2.3 Optimal paths

Optimal (shortest or longest) paths have a wide range of applications:

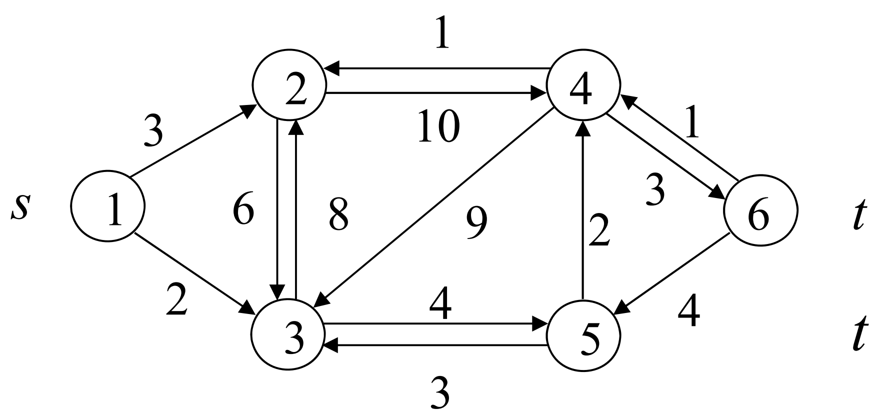
- Google maps, GPS navigators
- planning and management of transportation, electrical and telecommunication networks
- project planning
- VLSI design
- subproblems of more complex problems
- ...

2.3.1 Shortest path problem

Problem

Given a directed graph $G = (N, A)$ with a cost $c_{ij} \in \mathbb{R}$ for each arc $(i, j) \in A$, and two nodes s and t, determine a minimum cost (shortest) path from s to t.

s is the *origin*

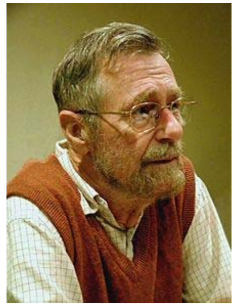


t is the *destination*

c_{ij} represents the cost (length, travel time,…) of arc $(i, j) \in A$

2.3.2 Dijkstra's algorithm

Assumption $c_{ij} \geq 0 \quad \forall (i,j) \in A$

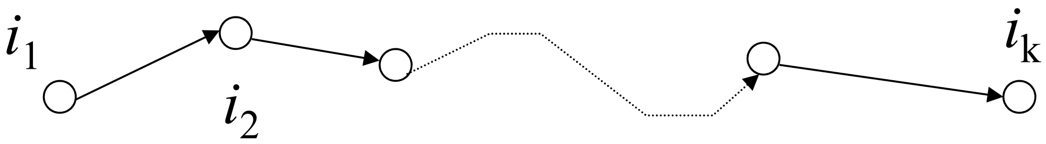


Edsger Dijkstra (1930-2002)

Definition: A path $(i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k)$ is ***simple*** if

$$i_u \neq i_v \text{ for all } u \neq v$$

(a node is visited at most once)



Property: If $c_{ij} \geq 0$ for all $(i,j) \in A$, every shortest path is simple.

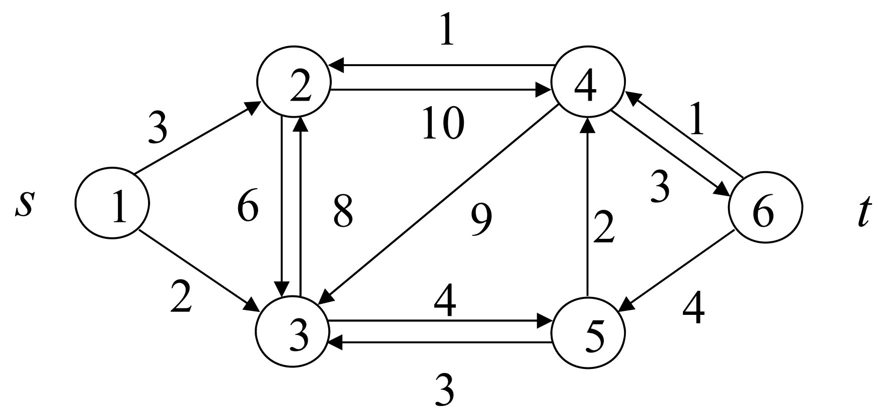
input

$G = (N, A)$ with $n = |N|$ and $m = |A|$, a node $s \in N$,

$c_{ij} \geq \underline{0} \quad \forall (i, j) \in A$ with $c_{ij} = +\infty$ if $(i, j) \notin A$

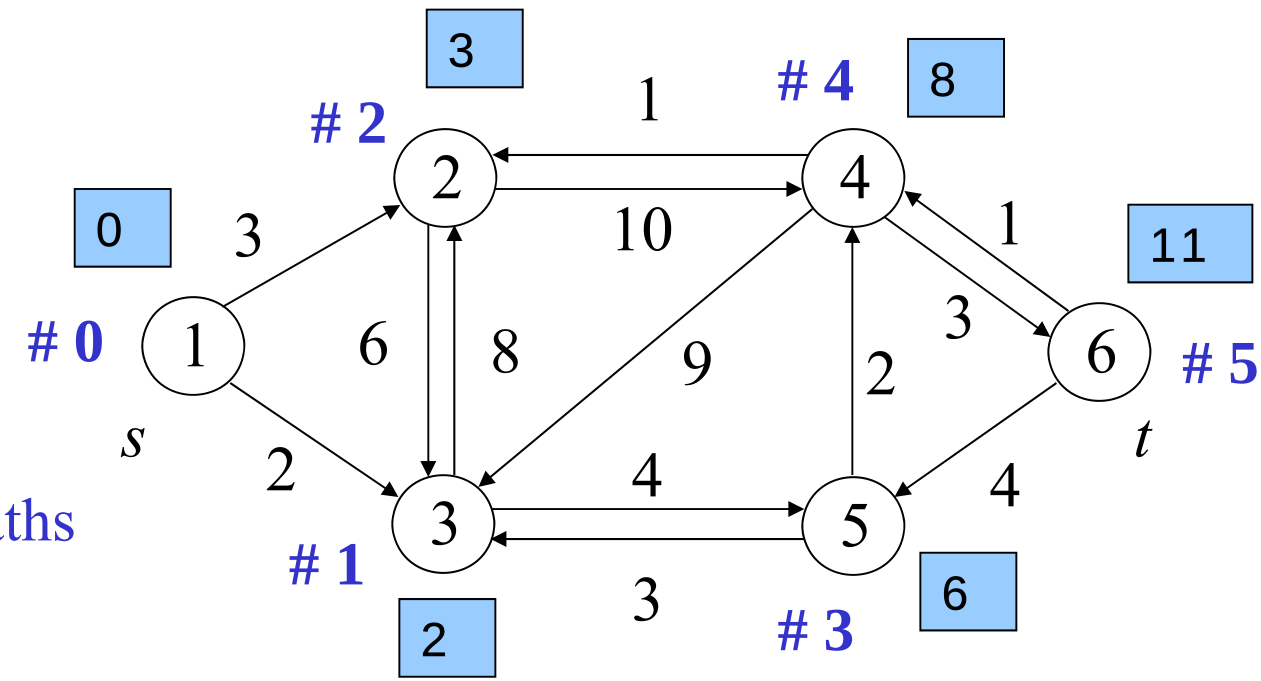
output

Shortest paths from s to <u>all other nodes</u> of G



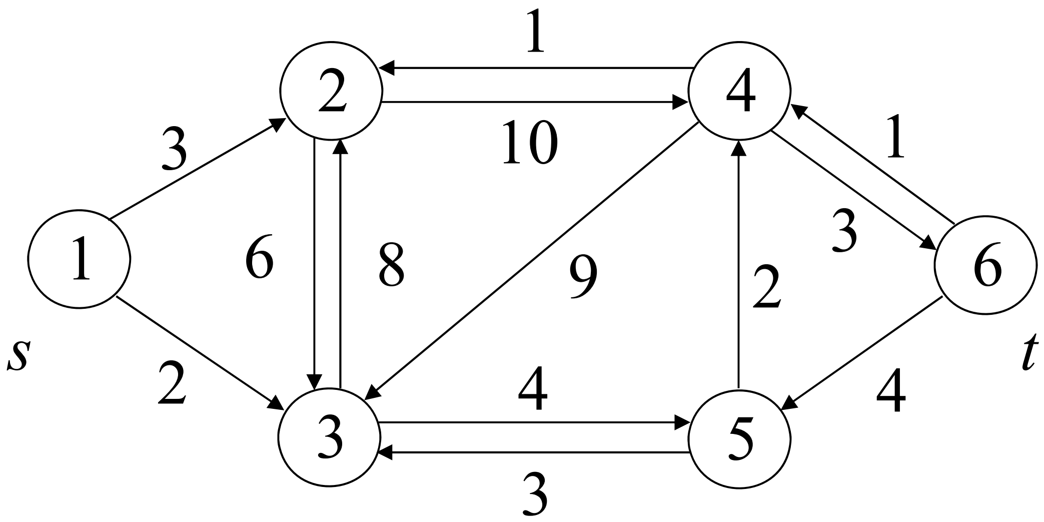
Idea: Consider the nodes in order of increasing cost of the shortest path from *s* to any one of the other nodes.

To each node $j \in N$, we assign a *label* $L[j]$ which corresponds, at the end of the algorithm, to the cost of a minimum cost path from *s* to *j*.



"Greedy" w.r.t. paths from *s* to *j* !

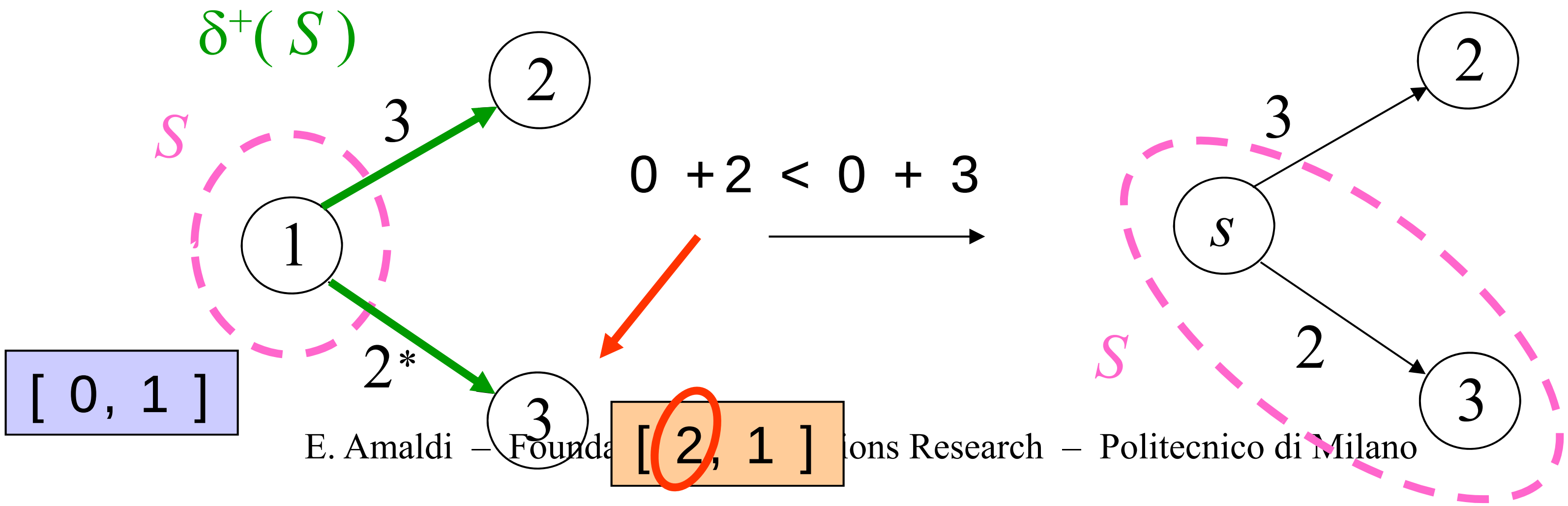
Example

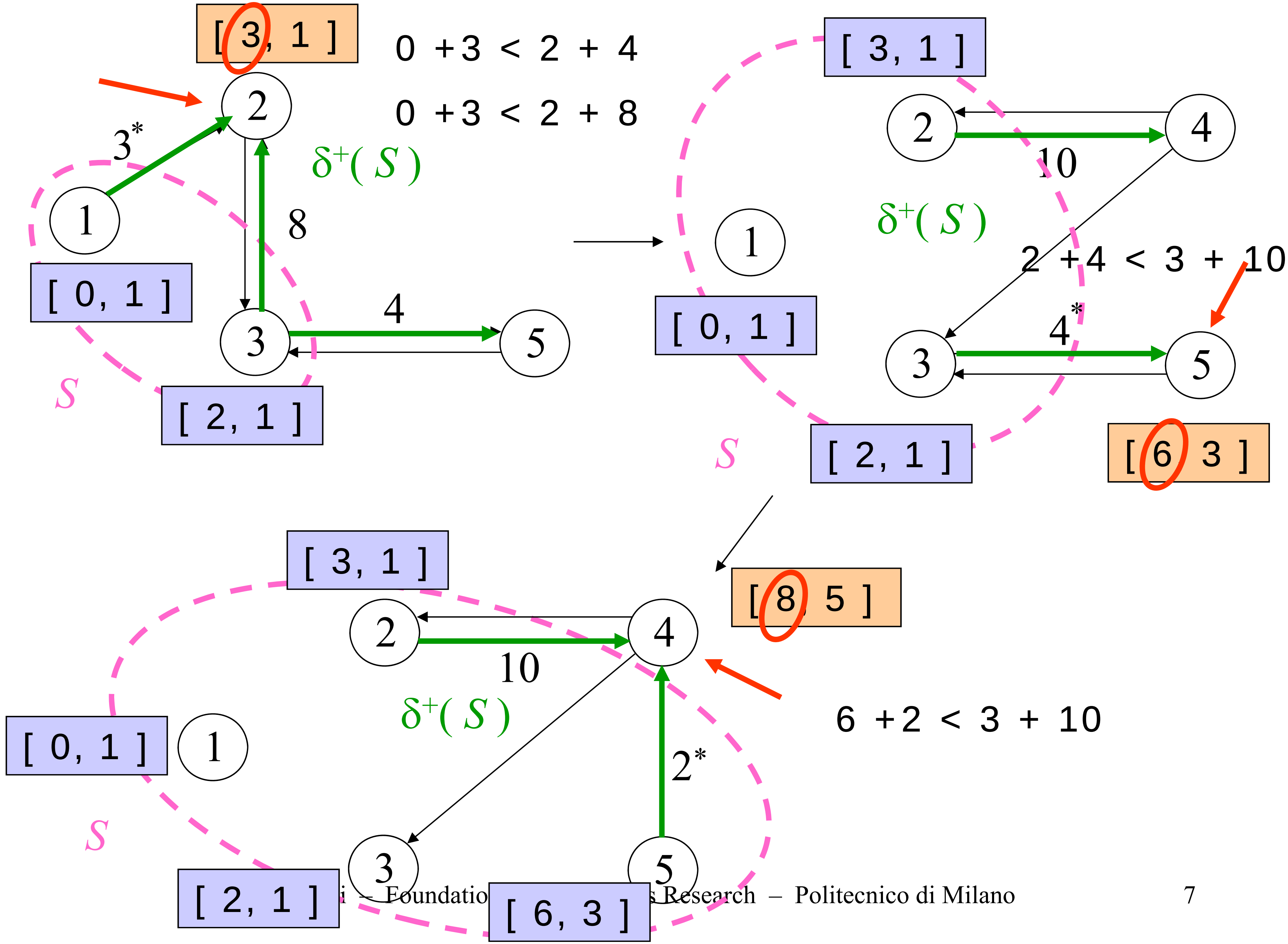


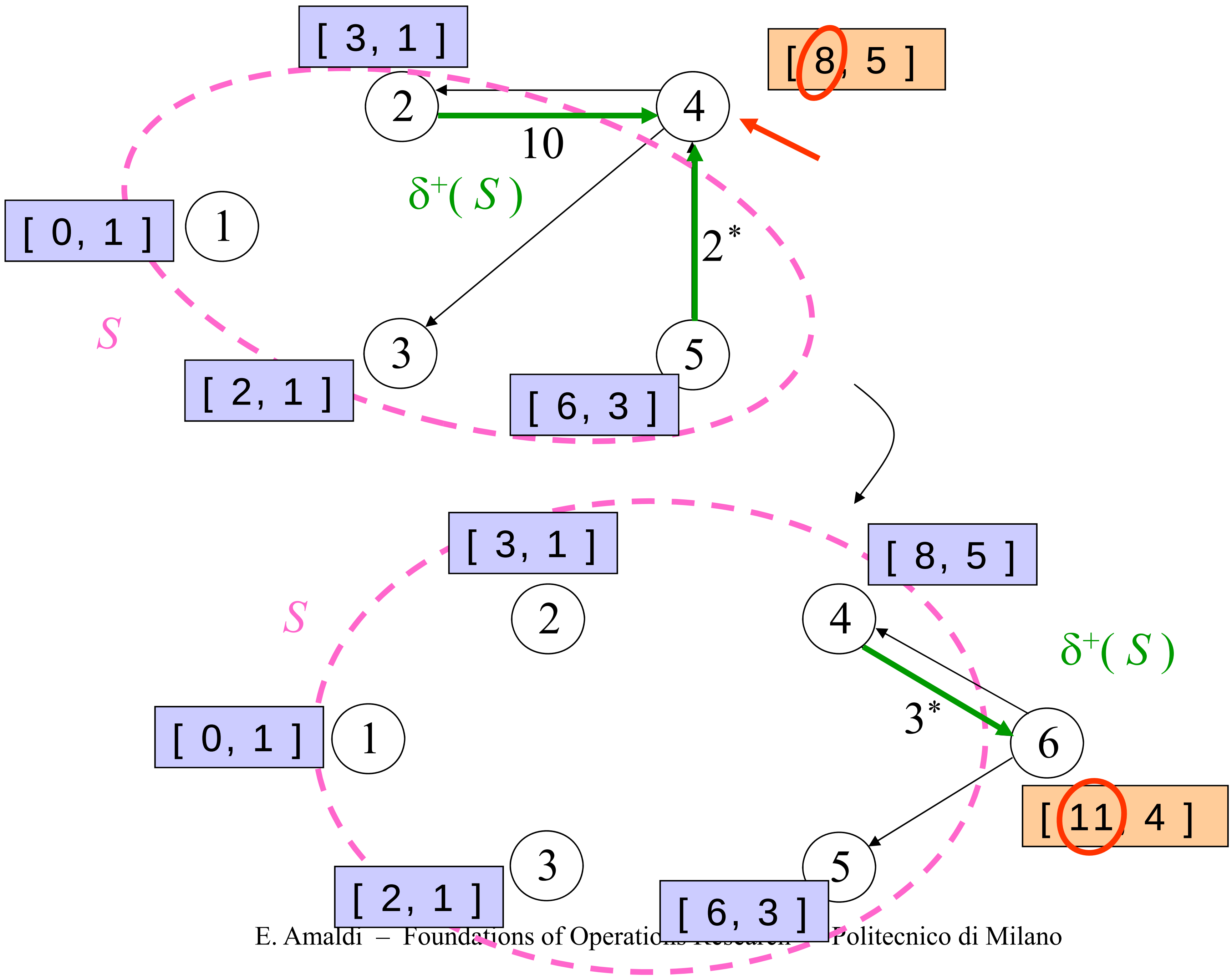
Two labels associated with each node j in S: [L[j], pred[j]]

where L[j] = cost of a shortest path from s to j,

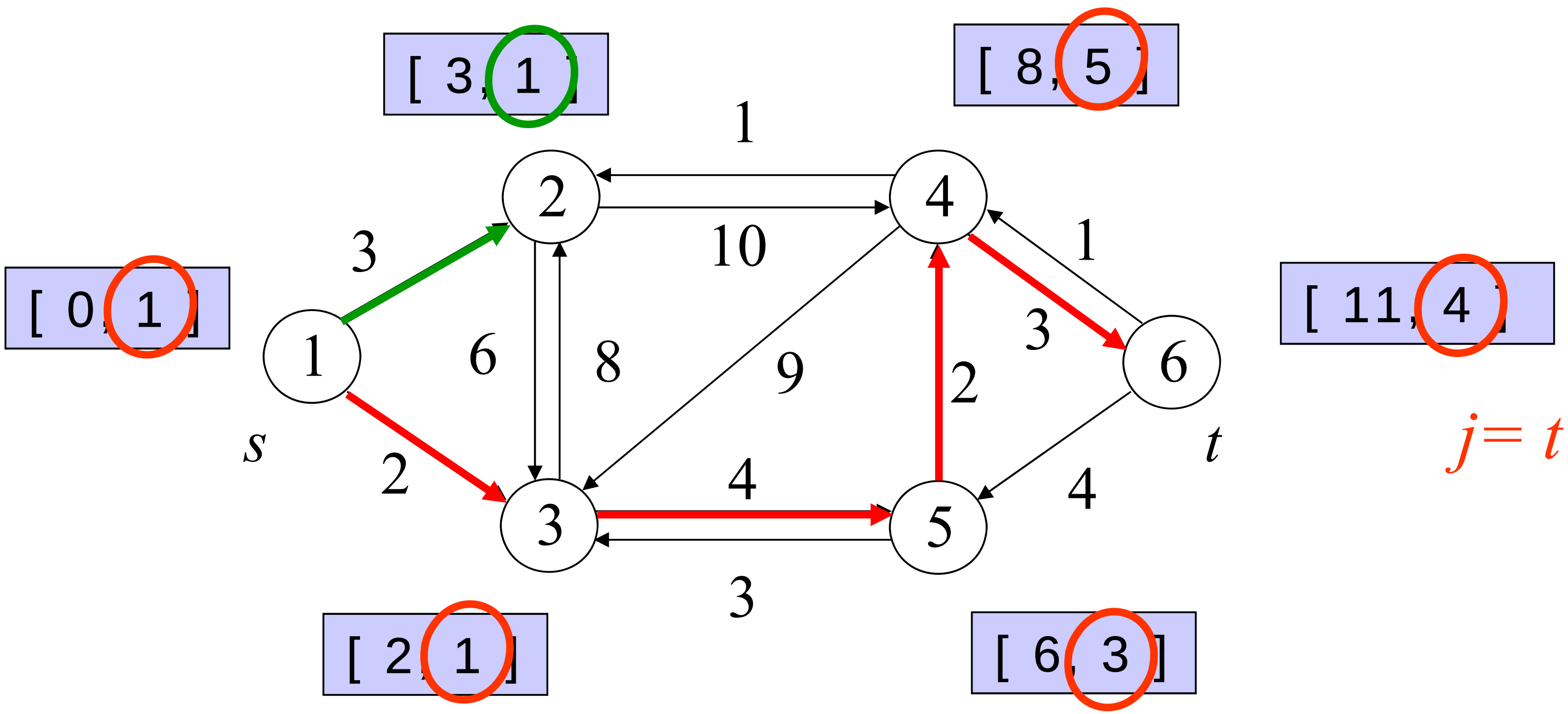
pred[j] = "predecessor" of j in the shortest path from s to j




[3, 1]
0 +3 < 2 + 4
0 +3 < 2 + 8
3*
8
4
δ+(S)
[0, 1]
[2, 1]
S
[3, 1]
10
δ+(S)
2 +4 < 3 + 10
4*
[0, 1]
[2, 1]
[6 3]
S
[3, 1]
[8 5]
10
δ+(S)
2*
6 +2 < 3 + 10
[0, 1]
S
[2, 1]
[6, 3]


[3, 1]
[8, 5]
2
4
10
δ+(S)
[0, 1]
1
2*
S
3
5
[2, 1]
[6, 3]
[3, 1]
[8, 5]
S
2
4
δ+(S)
[0, 1]
1
3*
6
[11, 4]
3
5
[2, 1]
[6, 3]

A set of shortest path from *s* to every other node *j* can be retrieved backwards: pred[*j*], pred[pred [*j*]],..., *s*



resulting path from *s* to *t*

resulting path from *s* to node 2

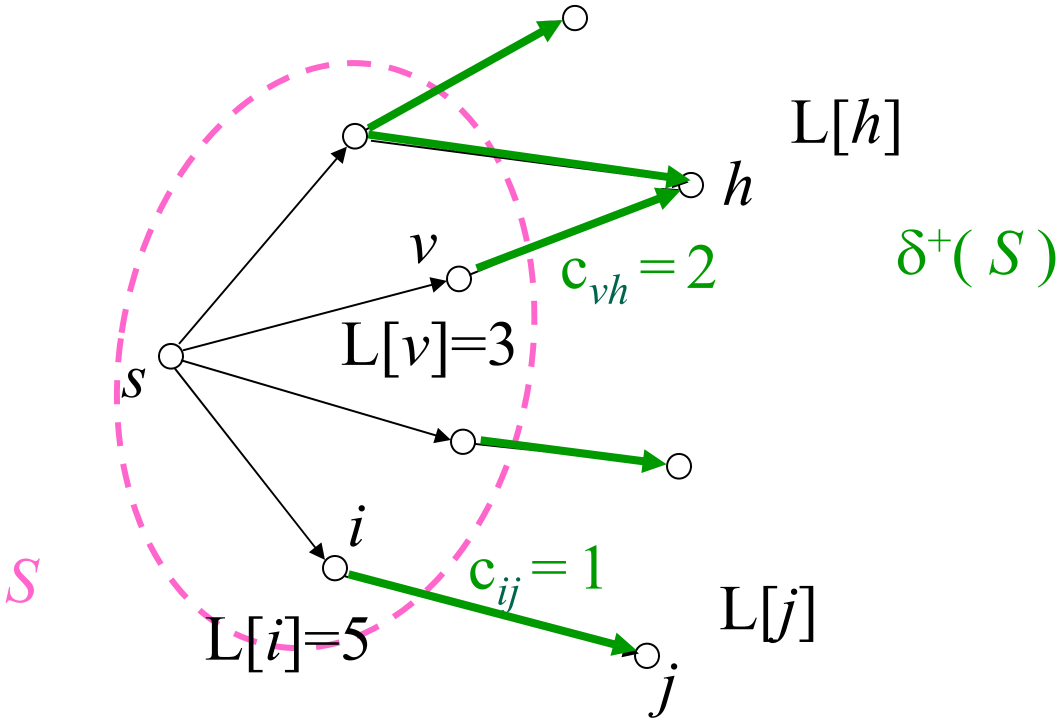
Dijkstra's algorithm

<u>Data structure</u>

- $S \subseteq N$ subset of nodes whose <u>labels</u> are <u>final</u>

- $$\mathrm{L}[j] = \begin{cases} \underline{\text{cost}} \text{ of a } \underline{\text{shortest path}} \text{ from } s \text{ to } j, & \forall\, j \in S \\ \min\{\, \mathrm{L}[i] + c_{ij} : (i, j) \in \delta^{+}(S) \,\}, & \forall\, j \notin S \end{cases}$$

…

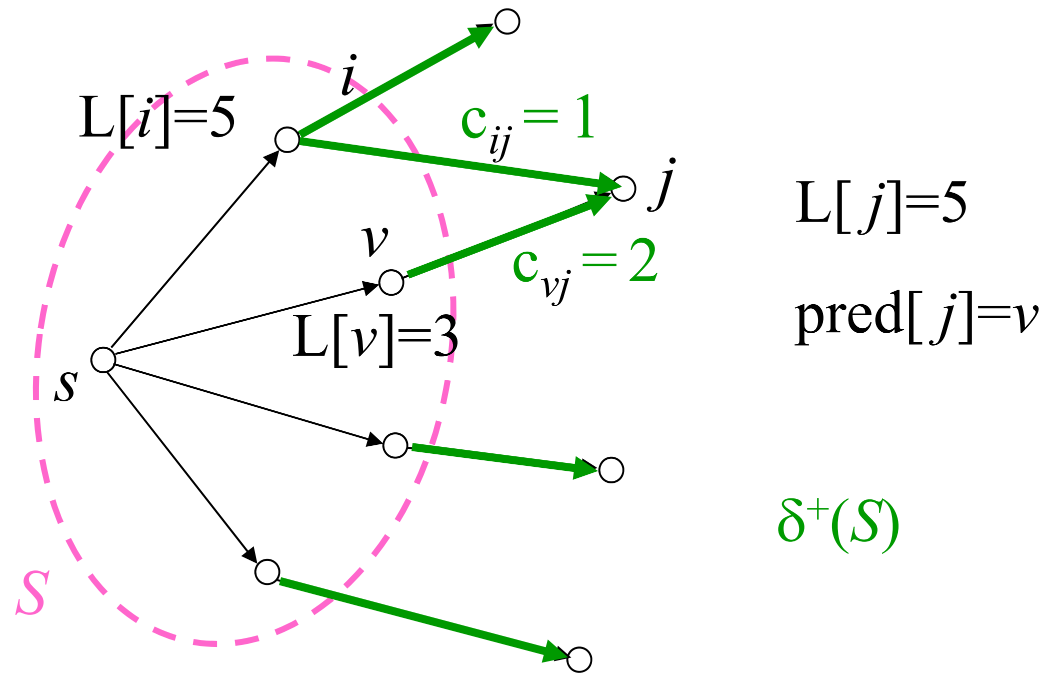


Given a directed graph G and the current subset of nodes $S \subset N$, consider the "outgoing" cut $\delta^+(S)$ and select $(v,h) \in \delta^+(S)$ such that:

$$L[v] + c_{vh} = \min \{ L[i] + c_{ij} : (i, j) \in \delta^+(S) \}$$

thus $L[v] + c_{vh} \leq L[i] + c_{ij} \quad \forall (i,j) \in \delta^+(S)$

- $\text{pred}[j] = \begin{cases} \text{"predecessor" of } j \text{ in the shortest path from } s \text{ to } j & \forall\, j \in S \\ v \text{ such that } L[v] + c_{vj} = \min \{L[i] + c_{ij} : i \in S\} & \forall\, j \notin S \\ \text{with } c_{ij} = \infty \text{ if } (i,j) \notin A & \end{cases}$



Pseudocode of Dijkstra's algorithm

input $G = (N, A)$, $n = |N|$, $m = |A|$, $s \in N$, $c_{ij} \geq 0 \;\; \forall\, (i, j) \in A$

output Shortest paths from s to <u>all the other nodes</u>

```
BEGIN
  S := {s}; L[s] := 0;
  pred[s] := s;
  WHILE |S| ≠ n DO
    Select (v,h) ∈ δ+(S) ={ (i,j) : (i,j)∈A, i∈S, j∉S} such that
            L[v] + c_vh = min { L[i] + c_ij : (i,j) ∈ δ+(S) };
    L[h] := L[v] + c_vh;
    pred[h] := v;
    S := S ∪ {h};
  END-WHILE
END
```

outgoing cut induced by S

N.B.: If $\delta^+(S)=\emptyset$, the algorithm ends: no $h \in N \setminus S$ is reachable from s

Complexity

Depends on how, at each iteration, the arc (v,h) is selected among those of the current outgoing cut $\delta^+(S)$.

If all m arcs are scanned and those that do not belong to $\delta^+(S)$ are discarded, the overall complexity would be $O(nm)$, hence $O(n^3)$ for dense graphs.

If all labels L[j] are determined by appropriate updates ($\approx$ Prim's algorithm), we need to consider a single arc of $\delta^+(S)$ for each node $j \notin S$ $\Rightarrow$ overall **complexity** $\boldsymbol{O(n^2)}$.

Proposition: Dijkstra's algorithm is exact.

Proof

At k-th step: $S = \{s, i_2, ..., i_k\}$ and

$$L[j] = \begin{cases} \underline{\text{cost}} \text{ of a } \underline{\text{minimum}} \text{ cost } \underline{\text{path}} \text{ from } s \text{ to } j, & \forall j \in S \\ \underline{\text{cost}} \text{ of a } \underline{\text{minimum}} \text{ cost } \underline{\text{path}} \text{ with } \underline{\text{all}} \text{ intermediate } \underline{\text{nodes}} \text{ in } \underline{S} & \forall j \notin S \end{cases}$$

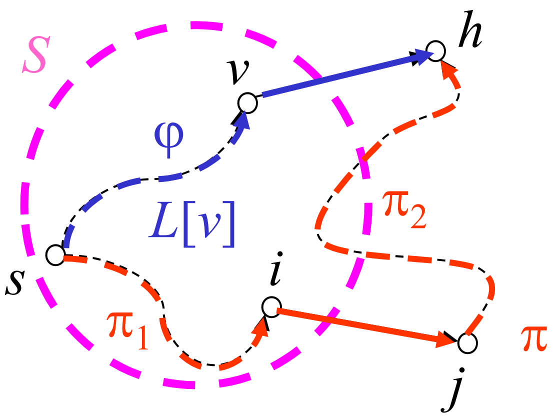
By induction on the number k of steps :

- *inductive basis* : it is true for $k = 1$ since
  $S = \{s\}$, $L[s] = 0$ and $L[j] = c_{sj}$ $\forall j \neq s$.
- *inductive step* : if it is true at the k-th step, it is also true at the (k+1)-th step.

(*k+1*)-th step: Let $h \notin S$ be the node that is inserted in S and φ the path from s to h such that:

$$L[v] + c_{vh} \leq L[i] + c_{ij} \quad \forall\, (i, j) \in \delta^+(S).$$

Let us verify that every path π from s to h has $c(\pi) \geq c(\varphi)$.



There exist $i \in S$ and $j \notin S$ such that

$$\pi = \pi_1 \cup (i, j) \cup \pi_2$$

where (i, j) is the first arc $\in \pi \cap \delta^+(S)$.

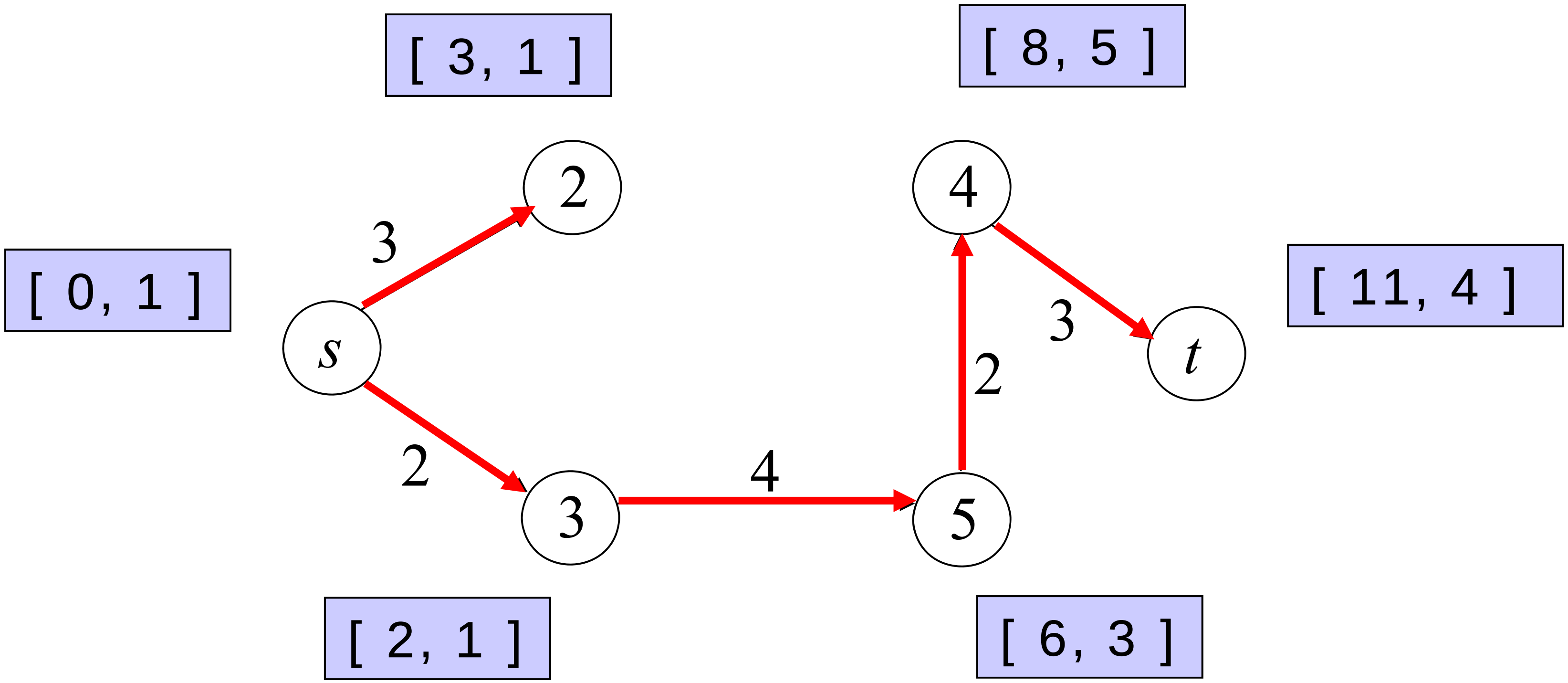
by choice of (v,h)

$$c(\pi) = c(\pi_1) + c_{ij} + c(\pi_2) \;\geq L[i] + c_{ij} \;\geq L[v] + c_{vh} = c(\varphi).$$

$c(\pi_1) \geq L[\, i \,]$: by induction assumption

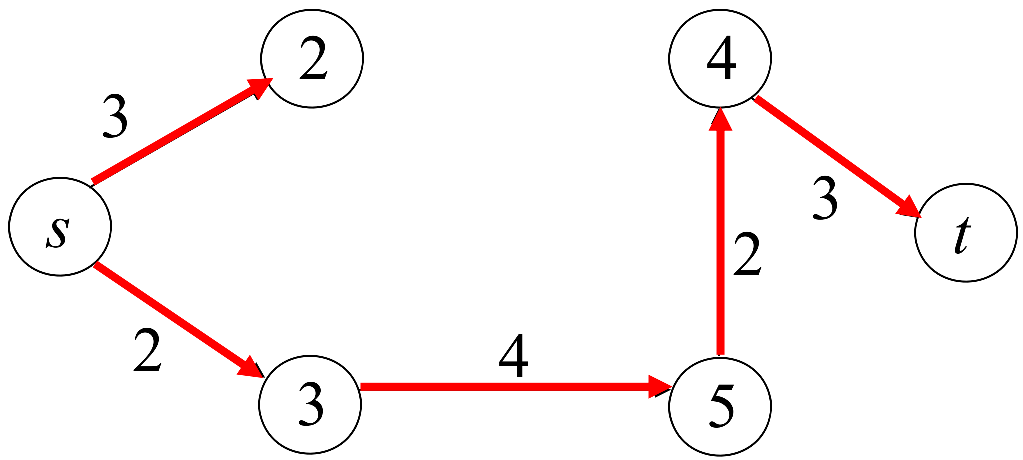
$c(\pi_2) \geq 0$ because $c_{ij} \geq 0$

A set of shortest paths from *s* to all the nodes *j* can be retrieved via the vector of predecessors.



Remarks

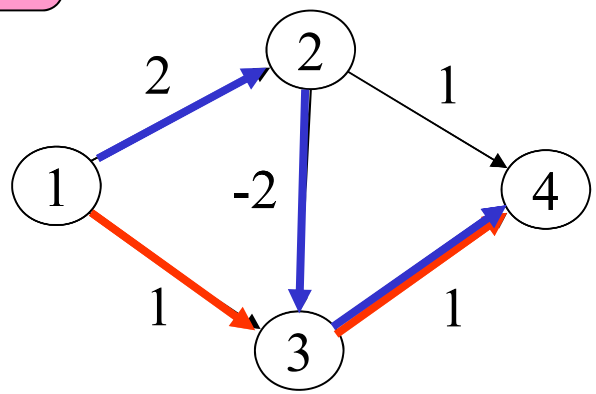
1) Taking the union of a set of shortest paths from node s to all the other nodes of G, we obtain an arborescence rooted at s.



Such arborescences, which are referred to as ***shortest path trees***, have nothing to do with minimum cost spanning trees!

2) Dijkstra's algorithm is not applicable when there are arcs with negative cost c_{ij}.

Example



Dijkstra's algorithm yields the path (1,3),(3,4) of cost 2,

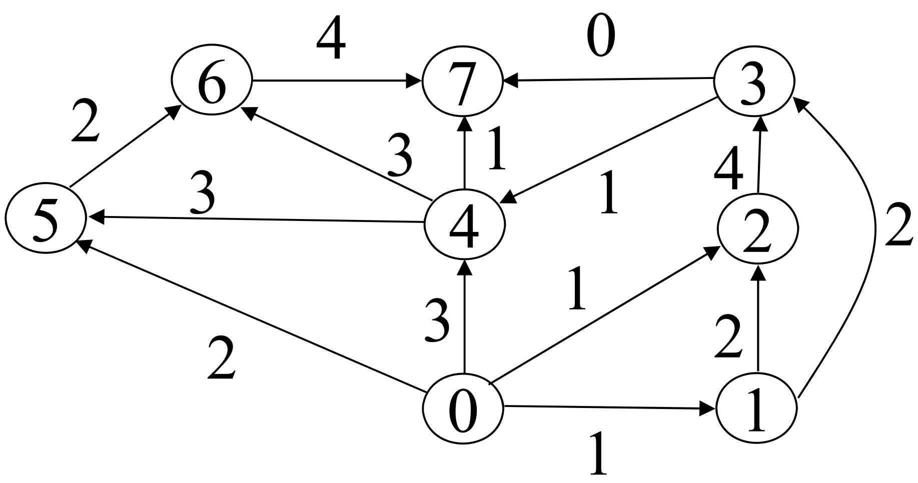
but path (1,2),(2,3),(3,4) has cost 1.

Due to $c_{23} < 0$ the last step in the exactness proof fails!

The minimum cost from 1 to 3 is not updated after the first step. According to a "greedy" choice on the arcs in $\delta^+(\{1\})$, it is taken as c_{13} which is "locally" optimal ($c_{13} < c_{12}$) even though the path (1,2),(2,3) has a strictly smaller cost because $c_{23} < 0$.

Exercise

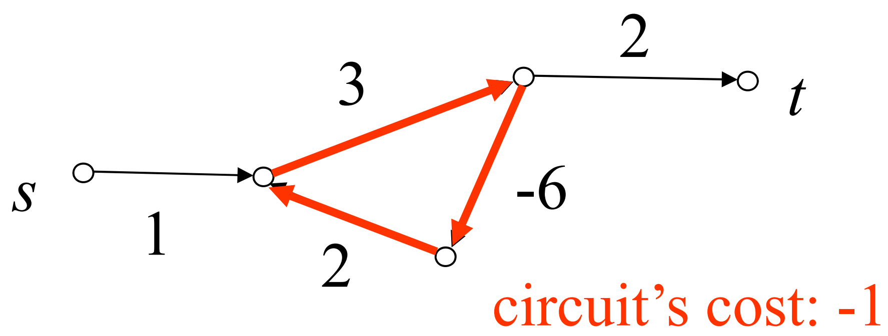
Determine the shortest paths form node 0 to all the other nodes of the following graph:



2.3.3 Shortest path problem with negative costs

Observation: If the graph *G* contains a circuit of negative cost, the shortest path problem may not be well-defined.

Example



Each time we go through the circuit, the cost decreases.
There is no *finite* shortest path from *s* to *t*.

Floyd-Warshall's algorithm allows to detect the presence of circuits with negative cost, and hence identify the cases in which the problem is ill-defined.

It provides a set of shortest paths between <u>all pairs of nodes</u>, even when there are arcs with <u>negative cost</u>.

It is based on iteratively applying a <u>triangular operation</u> described below.

Floyd-Warshall's algorithm

input

Directed graph $G = (N, A)$ described via the n x n cost matrix $C = [c_{ij}]$.

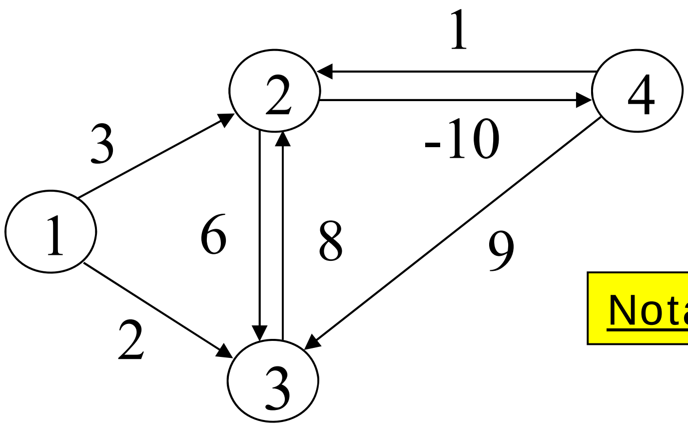
output

For each pair of nodes $i, j \in N$, the cost d_{ij} of shortest path from i to j.

Data structure: two n x n matrices D and P whose elements correspond, at the end of the algorithm, to

- d_{ij} = cost of a shortest path from i to j
- p_{ij} = predecessor of j in shortest path from i to j

Example



Notation: from to , from to .

	1	2	3	4
1	0	3	2	∞
2	∞	0	6	-10
3	∞	8	0	∞
4	∞	1	9	0

D

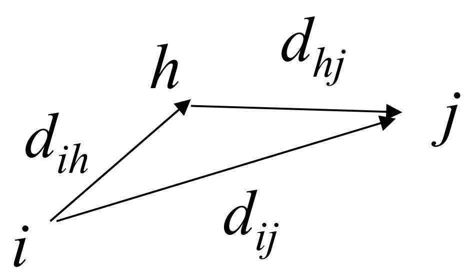
	1	2	3	4
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

P

For $(i, j) \in A$ set $d_{ij} = c_{ij}$, for loops $d_{ii} = 0$, and for $(i, j) \notin A$ set $d_{ij} = \infty$.

The matrix of predecessors is initialized with $p_{ij} = i$, for all i.

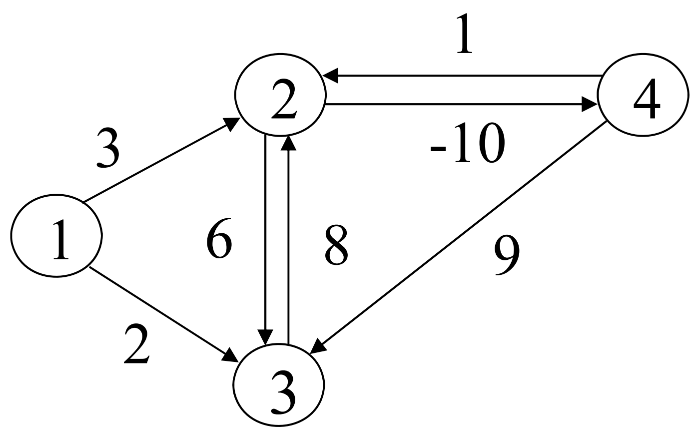
Triangular operation with respect to node h:



For each pair of nodes $i, j,$ with $i \neq h$ and $j \neq h$ (including case $i=j$), check whether when going from i to j it is more convenient to go via h :

$$d_{ih} + d_{hj} < d_{ij}$$

Cycle $h=1$: Since there are no such arcs, the matrices D and P remain unchanged.



Cycle for h=1

skip $i = 1$ and $j = 1$

D	~~1~~	2	3	4
~~1~~	0	3	2	∞
2	∞	0	6	-10
3	∞	8	0	∞
4	∞	1	9	0

P	~~1~~	2	3	4
~~1~~	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

$0 = d_{22} < d_{21} + d_{12} = \infty + 3 = \infty$	no update
$6 = d_{23} < d_{21} + d_{13} = \infty + 2 = \infty$	no update
$-10 = d_{24} < d_{21} + d_{14} = \infty + \infty = \infty$	no update
$8 = d_{32} < d_{31} + d_{12} = \infty + 3 = \infty$	no update
$0 = d_{33} < d_{31} + d_{13} = \infty + 2 = \infty$	no update
$\infty = d_{34} < d_{31} + d_{14} = \infty + \infty = \infty$	no update
$1 = d_{42} < d_{41} + d_{12} = \infty + 3 = \infty$	no update
$9 = d_{43} < d_{41} + d_{13} = \infty + 2 = \infty$	no update
$0 = d_{44} < d_{41} + d_{14} = \infty + \infty = \infty$	no update

Cycle for h=2

skip $i = 2$ and $j = 2$

D	1	$\cancel{2}$	3	4
1	0	3	2	$\cancel{\infty}$ -7
$\cancel{2}$	∞	0	6	-10
3	∞	8	0	$\cancel{\infty}$ -2
4	∞	1	$\cancel{9}$ 7	$\cancel{0}$ -9

→ HALT!

P	1	$\cancel{2}$	3	4
1	1	1	1	$\cancel{1}$ 2
$\cancel{2}$	2	2	2	2
3	3	3	3	$\cancel{3}$ 2
4	4	4	$\cancel{4}$ 2	4

$0 = d_{11} < d_{12} + d_{21} = 3 + \infty = \infty$	no update
$2 = d_{13} < d_{12} + d_{23} = 3 + 6 = 9$	no update
$\boldsymbol{\infty = d_{14} > d_{12} + d_{24} = 3 - 10 = -7}$	$\boldsymbol{p_{14} := p_{24} = 2}$
$\infty = d_{31} = d_{32} + d_{21} = 8 + \infty = \infty$	no update
$0 = d_{33} < d_{32} + d_{23} = 8 + 6 = 14$	no update
$\boldsymbol{\infty = d_{34} > d_{32} + d_{24} = 8 - 10 = -2}$	$\boldsymbol{p_{34} := p_{24} = 2}$
$1 = d_{41} < d_{42} + d_{21} = 1 + \infty = \infty$	no update
$\boldsymbol{9 = d_{43} > d_{42} + d_{23} = 1 + 6 = 7}$	$\boldsymbol{p_{43} := p_{23} = 2}$
$\boldsymbol{0 = d_{44} > d_{42} + d_{24} = 1 - 10 = -9}$	**negative cost cycle found!**

Pseudocode of Floyd-Warshall's algorithm

```
BEGIN
  FOR i:=1 TO n DO
    FOR j:=1 TO n DO  p_ij := i; END-FOR
  END-FOR
  FOR h:=1 TO n DO   /* triangular operation w.r.t. h */
    FOR i:=1 TO n WITH i ≠ h DO
      FOR j:=1 TO n WITH j ≠ h DO
        IF (d_ih + d_hj < d_ij) THEN
          d_ij  = d_ih + d_hj ;
          p_ij:= p_hj ;
        END-IF
      END-FOR
    END-FOR
    FOR i:=1 TO n DO
      IF d_ii < 0 THEN STOP; END-IF /* ∃ a circuit with negative cost*/
    END-FOR
  END-FOR
END
```

Triangular operation:

Update d_{ij} if it is convenient in terms of cost to reach *j* from *i* via *h*

Proposition: Floyd-Warshall's algorithm is exact.

Proof's idea: Assume the nodes of G are numbered from 1 to n. Just verify that, if the node index order is followed, after the h-th cycle the value d_{ij} for any i and j corresponds to the cost of a shortest path from i to j with only intermediate nodes in $\{1, \ldots, h\}$.

Complexity

Since in the worst case the triangular operation is executed for all nodes h and for each pair of nodes i and j,

the overall complexity is $O(n^3)$.

Exercise

Find the shortest paths between every pair of nodes of the following graph: