Algorithms Theory

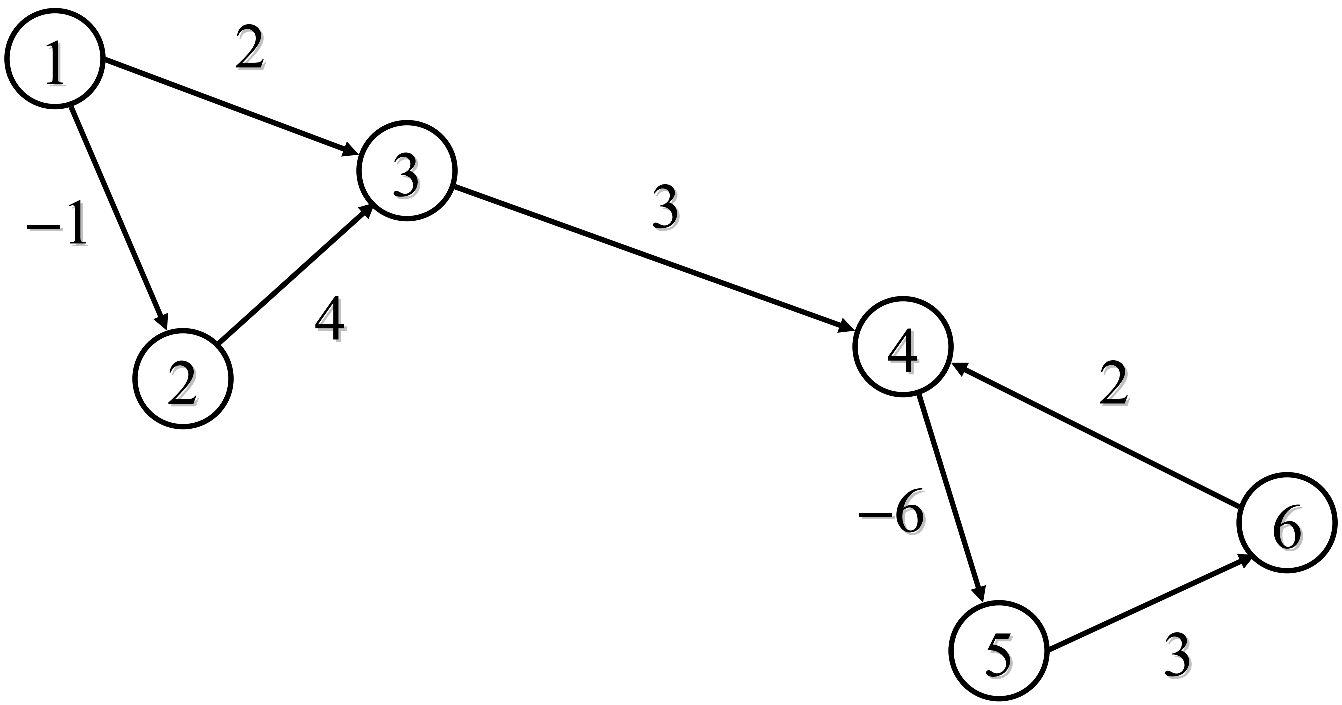
11 – Shortest Paths

Prof. Dr. S. Albers

1. Shortest-paths problem

Directed graph $G = (V, E)$

Cost function $c : E \rightarrow R$



Distance between two vertices

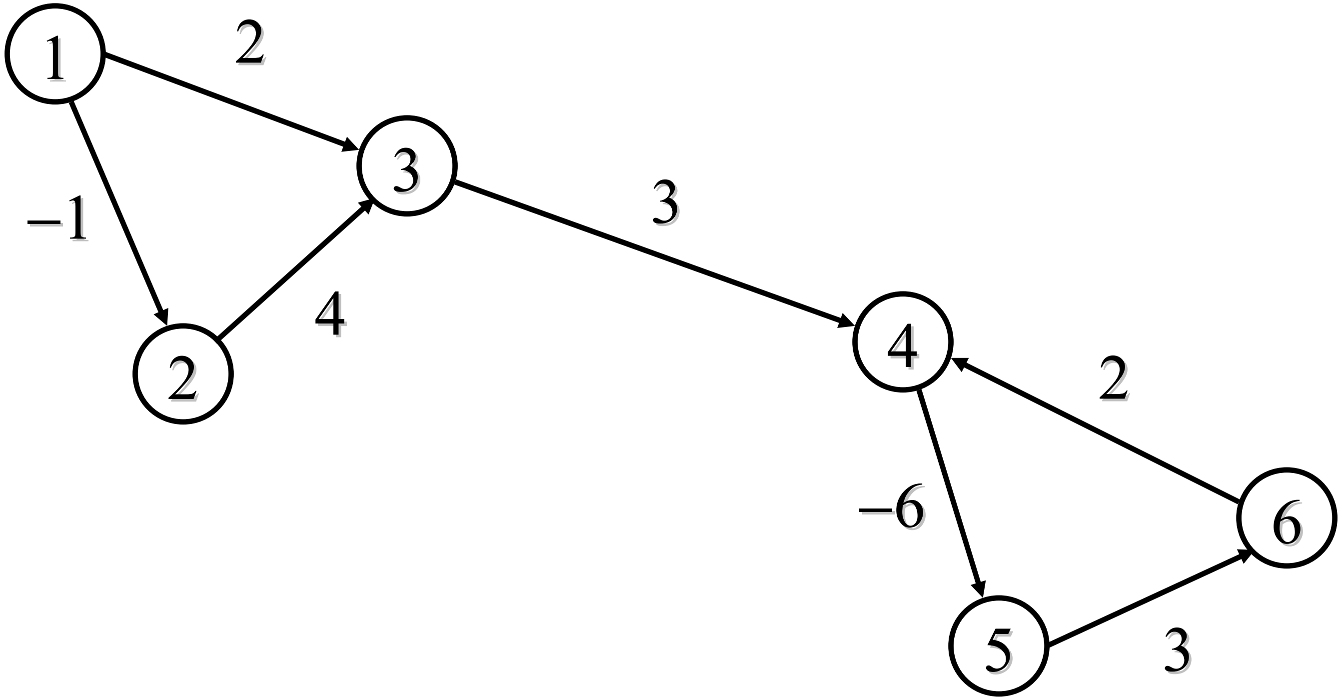
Cost of a path $P = v_0, v_1, \ldots, v_l$ from u to v :

$$c(P) = \sum_{i=0}^{l-1} c(v_i, v_{i+1})$$

Distance between u and v (not always defined):

$dist(v,w) = \inf \{ c(P) \mid P \text{ is a path from } u \text{ to } v \}$

Example



dist(1,2) =

dist(1,3) =

dist(3,1) =

dist(3,4) =

2. Single-source shortest paths problem

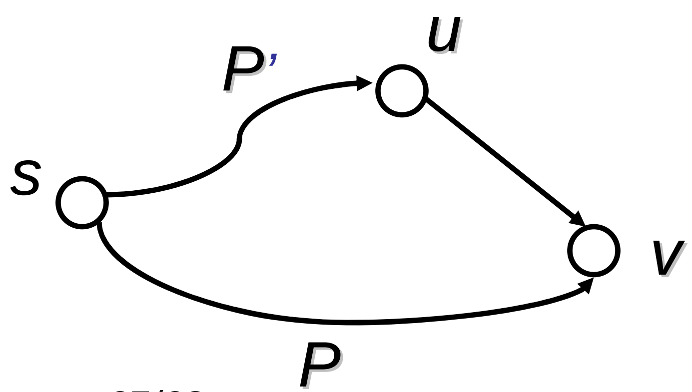
Input: network $G = (V, E, c)$, $c : E \rightarrow R$, vertex s

Output: $dist(s,v)$ for all $v \in V$

Observation: The function *dist* satisfies the triangle inequality.

For any edge $(u,v) \in E$:

$$dist(s,v) \leq dist(s,u) + c(u,v)$$



P = shortest path from s to v

P' = shortest path from s to u

Greedy approach to an algorithm

1. Overestimate the function *dist*

$$dist(s,v) = \begin{cases} 0 & \text{if } \; v = s \\ \infty & \text{if } \; v \neq s \end{cases}$$

2. While there exists an edge $e = (u,v)$ with

$$dist(s,v) > dist(s,u) + c(u,v)$$

set $dist(s,v) \leftarrow dist(s,u) + c(u,v)$

Generic algorithm

1. DIST[s] $\leftarrow 0$;
2. **for all** $v \in V \setminus \{s\}$ **do** DIST[v] $\leftarrow \infty$ **endfor;**
3. **while** $\exists$ $e = (u,v) \in E$ with DIST[v] > DIST[u] + $c(u,v)$ **do**
4. Choose such an edge $e = (u,v)$;
5. DIST[v] $\leftarrow$ DIST[u] + $c(u,v)$;
6. **endwhile;**

Questions:

1. How can we check in line 3 if the triangle inequality is violated?
2. Which edge shall we choose in line 4?

Solution

Maintain a set U of all those vertices that might have an outgoing edge violating the triangle inequality.

- Initialize $U = \{s\}$
- Add vertex v to U whenever DIST[v] decreases.

1. Check if the triangle inequality is violated: $U \neq \emptyset$?
2. Choose a vertex from U and restore the triangle inequality for all outgoing edges (relaxation).

Refined algorithm

```
DIST[s] ← 0;
for all v ∈ V \ {s} do DIST[v] ← ∞ endfor;
U ← {s};
while U ≠ ∅ do
    Choose a vertex u ∈ U and delete it from U;
    for all e = (u,v) ∈ E do
        if DIST[v] > DIST[u] + c(u,v) then
            DIST[v] ← DIST[u] + c(u,v);
            U ← U ∪ {v};
        endif;
    endfor;
endwhile;
```

Invariant for the DIST values

Lemma 1: For each vertex $v \in V$ we have DIST[v] $\geq$ *dist*(s,v).

Proof: (by contradiction)

Let v be the first vertex for which the relaxation of an edge (u,v) yields DIST[v] $<$ *dist*(s,v).

Then:

$$\text{DIST}[u] + c(u,v) = \text{DIST}[v] < \mathit{dist}(s,v) \leq \mathit{dist}(s,u) + c(u,v)$$

Important properties

Lemma 2:

a) If $v \notin U$, then for all $(v,w) \in E$: DIST[w] $\leq$ DIST[v] + $c(v,w)$

b) Let $s = v_0, v_1, ..., v_l = v$ be a shortest path from s to v.
If DIST[v] > $dist(s,v)$, then there exists v_i, $0 \leq i \leq l-1$, with $v_i \in U$ and DIST[v_i] = $dist(s,v_i)$.

c) If G has no negative-cost cycles and DIST[v] > $dist(s,v)$ for any $v \in V$, then there exists a $u \in U$ with DIST[u] = $dist(s,u)$.

d) If in line 5 we always choose $u \in U$ with DIST[u] = $dist(s,u)$, then the while-loop is executed only once per vertex.

Efficient implementations

Line 5: How can we find a vertex $u \in U$ with DIST[u] = $dist(s,u)$?

This is not known in general, but for some important special cases.

- Nonnegative networks (only non-negative edge costs)

  Dijkstra's algorithm
- Networks without negative-cost cycles

  Bellman-Ford algorithm
- Acyclic networks

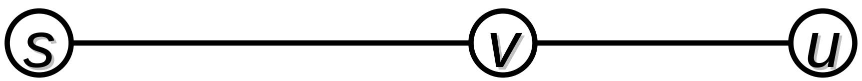
3. Non-negative networks

5'. Choose a vertex $u \in U$ with minimum DIST[u] and delete it from U.

Lemma 3: Using 5' we have DIST[u] = $dist(s,u)$.

Proof: By Lemma 2b) there is a vertex $v \in U$ on the shortest path from s to u with DIST[v] = $dist(s,v)$.

$$\text{DIST}[u] \le \text{DIST}[v] = dist(s,v) \le dist(s,u)$$



Implementing *U* as priority queue

The elements of the form (*key*, *inf*) are the pairs (DIST[*v*], *v*).

Empty(*Q*): Is *Q* empty?

Insert(*Q*, *key*, *inf*): Inserts (*key*,*inf*) into *Q*.

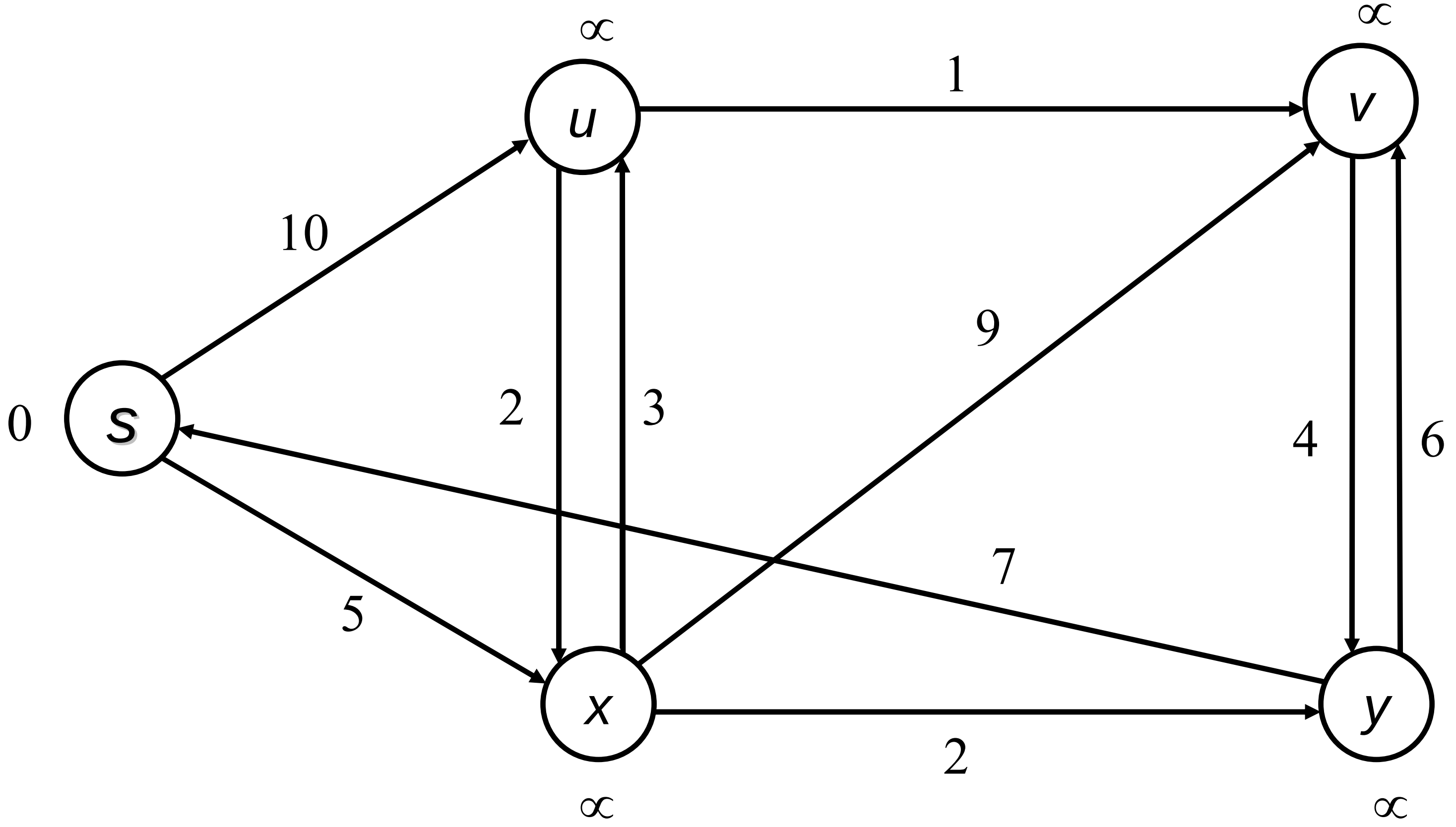
DeleteMin(*Q*): Returns the element with minimum key and deletes it from *Q*.

DecreaseKey(*Q*, *element*, *j*): Decreases the value of *element*‘s key to the new value *j*, provided that *j* is less than the former key.

Dijkstra‘s algorithm

```
DIST[s] ← 0;   Insert(U,0,s);
for all  v ∈ V \ {s}  do DIST[v] ← ∞;  Insert(U, ∞, v);  endfor;
while  ¬Empty(U) do
    (d,u) ← DeleteMin(U);
    for all e = (u,v) ∈ E do
        if DIST[v] > DIST[u] + c(u,v) then
            DIST[v] ← DIST[u] + c(u,v);
            DecreaseKey(U, v, DIST[v]);
        endif;
    endfor;
endwhile;
```

Example



Running time

$O(\ n\ (T_{\text{Insert}} + T_{\text{Empty}} + T_{\text{DeleteMin}}) + m\ T_{\text{DecreaseKey}} + m + n\ )$

Fibonacci heaps:

T_{Insert} :	O(1)
$T_{\text{DeleteMin}}$:	O(log n) amortized
$T_{\text{DecreaseKey}}$:	O(1) amortized

$O(\ n \log n + m\ )$

4. Networks without negative-cost cycles

Implement U as a queue.

Lemma 4: Each vertex v is inserted into U at most n times.

Proof: Suppose that DIST[v] > $dist(s,v)$ and v is appended at U for the i-th time. Then, by Lemma 2c) there exists $u_i \in U$ with DIST[u_i] = $dist(s,u_i)$.

Vertex u_i is deleted from U before v and will never be appended at U again.

Vertices u_1, u_2, u_3, ... are pairwise distinct.

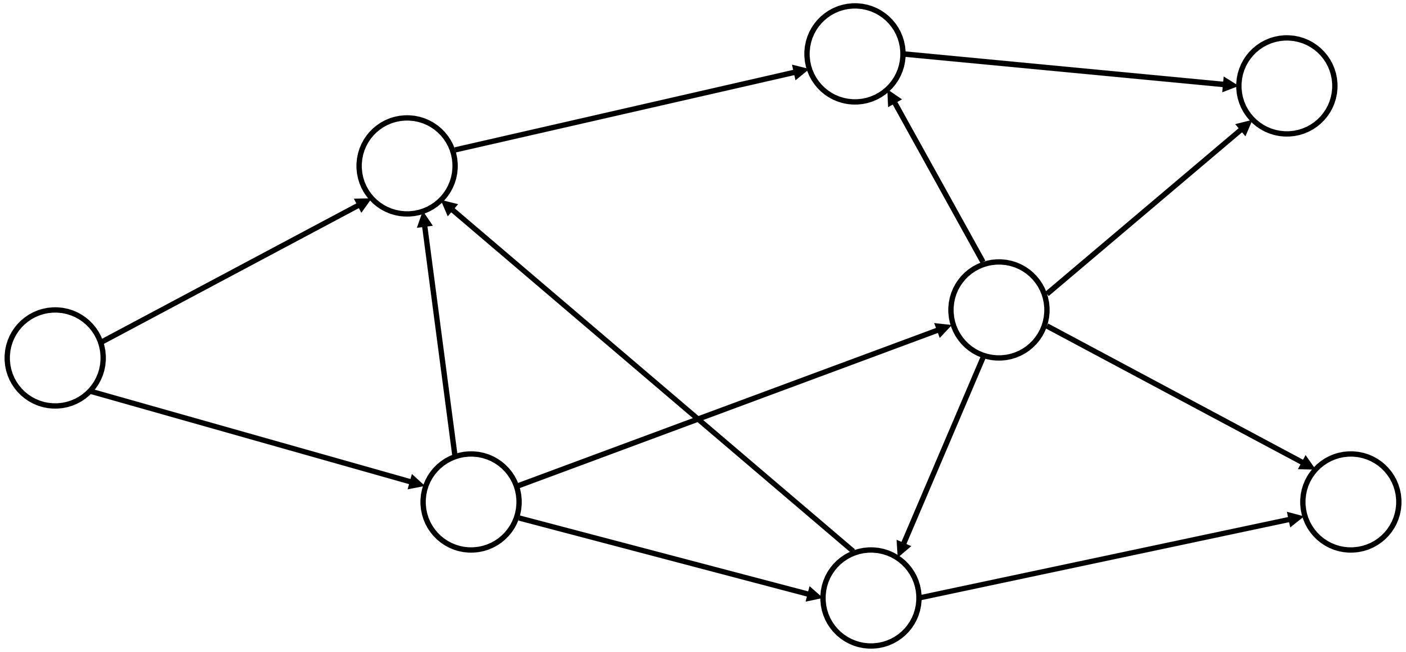
Bellman-Ford algorithm

```
DIST[s] ← 0;   A[s] ← 0;
for all  v ∈ V \ {s}  do DIST[v] ← ∝ ; A[v] ← 0; endfor;
U ← {s};
while  U ≠ ∅  do
     Choose the first vertex u in U and delete it from U;  A[u] ← A[u]+1;
     if  A[u] > n  then  return „negative-cost cycle“;
     for all e = (u,v) ∈ E do
         if DIST[v] > DIST[u] + c(u,v) then
                DIST[v] ← DIST[u] + c(u,v);
                U ← U ∪ {v};
         endif;
     endfor;
endwhile;
```

5. Acyclic networks

Topological sorting: num: $V \rightarrow \{1, \ldots, n\}$

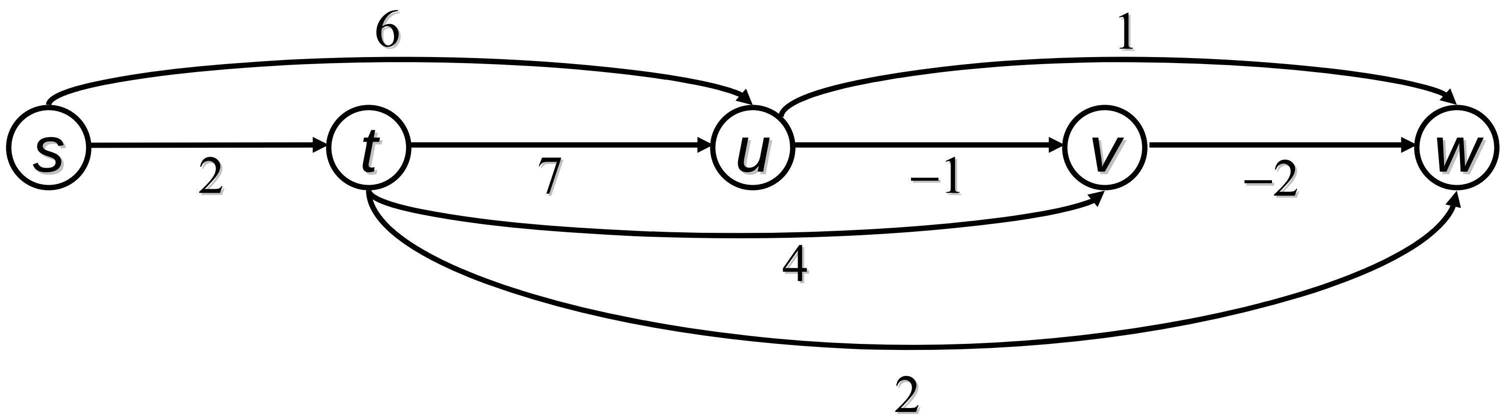
such that for all $(u,v) \in E$: $\text{num}(u) < \text{num}(v)$



Algorithm for acyclic graphs

1. Sort $G = (V, E, c)$ topologically;
2. DIST[s] $\leftarrow 0$;
3. **for all** $v \in V \setminus \{s\}$ **do** DIST[v] $\leftarrow \propto$; **endfor;**
4. $U \leftarrow \{ v \mid v \in V$ with num(v) $< n\}$;
5. **while** $U \neq \emptyset$ **do**
6.   Choose the vertex $u \in U$ with minimum num;
7.   **for all** $e = (u,v) \in E$ **do**
8.   **if** DIST[v] > DIST[u] + $c(u,v)$ **then**
9.   DIST[v] $\leftarrow$ DIST[u] + $c(u,v)$;
10.   **endif;**
11.   **endfor;**
12. **endwhile;**

Example



Correctness

Lemma 5: When the *i*-th vertex u_i is deleted from *U*, then

$$\text{DIST}[u_i] = dist(s,u_i).$$

Proof: Induction on *i*.

$i = 1$: ok

$i > 1$: Let $s = v_1, v_2, \ldots, v_l, v_{l+1} = u_i$ be a shortest path from *s* to u_i.

v_l is deleted from *U* before u_i.

Then, by induction hypothesis: $\text{DIST}[v_l] = dist(s,v_l)$.

After (v_l, u_i) has been relaxed:

$$\text{DIST}[u_i] \leq \text{DIST}[v_l] + c(v_l, u_i) = dist(s,v_l) + c(v_l, u_i) = dist(s,u_i)$$