Bellman-Ford

Bedingungen: negative Kantengewichte sind erlaubt, aber es darf keinen negativen Zyklus geben

$S_{\leq \ell} = \{ v \in V \mid \exists$ ein kürzester (= günstigster) Weg von s zu v mit $\leq \ell$ Kanten $\}$

ℓ-genaue Schranke: alle kürzesten (= günstigsten) Wege mit $\leq \ell$ Kanten sind bereits gefunden

d.h $\quad d[v] = d(s,v) \quad \forall v \in S_{\leq \ell}$

$\quad\quad d[v] \geq d(s,v) \quad \forall v \notin S_{\leq \ell}$

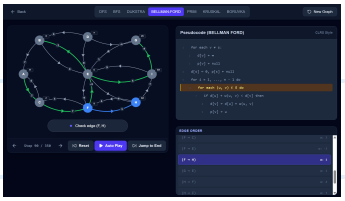
BELLMAN-FORD$(G = (V, E), s)$ in $O(n \cdot m)$

```
1  for each v ∈ V\{s} do                    ▷ Initialisiere für alle Knoten die
2      d[v] ← ∞; p[v] ← null                ▷ Distanz zu s sowie Vorgänger
3  d[s] ← 0; p[s] ← null                    ▷ Initialisierung des Startknotens
                                              } 0-genaue Schranken
4  for i ← 1, 2, ..., |V| − 1 do            ▷ Wiederhole |V| − 1 Mal
5      for each (u, v) ∈ E do               ▷ Iteriere über alle Kanten (u, v)
6          if d[v] > d[u] + w((u, v)) then  ▷ Relaxiere Kante (u, v)
7              d[v] ← d[u] + w((u, v))      ▷ Berechne obere Schranke
8              p[v] ← u                     ▷ Speichere u als Vorgänger von v
9  for each (u, v) ∈ E do                   ▷ Prüfe, ob eine weitere Kante
10     if d[u] + w((u, v)) < d[v] then      ▷ relaxiert werden kann
11         Melde Kreis mit negativem Gewicht
```

Invariante: i-genaue Schranken nach der i-ten Iteration

Achtung: Reihenfolge der Kanten ist nicht vorgegeben (→ Quizfrage)

$\exists$ neg Zyklus falls sich nach der n-ten Iteration eine Distanz ändert



MST

Sei $G = (V, E)$ ein zusammenhängender Graph

Baum: ungerichteter, zusammenhängender, kreisfreier Graph

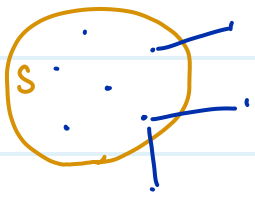
Aufspannende Kanten $A \subseteq E$ falls $G(V, A)$ zusammenhängend ist

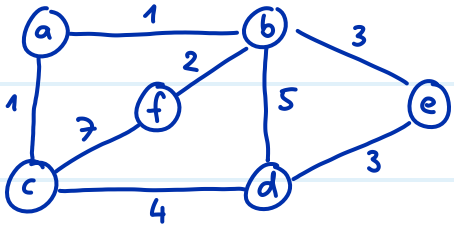
Spannbaum $T \subseteq E$ falls T aufspannend ist und keinen Kreis hat

Minimaler Spannbaum $MST \subseteq E$ falls MST ein Spannbaum mit minimalem Gewicht ist

Sichere Kante $e \in E$ falls e in jedem MST vorkommt

- Kante mit kleinstem Gewicht an jeden Knoten ist sicher
- Kante mit kleinstem Gewicht an jeden Schnitt ist sicher





Ist $A = \{\{a,b\}, \{a,c\}, \{f,b\}, \{e,d\}\}$ aufspannend?

Welche Kanten könnte man zu A hinzufügen, sodass A aufspannend ist?

Welche Kanten könnte man zu A hinzufügen, sodass A ein Spannbaum ist?

Welche Kanten könnte man zu A hinzufügen, sodass A ein minimaler Spannbaum ist?

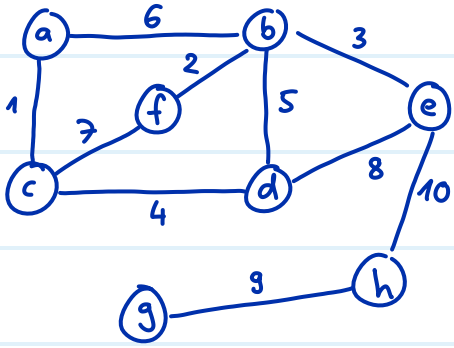
Welches ist die Kante mit kleinstem Gewicht an Schnitt $S = \{a, b, d\}$

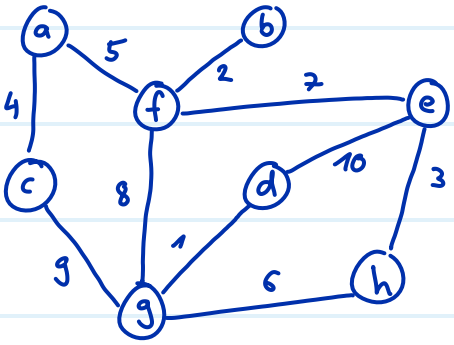
Wie viele sichere Kanten gibt es?

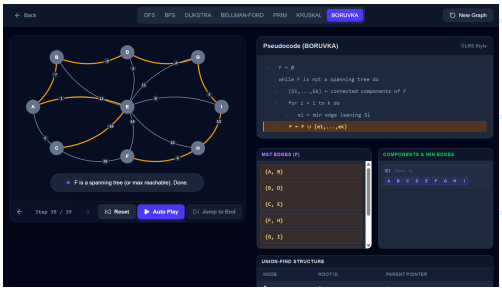
Algorithm 1 BORUVKA(G) in $O((n+m)\cdot \log n)$

1: $F \leftarrow \emptyset$
2: **while** F is not a spanning tree **do**
3: $\quad (S_1, \ldots, S_k) \leftarrow$ connected components of F
4: $\quad (e_1, \ldots, e_k) \leftarrow$ minimum edges leaving $S_1, \ldots, S_k$
5: $\quad F \leftarrow F \cup \{e_1, \ldots, e_k\}$

Bedingung: paarweise verschiedene Kantengewichte





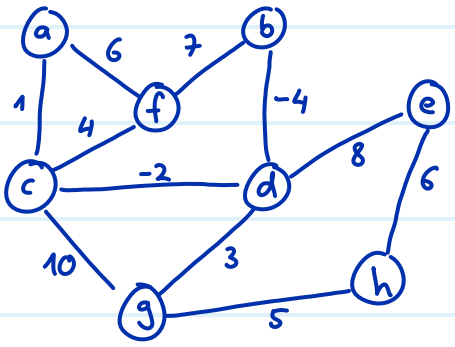


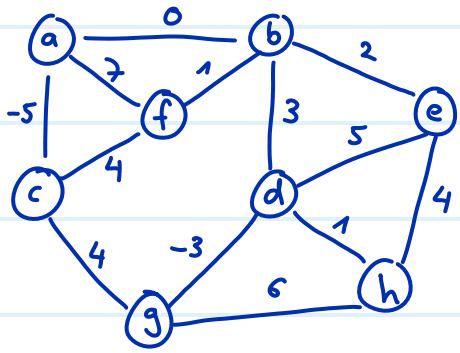
Algorithm 1 PRIM(G, s) in $O((n+m)\log n)$

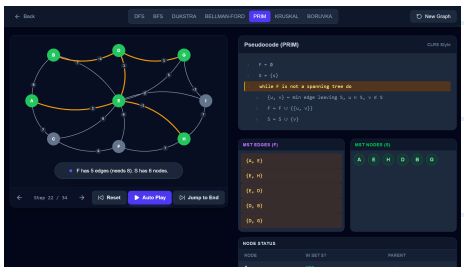
1: $F \leftarrow \emptyset$
2: $S \leftarrow \{s\}$
3: **while** F is not a spanning tree **do**
4: $\quad \{u, v\} \leftarrow$ minimale Kante an S, mit $u \in S,\ v \notin S$
5: $\quad F \leftarrow F \cup \{\{u, v\}\}$
6: $\quad S \leftarrow S \cup \{v\}$

- mehrere Kanten mit gleichem Gewicht erlaubt

Bsp Startknoten a



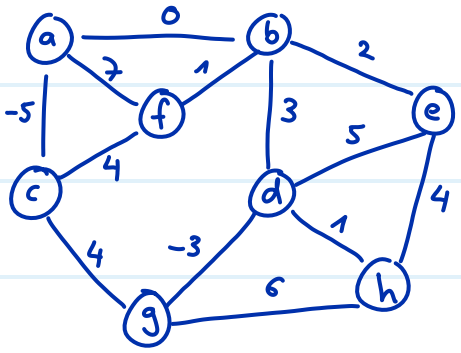




- **Dijkstra:** Read the blue lines and ignore the orange ones.
- **Prim:** Read the orange lines and ignore the blue ones.

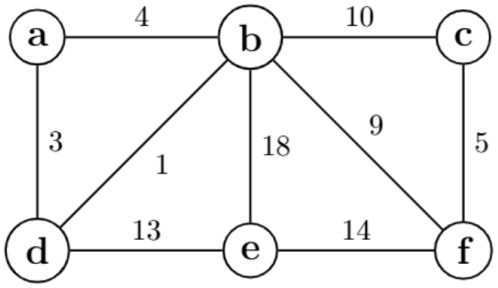
Algorithm 1 DIJKSTRA / PRIM$(G = (V, E), s)$ in $O((n + m) \cdot \log n)$

```
1:  for each v ∈ V \ {s} do                          ▷ Initialisiere für alle Knoten
2:      d[v] ← ∞; p[v] ← null                        ▷ Distanz/Kosten sowie Vorgänger
3:  d[s] ← 0; p[s] ← null                            ▷ Init. Startknoten (Dijkstra) / Wurzel (Prim)
4:  Q ← ∅                                            ▷ Leere Prioritätswarteschlange Q
5:  INSERT(s, 0, Q)                                  ▷ Füge s zu Q hinzu
6:  while Q ≠ ∅ do
7:      u ← EXTRACT-MIN(Q)                           ▷ Wähle Knoten mit min. d-Wert
8:      for each {u, v} ∈ E do                       ▷ Inspiziere Nachfolger
9:          if p[v] = null then                      ▷ v wurde noch nicht entdeckt
10:             d[v] ← d[u] + w({u, v})              ▷ Dijkstra: Berechne Distanz
11:             d[v] ← w({u, v})                     ▷ Prim: Berechne Kosten
12:             p[v] ← u                             ▷ Speichere u als Vorgänger
13:             ENQUEUE(v, d[v], Q)                  ▷ Füge v zu Q hinzu
14:         else if d[u] + w({u, v}) < d[v]    w({u, v}) < d[v] then
15:             d[v] ← d[u] + w({u, v})              ▷ Dijkstra: Update Distanz
16:             d[v] ← w({u, v})                     ▷ Prim: Update Kosten
17:             p[v] ← u                             ▷ Vorgänger aktualisieren
18:             DECREASE-KEY(v, d[v], Q)             ▷ Priorität anpassen
```



3 min / Punkt

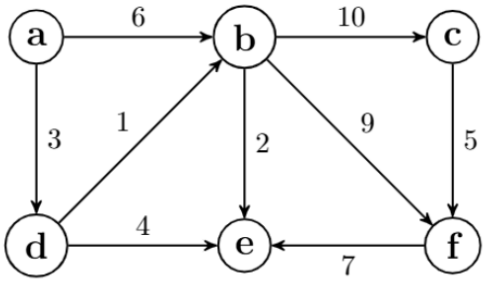
/ 2 P e) *Minimum Spanning Tree:* Consider the following graph:



i) Highlight the edges that are part of the minimum spanning tree. (Either in the picture above, or you can recreate the graph below).

ii) Write out all positive integers x such that if we replace the weight 1 of edge $\{b, d\}$ in the above graph with x, then edge $\{b, d\}$ would be in at least one minimum spanning tree of the resulting graph.

/ 2 P f) *Shortest Path Tree:* Consider the following graph:



i) Highlight the edges that are part of the shortest-path tree rooted at vertex a (i.e., the output of Dijkstra's algorithm if we were to start from vertex a).

ii) Does the above graph have a topological ordering? If yes, write down one topological ordering. If no, give an argument.

Theory Task T4. / 14 P

The city center of Amsterdam is known for its many tourists and many bridges. There are $n \geq 4$ tourist attractions in Amsterdam labeled $\{1, 2, \ldots, n\}$. For some pairs $i, j \in \{1, 2, \ldots, n\}$, the attractions i, j are connected by a two-way bridge, represented by $\{i, j\}$. The set of all bridges is denoted with $B = \{\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_m, j_m\}\}$. You may assume that $m \geq n$. Using the bridges, **any attraction i can be reached from any other attraction j.**

Remark: When asked to describe an algorithm, you need to address the following aspects:

1) the graph algorithm that you use as a subroutine in your solution;
2) the construction of the graph that you run this algorithm on;
3) the correctness of your solution, with a short justification (i.e., how and why does running the chosen graph algorithm on the graph you constructed solve the original problem);
4) the total running time of your solution, with a short justification.

You can directly use the algorithms covered in the lecture material, and you can directly use their running time bounds without proof.

Remark: It is not needed to solve part a) to attempt parts b) and c). It is not needed to solve any earlier parts to attempt part d).

/ 3 P

a) To accommodate for boats, the city government wants to remove some of the bridges. Each bridge $b_k = \{i_k, j_k\}$ has a *length* $L_k \in \mathbb{N}$. Describe an algorithm which outputs a subset $D \subseteq B$ of bridges, of largest possible total length $L(D) := \sum_{k:b_k \in D} L_k$, which can be removed while still making sure that tourists can travel between any pair of attractions $i, j \in \{1, 2, \ldots, n\}$ (without swimming). The running time of your algorithm should be $O(m \log m)$. ¡¡¡¡¡¡¡ HEAD

1)

2)

3)

4)

It turns out the bridges are historical buildings, and so they cannot be removed. Instead, the government decides to only open each bridge during some parts of the day. They divide a day into $N \geq 2$ units of time, labeled $\{0, 1, \ldots, N-1\}$, and give each bridge $b_k = \{i_k, j_k\}$ *opening times* $O_k \subseteq \{0, 1, \ldots, N-1\}$. Each bridge also has a *crossing time* $C_k \in \mathbb{N}$, which represents the number of time-units required to cross it. **Tourists are only allowed to start crossing a bridge when it is open** (but it is not a problem if it closes while they are crossing it). Note that a tourist's travel is allowed to span multiple days. For example, they might start to cross a bridge b_k with crossing time $C_k = 4$ at time $T = N - 2$, and finish the next day at time $T' = 2$ (as long as $N - 2 \in O_k$).

/ 4 P

b) We are given two attractions $s, t \in \{1, 2, \ldots, n\}$, and a starting time $T_{\text{start}} \in \{0, 1, \ldots, N-1\}$. Describe an algorithm which decides if it is possible for tourists to travel from s to t, starting at time T_{start}, while respecting the opening times of the bridges, but **without having to wait** at any of the attractions. The running time of your algorithm should be $O(m \cdot N)$.

Hint: *Use a directed graph whose vertex set consists of N copies of the attractions $\{1, 2, \ldots, n\}$, labeled $\{(i, T) : i \in \{1, 2, \ldots, n\}, T \in \{0, 1, \ldots, N-1\}\}$. When should there be an edge from (i, T) to (i', T')?*

1)

2)

3)

4)

/ 4 P

d) We are given two attractions $s, t \in \{1, 2, \ldots, n\}$ and a starting time $T_{\text{start}} \in \{0, 1, \ldots, N-1\}$. Describe an algorithm which outputs the minimum amount of time (in time-units) required for tourists to travel from s to t, starting at time T_{start} (or ∞ when no such travel is possible). Now, **tourists are willing to wait** at an attraction for a bridge to open if this reduces their total travelling time. The running time of your algorithm should be $O(mN \cdot \log(mN))$.

Hint: *Use a weighted, directed graph with the same vertex set as in part b). Which edges should you add to model tourists waiting at an attraction for a bridge to open?*

Based on negative feedback, the government decides to **open all bridges during the entire day**. In an attempt to steer traffic, they release a traveller's guide which rates how nice the view is from each bridge. As the view depends on the direction in which a bridge $b_k = \{i_k, j_k\}$ is crossed, the guide lists two numbers $R_k^+, R_k^- \in \mathbb{Z}$ (i.e, they are **possibly negative** integers). Assuming $i_k < j_k$, R_k^+ corresponds to crossing b_k from i_k to j_k, and R_k^- corresponds to crossing b_k from j_k to i_k. To avoid tourists getting lost, the government made sure that any walk around Amsterdam which starts and ends in the same attraction has total rating strictly less than 0.

/ 3 P

d) We are given two attractions $s, t \in \{1, 2, \ldots, n\}$. Describe an algorithm which outputs the **maximum** possible total rating of a walk from s to t. The running time of your algorithm should be $O(nm)$.

Hint: *Use a weighted, directed graph whose vertex set consists of the attractions $\{1, 2, \ldots, n\}$. (The opening times O_k and crossing times C_k do not play any role in this part of the exercise!)*