- Maskenpflicht
- Ab nächste Woche 2G

- Code Abgaben: Nur wenn in der Aufgabe verlangt
- Notation vom Präsenzblatt verwenden
- Erklärungen gerne mit Grafik untermalen

# Klausur-relevanz

Von den bisherigen Themen war folgendes klausurrelevant:

- Rekursionsgleichungen (Kommen immer dran)
- Mastertheorem (Hin und wieder)
- Laufzeiten (Manchmal)
- Hashing (Manchmal)
- Baumalgorithmen (Manchmal)
- SSSP (Single-Source-Shortest Path Algorithmen) (Kommen oft dran in irgendeiner Form)

# Kürzeste Wege Algorithmen

**Single Source Shortest Path:**

- Dijkstra/ A*
- Bellman-Ford

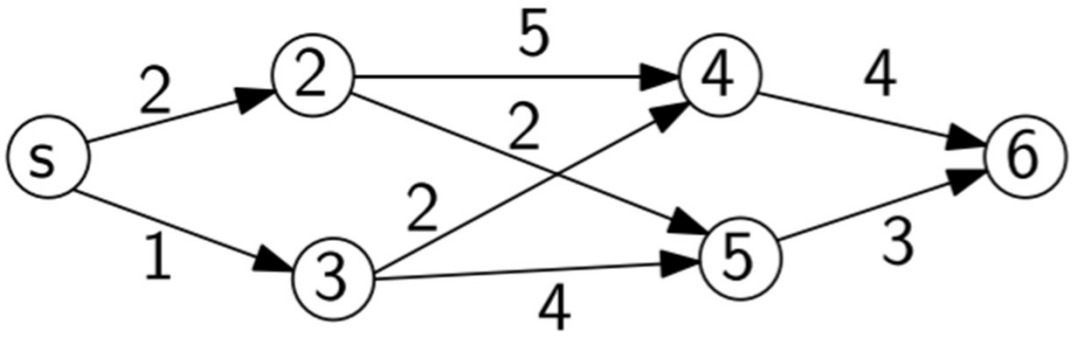
**All Pair Shortest Path:**

- Floyd-Warshall

# Wenn Graph DAG ist

- Simpler Algorithmus in O(n+m)

Distanzen $d(s) \leftarrow 0$; $\texttt{Pfad}(s) \leftarrow (s)$;
**for all** $(v \in V \backslash \{s\})$ $\{d(v) \leftarrow \infty;\}$
**for** $(v \leftarrow s+1$ to $n)$ {
$\quad d(v) \leftarrow \min_{u \geq s}\{d(u) + c(u,v) \mid (u,v) \in E\}$;
$\quad \texttt{Pfad}(v) \leftarrow \texttt{concat}(\texttt{Pfad}(u), v))$;
}



# Dijkstra – allgemein

- SSSP-Algorithmus, Greedy-Algorithmus
- Kantengewichte ≠ negativ
- Idee:
  - bereits bekannte kürzesteDistanzen in S
  - unbekannte in S‘

  **vergrößere S!**
- Abarbeitung durch Priority-Queue
- füllen dazu Tabelle mit $(a, k, b)$ d.h. nach $a$ kommt man mit Kosten/ Distanz $k$ und der Vorgänger ist $b$

! ist temporär zu lesen (kann sich noch ändern in S‘)

# Dijkstra Pseudocode

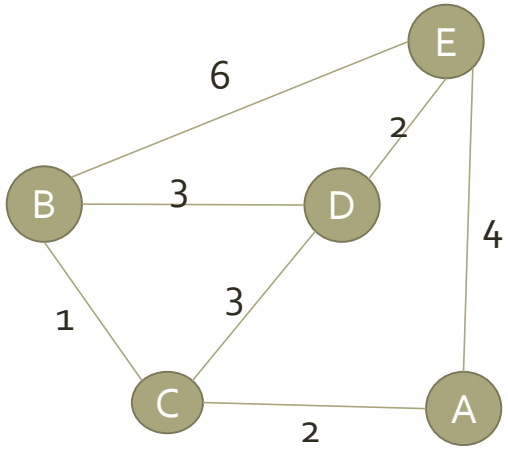
$S \leftarrow \{s\}; d(s), d'(s) \leftarrow 0;$
$S' \leftarrow \texttt{OutAdj}(s)$; for all $u \in S'$ $\{d'(u) = c(s,u);\}$
**for all** $(u \in V \backslash (S' \cup \{s\}))$ $\{d'(u) \leftarrow \infty;\}$

**while** $(S \neq V)$ {
    wähle $w \in S'$ geeignet  // ('findmin')
    $d(w) \leftarrow d'(w);$
    $S \leftarrow S \cup \{w\}; S' \leftarrow S' \backslash \{w\};$

    **for all** $(u \in \texttt{OutAdj}(w))$ {
        **if** $(u \notin (S \cup S'))$ $\{S' \leftarrow S' \cup \{u\};\}$
        $d'(u) \leftarrow min\{d'(u), d'(w) + c(w,u)\}$
    }
}

# Dijkstra – Beispiel 1

## Graph

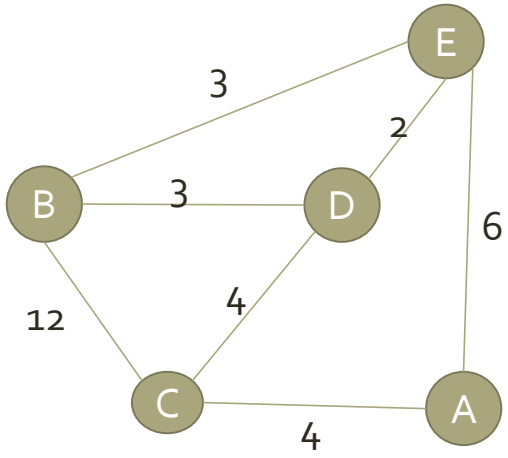



## Tabelle

| step (v) | S | S‘ |
|---|---|---|
| 0 (init) | | $(a, 0, -)$ |
| 1 (A) | $(a, 0, -)$ | $(c, 2, a)$, $(e, 4, a)$ |
| 2 (C) | $(a, 0, -)$, $(c, 2, a)$ | $(b, 3, c)$, $(d, 5, c)$, $(e, 4, a)$ |
| 3 (B) | $(a, 0, -)$, $(c, 2, a)$, $(b, 3, c)$ | $(d, 5, c)$, $(e, 4, a)$ |
| 4 (E) | $(a, 0, -)$, $(c, 2, a)$, $(b, 3, c)$, $(e, 4, a)$ | $(d, 5, c)$ |
| 5 (D) | $(a, 0, -)$, $(c, 2, a)$, $(b, 3, c)$, $(e, 4, a)$, $(d, 5, c)$ | |

# Dijkstra – Beispiel 2

## Graph




## Tabelle

| step (v) | S | S‘ |
|---|---|---|
| 0 (init) | | $(a, 0, -)$ |
| 1 (A) | $(a, 0, -)$ | $(c, 4, a)$, $(e, 6, a)$ |
| 2 (C) | $(a, 0, -)$, $(c, 4, a)$ | $(b, 16, c)$, $(d, 8, c)$, $(e, 6, a)$ |
| 3 (E) | $(a, 0, -)$, $(c, 4, a)$, $(e, 6, a)$ | $(b, 9, e)$, $(d, 8, c)$ |
| 4 (D) | $(a, 0, -)$, $(c, 4, a)$, $(e, 6, a)$, $(d, 8, c)$ | $(b, 9, e)$ |
| 5 (E) | $(a, 0, -)$, $(c, 4, a)$, $(e, 6, a)$, $(d, 8, c)$, $(b, 9, e)$ | |

# Bellman-Ford – allgemein

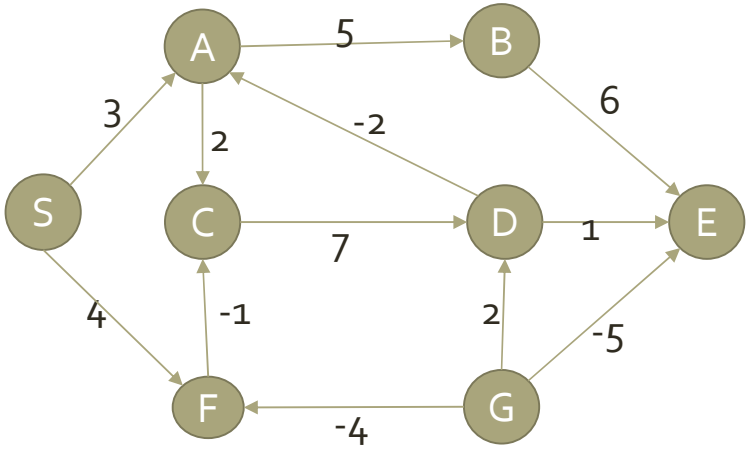
- SSSP-Algorithmus
- für alle gewichteten Graphen, auch negative Kantengewichte
- Idee:
  - betrachte nacheinander alle Kanten
  - Überprüfe für jeden Knoten $v$ für jede Kante $(u, v)$, ob $v$ von $u$ aus billiger erreicht werden kann als bisher bekannt

- füllen dazu wieder Tabelle $\mathrm{a}[k, b]$ d.h. nach $a$ kommt man mit Kosten/ Distanz $k$ und der Vorgänger ist $b$

! ist wieder temporär zu lesen (Änderungen möglich)

- $\leq n - 1$ Iterationen: Shortest Paths
- $n$ Iterationen: negative Zyklen können identifiziert werden

# Bellman-Ford – Beispiel 1



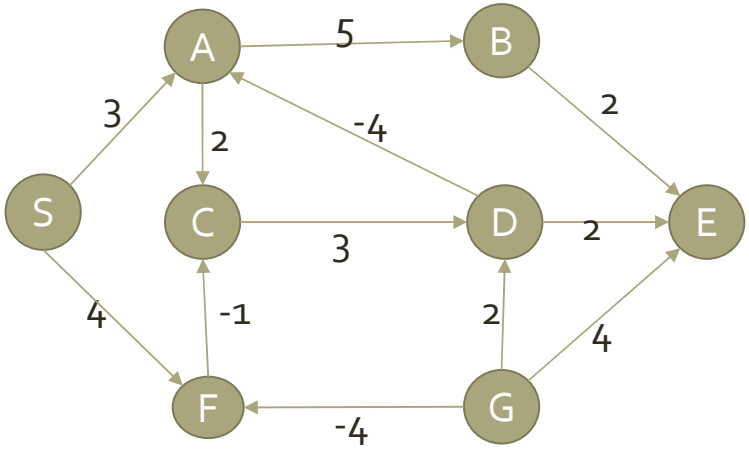

**Kantenreihenfolge:**

(S,A) – (S,F) – (A,B) – (A,C) – (C,D) – (D,A) – (B,E)– (D,E) – (F,C) – (G,D) – (G,E) – (G,F)

| step | S | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|---|
| (init) | $(0,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ |
| (1) | $(0,-)$ | $(3,S)$ | $(8,A)$ | $(5,A)$ $(3,F)$ | $(12,C)$ | $(14,B)$ $(13,D)$ | $(4,S)$ | $(\infty,-)$ |
| (2) | $(0,-)$ | $(3,S)$ | $(8,A)$ | $(3,F)$ | $(10,C)$ | $(11,D)$ | $(4,S)$ | $(\infty,-)$ |
| (3) | $(0,-)$ | $(3,S)$ | $(8,A)$ | $(3,F)$ | $(10,C)$ | $(11,D)$ | $(4,S)$ | $(\infty,-)$ |

STOP

# Bellman-Ford – Beispiel 2




**Kantenreihenfolge:**

(S,A) – (S,F) – (A,B) – (A,C) - (C,D) – (D,A) – (B,E) –(D,E) – (F,C) – (G,D) – (G,E) – (G,F)

| step | S | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|---|
| (init) | $(0,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ | $(\infty,-)$ |
| (1) | $(0,-)$ | $(3,S)$ | $(8,A)$ | $(3,F)$ | $(8,C)$ | $(8,B)$ | $(4,S)$ | $(\infty,-)$ |
| (2) | $(0,-)$ | $(3,S)$ | $(8,A)$ | $(3,F)$ | $(6,C)$ | $(11,D)$ | $(4,S)$ | $(\infty,-)$ |
| (3) | $(0,-)$ | $(2,D)$ | $(8,A)$ | $(3,F)$ | $(6,C)$ | $(8,D)$ | $(4,S)$ | $(\infty,-)$ |
| (4) | $(0,-)$ | $(2,D)$ | $(7,A)$ | $(3,F)$ | $(6,C)$ | $(8,D)$ | $(4,S)$ | $(\infty,-)$ |
| (5) | $(0,-)$ | $(2,D)$ | $(7,A)$ | $(3,F)$ | $(6,C)$ | $(8,D)$ | $(4,S)$ | $(\infty,-)$ |

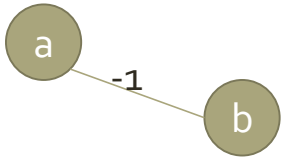
**STOP**

# Gerichtet/ ungerichtet?

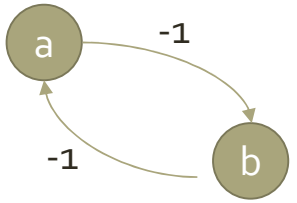
- beide Algorithmen funktionieren sowohl mit ungerichteten als auch gerichteten Kanten (bei letzterem sind ggfs. nicht alle Knoten von der Source erreichbar, kann man durch DFS/BFS prüfen)

- Will man die Kanten richten (ungerichtet → gerichtet), so gilt für:
  - Dijkstra: kein Problem
  - Bellman/Ford: nur, wenn Kantengewichte positiv sind (sonst negativer Zykel!)

denn aus



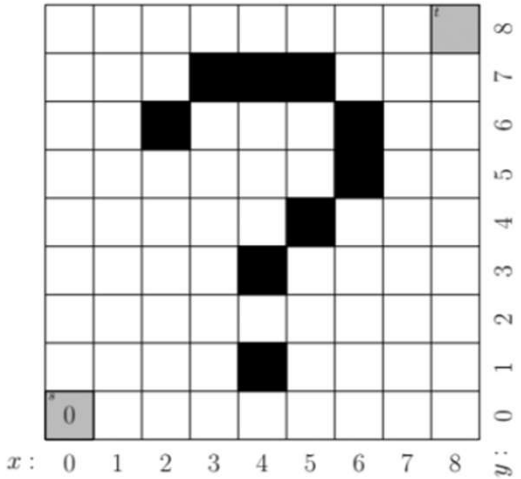

wird sonst



PÜ

# A1

In dieser Aufgabe betrachten wir die Anwendung von Dijsktra auf einen Gittergraphen:




Dabei soll jedes Gitterfeld $(x, y)$ dem Knoten eines Graphen entsprechen. Die schwarz hinterlegten Felder sind dabei nicht Teil des Graphen. Jedes Feld ist dabei mit seinen *horizontalen* und *vertikalen* Nachbarn verbunden. Zudem sind zwei diagonal benachbarte Felder $(x, y)$ und $(x', y')$ miteinander verbunden falls mindestens eines der Felder $(x, y')$ und $(x', y)$ Teil des Graphen ist. So sind beispielsweise die Felder $(2, 7)$ und $(3, 6)$ nicht verbunden. Die Kosten aller Kanten sind 1.

Führen Sie den Dijkstra-Algorithmus aus, um den kürzesten Weg von $s = (0, 0)$ zu $t = (8, 8)$ zu finden. Schreiben Sie in jedes Feld $(x, y)$ den Wert von $d((x, y))$ sobald $d((x, y)) \neq \infty$ wie für Feld $s$ in der Abbildung gezeigt. Es genügt alle Schritte in einer Zeichnung zu visualisieren, sie können die gegebene Abbildung mit der Software ipe (siehe auch `http://ipe.otfried.org/`) editieren.

# A2

In der Vorlesung haben Sie mehrere Algorithmen kennengelernt, die die Länge eines kürzesten Wegs bestimmen. In vielen Anwendungen, wie z.B. bei Navigationssystemen oder dem Routing von Paketen, ist aber auch der tatsächliche kürzeste Weg von Bedeutung.

a) Geben Sie einen Algorithmus in Pseudocode an, der für einen gegebenen zusammenhängenden Graphen $G = (V, E)$ mit positiven Kantengewichten $c : E \to \mathbb{N}$ den kürzesten Weg von Startknoten $s$ zum Zielknoten $t$ ausgibt. Wandeln Sie dafür Dijkstras Algorithmus so um dass ein Pfad zurückgegeben wird, statt einer Weglänge.

b) Bestimmen Sie die Laufzeit Ihres Algorithmus aus a) und zeigen Sie dessen Korrektheit.

# Besprechung PÜ

# A1

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 8 | 8 | 8 | 9 | 10 | 10 | 10 | $t$ 10 | 8 |
| 7 | 7 | 7 | ■ | ■ | ■ | 9 | 9 | 9 | 7 |
| 6 | 6 | ■ | 6 | 6 | 6 | ■ | 8 | 8 | 6 |
| 5 | 5 | 5 | 5 | 5 | 5 | ■ | 7 | 8 | 5 |
| 4 | 4 | 4 | 4 | 4 | ■ | 6 | 7 | 8 | 4 |
| 3 | 3 | 3 | 3 | ■ | 5 | 6 | 7 | 8 | 3 |
| 2 | 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 2 |
| 1 | 1 | 2 | 3 | ■ | 5 | 6 | 7 | 8 | 1 |
| $s$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 0 |
| $x$ : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | $y$ : |

A2
a)

**Input:** Zusammenhängender Graph $G = (V, E, c)$ und ein Startknoten $s$, sowie ein Zielknoten $t$

**Output:** `true` falls $G$ in der Zusammenhangskomponente von $v$ einen Zykel enthält, `false` sonst

```
1  S ← {s};
2  d(s), d'(s) ← 0;
3  S' ← OutAdj(s);
4  for u ∈ S' do
5  |   d'(u) ← c(s, u);
6  |   pred(u) ← s;
7  end
8  while S ≠ V do
9  |   wähle w ∈ S', so dass d'(w) ≤_{v∈S'} d'(v);
10 |   d(w) ← d'(w);
11 |   S ← S ∪ {w});
12 |   S' ← S' \ {w};
13 |   for u ∈ OutAdj(E do
14 |   |   if u ∉ (S ∪ S') then
15 |   |   |   S' ← S' ∪ {u};
16 |   |   end
17 |   |   if d(w) + c(w, u) ≤ d'(u) then
18 |   |   |   d'(u) ← d(w) + c(w, u);
19 |   |   |   pred(u) ← w
20 |   |   end
21 |   end
22 end
23 path ←empty stack;
24 v ← t;
25 path.push(t);
26 while v ≠ s do
27 |   v ← pred(v);
28 |   path.push(v);
29 end
30 return path;
```

# A2
# a)

- Der Algorithmus funktioniert wie Dijkstras Algorithmus, nur dass sich jeder Knoten seinen Vorgänger merkt, welcher auf dem kürzesten Pfad von s zum aktiven Knoten führt
- Damit lässt sich dann auch der Pfad von s zu t rekonstruieren, indem man von Vorgänger zu Vorgänger bis zu s zurückwandert. (Es ist hier auch möglich den Algorithmus abzubrechen, sobald t in S eingefügt wurde, da uns nur dieser Pfad interessiert).

# A2
# b)

- Die Laufzeit entspricht der Laufzeit von Dijkstras Algorithmus also $O(n \log(n) + m)$.
- Der Korrektheitsbeweis folgt auch aus dem Korrektheitsbeweis von Dijkstra:
  - Betrachten wir einen Knoten v.
  - Angenommen es gibt einen Knoten $w \neq pred(v)$, mit welchem v über einen kürzeren Pfad erreicht werden kann, also $d(pred(v)) + c(pred(v), v) > d(w)+c(w, v)$ so würde w in Zeile 19 des Pseudocodes pred(v) überschreiben und nicht wieder überschrieben werden, dies ist ein Widerspruch zu unserer Annahme. Somit kennt jeder Knoten v seinen Vorgänger auf einem optimalen Weg von s zu v, entsprechend erhalten wir auch den optimalen Weg von s zu t.

# Check-in nächste Woche