ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithm Theory

14 Shortest Paths

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

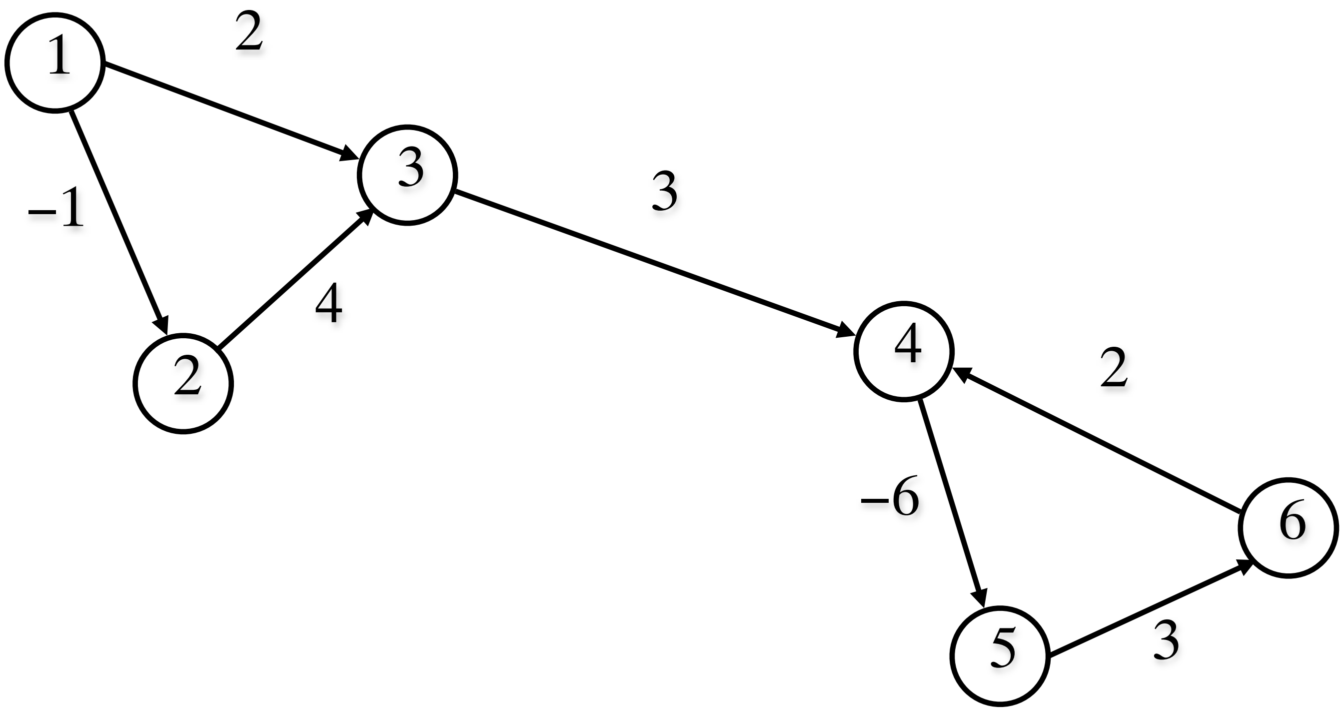
CoNe Freiburg

IIF
INSTITUT FÜR INFORMATIK FREIBURG

Shortest Path Problem

Directed graph $G = (V, E)$

Cost function $c : E \rightarrow R$



Distance between two vertices

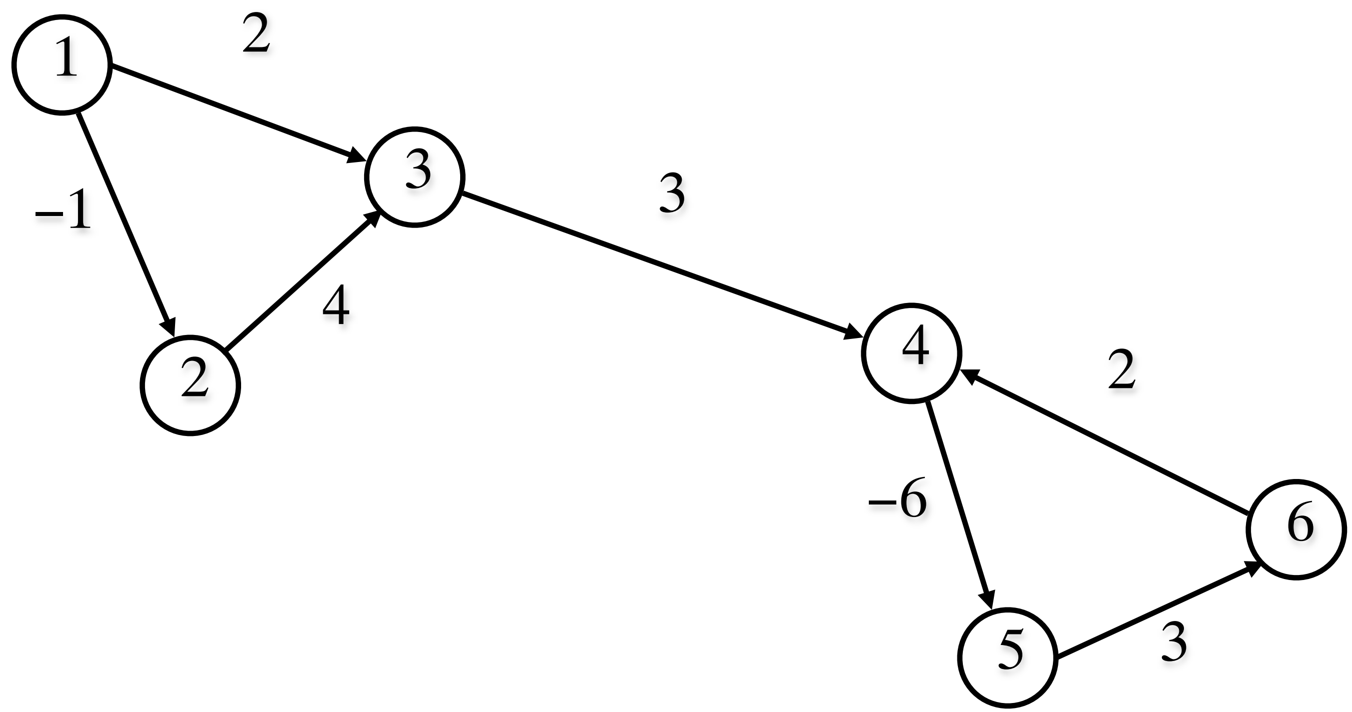
Cost of a path $P = v_0, v_1, \ldots, v_l$ from v to w

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

Distance from v to w (not always defined)

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

Example



*dist(*1,2) =

*dist(*3,1) =

*dist(*1,3) =

*dist(*3,4) =

Single Source Shortest Paths Problems

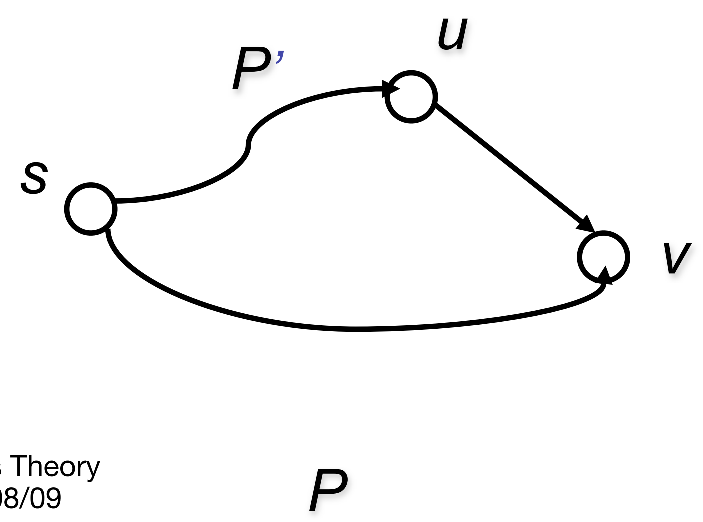
Input: network $G = (V, E, c)$ $c : E \rightarrow R$ Node s

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

Observation: *dist* fulfills the triangle inequality

For all edges $(u,v) \in E$ and for all $s \in V$:

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



P = shortest path from s to v

P' = shortest path from s to u

Greedy Algorithm

1. Overestimate *dist*-function

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

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] $\leftarrow \propto$ **endfor;**

3. **while** $\exists\ e = (u,v) \in E$ with DIST[v] $>$ DIST[u] + $c(u,v)$ **do**

4. Choose an edge $e = (u,v)$;

5. DIST[v] $\leftarrow$ DIST[u] + $c(u,v)$;

6. **endwhile;**

Questions:

1. How to check in line 3 whether the triangle inequality holds.
2. Which edge needs to be choosen in line 4?

Solution

Maintain a set U of all 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 $\text{DIST}[v] \geq \mathit{dist}(s,v)$.

Proof: (by contradiction)

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

Then:

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

which implies

$$\text{DIST}[u] < \text{dist}(s,u)$$

contradicts the assumption that at v the first violation occurred.

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 $s=v_0, v_1, \ldots, v_l=v$ be the shortest path 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 $u \in U$ and 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.

Important Properties

- **Proof of 2a:**
  - If $v \notin U$, then for all $(v,w) \in E$: $DIST[w] \leq DIST[v] + c(v,w)$
- **Induction on the number i of executions of while-loops**
  - $i = 0$: nodes $v \neq s$ are not in U
    - $DIST[w] \leq DIST[v] + c(v,w)$ is true since $DIST[v] = \infty$
  - $i > 0$: Assume 2a is true before i-th execution of the while loop
    - To show: it is true after the i-th execution of the while loop
  - Let $v \notin U$ after the execution i-th execution of the while loop
  - 1. case $v \notin U$ before the i-th execution of the while loop
    - Dist[v] does not change.
    - Dist[w] may be decreased.
  - 2. case $v \in U$ before the i-th execution of the while loop
    - follows by algorithm since v was chosen and hence $DIST[w]= DIST[v]+c(v,w)$

Important Properties

- **Proof of Lemma 2b:**
  - Let $s=v_0, v_1, ..., v_l=v$ be the shortest path 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] = $dist(s,v_i)$.
- **Let i be the maximum index with**
  - DIST[v_i] = $dist(s,v_i)$
  - i exists because DIST[s]=dist(s,s)=0
- **Assume** $v_i \notin U$
  - By Lemma 2a):
    $$\begin{aligned} \text{DIST}[v_{i+1}] &\leq \text{DIST}[v_i] + c(v_i,v_{i+1}) \\ &= dist(s,v_i) + c(v_i,v_{i+1}) \\ &= dist(s,v_{i+1}) \end{aligned}$$
  - This implies DIST[v_{i+1}] = $dist(s,v_{i+1})$
  - which contradicts that i is maximal.

Important Properties

- **Proof of Lemma 2c:**
  - If G has no negative cost cycle and DIST[v] > dist(s,v) for any $v \in V$, then there exists a $u \in U$ and DIST[u] = dist(s,u).
- **There is a finite shortest path**
  - if there is no negative cost cycle
- **From 2b it follows that U is non-empty**
  - Then there is $v_i \in U \Rightarrow$ DIST(v_i) = dist(s,v_i)
- **Set v_i = u then 2c follows**

Important Properties

- **Proof of Lemma 2d:**
  - 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.
- **A node u can only be added again to U**
  - if DIST[u] decreases again
  - But then DIST[u] < dist(s,v)
  - this contradicts Lemma 1

Efficient Implementations

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

Important special cases.

- Non negative networks (only non-negative edge costs)

  Dijkstra´s algorithm

- Networks without negative cost cycles

  Bellman-Ford algorithm

- Acyclic networks

Non Negative Networks

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

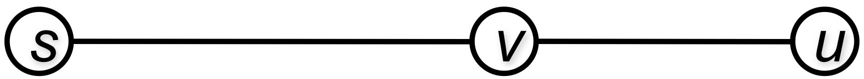
Lemma 3: Using 5' we have DIST[u]=$dist(s,u)$.

Proof: Assume DIST[u] > dist(s,u)

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

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

Then, DIST[u] = 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*.

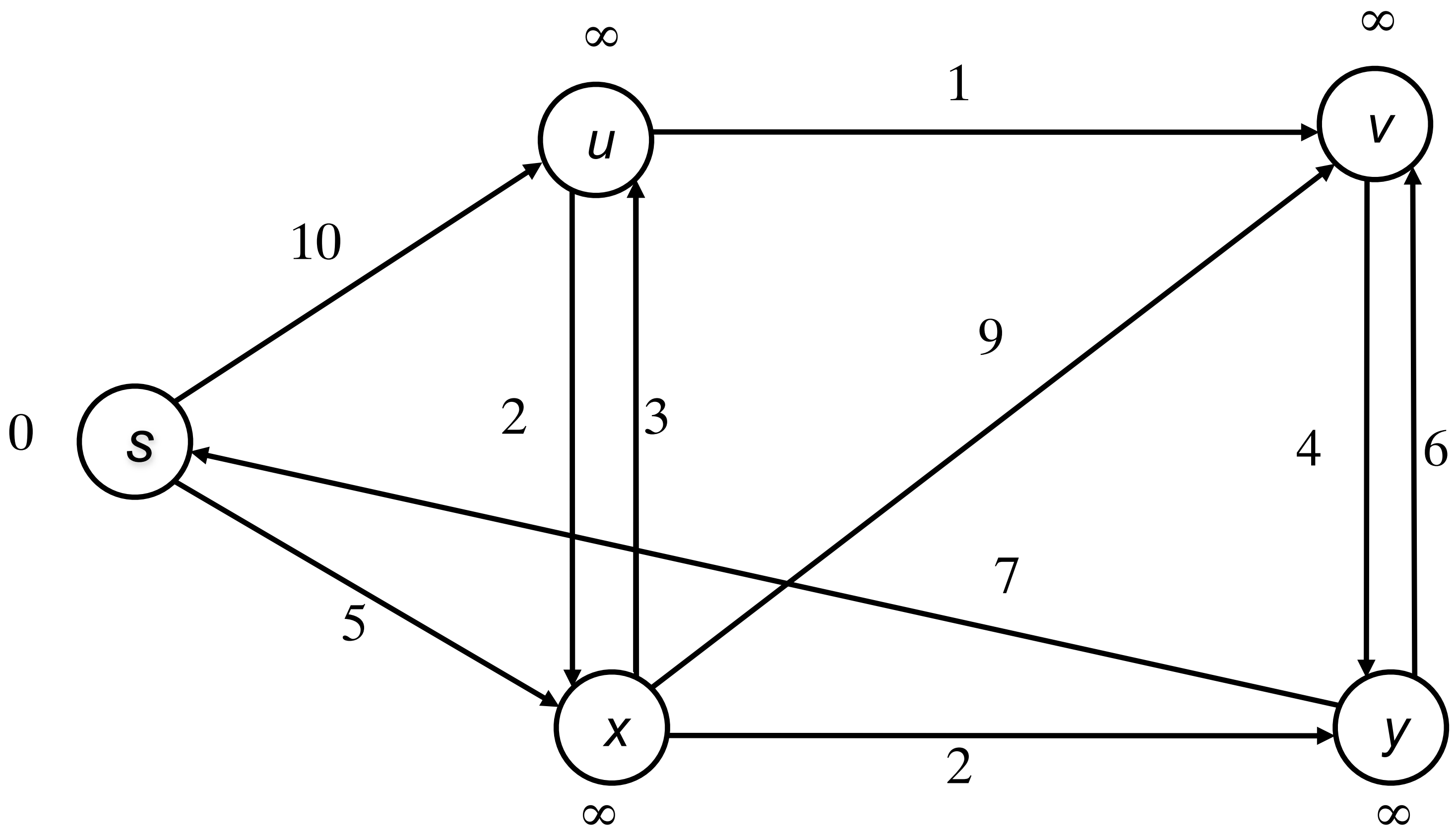
DeleteMin(*Q*): Returns the element with minimum key and deletes it from *Q*.

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.

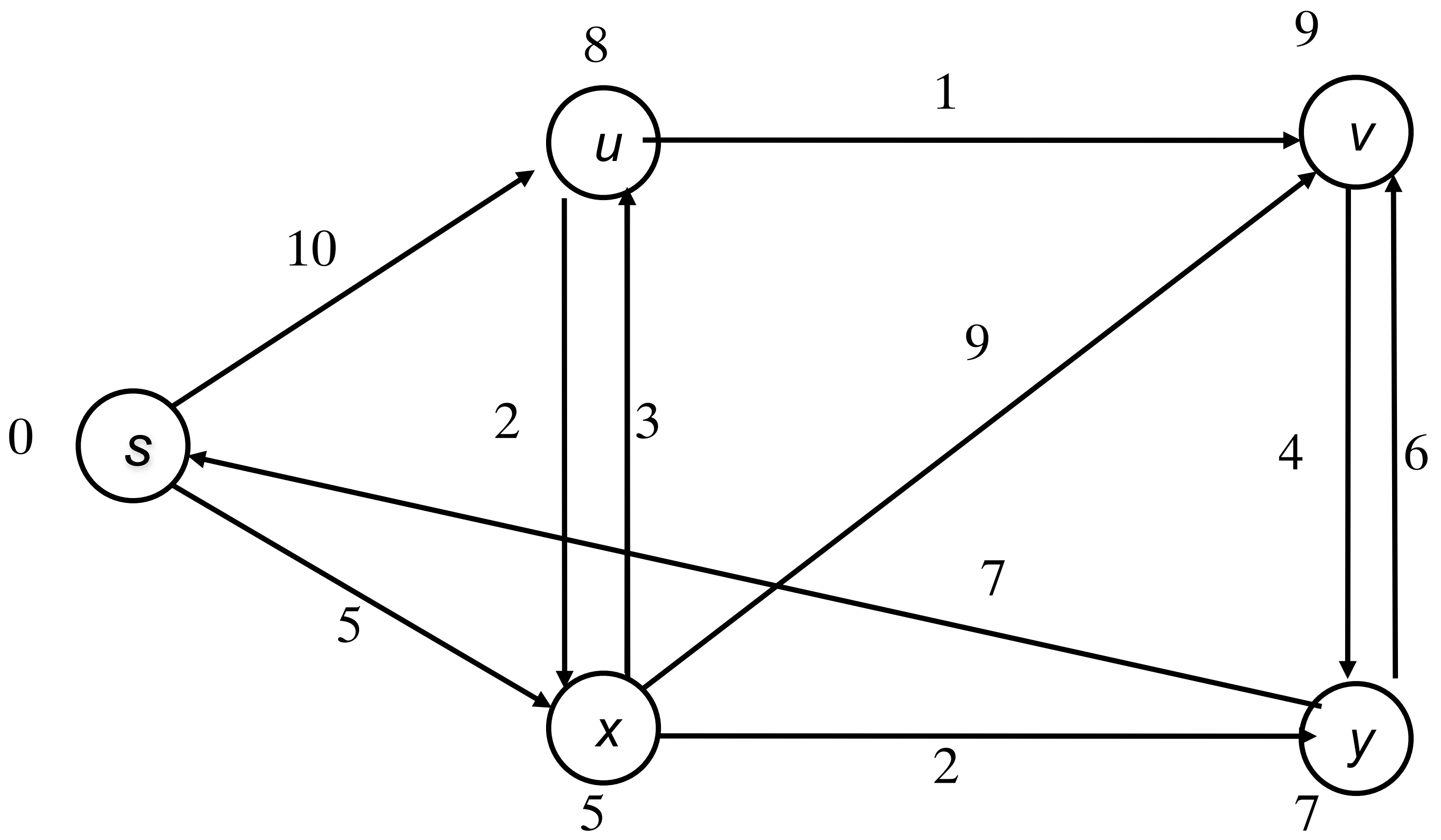
Dijkstra´s Algorithm

```
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



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**

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

Networks without Negative Cost Cycles

Organize 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.

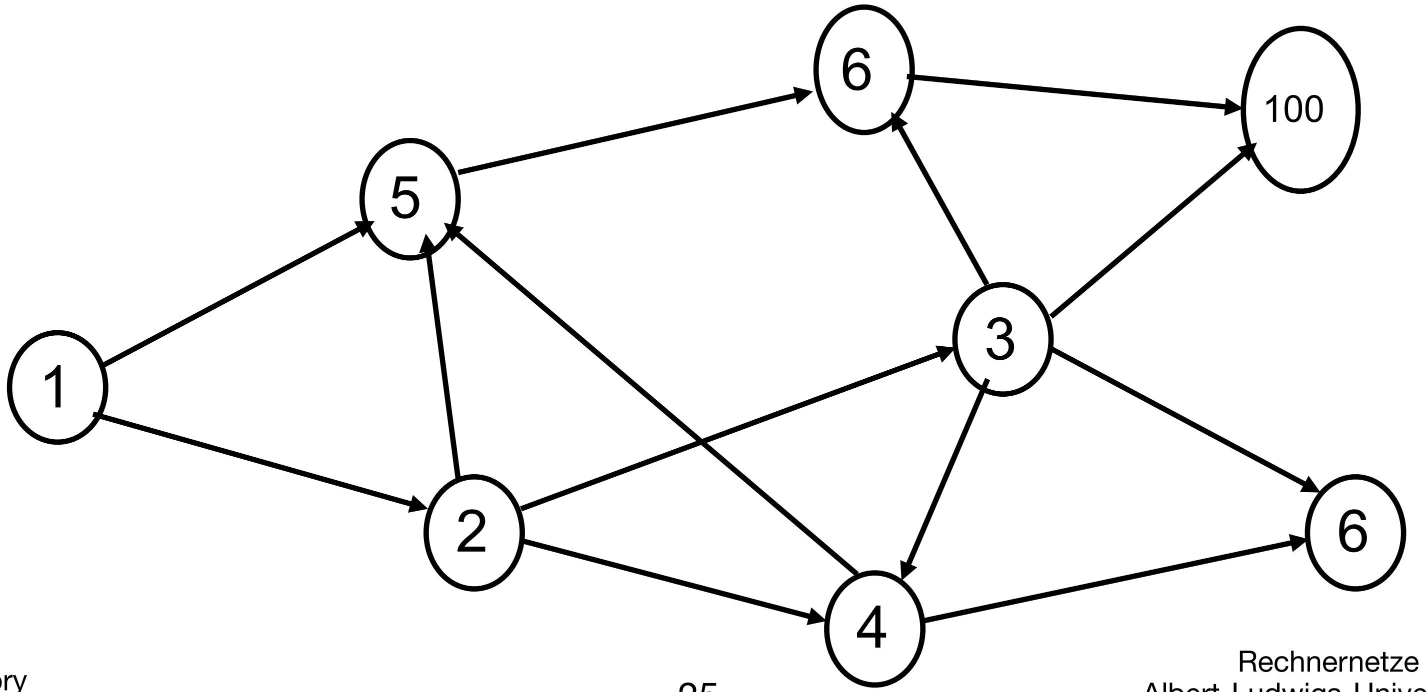
Bellman-Ford-Algorithmus

```
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;
```

Acyclic Networks

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

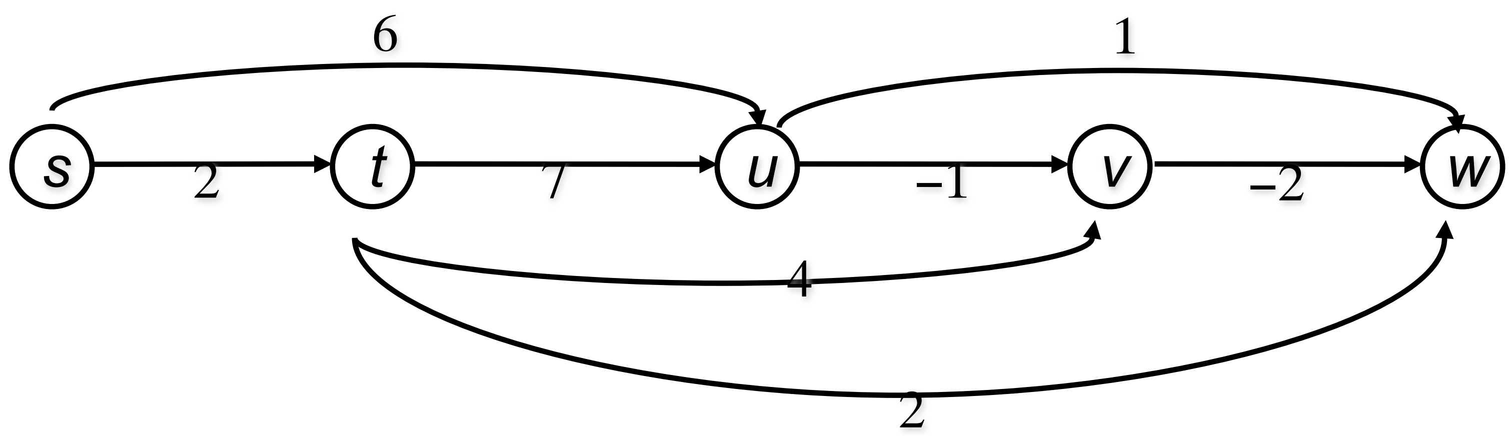
such that for all $(u,v) \in E$: $\text{num}(u) < \text{num}(v)$



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 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 over 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:

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

ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithm Theory

14 Shortest Paths

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

CoNe Freiburg

IIF
INSTITUT FÜR INFORMATIK FREIBURG