{EPITECH.}
PARIS GRADUATE SCHOOL OF DIGITAL INNOVATION

Algorithms, Matching

Part 1. Networks and Matchings

B9 - Algorithms Matching

M-ALG-102

Overview of the module

Day 1 Networks, the matching problem and the maximum flow problem

Day 2 Data clustering and representation

Organisation of the module

- Course and exercises in python 3
- Small coding exercises, also paper + pen
- Project : explained tomorrow
- Please clone the following repository
  `https://github.com/nlehir/ALGO2`

Introductory example 1 : Max Flow

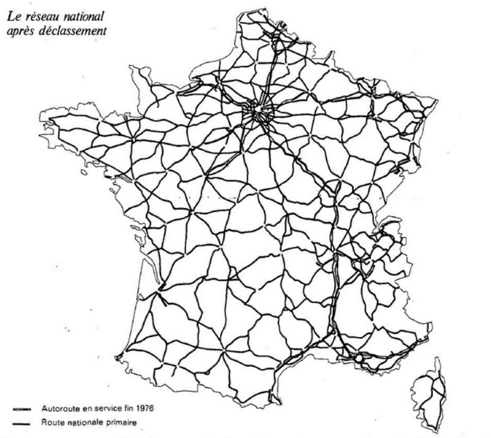


Figure: Problem 1 : transporting merchandise through a network

Introductory example 2 : Optimal allocation

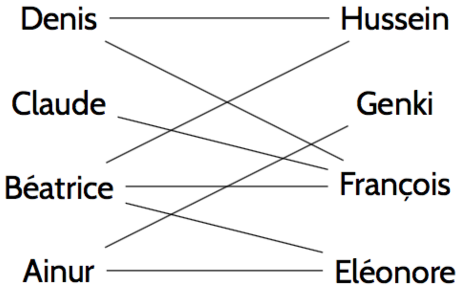


Figure: Problem 2 : Building the largest possible number of teams of 2 persons.

Introductory example 2

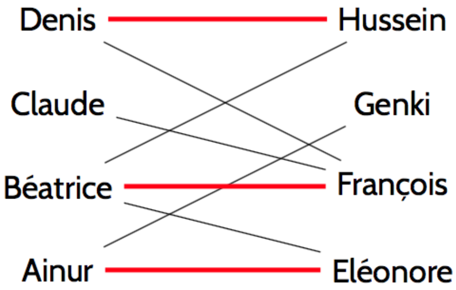


Figure: Problem 2 : not optimal allocation

Introductory example 2

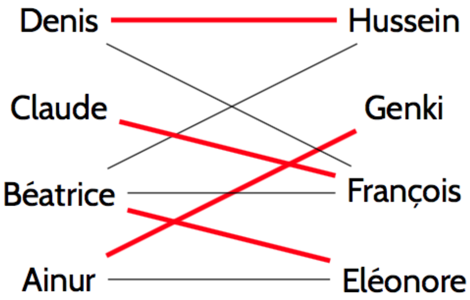


Figure: Problem 2 : optimal allocation

Introductory example 2 : allocation

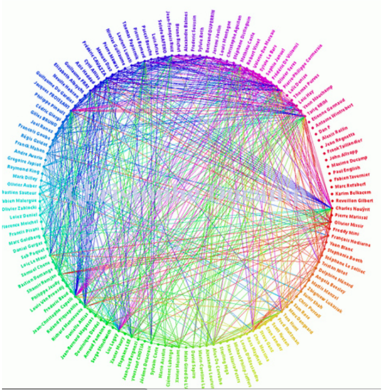


Figure: Problem 2 : not that easy if the dataset is big !

Other examples

- Assigning students to internships

Other examples

- Assigning students to internships
- Assigning machines to a task (no analogy intended !)

Introduction

We will see that these problems (flow and allocation) are **related**, and under some restrictions, **equivalent** !

Day 1

The matching problem

- Definition of the problem
- Experimental solutions
- Greedy algorithm

The Maximum flow problem

- Presentation of the problem
- Solution with the Ford-Fulkerson algorithm
- Connection with the matching problem
- More results on the two problems

Reminders on graphs

- A graph is defined by ?

Reminders on graphs

- A graph is defined by set of vertices V and a set of edges E.

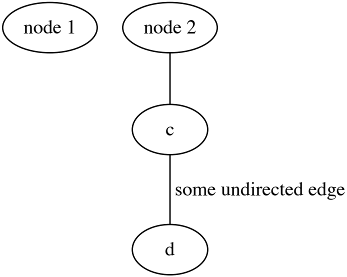


Figure: Simple graph (graphviz demo)

Reminders on graphs

- It can be **undirected**, as this one :

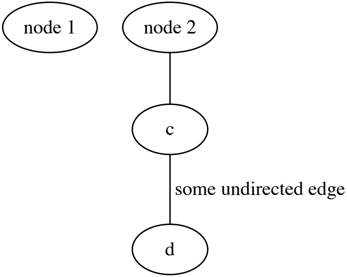
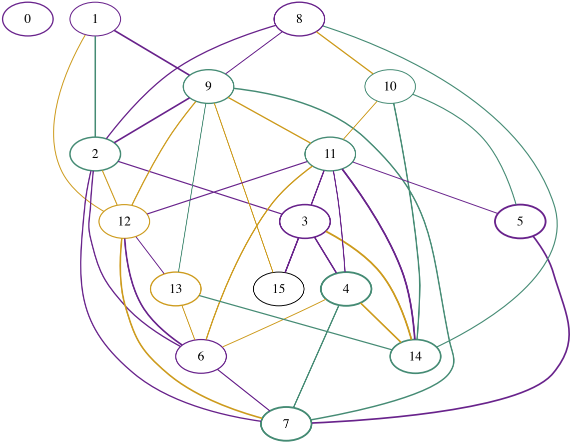


Figure: Simple graph (graphviz demo)

Reminders on graphs

Undirected graph



Reminders on graphs

- Or **directed**, as this one. (it is then called a **digraph**)

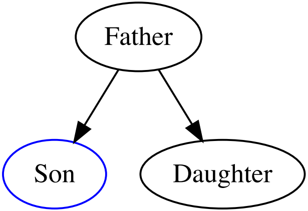
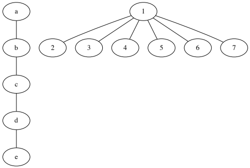


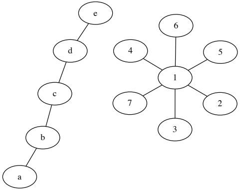
Figure: Digraph (graphviz demo)

Useful tool : graphviz

- A tool to visualize graphs
- Several **generator programs** : dot, neato



(a) Image generated with **dot**



(b) Image generated with **neato**

Warm up question

Given an **unoriented** graph with n nodes, how many edges can we build ?

Notation of a graph : $G(V, E)$

- V : set of n vertices
- E : set of edges

Warm up question

Given an **unoriented** graph with n nodes, how many edges can we build ?

Notation of a graph : $G(V, E)$

- V : set of n vertices
- E : set of edges, maximum size : $\frac{n(n-1)}{2} = \binom{n}{2} = \frac{n!}{2!(n-2)!}$

Back to our problem

Given a graph $G = (V, E)$, we want a **matching** M, which means:

- A subset of edges $M \subset E$

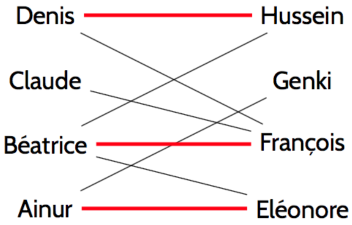


Figure: Non optimal allocation

Back to our problem

Given a graph $G = (V, E)$, we want a **matching**, which means:

- A subset of edges $M \subset E$
- Such that no pairs of edges of M are incident
- Equivalently, each node in the graph has **at most** one edge connected

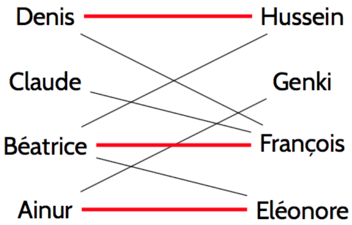


Figure: Non optimal allocation

Back to our problem

Given a graph $G = (V, E)$, we want a **matching**, which means:

- A subset of edges $M \subset E$
- Equivalently, each node in the graph has **at most** one edge connected
- Such that no pairs of edges of M are incident
- Of **Maximum size** (maximum number of edges)

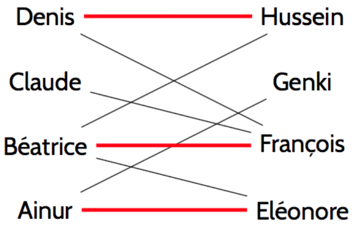


Figure: snippet

Example 1

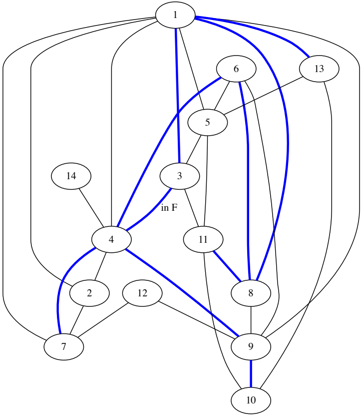


Figure: Is this a matching ?

Example 2

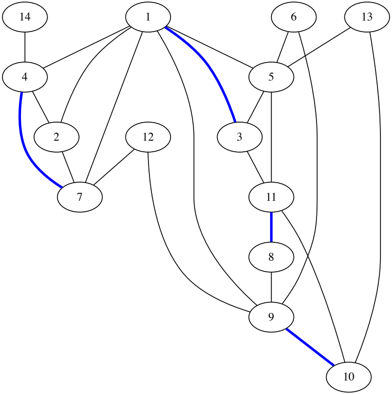


Figure: Is this a matching ?

Example 3

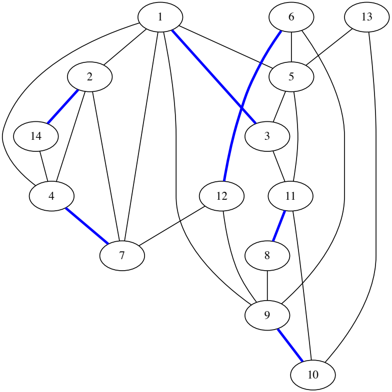


Figure: Is this an optimal matching ?

Example 4

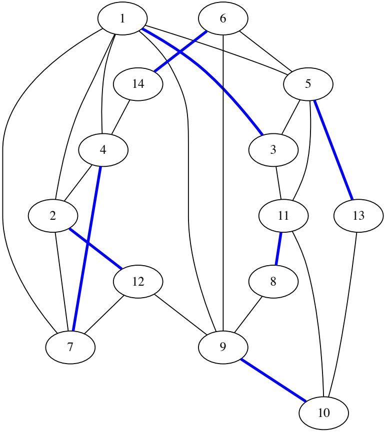


Figure: Is this an optimal matching ?

Example 5

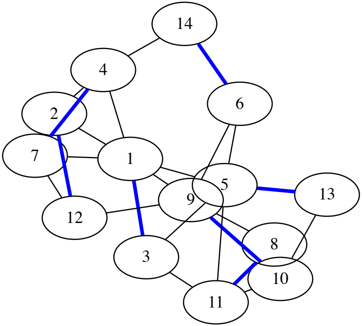


Figure: With neato

Experiments

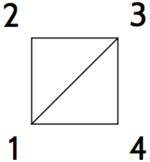
How would you code a graph ?

Experiments

How would you code a graph ?

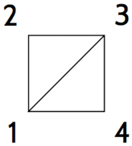
- list of sets of size 2 (for an undirected graph)
- a dictionary of successors

Coding a graph : as a list




```
g1 = [{1,2},{1,3},{2,3},{3,4},{1,4}]
```

Coding a graph : as a dictionary




```
g1 = { 1:{2,3,4}, 2:{1,3}, 3:{1,2,4}, 4:{1,3} }
```

Exercise 1

- **cd other_graphs** and please use **random_graph** to build a graph with 20 vertices and 50 edges.
- You will need to install **graphviz**

Exercise 2

- **cd other_graphs** and please use **random_graph** to build a graph with 20 vertices and 50 edges.
- You will need to install **graphviz**
- Please manually find an **optimal matching in your graph**

Algorithms

- We now have an idea of what the problem is.
- When the size of the problem is large, is it possible to find an optimal matching manually ?

Algorithms

- We now have an idea of what the problem is.
- When the size of the problem is large, is it possible to find an optimal matching manually ?
- When the size of the problem is large, is it possible to find an optimal matching by trying all possible matching ?

Exercise 3

- What would be the necessary time to enumerate all possible matchings ?
- Formally : if the graph has n nodes, what is the worst case **complexity** of the exhaustive seach ?

Algorithms

- To find an optimal solution, we want to have an algorithm.
- Thus, let us introduce some theoretical notions.

Notion of maximal and maximum matching

We will say that a matching M of cardinality $|M|$ is:

- **Maximum** if is had the maximum possible number of edges (is is thus optimal)

Notion of maximal and maximum matching

We will say that a matching M of cardinality $|M|$ is:

- **Maximum** if is had the maximum possible number of edges (is is thus optimal)
- **Maximal** if the set of edges obtained by adding any edge to it is **not a matching**. This means that $M \cup \{e\}$ is not a matching for any $e \notin M$.

Question

Exercise : is being **Maximal** the same thing has beeing **Maximum** ?

Maximum implies maximal

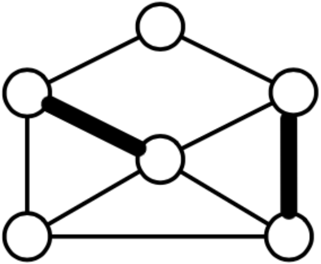
Let us show that a maximum matching is maximal.

Counter Example

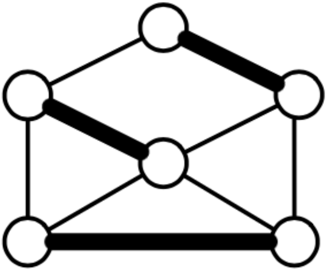
However, a matching that is maximal is **not necessary Maximum**.

Counter Example

However, a matching that is maximal is **not necessary Maximum**.
Can you find an example ?



(a) A maximal matching not maximum



(b) A maximum matching

Greedy algorithm

Can you propose a greedy algorithm to address the maximum matching problem ?

Greedy algorithm

Result: Matching M

$M \leftarrow \emptyset$;

for $e \in E$ **do**

  if $M \cup \{e\}$*is a matching* **then**

    $M \leftarrow M \cup \{e\}$

  end

end

return M

Algorithm 0: Greedy algorithm to find a matching

Greedy algorithm

- What is the type of matching algorithm returned by this algorithm ?
- What is the complexity of this algorithm ?

Greedy algorithm

- The greedy algorithm returns a **maximal** matching (proof)
- Its complexity is $\mathcal{O}(n^2)$

Greedy algorithm

- We will implement the greedy algotithm to find a maximal matching.

Exercise 4

- **cd matching_greedy** and use **generate_graph** to build a graph with a least 30 nodes.

Exercise 5

- We will use the functions written in **matching_functions** from the file **match_graphs**
- edit the lines below "CHANGE HERE" to pergorm the greedy algorithm.

Exercise 6

- Can you think of an example where the greedy algorithm gives a matching of the size **half** the size of an optimal matching ?

Changing the problem (for now)

We temporarily leave the maximum matching problem to focus on another problem : the **Maximum flow problem**

Max flow

Figure: Optimizing the quantity of something transported from one place to another, under constraints

Example

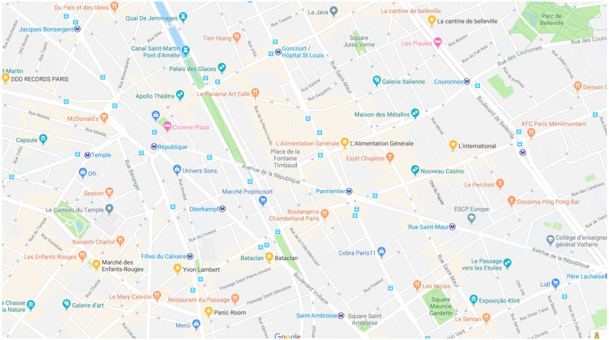


Figure: Optimizing the quantity of something transported from one place to another, under constraints

Formalizing the problem

What do we need to define the problem ?

Formalizing the problem

What do we need to define the problem ?

- A **Directed graph** $G = (E, V)$

Formalizing the problem

What do we need to define the problem ?

- A **Directed graph** $G = (E, V)$
- Each edge (u, v) must have a **capacity** $c(u, v) \geq 0$

Formalizing the problem

What do we need to define the problem ?

- A **Directed graph** $G = (E, V)$
- Each edge (u, v) must have a **capacity** $c(u, v) \geq 0$
- We define two special nodes : a **source** s and a **sink** t.

Formalizing the problem

What do we need to define the problem ?

- A **Directed graph** $G = (E, V)$
- Each edge (u, v) must have a **capacity** $c(u, v) \geq 0$
- We define two special nodes : a **source** E and a **sink** S.

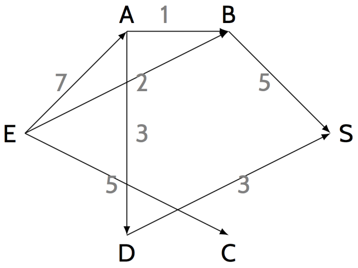


Figure: A **transport graph** with a capacities

Formalizing the problem

What do we need to define the problem ?

- A **Directed graph** $G = (E, V)$
- Each edge (u, v) must have a **capacity** $c(u, v) \geq 0$
- We define two special nodes : a **source** E and a **sink** S.
- A flow f is a function $f(u, v) \leq c(u, v)$ (+ additional constraints)

Formalizing the problem

What do we need to define the problem ?

- Each edge (u, v) must have a **capacity** $c(u, v) \geq 0$
- A flow f is a function $f(u, v) \leq c(u, v)$ (+ additional constraints)

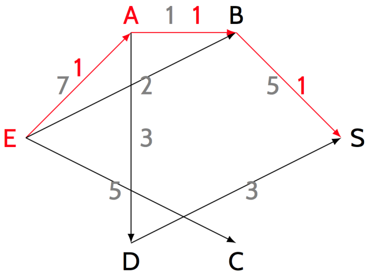


Figure: A non optimal flow

Formalizing the problem

What do we need to define the problem ?

- Each edge (u, v) must have a **capacity** $c(u, v) \geq 0$
- A flow f is a function $f(u, v) \leq c(u, v)$ (+ additional constraints)

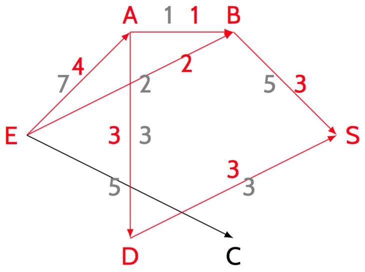


Figure: An optimal flow

Conservation of the flow

We must have :

- antisymetry : $f(v,u) = -f(u,v)$
- flow conservation : $\sum_{w \in V} f(u,w) = 0$ for $u \notin \{E, S\}$ (somme des flux entrant et sortant est nulle)
- equivalently : $\sum_{v \in V} f(v,u) = \sum_{w \in V} f(u,w)$

Maximum flow

- The **value of the flow**, noted $|f|$, is $\sum_{v \in S} f(E, v)$
- The problem is that of finding a flow with **maximum value**

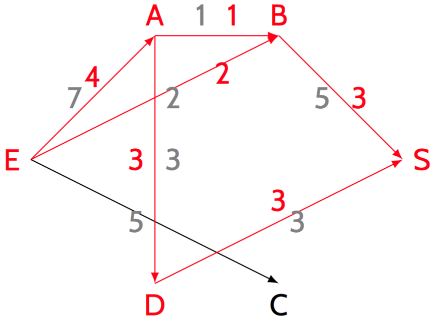


Figure: Max flow

Solution

- How would you solve this with an algorithm ?

Ford Fulkerson algorithm

We will introduce an algorithm to solve the problem. This algorithm :

- terminates
- is correct
- is polynomial

Ford Fulkerson algorithm

We will introduce an algorithm to solve the problem. This algorithm :

- terminates
- is correct
- is polynomial

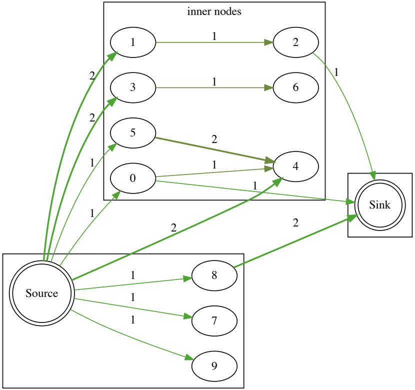
So it's great.

Residual graph

- Given a graph with capacities $c(u, v)$ and a flow $f(u, v)$, we will define its **residual graph** that has a capacity $c_r(u, v)$:

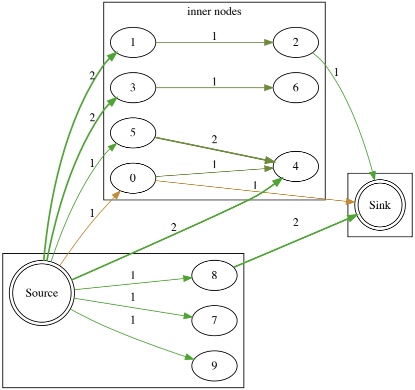
$$c_r(u, v) = c(u, v) - f(u, v) \tag{1}$$

Example of residual graph



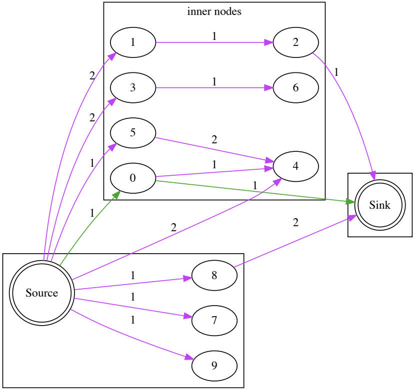
Initial Flow network

Example of residual graph



Flow
Algorithm step: 1
flow value: 1

Example of residual graph



Residual graph
Algorithm step: 2

Residual graph

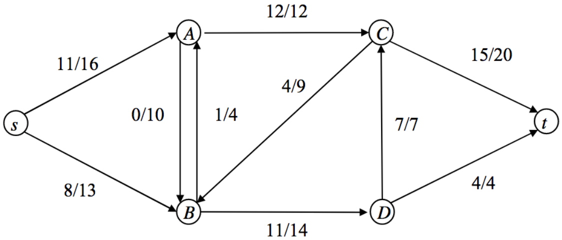


Figure: A transport graph with a flow

Residual graph

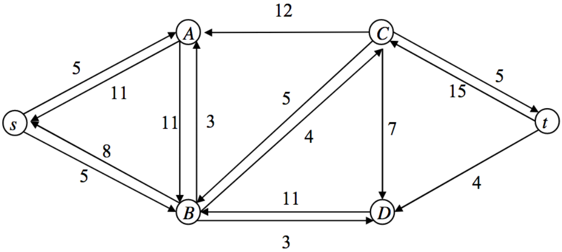
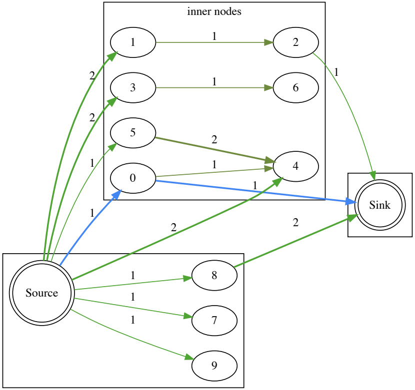


Figure: Residual graph

Augmenting path

An augmenting path is a path in the **residual graph** from the source to the sink with capacities > 0.

Augmenting path



augmenting path: [0, 1, 11]
Algorithm step: 1
path capacity: 1.0

Augmenting path

An augmenting path is a path in the **residual graph** from the source to the sink with capacities > 0.

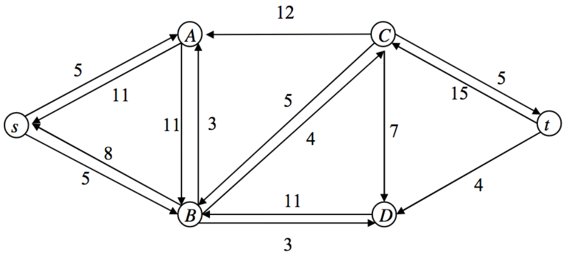


Figure: Residual graph

Augmenting path

An augmenting path is a path from the source to the sink with capacities > 0.
The Ford-Fulkerson algorithm uses augmenting paths until there are no more augmenting paths.

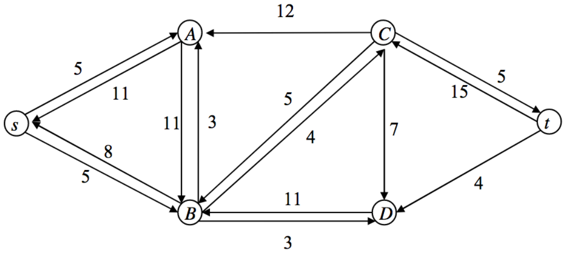


Figure: Residual graph

Ford Fulkerson algorithm

Can you deduce the algorithm from the previous remarks ?

Ford Fulkerson algorithm

Result: Flow f
for $(u, v) \in E$ **do**
  $f(u, v) = 0$
end
while $\exists \rho$ *augmenting path* **do**
  augment g with ρ
end
return f

Algorithm 1: Ford Fulkerson algorithm

Ford

Let's do an example

Ford Fulkerson algorithm

- We will implement the Ford Fulkerson algorithm (1956)

Exercise 5

- We will implement the Ford Fulkerson algorithm (1956)
- **cd ford_fulkerson** and edit **generate_flow_network** to generate a flow network.

Algorithm

- We will now use the functions contained in **ford_functions** and call them from **apply_ford _fulkerson**
- We will do it step by step.
- Let's look at the algorithm.

Exercise 6

- We will now use the functions contained in **ford_functions** and call them from **apply_ford _fulkerson**
- We will do it step by step.
- edit the function **show_residual** in order to plot the residual graph.

Exercise 7

- Modify **find_augmenting_paths** in order to find the augmenting paths.

Exercise 8

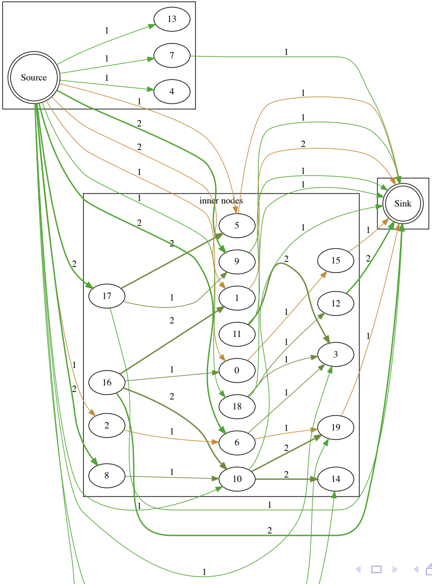
- now edit **augment_flow**

Exercise 9

- finally, edit the computation of the value of the flow

Exercise 10

- Now the algorithm should be able to run


Source
13
7
4
Sink
inner nodes
5
9
1
11
0
18
6
10
17
16
2
8
15
12
3
19
14

Complexity

What is the complexity of Ford Fulkerson ?

Complexity

What is the complexity of Ford Fulkerson ?

$$\mathcal{O}(|f^*| \times |E|) \tag{2}$$

Modification of Ford Fulkerson

What would we an intuitive and potentially faster modification of the algorithm ?

Edmonds Karp

What would we an intuitive and potentially faster modification of the algorithm ?
Use the shortest augmenting path with positive capacity.

Termination

- When the capacities are **real numbers** or **rational numbers** Ford Fulkerson terminates.

Termination

- When the capacities are **integer numbers** or **rational numbers** Ford Fulkerson terminates.
- However, when the capacities are general **real numbers**, the algorithm might not temrinate.

Link with the matching problem

- We now go back to the matching problem, in the case of a **bipartite graph**.
- We will show that in that case, we can connect the two problems.

Link with the matching problem

- We now go back to the matching problem, in the case of a **bipartite graph**.
- We will show that in that case, we can connect the two problems.
- how ?

Matching problem

We now go back to the matching problem, in the case of a **bipartite graph**.

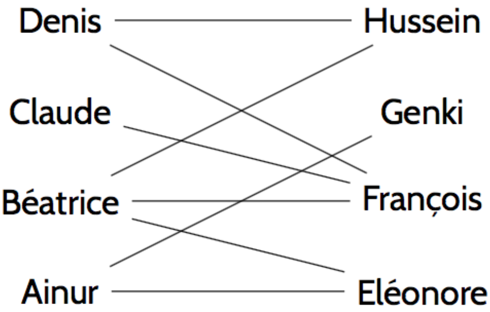


Figure: Bipartite graph

Equivalence between matching and flow

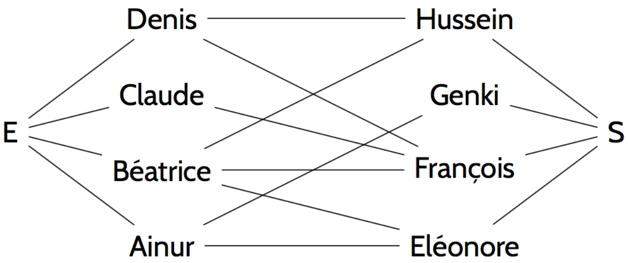


Figure: Introduce two more nodes. All edges have capacity 1. We consider **flows with integer values**

Ford Fulkerson for matching

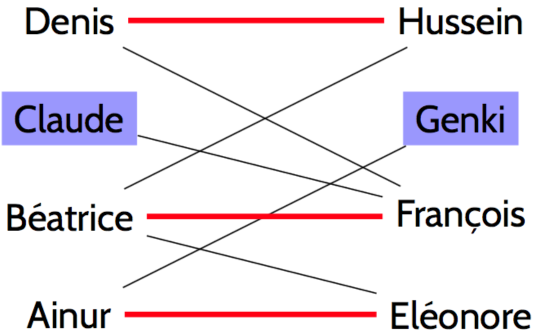


Figure: Non optimal solution

Ford Fulkerson for matching

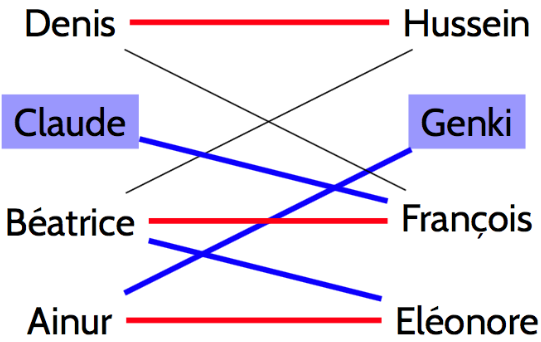


Figure: Optimal solution

Exercise 10

- We will transpose Ford Fulkerson to a bipartite graph in order to find an optimal matching.
- **cd ford_matching**
- edit **generate_matching_problem** in order to generate an instance of the problem.

Exercise 11

- Apply the algorithm on an example generated by the previous function.
- Apply the algorithm to as an instance of your choice.

Famous theorem

The maximum flow theorem is equivalent to another famous problem, the **minimum cut** theorem.

Perfect matching

In the case of a bipartite graph, what is the best matching possible ?

Perfect matching

In the case of a bipartite graph, what is the best matching possible ?
A matching where **all nodes are allocated.** It is called a **perfect** matching.
We obviously must have that the two parts of the graph are of same cardinalty.

Hall's marriage theorem

A theorem gives a condition that is necessary and sufficient for the existence of a perfect matching in a bipartite graph : the "marriage condition".
If $G = (U, V, E)$ is bipartite, the condition means that :

$$\forall X \subset U, |N_G(X)| \geq |X| \tag{3}$$

where $N_G(X)$ is the set of neighbors of X in G.

Conclusion

Ford Fulkerson and its variants (Edmonds-Karp) are polynomial. As a result thay can run on datasets that are way bigger than exhaustive search algotirhms.

See you tomorrow