

CH08-320201

# **Algorithms and Data Structures**

## **ADS**

### **Lecture 22**

Dr. Kinga Lipskoch

Spring 2019

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

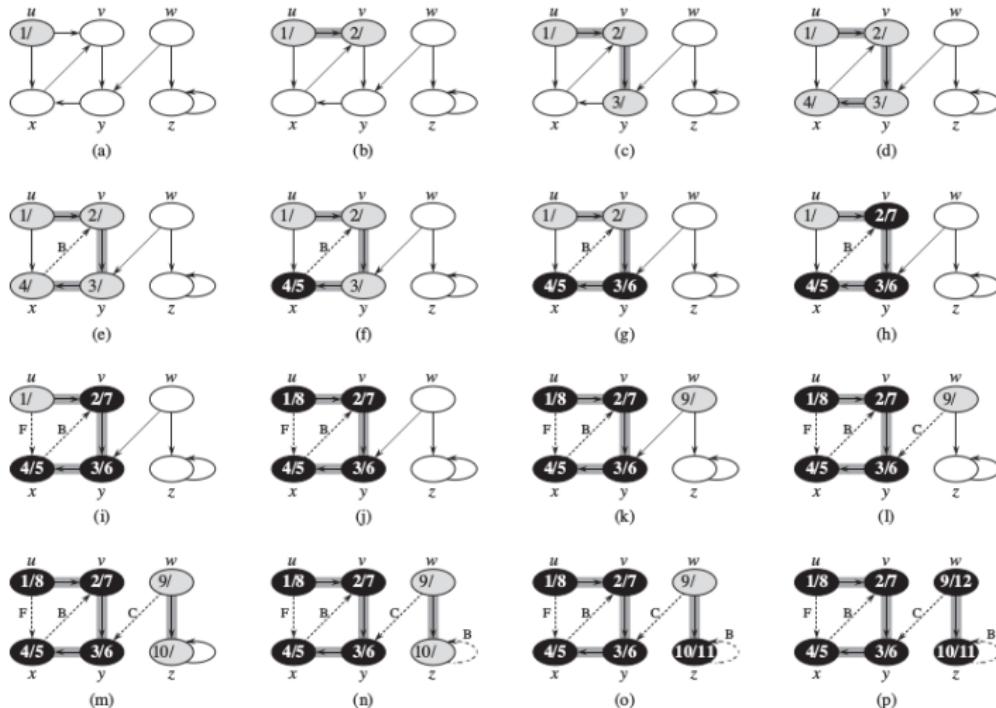
DFS( $G$ )

- 1 **for** each vertex  $u \in G.V$
- 2      $u.color = \text{WHITE}$
- 3      $u.\pi = \text{NIL}$
- 4      $time = 0$
- 5 **for** each vertex  $u \in G.V$
- 6     **if**  $u.color == \text{WHITE}$
- 7         DFS-VISIT( $G, u$ )

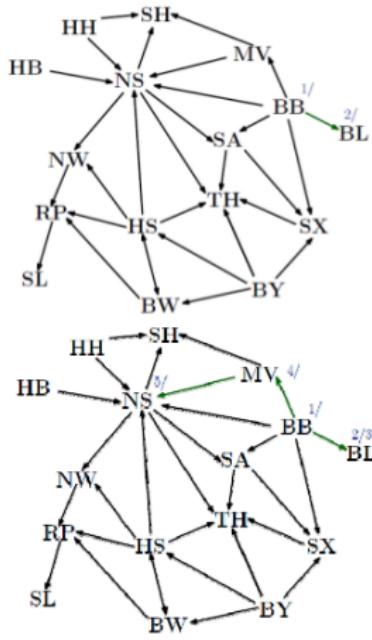
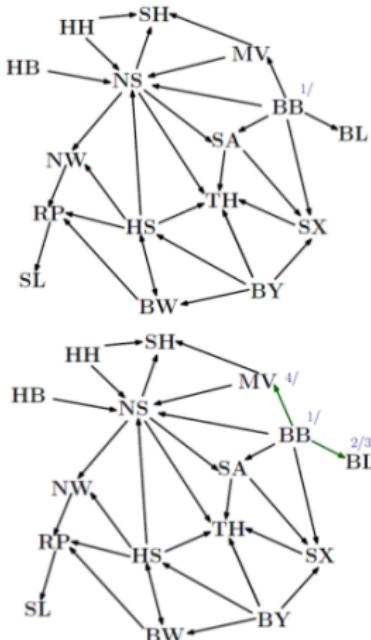
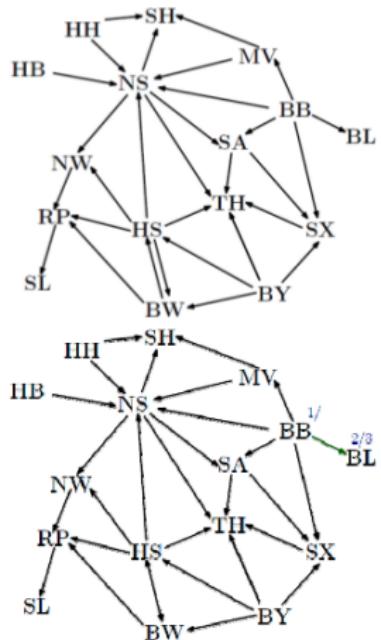
DFS-VISIT( $G, u$ )

- 1      $time = time + 1$
- 2      $u.d = time$
- 3      $u.color = \text{GRAY}$
- 4 **for** each  $v \in G.Adj[u]$
- 5     **if**  $v.color == \text{WHITE}$
- 6          $v.\pi = u$
- 7         DFS-VISIT( $G, v$ )
- 8      $u.color = \text{BLACK}$
- 9      $time = time + 1$
- 10     $u.f = time$

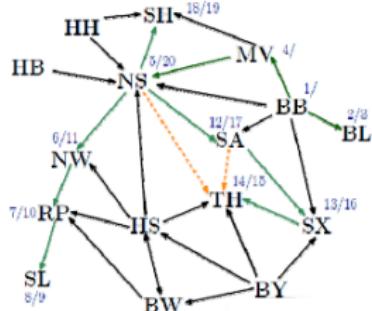
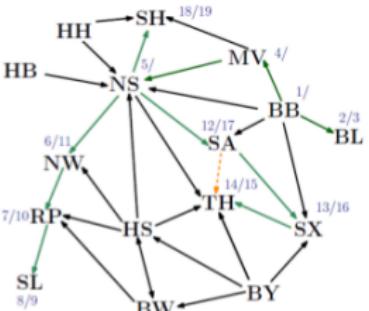
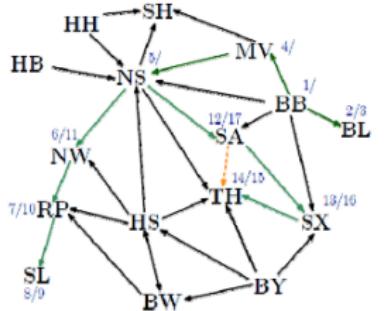
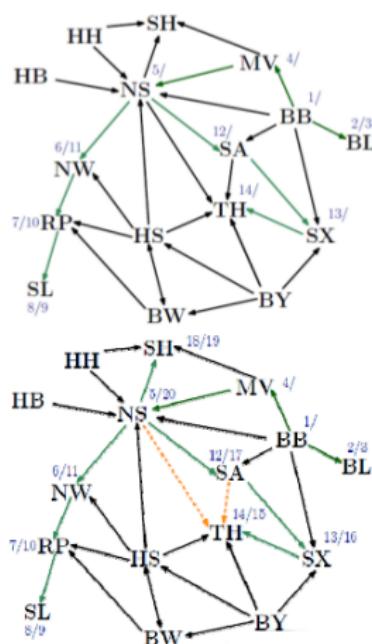
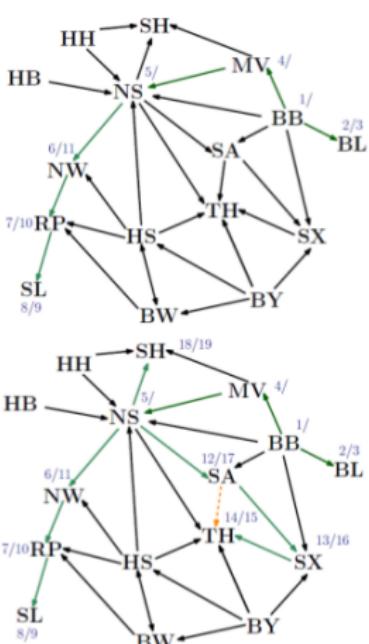
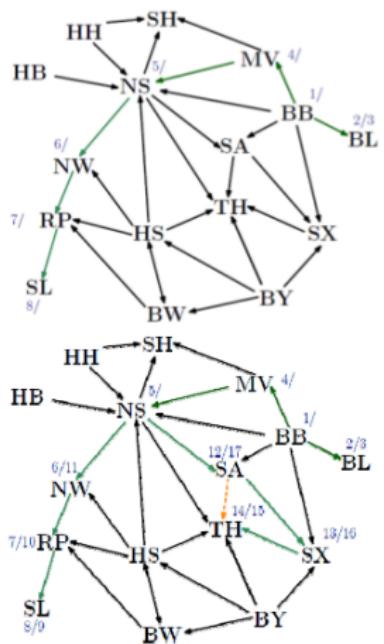
## DFS Example



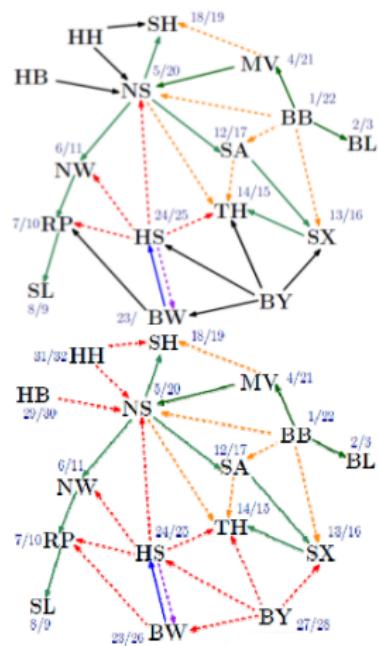
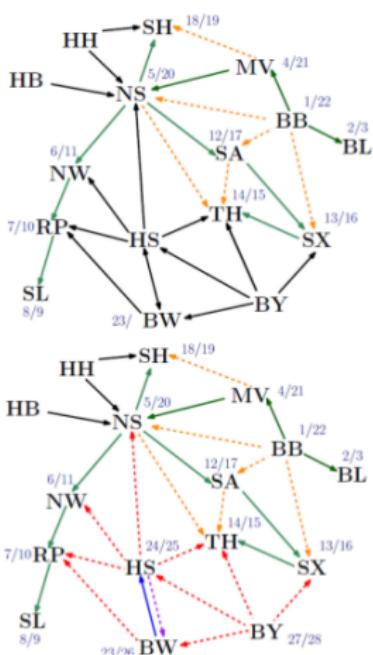
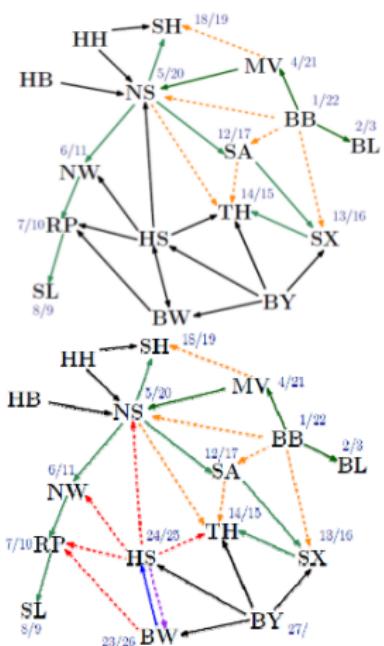
# Another DFS Example (1)



## Another DFS Example (2)



## Another DFS Example (3)



# DFS Analysis

DFS( $G$ )

```
1  for each vertex  $u \in G.V$ 
2       $u.color = \text{WHITE}$ 
3       $u.\pi = \text{NIL}$ 
4       $time = 0$ 
5  for each vertex  $u \in G.V$ 
6      if  $u.color == \text{WHITE}$ 
7          DFS-VISIT( $G, u$ )
```

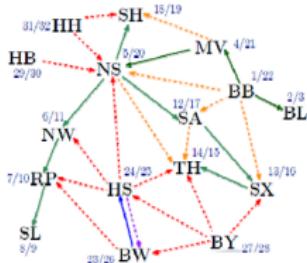
DFS-VISIT( $G, u$ )

```
1   $time = time + 1$ 
2   $u.d = time$ 
3   $u.color = \text{GRAY}$ 
4  for each  $v \in G.Adj[u]$ 
5      if  $v.color == \text{WHITE}$ 
6           $v.\pi = u$ 
7          DFS-VISIT( $G, v$ )
8   $u.color = \text{BLACK}$ 
9   $time = time + 1$ 
10  $u.f = time$ 
```

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.



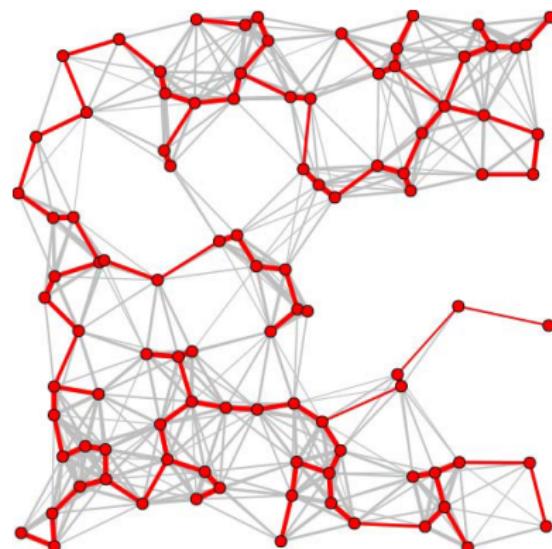
## Minimum Spanning Tree: Problem

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

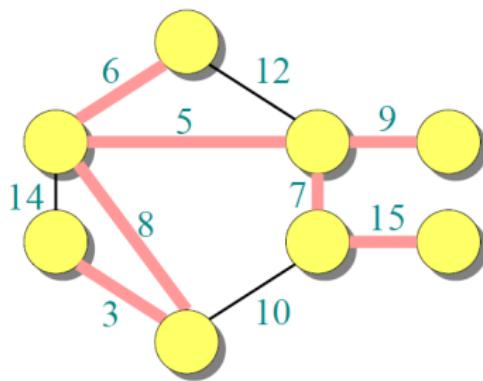
$$w(T) = \sum_{(u,v) \in T} w(u,v).$$

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

## Example Spanning Tree

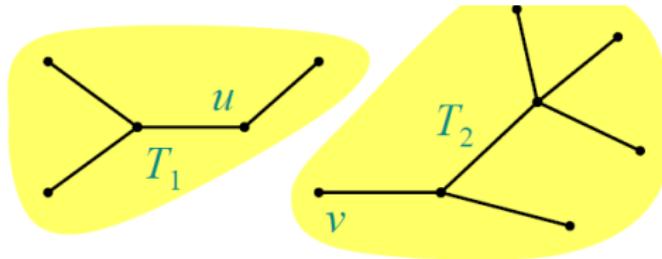


## Example MST



## Optimal Substructure

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



## MST: 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):

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

## Greedy Choice Property (1)

Theorem:

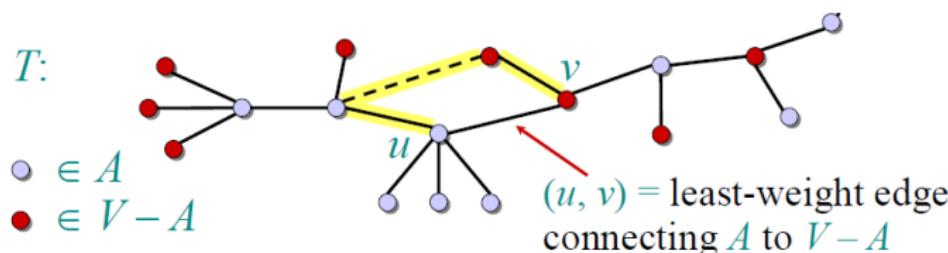
- ▶ Let  $T$  be the MST of graph  $G = (V, E)$  and let  $A \subset V$ .
- ▶ Let  $(u, v) \in E$  be the edge with least weight connecting  $A$  to  $V \setminus A$ .
- ▶ Then,  $(u, v) \in T$ .

## Greedy Choice Property (2)

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  $V \setminus A$  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  $V \setminus A$ .
- ▶ Maintain  $V \setminus A$  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  $V \setminus A$  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:

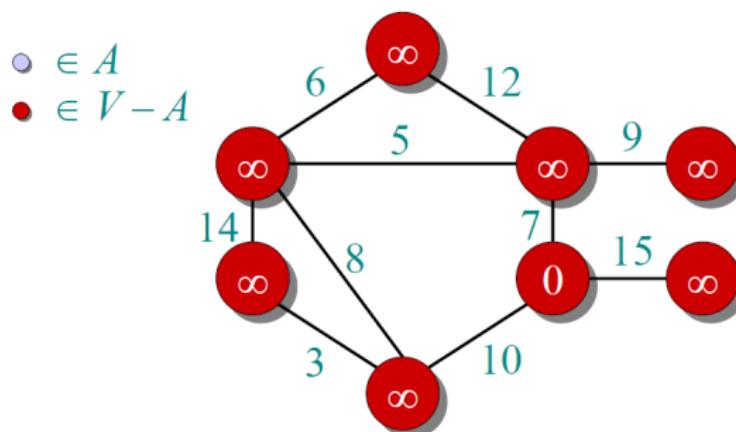
- ▶  $\text{Minimum}(S)$ : return element from  $S$  with smallest key. [ $O(1)$ ]
- ▶  $\text{Extract-Min}(S)$ : remove and return element from  $S$  with smallest key. [ $O(\lg n)$ ]
- ▶  $\text{Decrease-Key}(S, x, k)$ : decrease the value of the key of element  $x$  to  $k$ , where  $k$  is assumed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

## Prim's Algorithm Pseudocode

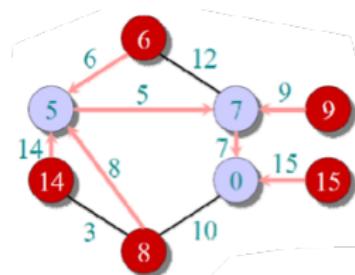
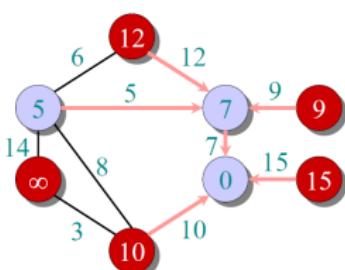
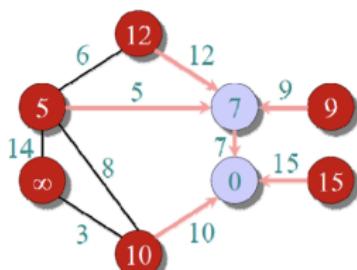
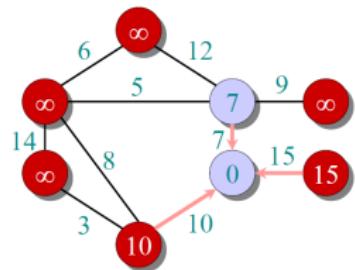
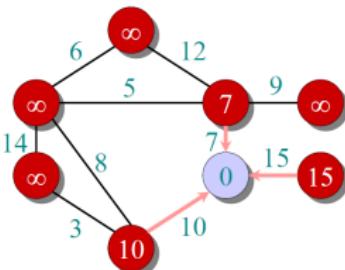
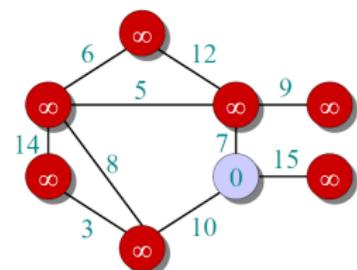
```
 $Q \leftarrow V$ 
 $key[v] \leftarrow \infty$  for all  $v \in V$ 
 $key[s] \leftarrow 0$  for some arbitrary  $s \in V$ 
while  $Q \neq \emptyset$ 
    do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
        for each  $v \in Adj[u]$ 
            do if  $v \in Q$  and  $w(u, v) < key[v]$ 
           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

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

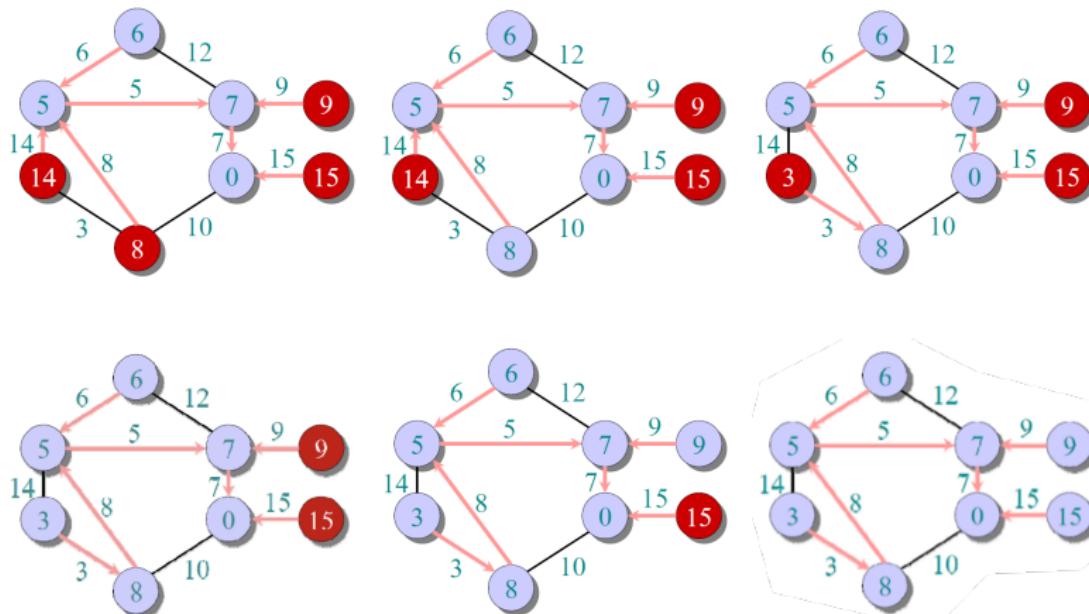
## Example (1)



## Example (2)



## Example (3)



# Complexity Analysis (1)

$\Theta(V)$  total  $\left\{ \begin{array}{l} Q \leftarrow V \\ key[v] \leftarrow \infty \text{ for all } v \in V \\ key[s] \leftarrow 0 \text{ for some arbitrary } s \in V \end{array} \right.$   
**while**  $Q \neq \emptyset$   
**do**  $u \leftarrow \text{EXTRACT-MIN}(Q)$   
**for each**  $v \in Adj[u]$   
**do if**  $v \in Q$  and  $w(u, v) < key[v]$   
**then**  $key[v] \leftarrow w(u, v)$   
 $\pi[v] \leftarrow u$

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

Notation  $\Theta(V)$  means  $\Theta(|V|)$ .

## Complexity Analysis (2)

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

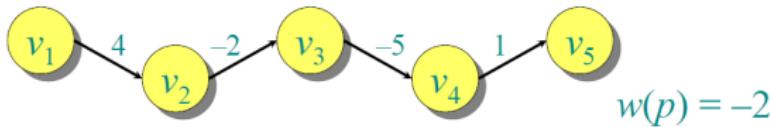
$Q$	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
min-heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
array	$O(V)$	$O(1)$	$O(V^2)$

## Definition: Path

- ▶ Consider a directed graph  $G = (V, E)$ , where each edge  $e \in E$  is assigned a non-negative weight  $w : E \rightarrow \mathbb{R}^+$ .
- ▶ 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  $p = (v_1, \dots, v_k)$  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.

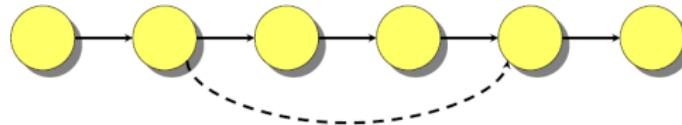
# Optimal Substructure

## Theorem:

A subpath of a shortest path is a shortest path.

## Proof:

- ▶ Let  $p = (v_1, \dots, v_k)$  be a shortest path and  $q = (v_i, \dots, v_j)$  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.

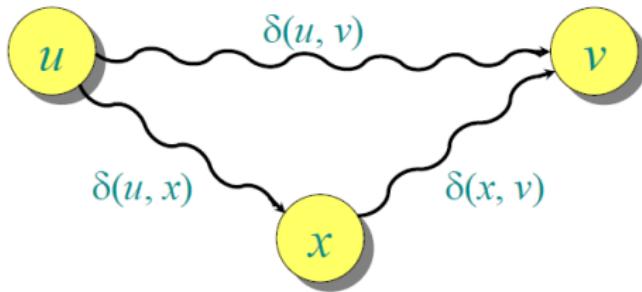


# Triangle Inequality

Theorem:

For all  $u, v, x \in V$ , we have that  $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$ .

Proof:



## (Single-Source) Shortest Paths

### Problem:

Given a source vertex  $s \in V$ , find for all  $v \in V$  the shortest-path weights  $\delta(s, v)$ .

Idea: Greedy approach.

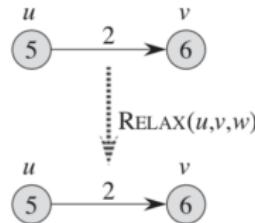
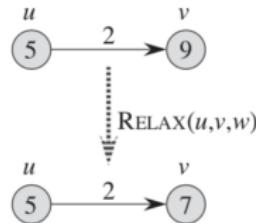
1. Maintain a set  $S$  of vertices whose shortest-path distances from  $s$  are known.
2. At each step, add to  $S$  the vertex  $v \in V \setminus S$  whose distance estimate from  $s$  is minimal.
3. Update the distance estimates of vertices adjacent to  $v$ .

# Dijkstra's Algorithm

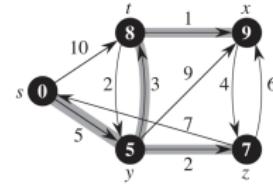
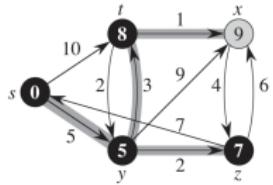
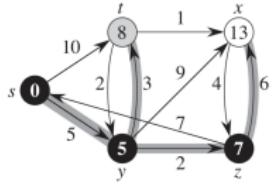
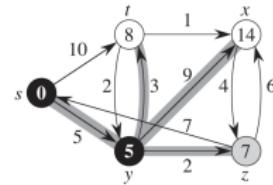
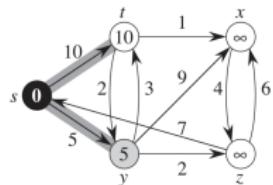
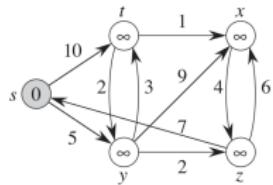
```

d[s] := 0
for each v  $\in V \setminus \{s\}$ 
    d[v] := infinity
S :=  $\emptyset$ 
Q := V      // min-priority queue maintaining  $V \setminus S$ .
while Q !=  $\emptyset$ 
    u := Extract-Min(Q)
    S := S  $\cup \{u\}$ 
    for each v  $\in \text{Adj}[u]$ 
        if d[v] > d[u] + w(u,v)      // *****
        then d[v] := d[u] + w(u,v)   // Relaxation
            pi[v] := u                // *****

```



## Example Dijkstra's Algorithm



```

while Q != ∅
    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)
            pi[v] := u
    
```

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

## Correctness of Dijkstra's Algorithm

Correctness can be 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  $u$ ,
- (iii) if  $(u, v)$  on shortest path (for all  $v$ ) algorithm terminates with  $d[v] = \delta(s, v)$

## Correctness (i)

**Lemma:**

- ▶ Initializing  $d[s] = 0$  and  $d[v] = \infty$  for all  $v \in V \setminus \{s\}$  establishes  $d[v] \geq \delta(s, v)$  for all  $v \in V$ .
- ▶ 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  $d[v] < \delta(s, v)$  and let  $u$  be the vertex that caused  $d[v]$  to change by  $d[v] = d[u] + w(u, v)$ . Then,

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

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  $(u, v)$ .

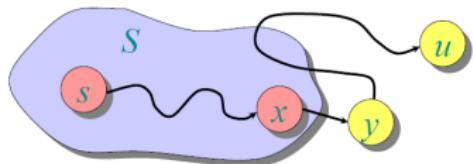
**Proof:**

- ▶ 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  $d[v] = \delta(s, v)$  for all  $v \in V$ .



### Proof:

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

## Complexity Analysis

$|V|$  times {  
   $\text{degree}(u)$  times {  
    **while**  $Q \neq \emptyset$   
     **do**  $u \leftarrow \text{EXTRACT-MIN}(Q)$   
      $S \leftarrow S \cup \{u\}$   
     **for each**  $v \in \text{Adj}[u]$   
        **do if**  $d[v] > d[u] + w(u, v)$   
          **then**  $d[v] \leftarrow d[u] + w(u, v)$

- ▶ Similar to 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] := infinity
Enqueue(Q, s)
while Q != ∅
    u := Dequeue(Q)
    for each v ∈ Adj[u]
        if d[v] = infinity
            then d[v] := d[u] + 1
                pi[v] := u
                Enqueue(Q, v)
```

## Analysis: BFS Algorithm

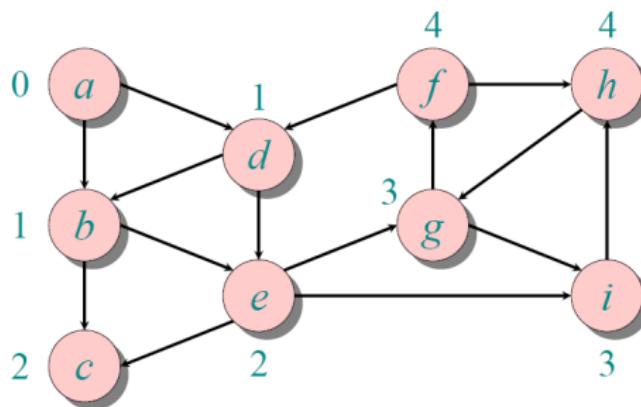
### Correctness:

- ▶ 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:

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

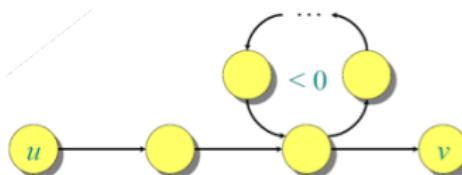
## Example: BFS Algorithm



$Q: a \ b \ d \ c \ e \ g \ i \ f \ h$

## 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  $s \in V$  to all vertices  $v \in V$  or determine existence of a negative-weight cycle.

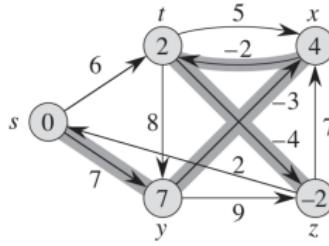
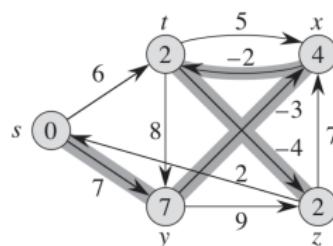
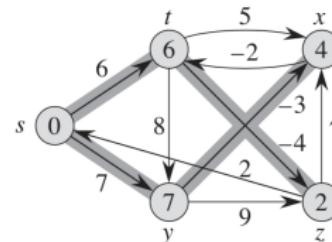
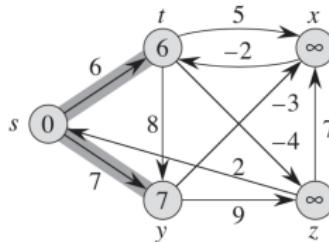
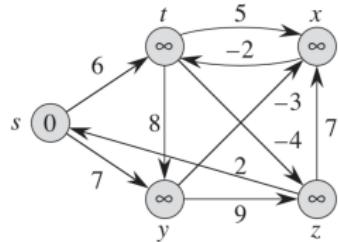
## Bellmann-Ford Algorithm

```
d[s] := 0
for each v ∈ V\{s}
    d[v] := infinity
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)
                  pi[v] := u

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

Time complexity:  $O(|V| \cdot |E|)$

## 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)
            pi[v] := u
    
```

## Bellmann-Ford Algorithm: Correctness (1)

### Theorem:

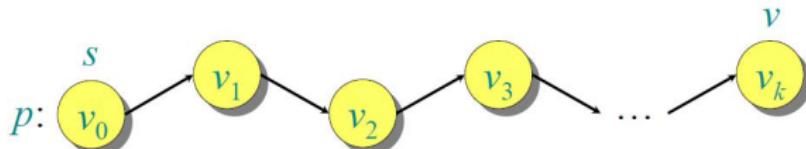
If  $G = (V, E)$  contains no negative-weight cycles, then the Bellman-Ford algorithm terminates with  $d[v] = \delta(s, v)$  for all  $v \in V$ .

### Proof:

Let  $v \in V$  be any vertex.

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

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



## Bellmann-Ford Algorithm: Correctness (2)

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

According to our Lemma from Dijkstra's algorithm we have

$d[v] \geq \delta(s, v)$ , i.e.,  $d[v_0]$  is not changed.

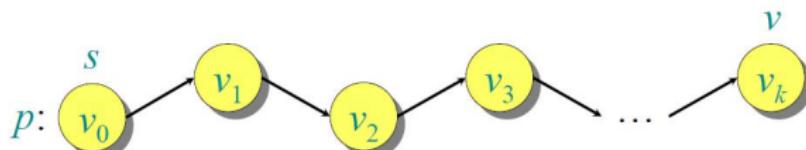
After the 1<sup>st</sup> pass, we have  $d[v_1] = \delta(s, v_1)$ .

After the 2<sup>nd</sup> pass, we have  $d[v_2] = \delta(s, v_2)$ .

...

After the  $k^{\text{th}}$  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:

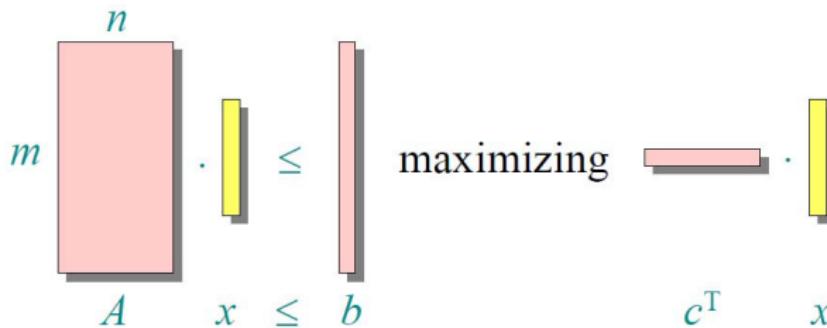
If a value  $d[v]$  fails to converge after  $|V| - 1$  passes, there exists a negative-weight cycle in  $G$  reachable from  $s$ .

## Excuse: Linear Programming

Linear programming problem:

Let  $A$  be matrix of size  $m \times n$ ,  $b$  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  $Ax \leq b$ , or determine that no such solution exists.

$$\begin{matrix} m \\ A \end{matrix} \cdot \begin{matrix} n \\ x \end{matrix} \leq \begin{matrix} b \\ \leq \end{matrix} \quad \text{maximizing} \quad \begin{matrix} c^T \\ \cdot \end{matrix} \begin{matrix} x \\ \end{matrix}$$


## Example: Difference Constraints

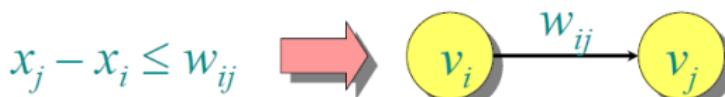
Linear programming example, where each row of  $A$  contains exactly one 1 and one  $-1$ , other entries are 0.

$$\begin{array}{l} x_1 - x_2 \leq 3 \\ x_2 - x_3 \leq -2 \\ x_1 - x_3 \leq 2 \end{array} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} x_j - x_i \leq w_{ij}$$

**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  $|E| \times |V|$ ):



## 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:

$$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow v_1.$$

Then,

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

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

Hence, no  $x$  exists that satisfies the inequations.

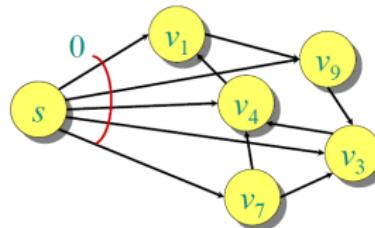
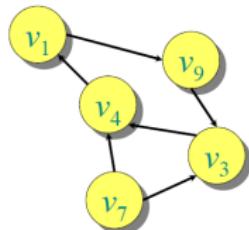
## Case 2: Satisfiable Constraints (1)

### 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 (2)

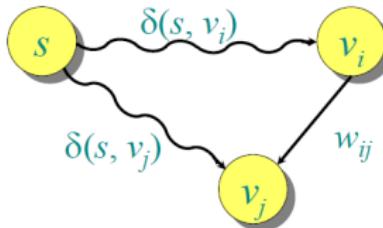
Show that the assignments  $x_i = \delta(s, v_i)$  for  $i = 1, \dots, n$  solve the constraints.

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.

## 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  $\Theta(|V||E|)$