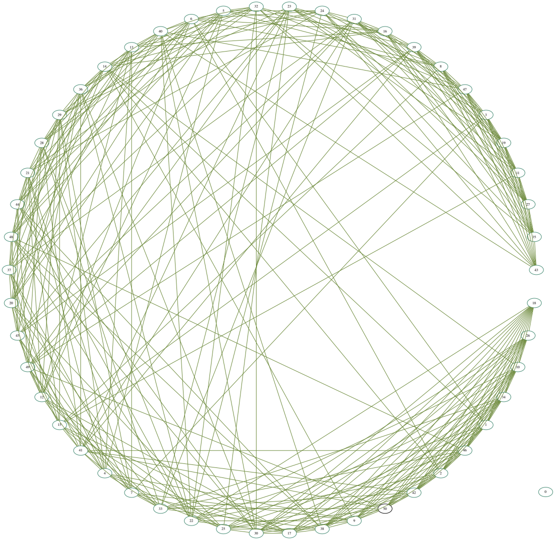
{EPITECH.}
PARIS GRADUATE SCHOOL OF DIGITAL INNOVATION

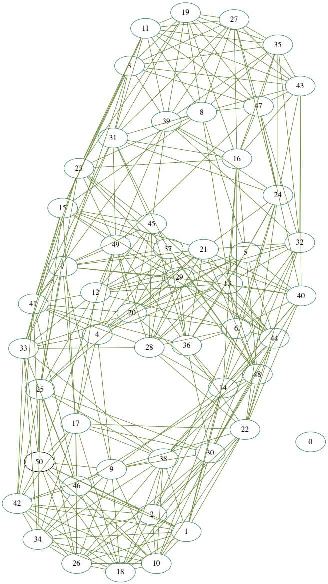
Algorithms, Matching

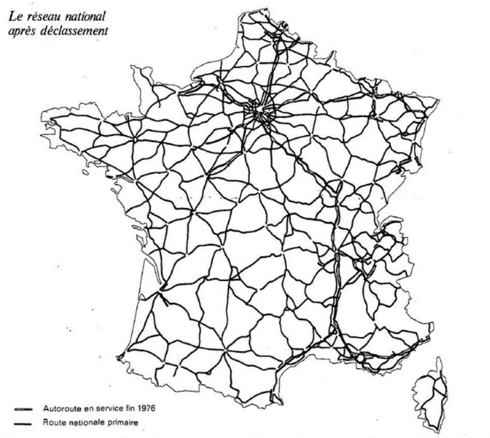
Part 1. Networks and Matchings

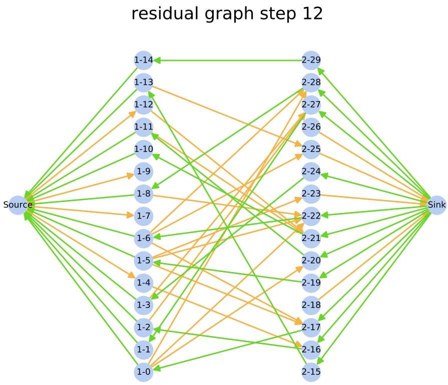
B9 - Algorithms Matching

M-ALG-102

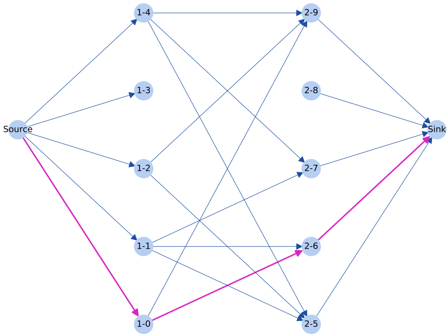


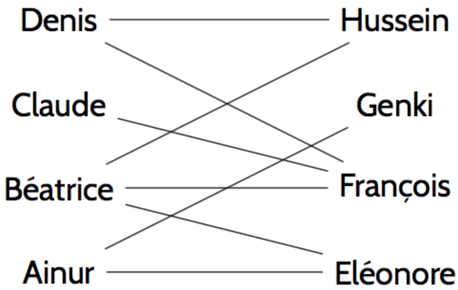

19
27
11
35
3
43
47
8
39
31
16
23
24
15
45
32
49
37
21
5
7
29
13
40
12
41
20
6
44
4
33
28
36
48
25
14
17
22
0
30
38
50
9
46
42
2
1
34
26
18
10

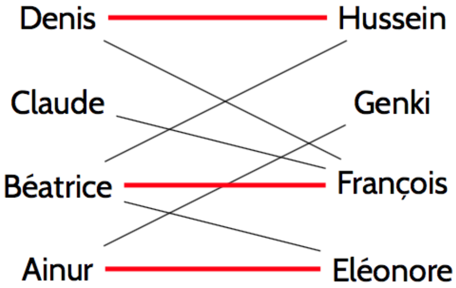

Le réseau national
après déclassement
Autoroute en service fin 1976
Route nationale primaire

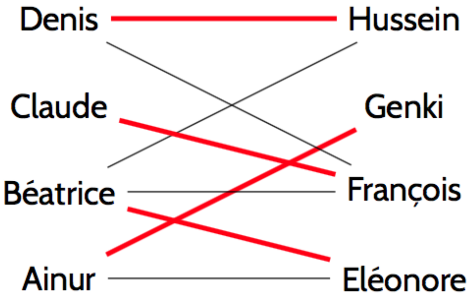

residual graph step 12
Source
Sink
1-14
1-13
1-12
1-11
1-10
1-9
1-8
1-7
1-6
1-5
1-4
1-3
1-2
1-1
1-0
2-29
2-28
2-27
2-26
2-25
2-24
2-23
2-22
2-21
2-20
2-19
2-18
2-17
2-16
2-15

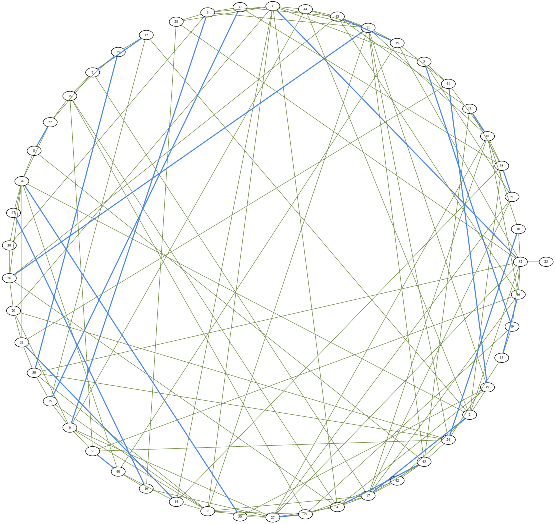
augmenting path step 1



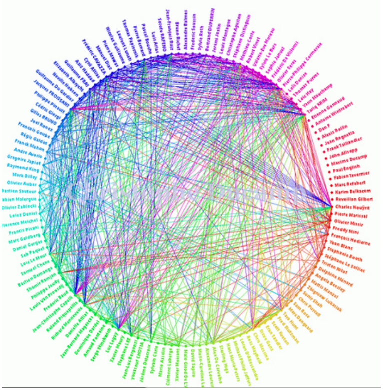

Denis
Claude
Béatrice
Ainur
Hussein
Genki
François
Eléonore

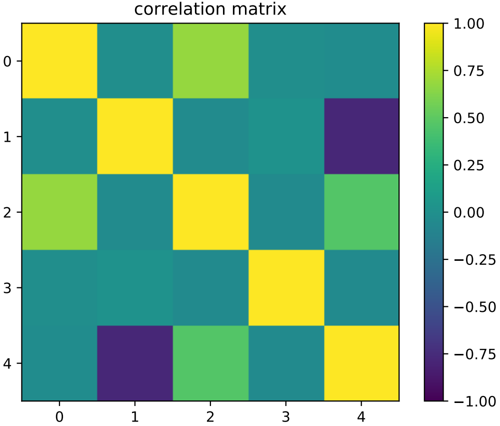

Denis
Claude
Béatrice
Ainur
Hussein
Genki
François
Eléonore

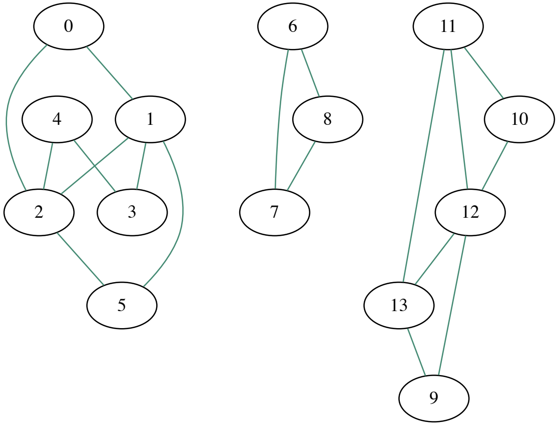

Denis
Hussein
Claude
Genki
Béatrice
François
Ainur
Eléonore



Matching size: 21
Algo step: 128
Nb nodes: 50




correlation matrix
0
1
2
3
4
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


0
6
11
4
1
8
10
2
3
7
12
5
13
9

- The content of the course will sometimes be mathematical.
- Don't hesitate to ask if you need more reminders.

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`

Libs

Day 1 networkx, numpy, matplotlib

Day 2 numpy, matplotlib, pandas, sklearn

Day 1

The matching problem

- Definition of the problem
- Experimental solutions
- Brute force algorithm
- 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

Introductory example 1 : Max Flow

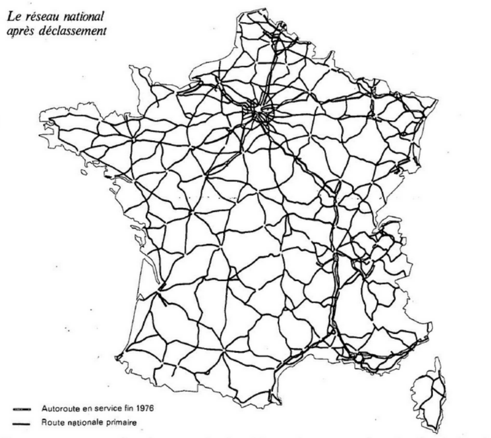


Figure: Problem 1 : transporting merchandise through a network

Introductory example 2 : Maximum matching (Optimal allocation)

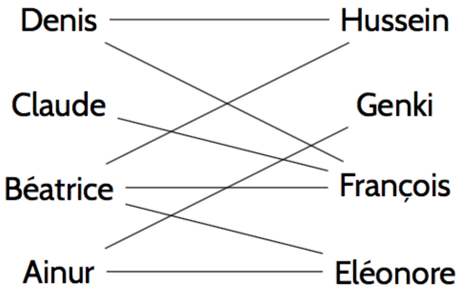


Figure: Problem 2 : Building the largest possible number of teams of 2 persons.

Introductory example 2

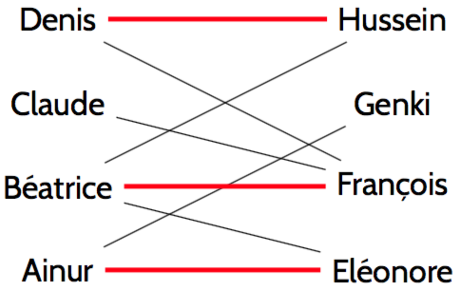


Figure: Problem 2 : not optimal allocation

Introductory example 2

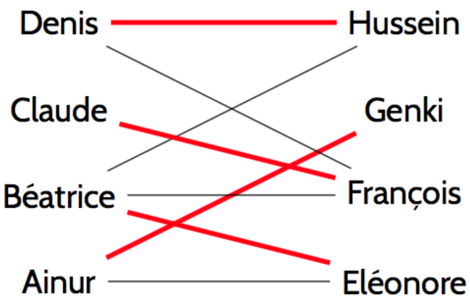


Figure: Problem 2 : optimal allocation

Other examples

- Assigning students to internships

Other examples

- Assigning students to internships
- Assigning machines to a task

Summary

- Today we will work on **connnecting the two problems**.

Summary

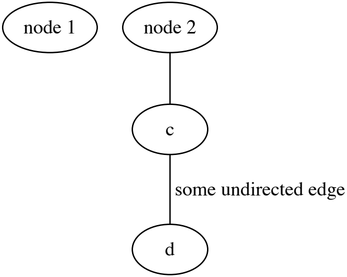
- Today we will work on **connnecting the two problems**.
- Under some restrictions, the two problems **equivalent**.

Reminders on graphs

- A graph is defined by ?

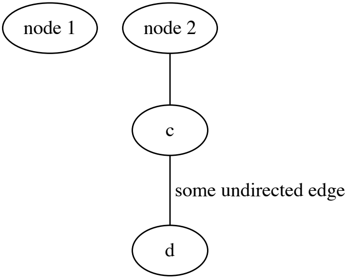
Reminders on graphs

- A graph is defined by set of **vertices** (or **nodes**) V and a set of **edges** E.



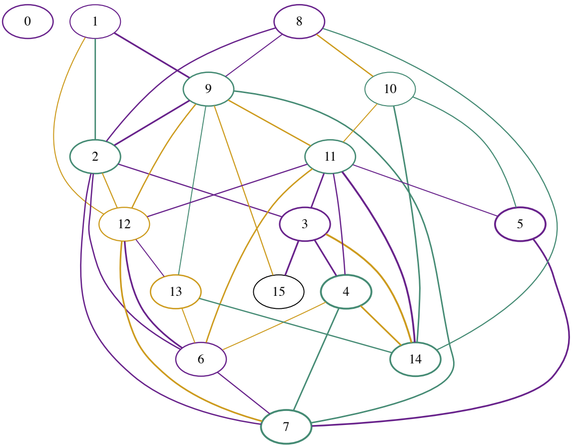
Reminders on graphs

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



Reminders on graphs

Undirected graph



Reminders on graphs

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

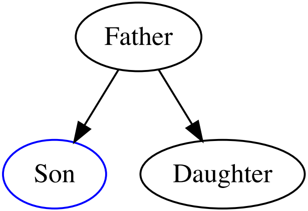
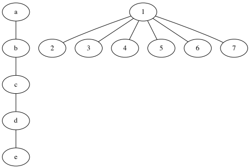


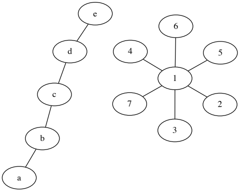
Figure: Digraph (graphviz demo)

Useful tool : graphviz

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



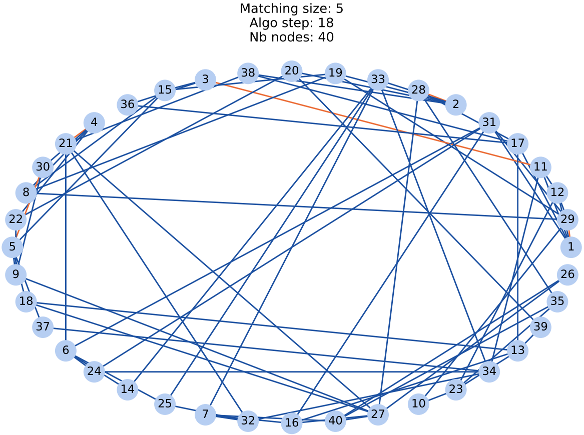
(a) Image generated with **dot**



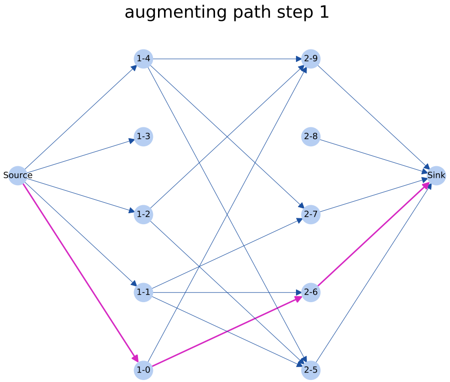
(b) Image generated with **neato**

Networkx

We will use networkx.



Networkx



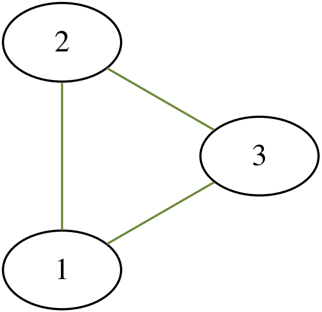
Warm up question

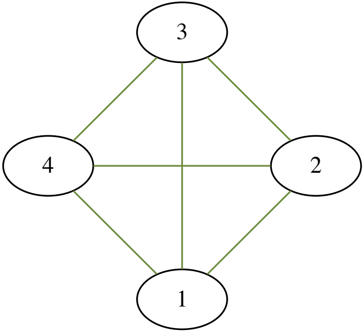
Given an **undirected** graph with n nodes, how many edges can we build ?

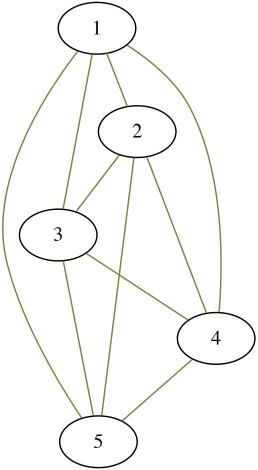
Notation of a graph : $G(V, E)$

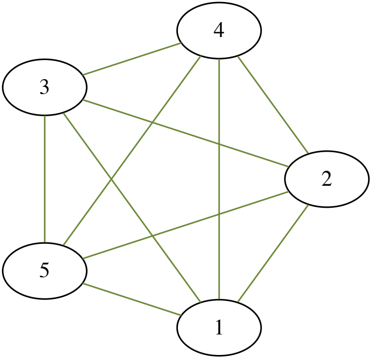
- V : set of n vertices
- E : set of edges

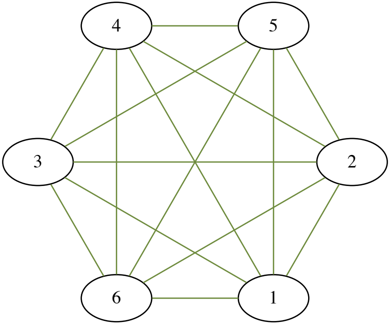

1
2


2
3
1


3
4
2
1


1
2
3
4
5


4
3
2
5
1


4
5
3
2
6
1

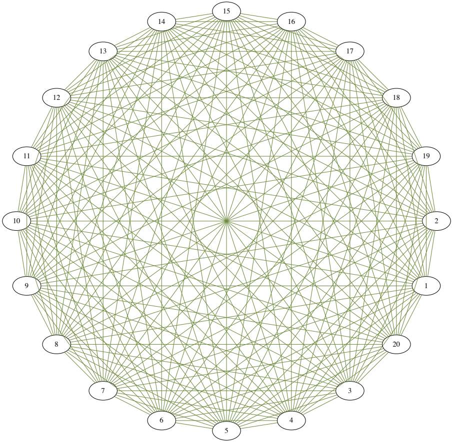


Figure: We cannot count anymore

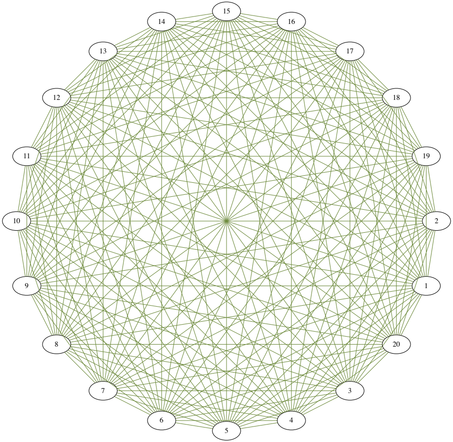
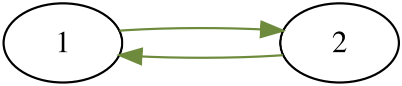
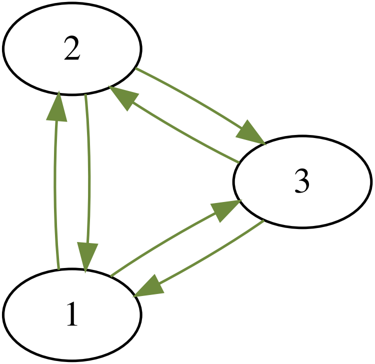


Figure: We cannot count anymore

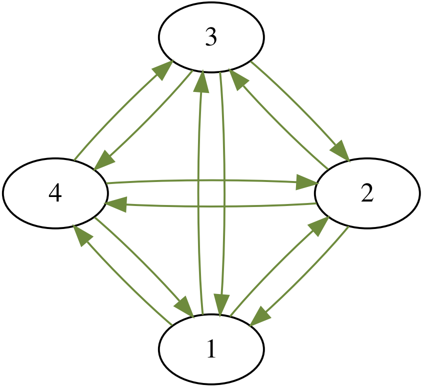
What if the graph is directed ?



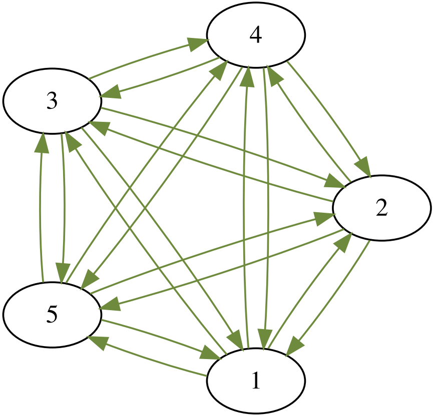
What if the graph is directed ?



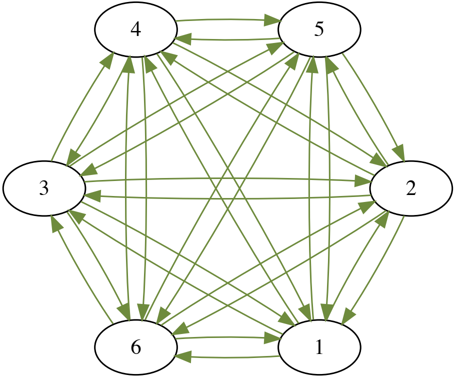
What if the graph is directed ?



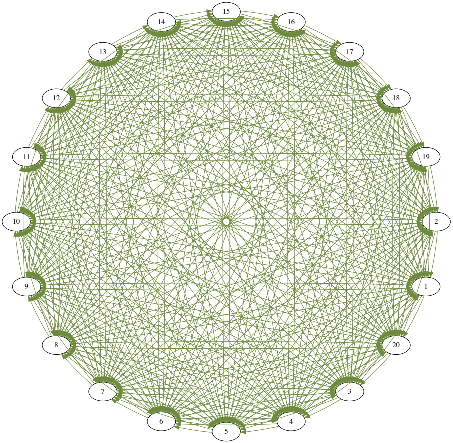
What if the graph is directed ?



What if the graph is directed ?



What if the graph is directed ?



Warm up question

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

Warm up question

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

$$n(n-1) \tag{1}$$

Warm up question

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

$$n(n-1) \tag{2}$$

So if the graph is **undirected**, we can build :

$$\frac{n(n-1)}{2} \tag{3}$$

edges.

Remark

$\frac{n(n-1)}{2}$ is also the number of subsets of size 2 in a set of size n.

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} \tag{4}$$

Famous graph problem

- Do you know some famous **graph problems ?**

Famous graph problem

- Do you know some famous **graph problems ?**
- Dominating set

Famous graph problem

- Do you know some famous **graph problems ?**
- Dominating set
- Maximum clique

Famous graph problem

- Do you know some famous **graph problems ?**
- Dominating set
- Maximum clique
- Coloring

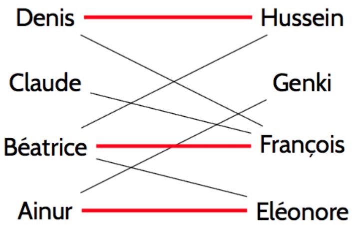
Matching problem

Let us now focus on the **matching problem** (problme du **couplage**)

Back to our problem

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

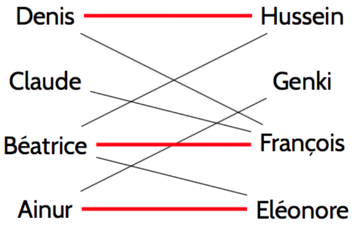
- A subset of edges $M \subset E$



Back to our problem

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

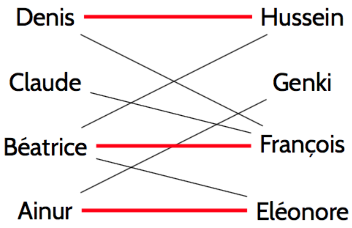
- A subset of edges $M \subset E$
- Such that no pairs of edges of M are incident
- Equivalently, each node in the graph has **at most** one edge connected



Back to our problem

Given **undirected** 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



Maximum matching

- The **size** of a matching is the number of edges it contains.

Maximum matching

- The **size** of a matching is the number of edges it contains.
- We want to find the matching of maximum size in a given graph.

Example 1

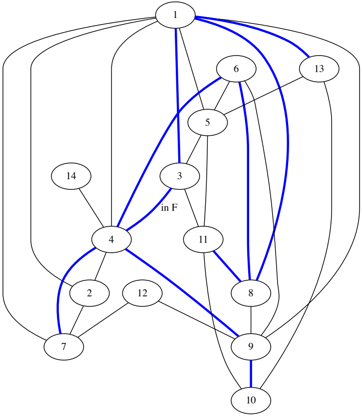


Figure: Is this a matching ?

Example 2

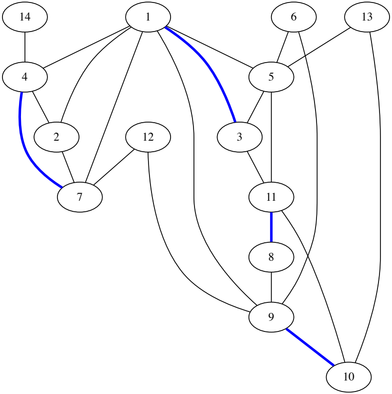


Figure: Is this a matching ?

Example 3

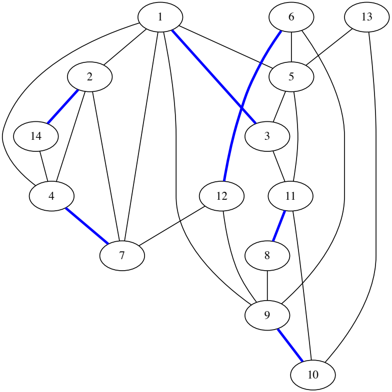


Figure: Is this an optimal matching ?

Example 4

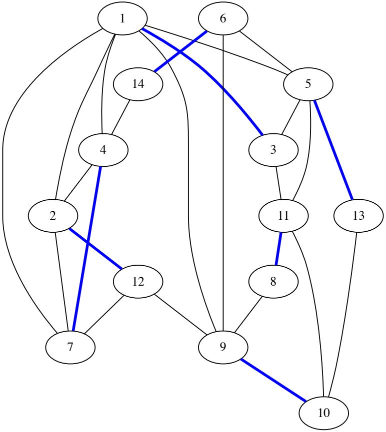


Figure: Is this an optimal matching ?

Example 5

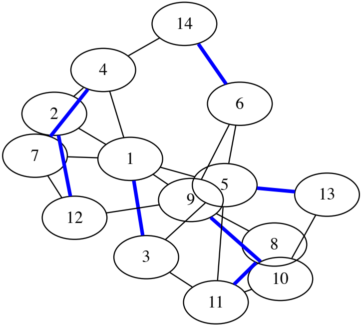


Figure: With neato

Optimal matching

Exercice 1 : Given a graph of size n, what is maximum size possible for a **matching ?**

Optimal matching

Exercice 1 : Given a graph of size n, what is maximum size possible for a **matching ?**

- If n is even : $\frac{n}{2}$
- Else n is odd : $\frac{n-1}{2}$

Optimal matching

Exercice 1 : Can you think of a graph that contains a matching of size n ? (assuming n is even)

Optimal

Exercice 1 : Can you think of a graph that contains a matching of size n ? (assuming n is even)

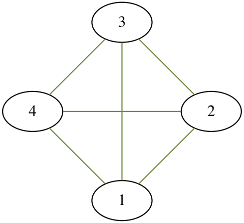


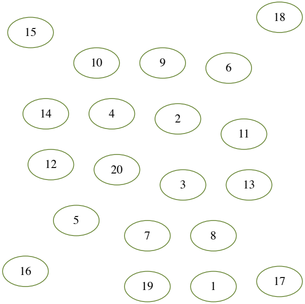
Figure: The complete graph works

Optimal matching

Exercice 1 : Can you think of a graph that does **not** contains a matching of size n ? (assuming n is even)

Optimal matching

Exercice 1 : Can you think of a graph that does **not** contains a matching of size n ? (assuming n is even)



Optimal matching

Exercice 1 : Can you think of a **non trivial** graph that does **not** contains a matching of size n ? (assuming n is even)

Optimal matching

Exercice 2 : Can you think of a **non trivial** graph that does **not** contains a matching of size n ? (assuming n is even)

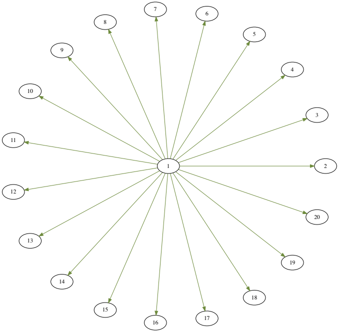


Figure: Star graph

Experiments

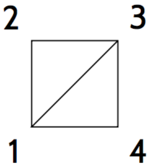
How would you code a graph ?

Experiments

How would you code a graph ?

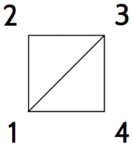
- list of sets of size 2 (for an undirected graph)
- a dictionary of successors (directed of undirected)

Coding a graph : as a list




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

Coding a graph : as a dictionary



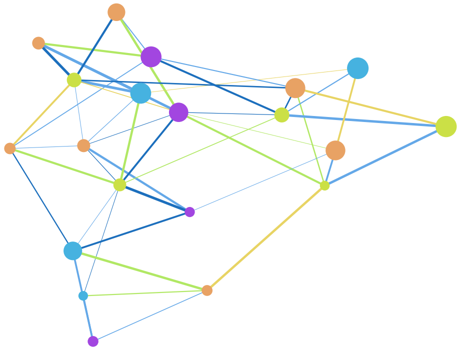

```
g1 = { 1:{2,3,4}, 2:{1,3}, 3:{1,2,4}, 4:{1,3} }
```

Random graph

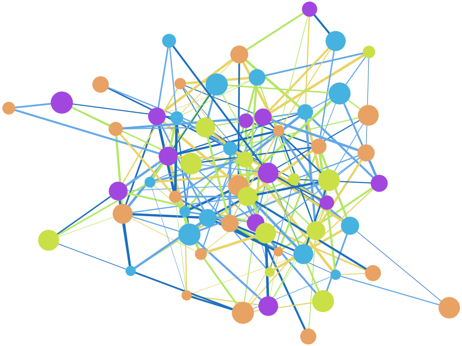
Exercice 3 : **cd other_graphs/** and please use **random_undirected_graph.py** to build a graph with 20 vertices and 50 edges.

- You will need to install **networx**

Example undirected graph obtained



Example undirected graph obtained

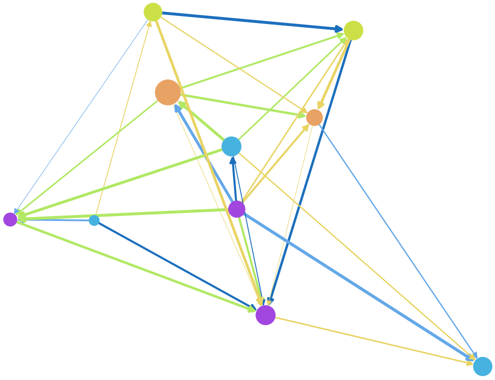


Example undirected graph obtained

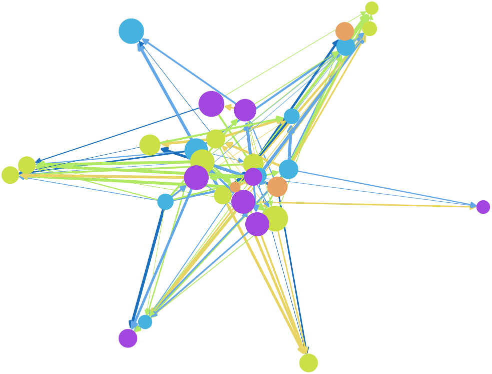
Exercice 4 : Random directed graph.

- Please use **random_directed_graph.py** to build a **directed** graph with a chosen number of vertices and **directed edges**.

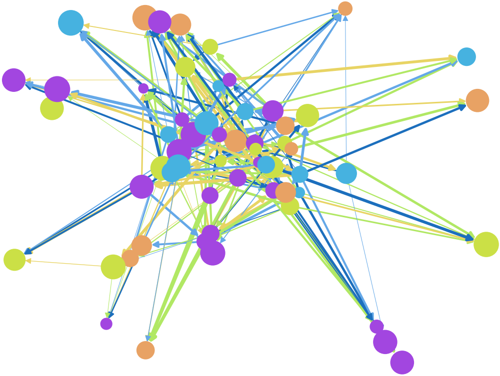
Example directed graph



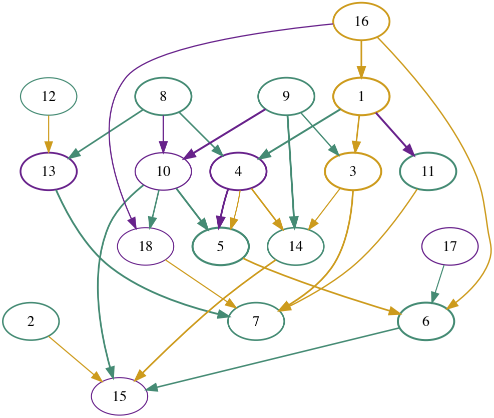
Example directed graph



Example directed graph



Example directed graph

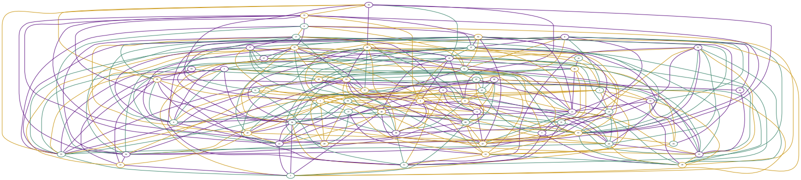


Manual matching

Exercice 5 : Please manually find an **optimal matching** in your **undirected** graph.

Big graph

We could not manually find an optimal matching in this graph :



Summary

- We have defined the matching problem.
- When the size of the problem is large, we can not manually find an optimal matching.

Brute force approach

Exercice 6 : Enumeration

- Given a graph, what would a brute force approach on the matching problem be ?

Brute force approach

Exercice 6 : Exhaustive search

- Given a graph what would a brute force approach on the matching problem be ?
  - 1) Enumerate all possible subsets in the set of the edges.
  - 2) Check if each subset is a matching.
  - 3) Return the biggest one obtained.

Brute force approach

Exercice 6 : Exhaustive search

- Given a graph what would a brute force approach on the matching problem be ?
  - 1) Enumerate all possible subsets in the set of the edges.
  - 2) Check if each subset is a matching.
  - 3) Return the biggest one obtained.

If the graph contains n nodes, and given a subset of edges, what if the number of computations needed to perform step 2 ?

Brute force approach

Exercice 6 : Exhaustive search

- Given a graph what would a brute force approach on the matching problem be ?
  - 1) Enumerate all possible subsets in the set of the edges.
  - 2) Check if each subset is a matching.
  - 3) Return the biggest one obtained.

If the graph contains n nodes, and given a subset of edges, what if the number of computations needed to perform step 2 ?
You can give a rough approximation.

Brute force approach

Exercice 6 : Exhaustive search

- Given a graph what would a brute force approach on the matching problem be ?
  - 1) Enumerate all possible subsets in the set of the edges.
  - 2) Check if each subset is a matching.
  - 3) Return the biggest one obtained.

If the graph contains n nodes, and given a subset of edges, what if the number of computations needed to perform step 2 ?

It is a **polynomial** number of computations : so it is ok.

Notion of complexity

- The **time complexity** of an algorithm is a measure of the **number of elementary** operations needed for the algorithm to termiate with respect to the input size.

Brute force search

Exercice 7 : Complexity of brute force

- 1) Enumerate all possible subsets in the set of the edges.
- 2) Check if each subset is a matching.
- 3) Return the biggest one obtained.

What is the complexity of step 1 ?

Brute force search

Exercice 7 : Complexity of brute force

- 1) Enumerate all possible subsets in the set of the edges.
- 2) Check if each subset is a matching.
- 3) Return the biggest one obtained.

What is the complexity of step 1 ?

The number of subsets is $2^{\frac{n(n-1)}{2}}$ (in the worst case), which is exponential.

Brute force search

Exercice 7 : Complexity of brute force Assume that checking a subset requires 1 microsecond. How long should we wait in order to check all possible matching in a graph with 100 nodes ?

Other example of complexities

- linear search
- dichotomic search

Summary II

- For the matching problem on a large graph, we can neither
  - manually find an optimal matching
  - perform the exhaustive search (brute force algorithm)

Algorithms

- Hence, we need different algorithms to solve the problem.

Algorithms

- Hence, we need different algorithms to solve the problem.
- Let us first introduce some theoretical notions.

Notion of maximal and maximum matching

We will say that a matching M of cardinality (number of elements) $|M|$ is:

- **Maximum** if is has 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 it has 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 not in M.
- $\cup$ means union of sets.

is being a **maximal** matching the same thing as beeing a **maximum** matching ?

Maximum implies maximal

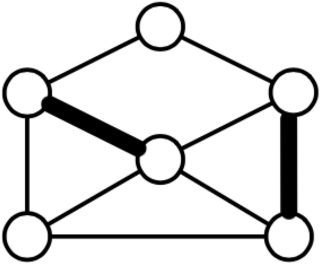
Let us show that a maximum matching is maximal.

Counter Example

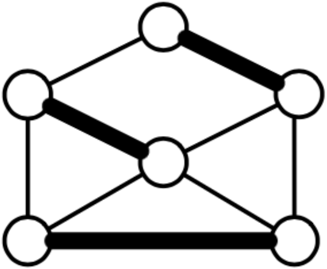
However, a matching that is maximal is **not necessary Maximum**.

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 ? (as a function of the number of nodes n of the graph)

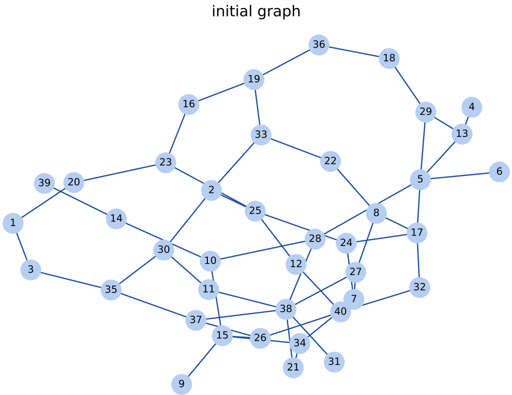
Greedy algorithm

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

Greedy algorithm

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

Exercice 8 : **cd matching_greedy/** and use **generate_graph.py** to build a graph with a least 30 nodes. The images are stored in **images/**, data stored in **data/**

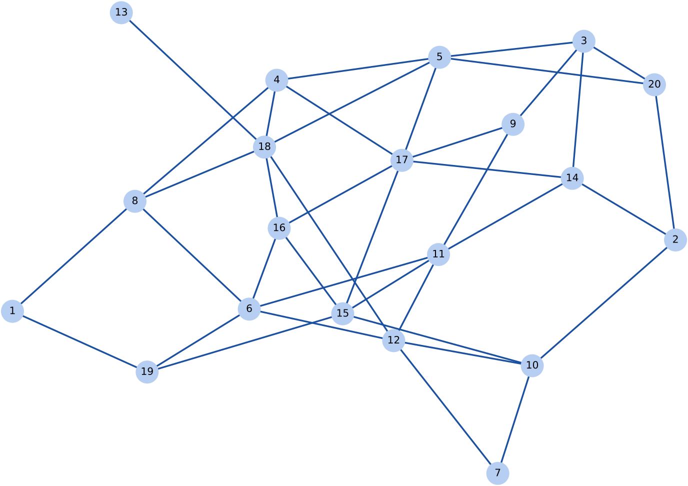


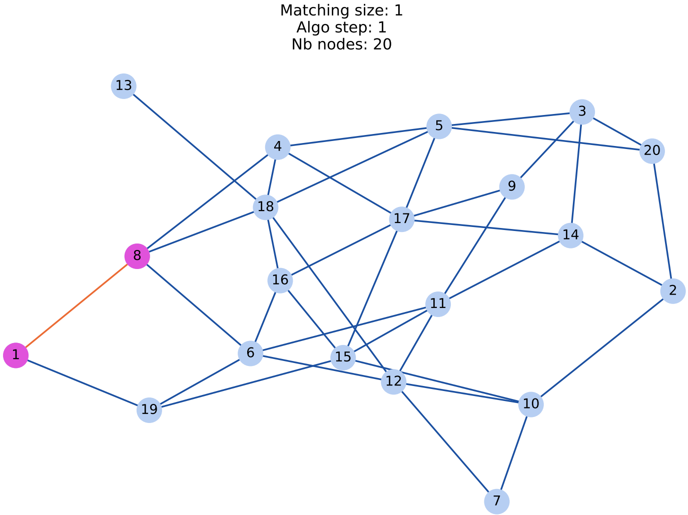
Implementing the greedy algorithm

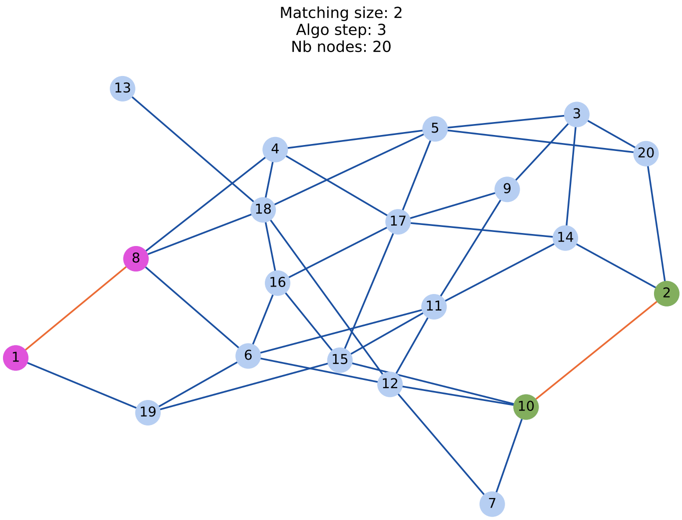
Exercice 8 : Implement the greedy algorithm on this graph.

- Use the functions in **matching_functions.py** and call them from **apply_matching_algorithm.py**
- More details in the file.

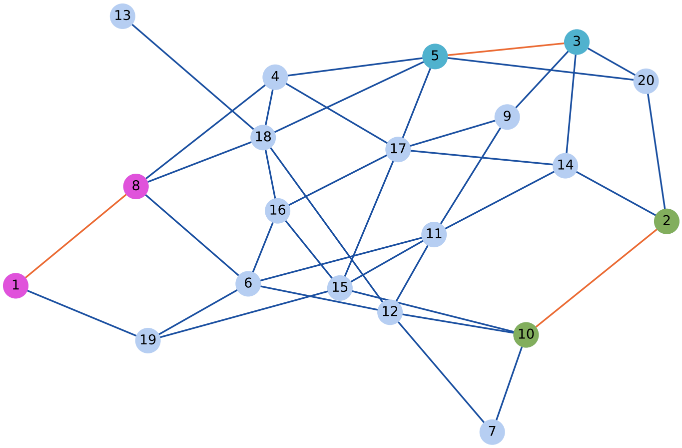
initial graph

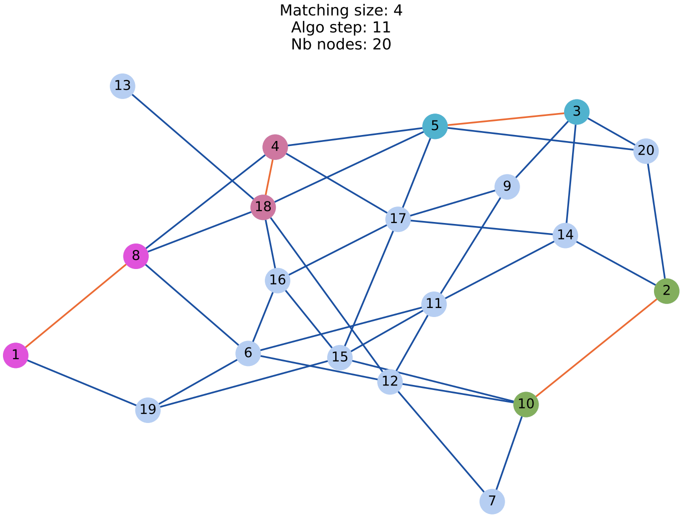



Matching size: 1
Algo step: 1
Nb nodes: 20
13
5
3
4
20
9
18
17
14
8
16
2
11
1
6
15
12
10
19
7

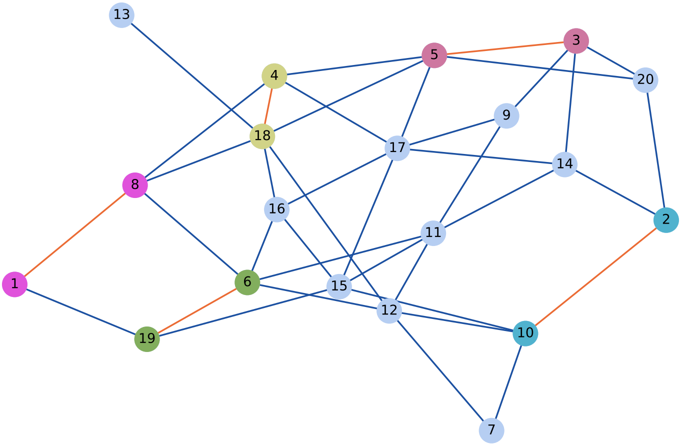

Matching size: 2
Algo step: 3
Nb nodes: 20
13
3
5
4
20
9
18
17
14
8
16
2
11
1
6
15
12
10
19
7

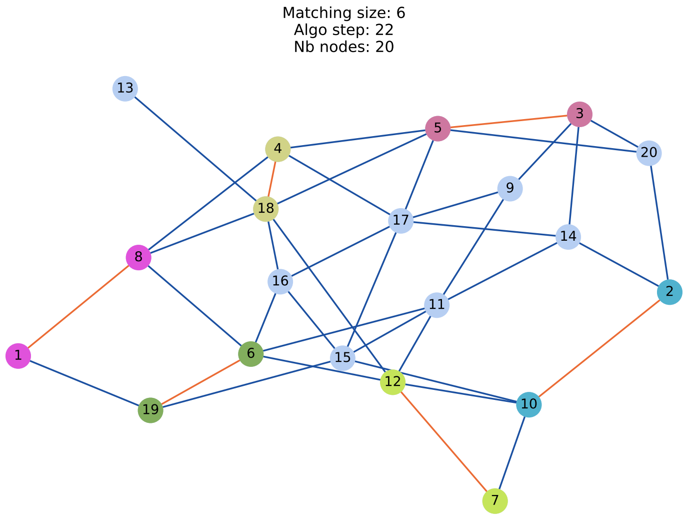
Matching size: 3
Algo step: 6
Nb nodes: 20

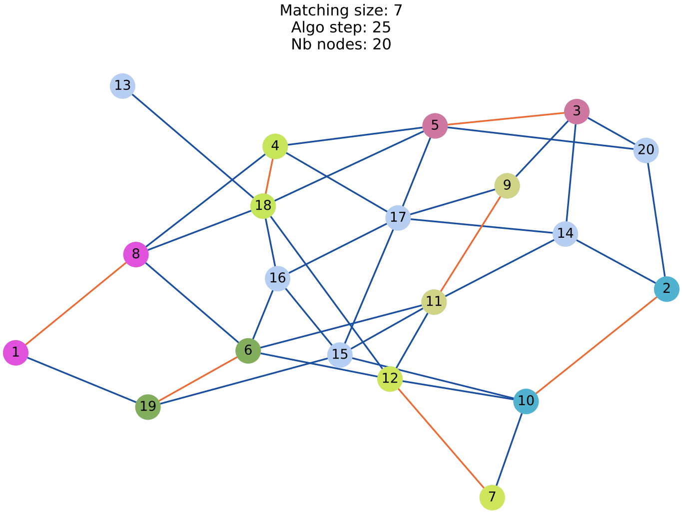



Matching size: 4
Algo step: 11
Nb nodes: 20
13
3
5
4
20
9
18
17
14
8
16
2
11
1
6
15
12
10
19
7

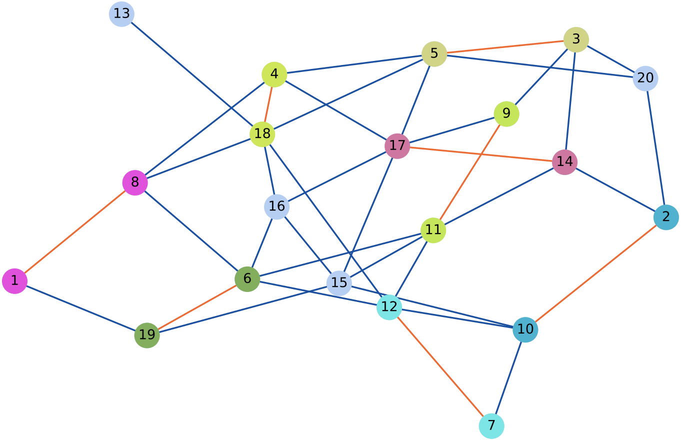
Matching size: 5
Algo step: 17
Nb nodes: 20

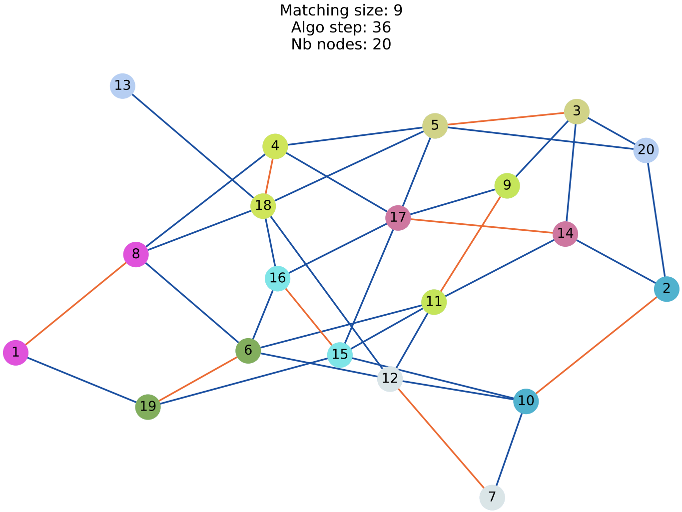



Matching size: 6
Algo step: 22
Nb nodes: 20
13
3
5
4
20
9
18
17
14
8
16
2
11
1
6
15
12
10
19
7

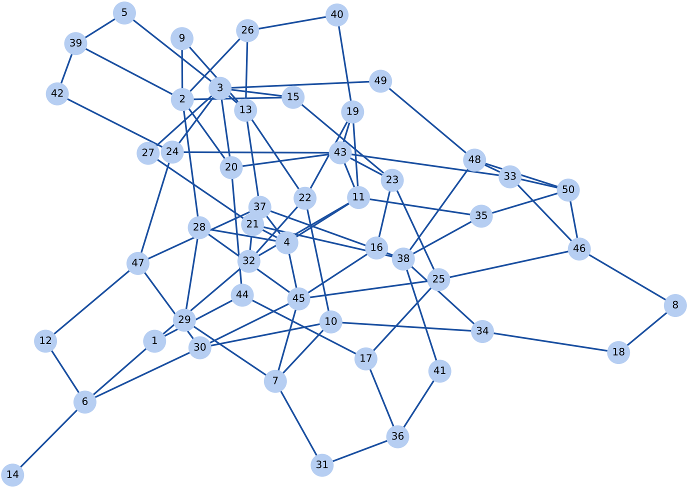

Matching size: 7
Algo step: 25
Nb nodes: 20
13
5
3
4
20
9
18
17
14
8
16
2
11
1
6
15
12
19
10
7

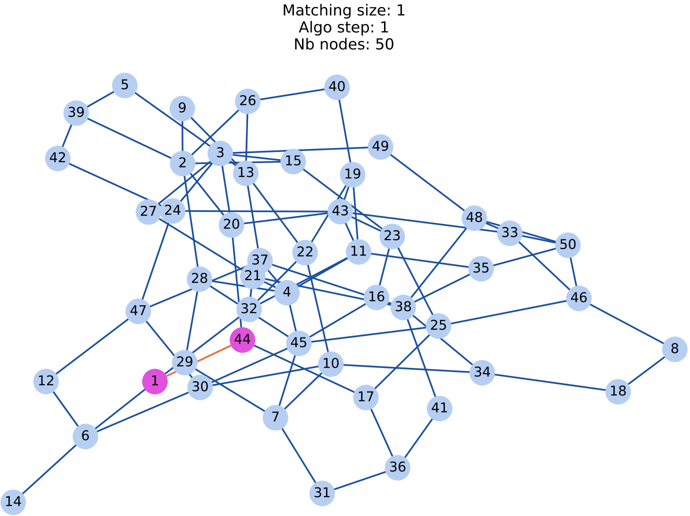
Matching size: 8
Algo step: 34
Nb nodes: 20

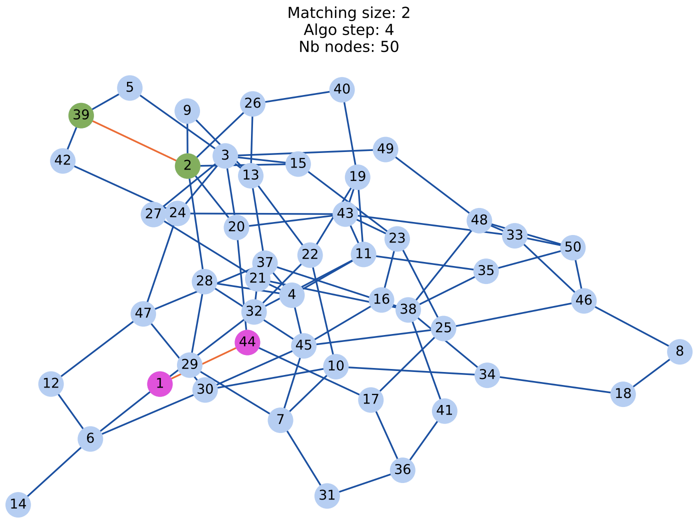


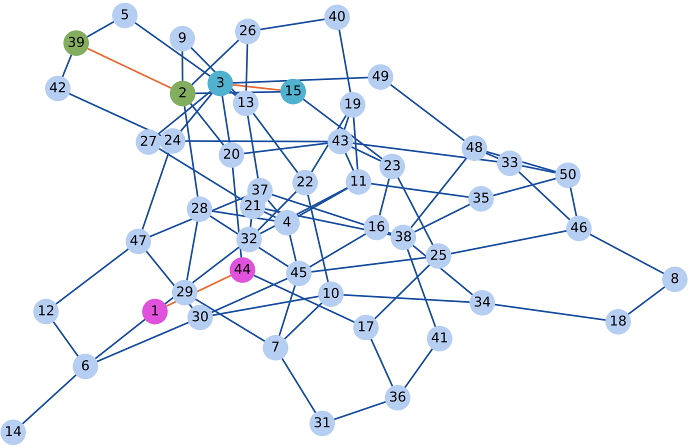

Matching size: 9
Algo step: 36
Nb nodes: 20
13
5
3
4
20
9
18
17
14
8
16
2
11
1
6
15
12
19
10
7

initial graph

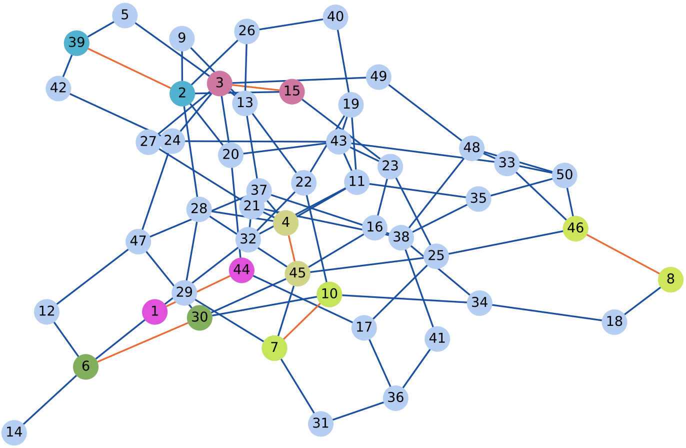


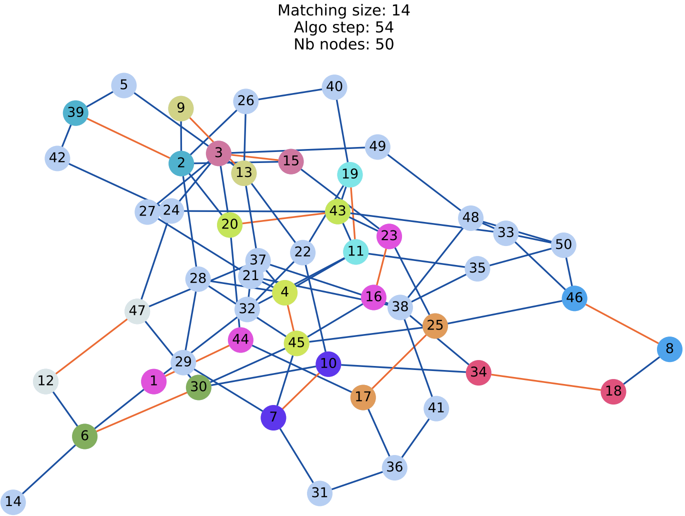

Matching size: 1
Algo step: 1
Nb nodes: 50
5
39
42
9
26
40
49
3
2
13
15
19
27
24
43
20
48
33
50
23
11
22
37
35
21
28
4
16
38
46
47
32
25
44
45
8
29
10
34
12
1
30
17
18
41
7
6
36
31
14


Matching size: 2
Algo step: 4
Nb nodes: 50

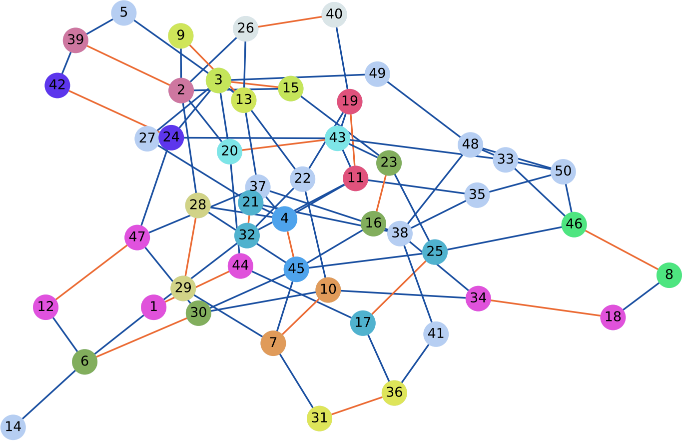

Matching size: 3
Algo step: 10
Nb nodes: 50

Matching size: 7
Algo step: 30
Nb nodes: 50

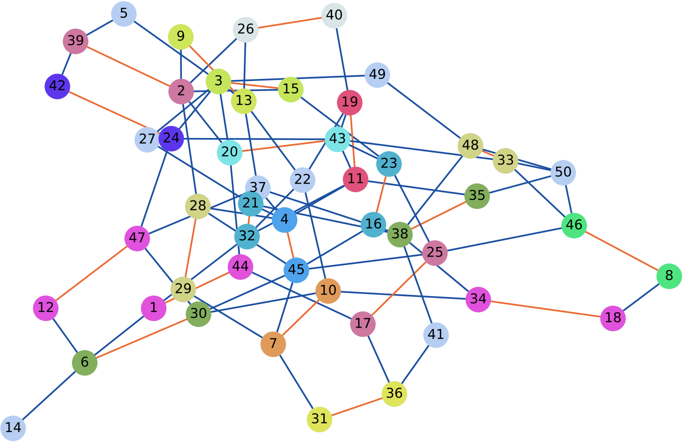



Matching size: 14
Algo step: 54
Nb nodes: 50
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50

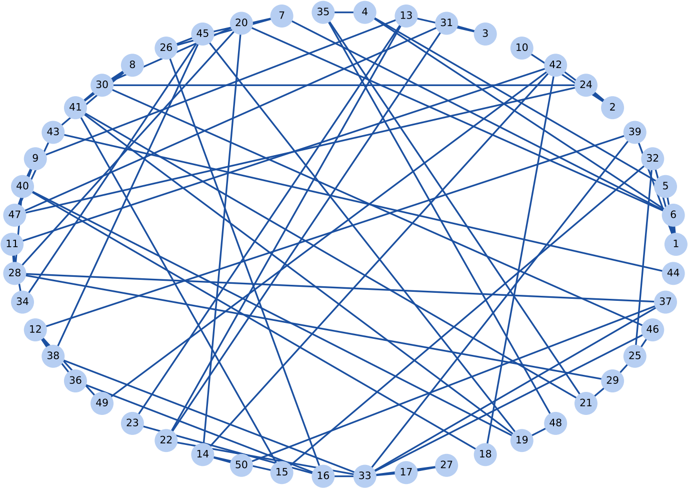
Matching size: 19
Algo step: 72
Nb nodes: 50

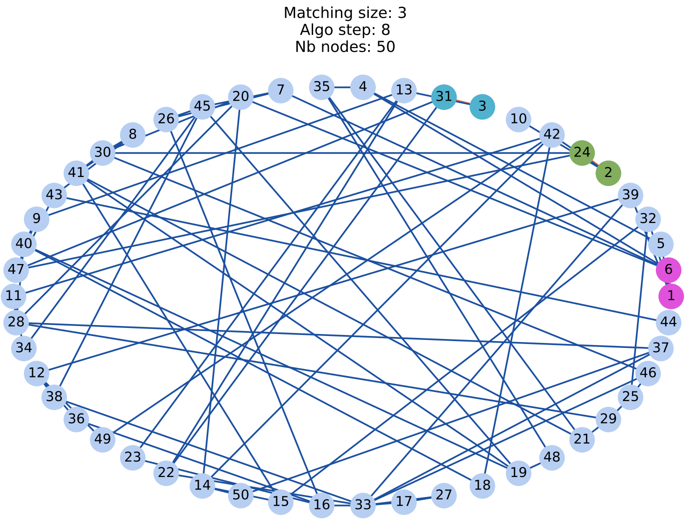


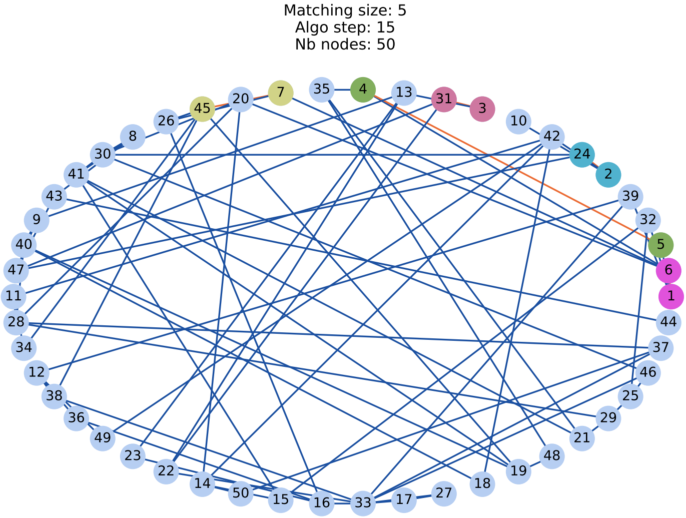
Matching size: 21
Algo step: 78
Nb nodes: 50



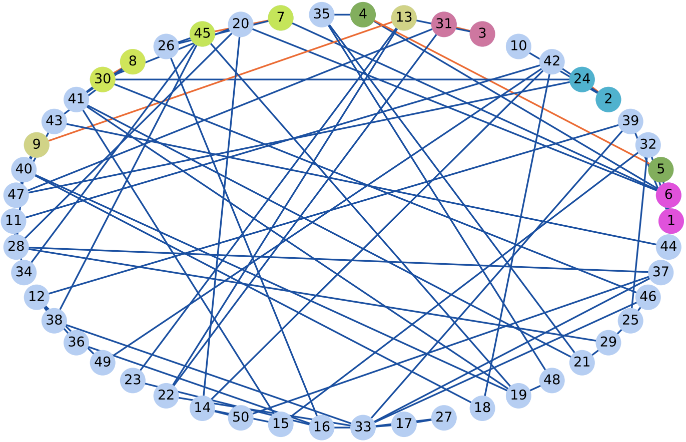
initial graph

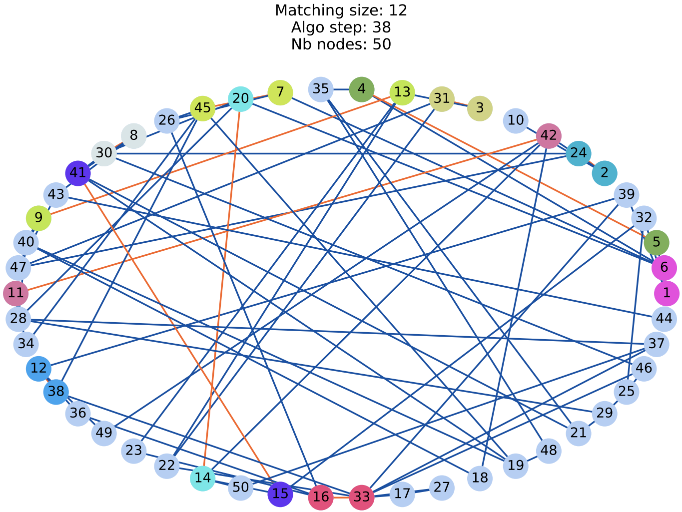



Matching size: 3
Algo step: 8
Nb nodes: 50
35
4
13
31
3
7
20
45
26
8
30
41
43
9
40
47
11
28
34
12
38
36
49
23
22
14
50
15
16
33
17
27
18
19
48
21
29
25
46
37
44
1
6
5
32
39
2
24
42
10

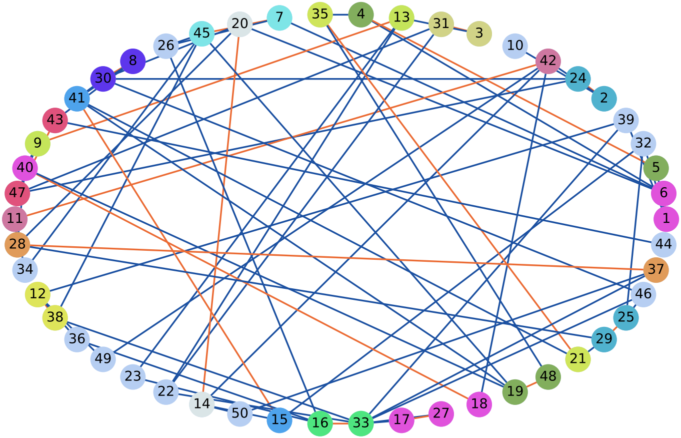

Matching size: 5
Algo step: 15
Nb nodes: 50
35
4
13
31
3
7
20
45
26
8
30
41
43
9
40
47
11
28
34
12
38
36
49
23
22
14
50
15
16
33
17
27
18
19
48
21
29
25
46
37
44
1
6
5
32
39
2
24
42
10

Matching size: 7
Algo step: 20
Nb nodes: 50




Matching size: 12
Algo step: 38
Nb nodes: 50
35
4
13
31
3
7
20
45
26
8
30
41
43
9
40
47
11
28
34
12
38
36
49
23
22
14
50
15
16
33
17
27
18
19
48
21
29
25
46
37
44
1
6
5
32
39
2
24
42
10

Matching size: 19
Algo step: 79
Nb nodes: 50



Example

Exercice 9 : Can you think of an example where the greedy algorithm gives a **bad** matching, e.g. of the size **half** the size of an optimal matching ?

Example

Exercice 10 : Can you think of an example where the greedy algorithm gives a **bad** matching, e.g. of the size **half** the size of an optimal matching ?

Greedy matching

However, is $|M|$ is the cardinality of a matching returned by the greedy algorithm, and if $|M^*|$ is the cardinal of the real optimal matching, we have :

$$|M| \geq \frac{|M^*|}{2} \tag{5}$$

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

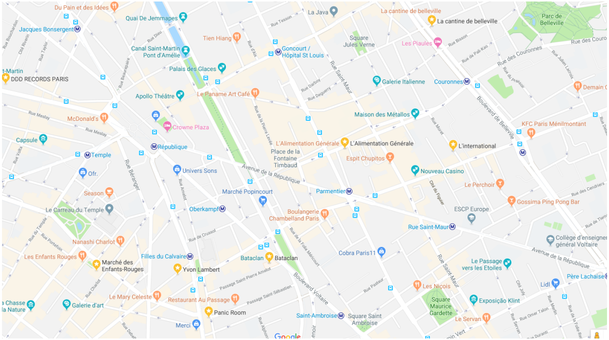


Figure: Optimizing the quantity of something transported from one place to another, under constraints

Formalizing the problem

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

Formalizing the problem

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

Formalizing the problem

We introduce the concept of **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

We introduce the concept of **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

We introduce the concept of **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.

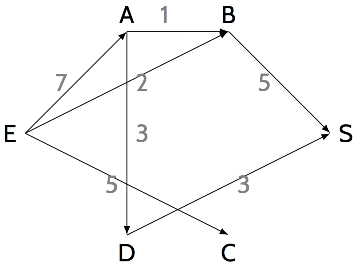


Figure: A **flow network (reseau de flot)** with capacities

Formalizing the problem

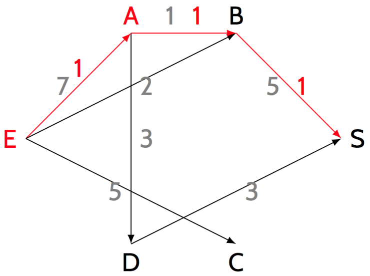
We introduce the concept of **flow network (reseau de flot).**

- 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)

Formalizing the problem

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

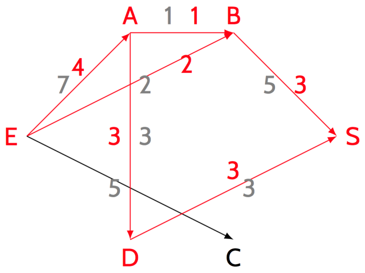
- Each edge (u, v) must have a **capacity** $c(u, v) \geq 0$
- A **flow** f is a function $f(u, v) \leq c(u, v)$ (+ additional constraints)



Formalizing the problem

We introduce the concept of **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)



Conservation of the flow

We must have :

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

conservation of the flow

we must have :

- antisymmetry : $f(v,u) = -f(u,v)$
- flow conservation : $\sum_{w \in V} f(u,w) = 0$ for $u \notin \{e,s\}$

Other formulation of the flow conservation

Let us show that for a flow f:

$$\sum_{f(u,v)>0} f(u,v) = \sum_{f(v,u)>0} f(v,u) \tag{6}$$

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**

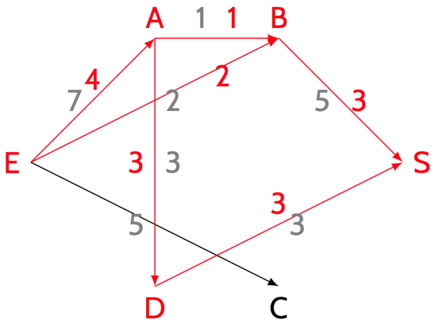


Figure: Max flow

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.

Ford Fulkerson algorithm

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

- terminates
- is correct
- is polynomial

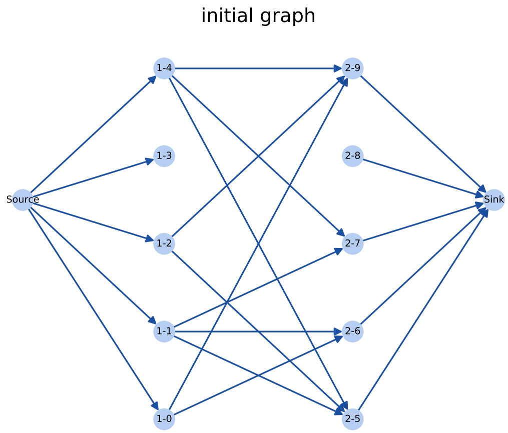
So it a good algorithm. **This section is going to be a little bit technical.**

Residual graph

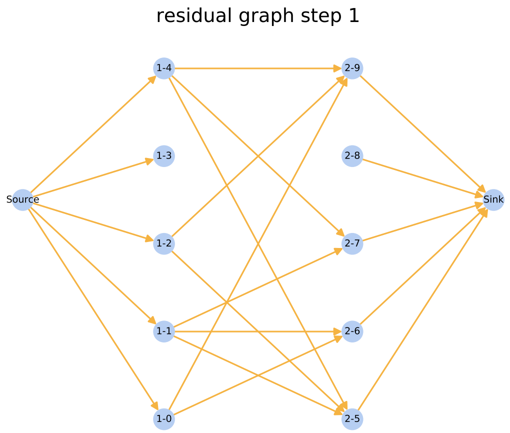
- Given a graph with capacities $c(u, v)$ and a flow $f(u, v)$, we will define its **residual graph** that has a capacity $c_r(u, v)$:

$$c_r(u, v) = c(u, v) - f(u, v) \tag{7}$$

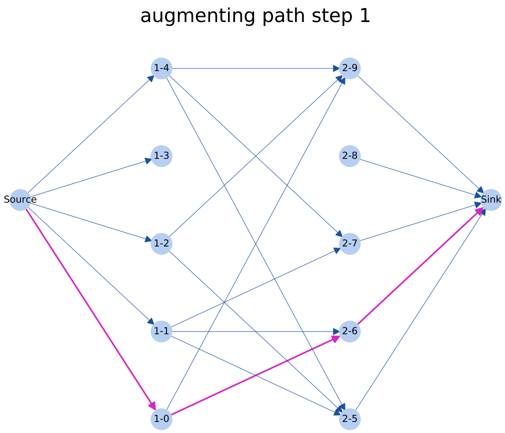
Example of residual graph



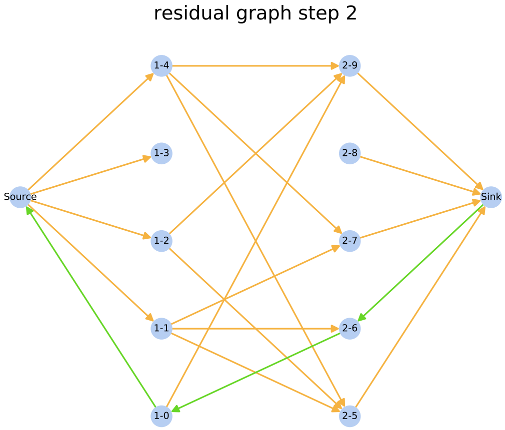
Example of residual graph



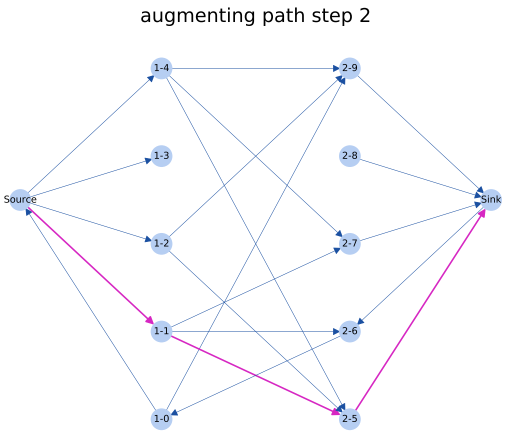
Example of residual graph



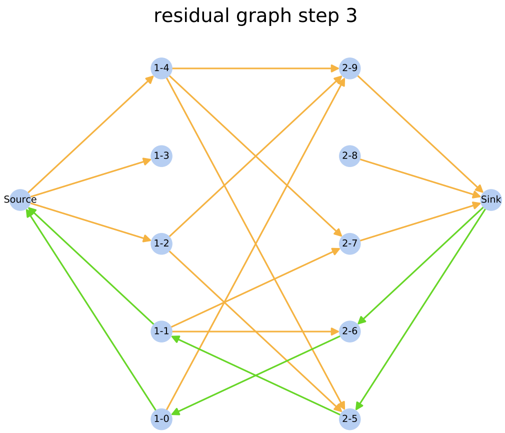
Example of residual graph



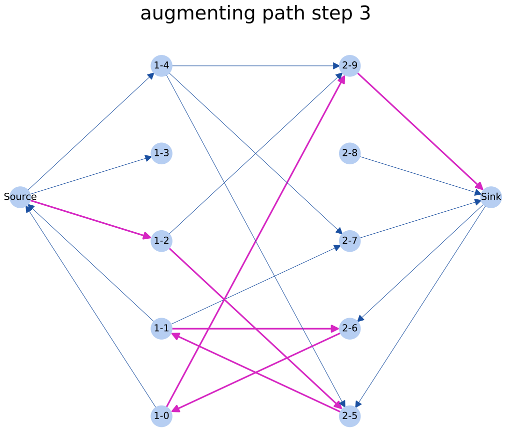
Example of residual graph



Example of residual graph



Example of residual graph



Residual graph

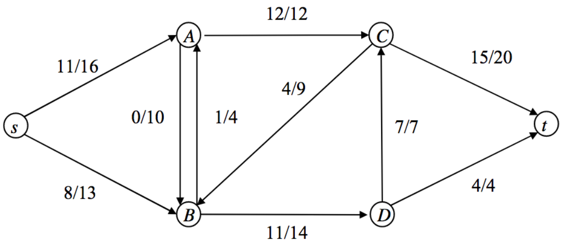


Figure: Another flow network

Residual graph

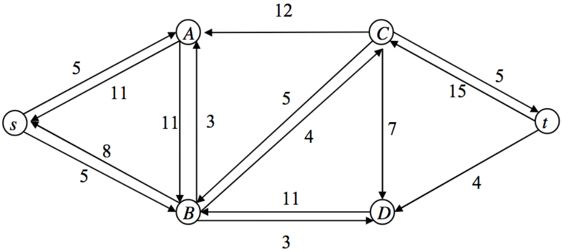


Figure: Residual graph

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.

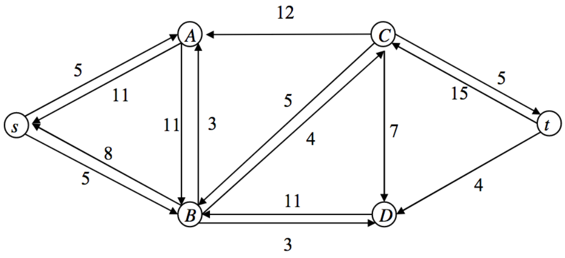


Figure: Residual graph

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.

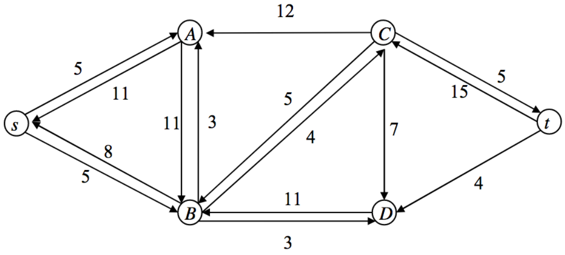


Figure: Residual graph

Ford Fulkerson algorithm

Can you deduce the algorithm from the previous remarks ?

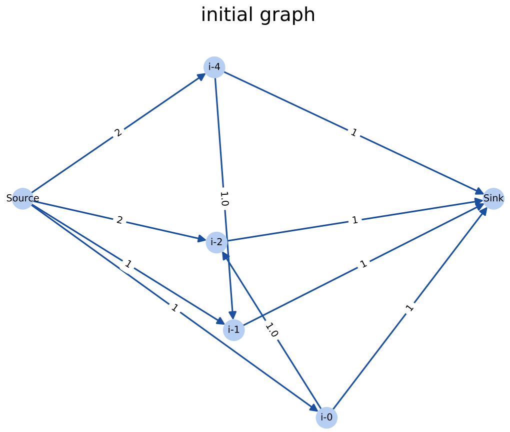
Ford Fulkerson algorithm

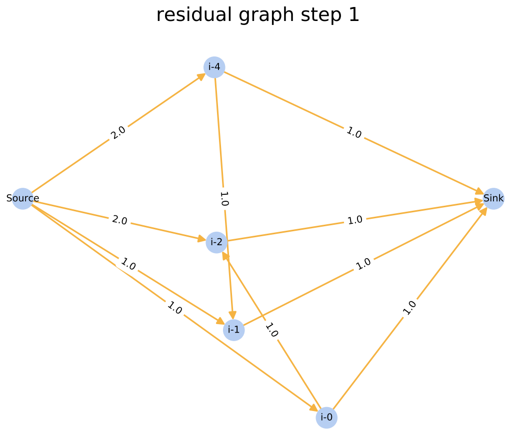
Result: Flow f
for $(u, v) \in E$ **do**
  $f(u, v) = 0$
end
while $\exists \rho$ *augmenting path* **do**
  augment f with ρ
end
return f

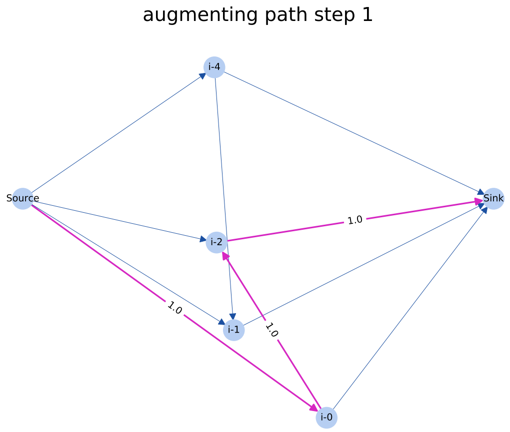
Algorithm 1: Ford Fulkerson algorithm

Ford-fulkerson algorithm

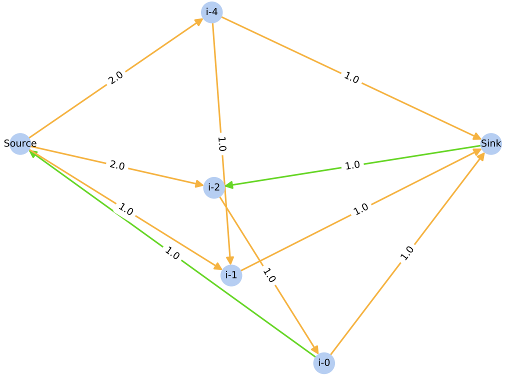
Let us some complete instances of the algorithm:

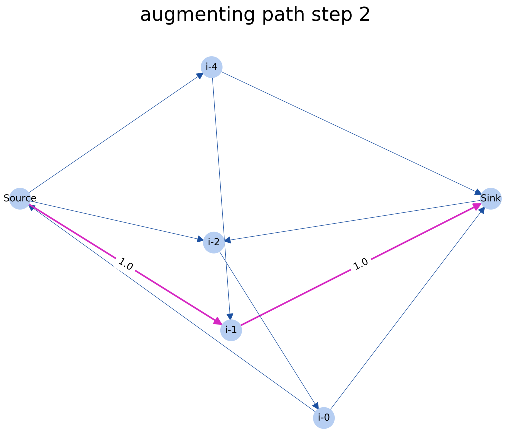

initial graph
i-4
Source
Sink
i-2
i-1
i-0
2
1
1.0
2
1
1
1
1
1
1.0

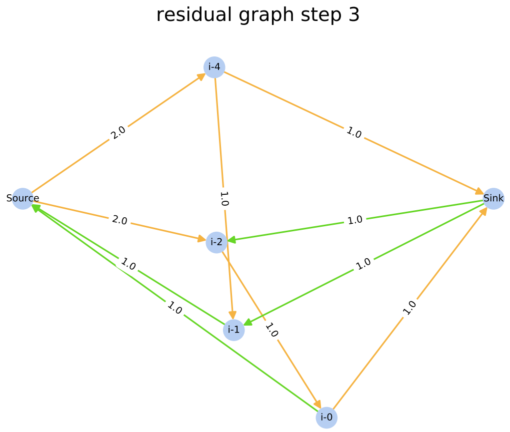

residual graph step 1
i-4
Source
Sink
i-2
i-1
i-0
2.0
1.0
1.0
2.0
1.0
1.0
1.0
1.0
1.0
1.0

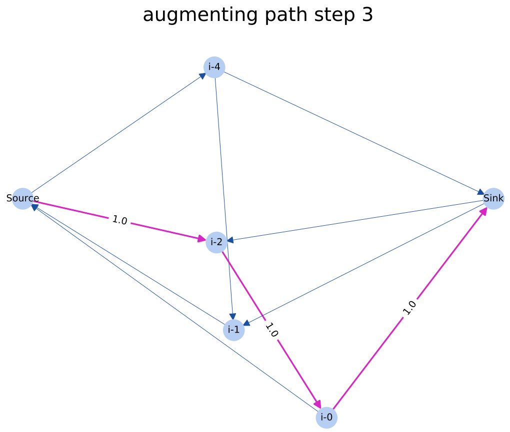

augmenting path step 1
i-4
Source
Sink
i-2
1.0
1.0
1.0
i-1
i-0

residual graph step 2

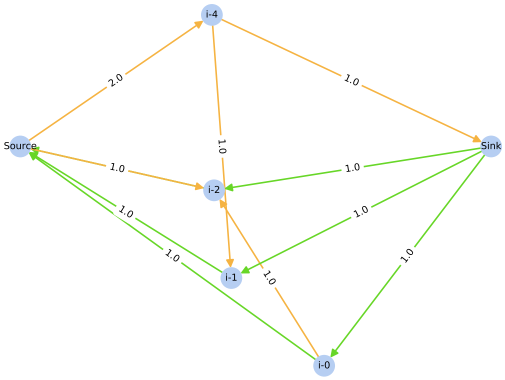


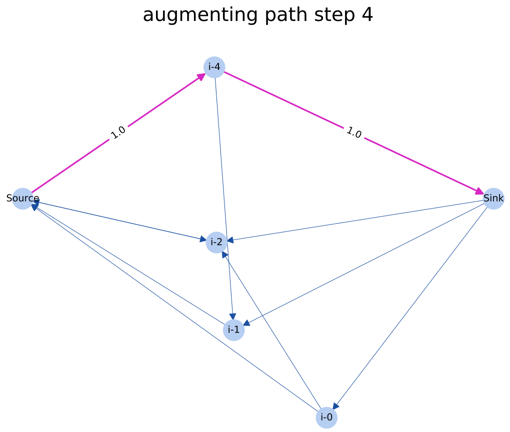

augmenting path step 2
i-4
Source
Sink
i-2
1.0
1.0
i-1
i-0

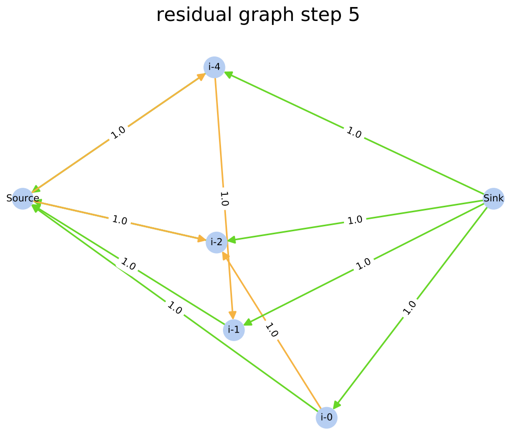

residual graph step 3
i-4
2.0
1.0
1.0
Source
Sink
2.0
1.0
i-2
1.0
1.0
1.0
1.0
i-1
1.0
i-0


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

residual graph step 4




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


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

initial graph

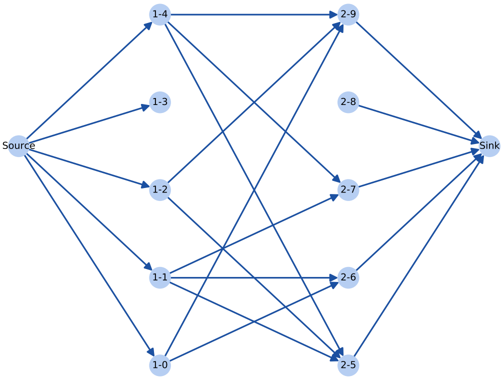
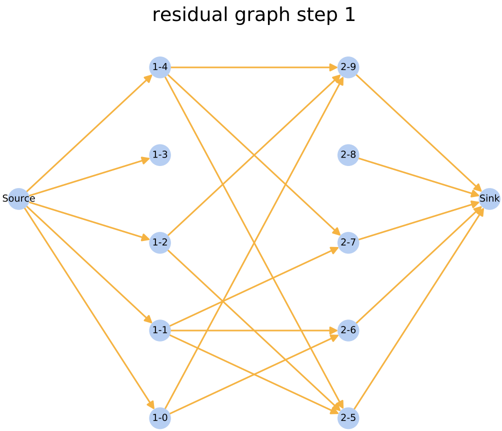
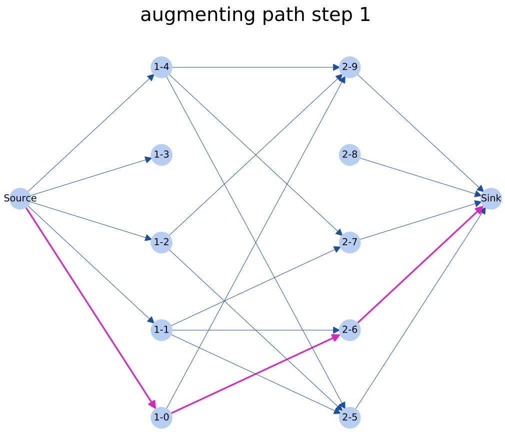
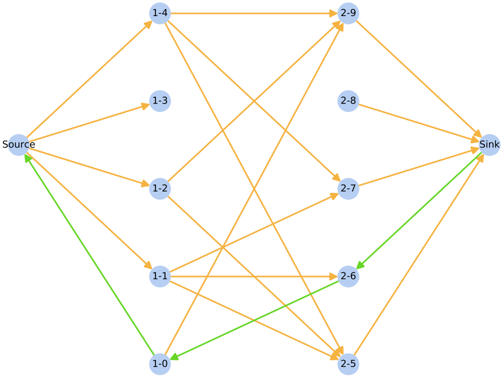


Figure: simple/initial

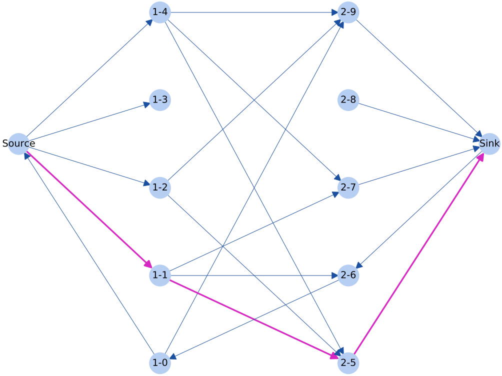

residual graph step 1
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

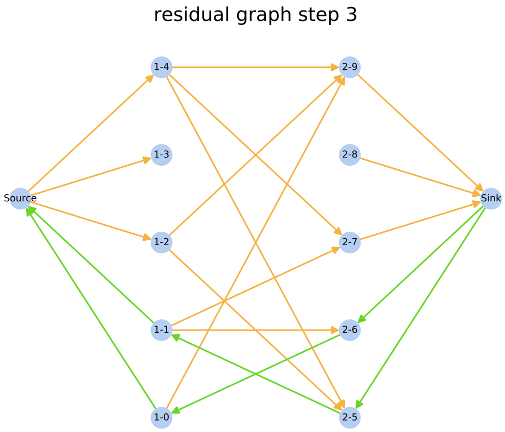

augmenting path step 1
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

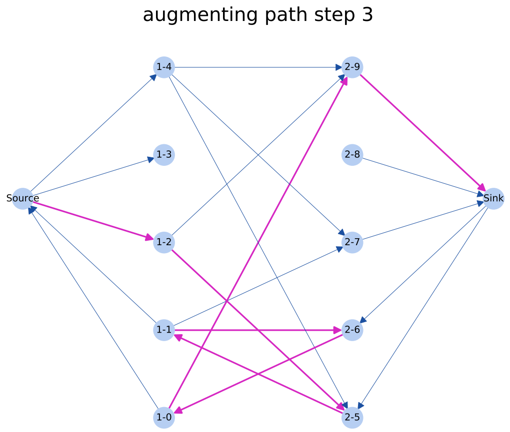
residual graph step 2



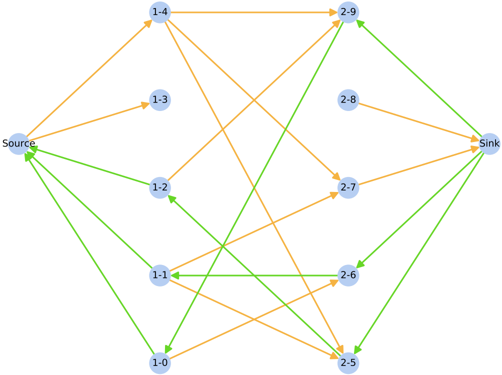
augmenting path step 2

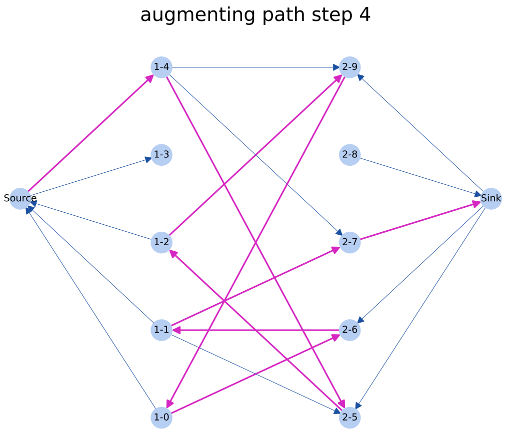



residual graph step 3
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

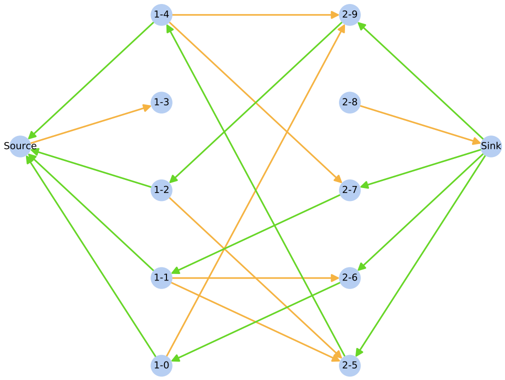

augmenting path step 3
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

residual graph step 4




augmenting path step 4
1-4
2-9
1-3
2-8
Source
Sink
1-2
2-7
1-1
2-6
1-0
2-5

residual graph step 5



Ford Fulkerson algorithm

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

Ford Fulkerson algorithm

Exercice 10 : We will implement the Ford Fulkerson algorithm (1956)

- **cd ford_fulkerson/** and edit **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 **apply_ford_fulkerson.py**

Algorithm

Exercice 11 : step 1

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

Algorithm

Exercice 11 : step 2

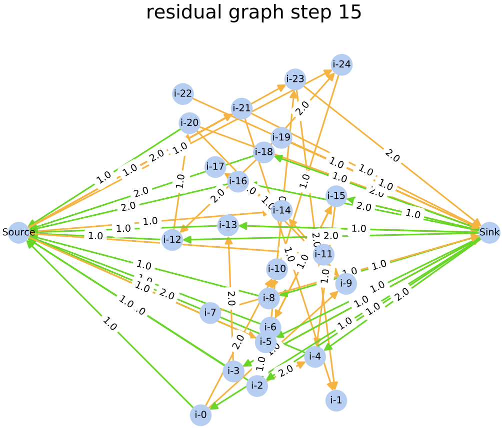
- now edit **augment_flow()**

Algorithm

Exercice 11 : step 3

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

- Now the algorithm should be able to run


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

Complexity

What is the complexity of Ford Fulkerson ?

Complexity

What is the complexity of Ford Fulkerson ?

$$\mathcal{O}(|f^*| \times |E|) \tag{8}$$

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 positive capacity.

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**, 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** ("problme d'affectation")
- We will show that in that case, we can connect the two problems.

Bipartite graph

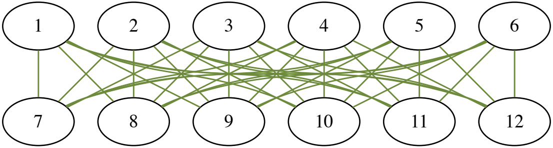


Figure: Complete bipartite graph (not all bipartite graphs are complete)

Matching problem

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

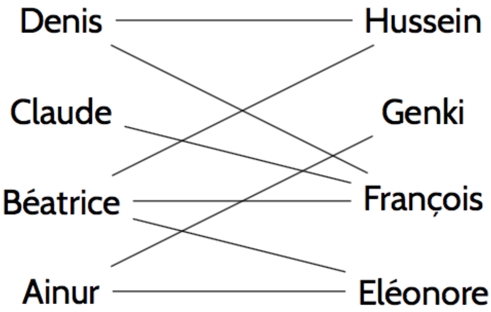


Figure: Bipartite graph

Equivalence between matching and flow

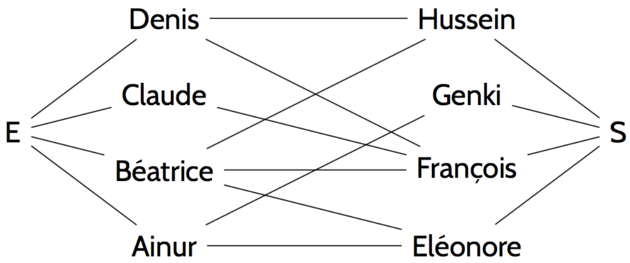


Figure: Introduce two more nodes. All edges have capacity 1. We consider **flows with integer values**

Ford Fulkerson for matching

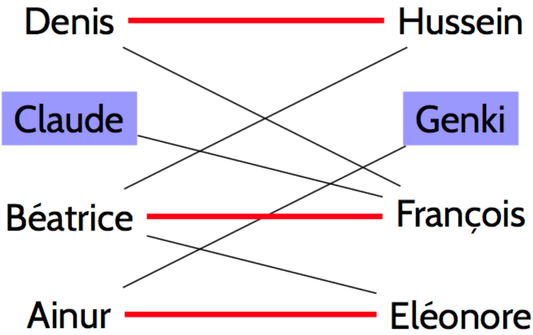


Figure: Non optimal solution

Ford Fulkerson for matching

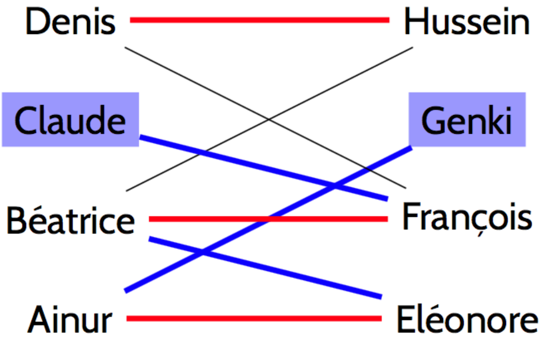


Figure: Optimal solution

Connection

Exercice 11 : Find a connection between the two problems

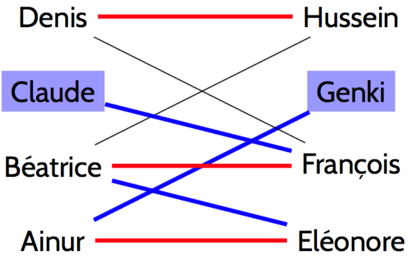


Figure: Optimal solution

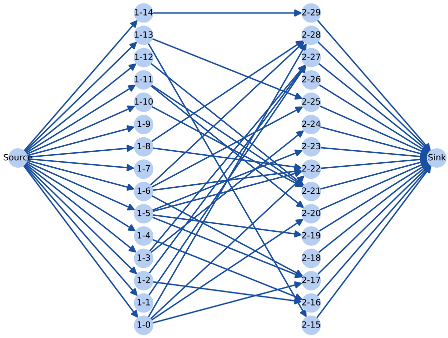
Exercice 12 : Ford Fulkerson and matching

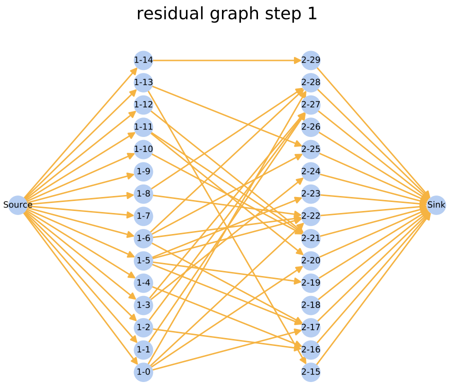
- We will transpose Ford Fulkerson to a bipartite graph in order to find an optimal matching.
- **cd ford_matching/**
- edit **generate_matching_problem.py** in order to generate an instance of the problem.

Exercice 12 : Ford Fulkerson and matching

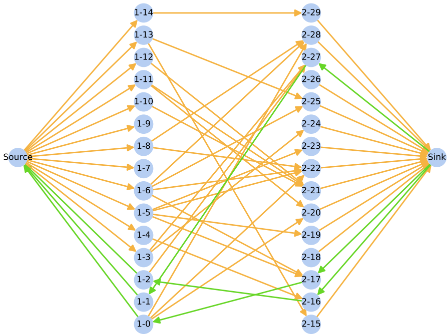
- Apply the algorithm on an example generated by the previous function.
- Apply the algorithm to as an instance of your choice.

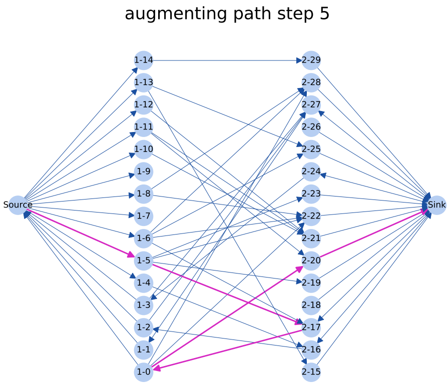
initial graph

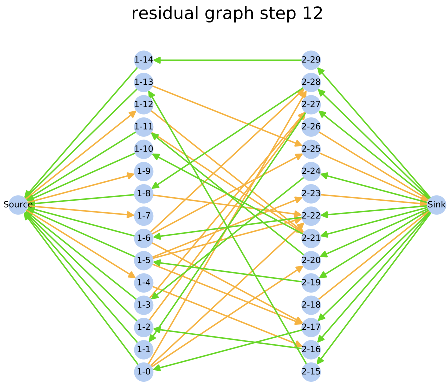



residual graph step 1
Source
1-14
1-13
1-12
1-11
1-10
1-9
1-8
1-7
1-6
1-5
1-4
1-3
1-2
1-1
1-0
2-29
2-28
2-27
2-26
2-25
2-24
2-23
2-22
2-21
2-20
2-19
2-18
2-17
2-16
2-15
Sink

residual graph step 4




augmenting path step 5
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 12
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

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 :

$$\forall X \subset U, |N_G(X)| \geq |X| \tag{9}$$

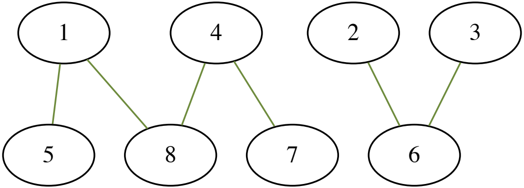
where $N_G(X)$ is the set of neighbors of X in G.

Hall's theorem

Exercice 13 : 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 13 : 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.
Other methods exist such as the **Blossom algorithm.**
(Edmonds-Karp)

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