Graphs
-
The Shortest Path Problem

Marcello Sanguineti

marcello.sanguineti@unige.it

DIBRIS – Università di Genova

The Shortest Path (SP) problem

$G=(V,E)$ directed graph.

A cost (length) c_{ij} is associated with each arc $(i,j) \in E$, $i,j=1,\ldots,n$.
The cost (length) of a directed path $P=\{v_0,v_1,\ldots,v_K\}$ is defined as

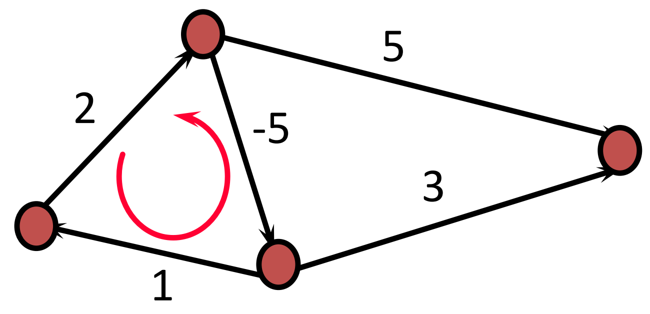
$$c(P) := \sum_{(i,j)\in E_P} c_{ij} \qquad \text{where} \quad E_P:=\{(v_{l-1},v_l),\ l=1,\ldots,k\}$$

Objective: find the directed path (i.e, the walk with no repeated nodes and hence no repeated arcs) P^* that connects two given nodes $s,t \in V$ with minimum cost (length).

Remark. In the following, we shall use interchangeably the words "length" and "cost" and the words "shortest" and "minimum".

If in G there exist no path from s to t, then the problem is unfeasible.

If in G there exists a directed cycle with negative cost, then the problem is unbounded.



cycle with negative cost

ILP model for the SP problem

$$x_{ij} = \begin{cases} 1 & if\ (i,j) \in P^* \\ 0 & if\ (i,j) \notin P^* \end{cases}$$

$$\forall (i,j) \in E$$

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij}$$

$$\sum_{(i,h)\in\delta^-(h)} x_{ij} - \sum_{(h,j)\in\delta^+(h)} x_{ij} = 0 \qquad \forall h \in V - \{s,t\} \qquad (1)$$

$$\sum_{(i,s)\in\delta^-(s)} x_{is} - \sum_{(s,j)\in\delta^+(s)} x_{sj} = -1 \qquad (2)$$

$$\sum_{(i,t)\in\delta^-(t)} x_{it} - \sum_{(t,j)\in\delta^+(t)} x_{tj} = 1 \qquad (3)$$

$$\sum_{(i,j)\in E(S)} x_{ij} \le |S| - 1 \qquad \forall S \subseteq V,\ S \neq \emptyset \qquad (4)$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E$$

$\delta^-(j)$ ingoing star of *j* $\delta^+(j)$ outgoing star of *j*

The constraints *(1)*, *(2)*, and *(3)* express that the solution must be a walk from s to t .

The constraints *(4)* exclude cycles. They are $\boldsymbol{2^{n}-1}$ and are called "**subtour elimination**" constraints.

The subtour elimination constraints are the same as those in the IP model of the MST problem. Their number increases **exponentially** with the number n of nodes: for large graphs, there is a huge number of constraints!

So, the IP model of SP has 2^n-1 constraints that are the same as those of the MST problem. Only other $n+1$ constraints differ.

The fact that the MST and the SP problems have “the great majority” of constraints in common **might induce** one to suppose that the two problems belong to the same complexity class. **Instead**:

whereas MST belongs to the class P (there even exists a polynomial algorithm that is greedy), **SP is NP-hard**

as the next theorem shows.

Theorem. The SP problem is NP-hard.

Proof. We proceed by exploiting a polynomial transformation.

We start observing that a path cannot have more than n arcs and it has n arcs if and only if it is an Hamiltonian cycle.

Hence the problem of finding, if it exists, an Hamoltonian cycle in $G=(V,E)$ can be solved in the following way:

"find the path from s to t that contains as many arcs as possible, i.e., find the minimum-cost path from s to t when each arc $(i,j) \in E$ is assigned the cost $c_{ij} = -1$".

Hence there exists an NP-complete problem, namely the Hamiltonian cycle, that can be transformed polynomially into the SP problem.

Thus, SP is NP-hard. ▪

Particular case: absence of negative or zero-cost cycles

When one knows **a priori** that the graph contains no negative or zero-cost cycle, **the subtour elimination constraints are redundant**: there is no “convenience” in walking a cycle!

Hence, in such a case the IP model becomes merely:

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij}$$

$$\sum_{(i,h)\in\delta^-(h)} x_{ij} - \sum_{(i,h)\in\delta^+(h)} x_{ij} = 0 \qquad \forall h \in V - \{s,t\} \qquad (1)$$

$$\sum_{(i,s)\in\delta^-(s)} x_{is} - \sum_{(s,j)\in\delta^+(s)} x_{sj} = -1 \qquad (2)$$

$$\sum_{(i,t)\in\delta^-(t)} x_{it} - \sum_{(t,j)\in\delta^+(t)} x_{tj} = 1 \qquad (3)$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E$$

So, given the incidence matrix A_G of the graph G, when there is no negative or zero-cost cycle the IP model can be written as

$$\begin{aligned} \min\ & \underline{w}^T \underline{x} \\ & A_G\, \underline{x} = \underline{b} \\ & \underline{x} \in \{0,1\}^m \end{aligned}$$

where $\underline{x}$ is the incidence vector of the arcs in G and

$$\underline{\mathrm{b}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -1 \\ 1 \end{bmatrix}$$

Since the incidence matrix of a directed graph is TUM (totally unimodular),

the SP problem with no negative or zero-cost cycle can be solved by relaxing the integrity constraints. Hence, it belongs to the class P of polynomial problems.

Particular case: absence of negative costs

Let us suppose that

$c_{ij} \geq 0$ for every $(i,j) \in E$.

In this case the shortest path enjoys the important property stated in the next theorem.

Theorem. Let us suppose to know the costs $g(i)$ of the shortest paths from s to every node i belonging to a set $S \subset V$ such that $s \in S$ and let $g(s):=0$. Moreover, let

$$(v,h):=\operatorname{argmin}\{g(i) + c_{ij} : (i,j) \in \delta^+(S)\}$$

If $c_{ij} \geq 0$ for every $(i,j) \in E$, then $g(v) + c_{vh}$ represents the cost of the shortest path from s to h.

We'll see that the proof of this theorem "suggests" a solution algorithm for the SP problem with no negative costs (namely, the Dijkstra Algorithm).

Proof.

$g(v) + c_{vh}$ represents the cost of a path from s to v followed by the arc (v,h).

We have to show that every other path P from s to h has a cost

$$c(P) \geq g(v) + c_{vh} .$$

Let $S \subset V$ be the set of nodes defined in the theorem (i.e., the set of nodes for which the shortest paths from s to each of them are known) and let (i,j) be the first arc in $P \cap \delta^{+}(S)$.

Let us make a partition of P into

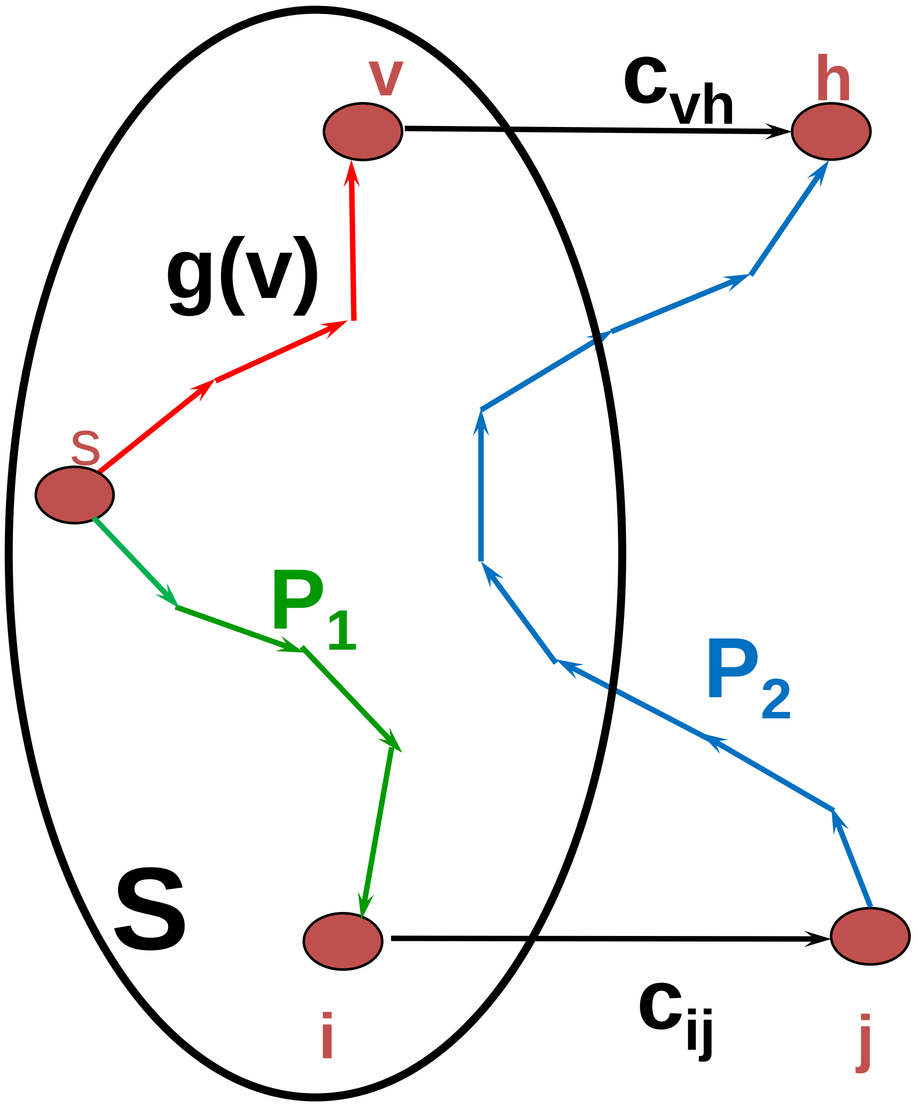
$P_1 \cup \{(i,j)\} \cup P_2$,

where P_1 and P_2 are two paths from s to i and from j to h, respectively. Since $c(P_1) \geq g(i)$ and $c(P_2) \geq 0$, we have (refer to the next figure):

$$c(P) = c(P_1) + c_{ij} + c(P_2) \geq g(i) + c_{ij} \geq g(v) + c_{vh}.$$

Hence, $g(v) + c_{vh}$ represents the cost of the shortest path from s to h.

$\blacksquare$


v
c_{vh}
h
g(v)
s
P_1
P_2
S
i
c_{ij}
j

The Dijkstra Algorithm

Applicability: directed graphs with no negative costs.

Notations:

c_{ij} : cost of the arc $(i,j)\in E$

s: departure node

t: arrival node

$g(i)$: cost of the shortest path from s to i

$h(i)$: label of node i (i.e., current value of the path from s to i)

$\pi(i)$: predecessor of node i along the shortest path from s to i

U: set of the currently-visited nodes

(1) Initialization

$g(s):=0,\ U:=\{s\}$

$h(i):=c_{si}\ \forall(s,i)\in E,\ \ h(j):=\infty\ \ \forall(s,j)\notin E$

$\pi(i):=s\ \ \forall(s,i)\in E,\ \pi(j)$ undefined $\forall(s,j)\notin E$.

(2) Select

$$i\in \operatorname{argmin}\{h(l) : l\notin U\}$$

and let $U:=U\cup\{i\}$, $g(i):=h(i)$. If $U=V$, then the algorithm terminates and the shortest s-t path is determined by the sequence of the labels $\pi(i)$.

(3) For every $j\notin U$ such that $(i,j)\in E$, update the corresponding label:

$$h(j):=\min\{g(i)+c_{ij},\ h(j)\}.$$

If $h(j)=g(i)+c_{ij}$, then $\pi(j):=i$.

Go to step (2).

Remarks

The Dijkstra Algorithm is a "**labelling algorithm**": it associates a label to each node and **at each iteration it checks whether the labels need to be updated**.

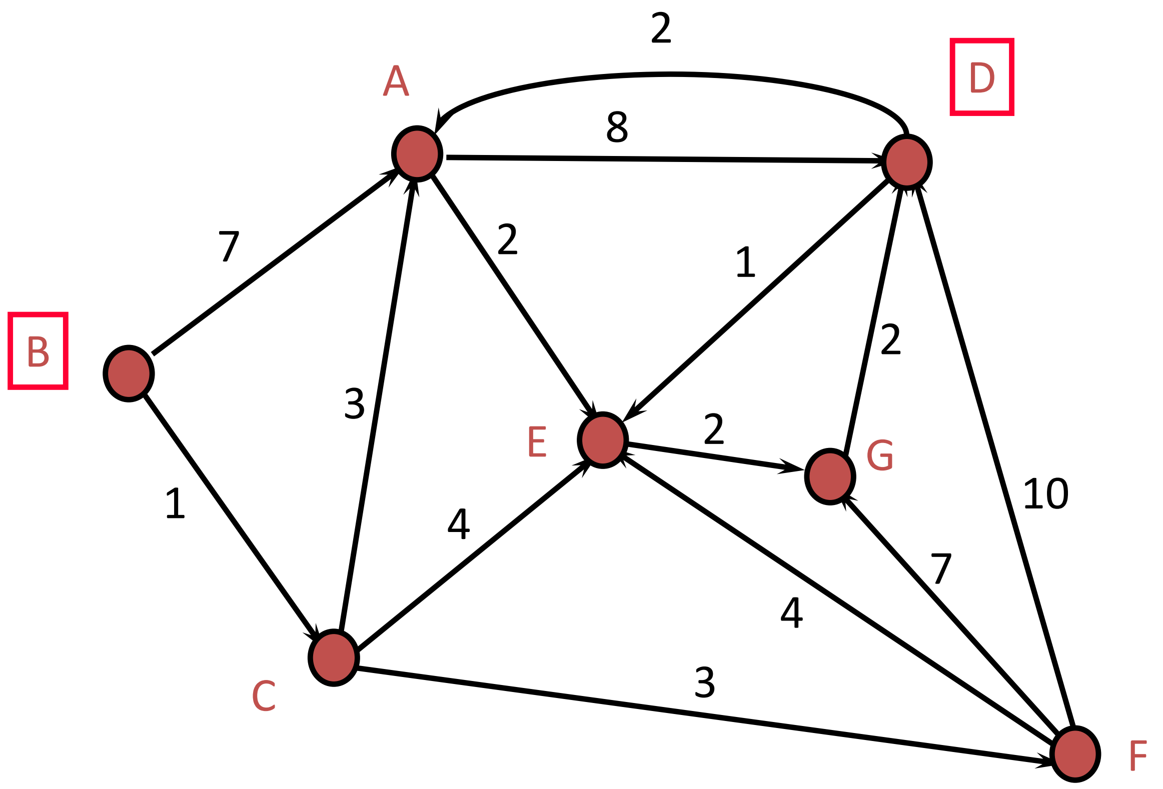
Since at each iteration it "calls into question" the choices made at the previous iterations, **it is <u>not</u> greedy**.

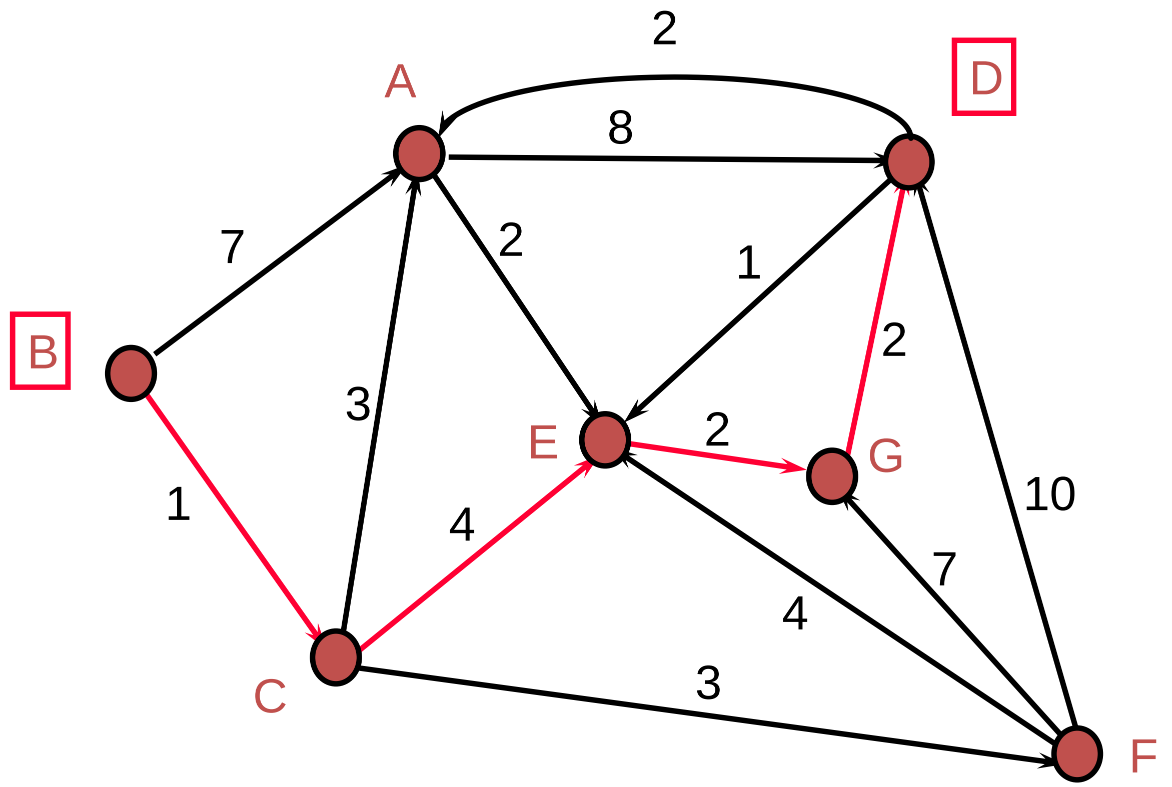
Label of a node: cost of the current shortest path from s to that node.

By the previous theorem, **at each iteration one finds the shortest path from s to at least one node**.

Hence, **the maximum number of iterations** required to find the shortest path **is n**.

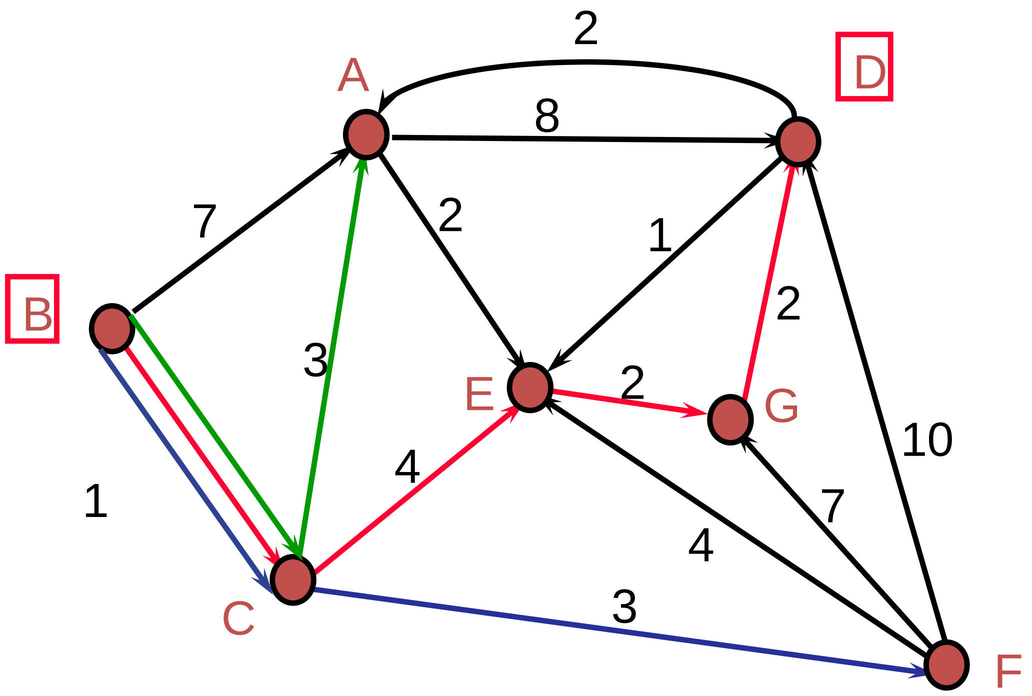
Example: find the shortest path from B to D in the following directed graph.





2
A
8
D
7
2
1
B
2
3
E
2
G
1
4
10
7
4
C
3
F

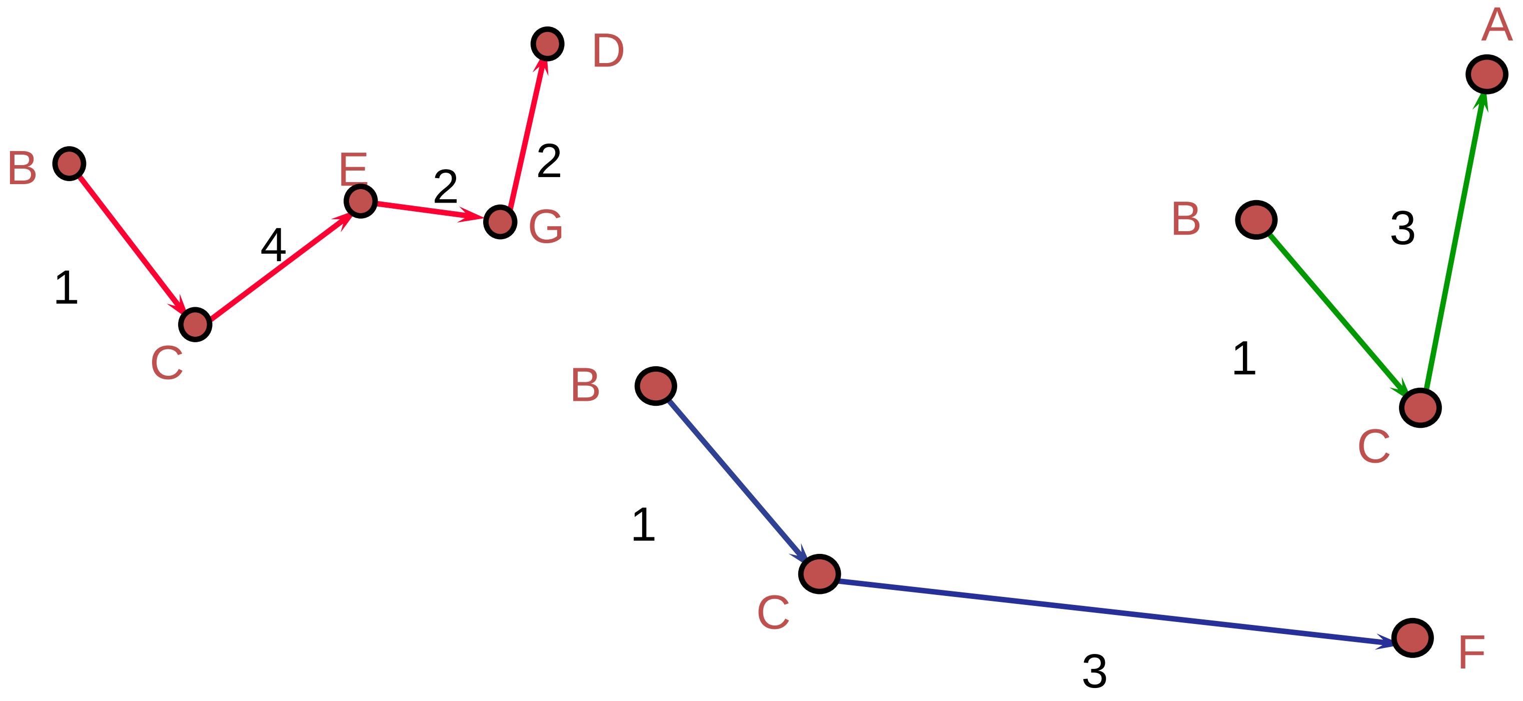
Arborescence of shortest paths



The Dijkstra Algorithms originates as “one-to-one”.

However, it provides as by-products the shortest paths from *s* to every other node (“**arborescence of shortest paths**”). Hence, it is a “**one-to-all**” algorithm.

To determine the shortest paths between every pair of nodes, one can apply the algorithm n times, each time choosing a different departure node.



Tabular implementation of the Dijkstra Algorithm

Notations:

$(h(i), \pi(i))$	cost of the **current shortest path** and **current predecessor**, resp., for node I
$*(g(i), \pi(i))$	cost of the **shortest path** and **predecesso**r, resp., for node i

guarda l'asterisco è l'ordine con cui lo considerano

	A	B	C	D	E	F	G
B	(7,B)	*(0,B)	(1,B)	(∞,-)	(∞,-)	(∞,-)	(∞,-)
	(7,B)	*(0,B)	*(1,B)	(∞,-)	(∞,-)	(∞,-)	(∞,-)
C	(4,C)	*(0,B)	*(1,B)	(∞,-)	(5,C)	(4,C)	(∞,-)
	(4,C)	*(0,B)	*(1,B)	(∞,-)	(5,C)	*(4,C)	(∞,-)
F	(4,C)	*(0,B)	*(1,B)	(14,F)	(5,C)	*(4,C)	(11,F)
	*(4,C)	*(0,B)	*(1,B)	(14,F)	(5,C)	*(4,C)	(11,F)
A	*(4,C)	*(0,B)	*(1,B)	(12,A)	(5,C)	*(4,C)	(11,F)
E	*(4,C)	*(0,B)	*(1,B)	(12,A)	*(5,C)	*(4,C)	(7,E)
G	*(4,C)	*(0,B)	*(1,B)	(9,G)	*(5,C)	*(4,C)	*(7,E)
	*(4,C)	*(0,B)	*(1,B)	*(9,G)	*(5,C)	*(4,C)	*(7,E)

The shortest path is B-C-E-G-D

If there are also arcs with negative costs…

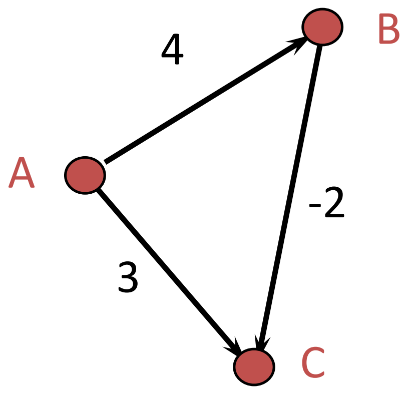
… **then the Dikstra Algorithm may fail**.

Indeed, when there is at least one negative cost, it is not guaranteed that at each iteration one finds the definitive shortest path from s to one of the other nodes.

Example

A	B	C
*(0,A)	(4,A)	(3,A)
*(0,A)	(4,A)	*(3,A)
*(0,A)	*(4,A)	(2,B)

The algorithm “contradicts itself”!



Complexity of the Dijkstra Algorithm

(1) Initialization

$g(s):=0, U:=\{s\}$

$h(i):=c_{si}\ \forall(s,i)\in E,\ \ h(j):=\infty\ \forall(s,j)\notin E$

$\pi(i):=s\ \ \forall(s,i)\in E, \pi(j)$ undefined $\forall(s,j)\notin E$

(2) Select

$$i \in \text{argmin}\,\{h(l) : l \notin U\}$$

and let $U:=U\cup\{i\}$, and $g(i):=h(i)$. *If* $U=V$, then the algorithm terminates and the shortest s - t path is specified by the sequence of the labels $\pi(i)$.

(3) For every $j \notin U$ such that $(i,j) \in E$, update the corresponding label:

$$h(j):=\min\{g(i)+c_{ij}, h(j)\}.$$

If $h(j)=g(i)+c_{ij}$, then $\pi(j):=i$. Go to step *(2)*.

- Initialization: $O(n)$
- Steps 2 e 3: $O(n)$ and they are executed n times each
- Hence, the complexity is $\boldsymbol{O(n^2)}$
- There exist other implementations of the algorithm. E.g., one of them allows to get the complexity $\boldsymbol{O(m \log n)}$

The Bellman-Ford Algorithm

Notations:

$g(i)$: cost of the shortest path from s to i

$h^k(i)$: label of node i at step k (current cost of the path from s to i)

c_{ij}: cost of the arc $(i,j)\in E$

$\pi(i)$: predecessor of node i in the shortest path from s to i

(1) Initialization:

$$h^0(s):=0,\ h^0(j):=\infty \quad \forall j \in V\text{-}\{s\}; \qquad \pi(j):=j \quad \forall j \in V; \qquad k:=1$$

(2) $\forall j \in V$ compute $$h^k(j):=\min\left\{h^{k-1}(j),\ \min_{i:(i,j)\in E}\left[c_{ij}+h^{k-1}(i)\right)\right]\right\}$$

If $h^k(j)=h^{k-1}(i)+c_{ij}$, then $\pi(j):=i$.

(3) If $h^k(j)=h^{k-1}(j)\ \forall j \in V$,

then $g(j):= h^k(j)\ \forall j \in V$, the algorithm terminates, and the shortest s-t path is specified by the sequence of the labels $\pi(i)$.

Otherwise,

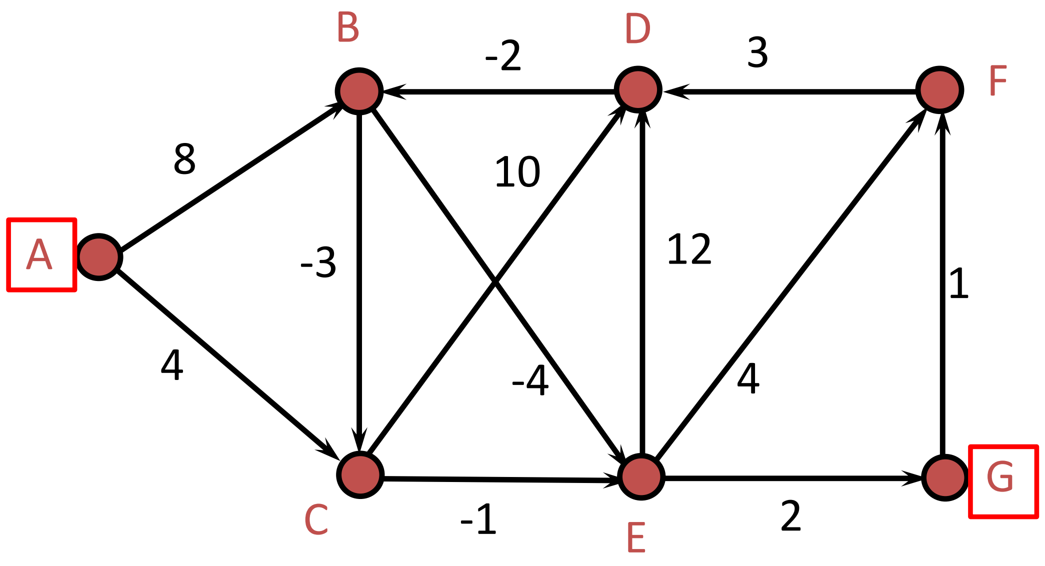
if $k<n$, then $k:=k+1$ and go to step (2);

if $k=n$, then the graph contains a negative cycle, the algorithm terminates, and the problem is unbounded.

Tabular implementation of the Bellman-Ford Algorithm

Example: find the shortest path from A to G in the following directed graph

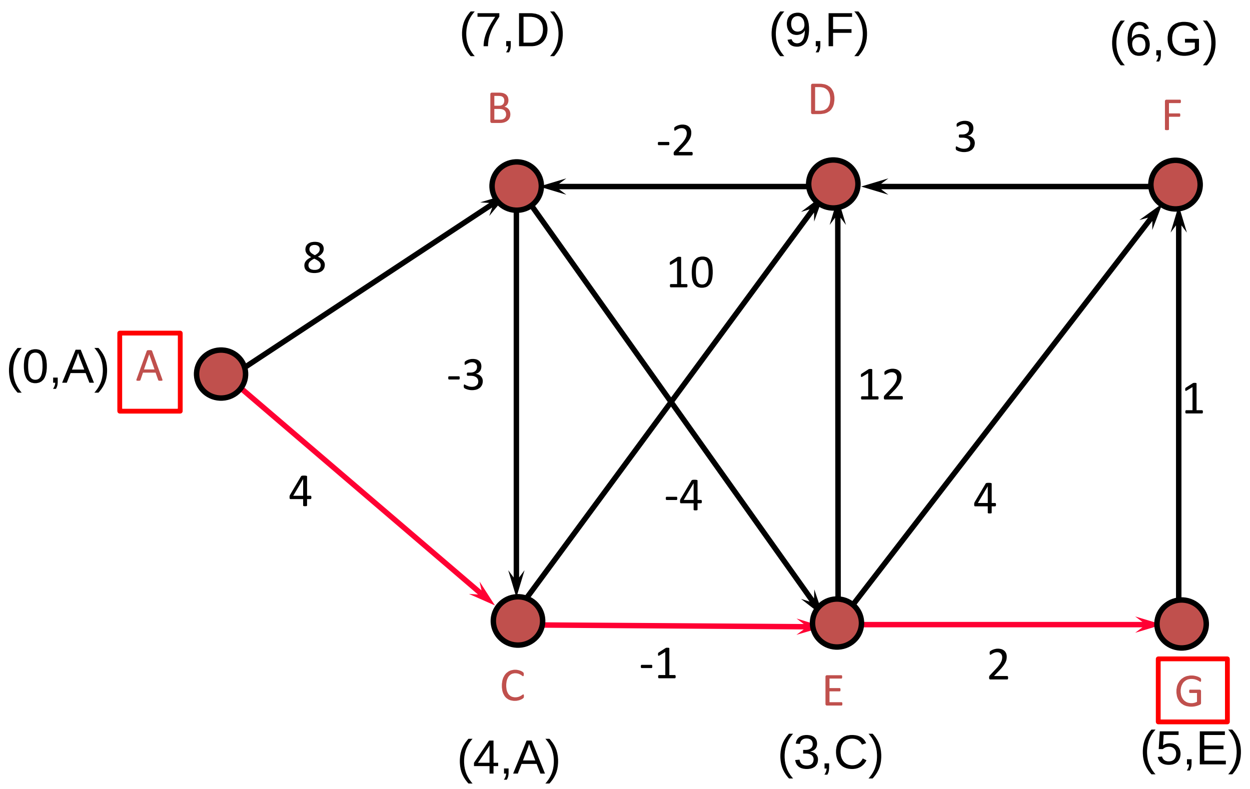
n=7, m=12



k	A	B	C	D	E	F	G
0	(0,A)	(∞,B)	(∞,C)	(∞,D)	(∞,E)	(∞,F)	(∞,G)
1	(0,A)	(8,A)	(4,A)	(∞,D)	(∞,E)	(∞,F)	(∞,G)
2	(0,A)	(8,A)	(4,A)	(14,C)	(3,C)	(∞,F)	(∞,G)
3	(0,A)	(8,A)	(4,A)	(14,C)	(3,C)	(7,E)	(5,E)
4	(0,A)	(8,A)	(4,A)	(10,F)	(3,C)	(6,G)	(5,E)
5	(0,A)	(8,A)	(4,A)	(9,F)	(3,C)	(6,G)	(5,E)
6	(0,A)	(7,D)	(4,A)	(9,F)	(3,C)	(6,G)	(5,E)

The shortest path is A-C-E-G

Iteration $k = 7$: no further modification, the paths are optimal



Remarks

Likewise the Dijkstra Algorithm, the Bellman-Ford Algorithm is a "**labelling algorithm**": it associates a label to each node and **at each iteration it checks whether the labels need to be updated.**

Since at each iteration it "calls into question" the choices made at the previous iterations, **it is <u>not</u> greedy**.

Label of a node: cost of the path from s to i at step k.

In the worst case, the Bellman-Ford Algorithm processes all the arcs and does this n-1 times.

The repeated processing of the arcs allows to propagate the shortest distances through the graph.

Since there is no negative cycle, the shortest path visits each node at most once.

If a graph contains a negative cycle, it is found and the algorithm terminates.

Arborescence of shortest paths

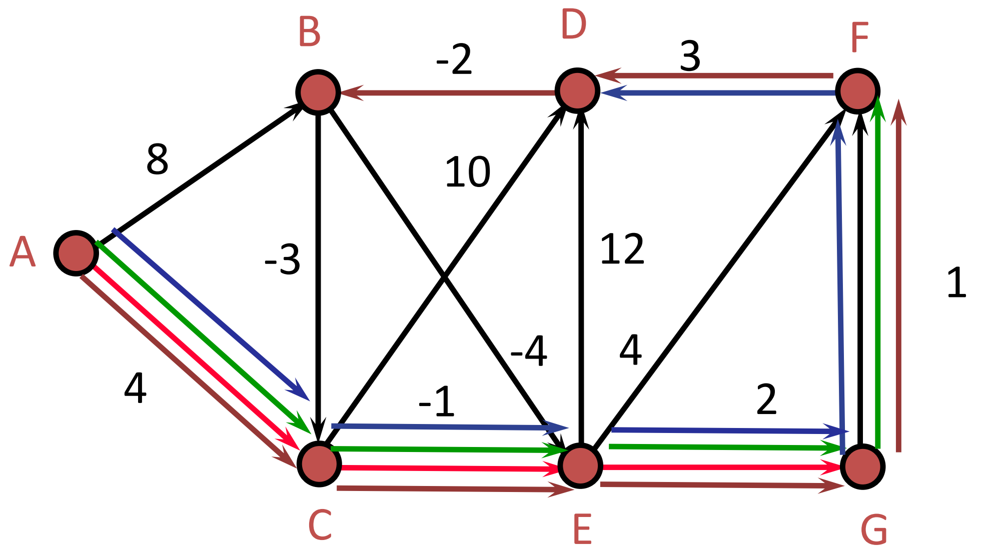
Likewise the Dijkstra Algorithm, the Bellman-Ford Algorithm:

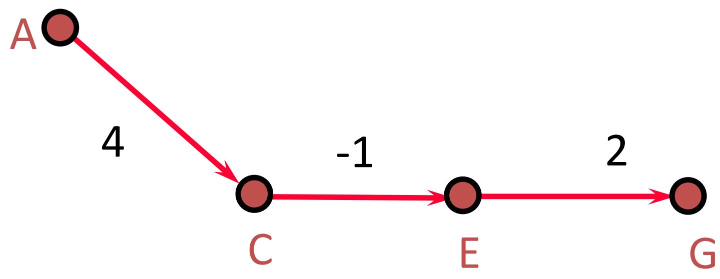
- originates as “one-to-one”, but it provides as by-products the shortest paths from s to every node of the graph (“**arborescence of shortest paths**”);

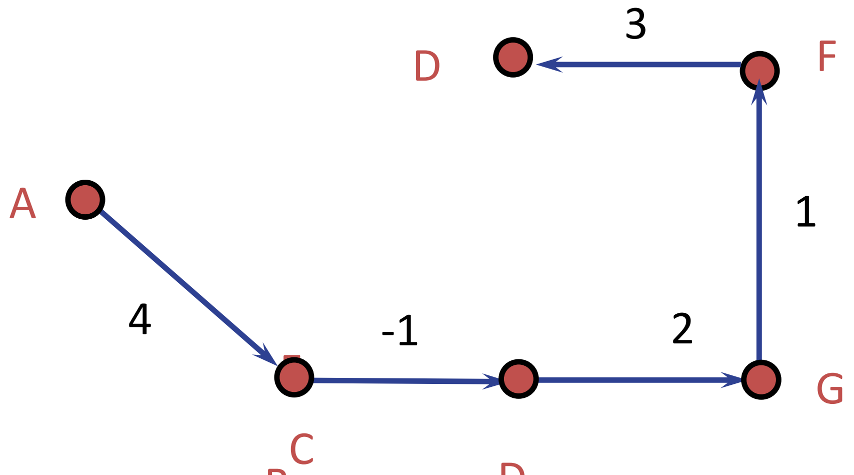
- hence, it is an algorithm “**one-to-all**”;

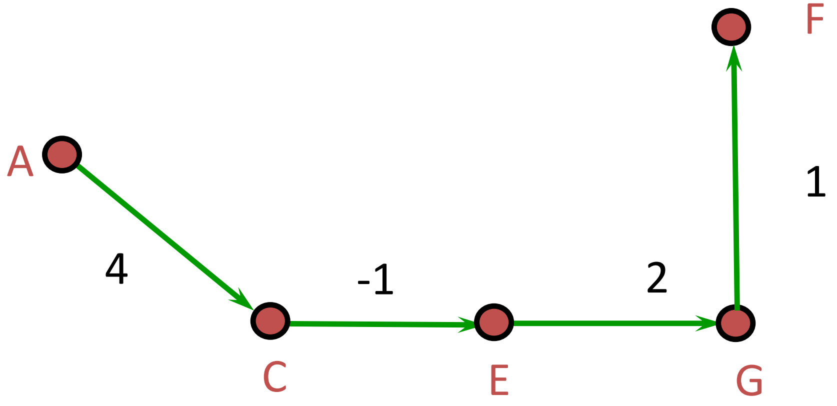
- to find the shortest paths between every pair of nodes it must be applied n times, each time changing the source.

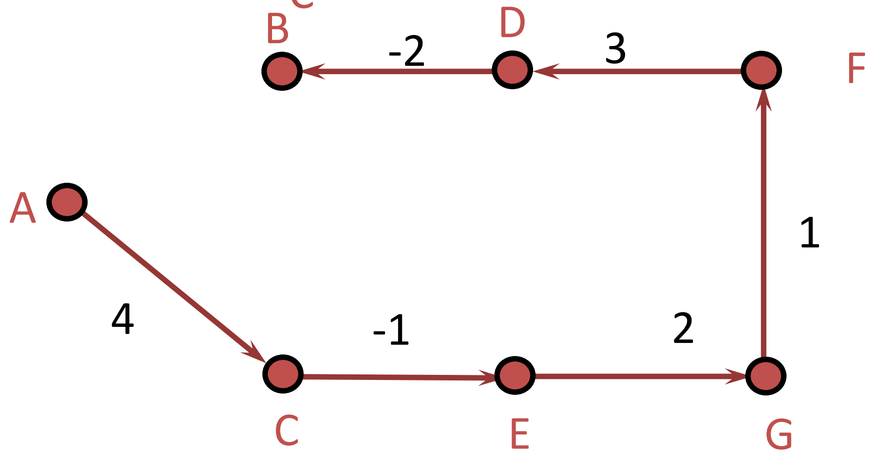
k	A	B	C	D	E	F	G
0	(0,A)	(∞,B)	(∞,C)	(∞,D)	(∞,E)	(∞,F)	(∞,G)
1	(0,A)	(8,A)	(4,A)	(∞,D)	(∞,E)	(∞,F)	(∞,G)
2	(0,A)	(8,A)	(4,A)	(14,C)	(3,C)	(∞,F)	(∞,G)
3	(0,A)	(8,A)	(4,A)	(14,C)	(3,C)	(7,E)	(5,E)
4	(0,A)	(8,A)	(4,A)	(10,F)	(3,C)	(6,G)	(5,E)
5	(0,A)	(8,A)	(4,A)	(9,F)	(3,C)	(6,G)	(5,E)
6	(0,A)	(7,D)	(4,A)	(9,F)	(3,C)	(6,G)	(5,E)




A
4
C
-1
E
2
G

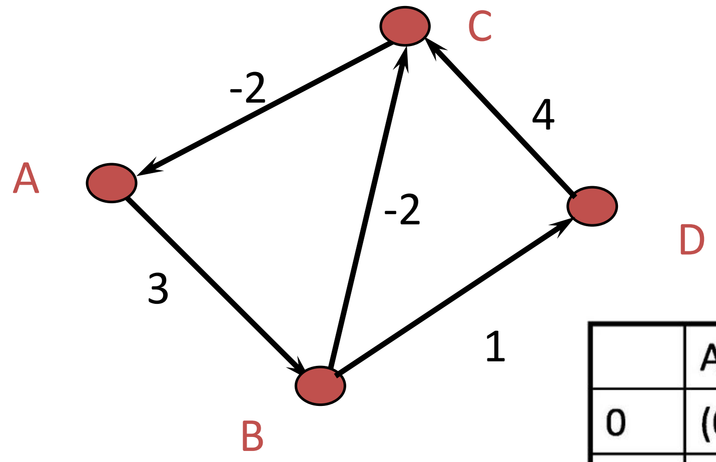

D
3
F
A
1
4
-1
2
C
G


F
A
1
4
-1
2
C
E
G


B
-2
D
3
F
A
1
4
-1
2
C
E
G

Example of unbounded solution

Directed graph with a negative cycle:



Find the shortest path between A and D

	A	B	C	D
0	(0,A)	(∞,B)	(∞,C)	(∞,D)
1	(0,A)	(3,A)	(∞,C)	(∞,D)
2	(0,A)	(3,A)	(1,B)	(4,B)
3	(-1,C)	(3,A)	(1,B)	(4,D)
4	(-1,C)	(2,B)	(1,B)	(4,D)
5	…	…	…	…

The paths are continuously updated by the algorithm!

Complexity of the Bellman-Ford Algorithm

(1) Initialization:

$$h^0(s):=0,\ h^0(j):=\infty\quad \forall j \in V\text{-}\{s\};\ \pi(j):=j\quad \forall j \in V;\ \ k:=1$$

(2) For $\forall j \in V$ compute $h^k(j):=min\left\{h^{k-1}(j), \min_{i:(i,j)\in E}\left[c_{ij}+h^{k-1}(i)\right)\right]\right\}$

If $h^k(j)=h^{k-1}(i)+c_{ij}$, then $\pi(j):=i$.

(3) If $h^k(j)=h^{k-1}(j)\ \ \forall j \in V$,

then $g(j):= h^k(j)\ \ \forall j \in V$, the algorithm terminates, and the shortest s-t path is specified by the sequence of the labels $\pi(i)$.

Otherwise,

if $k<n$, then $k:=k+1$ and go to step (2);

if $k=n$, then the graph contains a negative cycle, the algorithm terminates, and the problem is unbounded.

Likewise in the Dijkstra Algorithm, there are ***n*** **iterations**.

At each iteration, one has to **check whether the labels must be updated**. The **checks** (and possible updates) to be done at each iteration are **at most m**.

Hence, the worst-case complexity is ***O(n m)***.

Since, usually, $m>n$, the complexity is typically larger that for the Dijkstra Algorithm.

The Floyd-Warshall Algorithm

The Dijkstra Algorithm and the Bellman-Ford Algorithm provide the shortest paths between a fixed source and every other node.

To obtain the shortest paths between every pair of nodes they must be applied n times, each time changing the source node.

There exists an algorithm that provides directly the matrix of the shortest paths between every pair if nodes and their associated costs: "**all-pair shortest paths**".

It is the **Floyd-Warshall Algorithm**, which is "**all-to-all**".

The Floyd-Warshall Algorithm executes as many iterations as the number of nodes, iteratively updating the matrix of the shortest paths.

Basic idea: at the generic iteration k one evaluates whether the paths so far computed can be shortened via the inclusion of node k as intermediate node.

Notations

For simplicity, in the following we omit the iteration index in the matrices and in their elements.

$C := [C_{ij} : i,j \in V]$ **cost matrix**

C_{ij} is the cost of the shortest path from i to j at the current iteration

$\prod := [\prod_{ij}, i,j \in V]$ **predecessor matrix**

$\prod_{ij}$ is the node immediately preceding j in the shortest path from i to j at the current iteration.

distanza minore con il nodo subito prima precedente

1. Initialization of the matrices C and $\prod$

$C_{ii}:=0 \;\forall i \in V$, C_{ij} := cost of arc (i,j) if $(i,j) \in E$, $C_{ij}:=\infty$ if $(i,j) \notin E$

$\prod_{ij}:=i \;\forall i,j \in V$ provo le altre possibilita per vedere se c'è un miglioramento

$k:=1$

2. Triangular operation for node k

$\forall i \neq k$ such that $C_{ik} \neq \infty$ and $\forall j \neq k$ such that $C_{kj} \neq \infty$

if $C_{ik} + C_{kj} < C_{ij}$

update the cost matrix: $C_{ij} := C_{ik} + C_{kj}$

update the predecessor matrix: $\prod_{ij} := \prod_{kj}$

3. If there exists i such that $C_{ii} < 0$, then there is a negative cycle that includes i, the problem is unbounded, and the algorithm stops.

If $\forall i \; C_{ii} \geq 0$ and $k = n$, then the matrices $\prod$ and C provide the shortest paths between every pair of nodes and their costs, respectively. The algorithm stops.

If $\forall i \; C_{ii} \geq 0$ and $k < n$, then $k := k+1$ and go to step 2.

1. Initialization of the matrices C and $\prod$

C_{ii}:=0 $\forall\, i \in V$, C_{ij} := cost of the arc (i,j) if $(i,j) \in E$,

C_{ij}:=∞ if $(i,j) \notin E$

$\prod_{ij}$:=i $\forall\, i,j \in V$

k:=1

2. Triangular operation for node k

$\forall i \neq k$ such that $C_{ik} \neq \infty$ and $\forall j \neq k$ such that $C_{kj} \neq \infty$

if $C_{ik} + C_{kj} < C_{ij}$

update the cost matrix:

$$C_{ij} := C_{ik} + C_{kj}$$

update the predecessor matrix:

$$\Pi_{ij} := \Pi_{kj}$$

3. If there exists i such that $C_{ii} < 0$, then there is a negative cycle that includes i, the problem is unbounded, and the algorithm stops.

If $\forall i\ C_{ii} \geq 0$ and $k = n$, then the matrices $\prod$ and C provide the shortest paths between every pair of nodes and their costs, respectively. The algorithm stops.

If $\forall i\ C_{ii} \geq 0$ and $k < n$, then $k := k+1$ and go to step 2.

Remarks

The computations can be performed directly on the matrices C and $\prod$.

The **presence of a negative cycle** containing node i is revealed by the appearance of a **negative value in the entry** $\boldsymbol{C_{ii}}$ of the matrix C. In such a case, the problem is unbounded.

The shortest paths are obtained by **following backward the sequence defined by the matrix $\prod$**:

$\prod_{ij}=w$, $\prod_{iw}=z$, $\prod_{iz}=k$,...., $\prod_{ih}=i$

By initializing the matrix C with $C_{ii}:=\infty$, if there exists an unbounded solution then the value C_{ii} in the final matrix C represents the cost of the minimum-cost cycle that goes through node i.

Complexity of the Floyd-Warshall Algorithm

- The algorithm executes ***n* iterations** of this kind:

2. Triangular operation for node k

```
$\forall i \neq k$ such that $C_{ik} \neq \infty$ and $\forall j \neq k$ such that $C_{kj} \neq \infty$
    if $C_{ik} + C_{kj} < C_{ij}$
        update the cost matrix: $C_{ij} := C_{ik} + C_{kj}$
        update the predecessor matrix: $\prod_{ij} := \prod_{kj}$
```

- **In each iteration the number of updates** is bounded from above by the number of node pairs, i.e., ***O(n²)***.

- Hence, the complexity is ***O(n³)***.

1. Initialization of the matrices C and $\prod$

   $C_{ii}:=0 \ \forall i \in V$, $C_{ij} :=$ cost of arc (i,j) if $(i,j) \in E$, $C_{ij}:=\infty$ if $(i,j) \notin E$

   $\prod_{ij}:=i \ \forall i,j \in V$

   $k:=1$

2. Triangular operation for node k

```
$\forall i \neq k$ such that $C_{ik} \neq \infty$ and $\forall j \neq k$ such that $C_{kj} \neq \infty$
    if $C_{ik} + C_{kj} < C_{ij}$
        update the cost matrix: $C_{ij} := C_{ik} + C_{kj}$
        update the predecessor matrix: $\prod_{ij} := \prod_{kj}$
```

3. If there exists i such that $C_{ii} < 0$, then there is a negative cycle that includes i, the problem is unbounded, and the algorithm stops.

   If $\forall i \ C_{ii} \geq 0$ and $k = n$, then the matrices $\prod$ and C provide the shortest paths between every pair of nodes and their costs, respectively. The algorithm stops.

   If $\forall i \ C_{ii} \geq 0$ and $k < n$, then $k := k+1$ and go to step 2.

The next theorem guarantees that the Floyd-Warshall Algorithm provides the all-pair shortest paths.

Theorem. Let $\prod$ and C be the matrices obtained after executing the triangular operation for node k. Then, for every $i,j \in V$, $\prod_{ij}$ and C_{ij} represent, respectively, the node immediately preceding j in the shortest path from i to j and the cost of the shortest path from i to j in the subgraph induced by the set of nodes $\{1,\dots,k\} \cup \{i,j\}$.

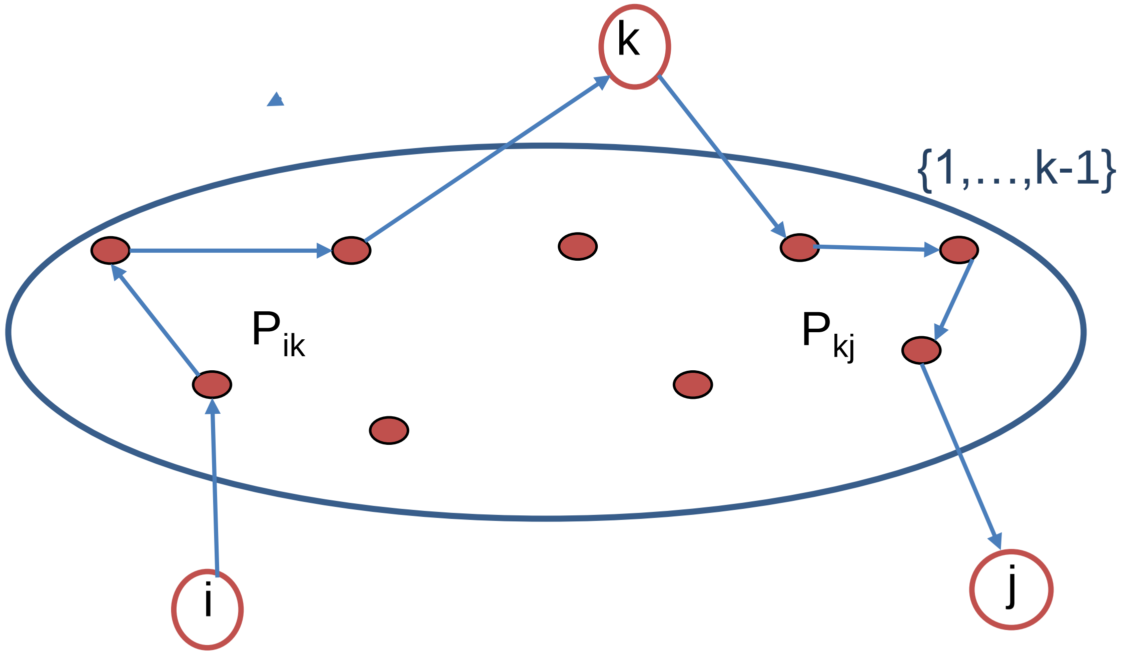
Proof. By induction on k.

For $k=0$ the matrix C coincides with the matrix of the arc costs. Hence, C_{ij} represents the cost of the shortest path from i to j in the subgraph induced by $\{i,j\}$.

Now, let us suppose by induction that the statement of the theorem holds at iteration k-1. Let us consider a shortest path P_{ij} from i to j in the subgraph indiced by $\{1,\dots,k\} \cup \{i,j\}$. Let $c(P_{ij})$ be its cost. There are two possible cases.

(a) P_{ij} does not go through node k. Then $c(P_{ij})= C_{ij}$ by the inductive hypothesis.

(b) P_{ij} goes through node k. In such a case, P_{ij} can be partitioned into $P_{ik} \cup P_{kj}$, where P_{ik} is a shortest path from i to k in the subgraph induced by the nodes $\{1, \ldots, k-1\} \cup \{i,k\}$ and P_{kj} is a shortest path from k to j in the subgraph induced by the nodes $\{1,\ldots,k-1\} \cup \{k,j\}$.



By the inductive hypothesis, $c(P_{ik}) = C_{ik}$ and $c(P_{kj})=C_{kj}$. Hence

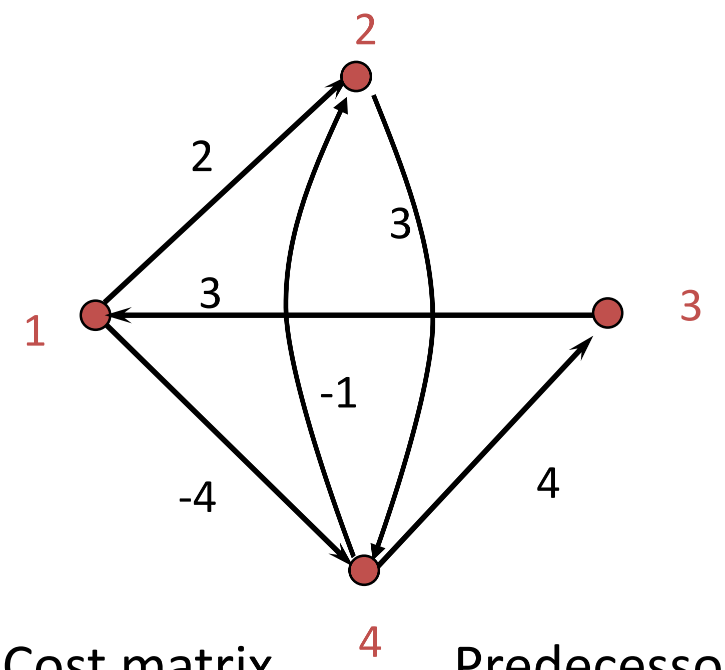
$$c(P_{ij}) = min\,\{C_{ij}, C_{ik} + C_{kj}\}. \qquad ▪$$

ad ogni iterazione a abbiamo la strada piu corta in un certo sottografo

The theorem omplies the following:

at iteration k the matrices $\prod$ and C contain the shortest paths between every pair of nodes and their costs, respectively, **when only the nodes *{1,2,...,k}* are considered as possible intermediate nodes**.

Example



Intialization

Cost matrix

0	2	99	-4
99	0	99	3
3	99	0	99
99	-1	4	0

Predecessor matrix

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

k=1

Cost matrix

0	2	99	-4
99	0	99	3
3	5	0	-1
99	-1	4	0

Predecessor matrix

1	1	1	1
2	2	2	2
3	1	3	1
4	4	4	4

k=2

Cost matrix

0	2	99	-4
99	0	99	3
3	5	0	-1
98	-1	4	0

Predecessor matrix

1	1	1	1
2	2	2	2
3	1	3	1
2	4	4	4

k=3

Cost matrix

0	2	99	-4
99	0	99	3
3	5	0	-1
$\textcircled{7}$	-1	4	0

Predecessor matrix

1	1	1	1
2	2	2	2
3	1	3	1
$\textcircled{3}$	4	4	4

k=4

Cost matrix

0	$\boxed{-5}$	$\boxed{0}$	-4
$\boxed{10}$	0	$\boxed{7}$	3
3	$\boxed{-2}$	0	-1
7	-1	4	0

Predecessor matrix

1	$\boxed{4}$	$\boxed{4}$	1
$\boxed{3}$	2	$\boxed{4}$	2
3	$\boxed{4}$	3	1
3	4	4	4