Neue Gruppen.

Group	Names	Emails
Group 1	Matin Kamil Saih Durvish Battini	kamilm@student.ethz.ch dbattini@student.ethz.ch
Group 2	Paul Iurian Silvan Wohlgensinger	piurian@student.ethz.ch swohlgensing@student.ethz.ch
Group 3	Max Schaffner Lucien Isler	schaffnerma@student.ethz.ch islerlu@student.ethz.ch
Group 4	Sih-Cheng Chen Marc Leonard Overbeck	chensih@student.ethz.ch moverbeck@student.ethz.ch
Group 5	Nelio Gautschi Nico Reali	ngautsch@student.ethz.ch nreali@student.ethz.ch
Group 6	Besart Memisi Raghav Tandon	bmemisi@student.ethz.ch tandonr@student.ethz.ch
Group 7	Floria Schwedt Lauren Thai	fschwedt@student.ethz.ch lathai@student.ethz.ch
Group 8	Zhentian Qi Nicolas Fierz	zhenqi@student.ethz.ch nfierz@student.ethz.ch
Group 9	Elaine Banaag Aquino Lucien Gees	ebanaag@student.ethz.ch lugees@student.ethz.ch
Group 10	Noa Kündig Colin Jenny Noel Hobi	nkuendig@student.ethz.ch cojenny@student.ethz.ch hobino@student.ethz.ch
Group 11	Simon Spycher Gréta Frei	sispycher@student.ethz.ch grfrei@student.ethz.ch
Group 12	Benjamin Stupf Charlyne Dippon	bstupf@student.ethz.ch cdippon@student.ethz.ch

Discussion of Exercise Sheet 9

Exercise 9.3 *Number of paths in DAGs* **(2 bonus points).** → Peergrading heute

Let $G = (V, E)$ be a DAG with $n = |V|$ vertices and $m = |E|$ edges, and let $s, t \in V$ be distinct vertices. Describe a bottom-up dynamic programming algorithm that returns the number of paths from s to t in $O(n + m)$ time. Assume that G is given by adjacency lists, where $Adj[u]$ contains all vertices v with $(u, v) \in E$. You may use the algorithm for topological sorting from the lecture.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

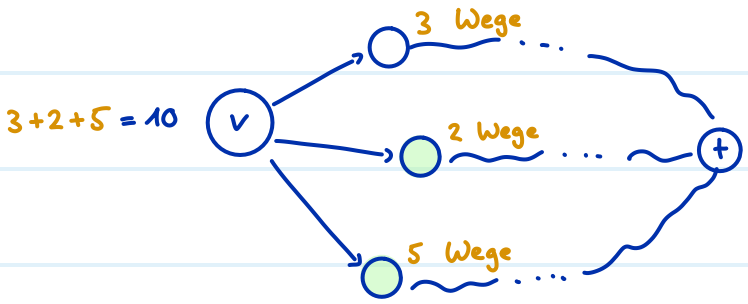
Hint: *Define the entry $DP[v]$ as the number of paths in G from v to t.*

Dimension: DP[1 ... n]

Teilproblem: DP[v] := number of paths in G from v to t

Base Case: DP[t] = 1 → der leere Weg von t nach t

Rekursion: $DP[v] = \sum_{u \in Adj[v]} DP[u]$ for all $v \neq t$



(+ Justification)

Berechnungsreihenfolge: for $i = n \dots 1$: (d.h. topologische Sortierung absteigend)
compute DP[i]

Lösung: DP[s]

Laufzeit: Das bestimmen der topologischen Sortierung braucht $O(n+m)$ Zeit

Berechnung für $DP[i] = \Theta(1 + deg_{out}(v_i))$

Berechnung für gesamte Tabelle: $\sum_{i=1}^{n} \Theta(1 + deg_{out}(v_i))$

$= \Theta(n) + \sum_{i=1}^{n} \Theta(deg_{out}(v_i))$ ← Handschlaglemma

$= \Theta(n) + \Theta(m)$

$= \Theta(n+m)$

Auslesen der Lösung in $\Theta(1)$

Total: $\underline{\underline{\Theta(n+m)}}$

→ wichtige Aufgabe für Prüfung

Gewichtete Digraphen

$c(e)$ ist das Gewicht / die Kosten der Kante e (manchmal auch $c(u,v)$ geschrieben)

und darf auch negativ sein

$c(W)$ ist das Gewicht / die Kosten des Weges W, d.h. die Summe aller Kantengewichte von W

$d(u,v)$ ist die Distanz von u nach v, d.h. die kleinstmöglichen Kosten eines Weges von u nach v

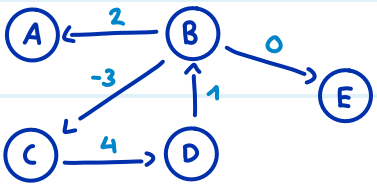
↳ das "Infimum" kann auch $-\infty$ sein

($:= \infty$ falls es keinen Weg von u nach v gibt)

Annahme: es gibt keine negativen Zyklen

Achtung: Länge = { Anzahl Kanten — für ungewichtete Graphen; Summe aller Kantengewichte — für gewichtete Graphen

Bsp



$c(B,A) =$

$d(C,E) =$

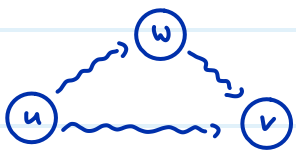
$d(D,C) =$

Lemma: Sei G ein gewichteter Digraph ohne negative Zyklen. Wenn wir einen Weg W zu einem Pfad P abkürzen, dann gilt: $c(P) \leq c(W)$ (Kosten können niemals grösser werden)

Dreiecksungleichung: Für alle $u,v,w \in V$ gilt: $d(u,v) \leq d(u,w) + d(w,v)$

weil $d(u,v)$ ist ja das Minimum der Kosten aller u-v-Wege

und dazu gehören auch alle u-v-Wege, die w benutzen



Aufgaben

Let $G = (V, E)$ be a directed graph with weight function c.

True or False: If there is a directed edge (u, v) with weight $c(u, v) = 5$, then $d(u, v)$ must be equal to 5.

True or False: For any vertex v, $d(v, v) = 0$

True or False: For any vertex v, $d(v, v) = 0$ (assuming no negative cycles).

True or False: If $d(u, w) = 5$ and $d(w, v) = -2$, then $d(u, v)$ must be equal to 3.

True or False: If G contains a negative cycle, then for all vertices v, $d(v, v) < 0$.

True or False: If G contains a negative cycle, then for some vertex v, $d(v, v) < 0$.

True or False: If G contains a negative cycle, then there exists a walk W with $c(W) = -\infty$.

True or False: Let $P = (p_1, p_2, \ldots, p_k)$ be a path. Then $d(p_1, p_{k-1}) \leq d(p_1, p_k)$.

True or False: It can happen that $d(u, v) = \infty$ and $d(v, u) = -\infty$. (Say something about proof by example)

True or False: If there is a directed edge (u, v) and u is part of a negative cycle, then $d(u, v) < c(u, v)$.

True or False: Assume that every node is part of a negative cycle. Then $d(u, v) = -\infty$ for all $u, v \in V$.

True or False: In a DAG, the distance $d(u, v)$ is always the cost of the path from u to v that uses the least number of edges.

True or False: If w is a vertex on a shortest path from u to v, then $d(u, v) = d(u, w) + d(w, v)$.

True or False: Let P be a shortest path from u to v. Assume that $w \notin P$. Then $d(u, w) + d(w, v) > d(u, v)$

Kürzeste Wege in DAGs

- Finde eine topologische Sortierung $v_1, .., v_n$
- Sei $s = v_k$ der Startknoten

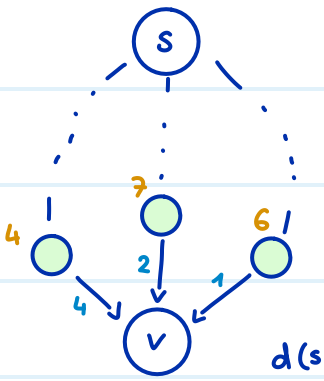
Dimension: $DP[1 \dots n]$

Teilproblem: $DP[v] := d(s,v)$

Base Case: $DP[s] = 0$ → der leere Weg von s nach s hat Distanz 0

Alle anderen DP-Einträge werden mit ∞ initialisiert

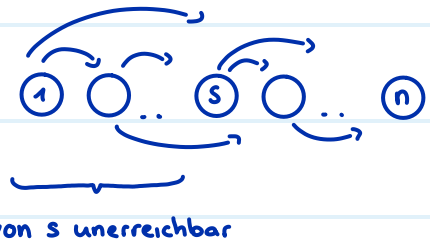
Rekursion: $DP[v] = \min \{ DP[u] + c(u,v) \mid (u,v) \in E \}$ for all $v \neq s$



$d(s,v) = \min \{ 4+4, 7+2, 6+1 \}$

$= 7$

Berechnungsreihenfolge: for i = s ... n:
compute DP[i]



Lösung: for all $v \in V$: $d(s,v) = DP[v]$

Laufzeit: $O(n+m)$ (Trick: Adjazenzliste umdrehen, gleich wie letzte Woche)

Bellman - Ford

- Es darf negative Kanten geben
- Sei s der Startknoten

Dimension: $T[1 \dots n][0 \dots n-1]$

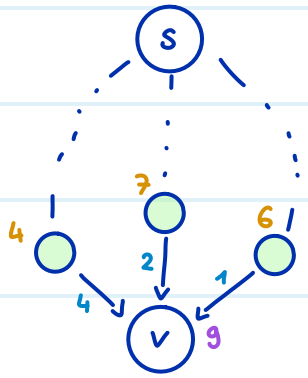
Teilproblem: $T[v, k]$:= Kosten eines günstigsten s-v-Weges mit $\leq k$ Kanten

Base Case: $T[s, 0] = 0$ → der leere Weg von s nach s hat Kosten 0

$T[v, 0] = \infty$ für alle $v \neq s$ → nicht erreichbar mit 0 Kanten

Rekursion:

$$T[v, k] = \min \left\{ \underbrace{T[v, k-1]}_{\text{wir brauchen die k-te Kante gar nicht}}, \underbrace{\min_{(u,v) \in E} \{ T[u, k-1] + c(u,v) \}}_{\text{wir brauchen eine k-te Kante. Wir wählen diejenige Kante (u,v) sodass der Weg von s zu u zu v am günstigsten ist}} \right\} \quad \text{für } k \geq 1$$



= $T[u, k-1]$

= $T[v, k-1]$

$$T[v, k] = \min \{ 9, \min \{ 4+4, 7+2, 6+1 \} \}$$

$$= 7$$

Berechnungsreihenfolge:

```
for k = 0 . n-1 :
    for v = 1 ... n :
        compute T[v, k]
```

Lösung: for all $v \in V$: $d(s,v) = T[v, n-1]$

Laufzeit: $O(n(n+m))$

- n: n Reihen im DP Table
- $(n+m)$: pro Reihe laufen wir alle Kanten/Knoten einmal durch (wie bei DFS/BFS)

Falls G zusammenhängend ist, dann gibt es mindestens $m \geq n-1$ Kanten,

d.h $n \leq O(m)$, und somit vereinfacht sich die Laufzeit zu $O(nm)$

Wie herausfinden ob es negative Zyklen gibt?

Wie viele Kanten kann ein Pfad höchstens verwenden? -> $n-1$ weil kein Knoten mehrfach vorkommen darf n Knoten = $n-1$ Kanten

=> Jeder Weg mit n Kanten muss einen Knoten mehrfach verwenden

=> Falls $T[v,n] < T[v,n-1]$, dann werden die Kosten kleiner, obwohl wir einen Knoten mehrfach verwenden

=> Der Teilweg vom ersten bis zum zweiten Vorkommen ist ein negativer geschlossener Weg

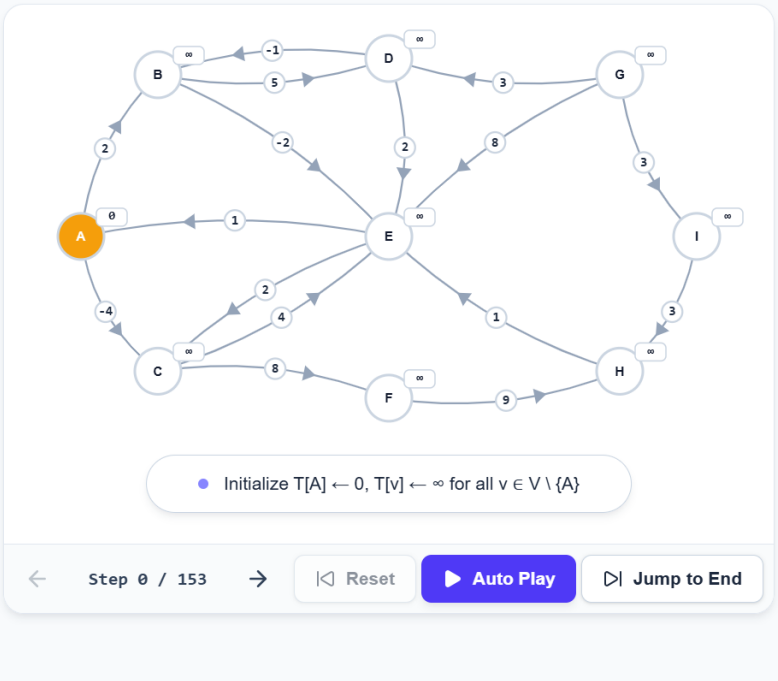
=> $\exists$ negativer Zyklus

Lemma: $T[v,n] < T[v,n-1]$ für ein $v \in V$ $\Rightarrow$ $\exists$ negativer Zyklus

Version mit optimiertem Speicher:

BELLMAN-FORD(s)

1 $T[s] \leftarrow 0; \quad T[v] \leftarrow \infty$ für alle $v \in V \setminus \{s\}$
2 **for** $k \leftarrow 1, \ldots, n-1$ **do**
3 **for** $v \in V$ **do**
4 $T[v] \leftarrow \min\left\{T[v], \min_{(u,v)\in E}\left(T[u] + c(u,v)\right)\right\}$
5 **return** T



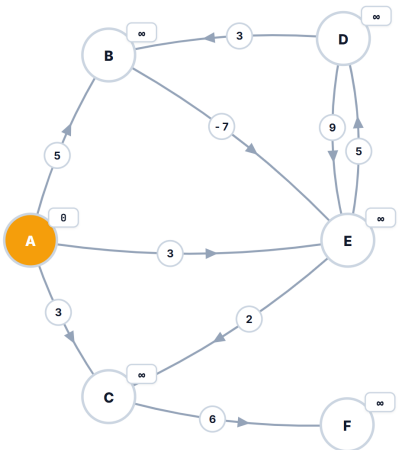
BELLMAN FORD $\mathcal{O}(n \cdot m)$

```
1  T[s] ← 0; T[v] ← ∞ für alle v ∈ V \ {s}
2  for k ← 1, ..., n - 1 do
3    for v ∈ V do
4      T[v] ← min { T[v], min_(u,v)∈E (T[u] + c(u, v)) }
5  return T
```

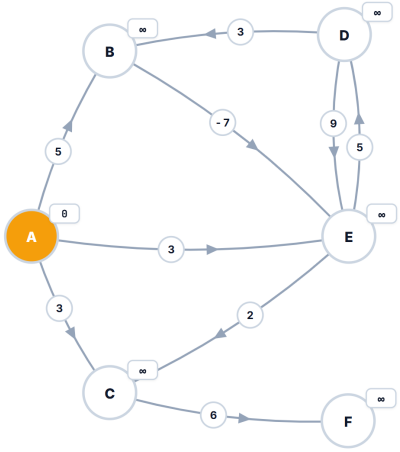
SHORTEST PATH TREE



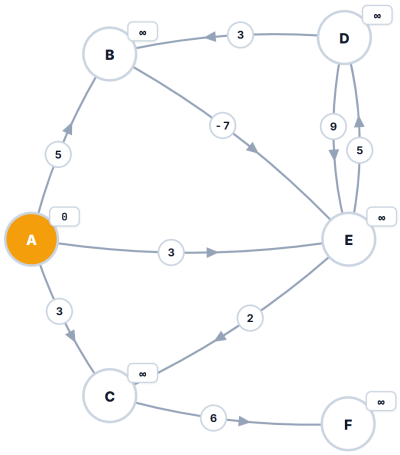
Aufgabe: Führe Bellman-Ford (optimierte version) aus und zeichne den shortest path tree



Nach der ersten Iteration (k=1):



Nach der zweiten Iteration (k=2)



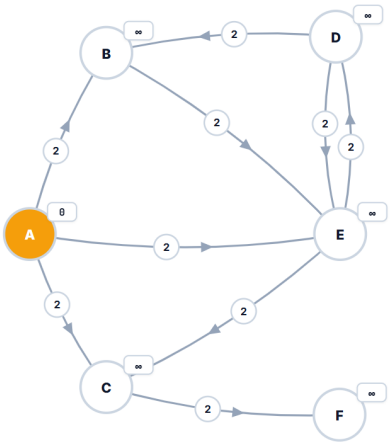
Nach der dritten Iteration (k=3)

Shortest path tree:

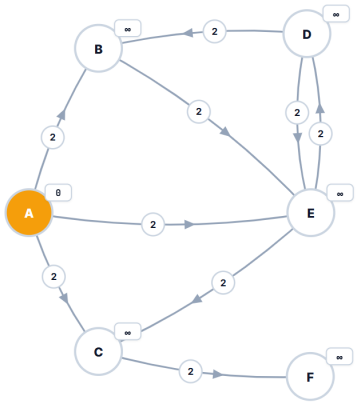
Aufgabe: Wie müssen wir den BFS Pseudocode modifizieren, um die günstigsten Wege in diesem Graphen zu berechnen?

BFS $\mathcal{O}(n+m)$

```
1   dist[s] ← 0
2   dist[v] ← ∞ for all v ≠ s
3   Q ← ∅
4   ENQUEUE(s, Q)
5   while Q ≠ ∅ do
6     u ← DEQUEUE(Q)
7     for each (u, v) ∈ E do
8       if dist[v] == ∞ then
9         dist[v] ← dist[u] + 1
10        ENQUEUE(v, Q)
```



Aufgabe: Führe diese modifizierte BFS aus und zeichne den shortest path tree



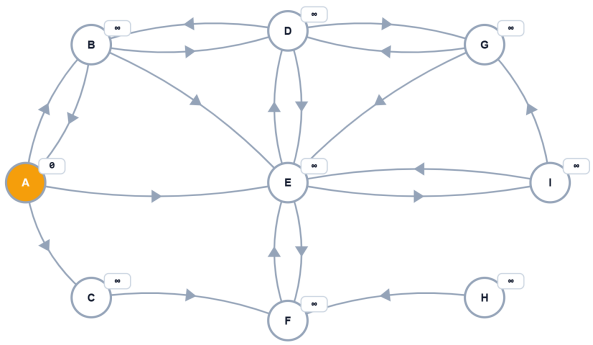
Shortest path tree:

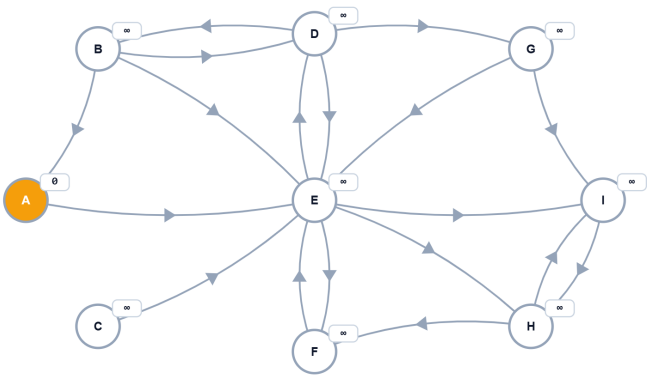
Trick für sehr schnelles Ausführen von BFS:

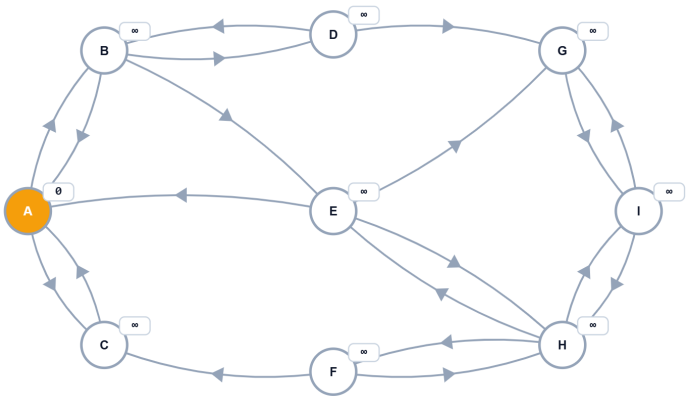
- direkt den shortest path tree zeichnen, und ihn implizit als Queue benutzen

  (d.h. Kinder eines Knotens im shortest path tree alphabetisch sortieren, und von links nach rechts Level für Level durchgehen)

Bsp:





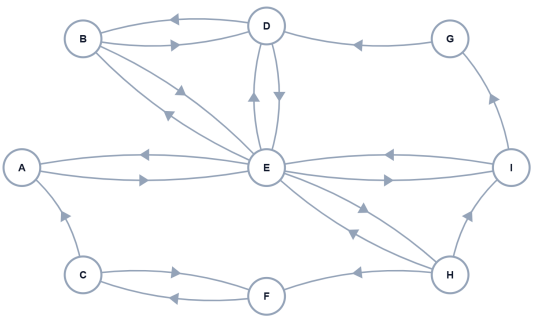


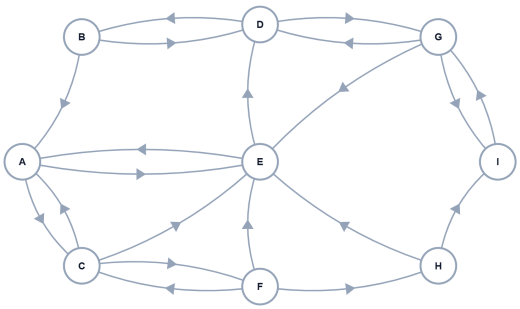
Kann man auch für DFS machen

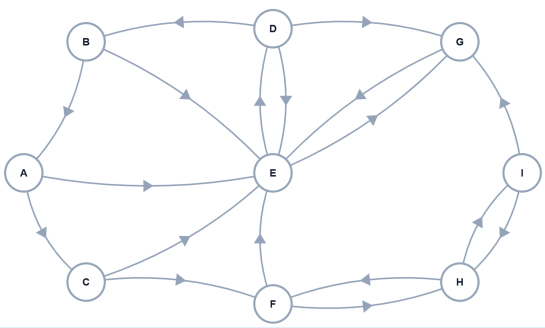
- direkt den shortest path tree zeichnen, und ihn implizit als Rekursions Stack benutzen

  (d h Kinder eines Knotens im DFS tree alphabetisch sortieren, und immer möglichst unten links weitermachen)

Bsp:







Aufgaben:

True or False: Suppose we add a constant $C > 0$ to the weight of every edge in a directed graph G (without negative cycles). The shortest path between any two vertices remains unchanged.

True or False: Suppose we multiply each weight of every edge in with a constant $C > 0$ in a directed graph G (without negative cycles). The shortest path between any two vertices remains unchanged.

True or False: Suppose we run Bellman-Ford from a starting vertex s on a directed graph with n vertices. If there is a negative cycle in G, then $T[v, n] < T[v, n-1]$ will hold for all vertices $v \in V$.

True or False: Suppose we run Bellman-Ford from a starting vertex s on a directed graph with n vertices. If there is a negative cycle in G, then $T[v, n] < T[v, n-1]$ will hold for at least one vertex $v \in V$.

True or False: Suppose we run Bellman-Ford from a starting vertex s on a directed graph with n vertices. If there is a negative cycle in G and v is on that cycle, and v is reachable from s then $T[v, n] < T[v, n-1]$.

True or False: In a directed graph with no negative cycles, if there is a shortest path from s to t , then for every vertex v on that path, the subpath from s to v is a shortest path from s to v.

True or False: If there are no negative cycles and the longest path (counting edges) has length 4, then we only have to do 4 iterations of Bellman Ford to compute the distances.

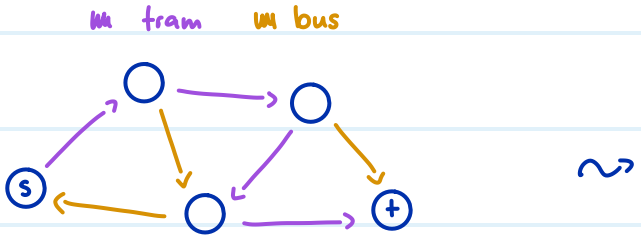
True or False: If no values change after 4 iterations of Bellman Ford, then we know that there is no path longer than length 4 (counting edges).

You want to travel from the main station s to your favourite exercise class in CHN F 42 (t) using Zürich's public transit network, which consists of n stations and m one-way transit connections.

There are two types of connections:

- Tram connections. They take 1 min.
- Bus connections. They take 2 min.

Define a graph and an algorithm that runs in $O(n+m)$ to compute the minimum travel time.



Graph:

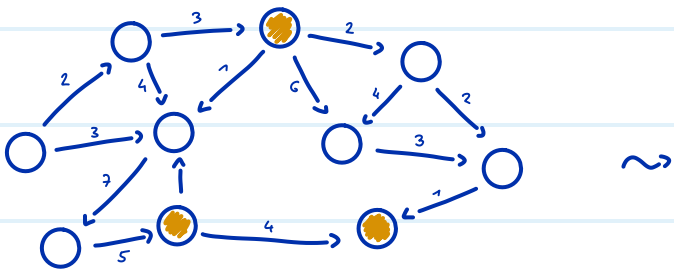
Algorithm:

Runtime:

An earthquake happened. There are n houses and m one-way streets between the houses. There are k safe bunkers among these houses. Traversing each way requires a specific amount of time (in minutes). The residents of every house run to the bunker they can reach the fastest. Assume that all houses are connected.

Define a graph and an algorithm that runs in $O(nm)$ to compute the minimum time required for everyone to reach a bunker.

m bunker



Graph:

Algorithm:

Runtime: