2.3. Optimal paths

Optimal (shortest, 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
- ...

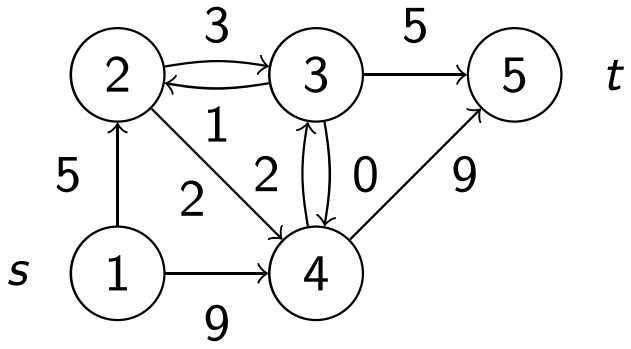
2.3. Optimal paths

Notice the difference between this problem and the minimum cost spanning tree problem.
Here the graph is directed and we want a path on the graph, then we had an undirected graph and wanted to build a spanning tree over it.

Shortest path 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.

- Each value c_{ij} represents the cost (or length, or travel time, ...) of arc $(i, j) \in A$.
- Node s is the origin, or source, t is the destination, or sink.



2.3. Optimal paths

Dijkstra's algorithm

- A path $\langle (i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k) \rangle$ is simple if no node is visited more than once.
- We assume that $c_{ij} \geq 0$, for any $(i, j) \in A$.

Property 6

If $c_{ij} \geq 0$ for all $(i, j) \in A$, there is at least one shortest path which is simple.

Edsger Dijkstra (1930-2002)

Method

- Input: Graph $G = (N, A)$ with non-negative arc costs, $s \in N$.
- Output: Shortest paths from s to all other nodes of G.

Idea Consider the nodes in increasing order of length (cost) of the shortest path from s to any one of the other nodes.

2.3. Optimal paths

Method

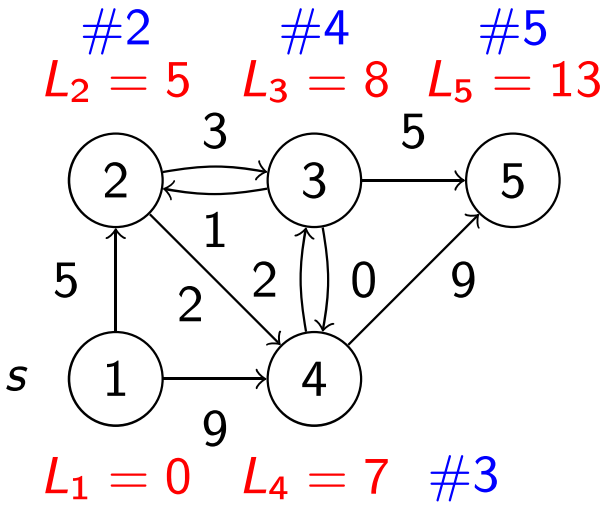
- Input: Graph $G = (N, A)$ with non-negative arc costs, $s \in N$.
- Output: Shortest paths from s to all other nodes of G.

Idea

Consider the nodes in increasing order of length (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!

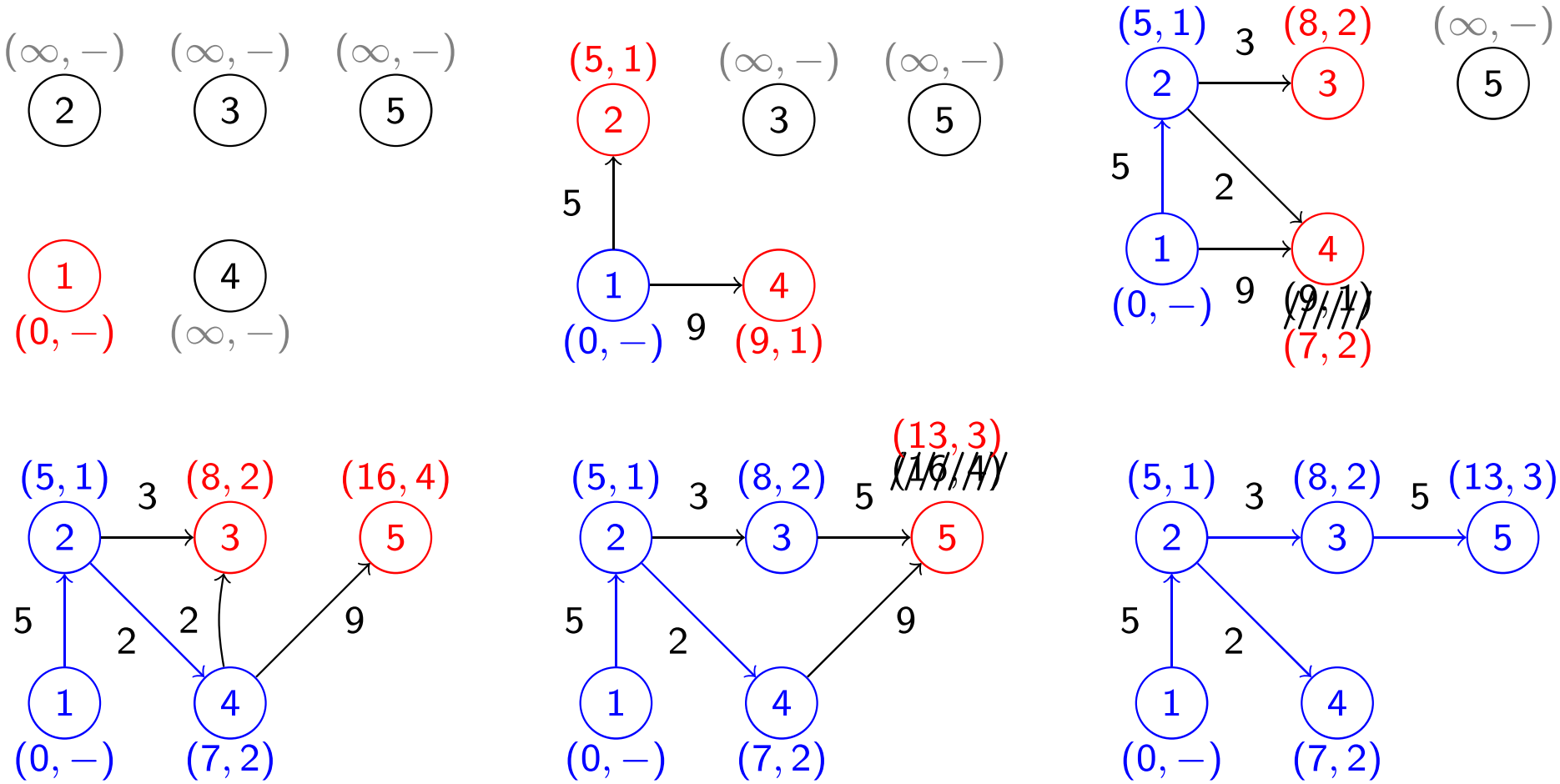
Example



2.3. Optimal paths

- Two labels are 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 a shortest path from s to j

Example

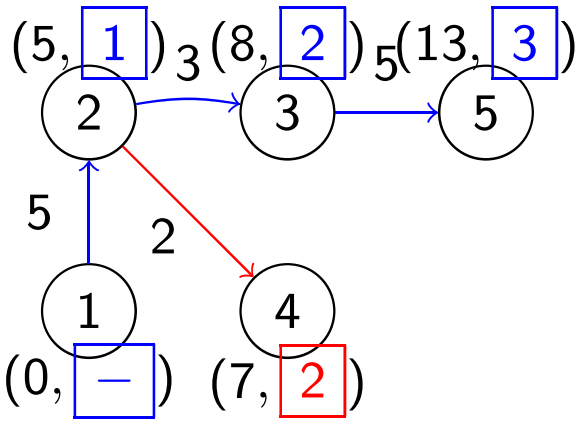


2.3. Optimal paths

A set of shortest paths from s to any node $j \neq s$ can be retrieved backwards:

$$pred_j, pred_{pred_j}, \ldots, s.$$

Example



- Resulting path from $s = 1$ to $t = 5$: $\langle (1,2),(2,3),(3,5) \rangle$
- Resulting path from $s = 1$ to node $j = 4$: $\langle (1,2),(2,4) \rangle$

2.3. Optimal paths

Data structure

- $S \subseteq N$: subset of nodes whose labels are permanent
- $X \subseteq N$: subset of nodes with temporary labels
- $L_j = \begin{cases} \text{cost of a shortest path from } s \text{ to } j, & j \in S \\ \min\{L_i + c_{ij} : (i,j) \in \delta^+(S)\}, & j \notin S \end{cases}$ (upper bound on the length of a shortest path)
  Given a directed graph G and the current subset of nodes $S \subset N$, select $v \notin S$ such that
  $$L_v = \min\{L_j : j \notin S\}.$$
- $pred_j = \begin{cases} \text{“predecessor” of } j \text{ in the shortest path from } s \text{ to } j, & j \in S \\ u \text{ such that } L_u + c_{uj} = \min\{L_i + c_{ij} : i \in S\}, & j \notin S \end{cases}$

2.3. Optimal paths

Dijkstra's algorithm

- Input: Graph $G = (N, E)$ with non-negative arc costs, $s \in N$.
- Output: Shortest paths from s to all other nodes of G.

Algorithm 4: Dijkstra's algorithm for the shortest path problem

```
1  S ← ∅
2  X ← {s}
3  for u ∈ N do  L_u ← +∞
4  L_s ← 0
5  while |S| ≠ n do
6      u ← argmin{L_i : i ∈ X}
7      X ← X\{u}
8      S ← S ∪ {u}
9      for (u, v) ∈ δ⁺(u) such that L_v > L_u + c_uv do
10         L_v ← L_u + c_uv
11         pred_v ← u
12         X ← X ∪ {v}
```

Complexity order

Each label is examined once, therefore the `while` loop has n iterations.

- If all m arcs are scanned in each iteration, the overall complexity would be $O(nm)$, hence $O(n^3)$ for dense graphs.
- However, we only need to consider the arcs of $\delta^+(j)$ for each node $j \in S$, hence the overall complexity is $O(n^2)$.

2.3. Optimal paths

Proposition 2

Dijkstra's algorithm is exact.

Proof.

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

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

By induction on the number k of steps :

- Base case: It is easy to see that the statement holds for $k = 1$, since

$$S = \{s\}, \quad L_s = 0, \quad L_j = +\infty, \quad \forall j \neq s.$$

- Inductive step: We must prove that, if the statement holds at the k-th step, it must also hold for the $(k+1)$-th step.

□

2.3. Optimal paths

Proof.

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

$$L_u \leq L_j, \quad \forall j \notin S.$$

Let us verify that $c(\phi) \leq c(\pi)$ for every path π from s to u.
There exist $i \in S$ and $j \notin S$ such that

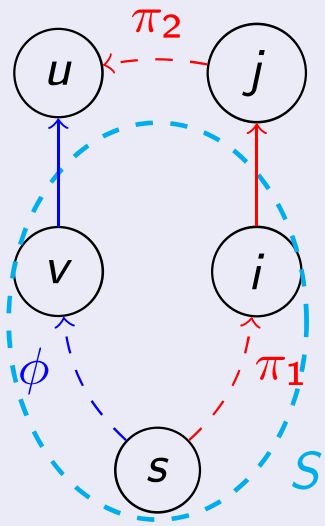
$$\pi = \pi_1 \cup \{(i,j)\} \cup \pi_2$$

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

$$c(\pi) = c(\pi_1) + c_{ij} + c(\pi_2) \geq L_i + c_{ij}$$

because $c_{ij} \geq 0$, thus, $c(\pi_2) \geq 0$, and by the induction assumption, $c(\pi_1) \geq L_i$. Finally, by the choice of u,

$$L_i + c_{ij} \geq L_v = c(\phi).$$

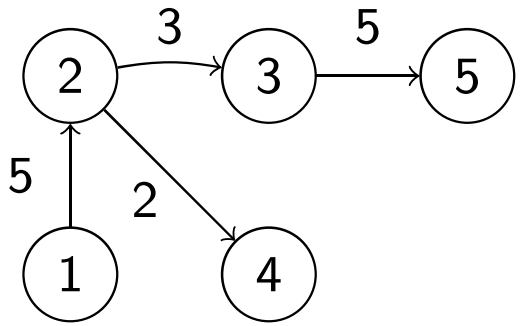


2.3. Optimal paths

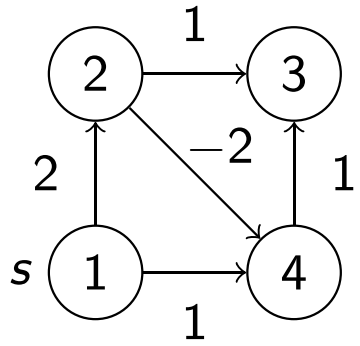
Notes

- A set of shortest paths from s to all the nodes j can be retrieved via the vector of predecessors.
- The union of a set of shortest paths from node s to all the other nodes of G is an arborescence rooted at s.

  Such arborescences (or shortest path trees), have nothing to do with minimum cost spanning trees!



- Dijkstra's algorithm does not work when there are arcs with negative cost.
  For instance, the algorithm yields the path $\langle(1,4),(4,3)\rangle$ of cost 2, but path $\langle(1,2),(2,4),(4,3)\rangle$ has cost 1.
  Due to $c_{24} < 0$ the last step in the exactness proof fails!
  The cost from 1 to 3 is not updated after the first step.
  Due to the "greedy" choice in $\delta^+(\{1\})$, it is taken as c_{14} which is "locally" optimal ($c_{14} < c_{12}$), even though the path $\langle(1,2),(2,4)\rangle$ is cheaper because $c_{24} < 0$.



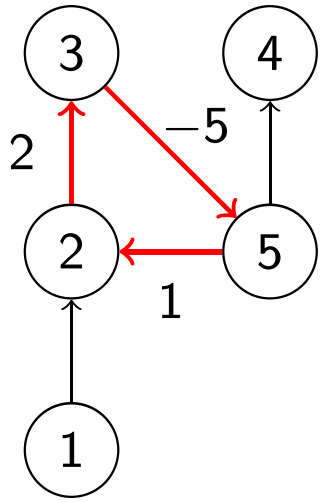
2.3. Optimal paths

Shortest path problem with negative costs

Notes

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

Example



Circuit's cost: -2

- Each time the circuit appears, the cost decreases. There is no finite shortest path from s to t.
- Floyd-Warshall's algorithm detects the presence of circuits with negative cost, thus it identifies ill-defined problems.
- It provides a set of shortest paths between all pairs of nodes, even when there are arcs with negative cost. It is based on iteratively applying a "triangular operation".

2.3. Optimal paths

Floyd-Warshall's algorithm

- Input: Directed graph $G = (N, A)$ with an $n \times n$ cost matrix, $C = [c_{ij}]$.
- Output: For each pair of nodes $i, j \in N$, the cost d_{ij} of the shortest path from i to j.

In Dijkstra the first node (s) was fixed, so a vector was enaugh for storing information about path cost, here we need a matrix because the first node is not fixed

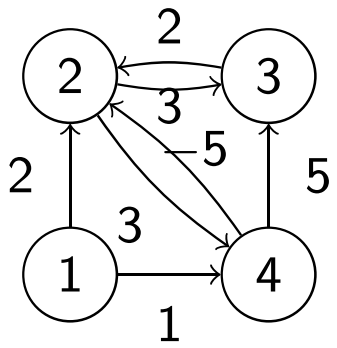
- Data structure: Two $n \times n$ matrices, D, P, whose elements correspond, at the end of the algorithm, to:
  - d_{ij} = cost of a shortest path from i to j; (Distance)
  - p_{ij} = predecessor of j in that shortest path from i to j.

- **Initialization**

$$d_{ij} = \begin{cases} 0 & i = j \\ c_{ij} & i \neq j \wedge (i,j) \in A \\ +\infty & \text{otherwise} \end{cases} \quad , \quad p_{ij} = i, i \in N$$

if there's an arc that connects the two (direct path)

(still don't know a path from i to j so we set distance to infinite)

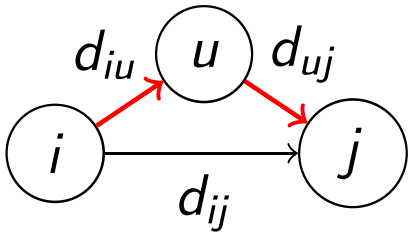


$$D = \begin{bmatrix} 0 & 2 & \infty & 1 \\ \infty & 0 & 3 & 3 \\ \infty & 2 & 0 & \infty \\ \infty & -5 & 5 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

Notice that for each column, the predecessor is equal to the index of the row

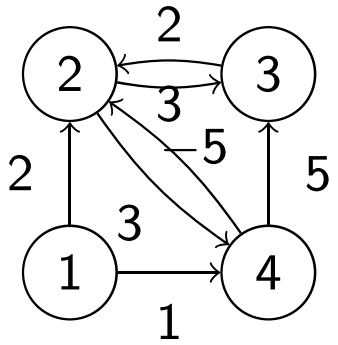
2.3. Optimal paths

- **Triangular operation** --> is the operation exploited by the Floyd - Warshall algorithm
  For each pair of nodes i, j with $i \neq u$ and $j \neq u$ (including case $i = j$), check whether when going from i to j it is more convenient to go via u:



$$d_{iu} + d_{uj} < d_{ij}$$ Check If the node u allows me to reach my destination node (j) in a less costly way

Iteration $u = 1$: Since no such pair exists, the matrices D, P remain unchanged. However, nodes $i = 1$ and $j = 1$ are skipped.

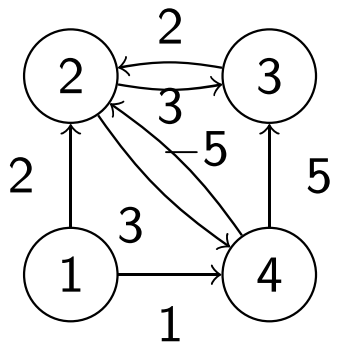


$$D = \begin{bmatrix} 0 & 2 & \infty & 1 \\ \infty & 0 & 3 & 3 \\ \infty & 2 & 0 & \infty \\ \infty & -5 & 5 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

- $0 = d_{22} < d_{21} + d_{12} = \infty + 2$ (no change)
- $3 = d_{23} < d_{21} + d_{13} = \infty + \infty$ (no change)
- $3 = d_{24} < d_{21} + d_{14} = \infty + 1$ (no change)
- $2 = d_{32} < d_{31} + d_{12} = \infty + 2$ (no change)
- $0 = d_{33} < d_{31} + d_{13} = \infty + \infty$ (no change)
- $\infty = d_{34} < d_{31} + d_{14} = \infty + 1$ (no change)
- $-5 = d_{42} < d_{41} + d_{12} = \infty + 2$ (no change)
- $5 = d_{43} < d_{41} + d_{13} = \infty + \infty$ (no change)
- $0 = d_{44} < d_{41} + d_{14} = \infty + 1$ (no change)

2.3. Optimal paths

Iteration $u = 2$:



$$D = \begin{bmatrix} 0 & 2 & \not\infty\ 5 & 1 \\ \infty & 0 & 3 & 3 \\ \infty & 2 & 0 & \not\infty\ 5 \\ \infty & -5 & \not 5\ -2 & \not\emptyset\ -2 \end{bmatrix} \quad P \begin{bmatrix} 1 & 1 & \not 1\ 2 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & \not 3\ 2 \\ 4 & 4 & \not 4\ 2 & 4 \end{bmatrix} =$$

- $0 = d_{11} < d_{12} + d_{21} = 2 + \infty$ (no change)
- $\infty = d_{13} > d_{12} + d_{23} = 2 + 3$ $(p_{13} \leftarrow p_{23})$ --> using 2 as intermediate for going from 1 to 3. It's convenient so we update distance and predecessor matrixes
- $1 = d_{14} < d_{12} + d_{24} = 2 + 3$ (no change)
- $\infty = d_{31} < d_{32} + d_{21} = 2 + \infty$ (no change)
- $0 = d_{33} < d_{32} + d_{23} = 2 + 3$ (no change)
- $\infty = d_{34} < d_{32} + d_{24} = 2 + 3$ $(p_{34} \leftarrow p_{24})$
- $\infty = d_{41} < d_{42} + d_{21} = -5 + \infty$ (no change)
- $5 = d_{43} > d_{42} + d_{23} = -5 + 3$ $(p_{43} \leftarrow p_{23})$
- $0 = d_{44} > d_{42} + d_{24} = -5 + 3$ (Negative cost circuit found! The algorithm halts!)

Halting -> we will not arrive to a solution so the algorithm stop

It's clear that there is a negative cycle since we've found a path from a node to itself with a cost which is less than zero: possible only if the cost of the found path is negative!

Algorithm 5: Floyd-Warshall's algorithm for the all-pairs shortest path problem

1 **for** $i \in N$ **do**
2 **for** $j \in N$ **do**
3 $p_{ij} \leftarrow i$; $d_{ij} = \begin{cases} 0 & i = j \\ c_{ij} & i \neq j \wedge (i,j) \in A \\ +\infty & \text{otherwise} \end{cases}$
4 **for** $u \in N$ /* Triangular operation */
5 **do**
6 **for** $i \in N \backslash \{u\}$ **do**
7 **for** $j \in N \backslash \{u\}$ **do**
8 **if** $d_{iu} + d_{uj} < d_{ij}$ /* Update d_{ij} if it is cheaper to reach j from i via u */
9 **then**
10 $p_{ij} \leftarrow p_{uj}$; $d_{ij} \leftarrow d_{iu} + d_{uj}$
11 **for** $i \in N$ **do**
12 **if** $d_{ii} < 0$ /* There is a circuit with negative cost */
13 **then** Stop

2.3. Optimal paths

Proposition 3

Floyd-Warshall's algorithm is exact.

Proof.

Idea: Assume that the nodes of G are numbered from 1 to n. Verify that, if the node index order is followed, after the u-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 $\{1, \ldots, u\}$. □

Complexity

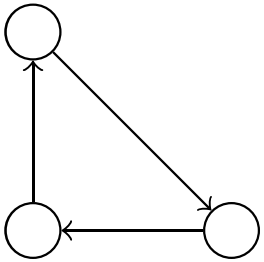
Since in the worst case the triangular operation is executed for all nodes u and for each pair of nodes i and j, the overall complexity is $O(n^3)$.

2.3. Optimal paths

Optimal paths in directed acyclic graphs

- A directed graph $G = (N, A)$ is acyclic if it contains no circuits. A directed acyclic graph is referred to as DAG.

More efficient algorithms apply to that kind of graphs, which have many applications



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

it's sufficient multiplying costs for -1

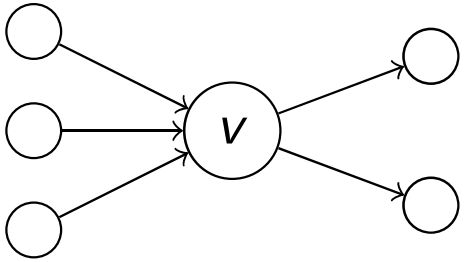
Property (Topological order) The nodes of any directed acyclic graph G can be ordered topologically, that is, indexed so that for each arc $(i, j) \in A$ we have $i < j$.

The topological order can be exploited in a very efficient dynamic programming algorithm to find shortest (or longest) paths in DAGs.

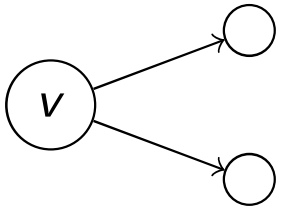
2.3. Optimal paths

Topological ordering method

$G = (N, A)$ is represented via the lists of predecessors $\delta^-(v)$ and successors $\delta^+(v)$ for each node v.



1. Assign the smallest positive integer not yet assigned to a node $v \in N$ with $\delta^-(v) = \emptyset$.

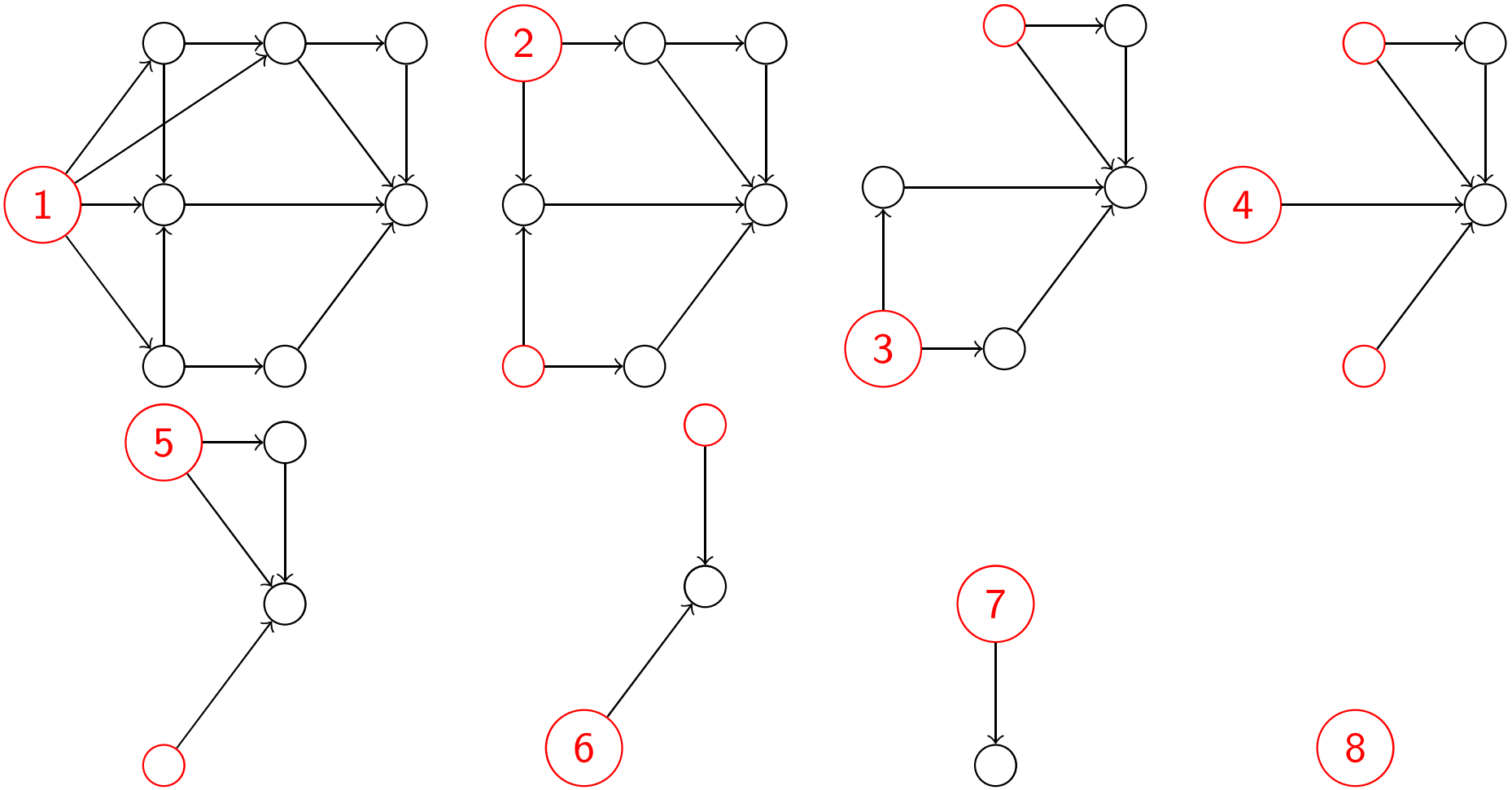


Such a node v exists because G does not contain circuits. (DAG)

2. Delete the node v with all its incident arcs.
3. Go back to 1. until there are nodes in the current subgraph.

2.3. Optimal paths

Example



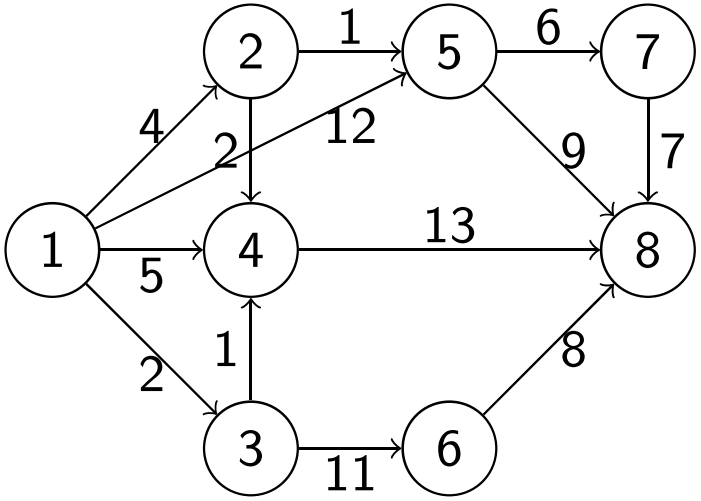
Complexity

$O(m)$ where $m = |A|$, because each node/arc is considered at most once.

2.3. Optimal paths

Problem

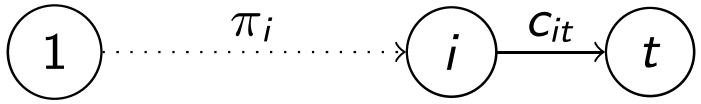
- Given G and a topological order of the nodes, how to find a shortest path from node 1 to node 8?



2.3. Optimal paths

Dynamic programming for shortest paths in DAGs

Any shortest path from 1 to t, π_t, with at least 2 arcs, can be subdivided into two parts: π_i and (i, t), where π_i is a shortest subpath from s to i.



- For each node $i = 1, \ldots, t$, let L_i be the cost of a shortest path from 1 to i. Then,

$$L_t = \min_{(i,t)\in\delta^-(t)} \{L_i + c_{it}\},$$

costo di tutti gli archi che vanno dai nodi predecessori di t a t stesso

where the minimum is taken over all possible predecessors i of t.

- If G is directed, acyclic and topologically ordered, the only possible predecessors of t in a shortest path π_t from 1 to t are those with index $i < t$. Thus,

$$L_t = \min_{i<t} \{L_i + c_{it}\}.$$

- In a graph with circuits, any node different from t can be a predecessor of t in π_t!

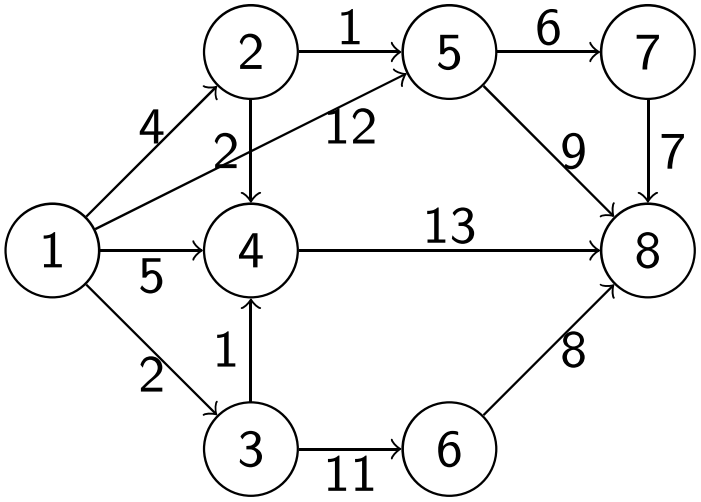
2.3. Optimal paths

For DAGs whose nodes are topologically ordered $L_{t-1}, \ldots, L_1$ satisfy the same type of recursive relations:

$$L_{t-1} = \min_{i<t-1}\{L_i + c_{i,t-1}\};\ \cdots;\ L_2 = \min_{i=1}\{L_i + c_{i2}\} = L_1 + c_{12};\ L_1 = 0$$

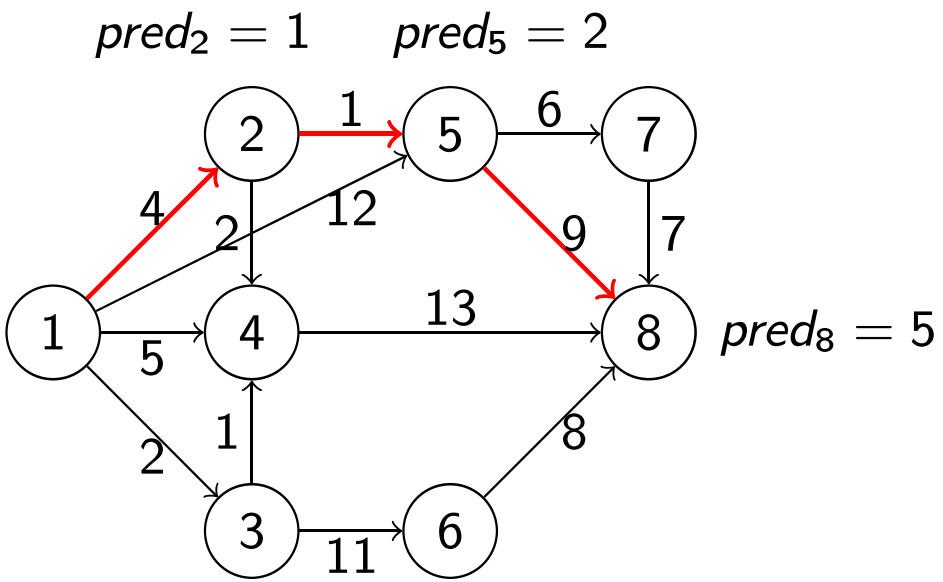
which can be solved in the reverse order:

$$L_1 = 0;\ L_2 = L_1 + c_{12};\ \cdots;\ L_t = \min_{i<t-1}\{L_i + c_t\}$$



2.3. Optimal paths

$$L_1 = 0 \qquad pred_1 = 1$$
$$L_2 = L_1 + c_{12} = 4 \qquad pred_2 = 1$$
$$L_3 = L_3 + c_{13} = 2 \qquad pred_3 = 1$$
$$L_4 = \min_{i=1,2,3}\{L_i + c_{i4}\} = \min\{0+5, 4+2, 2+1\} = 3 \qquad pred_4 = 3$$
$$L_5 = \min_{i=1,2}\{L_i + c_{i5}\} = \min\{0+12, 4+1\} = 5 \qquad pred_5 = 2$$
$$L_6 = L_3 + c_{36} = 2 + 11 = 13 \qquad pred_6 = 3$$
$$L_7 = L_5 + c_{57} = 5 + 6 = 11 \qquad pred_7 = 5$$
$$L_8 = \min_{i=4,5,6,7}\{L_i + c_{i8}\} = \min\{3+13, 5+9, 13+8, 11+7\} = 14 \qquad pred_8 = 5$$



2.3. Optimal paths

Algorithm 6: Dynamic programming to find shortest paths in DAGs

Sort the nodes of G topologically
$L_1 \leftarrow 0$
for $j = 2, \ldots, n$ `/* in topological order */`
do
  $L_j \leftarrow \min\{L_i + c_{ij} : (i,j) \in \delta^-(j) \wedge i < j\}$
  $pred_j \leftarrow v$ such that $(v,j) = \text{argmin}\{L_i + c_{ij} : (i,j) \in \delta^-(j) \wedge i < j\}$

- Topological ordering of the nodes: $O(m)$ with $m = |A|$
- Each node/arc is considered exactly once: $O(n + m)$

Overall: $O(m)$

Adaptation of DP for longest paths in DAGs

Straightforward adaptation of the DP method to find longest paths in directed acyclic graphs:

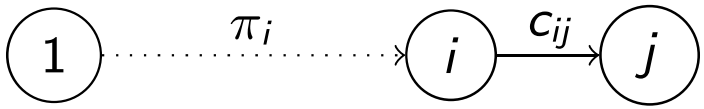
$$L_t = \max_{i<t}\{L_i + c_{it}\}, \ldots$$

2.3. Optimal paths

Proposition 4

The Dynamic Programming algorithm for finding shortest (longest) paths in DAGs is exact.

This is due to the
Optimality principle
For any shortest (longest) path from 1 to t, π_t, there exists $i < j$ such that the path can be subdivided into two parts: π_i and (i, t), where π_i is a minimum (maximum) lenght from s to i.



2.3. Optimal paths

Dynamic Programming (DP)

- Proposed in 1953 by Richard Bellman (1920-1984).
- General technique in which an optimal solution, composed of a sequence of elementary decisions, is determined by solving a set of recursive equations.
- DP is applicable to any "sequential decision problem" for which the optimality property is satisfied.
- Wide range of applications: optimal control (rocket launch), equipment maintenance and replacement, selection of inspection points along a production line, ...

2.3. Optimal paths

Example

To simplify, suppose that the cost of a new machine is constant (12 K€).

Goal

To minimize the net total cost (price + maintenance - recover) over a 5 year horizon.

Age	Main.	Reco.
0	2	–
1	4	7
2	5	6
3	9	2
4	12	1

If we buy a machine at the beginning of the 1st year and sell it at the beginning of the 2nd year, the net cost is: $12 + 2 - 7 = 7$

If we buy it at the beginning of the 1st year and sell it at the beginning of the 3rd year, the net cost is: $12 + 2 + 4 - 6 = 12$, etc, ...

Show how the problem of determining an optimal maintenance – replacement policy of minimum total cost (for the next 5 years) can be solved via DP.

Hint

Reduce it to a minimum cost path problem in an appropriate acyclic graph.
Find all the optimal maintenance-replacement policies.

2.3. Optimal paths

Project planning

- A project consists of a set of m activities with their duration: activity A_i has (estimated) duration $d_i \geq 0$, $i = 1, \ldots, m$.
  Some pairs of activities are subject to a precedence constraint: $A_i \propto A_j$ indicates that A_j can start only after the end of A_i.

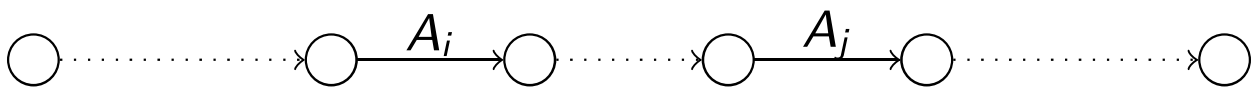
Model

A project can be represented by a directed graph $G = (N, A)$ where:

- each arc corresponds to an activity, and
- its arc length represents the duration of the corresponding activity.

To account for the precedence constraints, the arcs must be positioned so that:

$$A_i \propto A_j \Leftrightarrow \quad \text{there exists a directed path where the arc associated to } A_i \text{ precedes the arc associated to } A_j$$



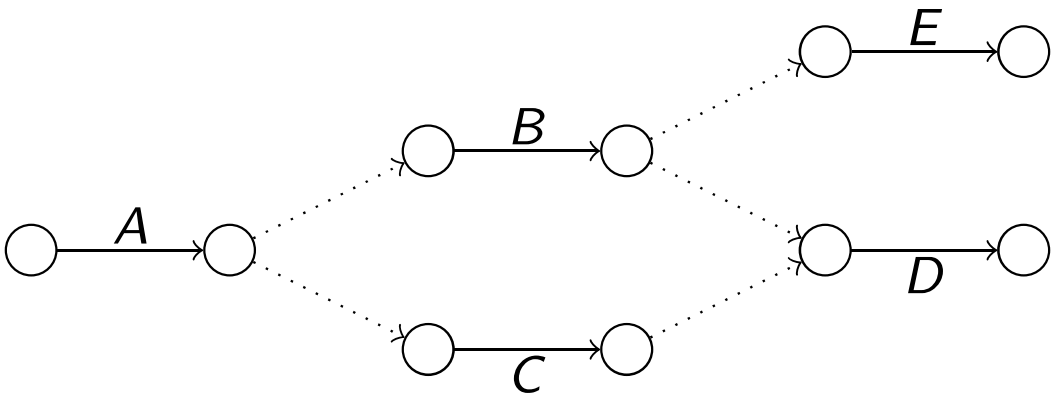
Therefore, a node v marks an event corresponding to the end of all the activities $(i, v) \in \delta^-(v)$ and, hence, the possible beginning of all those $(v, j) \in \delta^+(v)$.

2.3. Optimal paths

Example

Activities: A, B, C, D, E

Precedences: $A \propto B, A \propto C, B \propto D, C \propto D, B \propto E$.



Dummy activities with duration 0

It's not a real activity, it's just a way to represent the precedence

Property 7

The directed graph G representing any project is acyclic (it is a DAG).

Always, because if we had a cycle we would had eventually an activity that depends on itself

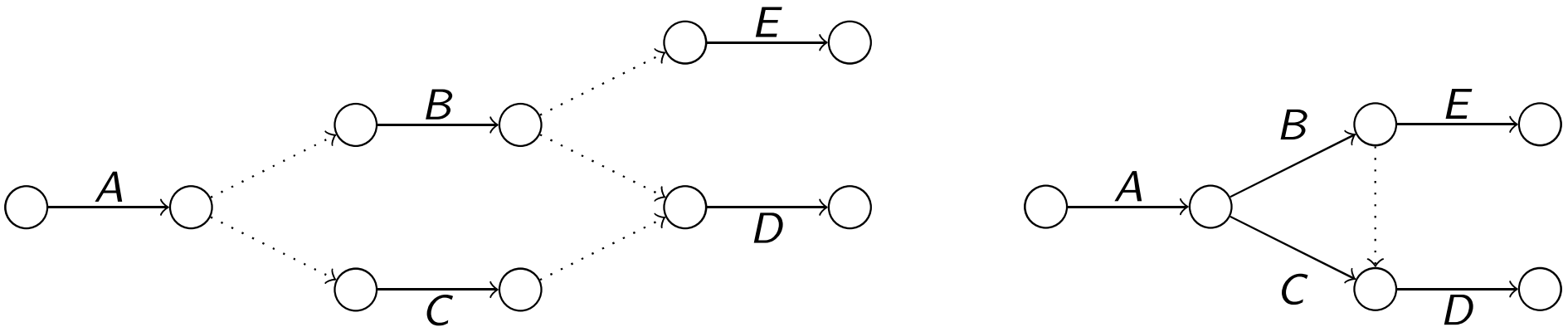
Proof.

By contradiction, if $A_{i1} \propto A_{12}, \ldots, A_{jk} \propto A_{ki}$ there would be a logical inconsistency. □

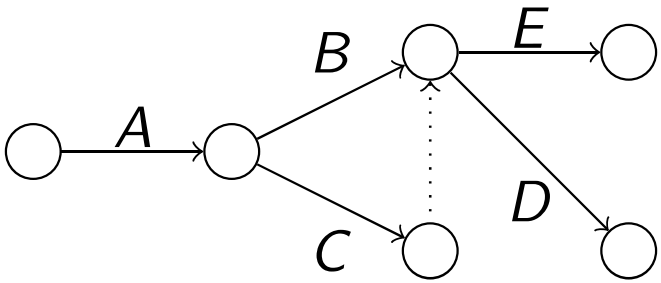
2.3. Optimal paths

Example

Graph G can be simplified by contracting some arcs:



When doing this, pay attention not to introduce unwanted precedence constraints.



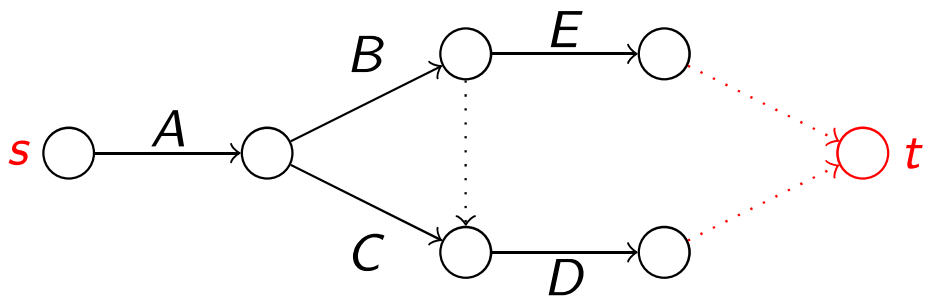
If this dummy activity is maintained, unwanted $C \propto E$ is implied, besides $A \propto B$, $A \propto C$, $B \propto D$, $C \propto D$, $B \propto E$.

2.3. Optimal paths

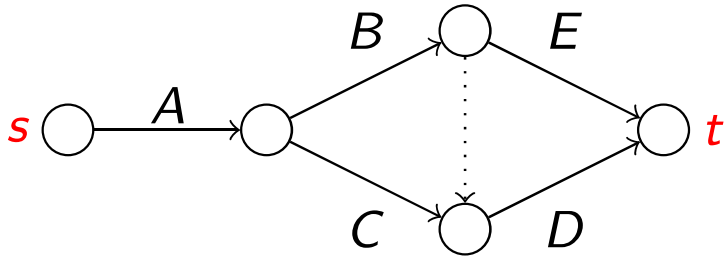
Artificial nodes or artificial arcs are introduced so that graph G:

- contains a unique initial node s corresponding to the event "beginning of the project";
- contains a unique final node t corresponding to the event "end of the project";
- does not contain multiple arcs (with same endpoints).

Example



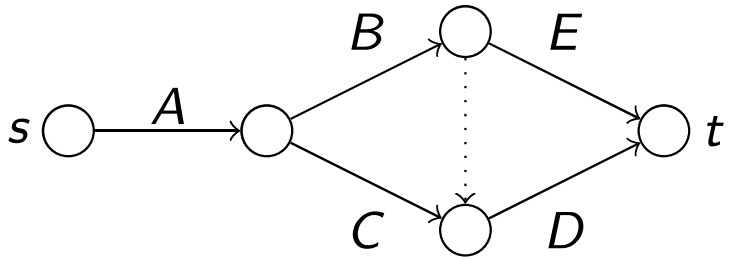
or



2.3. Optimal paths

Problem 3

Given a project (set of activities with durations and precedence constraints), schedule the activities so as to minimize the overall project duration, i.e., the time needed to complete all activities.



Property 8

The minimum overall project duration = length of a longest path from s to t.

Proof.

Since any $s - t$ path represents a sequence of activities that must be executed in the specified order, its length provides a lower bound on the minimum overall project duration. □

2.3. Optimal paths

Critical path method (CPM)

Determines:

- a schedule (a plan for executing the activities specifying the order and the alotted time) that minimizes the overall project duration;
- the slack of each activity, i.e., the amount of time by which its execution can be delayed, without affecting the overall minimum project duration.

Initialization: Construct the graph G representing the project and find a topological order of the nodes.

Step I: Consider the nodes by increasing indices and, for each $h \in N$, find:

- the earliest time $Tmin_h$ at which the event associated to node h can occur ($Tmin_n$ corresponds to the minimum project duration).

Step II: Consider the nodes by decreasing indices and, for each $h \in N$, find:

- the latest time $Tmax_h$ at which the event associated to node h can occur without delaying the project completion date beyond $Tmin_n$.

Step III: For each activity $(i, j) \in A$, find the slack: $\sigma_{ij} = Tmax_j - Tmin_i - d_{ij}$

2.3. Optimal paths

Example
Consider the following project:

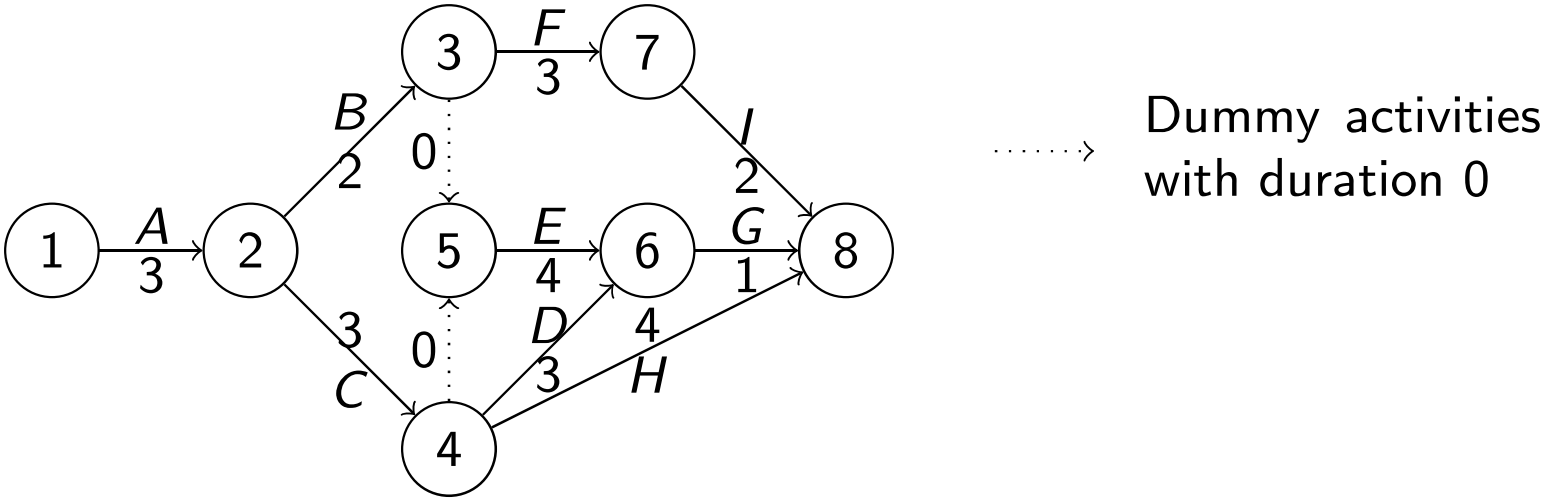
Activity	Duration	Predecessors
A	3	–
B	2	A
C	3	A
D	3	C
E	4	B, C
F	3	B
G	1	E, D
H	4	C
I	2	F

Precedence constraints:
$A \propto B, A \propto C, C \propto D, B \propto E,$
$C \propto E, B \propto F, E \propto G, D \propto G,$
$C \propto H, F \propto I$

Determine the overall minimum duration of the project and the slack for each activity.

2.3. Optimal paths

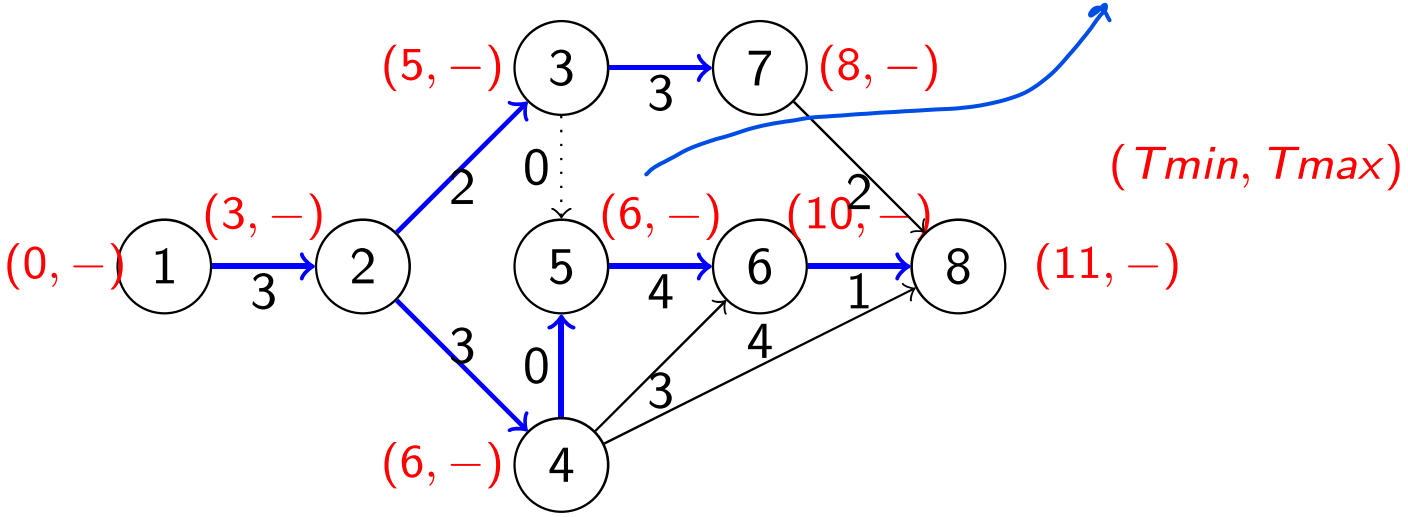
Activities: $A, B, C, D, E, F, G, H, I$
Precedences: $A \propto B, A \propto C, C \propto D, B \propto E,\ C \propto E, B \propto F, E \propto G, D \propto G,$
$C \propto H, F \propto I$



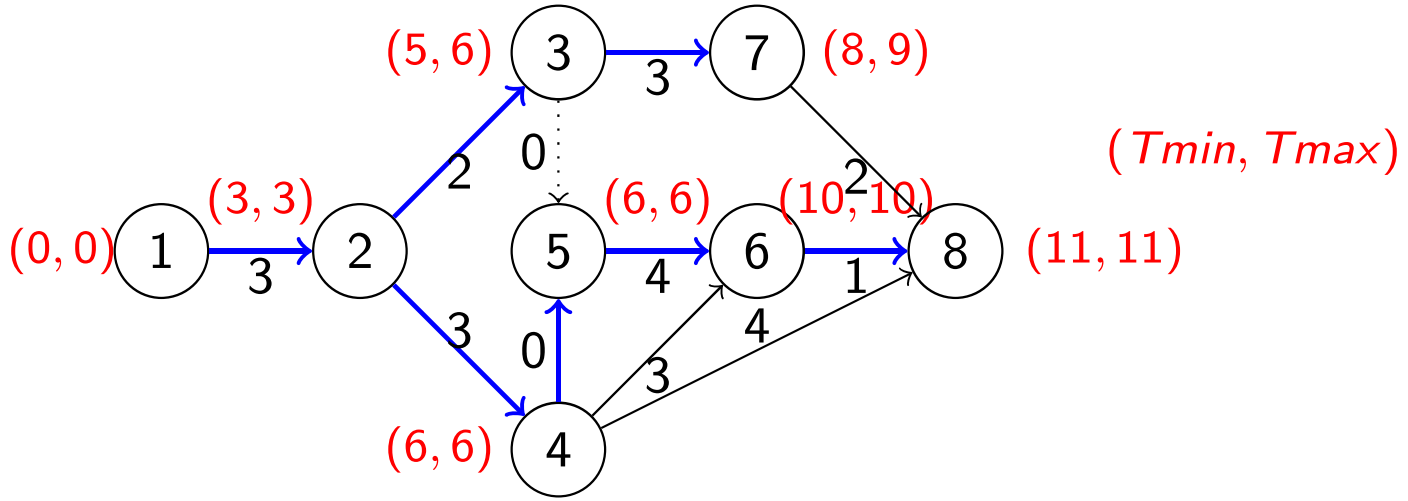
2.3. Optimal paths

Phase I

whenever you have more than one arc going into node d,
you need to take the biggest cost (time). Because it is the minimum
time at which the node can occur

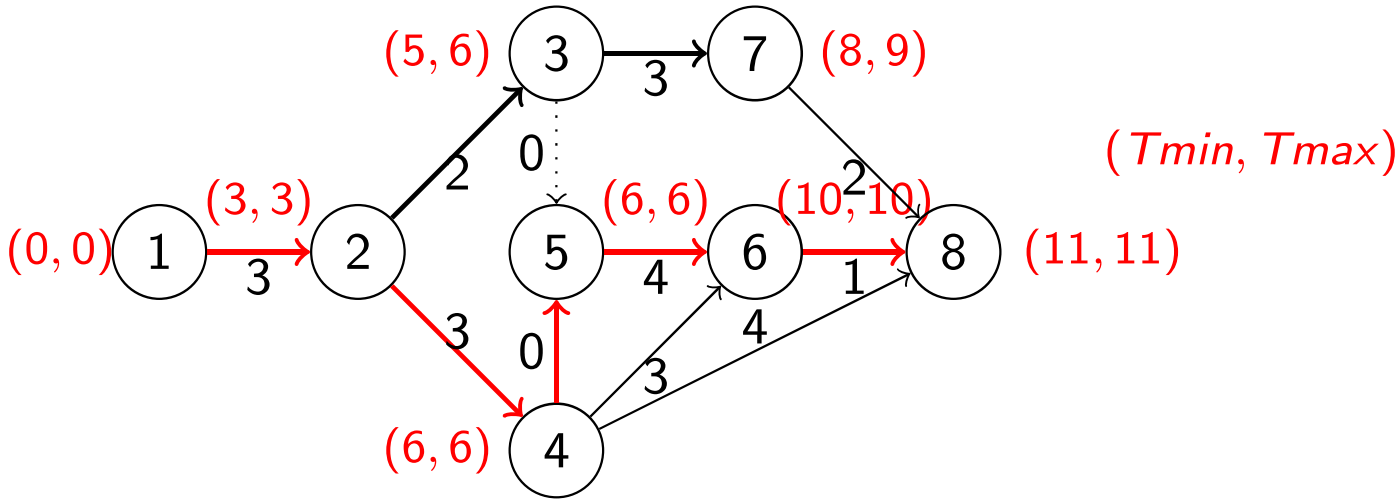


Phase II



2.3. Optimal paths

Longest path



2.3. Optimal paths

Algorithm for the critical path method (CPM)

- Input: Graph $G = (N, A)$, with $n = |N|$ and the duration d_{ij} associated to each $(i, j) \in A$.
- Output: $(Tmin_i, Tmax_i)$, for $i = 1, \ldots, n$.

Algorithm 7: Dynamic programming to find shortest paths in DAGs

1 Sort the nodes topologically
2 $Tmin_1 \leftarrow 0$
3 **for** $j = 2, \ldots, n$ **do**
4 $\quad Tmin_j \leftarrow \max\{Tmin_i + d_{ij} : (i, j) \in \delta^-(j)\}$
5 $Tmax_n \leftarrow Tmin_n$/* minimum project duration */
6 **for** $i = n - 1, \ldots, 1$ **do**
7 $\quad Tmax_i \leftarrow \min\{Tmax_j - d_{ij} : (i, j) \in \delta^+(i)\}$

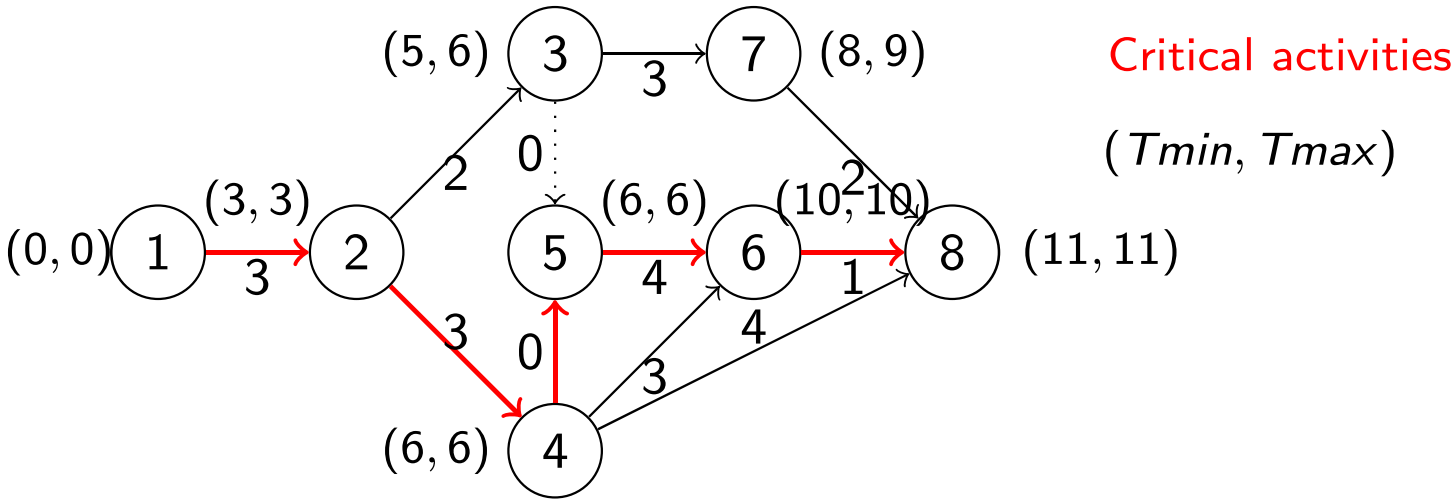
Complexity order

$(n + m) + (n + m) + (n + m)$, thus $O(n + m)$, or simply $O(m)$.

2.3. Optimal paths

Definition 4

An activity (i, j) with zero slack ($\sigma_{ij} = Tmax_j - Tmin_i - d_{ij} = 0$) is critical.



$(4, 6)$, $(3, 7)$ and $(4, 8)$ are not critical, since:

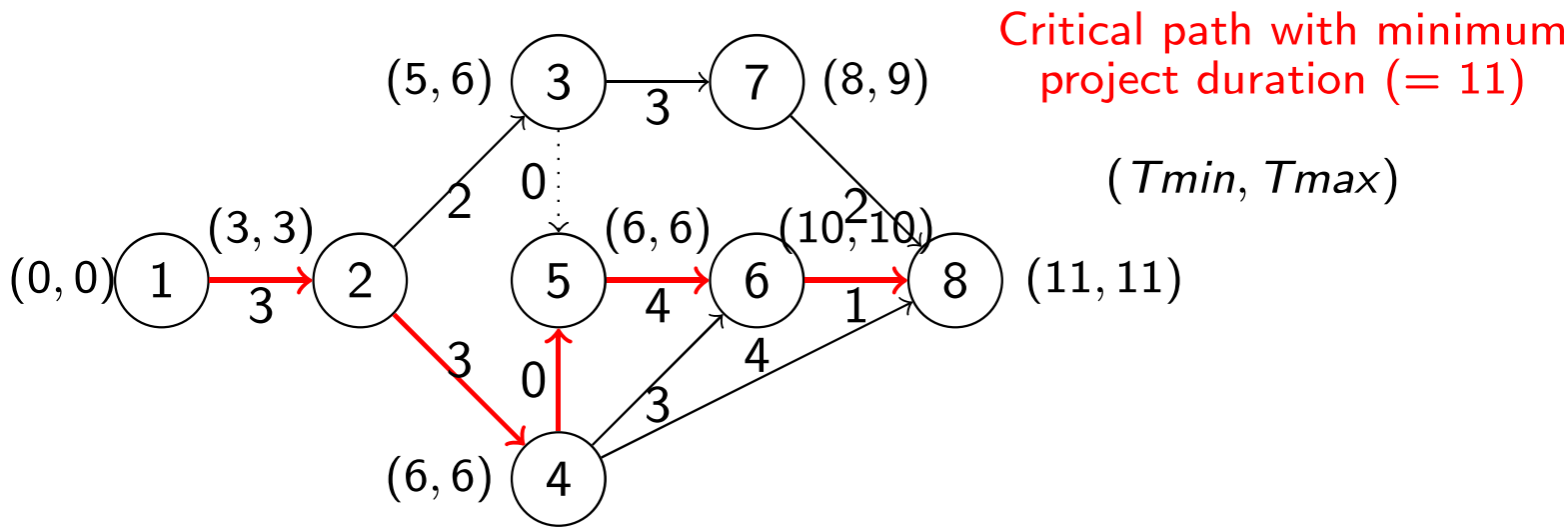
$$\sigma_{46} = 10 - 6 - 3 = 1, \sigma_{37} = 9 - 5 - 3 = 1, \sigma_{48} = 11 - 6 - 4 = 1.$$

Note: $Tmin_i = Tmax_i$ and $Tmin_j = Tmax_j$ do not suffice to have: $\sigma_{ij} = Tmax_j - Tmin_i - d_{ij} = 0$!

2.3. Optimal paths

Definition 5

A critical path is an $s - t$ path only composed of critical activities (one such path always exists).



2.3. Optimal paths

Gantt charts

Henry Gantt
(1861-1919)

Introduced in the 1910s by Henry Gantt.
Provide temporal representations of the project.

Gantt chart at earliest: Each activity (i, j) starts at time $Tmin_i$
Gantt chart at latest: Each activity (i, j) ends at time $Tmax_j$

(i,j)	d_{ij}	$Tmin_i$	$Tmax_j$
A	3	0	3
B	2	3	6
C	3	3	6
D	3	6	10
E	4	6	10
F	3	5	9
G	1	10	11
H	4	6	11
I	2	8	11