Algorithmic complexity and graphs: flow networks

September 14, 2024

Max flow

Figure: Optimizing the quantity of merchandise transported from one place to another, respecting some constraints

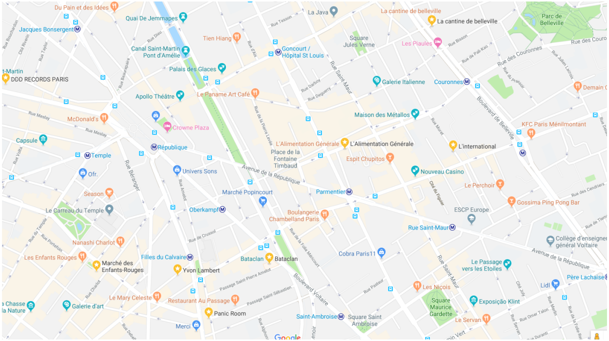


Figure: Optimizing the quantity of merchandise transported from one place to another, respecting some constraints

Formalizing the problem

We introduce the concept of **flow network (reseau de flot).**

Formalizing the problem

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

Formalizing the problem

Flow network (reseau de flot) :

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

Formalizing the problem

Flow network (reseau de flot) :

- 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.

Formalizing the problem

Flow network (reseau de flot) :

- 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.

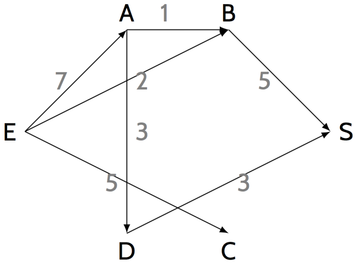


Figure: A **flow network (reseau de flot)** with capacities

Formalizing the problem

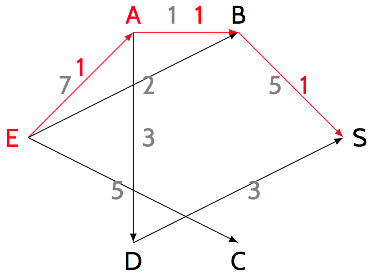
Flow network (reseau de flot) :

- 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, see below)

Formalizing the problem

Flow network (reseau de flot) :

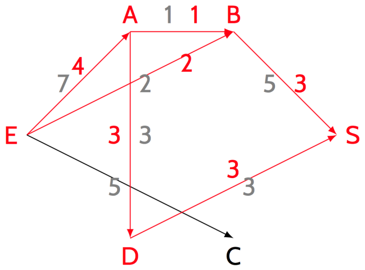
- Each edge (u, v) must have a **capacity** $c(u, v) \geq 0$
- A **flow** f is a function $f(u, v) \leq c(u, v)$ (+ additional constraints, see below)



Formalizing the problem

Flow network (reseau de flot) :

- 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, see below)



Definition of a flow

We must also have :

- antisymmetry : $f(v,u) = -f(u,v)$

Definition of a flow

We must also have :

- antisymmetry : $f(v,u) = -f(u,v)$
- flow conservation : $\sum_{v \in V} f(u,v) = 0$ for any $u \notin \{E, S\}$

Other formulation of the flow conservation

Exercice 1 : Other formulation of the flow conservation
Let us show that for a flow f, we have for any node $u \notin \{e, s\}$:

$$\sum_{f(u,v)>0} f(u, v) = \sum_{f(v,u)>0} f(v, u) \tag{1}$$

Maximum flow

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

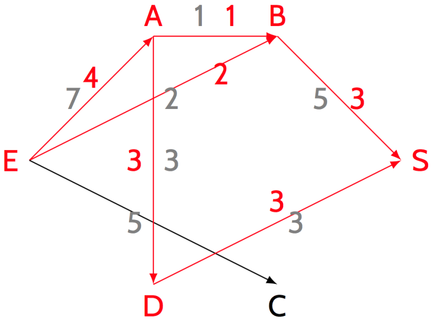


Figure: Max flow

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 is a good algorithm.

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 $c_r(u,v)$:

$$c_r(u,v) = c(u,v) - f(u,v) \tag{2}$$

Example of residual graph

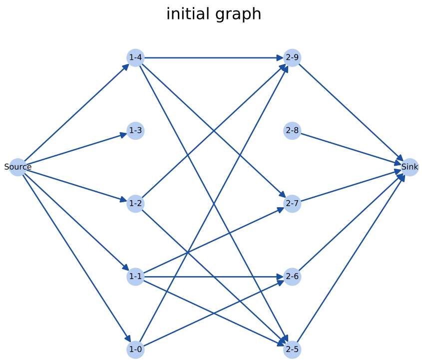


Figure: All initial capacities set to 1

Example of residual graph

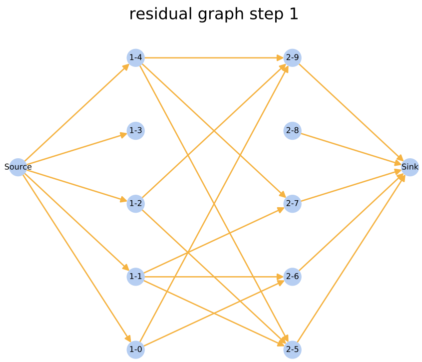


Figure: All initial capacities set to 1

Example of residual graph

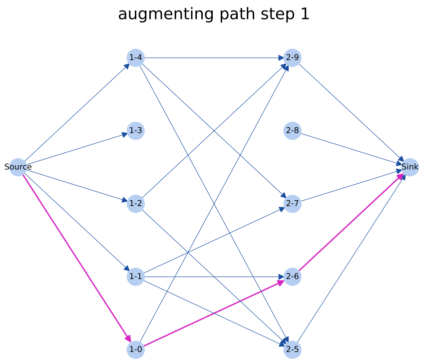


Figure: All initial capacities set to 1

Example of residual graph

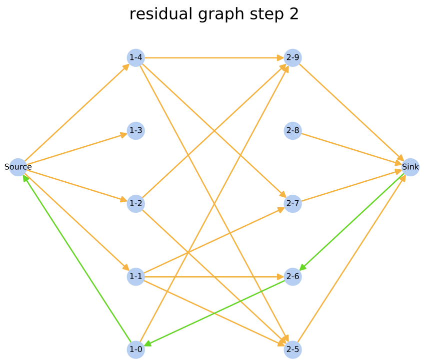


Figure: All initial capacities set to 1

Example of residual graph

augmenting path step 2

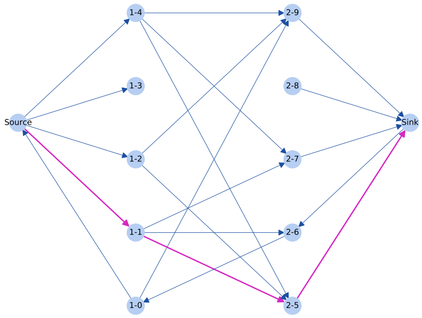


Figure: All initial capacities set to 1

Example of residual graph

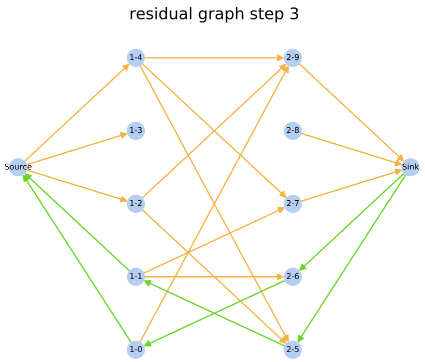


Figure: All initial capacities set to 1

Example of residual graph

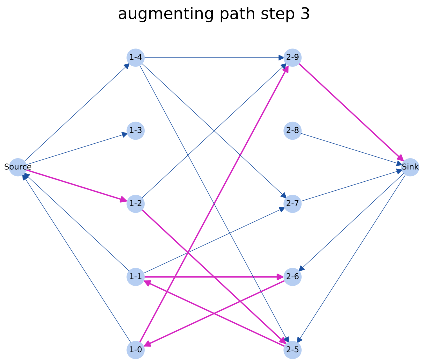


Figure: All initial capacities set to 1

Flow network and residual graph

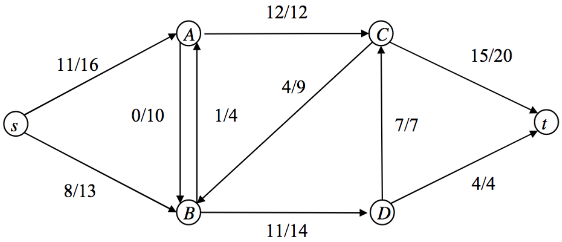


Figure: Another flow network

Flow network and residual graph

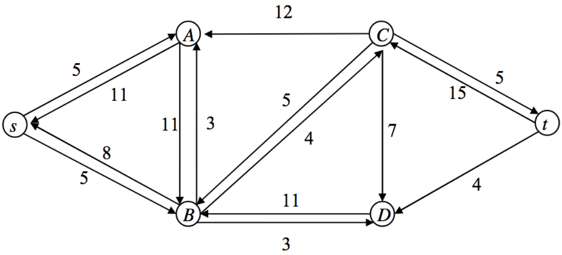


Figure: Residual graph

Augmenting path

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

Augmenting path

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

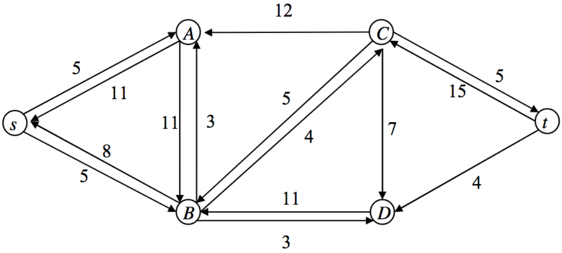


Figure: Residual graph

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.

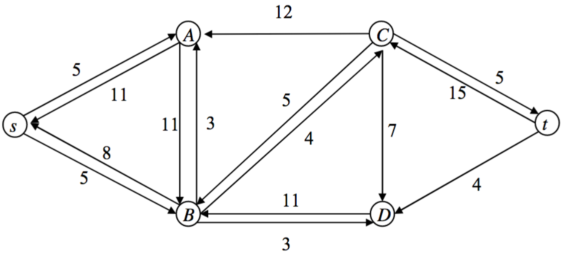


Figure: Residual graph

Ford Fulkerson algorithm

Can you deduce the algorithm from the previous remarks ?

Ford Fulkerson algorithm

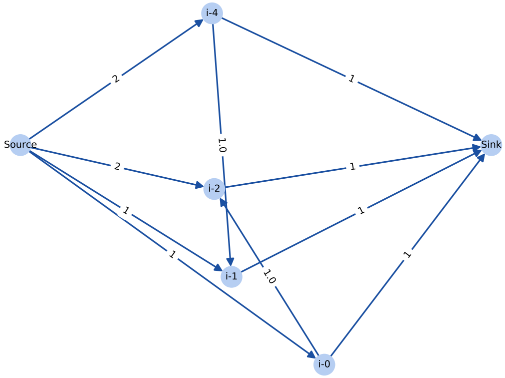
Result : Flow f
for $(u, v) \in E$ **do**
    $f(u, v) = 0$
end
while $\exists \rho$ *augmenting path* **do**
    augment f with ρ
end
return f

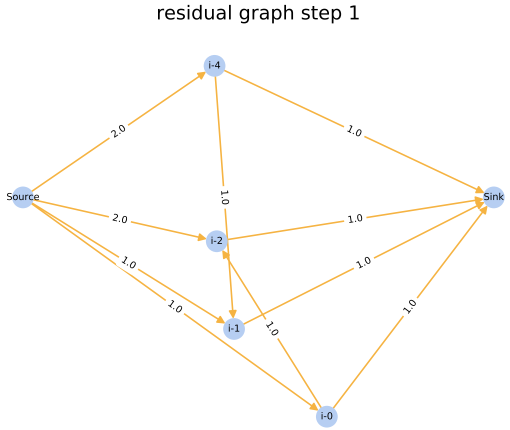
Algorithme 1 : Ford Fulkerson algorithm

Ford-fulkerson algorithm

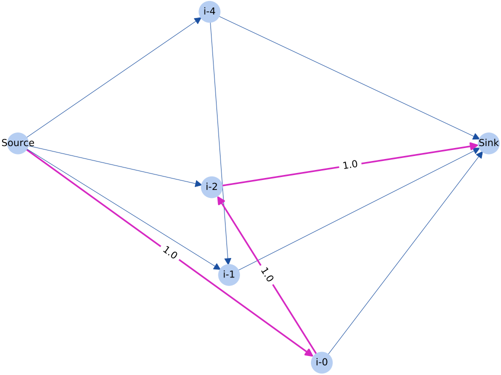
Let us apply the algorithm to some instances:

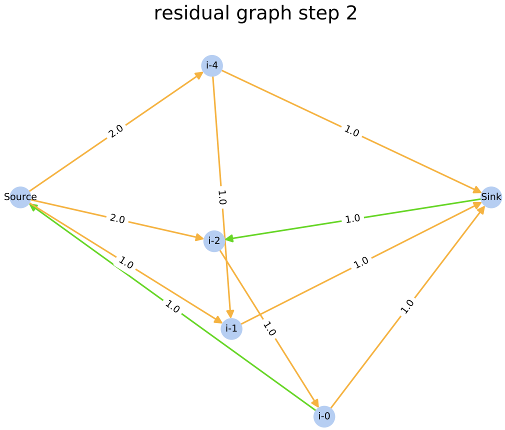
initial graph



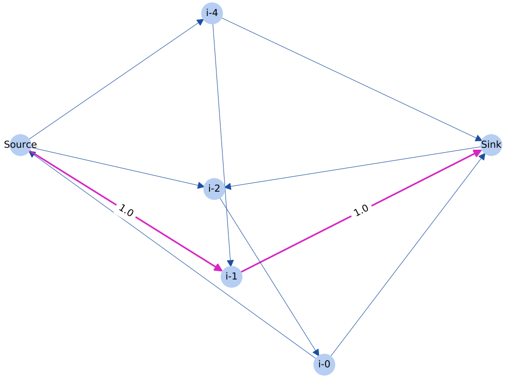

residual graph step 1
i-4
Source
Sink
i-2
i-1
i-0
2.0
1.0
1.0
2.0
1.0
1.0
1.0
1.0
1.0
1.0

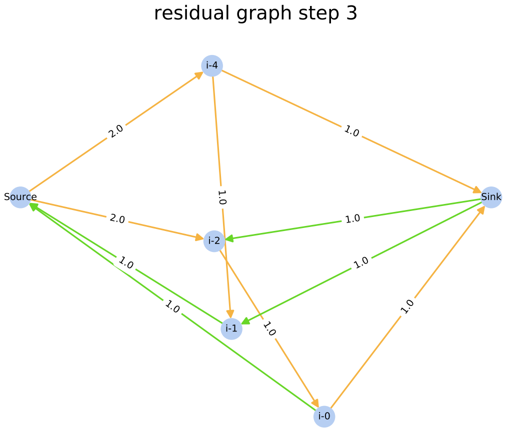
augmenting path step 1

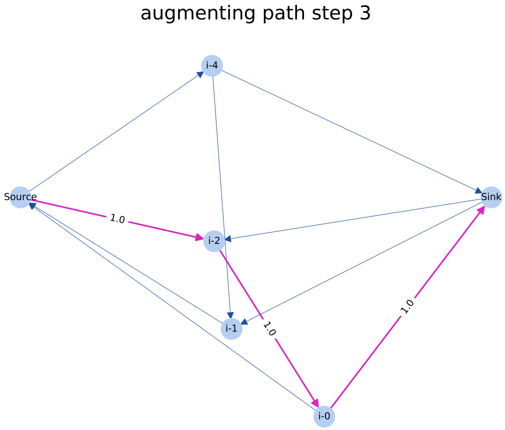


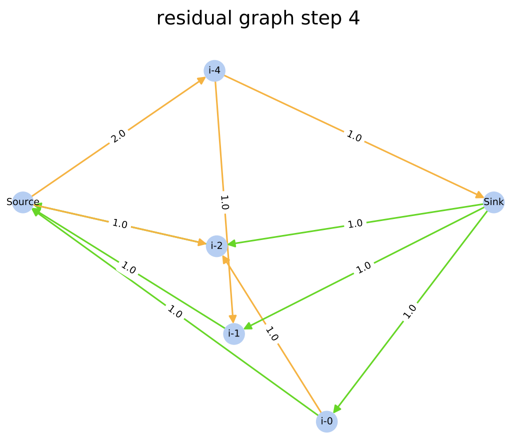

residual graph step 2
i-4
2.0
1.0
1.0
Source
Sink
2.0
1.0
i-2
1.0
1.0
1.0
1.0
i-1
1.0
1.0
i-0

augmenting path step 2

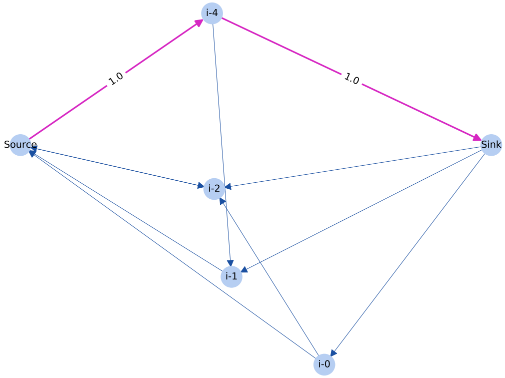


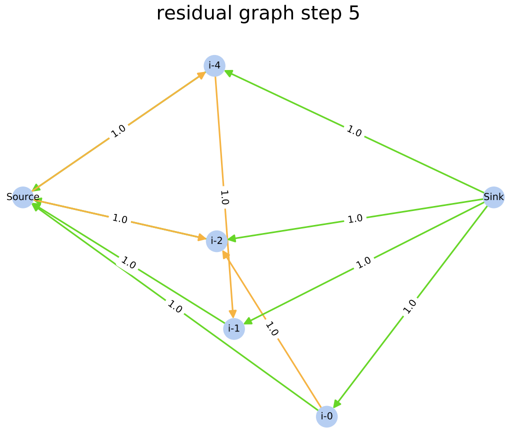

residual graph step 3
i-4
Source
Sink
i-2
i-1
i-0
2.0
1.0
1.0
2.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0


augmenting path step 3
i-4
Source
Sink
i-2
i-1
i-0
1.0
1.0
1.0


residual graph step 4
i-4
Source
Sink
i-2
i-1
i-0
2.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0

augmenting path step 4




residual graph step 5
i-4
Source
Sink
i-2
i-1
i-0
1.0

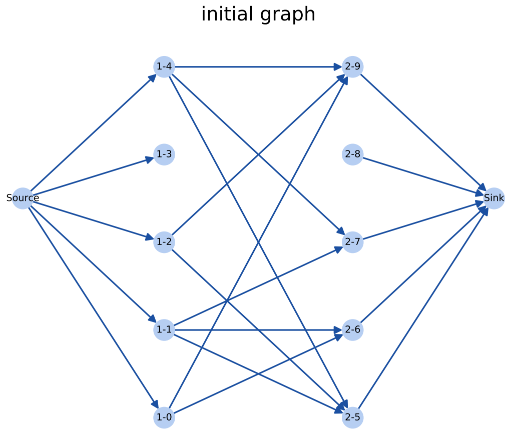
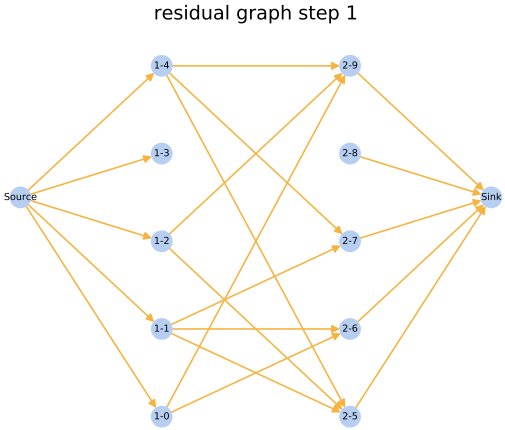
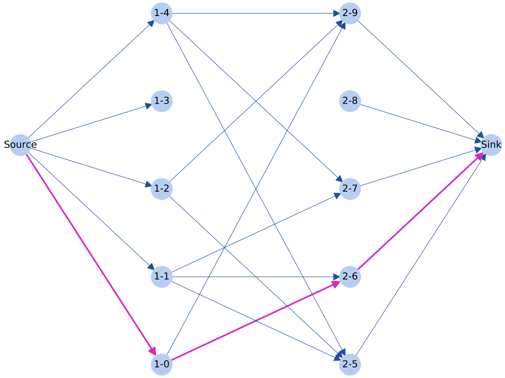
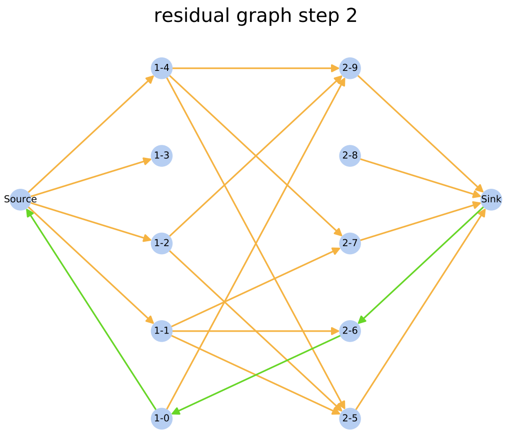


Figure: Initial capacity set to 1

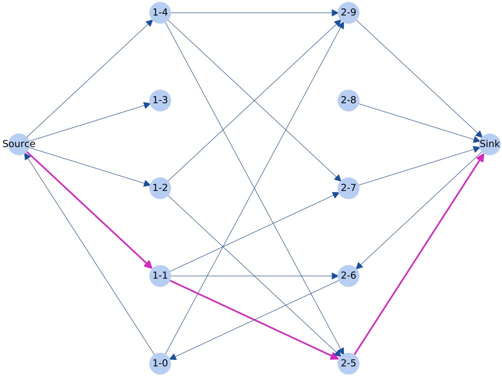

residual graph step 1
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

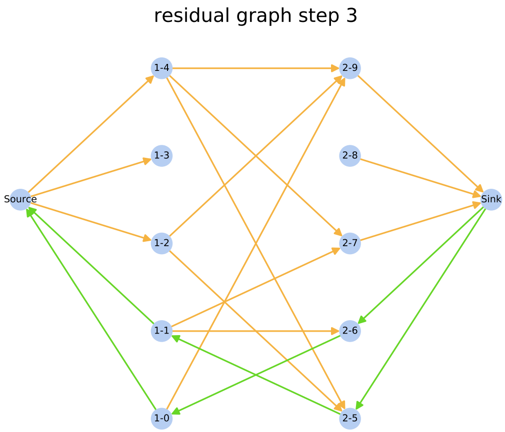
augmenting path step 1



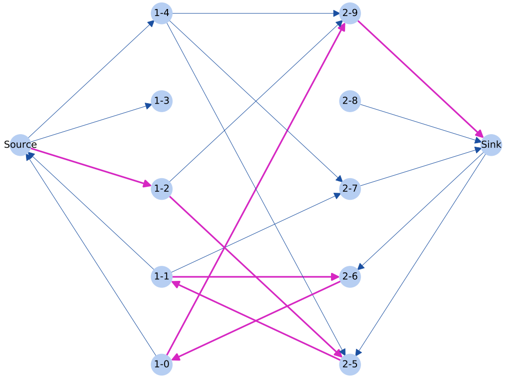

residual graph step 2
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

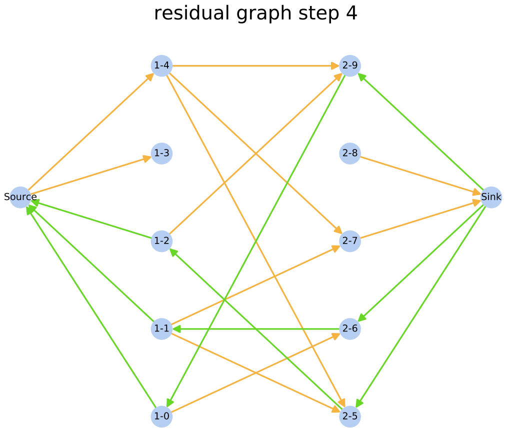
augmenting path step 2



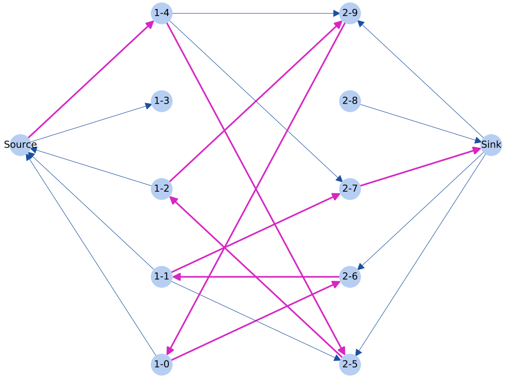

residual graph step 3
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

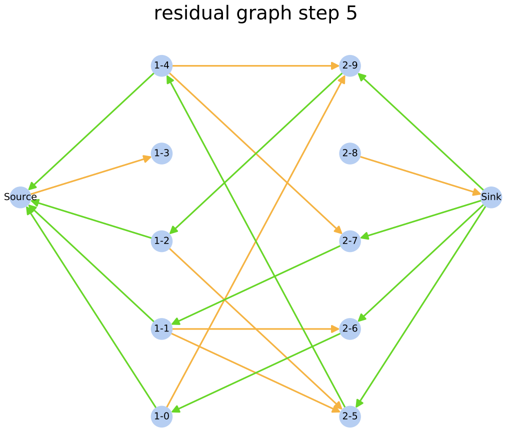
augmenting path step 3




residual graph step 4
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

augmenting path step 4




residual graph step 5
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

Ford Fulkerson algorithm

- We will implement the Ford Fulkerson algorithm (1956) on a general graph.

Numpy exercise

Exercice 2 : **Numpy arrays.**
First, we will do an exercise to get more familiar with numpy.
Please follow the notebook **numpy_demo/numpy_demo.ipynb**.

Ford Fulkerson algorithm

Exercice 3 : We will implement the Ford Fulkerson algorithm (1956)

- **cd ford_fulkerson/** and use **main_generate_flow_network.py** to generate a flow network.

Algorithm

- We will now use the functions contained in **ford_functions.py** and call them from **main_process_flow_network.py**

Algorithm

Exercice 4 : step 1

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

Algorithm

Exercice 4 : step 2

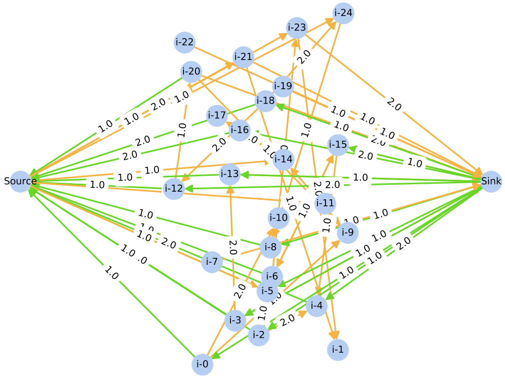
- now edit **augment_flow()**

Algorithm

Exercice 4 : step 3

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

- Now the algorithm should be able to run residual graph step 15



Complexity

Complexity of Ford Fulkerson:

$$\mathcal{O}(|f^*| \times |E|) \tag{3}$$

Termination

- When the capacities are **integer 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** (that can be irrationnal), the algorithm might not terminate.

Modification of Ford Fulkerson

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

Modification of the algorithm

What would we an intuitive and potentially faster modification of the algorithm ?
Use the shortest augmenting path with strictly positive capacity.
(**Edmonds-Karp algorithm, 1972**).
The time complexity is now $\mathcal{O}(|V||E|^2)$, thus **independent** on $|f^*|$.

Link with the matching problem

- We now go back to the matching problem, in the case of a **bipartite graph** ("problème d'affectation")
- We will show that in that case, we can connect the two problems.

Bipartite graph

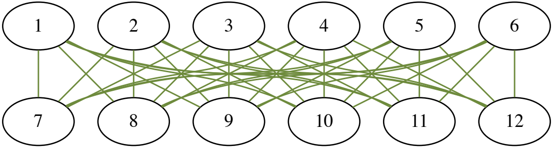


Figure: Complete bipartite graph (not all bipartite graphs are complete)

Matching problem

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

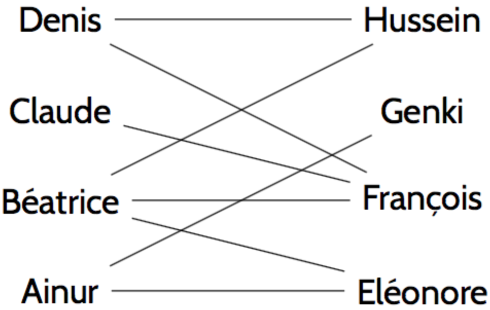


Figure: Bipartite graph

Equivalence between matching and flow

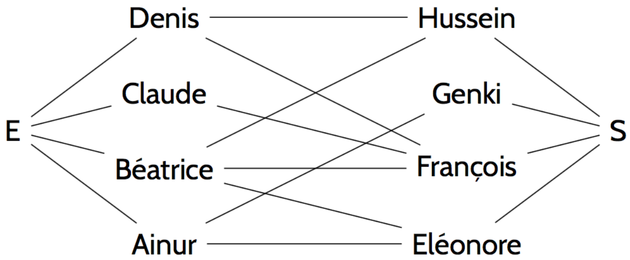


Figure: Introduce two more nodes. All edges have capacity 1. We consider **flows with integer values**

Ford Fulkerson for matching

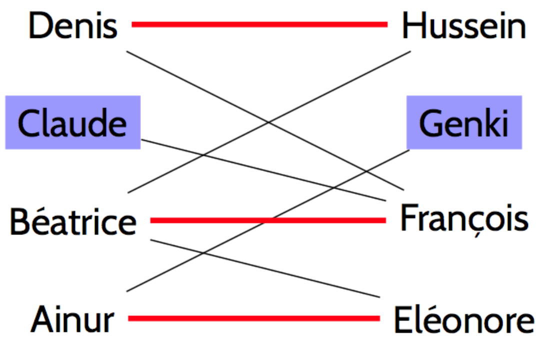


Figure: Non optimal solution

Ford Fulkerson for matching

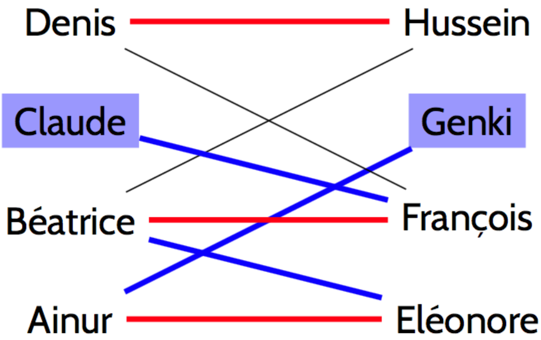


Figure: Optimal solution

Connection

Exercice 4 : Find a connection between the two problems

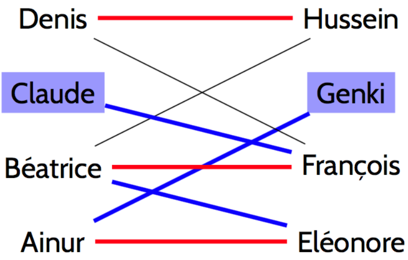
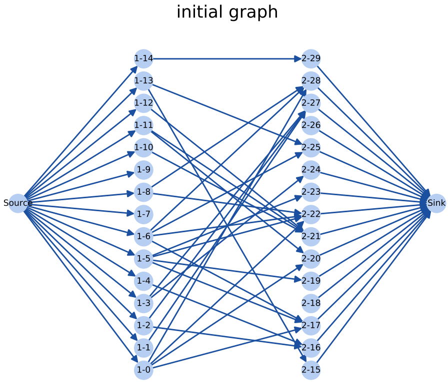
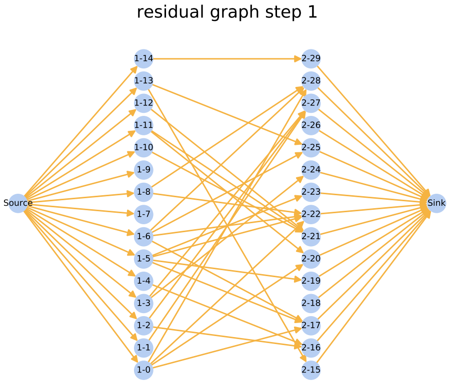


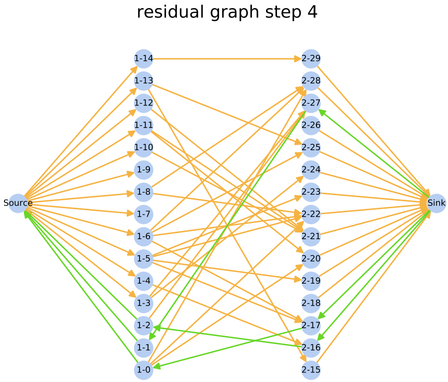
Figure: Optimal solution

Ford Fulkerson and matching

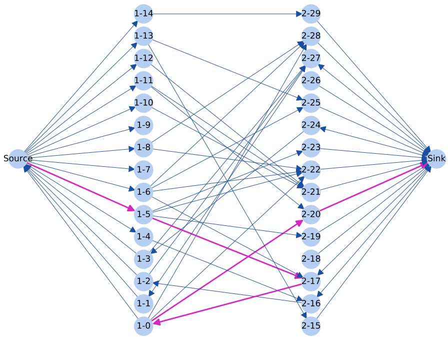
- In the folder **ford_matching**/, the scripts apply Ford Fulkerson to a bipartite graph in order to find an optimal matching.


initial graph
Source
1-14
1-13
1-12
1-11
1-10
1-9
1-8
1-7
1-6
1-5
1-4
1-3
1-2
1-1
1-0
2-29
2-28
2-27
2-26
2-25
2-24
2-23
2-22
2-21
2-20
2-19
2-18
2-17
2-16
2-15
Sink

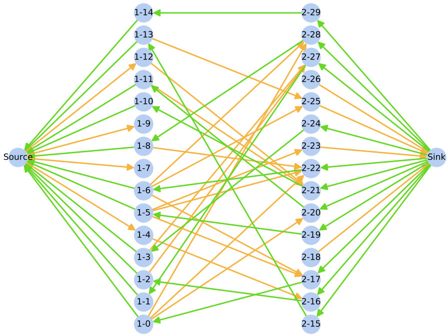

residual graph step 1
Source
Sink
1-14
1-13
1-12
1-11
1-10
1-9
1-8
1-7
1-6
1-5
1-4
1-3
1-2
1-1
1-0
2-29
2-28
2-27
2-26
2-25
2-24
2-23
2-22
2-21
2-20
2-19
2-18
2-17
2-16
2-15


residual graph step 4
Source
Sink
1-14
1-13
1-12
1-11
1-10
1-9
1-8
1-7
1-6
1-5
1-4
1-3
1-2
1-1
1-0
2-29
2-28
2-27
2-26
2-25
2-24
2-23
2-22
2-21
2-20
2-19
2-18
2-17
2-16
2-15

augmenting path step 5



residual graph step 12



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 must have that the two parts of the graph are of same cardinalty in order to have a perfect matching.

Hall's marriage theorem

This 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 :

$$\forall X \subset U, |N_G(X)| \geq |X| \tag{4}$$

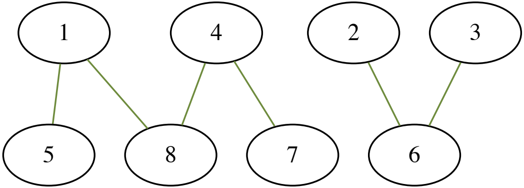
where $N_G(X)$ is the set of neighbors of X in G.

Hall's theorem

Exercice 5 : Application of the theorem.
Can you think of a graph that does not abide by the marriage condition and thus has **no perfect matching** ?

Illustration of Hall's theorem

Exercice 5 : Application of the theorem



Case of a non bipartite graph

In the case of a **non-bipartite**, we can not use the Ford-Fulkerson algorithm in order to solve the matching problem.
In that case, other methods exist such as the **Blossom algorithm.**
https://en.wikipedia.org/wiki/Blossom_algorithm

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.