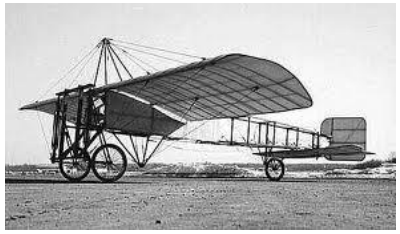
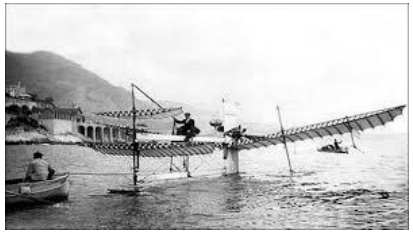
CentraleSupélec

Elements of Complex System Engineering

Antoine B. Rauzy
Department of Mechanical and Production Engineering (MTP)
Norwegian Science and Technology University (NTNU)
and
Chaire Blériot-Fabre
Centrale-Supélec, SAFRAN


CentraleSupélec

LECTURE 4.
GRAPHS

Notions:

- Graphs
- Main problems and algorithms

CentraleSupélec

LECTURE 4. PART 1. INTRODUCTION

CentraleSupélec

Objective of this lecture

Graphs are the most basic and widely used concepts in complex system modeling. From a very practical point of view, there is not a single modeling tool that does not embed some type of graphs as internal data structures. At a more conceptual level, modeling formalisms are all borrowing a lot to graph theory.

The objective of this lecture is thus:

- To present/recall the vocabulary and the main concepts of graph theory.
- To present/recall some of the most important graph algorithms.
- To show, via a use case, that graphs can be used rather directly as a modeling tool.

Case Study: GPS Navigators

GPS navigators arrived not that long time ago in everybody's cars (the first version of the TomTom system has been released in 2003). GPS navigators are now considered as mandatory and no one wants to use anymore old road and city maps.

The Global Positioning System (GPS) makes it possible to localize the position of a vehicle with a precision ranging from 3 to 50 meters (and even more precisely for some applications).
But how GPS navigators are doing to propose us shortest itineraries from one location to another one?

What to Do?


Arrival
Departure
Le Plessis-Robinson
Sceaux
Parc de la Vallée aux Loups
Châtenay-Malabry
Parc de Sceaux
Centrale Formation
Allée d'Honneur
Av Lulli
Av Galilée
Av du Général de Gaulle
D60
D75
D63
D67
D74
D920
D77
D2
D75E
D128
D986
A86
27
26
Antony

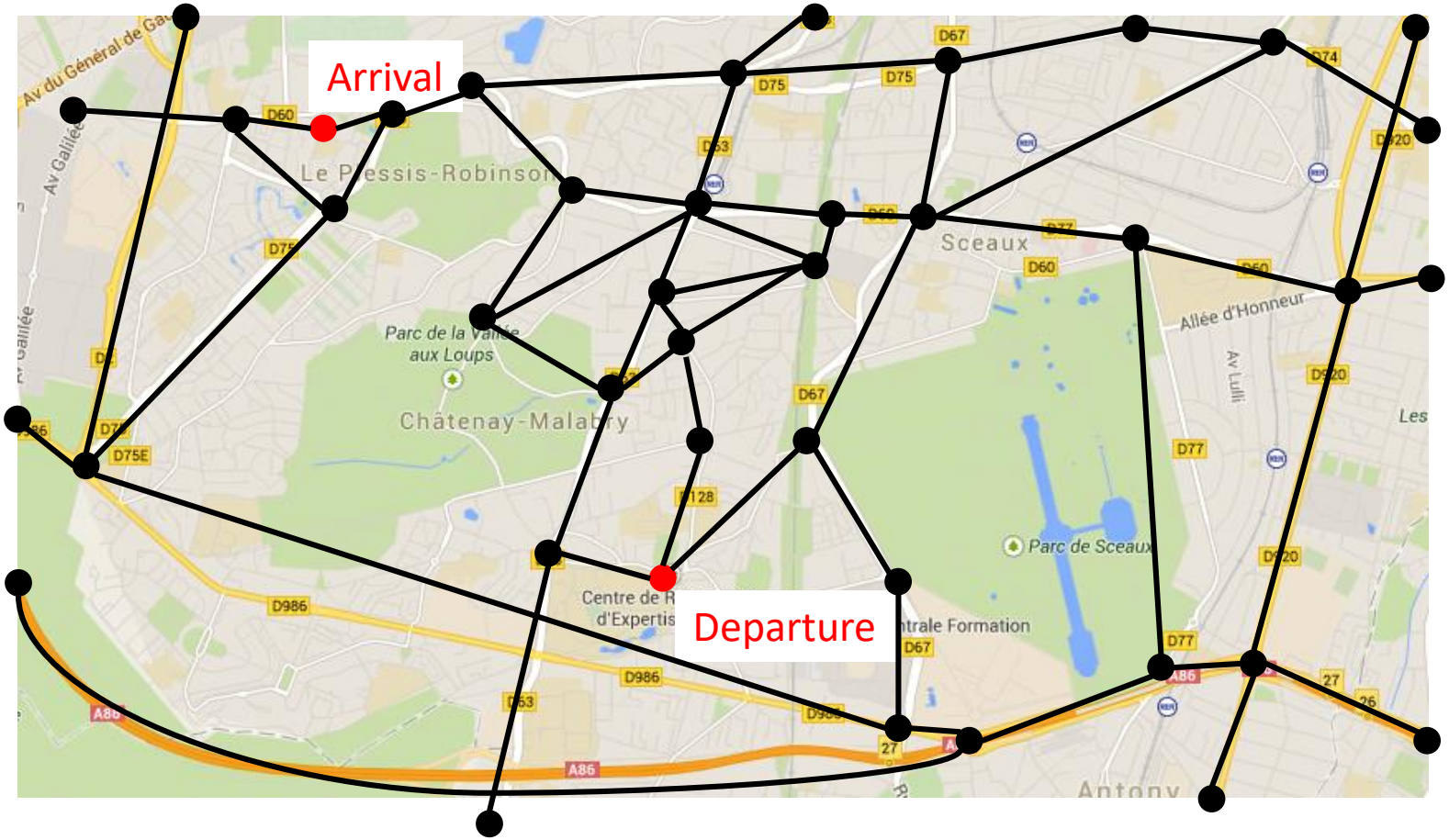
What to Do?

1. Abstract the territory as a map.

2. Find the road which is the closest from the departure point using some geometric algorithm.
   Do the same for the arrival point.

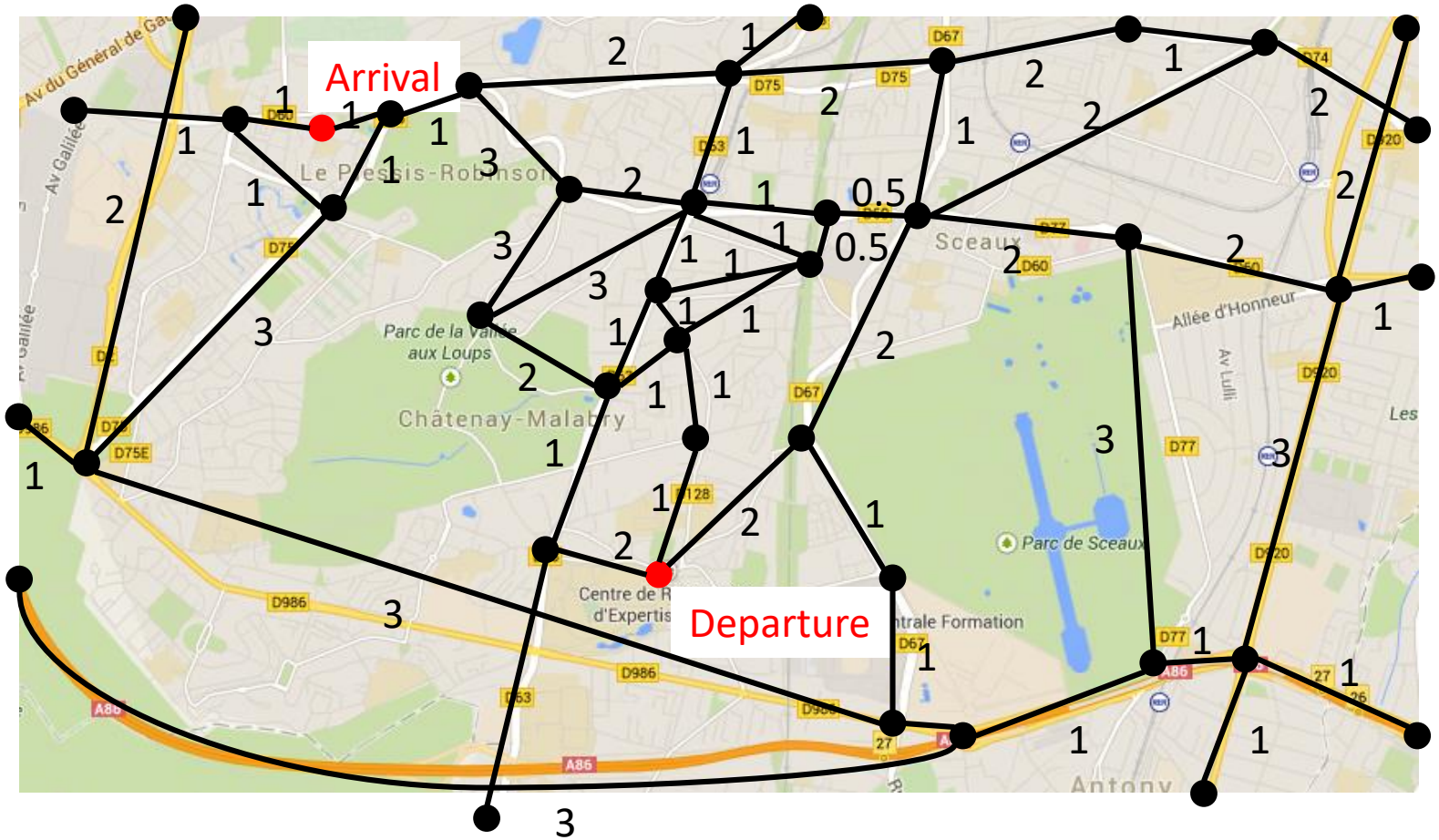
What to Do?

3. Abstract the map as a graph



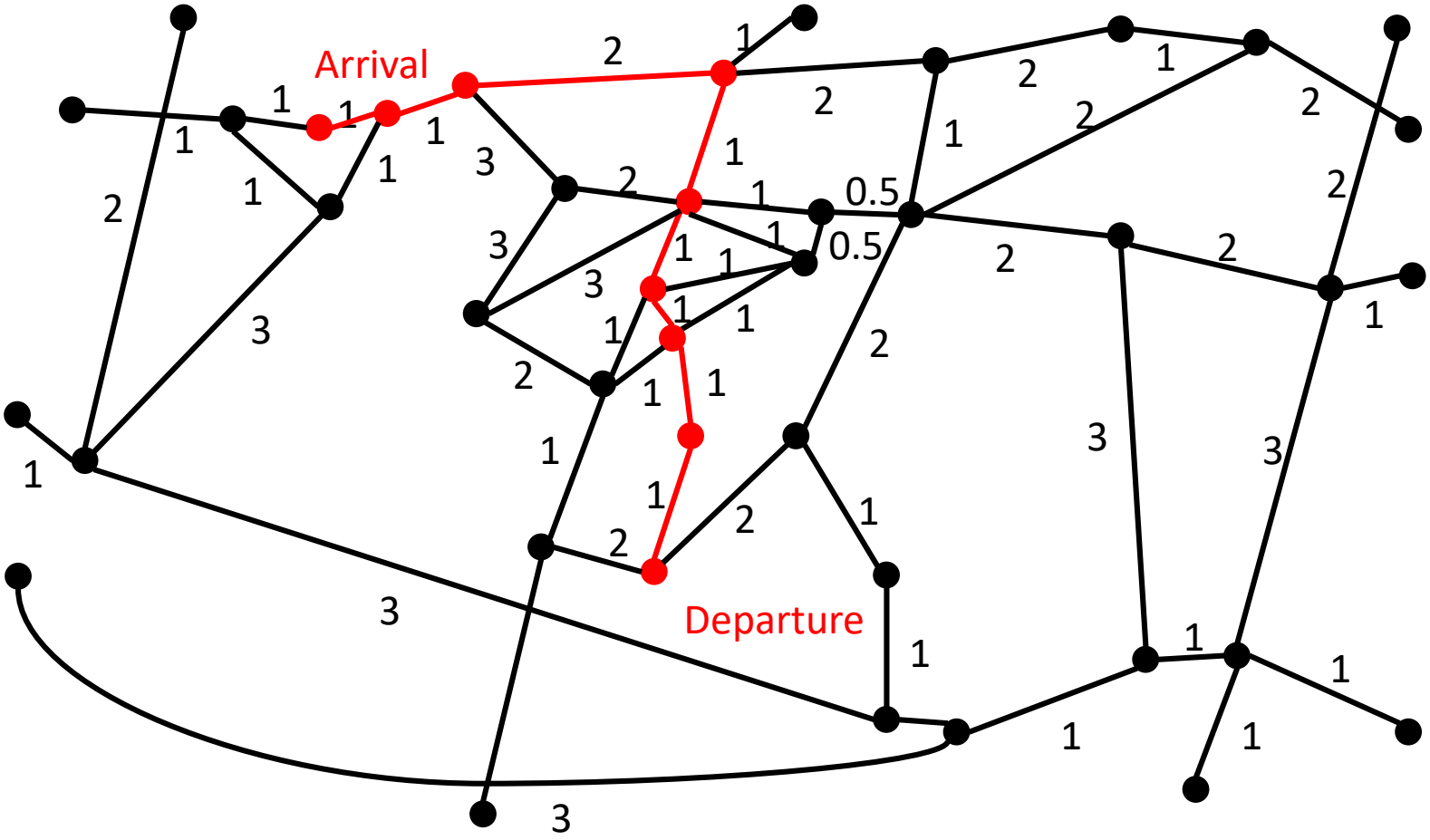
What to Do?

4. Estimate the length of each edge (for instance by considering the mean time to go from one node of the edge to the other one).



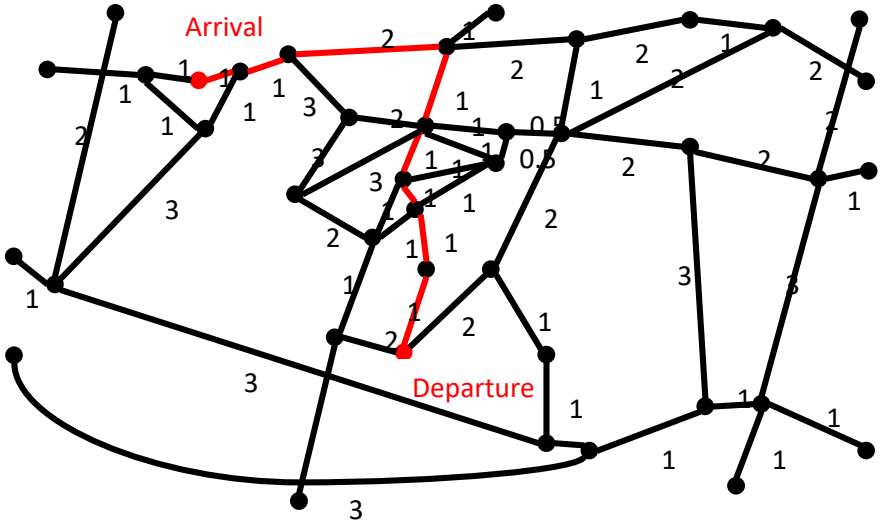
What to Do?

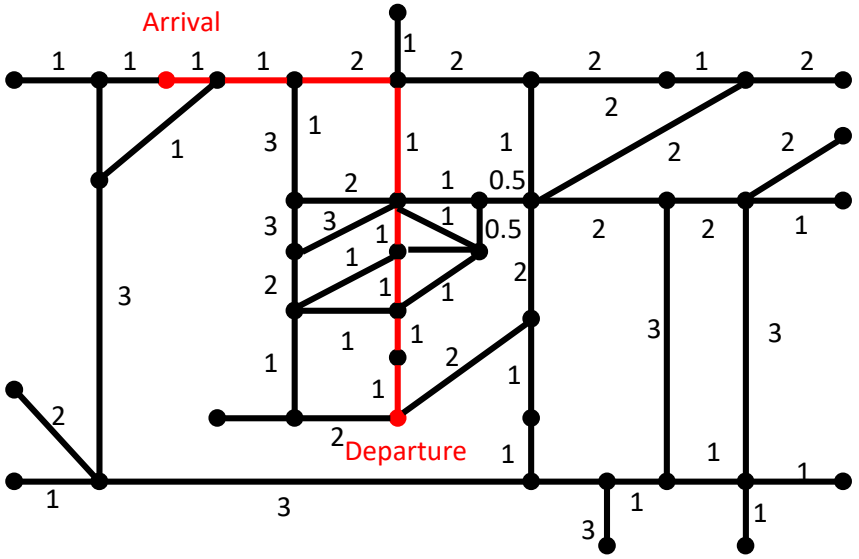
5. Find the shortest path from the departure to the arrival nodes



Discussion

A graph is an **abstraction** of the map (itself an abstraction of the territory). A graph is mathematical object. Its graphical representation does not matter.





The two graphs above are **isomorphic**.

CentraleSupélec

LECTURE 4. PART 2. DEFINITIONS

CentraleSupélec

Graphs

A **graph** is a pair (V, E) where V is set of **nodes**, also called **vertices** (a **vertex**, two **vertices**) and E is a set of ordered pairs of elements of V, called **edges**.

The above definition works for **oriented graphs**, also called **directed graphs**. In some case, we are interested in **non-oriented graphs**, i.e. graphs in which the elements of E non-ordered pairs of elements of V: (u, v) is equivalent to (v, u).

Let G = (V, E) be a graph and let (s, t) be an oriented edge of E. Vertices s and t are called respectively the **source node** and the **target node** of the edge. We say also that s is a **predecessor** of t and t is a **successor** of s.

Edges of the graph may **labeled** (typically by names, or lengths). The graph is then a triple (V, E, L) where L is a set of **labels** and edges are triples (s, l, t) where s and t are nodes of V and l is a label of L.

CentraleSupélec

Graphs (bis)

A graph may contain several edges between two nodes s and t. It may also contain edge from the node s to itself. Such an edge is called a **loops**.

Two nodes s and t are **adjacent** if there is an edge from s to t or an edge from t to s. Two edges are **adjacent** if they share at least one node.

The **in degree** of a node t is the number of edges (s, t) in the graph, i.e. the number of edge whose target is t. Similarly, the **out degree** of a node s is the number of edges of the graph whose source is s. The **degree** of a node is the sum of its in degree and its out degree.

The **degree** of a graph is maximum degree of its node.

CentraleSupélec

Graphs (ter)

A **path** from a node s_1 to a node s_k is a sequence of edges (s_1,s_2), (s_2, s_3)... (s_{k-1}, s_k) where the s_i's are non necessarily distinct nodes.

A node t is **reachable** from a node s if it exists a path from s to t.

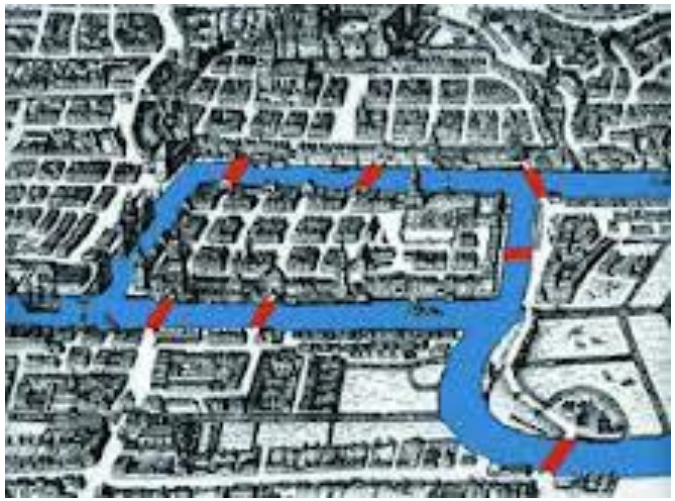
A **circuit** is a path from a node s to itself.

The **length** of a path is the number of edges of this path. When the graph is labeled, it is also possible to describe the length as the sum of the labels of the edges of the path (or any other convenient aggregation operation).

The **distance** from a node s to a node t is the length of the shortest path from s to t (by convention, this distance is infinity if t is not reachable from s).

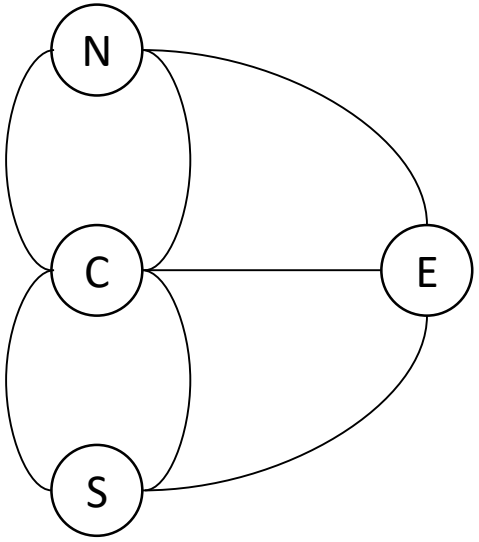
A **strongly connected component** is a subset U of nodes such a that for every nodes s and t in U, there exists a path from s to t.

Digression: Königsberg's bridges



The birth of the notion of graph is in general attribute to the great Swiss mathematician **Leonhard Euler** (1707-1783). Euler has been asked to find a promenade (actually a parad) through the different districts of the city of Königsberg that would go through all of the bridges of the city, but that would not go twice by any bridge.

In modern terms, solving the problem consists in looking for a circuit in the graph to the right that passes once and only once by each edge. Such a circuit is said **Eulerian**).
As Euler showed such a circuit cannot exist: all of the nodes of the graph have a odd degree. But going through a node "consumes" two edges.



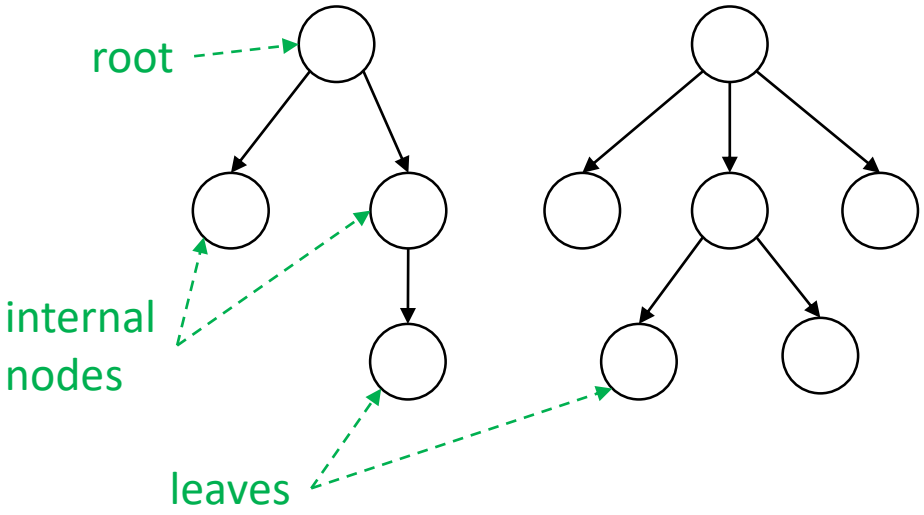
CentraleSupélec

Trees

In modeling, a wide use is made of two special types of graphs: trees and directed acyclic graphs.

A **tree** is a directed graph such that the in-degree of each node is at most 1.

A **root** of a tree, is a node of that tree whose in-degree is 0. A **leaf** of a tree is a node whose out degree is 0. Other nodes are said **internal**. Most of the time, only **uniquely rooted** trees are considered and trees with multiple roots are called **forests** because they can be seen as sets of independent trees.

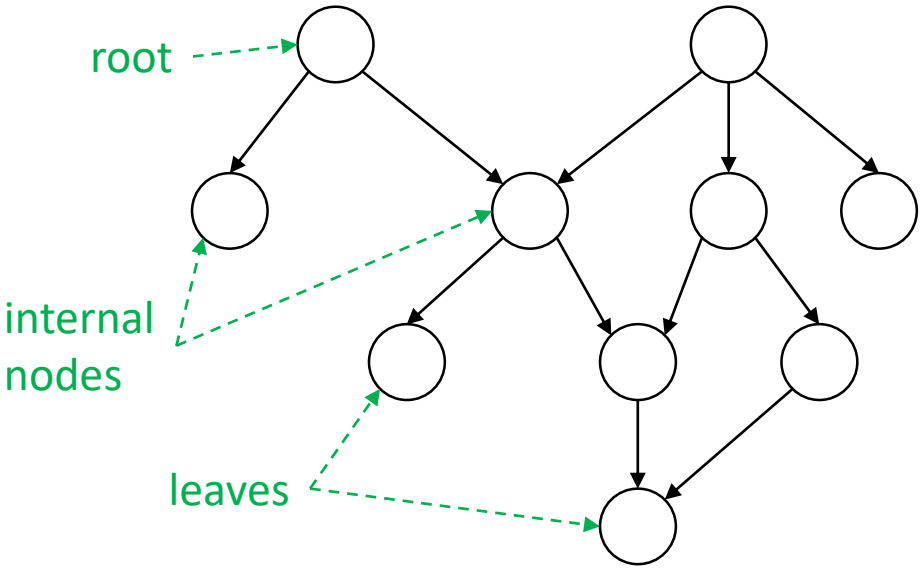


Strangely enough trees of mathematicians and computer scientists are growing top-down…

Directed Acyclic Graphs

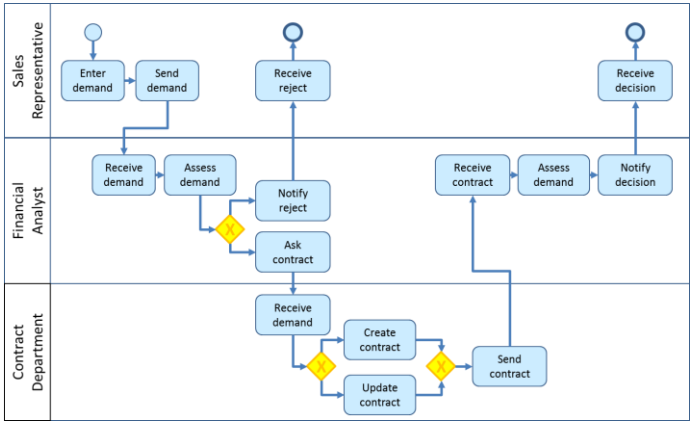
A **Directed Acyclic Graph (DAG)** is a graph that contains no circuit.

A directed graph (V, E) is acyclic if there it is possible to define an **order relation** on its nodes compatible with the orientation of its edges: for any two nodes u and v of V, $u < v$ if $u \neq v$ and there is a path from u to v.



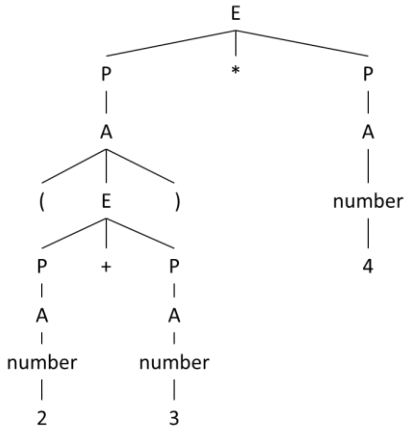
Examples

BPMN models:



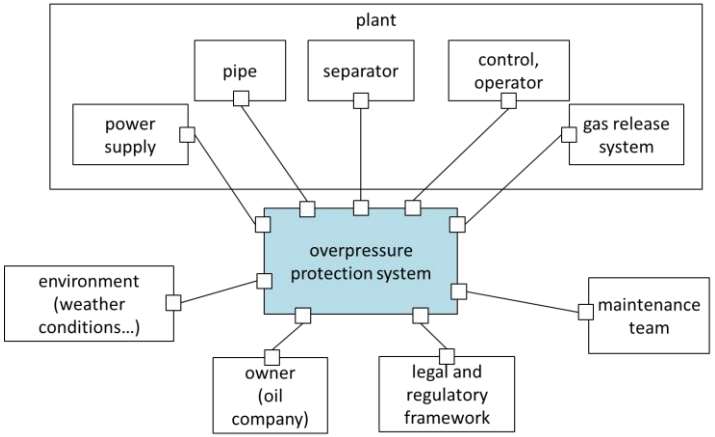
Directed Acyclic Graph

Production trees:



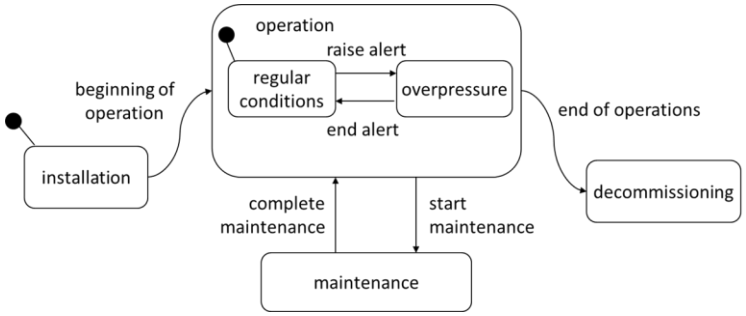
Tree

Environment Diagrams:



Tree (or graph)

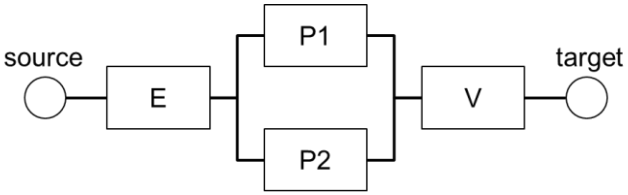
State Diagrams:



Graph

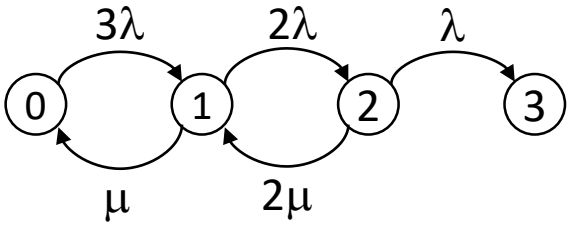
Examples

Block Diagrams:



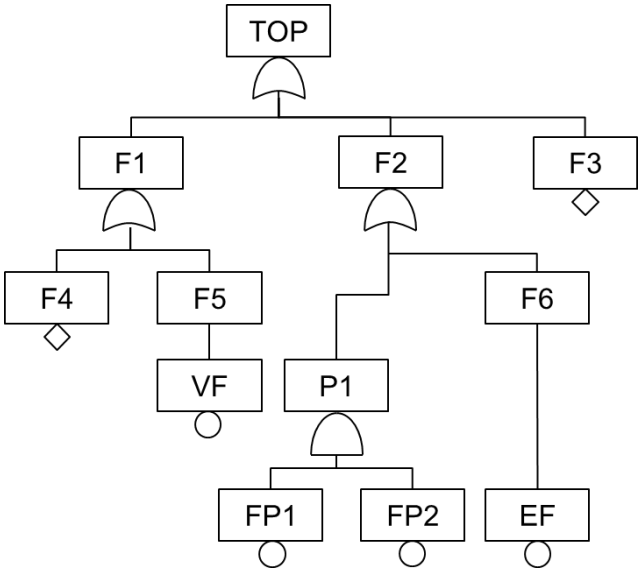
Directed Acyclic Graph

Markov Chains:



Graph

Fault Trees:



Directed Acyclic Graph

CentraleSupélec

LECTURE 4. PART 3. EXPLORATION ALGORITHMS

CentraleSupélec

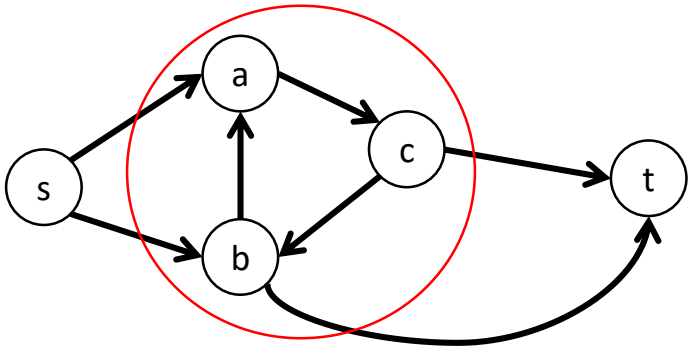
Paths

To implement the navigation system, we need at least an algorithm that given a graph G and two distinguished nodes s and t of this graph, finds a **path** from s to t, if any.

The recursive principle of such an algorithm is quite simple:

- There is an empty path [] from any node u to itself.
- There exists a path from the node u to the node t, $u \neq t$, if:
  - Either there is a edge from u to t, or
  - There exists another node v such that there is an edge from u to v and a path σ from v to t. In that case (u,v).σ is a path from u to t.

This principle is easy to turn into an algorithm but… take care to the circuits!



Depth-First Search

```
define Path(G = (V, E), s ∈ V, t ∈ V)
    forall v ∈ V
        v.visited = false
    return _Path(G, s, t)

define _Path(G = (V, E), u ∈ V, t ∈ V)
    if u.visited
        return None
    u.visited = true
    if u = t
        return []
    forall (u, v) ∈ E
        σ = _Path(v, t)
        if σ ≠ None
            return (u,v).σ
     return None
```

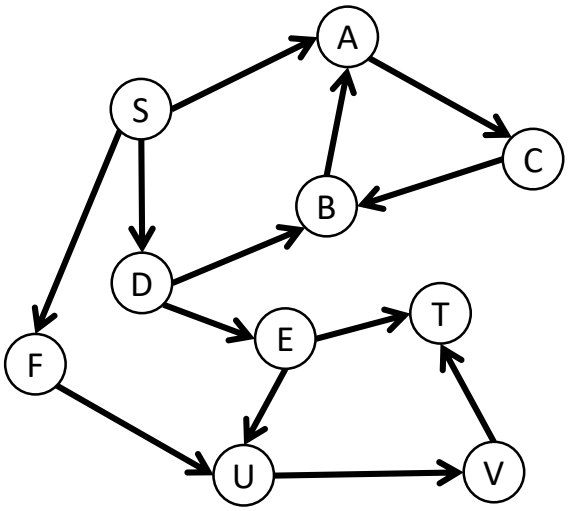
Remark 1: The function _Path is a **depth-first exploration** of the graph.

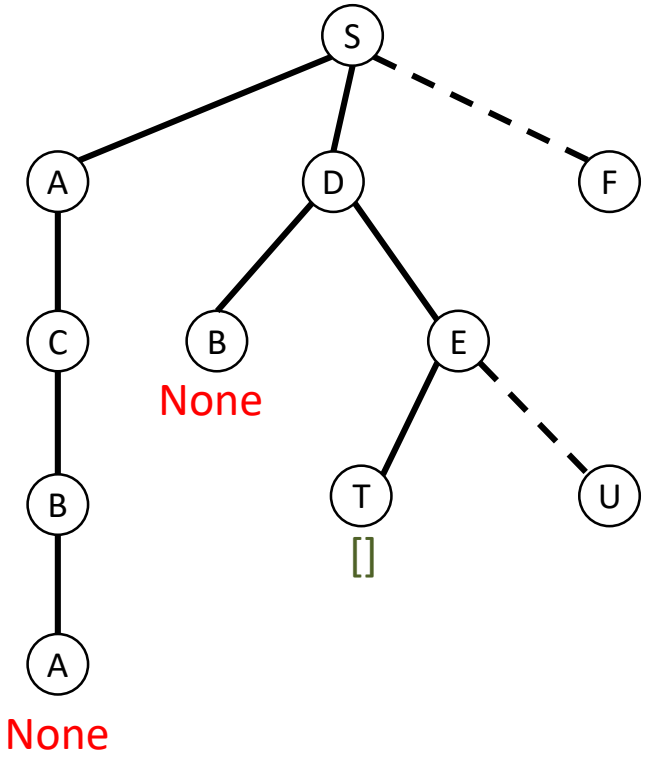
Remark 2: The flag "visited" is used to avoid problems due to circuits. Visited flags are like the pebbles of Tom Thumb…

Remark 3: Each node is treated at most once. As a consequence, each edge is also gone through at most once. The algorithm is thus of linear complexity with respect to the size of the graph.

CentraleSupélec

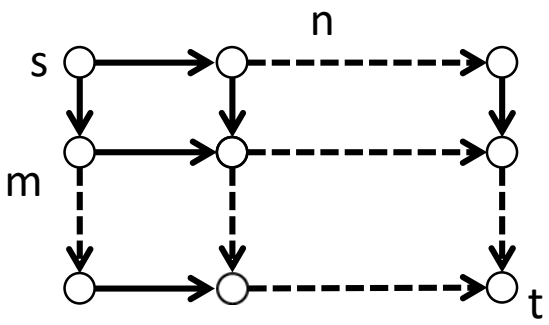
Example





Shortest Paths

We could try to transform our Path algorithm in order to seek not only for one path, but for all paths and to select the shortest one among those paths. However, the number of paths can grow exponentially with respect to the size of the graph.



The number of s-t paths on a mxn grid is $\binom{m+n}{n}$

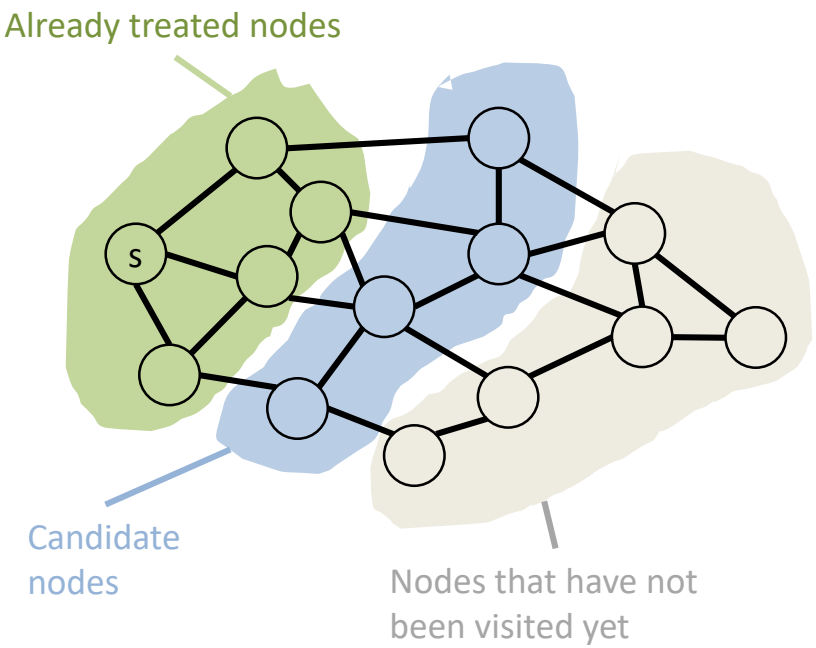
This is the reason why all shortest path algorithms rely more or less on the same principle

1. Calculate the **distance** from the source s to all other nodes in the graph.
2. Extract from the graph the shortest path from s to t.

Calculation of the Distance

An idea, due to **Dijskra**, to calculate the distance from the node s to all other nodes of the graph is to proceed as follows.

- We start by picking-up in the neighborhood of s the node u_1 which is the closest to s.
- Then, we pick-up in the neighborhood of s and u_1 the node u_2 which is the closest of s.
- And so on…



A each step of the algorithm, there three sets of nodes:

- The set of already treated nodes. These nodes are the closest to s.
- The set candidate nodes, i.e. nodes that have been reached, but not treated yet.
- The set of remaining nodes, i.e. nodes that have not been reached yet.

The next treated node will the node which is the closest to s amongst the candidate nodes.

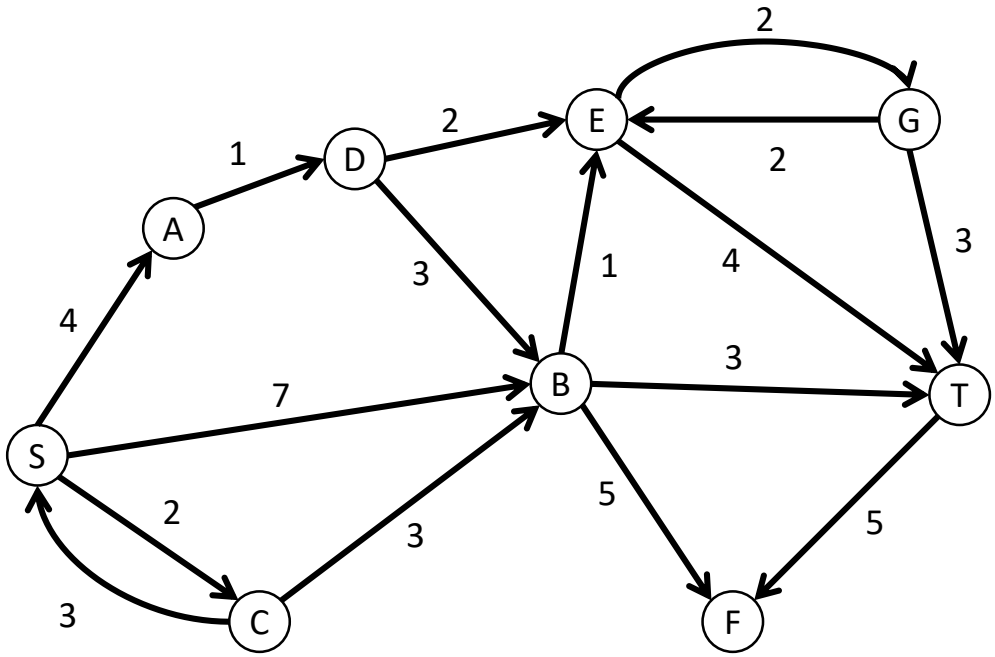
Dijskra's Algorithm

```
Distance(G = (V, E), s ∈ V)
forall v ∈ V, v ≠ s
    v.treated = false, v.distance = ∞
s.distance = 0
Candidates = {s}
while Candidates ≠ ∅
    select u in Candidates with minimum distance
    u.treated = true
    Candidates = Candidates \ {u}
    forall (u, l, v) ∈ E
        if not v.treated
            if v ∉ Candidates
                Candidates = Candidates ∪ {v}
            if v.distance > u.distance + l
                v.distance = u.distance + l
```

Remark 1: This works only if all distances are positive.

Remark 2 : In the worst case, the while loop is iterated |V| times and each edge is looked at most once.

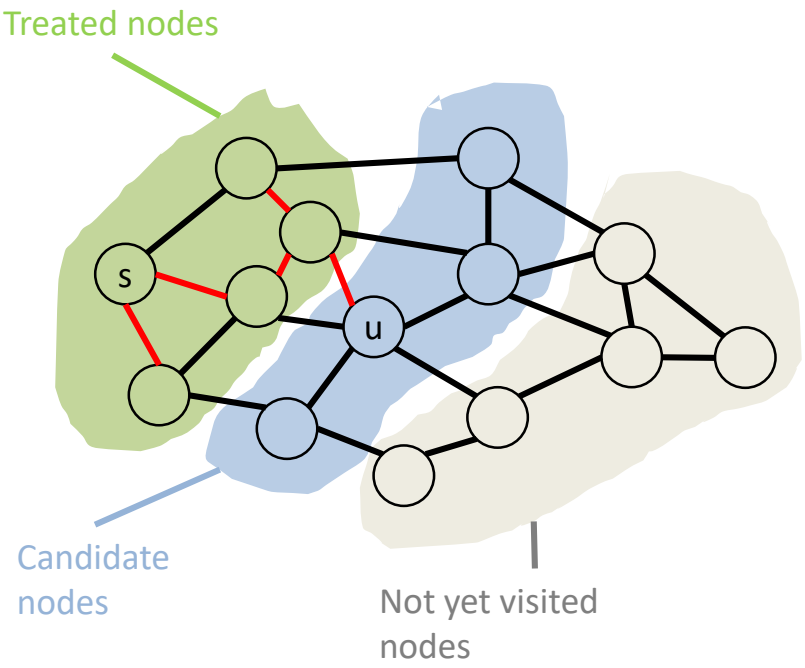
Example



Treated	Candidates
	S(0)
S(0)	C(2), A(4), B(7)
C(2)	A(4), B(5)
A(4)	B(5), D(5)
B(5)	D(5), E(6), T(8), F(10)
D(5)	E(6), T(8), F(10)
E(6)	G(8), T(8), F(10)
G(8)	T(8), F(10)
T(8)	F(10)

Remark: we can stop once the target node t is reached!

Calculation of the Distance (justification)



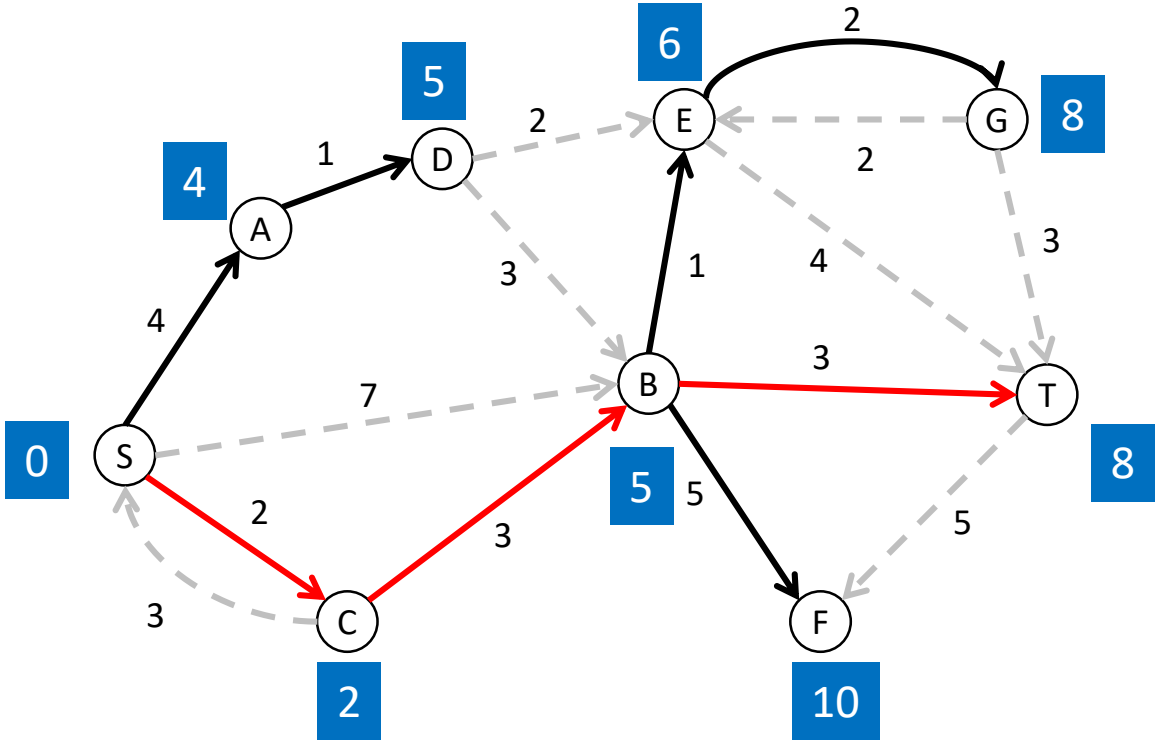
Property: The Dijskra algorithm calculates the shortest distance between s and other nodes of the graph.

Proof: consider the node u which is selected at a give step of the algorithm.
By construction, there is no other candidate node with a shortest distance from s.
But any alternative path from s to u must go trough another candidate node.

Extraction of shortest paths

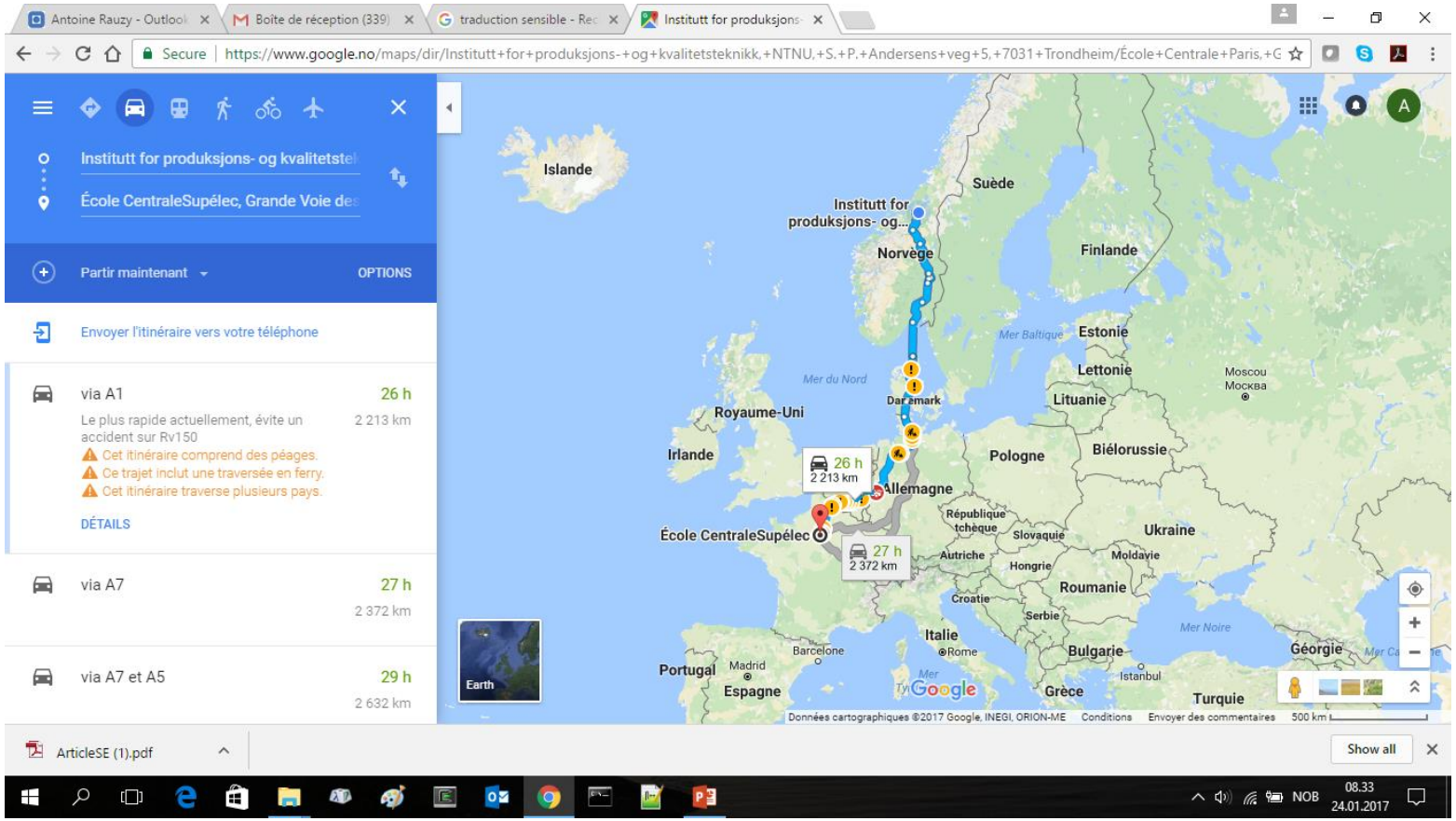
Each edge (u, l, v) of the shortest path must be such that distance(v) = distance(u) + l

As a consequence, to find the shortest path(s), it suffices to remove all edges that do not verify the property, and then simply seek for a path.



Back to GPS Navigation Systems

GPS navigation systems use **hierarchical graphs** (with zones, sub-zones…). Algorithms seeks for several shortest paths.

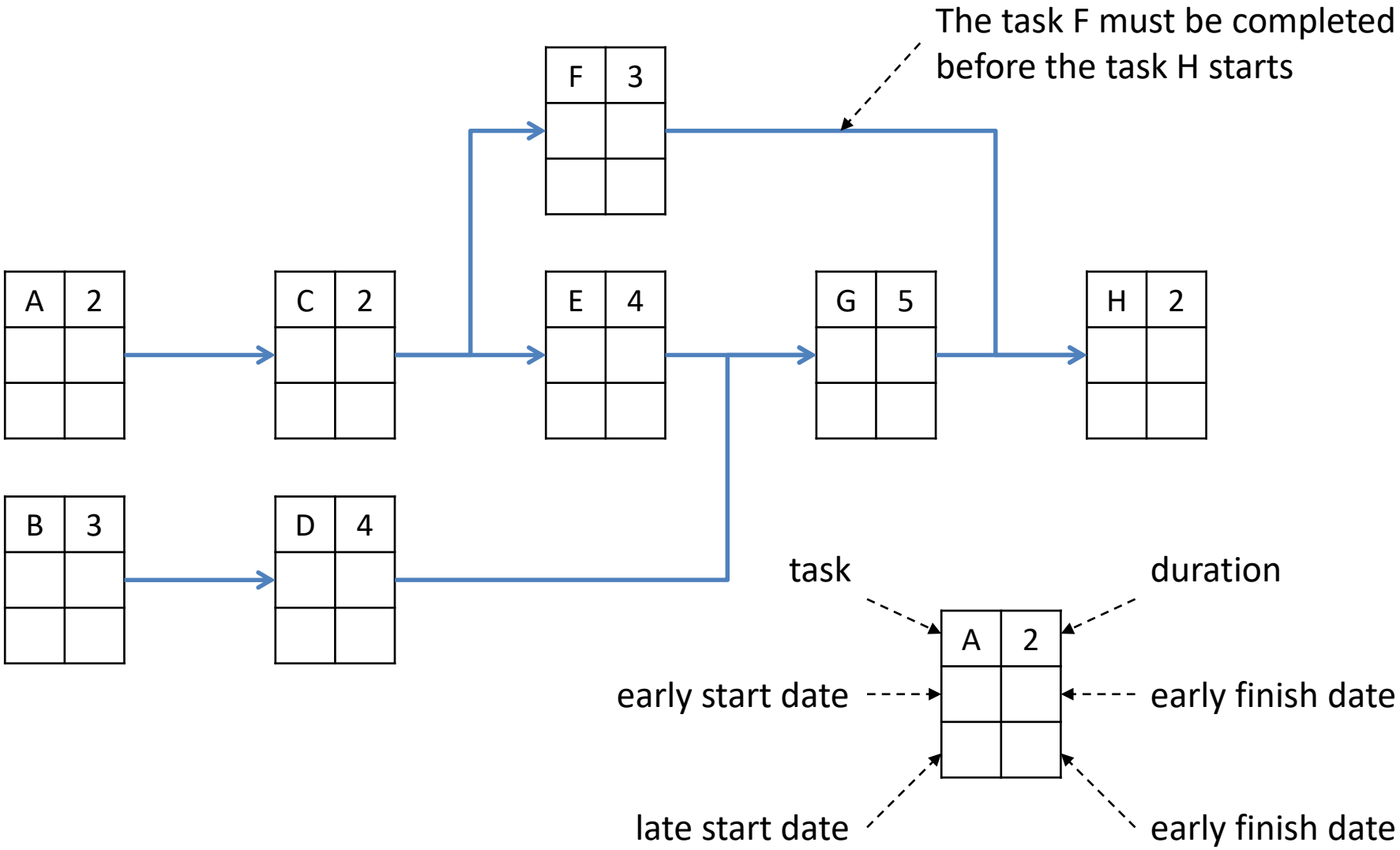


CentraleSupélec

LECTURE 4. PART 4.
SOME OTHER ALGORITHMS

CentraleSupélec

PERT diagrams



CentraleSupélec

Method to calculate early dates

Calculate early dates:

for each task (node)

mark the task as not treated

while there is a non treated task

1) Find a task T whose all predecessors are treated

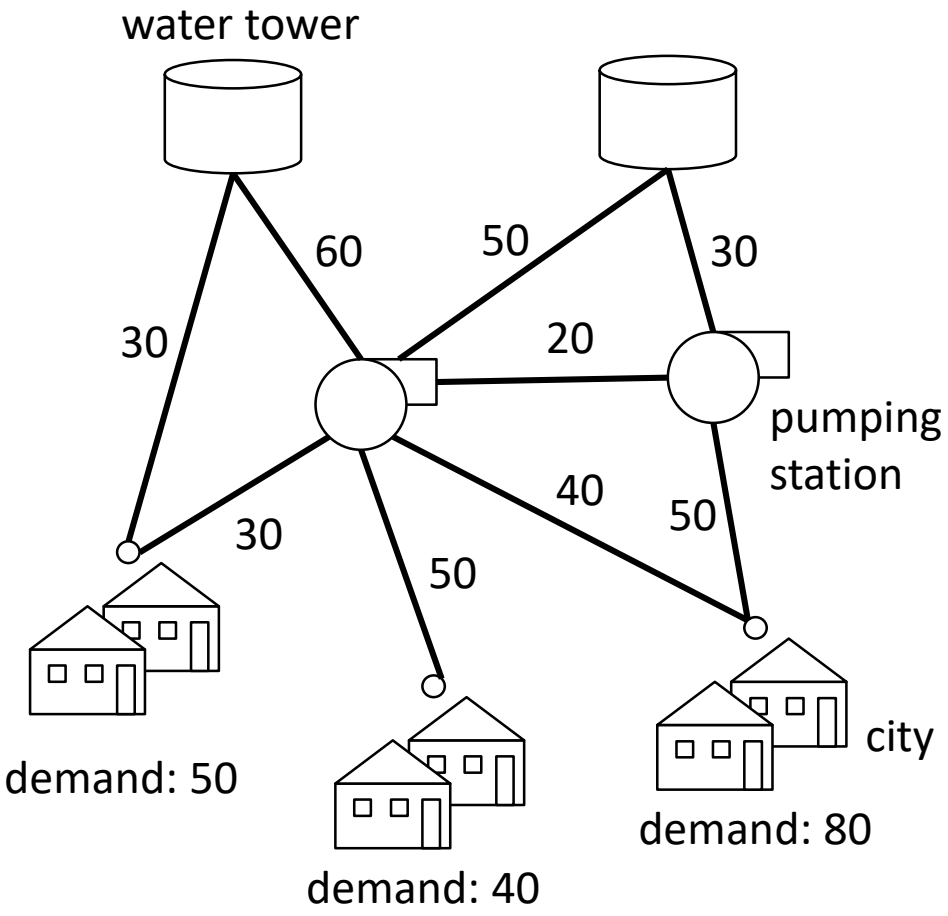
2) Set the early start date of T to the latest early finish date of its predecessors (0 if there is no predecessor)

3) Set the early finish date of T to its early start date + its duration

4) Mark T as treated

Maximum Flow

The analysis and optimization of transportation networks (for matters, energy or information) uses massively graph algorithms.

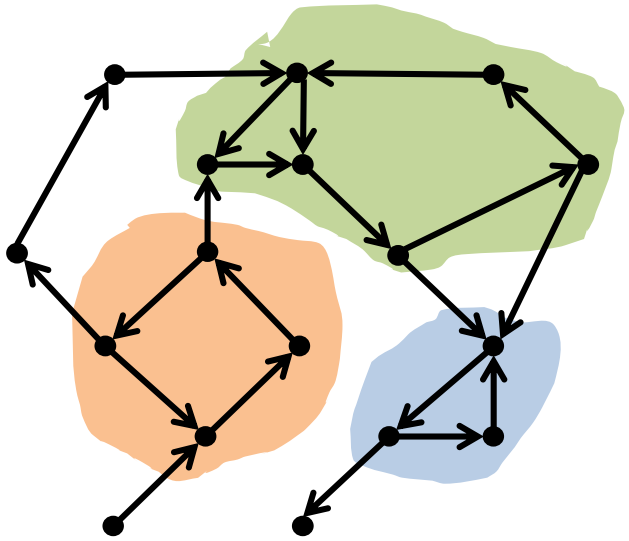


To water towers supply three cities through a network of pipes and pumping stations. The capacity of pipes is limited.
The problem is to ensure that the demand of the cities can be satisfied.
This network can be represented as a graph. The problem is then formulated as a **maximum flow** problem.
There exists elegant and efficient methods to solve this problem like the **Floyd and Fulkerson's algorithm**.

Strongly Connected Components

Clustering is another category of applications of graph theory (see lecture on “Design Structure Matrices”).

DSM	16	4	10	11	12	13	14	1	2	5	6	9	8	3	16	15	7
Heater Hoses	4									1		1					
Refrigiration Controls	10			2		1					2	1					
EATC Controls	11		2		2	1	2				2					2	
Sensors	12			2		1											
Command Distribution	13		1	1	1		1		1		1				1	1	
Actuators	14			2		1											2
Radiator	1								2	2							
Engine Fan	2					1		2		2							
Condenser	5							2	2		2		2				
Compressor	6		2	2		1				2		2	2				
Accumulator	9	1	1								2		2				
Evaporator Core	8									2	2	2		1	2		2
Heater Core	3												1		2		2
Blower Motor	16					1							2	2		2	2
Blower Controller	15			2		1									2		2
Evaporator Case	7						2						2	2	2	2	



Clustering consists in looking for **strongly connected components** of the graph, i.e. groups of nodes such that for any two nodes s and t in the group, s is reachable from t (and vice versa). There exists efficient (linear time) methods to extract strongly connected components, like **Tarjan’s algorithm**.

The Traveling Salesman Problem

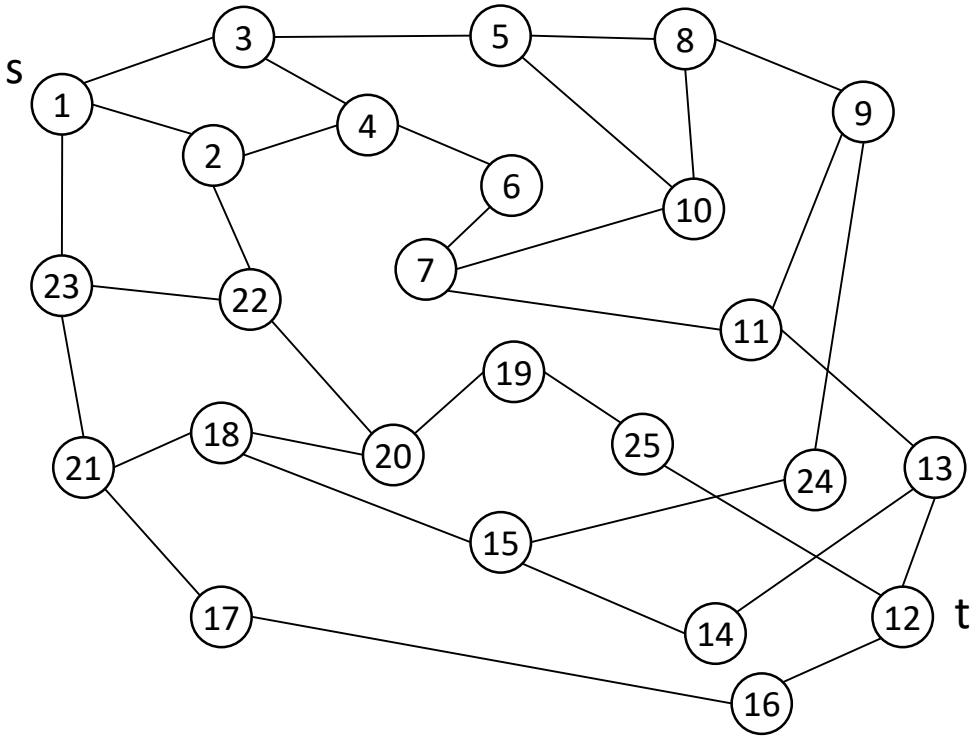
There are problems on graphs for which no efficient method is known and it is strongly suspected that **no efficient algorithm** can exist.

The **traveling salesman** problem is probably the most famous of this intractable problems. It consists, given a set of cities linked by roads (or flight routes or train tracks), in finding the shortest path going a least once through each city.
This problem can be expressed immediately in terms of graphs.

The traveling salesman problem has applications beyond logistics. It appears for instance in the optimization of the trajectory of robots, e.g. to minimize the total time taken by a CNC-controlled milling machines to drill n points in a steel plate.

Two Terminal Reliability

The **two-terminal reliability problem** is another provably hard problem.
It is stated as follows.
Given a transportation network whose nodes and edges may fail with known probabilities and two distinguished nodes s and t of that network, what is the probability that there is at least one working s-t path?



Examples of application:

- Power grids
- Gas transportation networks
- Rescue networks (in case of major accidents)
- …

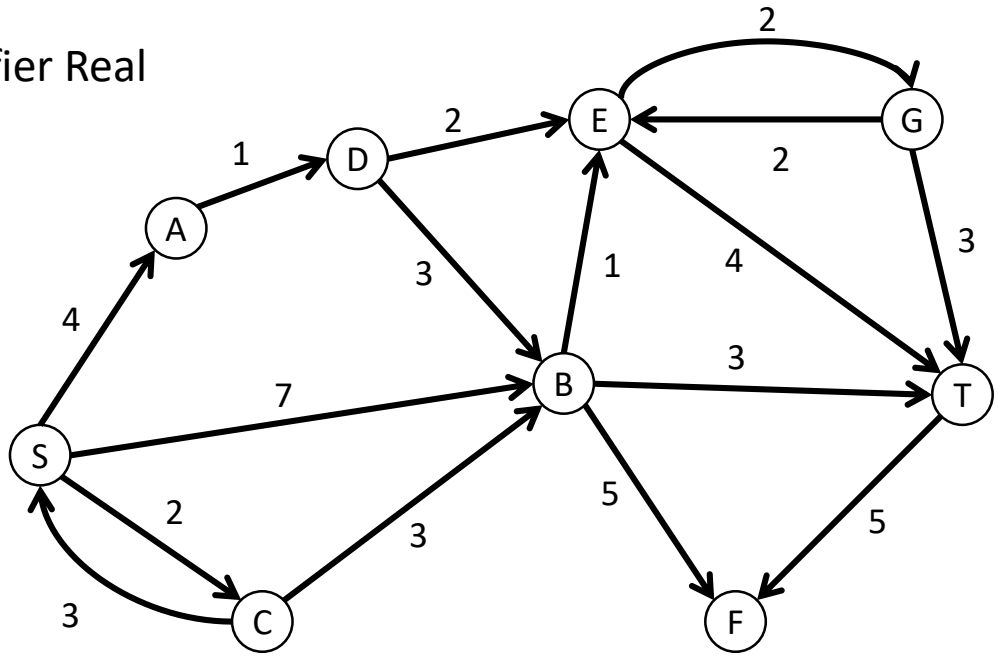
CentraleSupélec

LECTURE 4. PART 4.
TEXTUAL FORMAT FOR GRAPHS

A Textual Format for Graphs

Graph ::= “graph” Identifier Node* Edge* “end”
Node ::= “node” Identifier
Edge ::= “edge” Identifier Identifier Real

```
graph MyGraph1
    node S
    node A
    node B
    …
    node T
    edge e1 S A 4.0
    edge e2 S B 7.0
    edge e3 S C 2.0
    …
    edge e17 T F 5.0
end
```



This textual representation for graph is parsimonious: it is linear in the size of the graph conversely to representation based on matrices.

CentraleSupélec

myway.py

The Python program "myway.py" reads a graph in a text file and calculates the shortest path between a source and a target node of this graph.

As most, if not all, programs designed for this course, it is structured as follows.

1. Definition of data structures. In this lecture case, classes for nodes (Node), edges (Edge) and graphs (Graph).
2. Classes that makes it possible to read (Reader) and write (Writer) data structures into text files as well as results of calculations.
3. Classes that implement the calculation(s) of interest (Calculator).
4. Main part of the program where the classes are instanced and the calculations performed.

You can run the program either in a command shell or using IDLE.

CentraleSupélec

LECTURE 4. PART 5.
WRAP-UP & ASSIGNMENT

CentraleSupélec

Wrap-Up

- **Graphs** are used everywhere in modeling: directly as a modeling tool or indirectly as a data structure.
- **Graph theory** comes with a number of **problems** and **algorithms** to solve them that have immediate applications, e.g.:
  - Determination of the shortest path;
  - Calculation of the maximum flow;
  - Extraction of strongly connected components;
  - …
- For some of these problems, algorithms are efficient (typically linear in the size of the graph). For others, like the traveling salesman problem or the two-terminal reliability problem, **no efficient algorithm** is know, and it is strongly suspected that no efficient algorithm can exist.

CentraleSupélec

Assignment

See separated presentation.

Recommend Readings

Reference book on Algorithms:

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms (Second ed.). The MIT Press. Cambridge, MA, USA. ISBN ISBN 0-262-03293-7. 2001.

CentraleSupélec

Louis Charles Joseph Blériot (1872 -1936) is an airplane designer and one of the pioneer pilot of French aviation. He has been the first to cross the channel on July the 25th onboard of the Blériot XI.
He graduated from Ecole Centrale de Paris

Henri Marie Léonce Fabre (1882 -1984) is a French engineer and pilot. He invented the seaplane in 1910.
He graduated from Supélec.