# Graph Algorithms: Shortest Paths

**Zbyněk "Pedro" Křivka**
krivka@fit.vutbr.cz
Brno University of Technology
Faculty of Information Technology
Czech Republic

Lecture at Escuela de Ingeniería Informática de Segovia,
Universidad de Valladolid - Campus de Segovia, Spain
March 16, 2017

# Brno University of Technology

## Faculty of Information Technology

- $\approx$ 2 000 students (Bc, MSc, PhD)
- $\approx$ 50 Erasmus students each semester
- Erasmus+ Agreement with your faculty
- `http://www.fit.vutbr.cz/admissions/short.php.en`

# Outline (with hyperlinks)

Introduction
- Graph Theory
- Graph Representation

Single-Source Shortest Paths
- Bellman-Ford Algorithm
- Dijkstra Algorithm

All-Pairs Shortest Paths

# References

## Books


- Cormen, Leiserson, Rivest, Stein: *Introduction to algorithms.* The MIT Press and McGraw-Hill, 2001.
- Jiří Demel: *Grafy a jejich aplikace [in Czech].* Academia, 2002.

# Introduction

# Graph Theory

# Definitions

Directed graph (digraph) $G$ is a pair

$$G = (V, E),$$

where

- $V$ is a finite set of vertices (nodes) and
- $E \subseteq V^2$ is a set of edges (arrows, arcs).

An edge $(u, u)$ is called a self-loop.
If $(u, v)$ is an edge, we say that $(u, v)$ is incident from $u$ and incident to $v$, that is $v$ is adjacent to $u$.

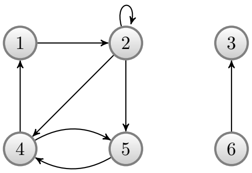



Figure: Digraph

# Definitions

Undirected graph $G$ is a pair

$$G = (V, E),$$

where

- $V$ is a finite set of vertices and
- $E \subseteq \binom{V}{2}$ is a set of edges.

## Note

An edge is a set $\{u, v\}$, where $u, v \in V$ and $u \neq v$. Self-loops are forbidden.

Convention: $\{u, v\}$, $(u, v)$, and $(v, u)$ denote the same edge.

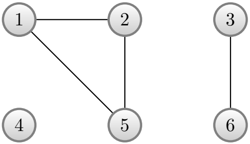



Figure: Undirected Graph

# Definitions

- A path $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a connected sequence of vertices where $(v_{i-1}, v_i) \in E$ for all $i = 1, 2, \ldots, k$.

# Definitions

- A path $p = \langle v_0, v_1, v_2, \dots, v_k \rangle$ is a connected sequence of vertices where $(v_{i-1}, v_i) \in E$ for all $i = 1, 2, \dots, k$.
- The length of $p$ equals to the number of edges in $p$.

# Definitions

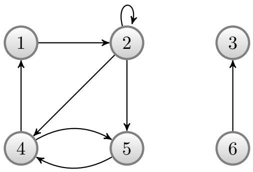
- A path $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a connected sequence of vertices where $(v_{i-1}, v_i) \in E$ for all $i = 1, 2, \ldots, k$.
- The length of $p$ equals to the number of edges in $p$.
- If there is $p$ from $u$ to $u'$, we say that $u'$ is reachable from $u$ by $p$, denoted as $u \overset{p}{\rightsquigarrow} u'$.

# Definitions

- A path $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a connected sequence of vertices where $(v_{i-1}, v_i) \in E$ for all $i = 1, 2, \ldots, k$.
- The length of $p$ equals to the number of edges in $p$.
- If there is $p$ from $u$ to $u'$, we say that $u'$ is reachable from $u$ by $p$, denoted as $u \overset{p}{\leadsto} u'$.
- A path is simple if all vertices in the path are distinct.

# Definitions

- A path $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a connected sequence of vertices where $(v_{i-1}, v_i) \in E$ for all $i = 1, 2, \ldots, k$.
- The length of $p$ equals to the number of edges in $p$.
- If there is $p$ from $u$ to $u'$, we say that $u'$ is reachable from $u$ by $p$, denoted as $u \overset{p}{\rightsquigarrow} u'$.
- A path is simple if all vertices in the path are distinct.




- Give some examples of a path and simple path.
- Give an example of unconnected sequence.

# Definitions

- A subpath $s$ of $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a contiguous subsequence, $s = \langle v_i, v_{i+1}, v_{i+2}, \ldots, v_j \rangle$, for $0 \leq i \leq j \leq k$.

# Definitions

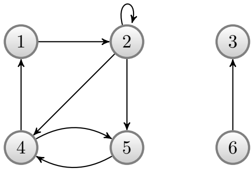
- A subpath $s$ of $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a contiguous subsequence, $s = \langle v_i, v_{i+1}, v_{i+2}, \ldots, v_j \rangle$, for $0 \leq i \leq j \leq k$.

- A path $c = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a cycle, if $k \geq 1$ and $v_0 = v_k$.

# Definitions

- A subpath $s$ of $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a contiguous subsequence, $s = \langle v_i, v_{i+1}, v_{i+2}, \ldots, v_j \rangle$, for $0 \le i \le j \le k$.

- A path $c = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a cycle, if $k \ge 1$ and $v_0 = v_k$.

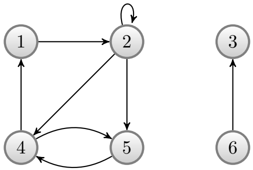



- What is $\langle 1, 2, 4, 5, 4, 1 \rangle$?
- What is $\langle 1, 2, 4, 1 \rangle$?
- What is $\langle 2, 2 \rangle$?

# Definitions

- A subpath $s$ of $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a contiguous subsequence, $s = \langle v_i, v_{i+1}, v_{i+2}, \ldots, v_j \rangle$, for $0 \leq i \leq j \leq k$.

- A path $c = \langle v_0, v_1, v_2, \ldots, v_k \rangle$ is a cycle, if $k \geq 1$ and $v_0 = v_k$.




- What is $\langle 1, 2, 4, 5, 4, 1 \rangle$?
- What is $\langle 1, 2, 4, 1 \rangle$?
- What is $\langle 2, 2 \rangle$?

- Acyclic graph contains no cycles.

# Definitions

- A graph $G' = (V', E')$ is a subgraph of $G$, if $V' \subseteq V$ and $E' \subseteq E$.

# Definitions

- A graph $G' = (V', E')$ is a subgraph of $G$, if $V' \subseteq V$ and $E' \subseteq E$.
- An undirected graph is connected if every pair of vertices is connected by a path.
- An connected, acyclic, undirected graph is a tree.
  - Homework: Prove that $|E| = |V| - 1$.
- An acyclic, undirected graph is a forest (several trees).

# Graph Representation

Let $G = (V, E)$ be a graph. Denote:

- $n = |V|$
- $m = |E|$.

1. Adjacency-list representation
   - effective for sparse graphs ($m \ll n^2$);

Let $G = (V, E)$ be a graph. Denote:

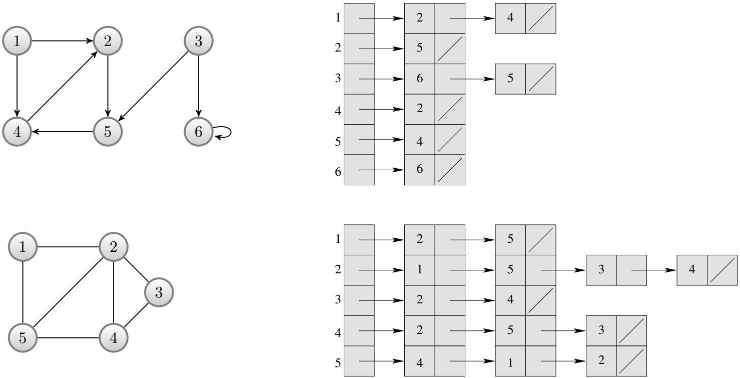
- $n = |V|$
- $m = |E|$.

1. Adjacency-list representation
   - effective for sparse graphs ($m \ll n^2$);
2. Adjacency-matrix representation
   - effective for dense graphs ($m$ close to $n^2$);
   - when we often need quick answer whether two given vertices are connected by an edge.

# Adjacency-list representation

$G = (V, E)$ is represented as

- an array $Adj[1 \ldots n]$ with $n$ lists, one (unsorted) list for each vertex,
- where $Adj[u]$ stores all vertices $v$ such that $(u, v) \in E$.



- Space complexity: $\Theta(m + n)$ (depends linearly on the size of the graph).

# Weighted graph

- A weighted graph is a (di)graph where there is a value assigned to every edge using weight function $w : E \to \mathbb{R}$.

# Weighted graph

- A weighted graph is a (di)graph where there is a value assigned to every edge using weight function $w : E \to \mathbb{R}$.
- Representation of $w(u, v)$ in adjacency list: extend the list item (a structure) for $v$ in $Adj[u]$ with value $w(u, v)$.

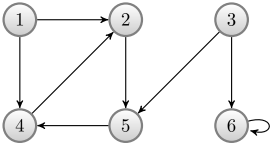
# Weighted graph

- A weighted graph is a (di)graph where there is a value assigned to every edge using weight function $w : E \to \mathbb{R}$.
- Representation of $w(u, v)$ in adjacency list: extend the list item (a structure) for $v$ in $Adj[u]$ with value $w(u, v)$.
- Disadvantage: Finding whether an edge $(u, v)$ belongs to $E$ requires the search of the whole list $Adj[u]$.

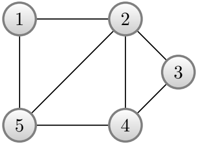
# Adjacency-matrix representation

Let $G = (V, E)$ be a graph and assume $V = \{1, 2, \ldots, n\}$. Adjacency matrix $A = (a_{ij})$ is a matrix of size $n \times n$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

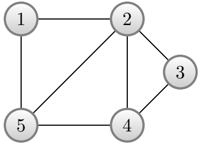



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 |



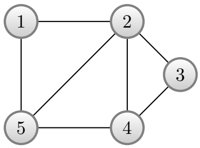

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

- Space complexity: $\Theta(n^2)$ (independent of the number of edges).



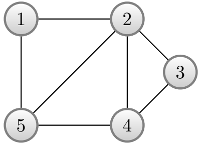

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

- Space complexity: $\Theta(n^2)$ (independent of the number of edges).
- Transpose matrix of $A = (a_{ij})$ is a matrix $A^T = (a_{ij}^T)$, where $a_{ij}^T = a_{ji}$.




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

- Space complexity: $\Theta(n^2)$ (independent of the number of edges).
- Transpose matrix of $A = (a_{ij})$ is a matrix $A^T = (a_{ij}^T)$, where $a_{ij}^T = a_{ji}$.
- If $A$ represents an undirected graph, then $A = A^T$. It is enough to store just one half of $A$.



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

- Space complexity: $\Theta(n^2)$ (independent of the number of edges).
- Transpose matrix of $A = (a_{ij})$ is a matrix $A^T = (a_{ij}^T)$, where $a_{ij}^T = a_{ji}$.
- If $A$ represents an undirected graph, then $A = A^T$. It is enough to store just one half of $A$.
- Let $G = (V, E)$ be a weighted graph, then

$$a_{ij} = \begin{cases} w(i,j) & \text{if } (i,j) \in E, \\ \text{NIL} & \text{otherwise,} \end{cases}$$

where NIL is a special value, mostly $0$ or $\infty$.

# Exercises

1. Let $deg_{-}(u)$ and $deg_{+}(u)$ be the number of outcoming edges from $u$ and incoming edges to $u$, respectively. Given an adjacency-list representation of a digraph and a vertex $v$, how long does it take to compute degrees $deg_{-}(v)$ and $deg_{+}(v)$?
2. The transpose of a directed graph $G = (V, E)$ is the graph $G^T = (V, E^T)$, where $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. Thus, $G^T$ is $G$ with all its edges reversed. Describe an efficient algorithm for computing $G^T$ from $G$ for the adjacency-list representation of $G$. Analyze the time complexity of your algorithm.
3. The square of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E^2)$ such that $(u, v) \in E^2$ if and only $G$ contains a path with at most two edges between $u$ and $v$. Describe an efficient algorithm for computing $G^2$ from $G$ for the adjacency-list representation of $G$. Analyze the time complexity of your algorithm.

# Single-Source Shortest Paths

# Shortest Paths – Motivation

- Transportation: How to get from $A$ into $B$ in the quickest/cheapest way?
- Optimization: cost minimization in static state space (e.g. knapsack problem, ...)

# Shortest Paths – Motivation

- Transportation: How to get from $A$ into $B$ in the quickest/cheapest way?
- Optimization: cost minimization in static state space (e.g. knapsack problem, ...)

## Measuring-cup Problem

- We have a 1-litre cup and a 3-litre cup. We can fill a cup and we can pour from one cup to another as much as possible without spilling.
- How to measure 2 litres? How to do it in the cheapest way, if each liter is paid?

# Shortest Paths – Motivation

- Transportation: How to get from $A$ into $B$ in the quickest/cheapest way?
- Optimization: cost minimization in static state space (e.g. knapsack problem, . . . )

## Measuring-cup Problem

- We have a 1-litre cup and a 3-litre cup. We can fill a cup and we can pour from one cup to another as much as possible without spilling.
- How to measure 2 litres? How to do it in the cheapest way, if each liter is paid?

- What if I have 3-litre and 5-litre cup and I need to measure 4 litres? Is it possible?

# Shortest Paths

- Given weighted directed graph $G = (V, E)$ and
- weight function $w : E \to \mathbb{R}$.

# Shortest Paths

- Given weighted directed graph $G = (V, E)$ and
- weight function $w : E \to \mathbb{R}$.

- The weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is

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

# Shortest Paths

- Given weighted directed graph $G = (V, E)$ and
- weight function $w : E \to \mathbb{R}$.

- The weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is

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

- The shortest-path weight from $u$ to $v$ is

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

- A shortest path from $u$ to $v$ is any path $p$ from $u$ to $v$ with $w(p) = \delta(u, v)$.

# Shortest Paths – Variants

- Single-source shortest-paths problem
- Single-destination shortest-paths problem – by reversing the direction of each edge
- Single-pair shortest-path problem – is there faster solution?
- All-pairs shortest-paths problem – single-source from each vertex or faster?

# Subpaths of Shortest Paths

**Lemma 1.**
*Let* $G = (V, E)$ *be directed graph with weight function* $w : E \to \mathbb{R}$*. Let* $p = \langle v_1, v_2, \dots, v_k \rangle$ *be a shortest path from* $v_1$ *to* $v_k$*.*
*For any* $1 \le i \le j \le k$*, let* $p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle$ *be the subpath of* $p$ *from* $v_i$ *to* $v_j$*.*
*Then,* $p_{ij}$ *is a shortest path from* $v_i$ *to* $v_j$*.*

Proof.

□

# Subpaths of Shortest Paths

**Lemma 1.**
*Let $G = (V, E)$ be directed graph with weight function $w : E \to \mathbb{R}$. Let $p = \langle v_1, v_2, \ldots, v_k \rangle$ be a shortest path from $v_1$ to $v_k$.*
*For any $1 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of $p$ from $v_i$ to $v_j$.*
*Then, $p_{ij}$ is a shortest path from $v_i$ to $v_j$.*

Proof.

- $p$ is $v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$, where $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$.

$\square$

# Subpaths of Shortest Paths

**Lemma 1.**

*Let $G = (V, E)$ be directed graph with weight function $w : E \to \mathbb{R}$. Let $p = \langle v_1, v_2, \ldots, v_k \rangle$ be a shortest path from $v_1$ to $v_k$.*
*For any $1 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of $p$ from $v_i$ to $v_j$.*
*Then, $p_{ij}$ is a shortest path from $v_i$ to $v_j$.*

Proof.

- $p$ is $v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$, where $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$.
- Assume that there is $p'_{ij}$ from $v_i$ to $v_j$ with $w(p'_{ij}) < w(p_{ij})$.

$\square$

# Subpaths of Shortest Paths

**Lemma 1.**
*Let $G = (V, E)$ be directed graph with weight function $w : E \to \mathbb{R}$. Let $p = \langle v_1, v_2, \ldots, v_k \rangle$ be a shortest path from $v_1$ to $v_k$.*
*For any $1 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of $p$ from $v_i$ to $v_j$.*
*Then, $p_{ij}$ is a shortest path from $v_i$ to $v_j$.*

Proof.

- $p$ is $v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$, where $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$.
- Assume that there is $p'_{ij}$ from $v_i$ to $v_j$ with $w(p'_{ij}) < w(p_{ij})$.
- Then, $v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p'_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$, where $w(p_{1i}) + w(p'_{ij}) + w(p_{jk}) < w(p)$. Contradiction.

$\square$

# Negative-weight edges

- If $G$ contains no negative-weight cycles reachable from the source $s$, then for all $v \in V$, $\delta(s, v)$ remains well defined (even if negative).

# Negative-weight edges

- If $G$ contains no negative-weight cycles reachable from the source $s$, then for all $v \in V$, $\delta(s, v)$ remains well defined (even if negative).
- If $G$ contains a negative-weight cycle reachable from $s$, $\delta$ is not well defined – repeating traverse of the negative-weight cycle.
- If there is negative-weight cycle on some path from $s$ to $v$, we define $\delta(s, v) = -\infty$.

# Negative-weight edges

- If $G$ contains no negative-weight cycles reachable from the source $s$, then for all $v \in V$, $\delta(s, v)$ remains well defined (even if negative).

- If $G$ contains a negative-weight cycle reachable from $s$, $\delta$ is not well defined – repeating traverse of the negative-weight cycle.
- If there is negative-weight cycle on some path from $s$ to $v$, we define $\delta(s, v) = -\infty$.

- Note: There is always the shortest simple path, but not path. The algorithms work with paths $\Rightarrow$ problem.

# Representing Shortest Paths

- Let $G = (V, E)$ be a graph.
- $\pi[v]$ is set to a predecessor to $v$; that is, a vertex or $\mathrm{NIL}$.
- If $\pi[v] = u \neq \mathrm{NIL}$, then $(u, v) \in E$ is highlighted in the graph drawing.

# Representing Shortest Paths

- Let $G = (V, E)$ be a graph.
- $\pi[v]$ is set to a predecessor to $v$; that is, a vertex or $\textsc{nil}$.
- If $\pi[v] = u \neq \textsc{nil}$, then $(u, v) \in E$ is highlighted in the graph drawing.
- Predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ induced by $\pi$
  - $V_\pi = \{v \in V : \pi[v] \neq \textsc{nil}\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$

# Representing Shortest Paths

- Let $G = (V, E)$ be a graph.
- $\pi[v]$ is set to a predecessor to $v$; that is, a vertex or $\text{NIL}$.
- If $\pi[v] = u \neq \text{NIL}$, then $(u, v) \in E$ is highlighted in the graph drawing.
- Predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ induced by $\pi$
  - $V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$
- After the algorithm is finished, $G_\pi$ is a shortest-paths tree rooted at $s$ containing shortest paths from $s$ to all other reachable vertices.

# Representing Shortest Paths

- Let $G = (V, E)$ be a graph.
- $\pi[v]$ is set to a predecessor to $v$; that is, a vertex or NIL.
- If $\pi[v] = u \neq \text{NIL}$, then $(u, v) \in E$ is highlighted in the graph drawing.
- Predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ induced by $\pi$
  - $V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$
- After the algorithm is finished, $G_\pi$ is a shortest-paths tree rooted at $s$ containing shortest paths from $s$ to all other reachable vertices.

```
PRINT-PATH(G, s, v)
if v = s
   then print s
   else if π[v] = NIL
           then print "No path from " s " to " v "!"
           else PRINT-PATH(G, s, π[v])
                print v
```

# Shortest paths are not necessarily unique – Example

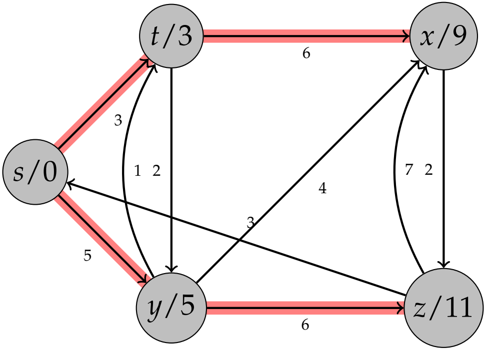



Figure: Shortest paths.

# Shortest paths are not necessarily unique – Example

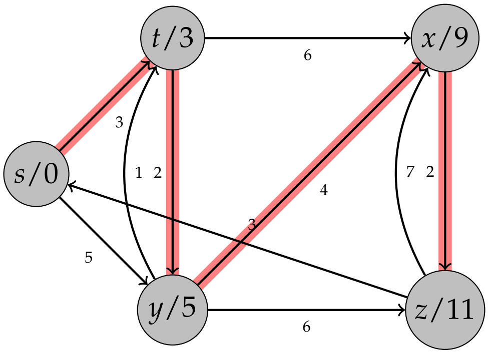



Figure: Shortest paths.

# Relaxation

- $d[v]$ – shortest-path estimate (upper bound of weight)

INITIALIZE-SINGLE-SOURCE$(G, s)$
**for** each vertex $v \in V$
  **do** $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

- Time complexity: $\Theta(n)$.

# Relaxation

- $d[v]$ – shortest-path estimate (upper bound of weight)

INITIALIZE-SINGLE-SOURCE$(G, s)$
**for** each vertex $v \in V$
    **do** $d[v] \leftarrow \infty$
        $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

- Time complexity: $\Theta(n)$.

RELAX$(u, v, w)$
**if** $d[v] > d[u] + w(u, v)$
    **then** $d[v] \leftarrow d[u] + w(u, v)$
        $\pi[v] \leftarrow u$

# Bellman-Ford Algorithm

# Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to n − 1
    do for each edge (u, v) ∈ E
           do RELAX(u, v, w)
for each edge (u, v) ∈ E
    do if d[v] > d[u] + w(u, v)
          then return FALSE
return TRUE
```

- If it returns FALSE, $G$ contains negative-weight cycles reachable from $s$.
- If it returns TRUE, $\pi$ contains the shortest paths.

# Bellman-Ford – Example

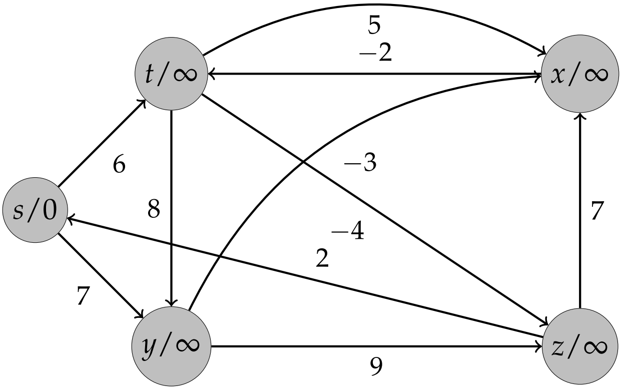



Figure: Computation by Bellman-Ford Algorithm.

- Edges are relaxed in the following order: $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

# Bellman-Ford – Example

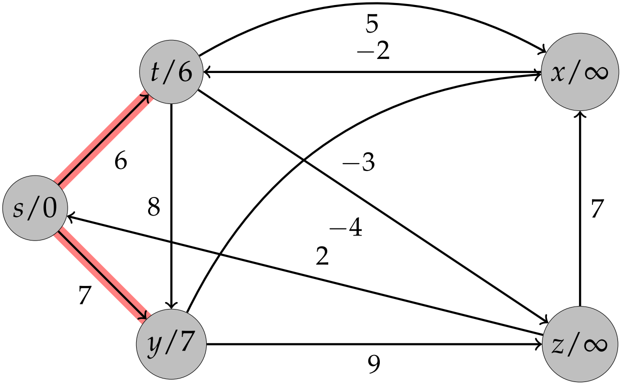



Figure: Computation by Bellman-Ford Algorithm.

- Edges are relaxed in the following order: $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

# Bellman-Ford – Example

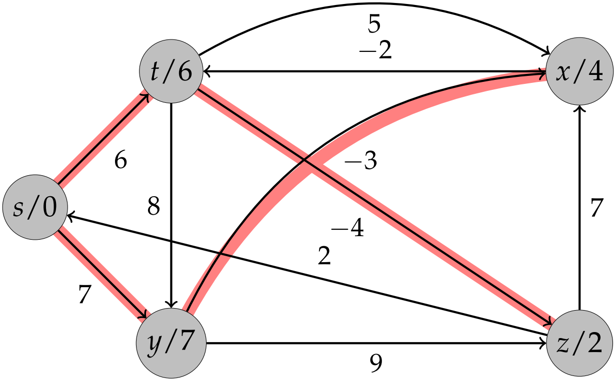



Figure: Computation by Bellman-Ford Algorithm.

- Edges are relaxed in the following order: $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

# Bellman-Ford – Example

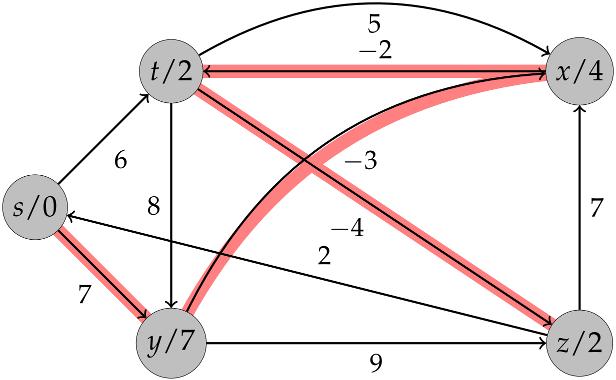



Figure: Computation by Bellman-Ford Algorithm.

- Edges are relaxed in the following order: $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

# Bellman-Ford – Example

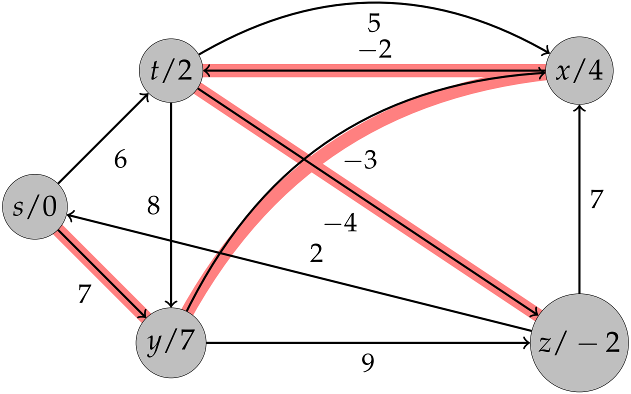



Figure: Computation by Bellman-Ford Algorithm.

- Edges are relaxed in the following order: $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

# Bellman-Ford Algorithm – Time Complexity

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to n − 1
3       do for each edge (u, v) ∈ E
4              do RELAX(u, v, w)
5  for each edge (u, v) ∈ E
6       do if d[v] > d[u] + w(u, v)
7             then return FALSE
8  return TRUE
```

- Line 1 takes $\Theta(n)$.

# Bellman-Ford Algorithm – Time Complexity

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to n − 1
3       do for each edge (u, v) ∈ E
4              do RELAX(u, v, w)
5  for each edge (u, v) ∈ E
6       do if d[v] > d[u] + w(u, v)
7            then return FALSE
8  return TRUE
```

- Line 1 takes $\Theta(n)$.
- Lines 2-4 take $(n-1)$-times $\Theta(m)$.

# Bellman-Ford Algorithm – Time Complexity

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to n − 1
3       do for each edge (u, v) ∈ E
4              do RELAX(u, v, w)
5  for each edge (u, v) ∈ E
6       do if d[v] > d[u] + w(u, v)
7             then return FALSE
8  return TRUE
```

- Line 1 takes $\Theta(n)$.
- Lines 2-4 take $(n-1)$-times $\Theta(m)$.
- Lines 5-7 take $O(m)$.

# Bellman-Ford Algorithm – Time Complexity

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to n − 1
3       do for each edge (u, v) ∈ E
4              do RELAX(u, v, w)
5  for each edge (u, v) ∈ E
6       do if d[v] > d[u] + w(u, v)
7             then return FALSE
8  return TRUE
```

- Line 1 takes $\Theta(n)$.
- Lines 2-4 take $(n-1)$-times $\Theta(m)$.
- Lines 5-7 take $O(m)$.
- In total, $\Theta(mn)$.

# Dijkstra Algorithm

# Dijkstra Algorithm

- Only for weighted, directed graphs without negative edges:
- $w(u, v) \geq 0$ for each edge $(u, v) \in E$.

# Dijkstra Algorithm

- Only for weighted, directed graphs without negative edges:
- $w(u,v) \geq 0$ for each edge $(u,v) \in E$.

- Can we implement it with lower time complexity than Bellman-Ford algorithm?

# Dijkstra Algorithm

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S ← ∅
Q ← V
while Q ≠ ∅
      do u ← EXTRACT-MIN(Q)
         S ← S ∪ {u}
         for each vertex v ∈ Adj[u]
             do RELAX(u, v, w)
```

- $S$ is a set of finished vertices (their shortest distance from $s$ is already computed).
- $Q$ is a min-priority queue; the vertex with the lowest $d$-value is at the beginning of $Q$.

# Dijkstra Algorithm – Example

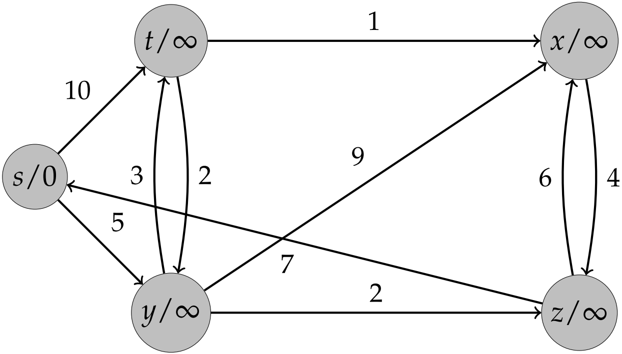



Figure: The computation by Dijkstra Algorithm. Highlighted vertices belong to set $S$.

# Dijkstra Algorithm – Example

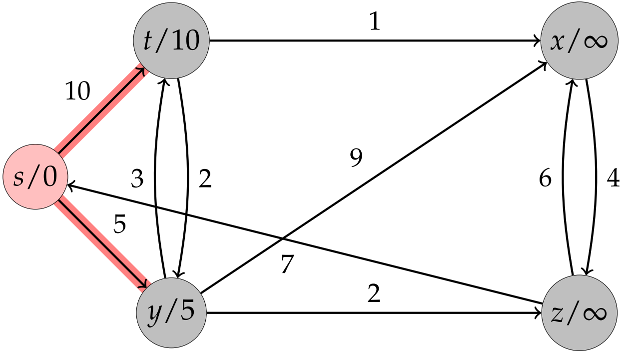



Figure: The computation by Dijkstra Algorithm. Highlighted vertices belong to set $S$.

# Dijkstra Algorithm – Example

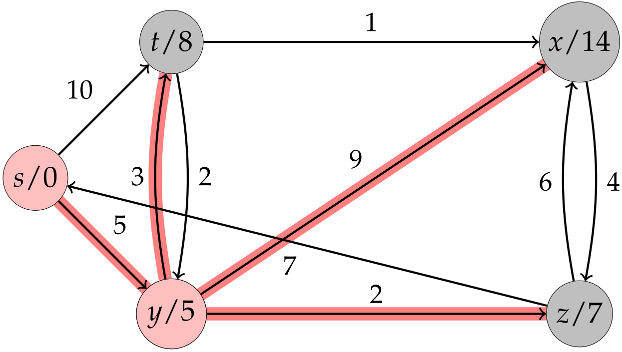



Figure: The computation by Dijkstra Algorithm. Highlighted vertices belong to set $S$.

# Dijkstra Algorithm – Example

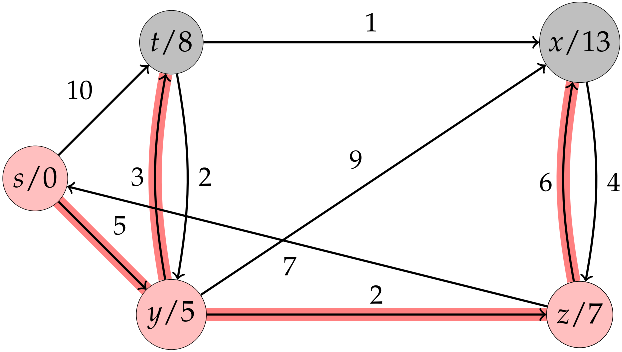



Figure: The computation by Dijkstra Algorithm. Highlighted vertices belong to set $S$.

# Dijkstra Algorithm – Example

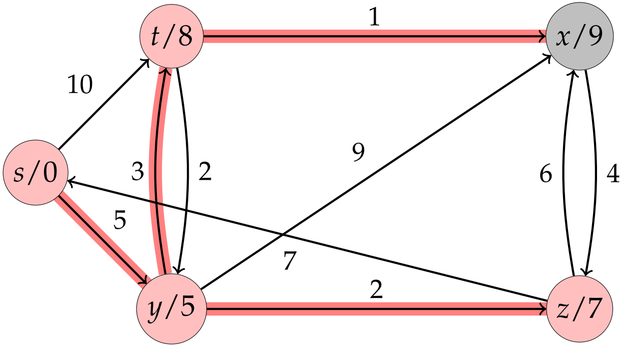



Figure: The computation by Dijkstra Algorithm. Highlighted vertices belong to set $S$.

# Dijkstra Algorithm – Example

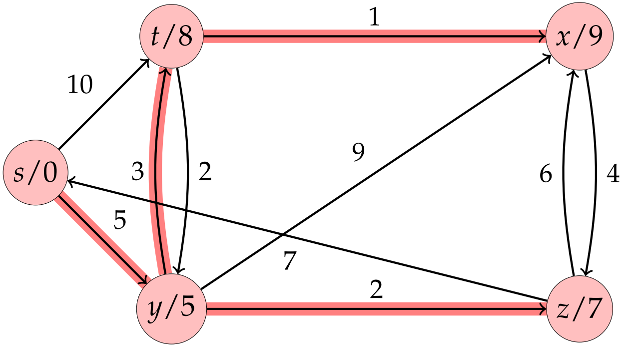



Figure: The computation by Dijkstra Algorithm. Highlighted vertices belong to set $S$.

# Time Complexity of Dijkstra algorithm

## Min-Priority Queue Implemented by Array

- INSERT and DECREASE-KEY take $O(1)$.
- EXTRACT-MIN takes $O(n)$ for each vertex (line 5).

# Time Complexity of Dijkstra algorithm

## Min-Priority Queue Implemented by Array

- INSERT and DECREASE-KEY take $O(1)$.
- EXTRACT-MIN takes $O(n)$ for each vertex (line 5).
- RELAX is repeated $m$-times (line 8).

# Time Complexity of Dijkstra algorithm

## Min-Priority Queue Implemented by Array

- Insert and Decrease-Key take $O(1)$.
- Extract-Min takes $O(n)$ for each vertex (line 5).
- Relax is repeated $m$-times (line 8).

- In total, $O(n^2 + m) = O(n^2)$.

## Min-Priority Queue Implemented by Heaps

# Time Complexity of Dijkstra algorithm

## Min-Priority Queue Implemented by Array

- INSERT and DECREASE-KEY take $O(1)$.
- EXTRACT-MIN takes $O(n)$ for each vertex (line 5).
- RELAX is repeated $m$-times (line 8).

- In total, $O(n^2 + m) = O(n^2)$.

## Min-Priority Queue Implemented by Heaps

- For sparse graphs, we get the time complexity $O(m \log n)$ using binary heap.

# Time Complexity of Dijkstra algorithm

## Min-Priority Queue Implemented by Array

- INSERT and DECREASE-KEY take $O(1)$.
- EXTRACT-MIN takes $O(n)$ for each vertex (line 5).
- RELAX is repeated $m$-times (line 8).

- In total, $O(n^2 + m) = O(n^2)$.

## Min-Priority Queue Implemented by Heaps

- For sparse graphs, we get the time complexity $O(m \log n)$ using binary heap.

- In general, using Fibonacci heap we get the time complexity $O(n \log n + m)$.

# Exercises

1. Modify the Bellman-Ford algorithm so that it sets $d[v]$ to $-\infty$ for all vertices $v$ for which there is a negative-weight cycle on some path from the source $s$ to $v$.
2. Give a simple example of a digraph with negative-weight edge(s) for which Dijkstra's algorithm produces incorrect answers. Why?

# Demonstration Tool

## Graph Simulator

- Application with GUI in Java by Jakub Varadinek and Otto Michalička
- Requirements: Java Runtime Environment 1.7 (32-bit or 64-bit version)
- Language: English, Czech, ?
- Algorithms: Breadth-First Search, Depth-First Search, Topological Sorting, Strongly-connected Components, Bellman-Ford and Dijkstra Algorithms
- Modes: Graph editing, Algorithm Simulation (stepping, breakpoints, variables)
- `http://www.fit.vutbr.cz/~krivka/graphsim`

# All-Pairs Shortest Paths

# Using Single-Source Shortest Paths Algorithms

- Given weighted directed graph $G = (V, E)$ and
- weight function $w : E \to \mathbb{R}$.

# Using Single-Source Shortest Paths Algorithms

- Given weighted directed graph $G = (V, E)$ and
- weight function $w : E \to \mathbb{R}$.

- Find all shortest paths from each vertex to the other vertices (solve $n$-times single-source shortests paths).
- Considering $n$-times Dijkstra algorithm: $O(n^3 + nm) = O(n^3)$ time for array, or $O(n^2 \log n + nm)$ for Fibonacci heap.

# Using Single-Source Shortest Paths Algorithms

- Given weighted directed graph $G = (V, E)$ and
- weight function $w : E \to \mathbb{R}$.

- Find all shortest paths from each vertex to the other vertices (solve $n$-times single-source shortests paths).
- Considering $n$-times Dijkstra algorithm: $O(n^3 + nm) = O(n^3)$ time for array, or $O(n^2 \log n + nm)$ for Fibonacci heap.
- For negative weights of edges, use $n$-times Bellman-Ford: $O(n^2 m)$ time (dense graphs: $O(n^4)$).

# Shortest Path Representation

- Even for sparse graphs, the input is adjacency-matrix with weights $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

# Shortest Path Representation

- Even for sparse graphs, the input is adjacency-matrix with weights $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- We allow negative-weight edges.

# Shortest Path Representation

- Even for sparse graphs, the input is adjacency-matrix with weights $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- We allow negative-weight edges.
- We assume no negative-weight cycles.

# Shortest Path Representation

- Even for sparse graphs, the input is adjacency-matrix with weights $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- We allow negative-weight edges.
- We assume no negative-weight cycles.

- The results stored in $n \times n$ matrix $D = (d_{ij})$, where $d_{ij} = \delta(i,j)$ when the algorithm is finished.

# Shortest Path Representation

- Even for sparse graphs, the input is adjacency-matrix with weights $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- We allow negative-weight edges.
- We assume no negative-weight cycles.

- The results stored in $n \times n$ matrix $D = (d_{ij})$, where $d_{ij} = \delta(i,j)$ when the algorithm is finished.
- Predecessor matrix $\Pi = (\pi_{ij})$, where $\pi_{ij}$ is

# Shortest Path Representation

- Even for sparse graphs, the input is adjacency-matrix with weights $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- We allow negative-weight edges.
- We assume no negative-weight cycles.

- The results stored in $n \times n$ matrix $D = (d_{ij})$, where $d_{ij} = \delta(i,j)$ when the algorithm is finished.
- Predecessor matrix $\Pi = (\pi_{ij})$, where $\pi_{ij}$ is
  1. NIL if $i = j$ or there is no path from $i$ to $j$,

# Shortest Path Representation

- Even for sparse graphs, the input is adjacency-matrix with weights $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- We allow negative-weight edges.
- We assume no negative-weight cycles.

- The results stored in $n \times n$ matrix $D = (d_{ij})$, where $d_{ij} = \delta(i,j)$ when the algorithm is finished.
- Predecessor matrix $\Pi = (\pi_{ij})$, where $\pi_{ij}$ is
  1. NIL if $i = j$ or there is no path from $i$ to $j$,
  2. otherwise the predecessor of $j$ on some shortest path from $i$.

# Printing of Shortest Paths

```
PRINT-ALL-SHORTEST-PATH(Π, i, j)
if i = j
   then print i
   else if π_ij = NIL
           then print "No path from " i " to " j " exists!"
           else PRINT-ALL-SHORTEST-PATH(Π, i, π_ij)
                print j
```

# Shortest Paths and Matrix Multiplication

# Matrix Multiplication – Structure of Shortest Paths

- Representation – adjacency matrix with weights $W = (w_{ij})$.

# Matrix Multiplication – Structure of Shortest Paths

- Representation – adjacency matrix with weights $W = (w_{ij})$.
- Let $p$ be a shortest path from $i$ to $j$ with $m'$ edges.

# Matrix Multiplication – Structure of Shortest Paths

- Representation – adjacency matrix with weights $W = (w_{ij})$.
- Let $p$ be a shortest path from $i$ to $j$ with $m'$ edges.
- If there is no negative-weight cycle in $p$, then $m' < \infty$.

# Matrix Multiplication – Structure of Shortest Paths

- Representation – adjacency matrix with weights $W = (w_{ij})$.
- Let $p$ be a shortest path from $i$ to $j$ with $m'$ edges.
- If there is no negative-weight cycle in $p$, then $m' < \infty$.
- If $i = j$, then $m' = 0$ and $w_{ij} = \delta(i,j) = 0$.

# Matrix Multiplication – Structure of Shortest Paths

- Representation – adjacency matrix with weights $W = (w_{ij})$.
- Let $p$ be a shortest path from $i$ to $j$ with $m'$ edges.
- If there is no negative-weight cycle in $p$, then $m' < \infty$.
- If $i = j$, then $m' = 0$ and $w_{ij} = \delta(i, j) = 0$.
- If $i \neq j$, then we decompose path $p$ into:

$$i \overset{p'}{\leadsto} k \to j \,,$$

where $p'$ has $m' - 1$ edges.

# Matrix Multiplication – Structure of Shortest Paths

- Representation – adjacency matrix with weights $W = (w_{ij})$.
- Let $p$ be a shortest path from $i$ to $j$ with $m'$ edges.
- If there is no negative-weight cycle in $p$, then $m' < \infty$.
- If $i = j$, then $m' = 0$ and $w_{ij} = \delta(i,j) = 0$.
- If $i \neq j$, then we decompose path $p$ into:

$$i \overset{p'}{\leadsto} k \to j\,,$$

  where $p'$ has $m' - 1$ edges.
- Observe that $p'$ is a shortest path from $i$ to $k$, so $\delta(i,j) = \delta(i,k) + w_{kj}$.

# Matrix Multiplication – Recursive Solution

- Let $l_{ij}^{(m)}$ be a minimum weight of any path from $i$ to $j$ with at most $m$ edges.

# Matrix Multiplication – Recursive Solution

- Let $l_{ij}^{(m)}$ be a minimum weight of any path from $i$ to $j$ with at most $m$ edges.
- $m = 0$, if and only if $i = j$. Thus, $l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$

# Matrix Multiplication – Recursive Solution

- Let $l_{ij}^{(m)}$ be a minimum weight of any path from $i$ to $j$ with at most $m$ edges.
- $m = 0$, if and only if $i = j$. Thus, $l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$
- $l_{ij}^{(m)} = \min(l_{ij}^{(m-1)}, \min_{1 \le k \le n} \{l_{ik}^{(m-1)} + w_{kj}\}) = \min_{1 \le k \le n} \{l_{ik}^{(m-1)} + w_{kj}\}$.

# Matrix Multiplication – Recursive Solution

- Let $l_{ij}^{(m)}$ be a minimum weight of any path from $i$ to $j$ with at most $m$ edges.
- $m = 0$, if and only if $i = j$. Thus, $l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$

- $l_{ij}^{(m)} = \min(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\}) = \min_{1 \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\}$.

- Observe that a shortest path from $i$ to $j$ has at most $n - 1$ edges, so

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \ldots.$$

(If there is no negative-weight cycle.)

# Matrix Multiplication – Computing

- Input matrix $W = (w_{ij})$.

# Matrix Multiplication – Computing

- Input matrix $W = (w_{ij})$.
- We compute a series of matrices $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$, where for $m = 1, 2, \ldots, n-1$,
$$L^{(m)} = (l_{ij}^{(m)}) .$$

# Matrix Multiplication – Computing

- Input matrix $W = (w_{ij})$.
- We compute a series of matrices $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$, where for $m = 1, 2, \ldots, n-1$,
$$L^{(m)} = (l_{ij}^{(m)}) \,.$$
- $L^{(n-1)}$ contains the actual shortest-path weights.

# Matrix Multiplication – Computing

- Input matrix $W = (w_{ij})$.
- We compute a series of matrices $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$, where for $m = 1, 2, \ldots, n-1$,
$$L^{(m)} = (l_{ij}^{(m)}) .$$
- $L^{(n-1)}$ contains the actual shortest-path weights.
- Observe that $l_{ij}^{(1)} = w_{ij}$; that is, $L^{(1)} = W$.

# The Heart of All-Pairs Shortest Paths Algorithm

EXTEND-SHORTEST-PATHS$(L, W)$

$n \leftarrow rows[L]$<br>
let $L' = (l'_{ij})$ be an $n \times n$ matrix<br>
**for** $i \leftarrow 1$ **to** $n$<br>
    **do for** $j \leftarrow 1$ **to** $n$<br>
        **do** $l'_{ij} \leftarrow \infty$<br>
            **for** $k \leftarrow 1$ **to** $n$<br>
                **do** $l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})$<br>
**return** $L'$

- $rows[L]$ denotes the number of rows of $L$.
- Time complexity: $\Theta(n^3)$.

# The relation to matrix multiplication (finally)

- Let $C = A \cdot B$, where $A$ and $B$ are $n \times n$ matrices.

# The relation to matrix multiplication (finally)

- Let $C = A \cdot B$, where $A$ and $B$ are $n \times n$ matrices.
- Then,

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

# The relation to matrix multiplication (finally)

- Let $C = A \cdot B$, where $A$ and $B$ are $n \times n$ matrices.
- Then,

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

- Compare to

$$l_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \}$$

# Find 3 differences (apart from algorithm/variable renaming)

EXTEND-SHORTEST-PATHS$(L, W)$

$n \leftarrow rows[L]$<br>
let $L' = (l'_{ij})$ be an $n \times n$ matrix<br>
**for** $i \leftarrow 1$ **to** $n$<br>
    **do for** $j \leftarrow 1$ **to** $n$<br>
        **do** $l'_{ij} \leftarrow \infty$<br>
           **for** $k \leftarrow 1$ **to** $n$<br>
               **do** $l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})$<br>
**return** $L'$

MATRIX-MULTIPLY$(A, B)$

$n \leftarrow rows[A]$<br>
let $C = (c_{ij})$ be an $n \times n$ matrix<br>
**for** $i \leftarrow 1$ **to** $n$<br>
    **do for** $j \leftarrow 1$ **to** $n$<br>
        **do** $c_{ij} \leftarrow 0$<br>
           **for** $k \leftarrow 1$ **to** $n$<br>
               **do** $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$<br>
**return** $C$

# Matrix Multiplication – Notation

- Letting $X \cdot Y$ denote the matrix computed by $\textsc{Extend-Shortest-Paths}(X, Y)$.

# Matrix Multiplication – Notation

- Letting $X \cdot Y$ denote the matrix computed by $\textsc{Extend-Shortest-Paths}(X, Y)$.
- Then, we compute the following matrices

$$
\begin{array}{ccccc}
L^{(1)} & = & L^{(0)} \cdot W & = & W \\
L^{(2)} & = & L^{(1)} \cdot W & = & W^2 \\
L^{(3)} & = & L^{(2)} \cdot W & = & W^3 \\
 & & \vdots & & \\
L^{(n-1)} & = & L^{(n-2)} \cdot W & = & W^{n-1}
\end{array}
$$

where $W^{n-1}$ contains the shortest path weights.

# Slow Multiplicative Method

SLOW-ALL-SHORTEST-PATHS($W$)
$n \leftarrow rows[W]$
$L^{(1)} \leftarrow W$
**for** $m \leftarrow 2$ **to** $n - 1$
  **do** $L^{(m)} \leftarrow$ EXTEND-SHORTEST-PATHS($L^{(m-1)}, W$)
**return** $L^{(n-1)}$

- Time complexity: $\Theta(n^4)$.

# How to Speed Up Multiplicative Method?

- We are often interested only in matrix $L^{(n-1)}$.

# How to Speed Up Multiplicative Method?

- We are often interested only in matrix $L^{(n-1)}$.
- If there is no negative-weight cycle, then $L^{(m)} = L^{(n-1)}$ for all $m \geq n-1$.

# How to Speed Up Multiplicative Method?

- We are often interested only in matrix $L^{(n-1)}$.
- If there is no negative-weight cycle, then $L^{(m)} = L^{(n-1)}$ for all $m \geq n-1$.
- Multiplicative operation defined in EXTEND-SHORTEST-PATHS is associative.

# How to Speed Up Multiplicative Method?

- We are often interested only in matrix $L^{(n-1)}$.
- If there is no negative-weight cycle, then $L^{(m)} = L^{(n-1)}$ for all $m \geq n-1$.
- Multiplicative operation defined in EXTEND-SHORTEST-PATHS is associative.
- Therefore, we can decrease the number of products from $n-1$ to $\lceil \log n - 1 \rceil$ and compute the sequence of matrices

$$
\begin{array}{lllll}
L^{(1)} & = & W & & \\
L^{(2)} & = & W^2 & & \\
L^{(4)} & = & W^4 & = & W^2 \cdot W^2 \\
L^{(8)} & = & W^8 & = & W^4 \cdot W^4 \\
 & & \vdots & & \\
L^{(2^{\lceil \log n-1 \rceil})} & = & W^{(2^{\lceil \log n-1 \rceil})} & = & W^{2^{\lceil \log n-1 \rceil - 1}} \cdot W^{2^{\lceil \log n-1 \rceil - 1}}
\end{array}
$$

Since $2^{\lceil \log n-1 \rceil} \geq n-1$, we get the final product $L^{(2^{\lceil \log n-1 \rceil})} = L^{(n-1)}$.

# Faster Multiplicative Method

```
FAST-ALL-SHORTEST-PATHS(W)
n ← rows[W]
L^(1) ← W
m ← 1
while m < n − 1
      do L^(2m) ← EXTEND-SHORTEST-PATHS(L^(m), L^(m))
         m ← 2m
return L^(m)
```

- Time complexity: $\Theta(n^3 \log n)$.