Lecture 11: Weighted graphs

Paths with minimum weight. Algorithms: Bellman-Ford, Dijkstra, Floyd-Warshall

december 2020

A weighted graph is a graph $G = (V, E)$ with a function $w : E \to \mathbb{R}$ which assigns a weight $w(e)$ to every edge $e \in E$.

- Weights can represent distances between node, but also other metrics, like costs, penalties, losses or other quantities that accumulate in a linear fashion along a path and we wish to minimize.
- We will study only **simple weighted graphs**, that is, graphs
  - without loops
  - with at most one edge from a node to another node
- We will write $w(x, y)$ instead of $w(e)$ if e is the edge $x{-}y$ or arc $x{\to}y$.
- Also, we will assume that $w(x, x) = 0$ and $w(x, y) = +\infty$ if there is no edge from x to y.

We write $x \overset{\pi}{\rightsquigarrow} y$ to indicate the fact that π is a list of nodes starting with x and ending with y.

We write $x \overset{\pi}{\rightsquigarrow} y$ to indicate the fact that π is a list of nodes starting with x and ending with y.

Weight of a list $\pi = [x_1, x_2, \ldots, x_k]$ is

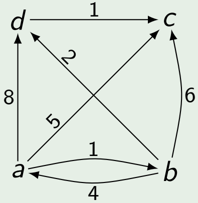
$$\mathrm{length}_w(\pi) = \sum_{i=1}^{k-1} w(x_i, x_{i+1}).$$

If $k = 1$ then $\pi = [x_1]$ and $\mathrm{length}_w(\pi) = 0$.

Weighted distance from x to y in G is

$$\delta_w(x, y) = \min\{\mathrm{length}_w(\pi) \mid x \overset{\pi}{\rightsquigarrow} y\}.$$

Example



$\delta_w(x,y)$	$y=a$	$y=b$	$y=c$	$y=d$
$x=a$	0	1	4	3
$x=b$	4	0	3	2
$x=c$	$+\infty$	$+\infty$	0	$+\infty$
$x=d$	$+\infty$	$+\infty$	$+\infty$	0

$\text{length}_w([a,b,c]) = 7$,
$\text{length}_w([a,d,c]) = 9$,
$\text{length}_w([a,b,d,c]) = 4$.

We will describe algorithmic solutions for the following problems:

1. Find paths with minimum weight from a source node s to all nodes that can be reached from s.
2. Find paths with minimum weight from x to y for all pairs of connected nodes $x \rightsquigarrow y$.

We will describe algorithmic solutions for the following problems:

1. Find paths with minimum weight from a source node s to all nodes that can be reached from s.
2. Find paths with minimum weight from x to y for all pairs of connected nodes $x \rightsquigarrow y$.

Remark

If $\pi = [x_1, x_2, \ldots, x_k]$ is a path from x_1 to x_k with $\text{length}_w(\pi) = \delta_w(x_1, x_k)$, then for all $1 \leq i \leq j \leq n$:

- If $\pi_{i,j} = [x_i, x_{i+1}, \ldots, x_j]$ then $\text{length}_w(\pi_{i,j}) = \delta(x_i, k_j)$.

That is, *all subpaths of a path with minimum weight have minimum weight*.

Cycles and negative weighted distances

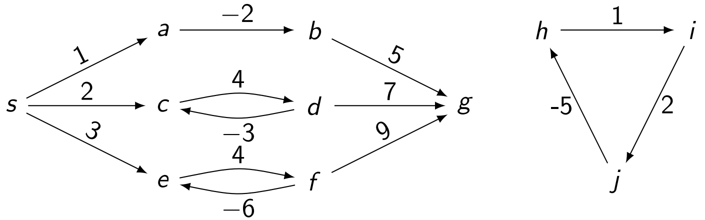
Edges e with $w(e) < 0$ can form cycles with minimum weight $\Rightarrow$ for all nodes x, y:

- If there is a node z of a cycle c with negative weight, andi $x \rightsquigarrow z \rightsquigarrow y$ then there is no $x \overset{\pi}{\rightsquigarrow} y$ with minimum weight because we can keep traversing c to produce paths whose weight decreases to $-\infty$. In this case, we define $\delta_w(x, y) = -\infty$.
- Otherwise, $\delta_w(x, y) \in \mathbb{R}$ and there is $x \overset{\pi}{\rightsquigarrow} y$ with $\text{length}_w(\pi) = \delta_w(x, y)$.

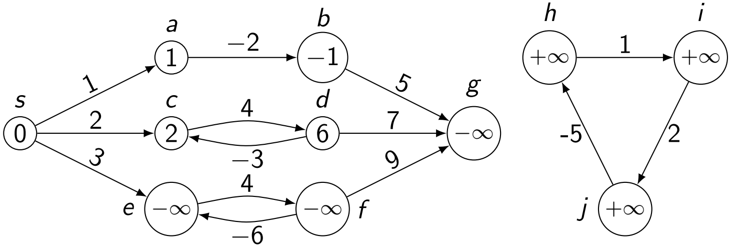
Cycles with negative weight

Example

The following digraph has cycles with negative weight:



The following figure indicates the values $\delta_w(s, x)$ for all x:



Let $x \overset{\pi}{\rightsquigarrow} y$ be a path with minimum weight. We note that

Let $x \overset{\pi}{\rightsquigarrow} y$ be a path with minimum weight. We note that

1. π can not contain a cycle with strictly negative weight because it would imply $\delta_w(x, y) = -\infty$.

Let $x \overset{\pi}{\rightsquigarrow} y$ be a path with minimum weight. We note that

1. π can not contain a cycle with strictly negative weight because it would imply $\delta_w(x, y) = -\infty$.
2. π can not contain a cycle with strictly positive weight because if we eliminate it from π we obtain $x \overset{\pi'}{\rightsquigarrow} y$ with $\text{length}_w(\pi') < \text{length}_w(\pi) = \delta_w(x, y)$, contradiction.

Let $x \overset{\pi}{\rightsquigarrow} y$ be a path with minimum weight. We note that

1. π can not contain a cycle with strictly negative weight because it would imply $\delta_w(x, y) = -\infty$.
2. π can not contain a cycle with strictly positive weight because if we eliminate it from π we obtain $x \overset{\pi'}{\rightsquigarrow} y$ with $\text{length}_w(\pi') < \text{length}_w(\pi) = \delta_w(x, y)$, contradiction.
3. We can assume π has no cycles with weight 0 because we can eliminate them from π without changing the weight.

Paths with minimum weight

Remarks

Let $x \overset{\pi}{\rightsquigarrow} y$ be a path with minimum weight. We note that

1. π can not contain a cycle with strictly negative weight because it would imply $\delta_w(x, y) = -\infty$.
2. π can not contain a cycle with strictly positive weight because if we eliminate it from π we obtain $x \overset{\pi'}{\rightsquigarrow} y$ with $\text{length}_w(\pi') < \text{length}_w(\pi) = \delta_w(x, y)$, contradiction.
3. We can assume π has no cycles with weight 0 because we can eliminate them from π without changing the weight.

Thus, we can restrict our search to acyclic paths $i \overset{\pi}{\rightsquigarrow} j$ with minimum weight. These paths contain at most $|V| = n$ nodes, thus at most $n - 1$ edges.

Both algorithms compute a representation with predecessors of a tree T_s with root s such that

1. The set of nodes of T_s is $S_s = \{x \in V \mid s \rightsquigarrow x\}$
2. For every $s \in S_s$, the list of nodes on the branches from s to x in T_s is a path with minimum weight from s to x in G.

Such a tree is called tree of paths with minimum weights from s in G.

Both algorithms compute a representation with predecessors of a tree T_s with root s such that

1. The set of nodes of T_s is $S_s = \{x \in V \mid s \rightsquigarrow x\}$
2. For every $s \in S_s$, the list of nodes on the branches from s to x in T_s is a path with minimum weight from s to x in G.

Such a tree is called tree of paths with minimum weights from s in G.

- **Dijkstra algorithm** is defined for weighted graphs with $w(e) > 0$ for all edges e.

Both algorithms compute a representation with predecessors of a tree T_s with root s such that

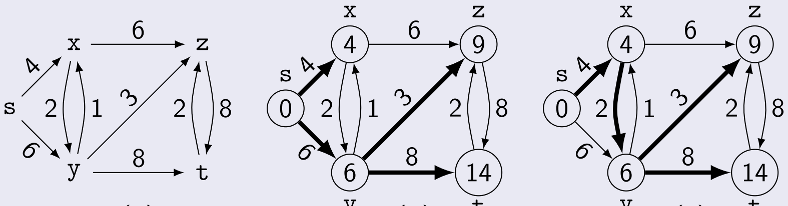
1. The set of nodes of T_s is $S_s = \{x \in V \mid s \rightsquigarrow x\}$
2. For every $s \in S_s$, the list of nodes on the branches from s to x in T_s is a path with minimum weight from s to x in G.

Such a tree is called tree of paths with minimum weights from s in G.

- **Dijkstra algorithm** is defined for weighted graphs with $w(e) > 0$ for all edges e.
- **Bellman-Ford algorithm** is defined for the general case, when we can have edges e with $w(e) < 0$.
  - It detects possible cycles with negative weight that can be reached from the source node s. In this case, it returns `false` to signal the existence of such a cycle, and it abandons the construction of T_s.

Illustrated example

The weighted digraph from Fig. (a) has 2 tree of paths with minimum weights from s. Figures (b) and (c) highlight the edges of these trees with thick arrows, and the value $\delta_w(\mathtt{s}, x)$ is written inside every node x.



(a) (b) (c)

The algorithms operate with

1. the representation with predecessors of a tree A_s with root s and set of nodes V. We will assume that, for every $x \in V$, π_x is the list of nodes from s to x in A_s.
2. $\mathtt{d}[x]$: an upper bound for $\text{length}_w(\pi_x)$:

$$\forall x \in V.\delta_w(s, x) \leq \text{length}_w(\pi_x) \leq \mathtt{d}[x].$$

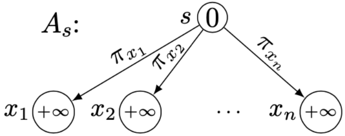
The algorithms operate with

1. the representation with predecessors of a tree A_s with root s and set of nodes V. We will assume that, for every $x \in V$, π_x is the list of nodes from s to x in A_s.
2. $\mathtt{d}[x]$: an upper bound for $\mathrm{length}_w(\pi_x)$:

$$\forall x \in V.\delta_w(s,x) \le \mathrm{length}_w(\pi_x) \le \mathtt{d}[x].$$

The initial values are

- $\mathtt{p}[s] = \mathtt{null}$ and $\mathtt{p}[x] = s$ for all $x \in V - \{s\}$, și
- $\mathtt{d}[s] = 0$ and $\mathtt{d}[x] = +\infty$ for all $x \in V - \{s\}$.



unde $V = \{s, x_1, x_2, \dots, x_n\}$.
Valorile lui $\mathtt{d}[x]$ sunt indicate
în interiorul nodurilor respective.

The values of `d[]` and `p[]` are modified by performing a finite number of edge relaxations; it is guaranteed that, when they stop:

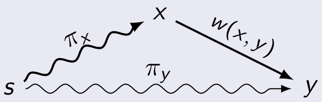
- A_s is a tree of paths with minimum weights from s in G.
- $\mathtt{d}[x] = \delta_w(s, x)$ for all $x \in V$.

Relaxing an edge from x to y

If $\mathtt{d}[x] + w(x, y) < \mathtt{d}[y]$ and we consider the path $\pi'_y = s \overset{\pi_x}{\rightsquigarrow} x \rightarrow y$ then

$$\delta_w(x, y) \leq \text{length}_w(\pi'_y) = \text{length}_w(\pi_x) + w(x, y) \leq \mathtt{d}[x] + w(x, y) < \mathtt{d}[y]$$

$\Rightarrow$ we can replace $\mathtt{p}[y]$ with $\mathtt{p}[x]$ and $\mathtt{d}[y]$ cu $\mathtt{d}[x] + w(x, y)$.




```
relax(x,y) {
   if (d[x] + w(x,y) < d[y]) {
        p[y] = x; d[y] = d[x] + w(x,y);
   }
}
```

Bellman-Ford algorithm

Pseudocode

```
boolean BellmanFord(G,s) {
  initialize(G,s);
  for i=1 to G.V()-1
     foreach x ∈V(G)
       for (y:adj[x])
           relax(x,y);
  foreach x ∈V(G)
     for (y:adj[x])
         if (d[x] > d[y]+w(x,y))
             return false;
  return true;
}
```

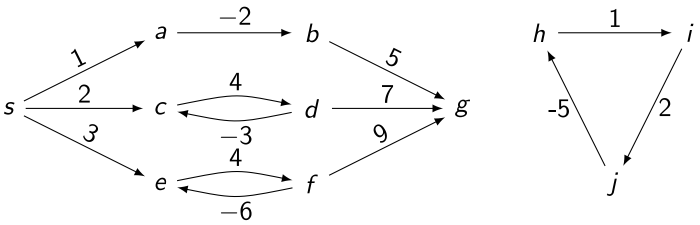
Bellman-Ford algorithm

Pseudocode

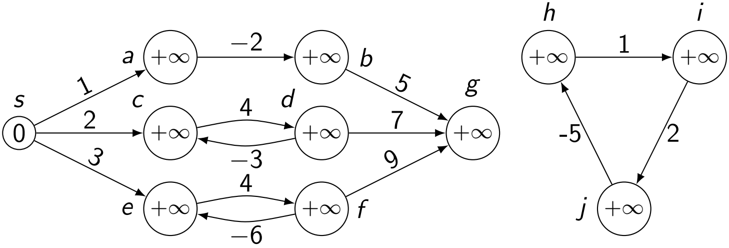
```
boolean BellmanFord(G,s) {
  initialize(G,s);
  for i=1 to G.V()-1
     foreach x ∈V(G)
       for (y:adj[x])
           relax(x,y);
  foreach x ∈V(G)
     for (y:adj[x])
         if (d[x] > d[y]+w(x,y))
             return false;
  return true;
}
```

Complexity (running time): $O(n^3)$

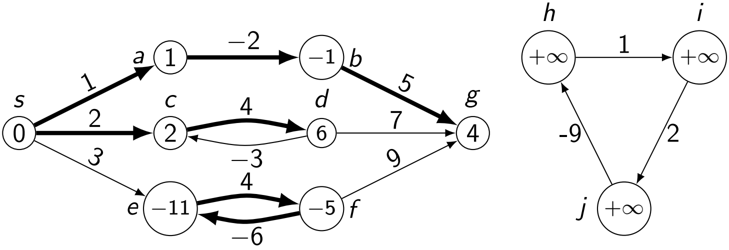
Bellman-Ford algorithm
Illustrated example



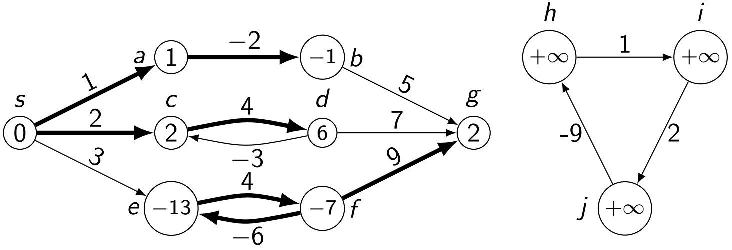
After initialization:



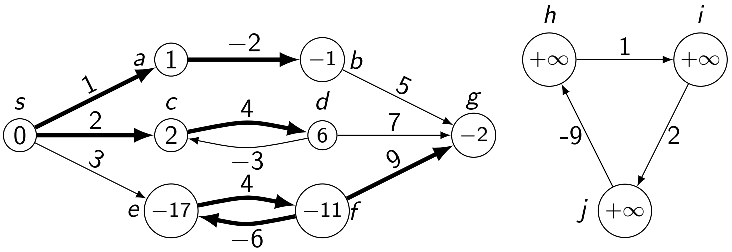
After the 6-th `for` loop:



After the 8-th `for` loop:



After the 10-th `for` loop:



The algorithm returns `false` because it detects

$$\mathtt{d}[f] = -11 > \mathtt{d}[e] + w(e, f).$$

```
void Dijkstra(G,s) {
  initialize(G,s);
  Q=set of nodes of G;
  while (!Q.isEmpty()) {
     extrage u cu d[u] = min{d[x] | x ∈ Q} din Q;
     for (v:G.adj(u))
        if (Q.contains(v))
           relax(u,v);
  }
}
```

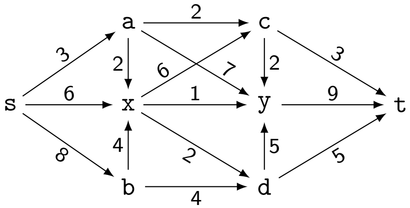
Dijkstra algorithm
Pseudocode

```
void Dijkstra(G,s) {
  initialize(G,s);
  Q=set of nodes of G;
  while (!Q.isEmpty()) {
     extrage u cu d[u] = min{d[x] | x ∈ Q} din Q;
     for (v:G.adj(u))
        if (Q.contains(v))
           relax(u,v);
  }
}
```

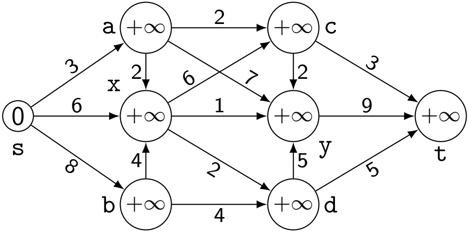
Complexity (running time): $O(n^2)$

Dijkstra algorithm

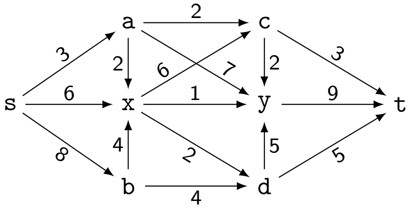
Illustrated example

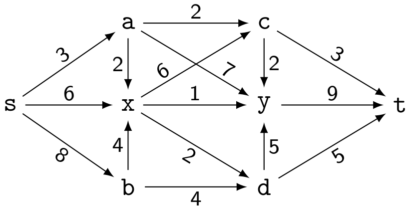


After initialization we have

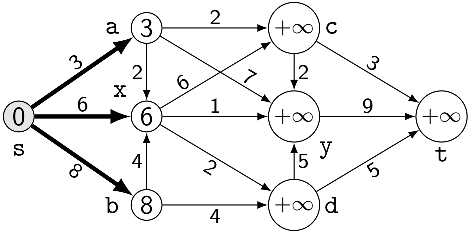


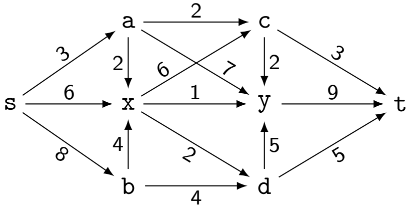
Dijkstra algorithm
Illustrated example



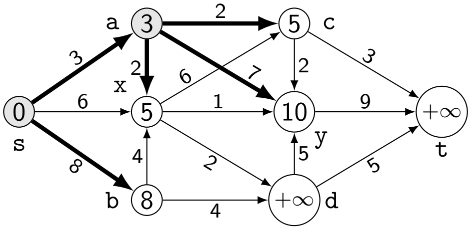


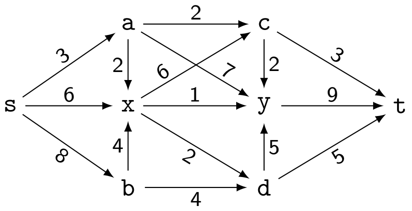
After relaxing all edges from s we have



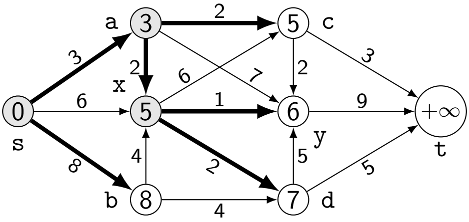


After relaxing all edges from a we have



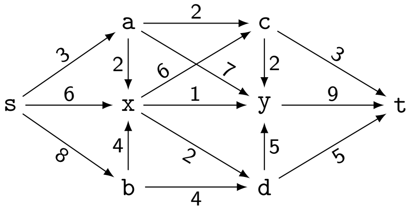


After relaxing all edges from x we have

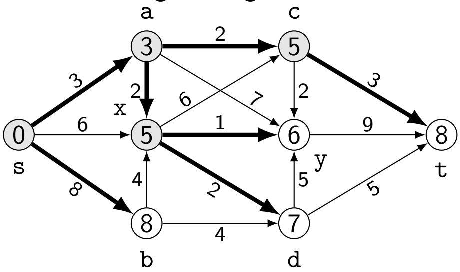


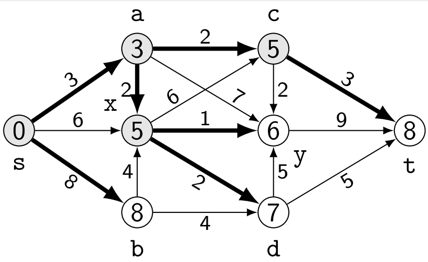
Dijkstra algorithm

Illustrated example



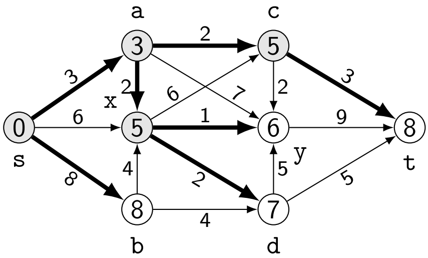
After relaxing all edges from c we have





Future relaxations do not change the values of p[] and d[]:

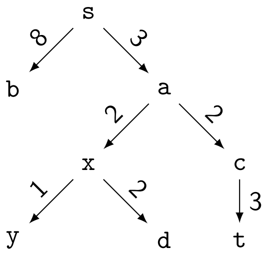
x	s	a	x	b	c	y	d	t
p[x]	null	s	a	s	a	x	x	c
d[x]	0	3	5	8	5	6	7	8



Future relaxations do not change the values of p[] and d[]:

x	s	a	x	b	c	y	d	t
p[x]	null	s	a	s	a	x	x	c
d[x]	0	3	5	8	5	6	7	8

$\Rightarrow$ the tree of paths with minimum weights computed by the algorithm is



Paths with minimum weights between all pairs of nodes

Given a weighted graph G with n nodes

Find for all $x, y \in V$ with $x \rightsquigarrow y$, a path $x \overset{\pi_{x,y}}{\rightsquigarrow} y$ with $\text{length}_w(\pi_{x,y}) = \delta_w(x, y)$.

Paths with minimum weights between all pairs of nodes

Given a weighted graph G with n nodes

Find for all $x, y \in V$ with $x \rightsquigarrow y$, a path $x \overset{\pi_{x,y}}{\rightsquigarrow} y$ with $\text{length}_w(\pi_{x,y}) = \delta_w(x, y)$.

REMARKS:

1. This problem can be solved by running n times one of the previous two algorithms, once for every node $x \in V(G)$ as source node.
2. Runtime complexity:
   - $O(n^4)$ if we use Bellman-Ford alg. for the general case when edges can have negative weights.
   - $O(n^3)$ if we use Dijkstra alg. for the special case when $w(e) > 0$ for all edges $e \in E$.
3. We will describe a new method – Floyd-Warshall algorithm:
   - Runtime complexitaty: $O(n^3)$ when we can have edges with negative weights, but no cycles with negative weight.

Two $n \times n$ arrays, such that, for all $x, y \in V$:

1. $\mathtt{d}[x][y]$: an upper bound for $\delta_w(x, y)$.
2. $\mathtt{P}[x][y] \in \{\mathtt{null}\} \cup V$.

When the algorithm stops, the values of $\mathtt{P}[][]$ and $\mathtt{d}[][]$ have the following properties:

- $\mathtt{d}[x][y] = \delta_w(x, y)$.
- If $x \neq y$ and there is a path with minimum weight from x la y then $\mathtt{P}[x][y]$ is the predecessor of x on a path $x \overset{\pi}{\rightsquigarrow} y$ with minimum weight.

Floyd-Warshall algorithm

Basic idea

If $x, y, z \in V$ then any path $\pi_{x,y}$ with minimum weight from x to y has one of the following two shapes:

1. $$x \xrightarrow{\pi_{x,y}} y$$

   where z is not an intermediary node of $\pi_{x,y}$, or

2. $$x \xrightarrow{\pi_{x,z}} z \xrightarrow{\pi_{z,y}} y$$

   where z is not an intermediary node of $\pi_{x,z}$ and $\pi_{z,y}$.

Floyd-Warshall algorithm

Basic idea

If $x, y, z \in V$ then any path $\pi_{x,y}$ with minimum weight from x to y has one of the following two shapes:

1. $$x \overset{\pi_{x,y}}{\rightsquigarrow} y$$
   where z is not an intermediary node of $\pi_{x,y}$, or
2. $$x \overset{\pi_{x,z}}{\rightsquigarrow} z \overset{\pi_{z,y}}{\rightsquigarrow} y$$
   where z is not an intermediary node of $\pi_{x,z}$ and $\pi_{z,y}$.

$\Rightarrow$ we can define a recursive method to compute the elements of the arrays `P[][]` and `d[][]`.

Let $[x_1, x_2, \ldots, x_n]$ be a fixed enumeration of the nodes of G. For $0 \leq k \leq n$ we define arrays $\mathtt{d}[k]$ and $\mathtt{P}[k]$ of size $n \times n$ as follows:

- $\mathtt{d}[k][i][j]$ este cea mai mică lungime ponderată a unei căi de la x_i la x_j care trece doar prin noduri intermediare din mulțimea $\{x_1, \ldots, x_k\}$. Dacă o astfel de cale nu există, atunci $\mathtt{d}[k][i][j] = +\infty$.
- $\mathtt{P}[k][i][j]$ este `null` dacă $i = j$ sau $\mathtt{d}[k][i][j] = +\infty$. În caz contrar, $\mathtt{P}[k][i][j]$ este predecesorul nodului x_j pe un drum cu lungime ponderată minimă de la x_i la x_j care trece doar prin noduri intermediare din mulțimea $\{x_1, \ldots, x_k\}$.

Floyd-Warshall algorithm

The recursive computation of d[][] and P[][] (continued)

We learn that, for all $i, j \in \{1, 2, \ldots, n\}$ we have

$$
\begin{aligned}
\mathtt{d}[0][i][j] &= w(x_i, x_j), \\
\mathtt{P}[0][i][j] &= \begin{cases} \mathtt{null} & \text{if } i = j \text{ or } w(x_i, x_j) = +\infty, \\ x_i & \text{otherwise} \end{cases}
\end{aligned}
$$

and if $1 \leq k \leq n$ then

$$
\begin{aligned}
\mathtt{d}[k][i][j] &= \min(\mathtt{d}[k-1][i][j], \mathtt{d}[k-1][i][k] + \mathtt{d}[k-1][k][j]), \\
\mathtt{P}[k][i][j] &= \begin{cases} \mathtt{P}[k-1][i][j] & \text{if } \mathtt{d}[k-1][i][j] = \mathtt{d}[k][i][j], \\ \mathtt{P}[k-1][k][j] & \text{otherwise.} \end{cases}
\end{aligned}
$$

We learn that, for all $i, j \in \{1, 2, \ldots, n\}$ we have

$$\begin{aligned} \mathtt{d}[0][i][j] &= w(x_i, x_j), \\ \mathtt{P}[0][i][j] &= \begin{cases} \mathtt{null} & \text{if } i = j \text{ or } w(x_i, x_j) = +\infty, \\ x_i & \text{otherwise} \end{cases} \end{aligned}$$

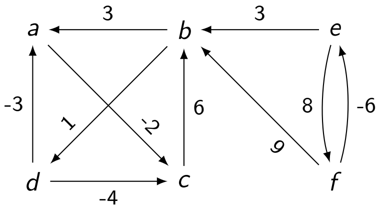
and if $1 \leq k \leq n$ then

$$\begin{aligned} \mathtt{d}[k][i][j] &= \min(\mathtt{d}[k-1][i][j], \mathtt{d}[k-1][i][k] + \mathtt{d}[k-1][k][j]), \\ \mathtt{P}[k][i][j] &= \begin{cases} \mathtt{P}[k-1][i][j] & \text{if } \mathtt{d}[k-1][i][j] = \mathtt{d}[k][i][j], \\ \mathtt{P}[k-1][k][j] & \text{otherwise.} \end{cases} \end{aligned}$$

FINAL REMARK: Because the intermediary nodes of every path are in the set $\{x_1, x_2, \ldots, x_n\}$, we can define

$$\mathtt{d}[x_i][x_j] = \mathtt{d}[n][i][j] \text{ and } \mathtt{P}[x_i][x_j] = \mathtt{P}[n][i][j].$$

1. Initialization of arrays $\mathtt{d}[0]$ and $\mathtt{P}[0]$ takes $O(n^2)$ time.
2. The computation of $\mathtt{d}[k]$ from $\mathtt{d}[k-1]$ and $\mathtt{P}[k]$ from $\mathtt{P}[k-1]$ takes $O(n^2)$ time.
3. This computation is repeated for k from 1 to $n \Rightarrow$ runtime complexity $n \cdot O(n^2) = O(n^3)$.

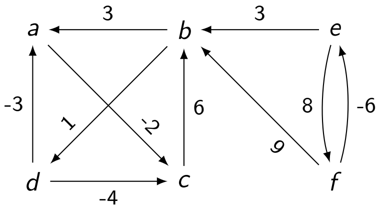


Nodes are enumerated in the order $[a, b, c, d, e, f]$

k	$\mathtt{d}[k]$	$\mathtt{P}[k]$
0	$\begin{pmatrix} 0 & +\infty & -2 & +\infty & +\infty & +\infty \\ 3 & 0 & +\infty & 1 & +\infty & +\infty \\ +\infty & 6 & 0 & +\infty & +\infty & +\infty \\ -3 & +\infty & -4 & 0 & +\infty & +\infty \\ +\infty & 3 & +\infty & +\infty & 0 & 8 \\ +\infty & 9 & +\infty & +\infty & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & \bullet & a & \bullet & \bullet & \bullet \\ b & \bullet & \bullet & b & \bullet & \bullet \\ \bullet & c & \bullet & \bullet & \bullet & \bullet \\ d & \bullet & d & \bullet & \bullet & \bullet \\ \bullet & e & \bullet & \bullet & \bullet & e \\ \bullet & f & \bullet & \bullet & f & \bullet \end{pmatrix}$

Floyd-Warshall algorithm

Illustrated example

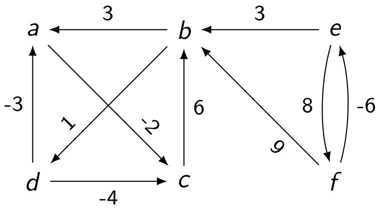


Nodes are enumerated in the order $[a, b, c, d, e, f]$

k	d$[k]$	P$[k]$
1	$\begin{pmatrix} 0 & +\infty & -2 & +\infty & +\infty & +\infty \\ 3 & 0 & 1 & 1 & +\infty & +\infty \\ +\infty & 6 & 0 & +\infty & +\infty & +\infty \\ -3 & +\infty & -5 & 0 & +\infty & +\infty \\ +\infty & 3 & +\infty & +\infty & 0 & 8 \\ +\infty & 9 & +\infty & +\infty & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & \bullet & a & \bullet & \bullet & \bullet \\ b & \bullet & a & b & \bullet & \bullet \\ \bullet & c & \bullet & \bullet & \bullet & \bullet \\ d & \bullet & a & \bullet & \bullet & \bullet \\ \bullet & e & \bullet & \bullet & \bullet & e \\ \bullet & f & \bullet & \bullet & f & \bullet \end{pmatrix}$

Floyd-Warshall algorithm

Illustrated example

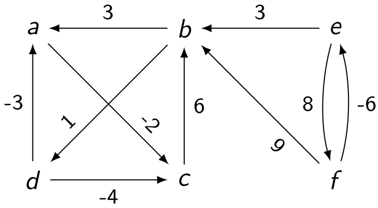


Nodes are enumerated in the order $[a, b, c, d, e, f]$

k	$\mathtt{d}[k]$	$\mathtt{P}[k]$
2	$\begin{pmatrix} 0 & +\infty & -2 & +\infty & +\infty & +\infty \\ 3 & 0 & 1 & 1 & +\infty & +\infty \\ 9 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & +\infty & -5 & 0 & +\infty & +\infty \\ 6 & 3 & 4 & 4 & 0 & 8 \\ 12 & 9 & 10 & 10 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & \bullet & a & \bullet & \bullet & \bullet \\ b & \bullet & a & b & \bullet & \bullet \\ b & c & \bullet & b & \bullet & \bullet \\ d & \bullet & a & \bullet & \bullet & \bullet \\ b & e & a & b & \bullet & e \\ b & f & a & b & f & \bullet \end{pmatrix}$

Floyd-Warshall algorithm

Illustrated example

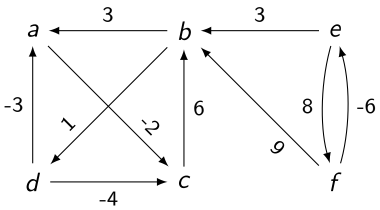


Nodes are enumerated in the order $[a, b, c, d, e, f]$

k	$\mathtt{d}[k]$	$\mathtt{P}[k]$
3	$\begin{pmatrix} 0 & 4 & -2 & 5 & +\infty & +\infty \\ 3 & 0 & 1 & 1 & +\infty & +\infty \\ 9 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & 1 & -5 & 0 & +\infty & +\infty \\ 6 & 3 & 4 & 4 & 0 & 8 \\ 12 & 9 & 10 & 10 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ b & \bullet & a & b & \bullet & \bullet \\ b & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ b & e & a & b & \bullet & e \\ b & f & a & b & f & \bullet \end{pmatrix}$

Floyd-Warshall algorithm

Illustrated example

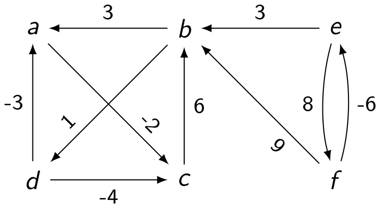


Nodes are enumerated in the order $[a, b, c, d, e, f]$

k	$\mathtt{d}[k]$	$\mathtt{P}[k]$
4	$\begin{pmatrix} 0 & 4 & -2 & 5 & +\infty & +\infty \\ -2 & 0 & -4 & 1 & +\infty & +\infty \\ 4 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & 1 & -5 & 0 & +\infty & +\infty \\ 1 & 3 & -1 & 4 & 0 & 8 \\ 7 & 9 & 5 & 10 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ d & f & a & b & f & \bullet \end{pmatrix}$

Floyd-Warshall algorithm

Illustrated example

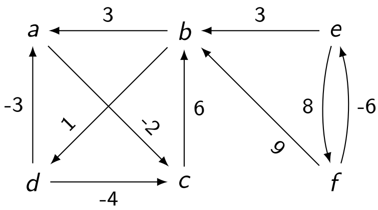


Nodes are enumerated in the order $[a, b, c, d, e, f]$

k	$\mathtt{d}[k]$	$\mathtt{P}[k]$
5	$\begin{pmatrix} 0 & 4 & -2 & 5 & +\infty & +\infty \\ -2 & 0 & -4 & 1 & +\infty & +\infty \\ 4 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & 1 & -5 & 0 & +\infty & +\infty \\ 1 & 3 & -1 & 4 & 0 & 8 \\ -5 & -3 & -7 & -2 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ d & e & a & b & f & \bullet \end{pmatrix}$

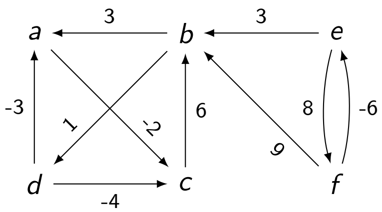
Floyd-Warshall algorithm

Illustrated example



Nodes are enumerated in the order $[a, b, c, d, e, f]$

k	d[k]	P[k]
6	$\begin{pmatrix} 0 & 4 & -2 & 5 & +\infty & +\infty \\ -2 & 0 & -4 & 1 & +\infty & +\infty \\ 4 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & 1 & -5 & 0 & +\infty & +\infty \\ 1 & 3 & -1 & 4 & 0 & 8 \\ -5 & -3 & -7 & -2 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ d & e & a & b & f & \bullet \end{pmatrix}$



Nodes are enumerated in the order $[a, b, c, d, e, f]$

In the end, the element values of arrays `P[][]` and `d[][]` are

	a	b	c	d	e	f
a	•	c	a	b	•	•
b	d	•	a	b	•	•
c	d	c	•	b	•	•
d	d	c	a	•	•	•
e	d	e	a	b	•	e
f	d	e	a	b	f	•

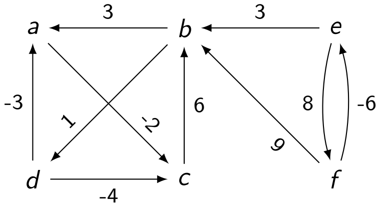
	a	b	c	d	e	f
a	0	4	-2	5	$+\infty$	$+\infty$
b	-2	0	-4	1	$+\infty$	$+\infty$
c	4	6	0	7	$+\infty$	$+\infty$
d	-3	1	-5	0	$+\infty$	$+\infty$
e	1	3	-1	4	0	8
f	-5	-3	-7	-2	-6	0

- Array P is called predecessor matrix.
- For every node $x \in G$ we can define the tree T_x with root x and
  - set of nodes $\{x\} \cup \{y \in V \mid \mathtt{P}[x][y] \neq \mathtt{null}\}$
  - set of edges $\{\mathtt{P}[x][y] \to y \mid y \in V - \{x\}\}$.
- T_x is a tree of paths with minimum weights from x in G, and it can be extracted from the row of node x in array P[][].

Floyd-Warshall algorithm

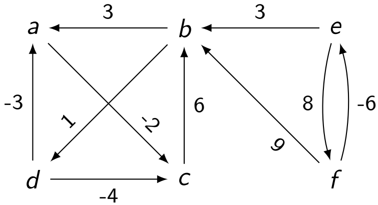
Aplication of the predecesor matrix P

Find a a tree of paths with minimum weight from source node f in



$$P[6] = \begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ d & e & a & b & f & \bullet \end{pmatrix}$$

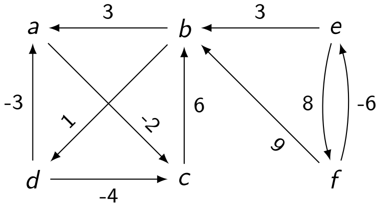
Find a a tree of paths with minimum weight from source node f in



$$P[6] = \begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ \color{red}{d} & \color{red}{e} & \color{red}{a} & \color{red}{b} & \color{red}{f} & \color{red}{\bullet} \end{pmatrix}$$

f is the 6-th element in the node enumeration $[a, b, c, d, e, f]$, thus T_f can be obtained from line 6 of the matrix P[6]:

Find a a tree of paths with minimum weight from source node f in



$$P[6] = \begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ d & e & a & b & f & \bullet \end{pmatrix}$$

f is the 6-th element in the node enumeration $[a, b, c, d, e, f]$, thus T_f can be obtained from line 6 of the matrix P[6]:

$$f \xrightarrow{-6} e \xrightarrow{3} b \xrightarrow{1} d \xrightarrow{-3} a \xrightarrow{-2} c$$