CH08-320201 Algorithms and Data Structures

Lecture 20 — 8 May 2018

Prof. Dr. Michael Sedlmair

Jacobs University
Spring 2018

Depth-first Search (DFS)

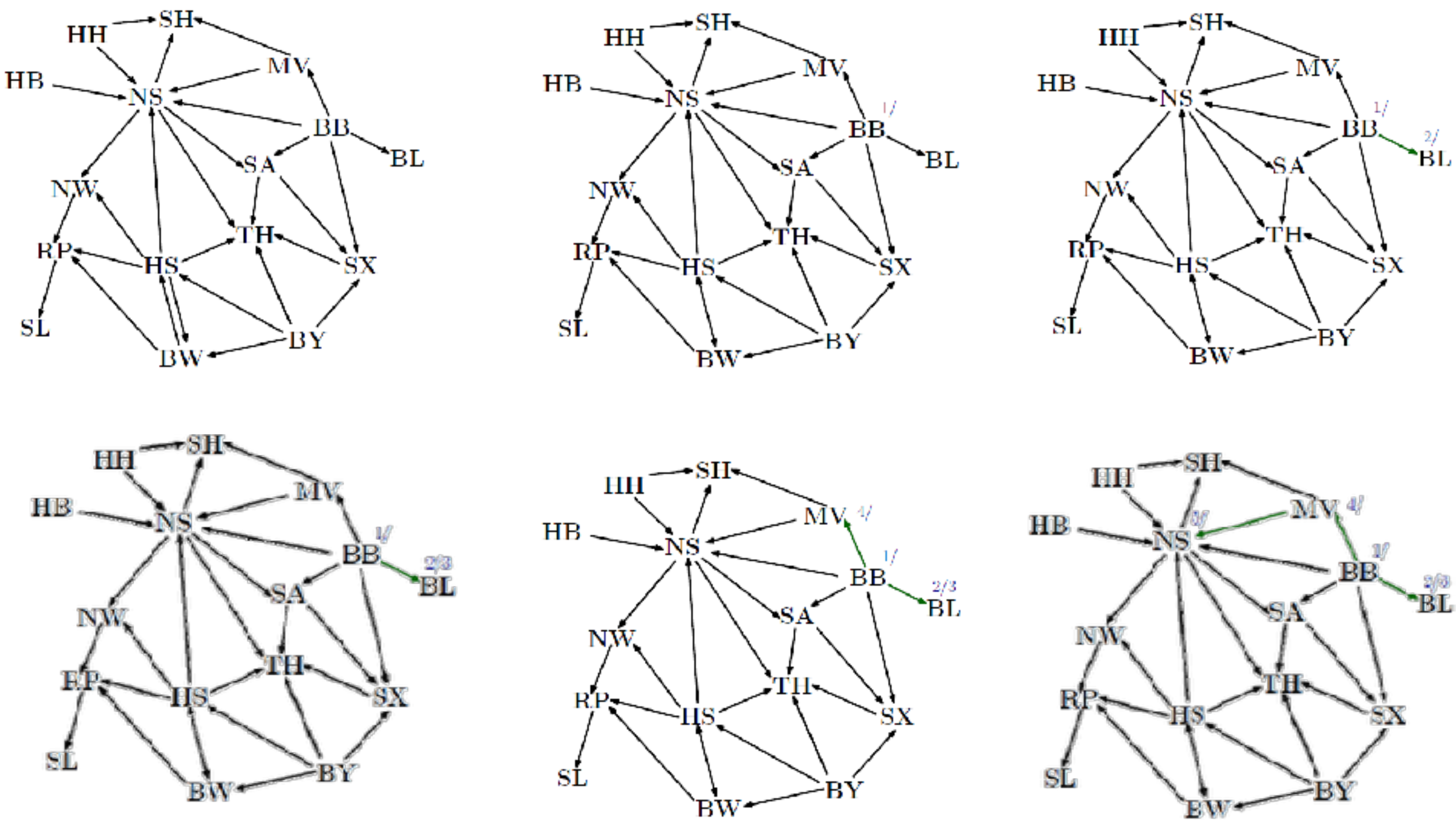
DFS Strategy: First follow one path all the way to its end, before we step back to follow the next path.

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

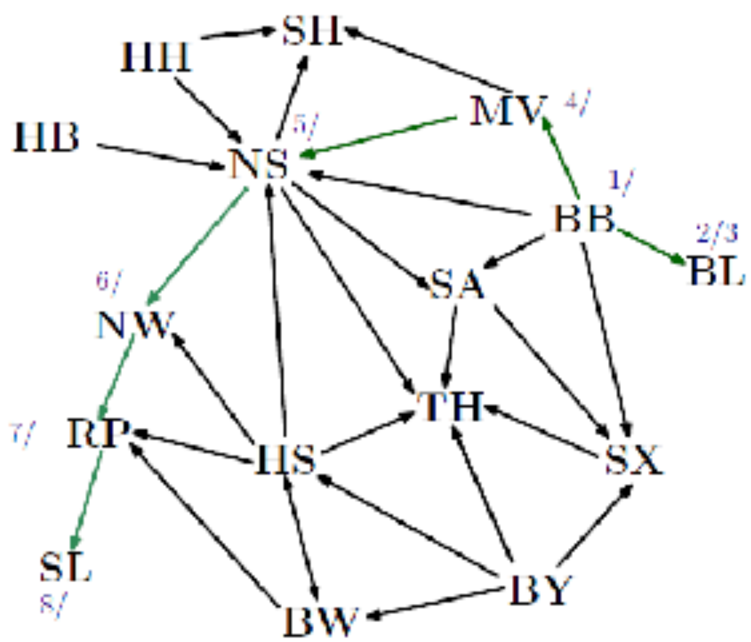
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

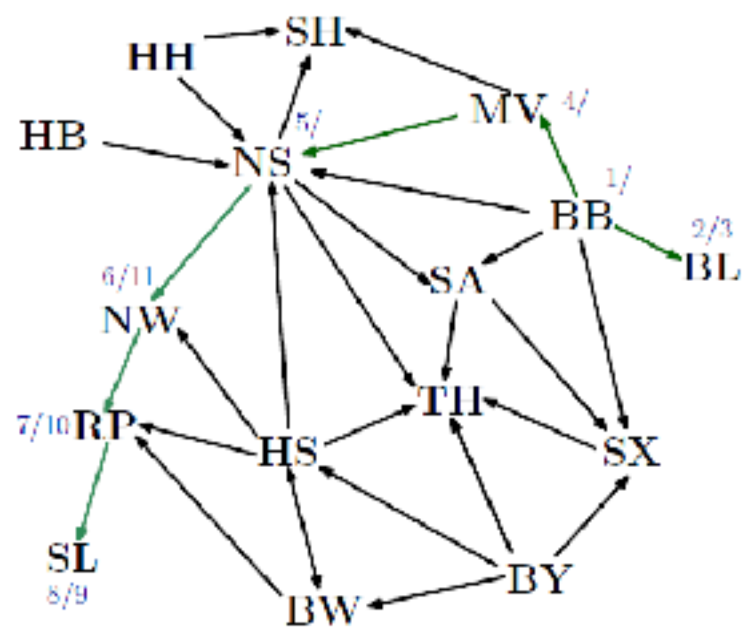
(u.d and u.f are start/finish time for vertex processing)

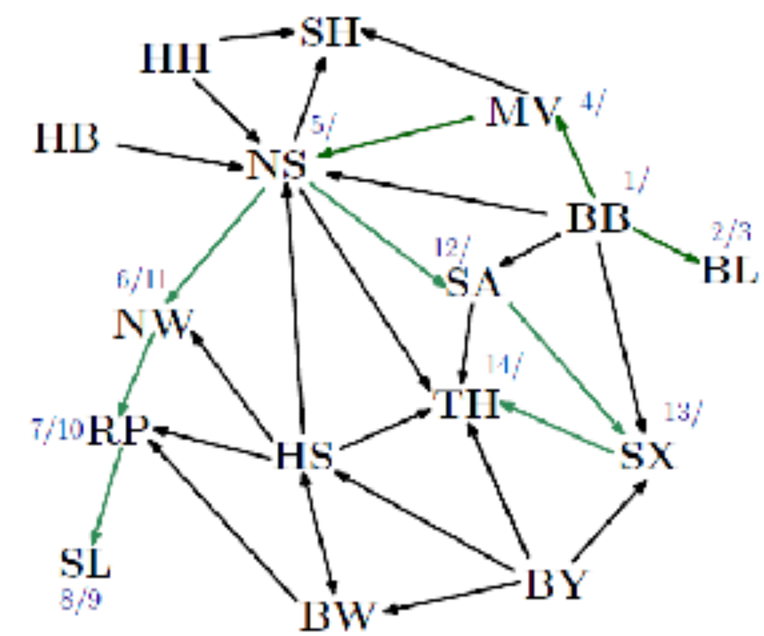
DFS example

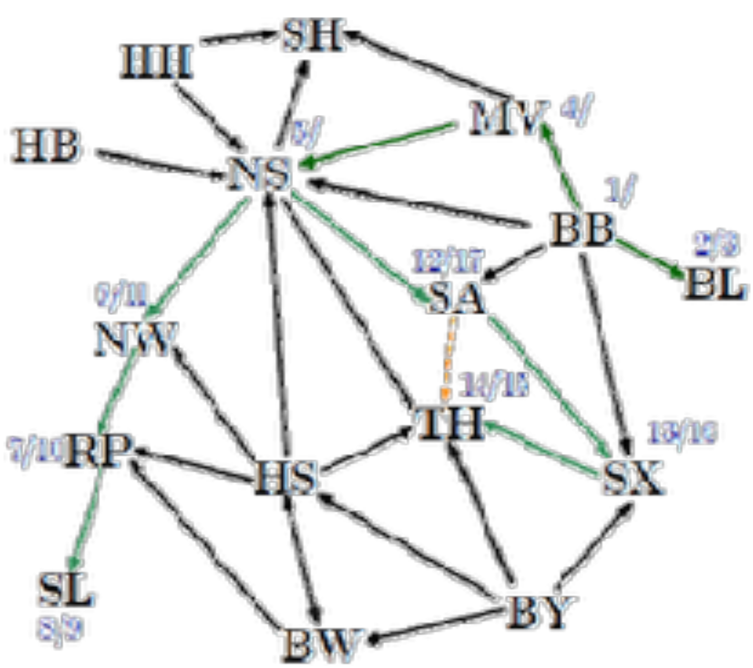


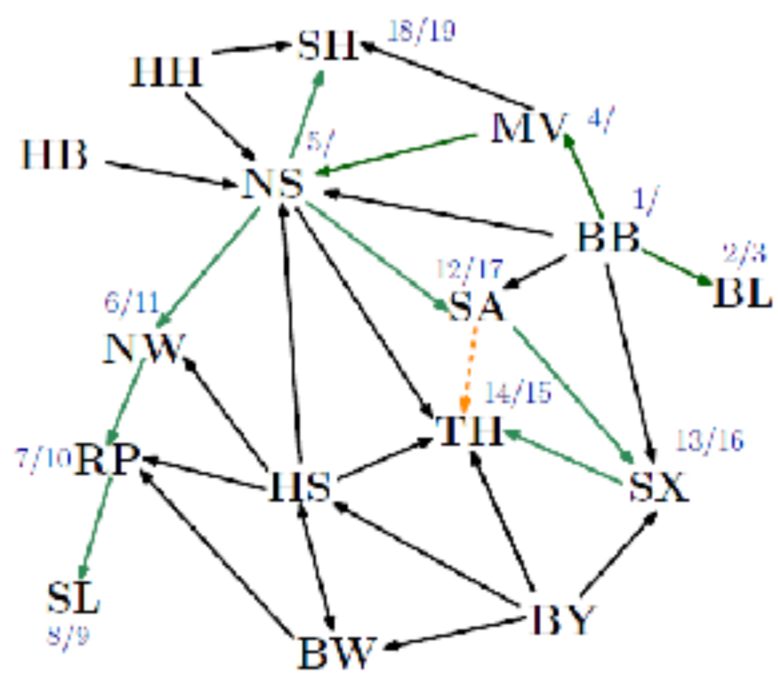
DFS example

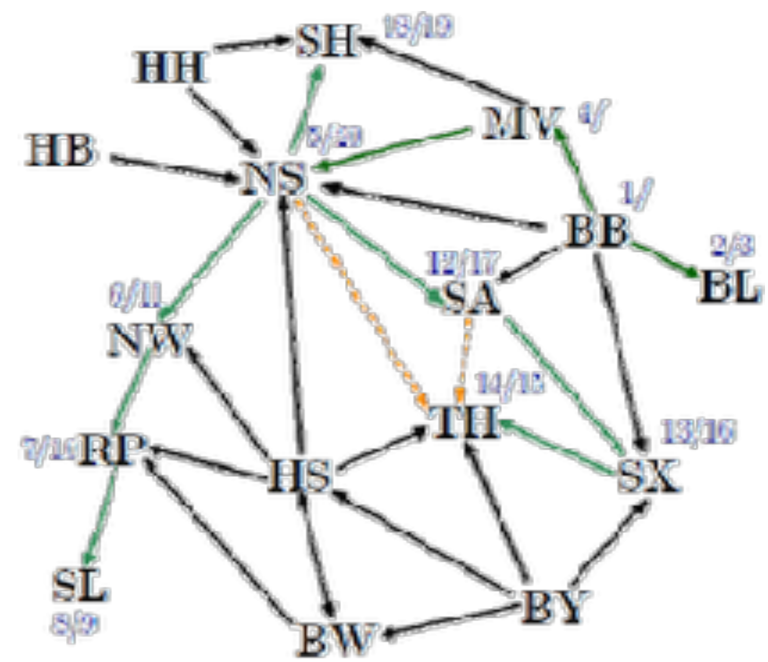

HH
SH
MV 4/
HB
NS 5/
BB 1/
BL 2/3
SA
6/ NW
7/ RP
TH
HS
SX
SL 8/
BW
BY


HH
SH
MV 4/
HB
NS 5/
BB 1/
BL 2/3
SA
6/11 NW
7/10 RP
TH
HS
SX
SL 8/9
BW
BY

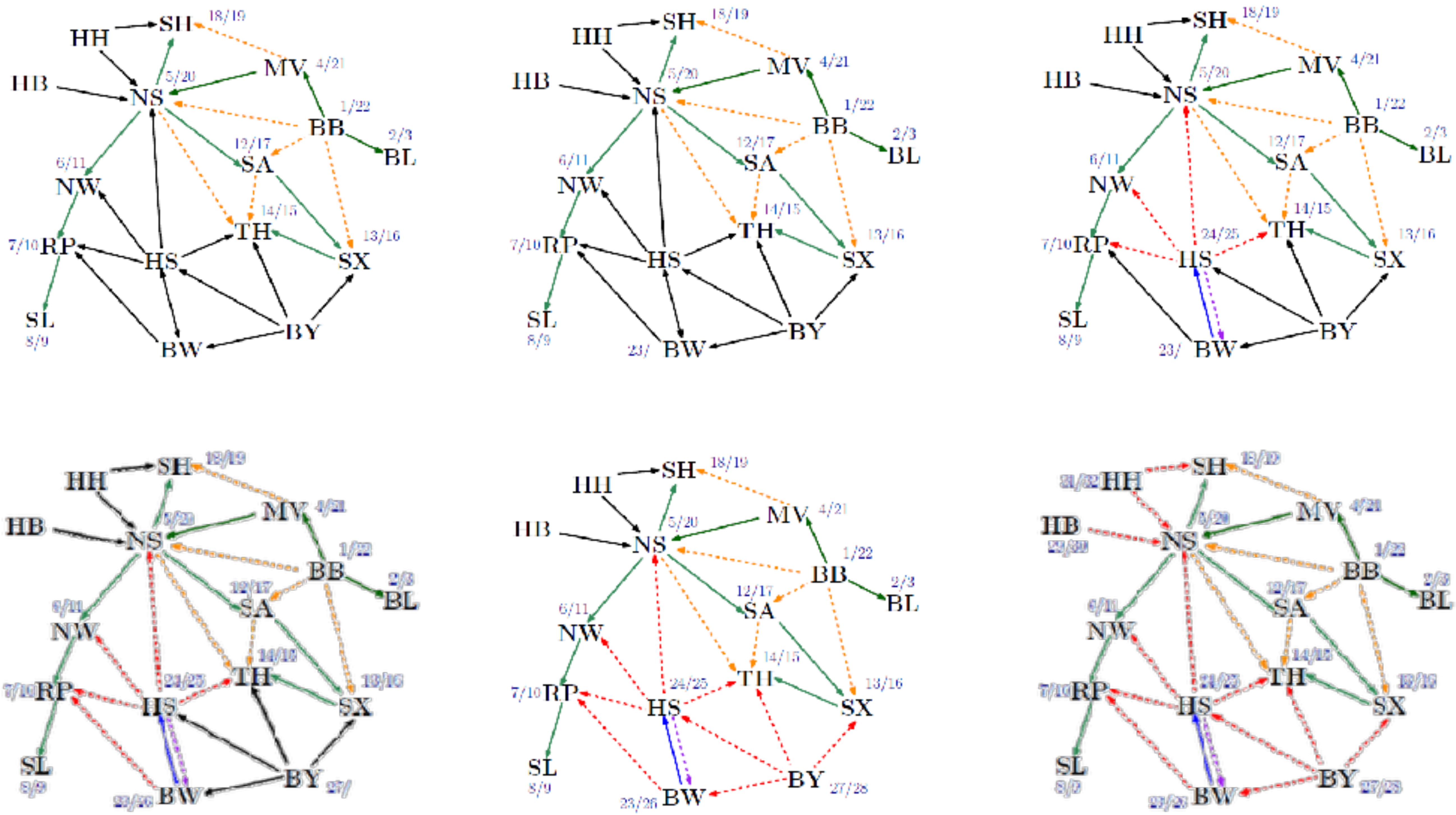

HH
SH
MV 4/
HB
NS 5/
BB 1/
BL 2/3
12/ SA
6/11 NW
7/10 RP
14/ TH
HS
SX 13/
SL 8/9
BW
BY


HH
SH
MV 4/
HB
NS 5/
BB 1/
BL 2/3
12/17 SA
6/11 NW
7/10 RP
14/15 TH
HS
SX 13/16
SL 8/9
BW
BY


HH
SH 18/19
MV 4/
HB
NS 5/
BB 1/
BL 2/3
12/17 SA
6/11 NW
7/10 RP
14/15 TH
HS
SX 13/16
SL 8/9
BW
BY


HH
SH 18/19
MV 4/
HB
NS 5/20
BB 1/
BL 2/3
12/17 SA
6/11 NW
7/10 RP
14/15 TH
HS
SX 13/16
SL 8/9
BW
BY

DFS example



HH
SH 18/19
MV 4/21
HB
NS 5/20
BB 1/22
BL 2/3
SA 12/17
NW 6/11
TH 14/15
RP 7/10
HS
SX 13/16
SL 8/9
BW
BY

DFS analysis

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

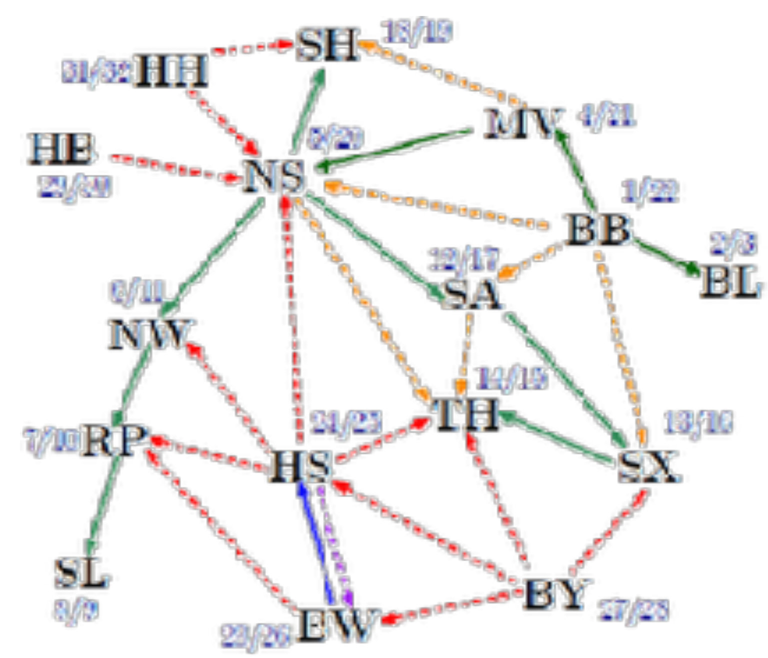
```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

Each vertex and each edge is processed once. Hence, time complexity is $\Theta(|V|+|E|)$.

Edge types

- Different edge types for (u,v):
  - Tree edges (solid): v is white.
  - Backward edges (purple): v is gray.
  - Forward edges (orange): v is black and u.d < v.d
  - Cross edges (red): v is black and u.d > v.d.

- The tree edges form a forest.
- This is called the depth-first forest.
- In an undirected graph, we have no forward and cross edges.



5.3 Minimum Spanning Tree

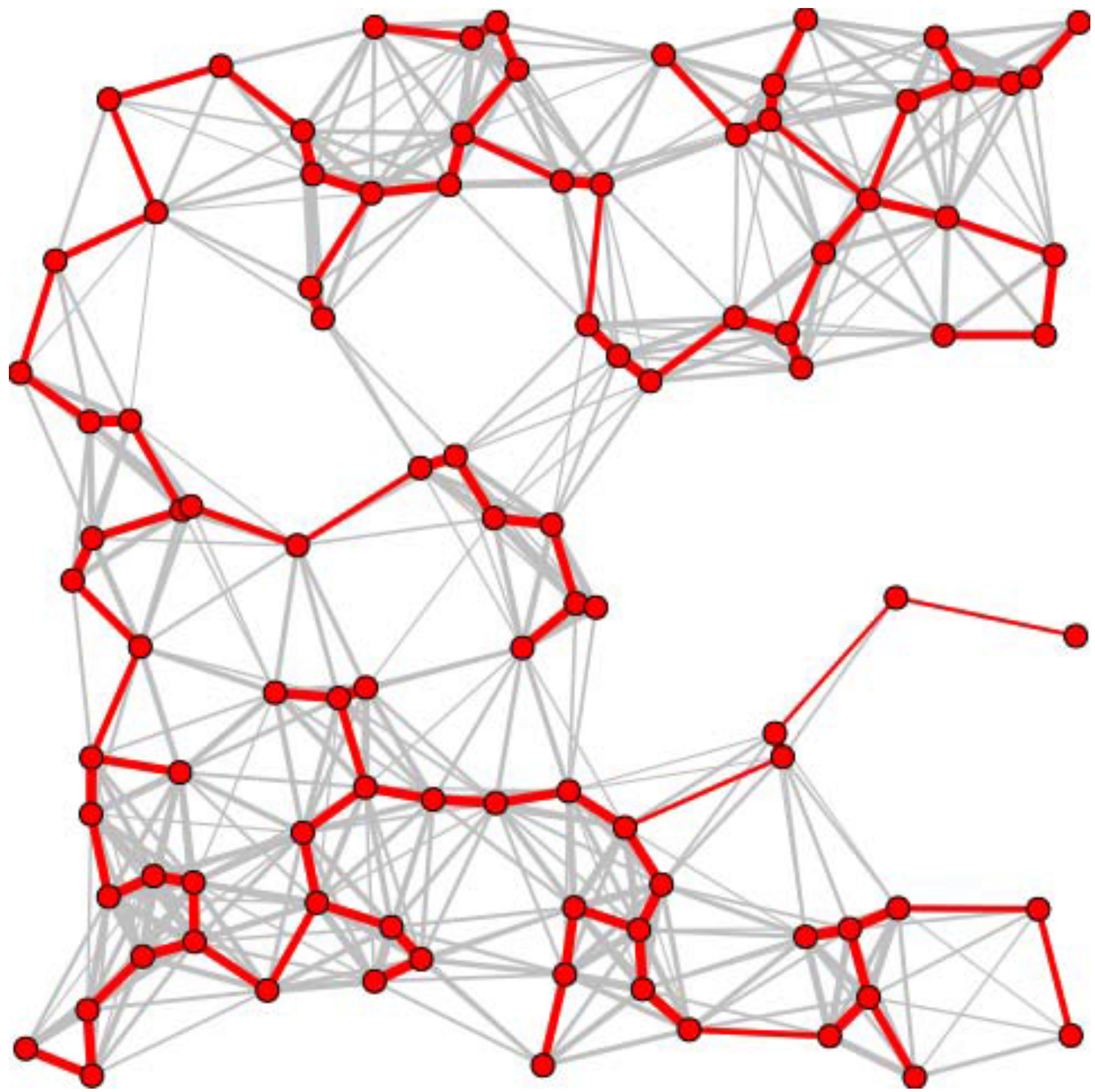
Problem

- Given a connected undirected graph G=(V,E) with weight function w: E —> $\mathbb{R}$.
- Compute a minimum spanning tree (MST), i.e., a tree that connects all vertices with minimum weight

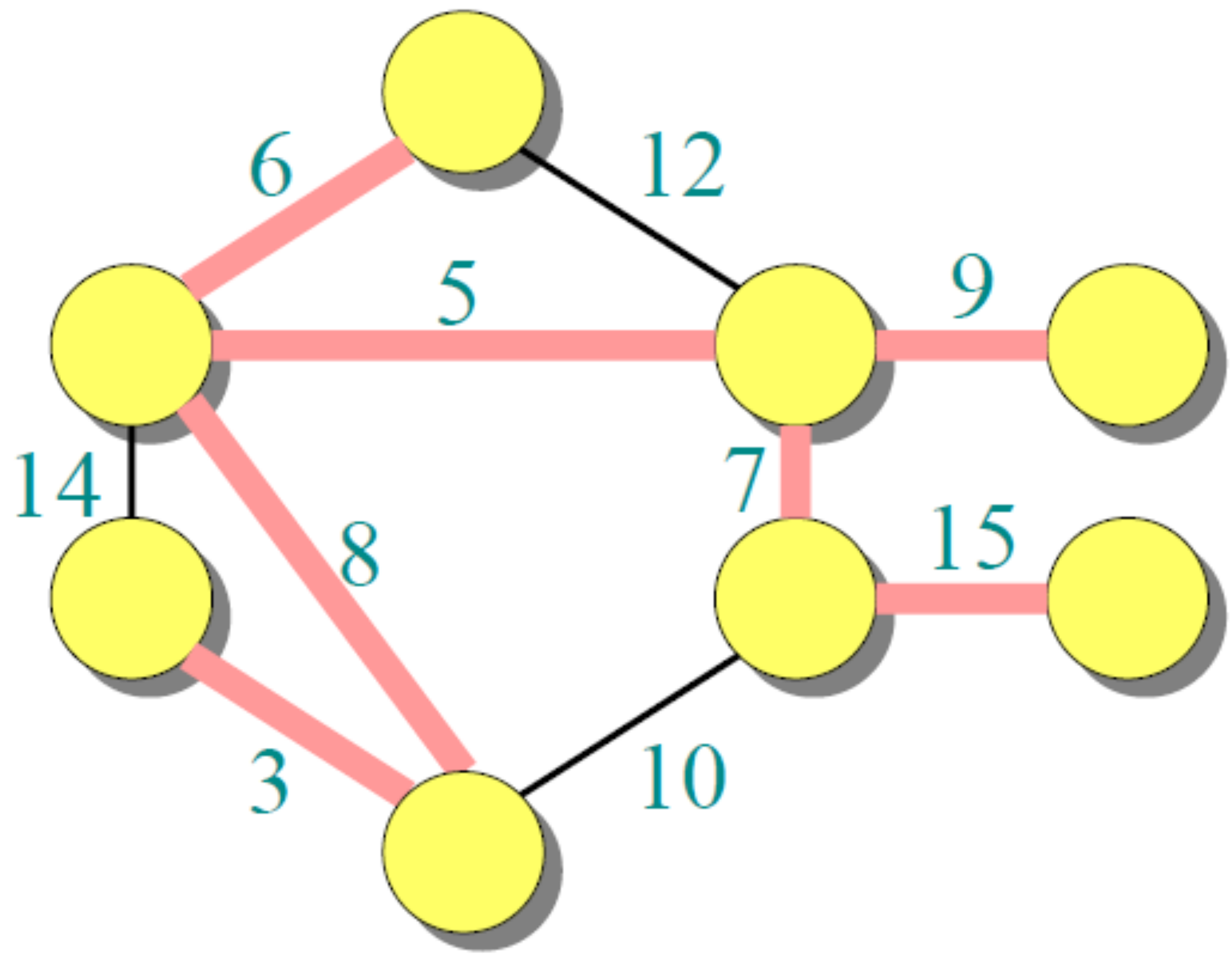
$$w(T) = \sum_{(u,v)\in T} w(u,v).$$

- Why of interest?
  One example would be a telecommunications company laying out cables to a neighborhood.

Spanning tree

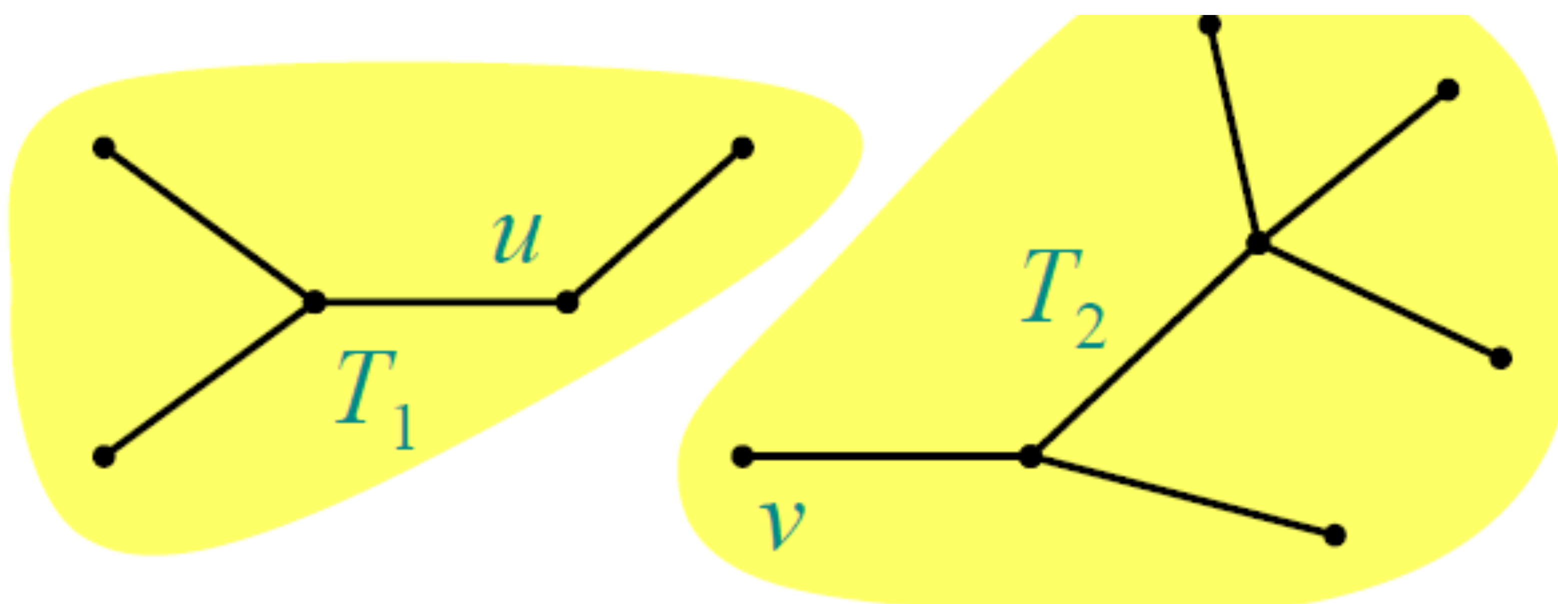


MST



Optimal substructure

- Consider an MST T of graph G (other edges not shown).
- Remove any edge (u,v) ϵ T.
- Then, T is partioned into subtrees T_1 and T_2.



Theorem

(a) Subtree T_1 is a MST of graph $G_1 = (V_1, E_1)$ with V_1 being the set of all vertices of T_1 and E_1 being the set of all edges $\in G$ that connect vertices $\in V_1$.

(b) Subtree T_2 is a MST of graph $G_2 = (V_2, E_2)$ with V_2 being the set of all vertices of T_2 and E_2 being the set of all edges $\in G$ that connect vertices $\in V_2$.

Proof (only (a), (b) is analogous):

- $w(T) = w(T_1) + w(T_2) + w(u,v)$
- Assume S_1 was a MST for G_1 with lower weight than T_1.
- Then, $S = S_1 \cup T_2 \cup \{(u,v)\}$ would be an MST for G with lower weight than T.
- Contradiction!

Greedy choice property

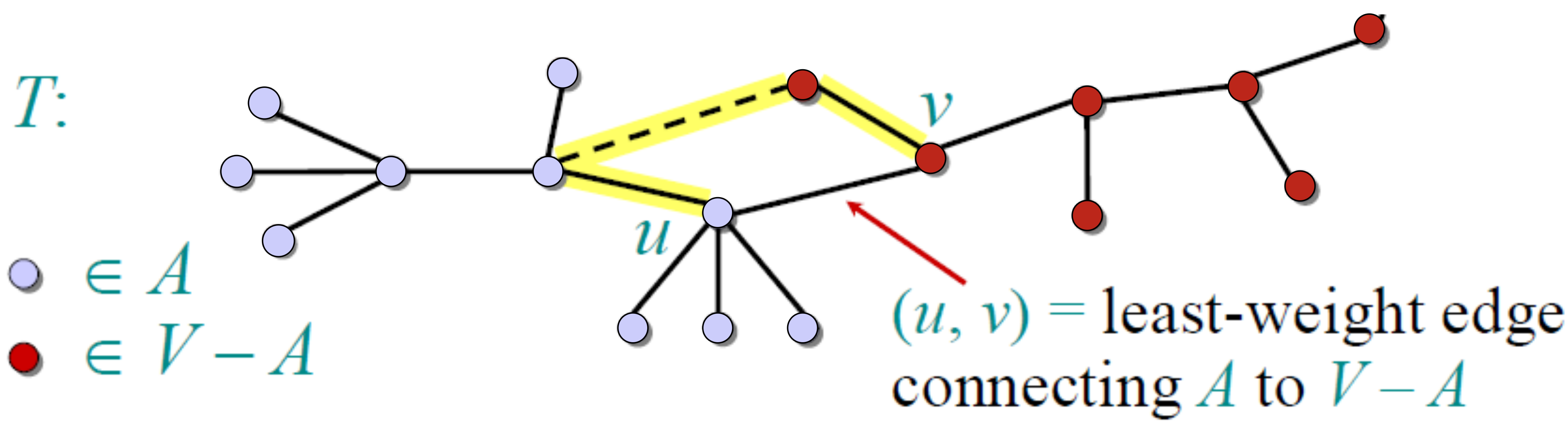
Theorem:

- Let T be the MST of graph G = (V,E) and let $A \subset V$.
- Let $(u,v) \in E$ be the edge with least weight connecting A to $V \setminus A$.
- Then, $(u,v) \in T$.

Greedy choice property

Proof:

- Suppose (u,v) is not part of T.
- Then, consider the path from u to v within T.
- Replace the first edge on this path that connects a vertex in A to a vertex in V \ A with (u,v).
- This results in an MST with smaller weight. Contradiction!



Prim's algorithm

Idea:

- Develop a greedy algorithm that iteratively increases A and, consequently, decreases V\A.
- Maintain V\A as a min-priority queue Q (min-priority queue analogous to max-priority queue).
- Key each vertex in Q with the weight of the least weight edge connecting it to a vertex in A (if no such edge exists, the weight shall be infinity).
- Then, always add the vertex of V\A with minimal key to A.

Min-priority queues

Definition (recall):

A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a key.

Definition (implementation as min-heap):

A min-priority queue is a priority queue that supports the following operations:

- Minimum (S): return element from S with smallest key. [O(1)]
- Extract-Min (S): remove and return element from S with smallest key. [O(lg n)]
- Decrease-Key (S,x,k): decrease the value of the key of element x to k, where k is assumed to be smaller or equal than the current key. [O(lg n)]
- Insert (S,x): add element x to set S. [O(lg n)]

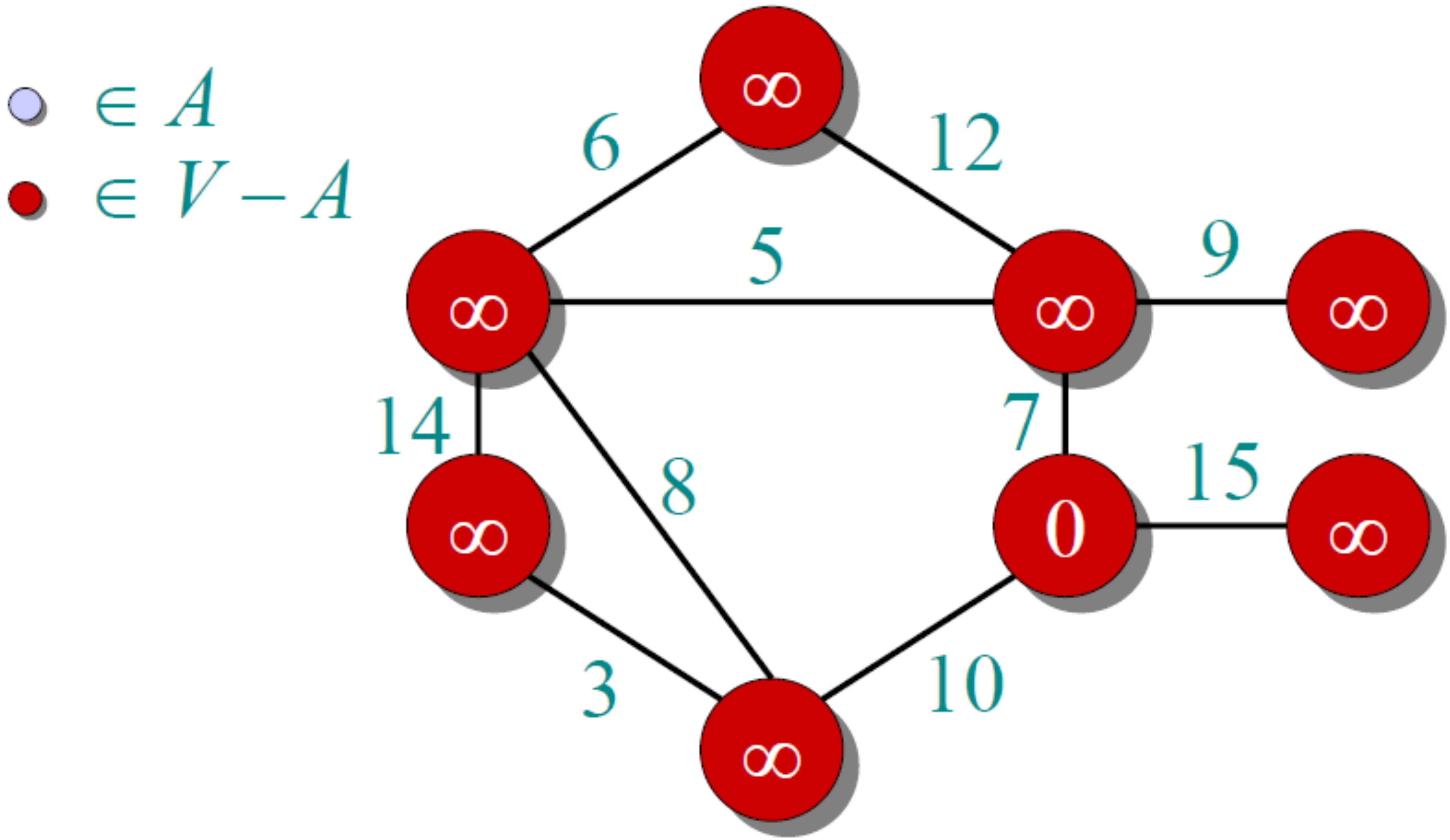
Prim's algorithm

$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        for each $v \in Adj[u]$
            do if $v \in Q$ and $w(u, v) < key[v]$
                then $key[v] \leftarrow w(u, v)$
                    $\pi[v] \leftarrow u$

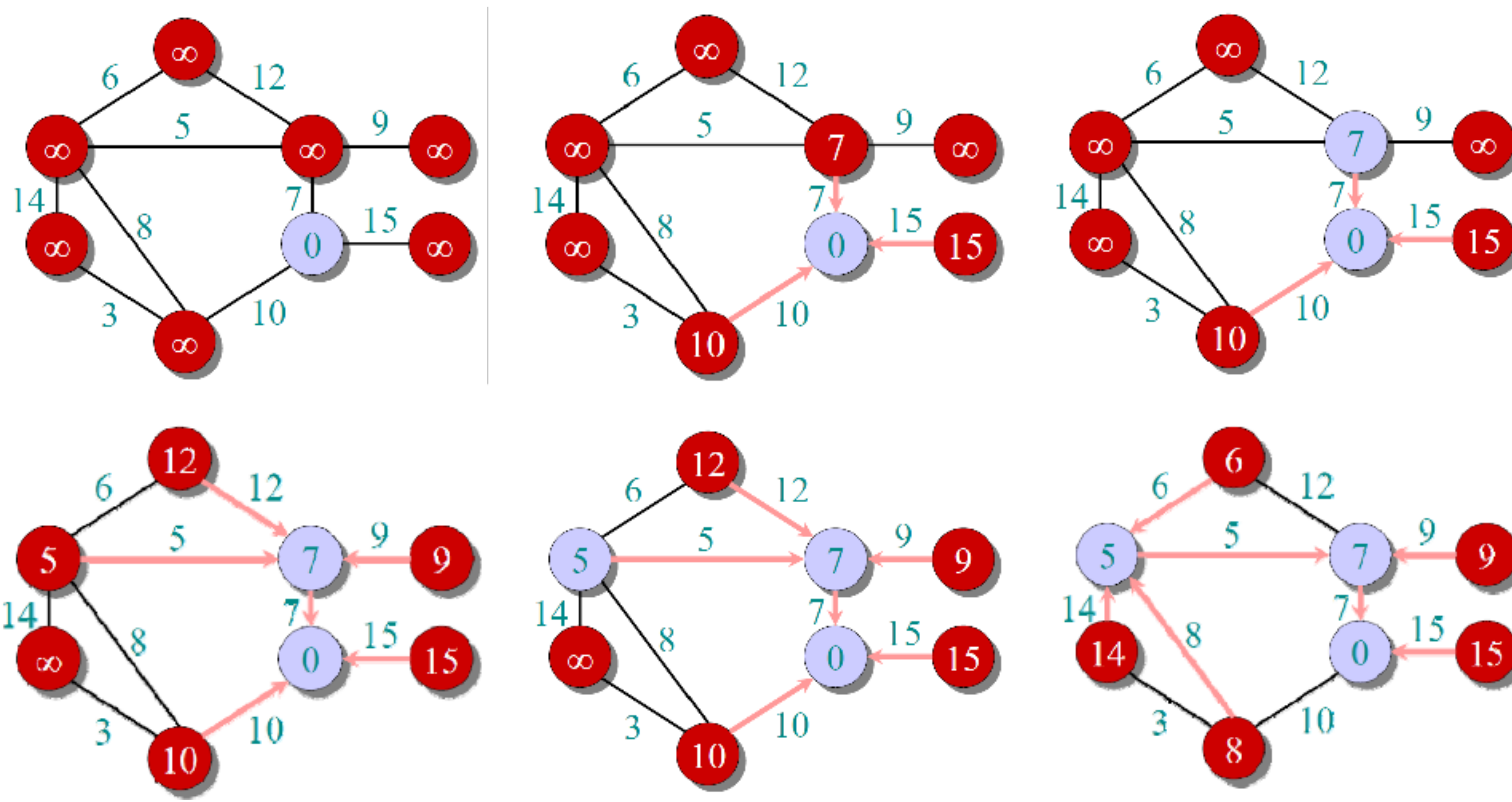
Prim’s algorithm

- The output is provided by storing predecessors $\pi[v]$ of each node v.
- The set $\{(v, \pi[v])\}$ forms the MST.

Example

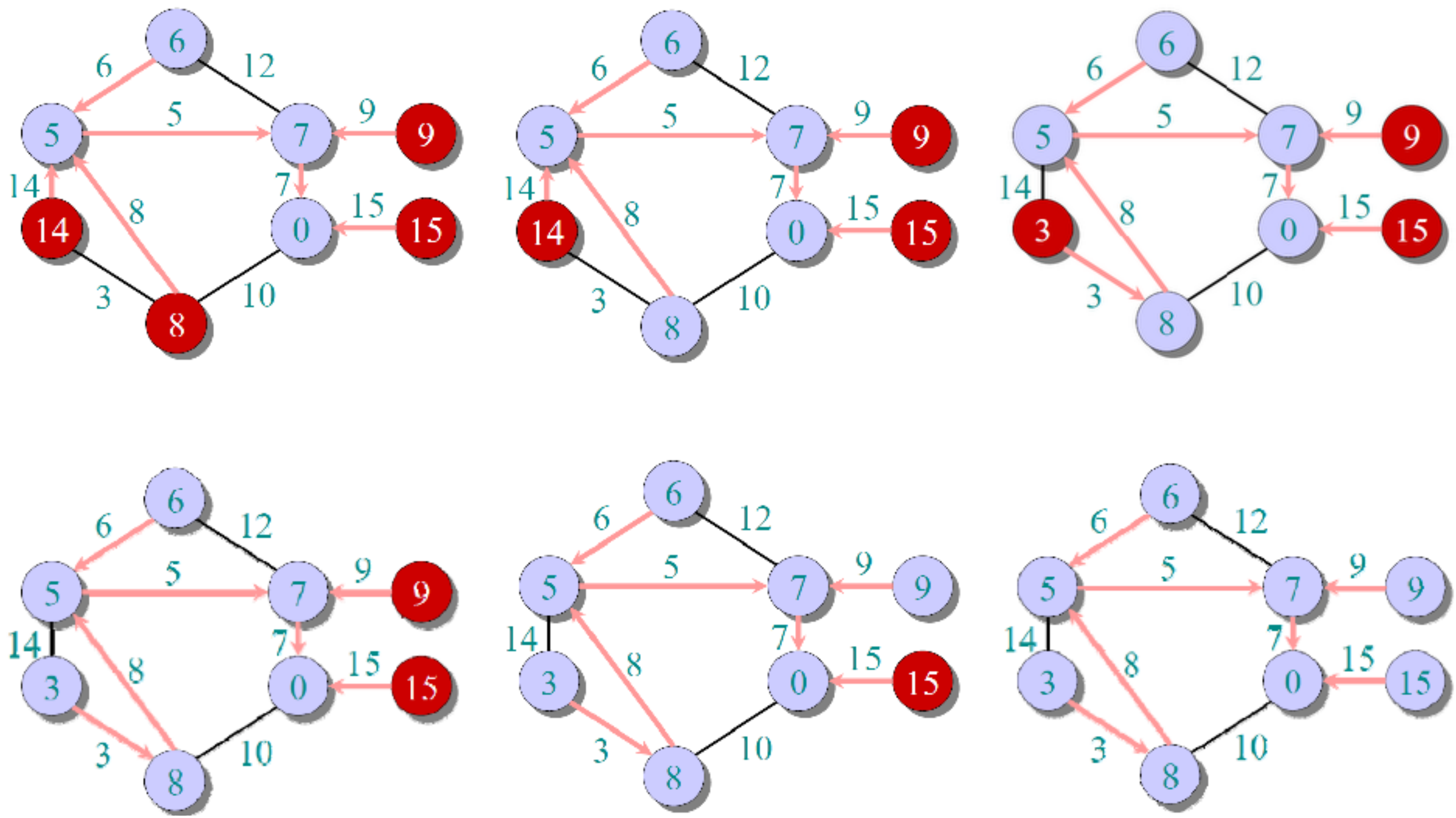


Example



∞ ∞ ∞ ∞ ∞ ∞ ∞ 0
6 12 5 9 14 8 7 15 3 10
∞ ∞ 7 ∞ ∞ 0 15 10
6 12 5 9 14 8 7 15 3 10
∞ ∞ 7 ∞ ∞ 0 15 10
6 12 5 9 14 8 7 15 3 10
12 5 7 9 ∞ 0 15 10
6 12 5 9 14 8 7 15 3 10
12 5 7 9 ∞ 0 15 10
6 12 5 9 14 8 7 15 3 10
6 5 7 9 14 0 15 8
6 12 5 9 14 8 7 15 3 10

Example



Analysis

$$\underbrace{\begin{array}{l} Q \leftarrow V \\ key[v] \leftarrow \infty \text{ for all } v \in V \\ key[s] \leftarrow 0 \text{ for some arbitrary } s \in V \end{array}}_{\Theta(V) \text{ total}}$$

$$\underbrace{\begin{array}{l} \textbf{while } Q \neq \emptyset \\ \quad \textbf{do } u \leftarrow \text{EXTRACT-MIN}(Q) \\ \quad \quad \underbrace{\begin{array}{l} \textbf{for each } v \in Adj[u] \\ \quad \textbf{do if } v \in Q \text{ and } w(u,v) < key[v] \\ \quad \quad \textbf{then } \boxed{key[v] \leftarrow w(u,v)} \\ \quad \quad \quad \pi[v] \leftarrow u \end{array}}_{degree(u) \text{ times}} \end{array}}_{|V| \text{ times}}$$

$\Theta(E)$ implicit DECREASE-KEY's.

Notation Θ(V) means Θ(|V|).

Analysis

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
min-heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
array	$O(V)$	$O(1)$	$O(V^2)$

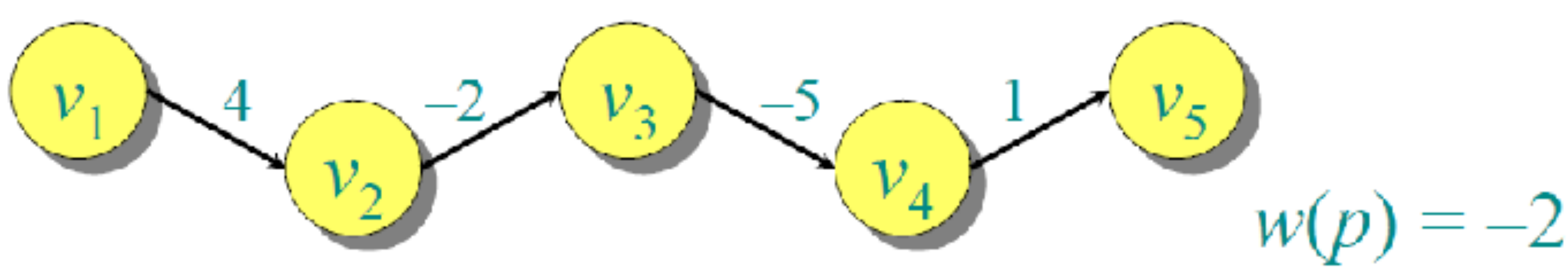
5.4 Shortest Paths

Definitions: Path

- Consider a directed graph G=(V,E), where each edge e є E is assigned a non-negative weight w: E —> R^+.
- A path is a sequence of vertices in the graph, where two consecutive vertices are connected by a respective edge.
- The weight of a path p=(v_1,…,v_k) is defined by

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}) .$$

- Example:



Definition: Shortest Path

- A shortest path from a vertex u to a vertex v in a graph G is a path of minimum weight.
- The weight of a shortest path from u to v is defined as $\delta(u,v) = \min \{w(p): p \text{ is a path from } u \text{ to } v\}$.
- Note that $\delta(u,v) = \infty$, if no path from u to v exists.

- Why of interest?

  One example is finding a shortest route in a road network.

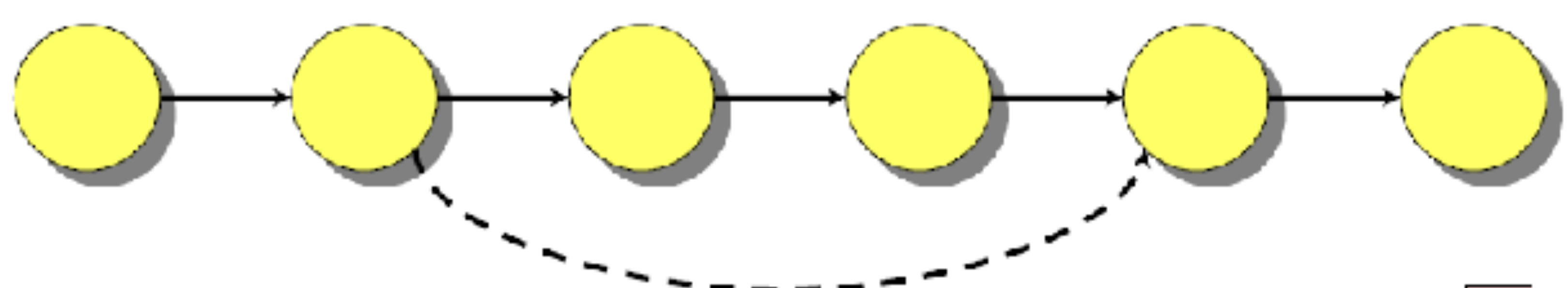
Optimal substructure

Theorem:

- A subpath of a shortest path is a shortest path.

Proof:

- Let $p=(v_1,\ldots,v_k)$ be a shortest path and $q=(v_i,\ldots,v_j)$ a subpath of p.
- Assume that q is not a shortest path.
- Then, there exists a shorter path from v_i to v_j than q.
- But then, there is also a shorter path from v_1 to v_k than p. Contradiction!

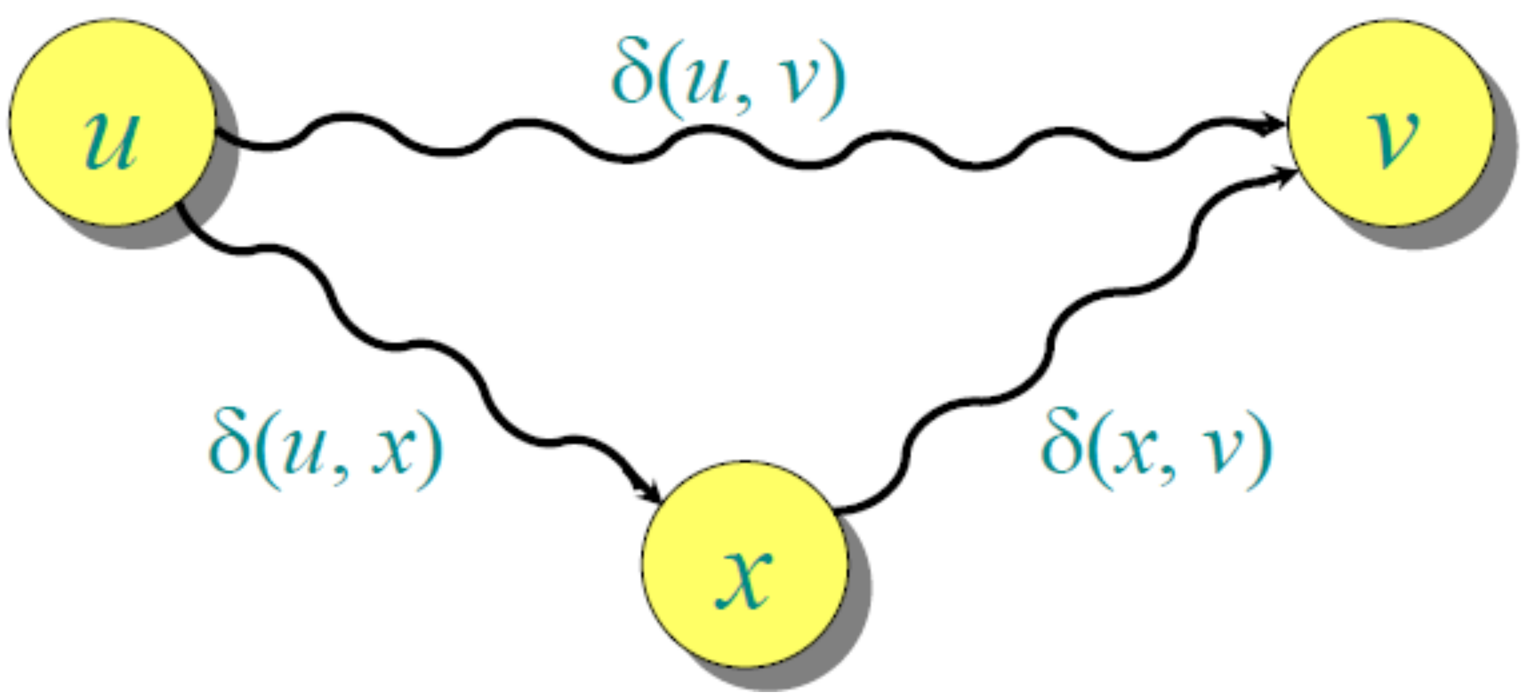


Triangle inequality

Theorem:

- For all u,v,x ϵ V, we have that δ(u,v) ≤ δ(u,x) + δ(x,v).

Proof:



(Single-source) Shortest Paths

Problem:

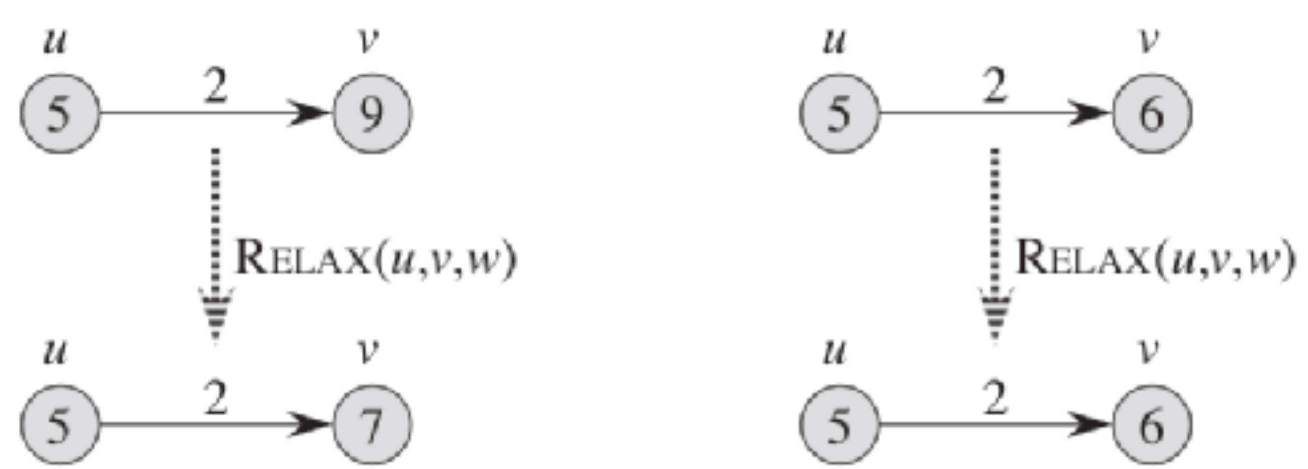
- Given a source vertex s є V, find for all v є V the shortest-path weights δ(s,v).

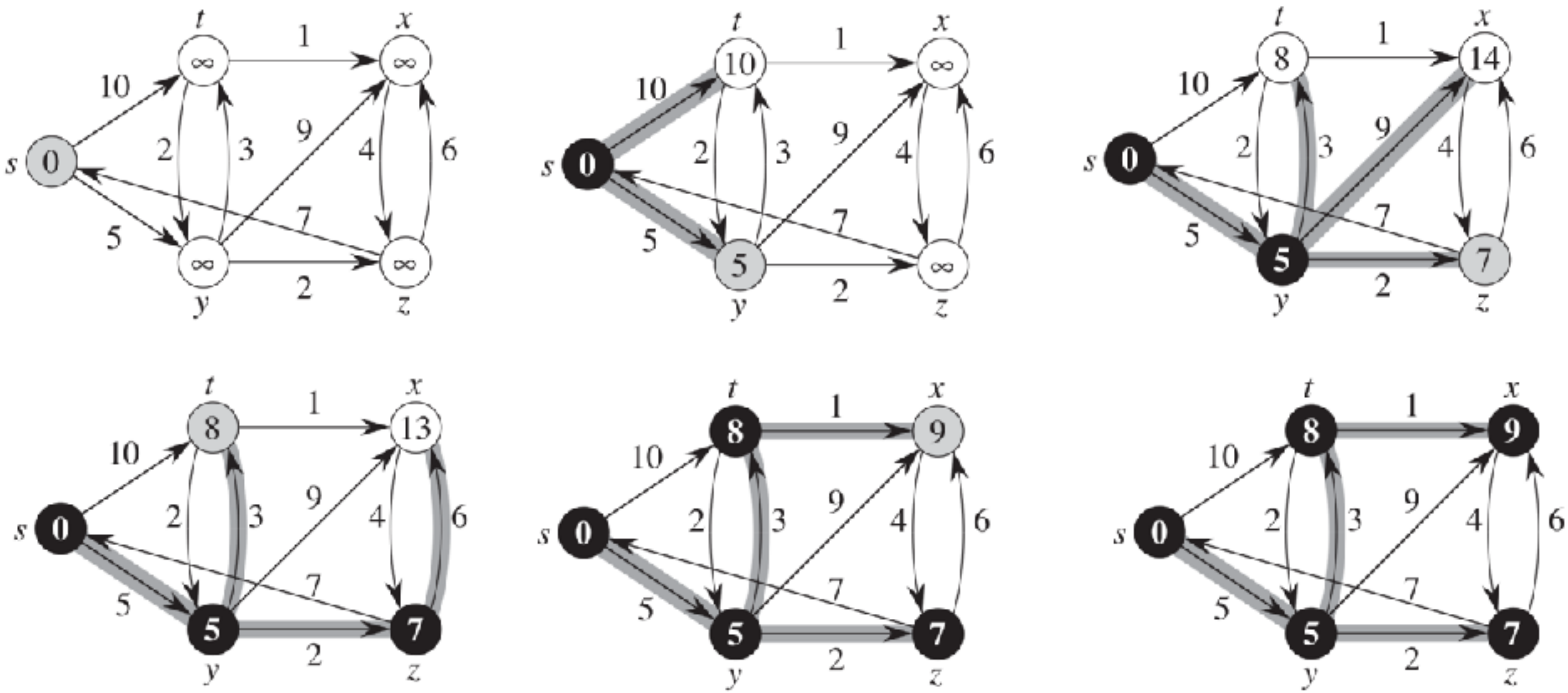
Idea: Greedy approach.

1. Maintain a set S of vertices whose shortest-path distances from s are known.
2. At each step, add to S the vertex v є V\S whose distance estimate from s is minimal.
3. Update the distance estimates of vertices adjacent to v.

Dijkstra’s algorithm

```
d[s] := 0
for each v ∈ V\{s}
  d[v] := ∞
S := Ø
Q := V   // min-priority queue maintaining V \ S.
while Q ≠ Ø
    u := Extract-Min(Q)
    S := S ∪ {u}
    for each v ∈ Adj[u]
        if d[v] > d[u] + w(u,v)     // *****
        then d[v] := d[u] + w(u,v)  // Relaxation
            π[v] := u               // *****
```






```
while Q ≠ Ø
    u := Extract-Min(Q)
    S := S ∪ {u}
    for each v ∈ Adj[u]
        if d[v] > d[u] + w(u,v)
        then d[v] := d[u] + w(u,v)
             π[v] := u
```

$S = \{s, y, z, t, x\}$

Correctness

Correctness is shown in 3 steps:

(I) $d[v] \geq \delta(s,v)$ at all steps (for all v)

(II) $d[v] = \delta(s,v)$ after relaxation from u, if (u,v) on shortest path (for all v)

(III) algorithm terminates with $d[v] = \delta(s,v)$

Correctness (i)

Lemma:

- Initialising d[s]:=0 and d[v]:=∞ for all v ϵ V\{s} establishes d[v] ≥ δ(s,v) for all v ϵ V.
- This invariant is maintained over any sequence of relaxation steps.

Proof:

Suppose the Lemma is not true, then let v be the first vertex for which d[v] < δ(s,v) and let u be the vertex that caused d[v] to change by d[v]:=d[u]+w(u,v).

Then,
$$
\begin{aligned}
d[v] &< \delta(s, v) && \text{supposition} \\
&\le \delta(s, u) + \delta(u, v) && \text{triangle inequality} \\
&\le \delta(s,u) + w(u, v) && \text{sh. path} \le \text{specific path} \\
&\le d[u] + w(u, v) && v \text{ is first violation}
\end{aligned}
$$

Contradiction!

Correctness (ii)

Lemma:

- Let u be v's predecessors on a shortest path from s to v.
- Then, if $d[u] = \delta(s,u)$, we have $d[v] = \delta(s,v)$ after the relaxation of edge (u,v).

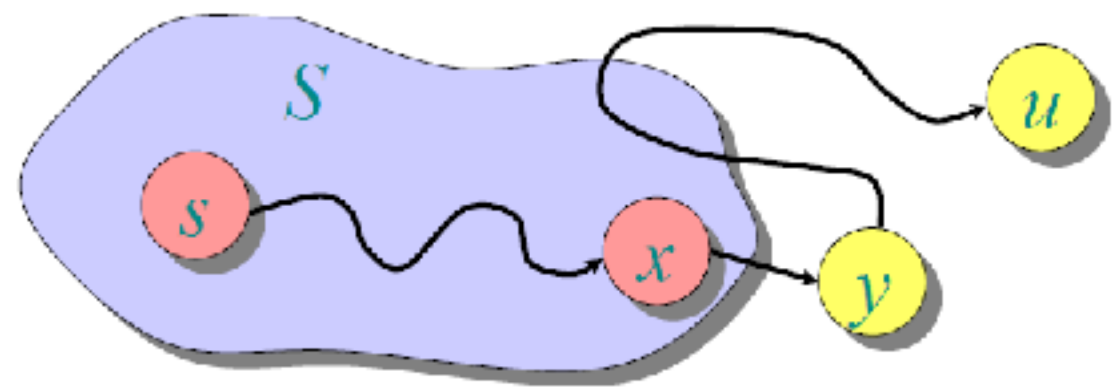
Proof:

- Observe that $\delta(s,v) = \delta(s,u) + w(u,v)$.
- Suppose that $d[v] > \delta(s,v)$ before relaxation (else: done).
- Then, $d[v] > \delta(s,v) = \delta(s,u) + w(u,v) = d[u] + w(u,v)$ (if clause in the algorithm).
- Thus, the algorithm sets $d[v] := d[u] + w(u,v) = \delta(s,v)$.

Correctness (iii)

Theorem:

- Dijkstra‘s algorithm terminates with d[v] = δ(s,v) for all v є V.



Proof:

- It suffices to show that d[v] = δ(s,v) for every v є V when v is added to S.
- Suppose u is the first vertex added to S with d[u] > δ(s,u).
- Let y be the first vertex in V \ S along the shortest path from s to u, and let x be its predecessor.
- Then, d[x] = δ(s,x) and d[y] = δ(s,y) ≤ δ(s,u) < d[u].
- But we chose u such that d[u] ≤ d[y]. Contradiction!

Analysis

$$
\left.\begin{array}{l}
\textbf{while } Q \neq \emptyset \\
\quad \textbf{do } u \leftarrow \text{EXTRACT-MIN}(Q) \\
\quad\quad S \leftarrow S \cup \{u\} \\
\quad\quad \left.\begin{array}{l}
\textbf{for each } v \in Adj[u] \\
\quad \textbf{do if } d[v] > d[u] + w(u, v) \\
\quad\quad \textbf{then } d[v] \leftarrow d[u] + w(u, v)
\end{array}\right\} \substack{degree(u) \\ \text{times}}
\end{array}\right\} \substack{|V| \\ \text{times}}
$$

- Just as for Prim's minimum spanning tree algorithm, we get the computation time

$$\Theta(V \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{DECREASE-KEY}})$$

- Hence, depending on what data structure we use, we get the same computation times as for Prim's algorithm.

Unweighted graphs

- Suppose that we have an unweighted graph, i.e., the weights $w(u,v)=1$ for all $(u,v) \in E$.
- Can we improve the performance of Dijkstra's algorithm?
- Observation: The vertices in our data structure Q are processed following the FIFO principle.
- Hence, we can replace the min-priority queue with a queue.
- This leads to a breadth-first search.

BFS algorithm

```
d[s] := 0
for each v ∈ V\{s}
  d[v] := ∞
Enqueue (Q,s)
while Q ≠ Ø
  u := Dequeue(Q)
  for each v ∈ Adj[u]
      if d[v] = ∞
      then d[v] := d[u] + 1
           π[v] :=u
           Enqueue(Q,v)
```

Analysis: BFS algorithm

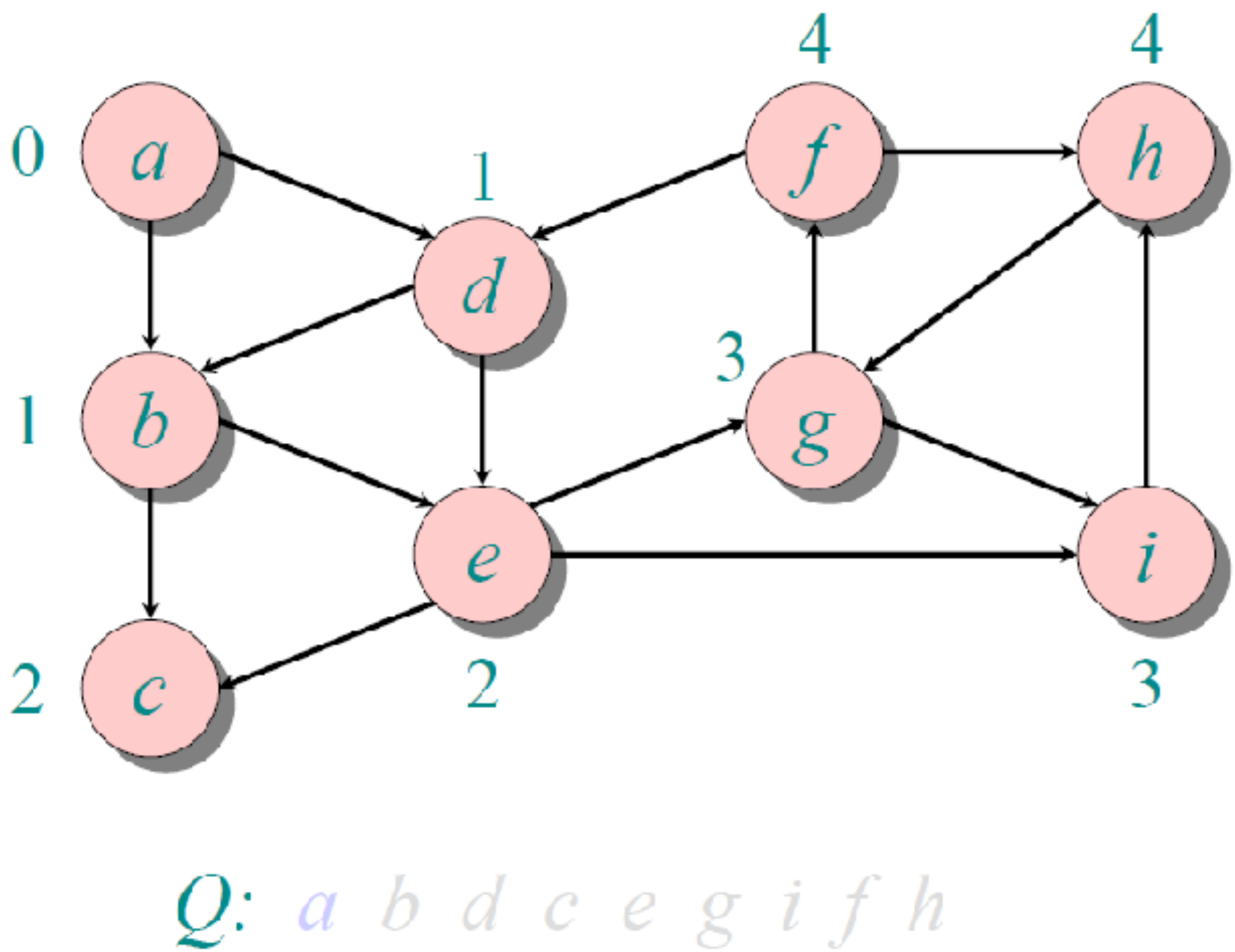
Correctness:

- The FIFO queue Q mimics the min-priority queue in Dijkstra‘s algorithm.
- Invariant:
  If v follows u in Q, then d[v]=d[u] or d[v]=d[u]+1.
- Hence, we always dequeue the vertex with smallest d.

Time complexity:

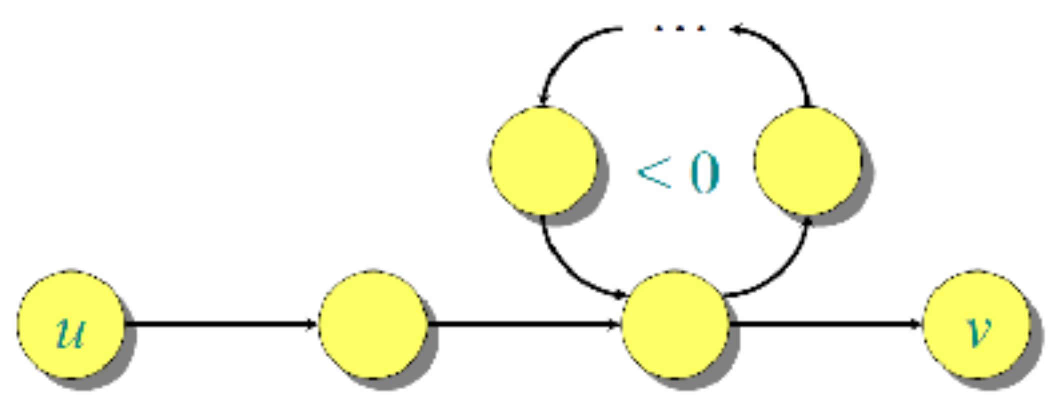
- $O(|V|\ T_{Dequeue} + |E|\ T_{Enqueue}) = O(|V|+|E|)$

Example: BFS algorithm



Negative weights

- We had postulated that all weights are nonnegative.
- How can we extend the algorithm to also handle negative entries?
- The problems are caused by negative weight cycles.



- Goal: Find shortest-path lengths from a source vertex s ϵ V to all vertices v ϵ V or determine existence of a negative-weight cycle.

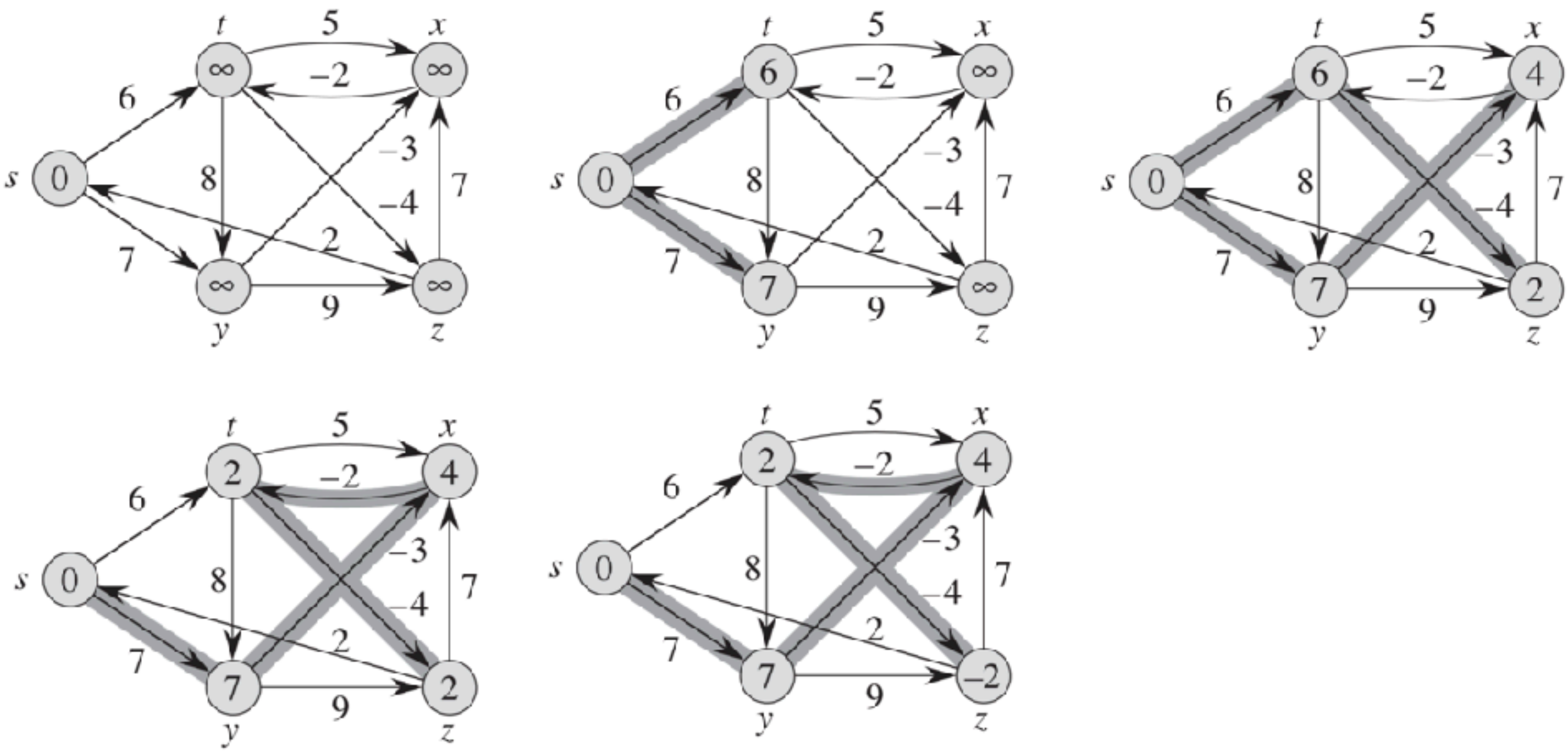
Bellmann-Ford algorithm

```
d[s] := 0
for each v ∈ V\{s}
  d[v] := ∞
for i:=1 to |V|-1
    for each (u,v) ∈ E
        if d[v] > d[u] + w(u,v)
        then  d[v] := d[u] + w(u,v)
              π[v] :=u

for each (u,v) ∈ E
  if d[v] > d[u] + w(u,v)
    report existence of negative-weight cycle
```

Computation time = $O(|V| \cdot |E|)$

Example: Bellman-Ford algorithm




```
for i:=1 to |V|-1
    for each (u,v) ∈ E
        if d[v] > d[u] + w(u,v)
        then  d[v] := d[u] + w(u,v)
              π[v] :=u
```

Correctness

Theorem:

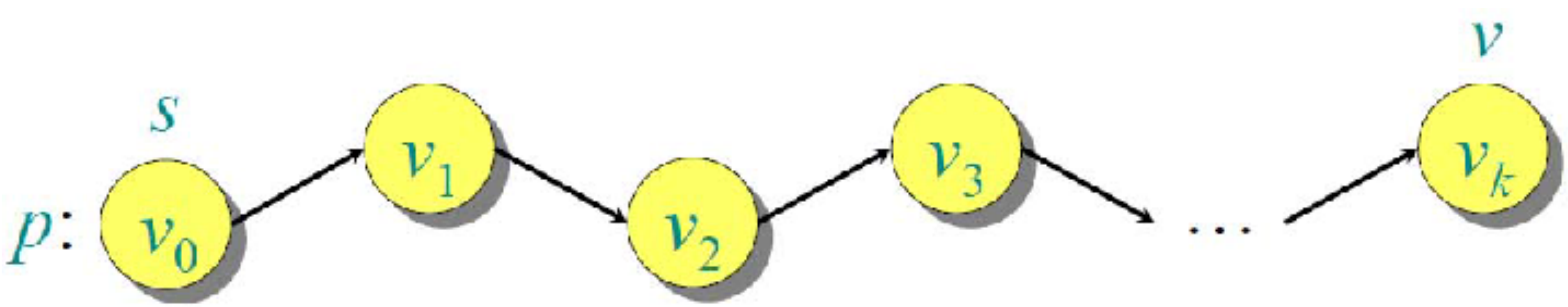
If G = (V,E) contains no negative-weight cycles, then the Bellman-Ford algorithm terminates with $d[v] = \delta(s,v)$ for all $v \in V$.

Proof:

Let $v \in V$ be any vertex.

Consider a shortest path $p=(v_0,\ldots,v_k)$ from s to v.

Then, $\delta(s,v_i) = \delta(s,v_i\text{-}1) + w(v_i\text{-}1,v_i)$ for $i=1,\ldots,k$.



Correctness

Initially, $d[v_0] = 0 = \delta(s,v_0)$.

According to our Lemma from the Dijkstra algorithm we have $d[v] \geq \delta(s,v)$, i.e., $d[v_0]$ is not changed.

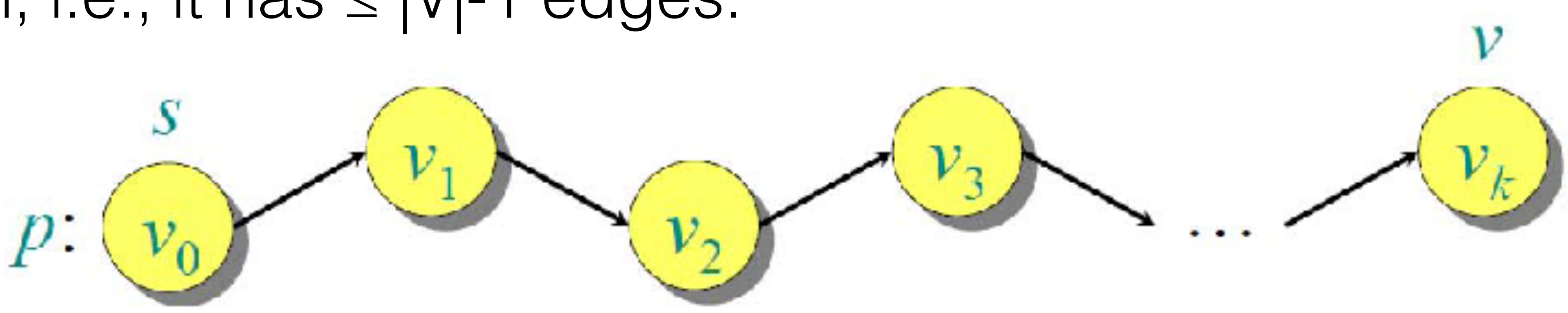
After 1st pass, we have $d[v_1] = \delta(s,v_1)$.

After 2nd pass, we have $d[v_2] = \delta(s,v_2)$.

…

After k-th pass, we have $d[v_k] = \delta(s,v_k)$.

Since G has no negative-weight cycles, p is a simple path, i.e., it has $\leq |V|-1$ edges.



Detecting negative-weight cycles

Corollary:

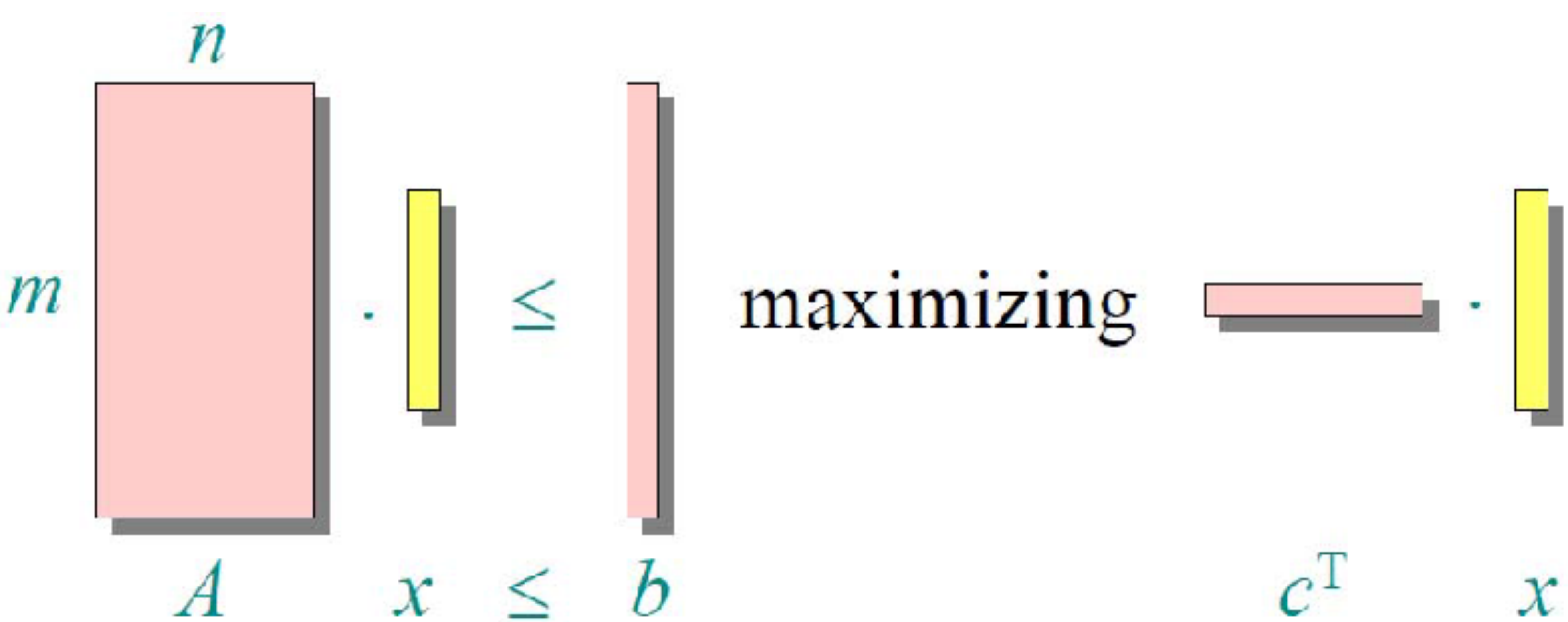
If a value d[v] fails to converge after |V|-1 passes, there exists a negative-weight cycle in G reachable from s.

Excurse: linear programming

Linear programming problem:

Let A be matrix of size m×n, b a vector of size m, and c a vector of size n.

Find a vector x of size n that maximizes $c^T x$ subject to Ax ≤ b, or determine that no such solution exists.



Example: difference constraints

Linear programming example, where each row of
A contains exactly one 1 and one -1, other entries are 0.

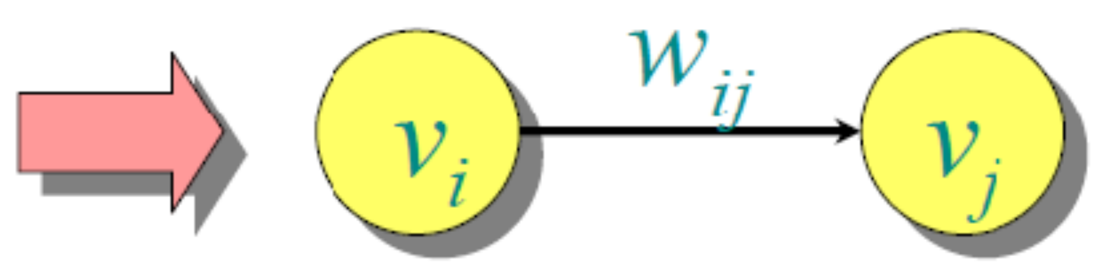
$$\left.\begin{aligned} x_1 - x_2 &\le 3 \\ x_2 - x_3 &\le -2 \\ x_1 - x_3 &\le 2 \end{aligned}\right\} \quad x_j - x_i \le w_{ij}$$

Goal: Find 3-vector x that satisfies these inequations.

Solution: $x_1 = 3$, $x_2 = 0$, $x_3 = 2$.

Build constraint graph (matrix A of size |E| x |V|):

$$x_j - x_i \le w_{ij}$$



Case 1: Unsatisfiable constraints

Theorem:

If the constraint graph contains a negative-weight cycle, then the constraints are unsatisfiable.

Proof:

Suppose we have a negative-weight cycle:

$$v_1 \to v_2 \to \cdots \to v_k \to v_1.$$

Then,

$$\begin{aligned} x_2 - x_1 &\le w_{12} \\ x_3 - x_2 &\le w_{23} \\ &\vdots \\ x_k - x_{k-1} &\le w_{k-1,k} \\ x_1 - x_k &\le w_{k1} \end{aligned}$$

Summing the inequations delivers: LHS = 0, RHS < 0.

Hence, no x exists that satisfies the inequations.

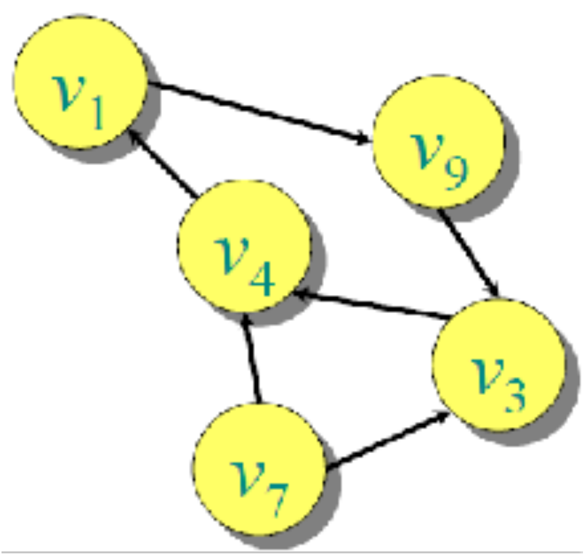
Case 2: Satisfiable constraints

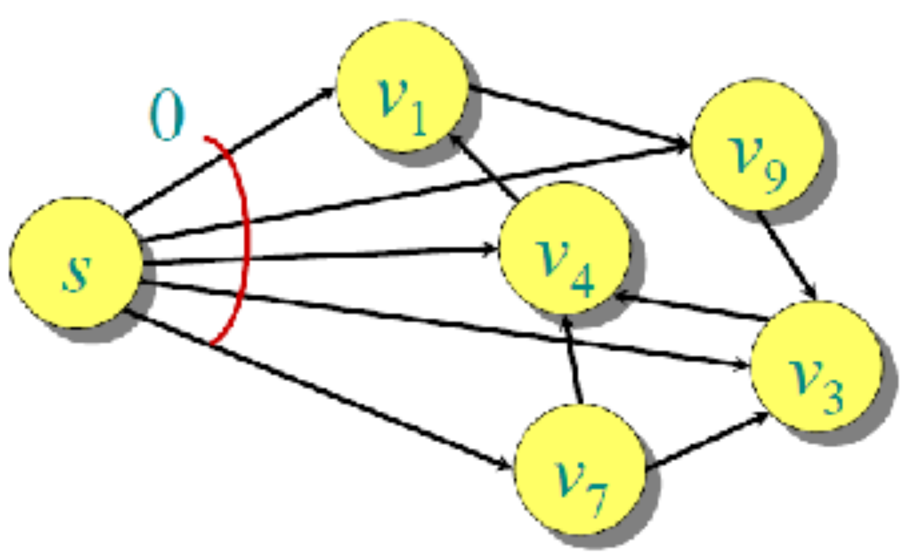
Theorem:

If no negative-weight cycle exists in the constraint graph, then the constraints are satisfiable.

Proof:

Add a vertex s with a 0-weight edge to all vertices. Note that this does not introduce a negative-weight cycle.





Case 2: Satisfiable constraints

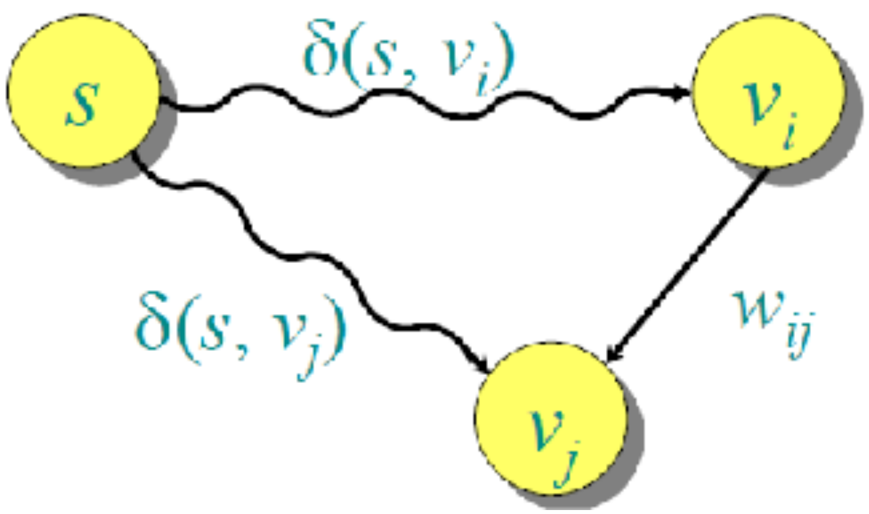
Show that the assignments $x_i = \delta(s,v_i)$ for $i=1,\ldots,n$ solve the constraints.

Consider any constraint $x_j - x_i \leq w_{ij}$.

Then, consider the shortest path from s to v_j and v_i.

The triangle inequality delivers $\delta(s,v_j) \leq \delta(s,v_i) + w_{ij}$.

Since $x_i = \delta(s,v_i)$ and $x_j = \delta(s,v_j)$, constraint $x_j - x_i \leq w_{ij}$ is satisfied.



Bellmann-Ford for linear programming

Corollary:

The Bellman-Ford algorithm can solve a system of m difference constraints on n variables in O(mn) time.

Remark:

Single-source shortest paths is a simple linear programming problem.

5.5 Summary

Summary

- Directed and undirected graphs
- Adjacency matrix vs. adjacency lists
- Graph search: BFS or DFS in $\Theta(|V|+|E|)$
- MST: Prim in $O(|E| \lg(|V|))$ for min-heap
- Single-source Shortest Paths:
  - Dijkstra for non-negative weights in $O((|V|+|E|) \lg(|V|))$ for min-heap
  - BFS for non-weighted edges in $\Theta(|V|+|E|)$
  - Bellman-Ford for all cases in $\Theta(|V|\,|E|)$