Informatik II

Übung 9

FS 2020

Program Today

1. Repetition BFS
2. Repetition of Lecture
3. In-Class-Exercise (practical)

Python: Datastructures

- Lists, Stacks, Queues:
  `https://docs.python.org/3/tutorial/datastructures.html`
- Heap: `https://docs.python.org/3/library/heapq.html`
- (Synchronized) Queue, PriorityQueue (=Heap):
  `https://docs.python.org/3/library/queue.html`

BFS

```
def BFS(v):
  color = {v: Grey} # white if not contained
  queue = Queue();
  queue.put(v);
  while not queue.empty():
    u = queue.get()
    for e in u.edges:
      w = e.target
      if w not in color:
        color[w] = Grey
        print(w)
        queue.put(w)
    color[u] = Black
```

2. Repetition of Lecture

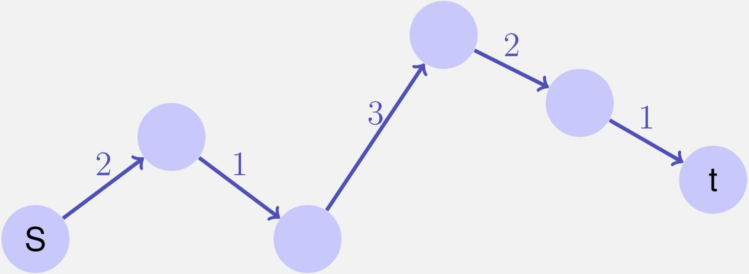
Weighted Graphs

Given: $G = (V, E, c)$, $c : E \to \mathbb{R}$, $s, t \in V$.

Wanted: Length (weight) of a shortest path from s to t.

Path: $p = \langle s = v_0, v_1, \ldots, v_k = t \rangle$, $(v_i, v_{i+1}) \in E$ $(0 \leq i < k)$

Weight: $c(p) := \sum_{i=0}^{k-1} c((v_i, v_{i+1}))$.



Path with weight 9

Shortest Paths

Weight of a shortest path from u to v:

$$\delta(u,v) = \begin{cases} \infty & \text{no path from } u \text{ to } v \\ \min\{c(p) : u \overset{p}{\rightsquigarrow} v\} & \text{sonst} \end{cases}$$

General Algorithm

1. Initialise d_s and π_s: $d_s[v] = \infty$, $\pi_s[v] =$ null for each $v \in V$
2. Set $d_s[s] \leftarrow 0$
3. Choose an edge $(u, v) \in E$

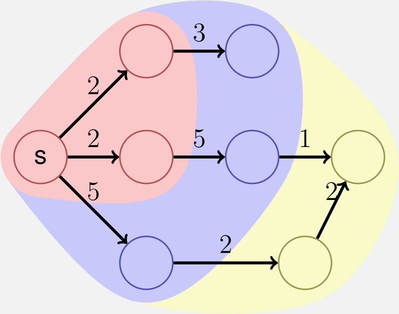
```
Relaxiere $(u, v)$:
    if $d_s[v] > d[u] + c(u, v)$ then
        $d_s[v] \leftarrow d_s[u] + c(u, v)$
        $\pi_s[v] \leftarrow u$
```

4. Repeat 3 until nothing can be relaxed any more.
   (until $d_s[v] \leq d_s[u] + c(u, v) \quad \forall (u, v) \in E$)

Dijkstra (positive egde weights)

Set V of nodes is partitioned into

- the set M of nodes for which a shortest path from s is already known,
- the set $R = \bigcup_{v \in M} N^+(v) \setminus M$ of nodes where a shortest path is not yet known but that are accessible directly from M,
- the set $U = V \setminus (M \cup R)$ of nodes that have not yet been considered.

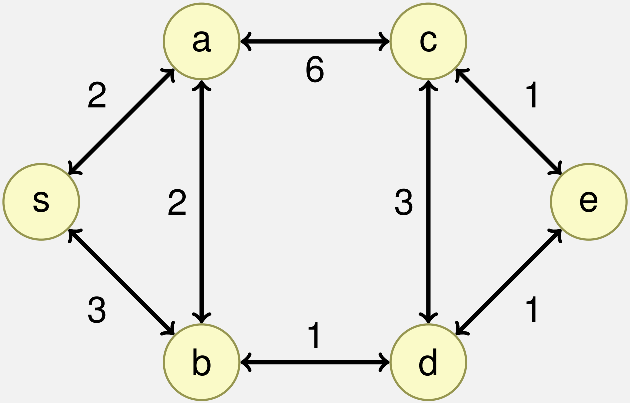


Algorithm Dijkstra(G, s)

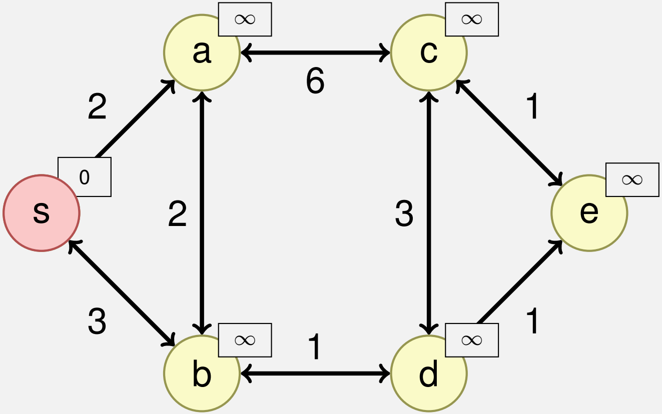
Input: Positively weighted Graph $G = (V, E, c)$, starting point $s \in V$,
Output: Minimal weights d of the shortest paths and corresponding predecessor node for each node.

```
foreach $u \in V$ do
    $d_s[u] \leftarrow \infty$; $\pi_s[u] \leftarrow$ null
$d_s[s] \leftarrow 0$; $R \leftarrow \{s\}$
while $R \neq \emptyset$ do
    $u \leftarrow$ ExtractMin($R$)
    foreach $v \in N^+(u)$ do
        if $d_s[u] + c(u, v) < d_s[v]$ then
            $d_s[v] \leftarrow d_s[u] + c(u, v)$
            $\pi_s[v] \leftarrow u$
            $R \leftarrow R \cup \{v\}$
```

Example



Example

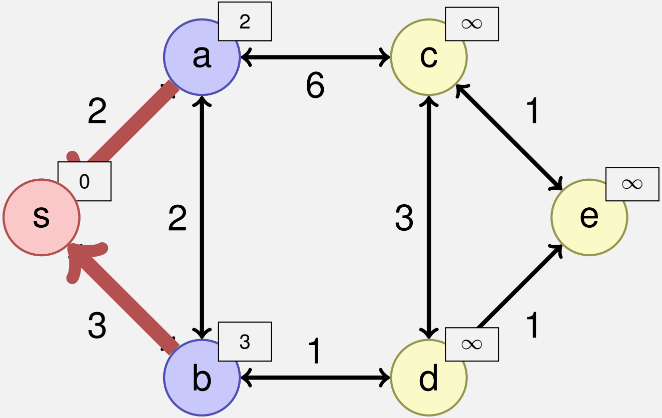


$$M = \{s\}$$

$$R = \{\}$$

$$U = \{a, b, c, d, e\}$$

Example

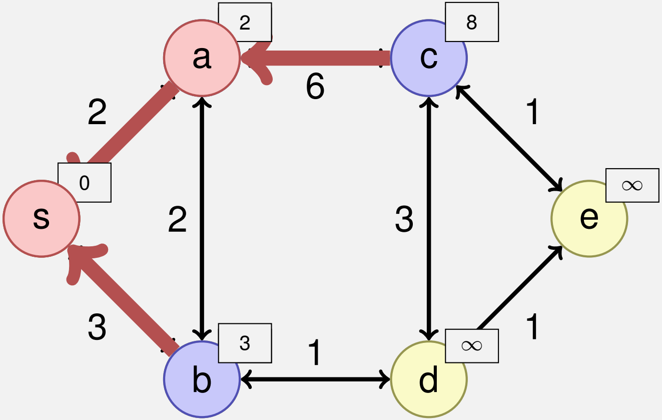


$$M = \{s\}$$

$$R = \{a, b\}$$

$$U = \{c, d, e\}$$

Example

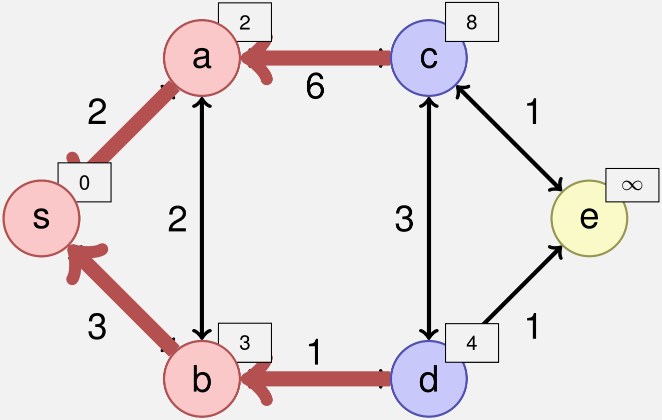


$$M = \{s, a\}$$

$$R = \{b, c\}$$

$$U = \{d, e\}$$

Example

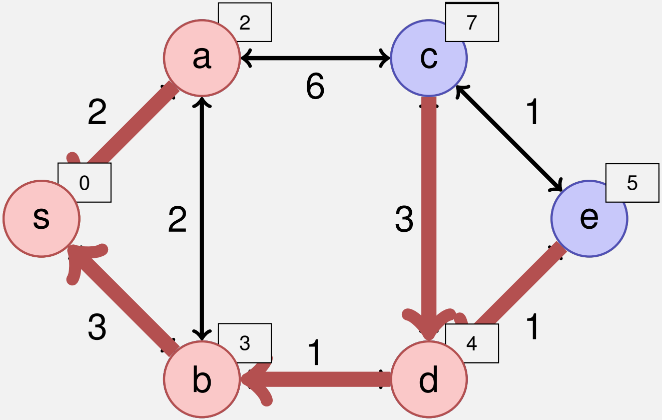


$$M = \{s, a, b\}$$

$$R = \{c, d\}$$

$$U = \{e\}$$

Example

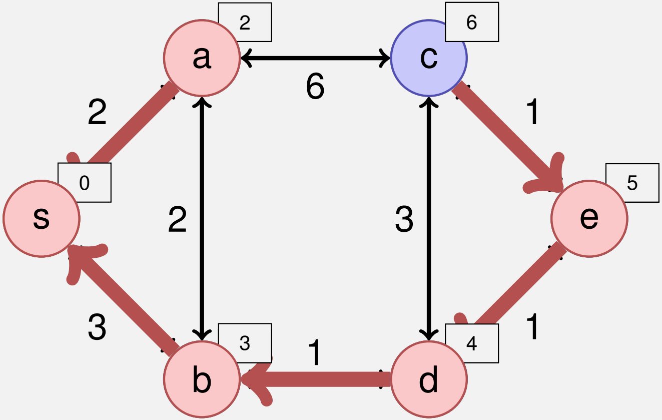


$$M = \{s, a, b, d\}$$

$$R = \{c, e\}$$

$$U = \{\}$$

Example

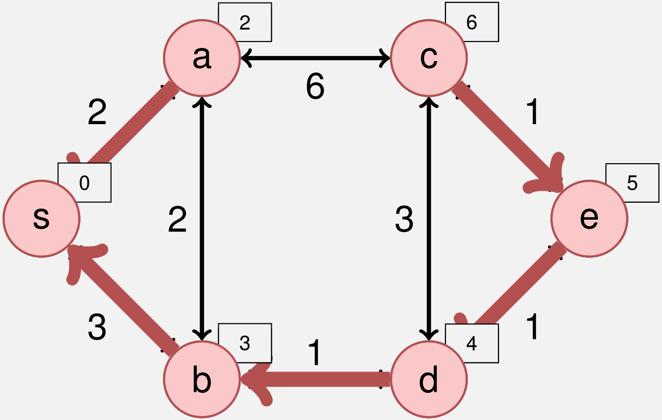


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Example



$$M = \{s, a, b, d, e, c\}$$

$$R = \{\}$$

$$U = \{\}$$

Implementation: Data Structure for R?

Required operations:

- Insert (add to R)
- ExtractMin (over R) and DecreaseKey (Update in R)

```
foreach $v \in N^+(u)$ do
    if $d_s[u] + c(u,v) < d_s[v]$ then
        $d_s[v] \leftarrow d_s[u] + c(u,v)$
        $\pi_s[v] \leftarrow u$
        if $v \in R$ then
            DecreaseKey($R, v$)            // Update of a $d(v)$ in the heap of $R$
        else
            $R \leftarrow R \cup \{v\}$            // Update of $d(v)$ in the heap of $R$
```

MinHeap!

DecreaseKey

- DecreaseKey: climbing in MinHeap in $\mathcal{O}(\log |V|)$
- Position in the heap (i.e. array index of element in the heap)?

DecreaseKey

- DecreaseKey: climbing in MinHeap in $\mathcal{O}(\log |V|)$
- Position in the heap (i.e. array index of element in the heap)?
    - alternative (a): Store position at the nodes

DecreaseKey

- DecreaseKey: climbing in MinHeap in $\mathcal{O}(\log |V|)$
- Position in the heap (i.e. array index of element in the heap)?
  - alternative (a): Store position at the nodes
  - alternative (b): Hashtable of the nodes

DecreaseKey

- DecreaseKey: climbing in MinHeap in $\mathcal{O}(\log |V|)$
- Position in the heap (i.e. array index of element in the heap)?
  - alternative (a): Store position at the nodes
  - alternative (b): Hashtable of the nodes
  - alternative (c): re-insert node each time after update-operation and mark it as visited ("deleted") once extracted (Lazy Deletion)

Runtime Dijkstra

$n := |V|$, $m := |E|$

- $n\times$ ExtractMin: $\mathcal{O}(n \log n)$
- $m\times$ Insert or DecreaseKey: $\mathcal{O}(m \log |V|)$
- $1\times$ Init: $\mathcal{O}(n)$
- Overal: $\mathcal{O}((n+m)\log n)$. for connected graphs: $\mathcal{O}(m \log n)$

Conclusion

$n := |V|, m := |E|$

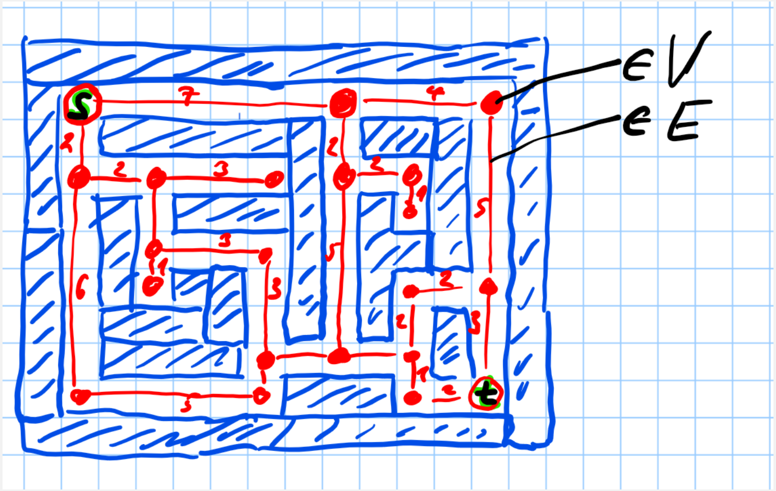
problem	method	runtime	dense $m \in \mathcal{O}(n^2)$	sparse $m \in \mathcal{O}(n)$
$c \equiv 1$	BFS	$\mathcal{O}(m+n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
DAG	Top-Sort	$\mathcal{O}(m+n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
$c \geq 0$	Dijkstra	$\mathcal{O}((m+n)\log n)$	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n \log n)$
general	Bellman-Ford[1]	$\mathcal{O}(m \cdot n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$

---

[1] will be covered later in class (dynamic programming)

3. In-Class-Exercise (practical)

Shortest Path in a Maze



BFS

```
color = {s: Grey}
dist = {s:0}
predecessor = {s: None}
queue = Queue();
queue.put(s);
while not queue.empty():
  u = queue.get()
  for e in u.edges:
    w = e.target
    if w not in color: # color is white
      color[w] = Grey
      predecessor[w] = u
      dist[w] = dist[u] + e.weight
      queue.put(w)
  color[u] = Black
```

Solution Dijkstra

```python
dist = {s:0}
predecessor = {s: None}
R = PriorityQueue()
R.put(PriorityEntry(0,s))
while not R.empty():
  p = R.get()
  u = p.data
  if p.priority == dist[u]: # lazy deletion
    for e in u.edges:
      v = e.target
      w = dist[u] + e.weight
      if v not in dist or w < dist[v]:
        dist[v] = w
        predecessor[v] = u
        R.put(PriorityEntry(w,v))
```

Questions / Suggestions?