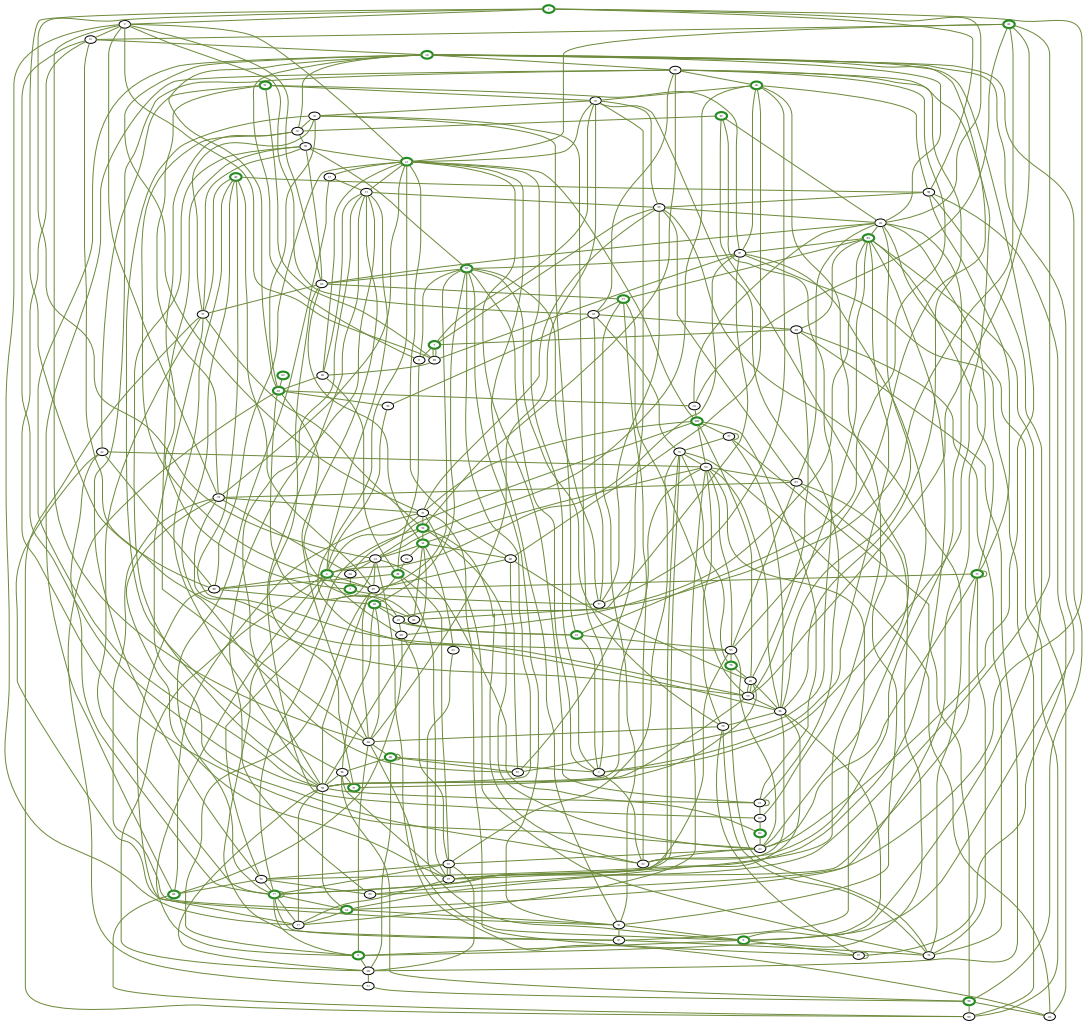
Algorithmic complexity and graphs: graph problems

14 septembre 2024

Graph problems



Graph problems

We will look at famous graph problems, typically of the form :

- "what is the largest subset of nodes of the graph, verifying some property ?"
- "what is the largest subset of edges of the graph, such that some property is verified ?"

Like some problems we discussed yesterday, these belong to **exploration problems**.

networx

We will use **networx** to visualize graphs and work with them.

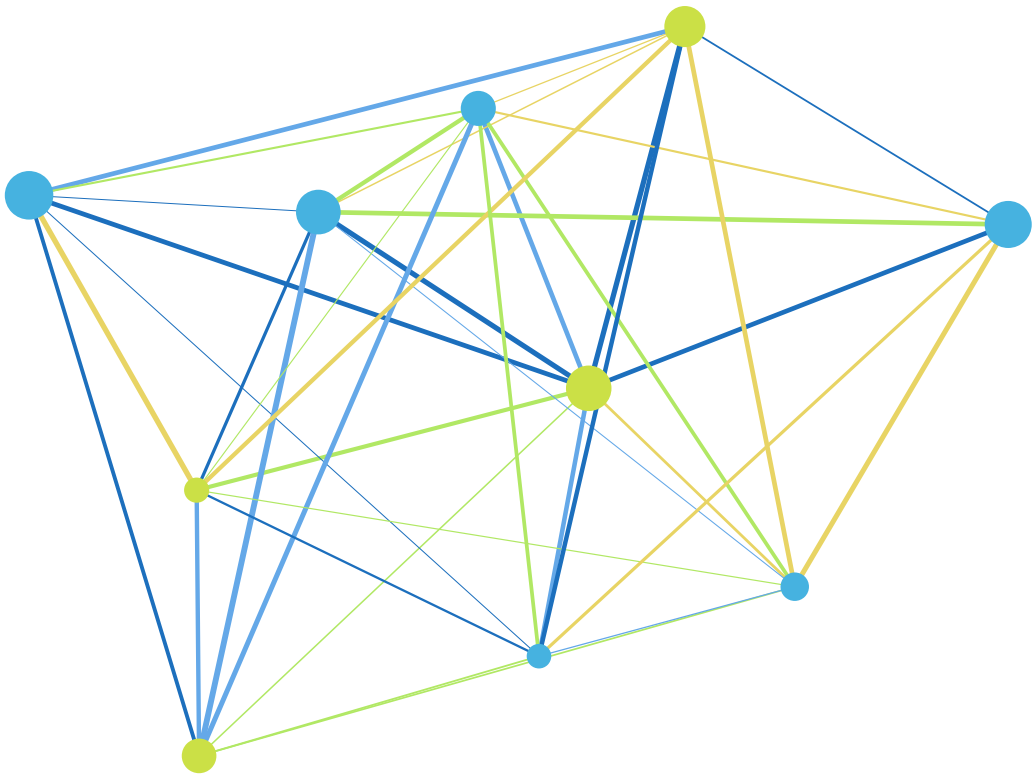


Figure – Undirected random graph generated with python

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 : $\frac{n(n-1)}{2} = \binom{n}{2} = \frac{n!}{2!(n-2)!}$

Exercice 1 : Please **cd ./graphs/random_graphs** and use the notebook **Random_undirected_graph.ipynb** or **random_undirected_graph.py** to generate a random undirected graph with a chosen number of nodes and edges.

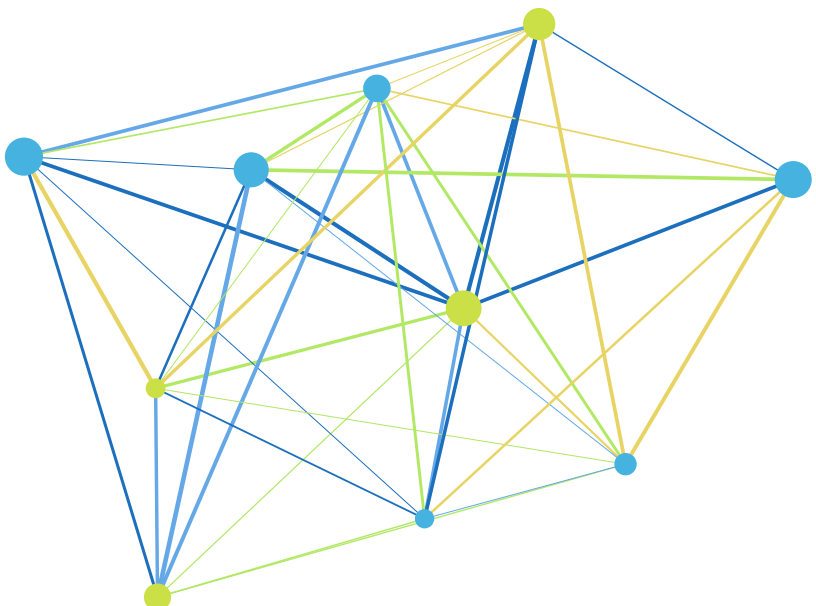


Figure – Random undirected graph with 10 nodes, 40 edges

Exercice 2 : Please use **random_directed_graph.py** to generate a random directed graph with a chosen number of nodes and edges.

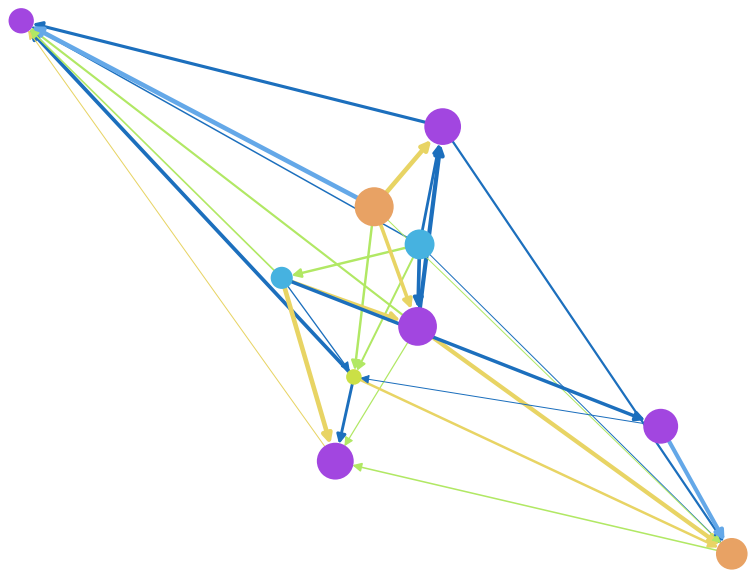
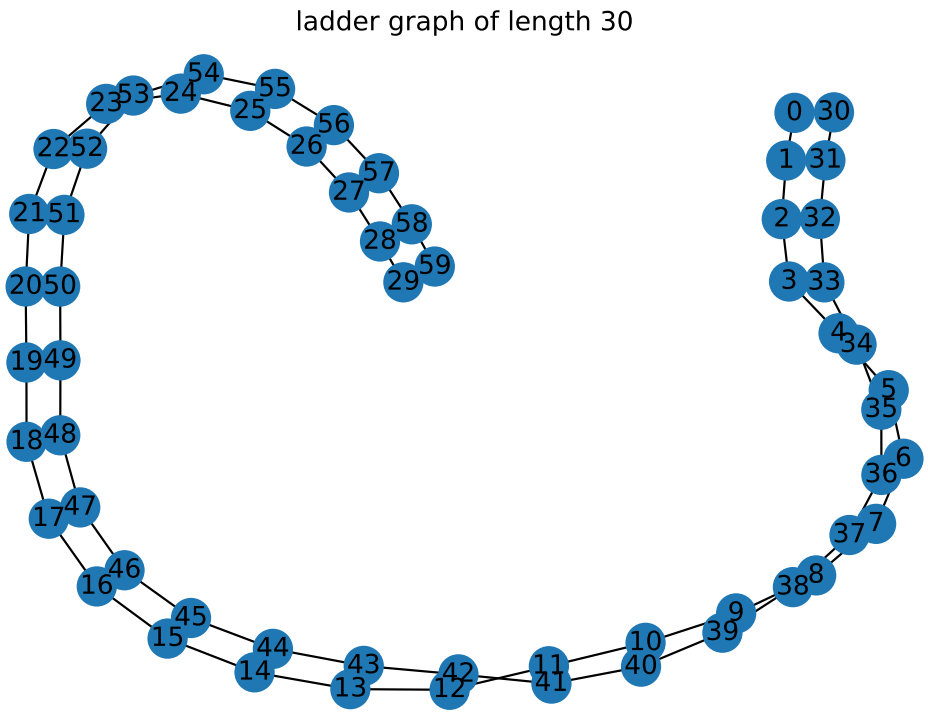


Figure – Random directed graph with 10 nodes, 30 edges

Networkx lib

We can also generate graphs with **networkx**. `https://networkx.org/documentation/stable/reference/generators.html`



Networkx

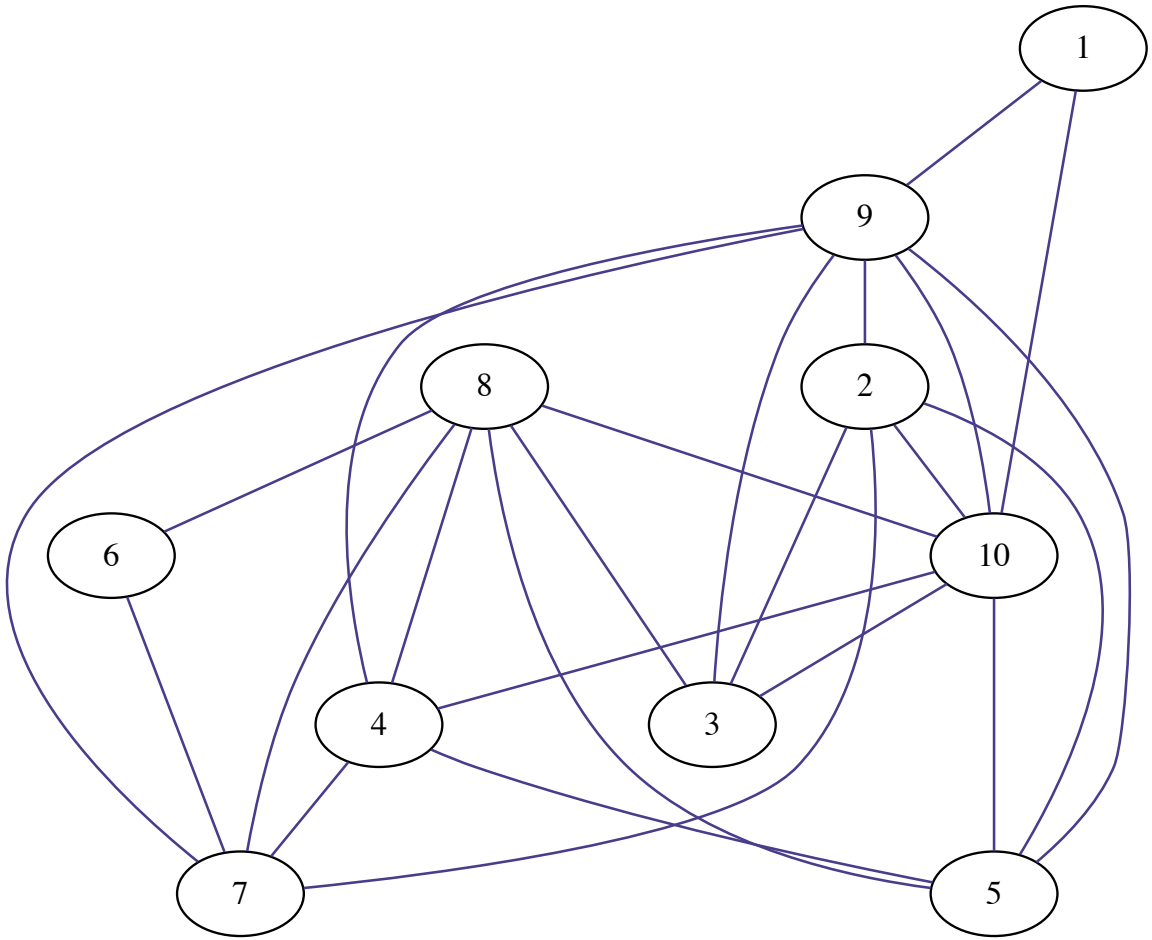
Networkx can be used to convert to or from common data structures (see **conversion_nx.py**)

```
G edges
[(0, 19), (0, 17), (0, 1), (0, 5), (0, 8), (0, 15), (0, 4), (1, 4), (1, 12),
 19), (6, 15), (6, 10), (7, 9), (7, 16), (8, 11), (8, 9), (9, 17), (9, 14), (
7), (15, 16), (17, 18)]

G as dictionary of lists
{0: [19, 17, 1, 5, 8, 15, 4], 1: [0, 4, 12, 13], 2: [10, 3, 12], 3: [16, 11,
, 14, 11, 8, 16, 12], 10: [2, 4, 6, 12], 11: [9, 8, 3, 17], 12: [4, 19, 5, 2
 [9, 0, 18, 13, 11], 18: [17], 19: [0, 12, 6, 5, 13]}

G as a numpy array
[[0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1.]
 [1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 0.]
 [0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 1. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]]
➜  random_graphs git:(correction) ✗ pp conversion_nx.py
```

The dominating set problem



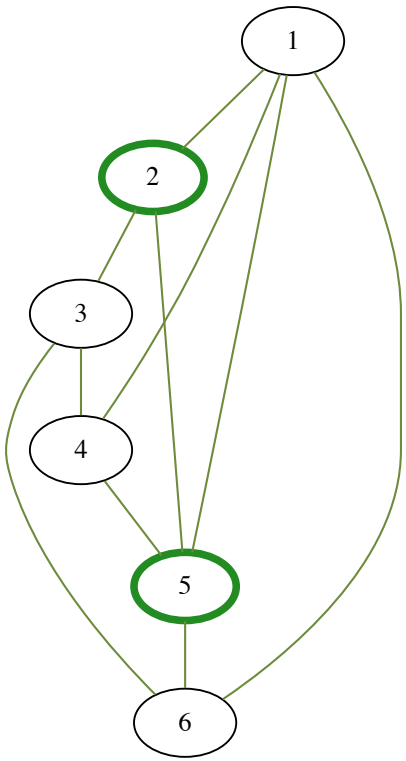
Dominating set

Say we want to cover an internet network. Some nodes (the emitters) are able to transmit information in the network, but not to all nodes : only to the nodes that are close enough.

Dominating set

Say you want to cover a internet network. Some nodes (the emitters) are able to transmit information in the network, but not to all nodes : only to the nodes that are close enough.
Optimization problem : You need to cover the network, but with the smallest possible number of emitters (in order to save money, infrastructure, material, etc.).
Exercice 3 : How could we formalize this problem with a **graph** ?

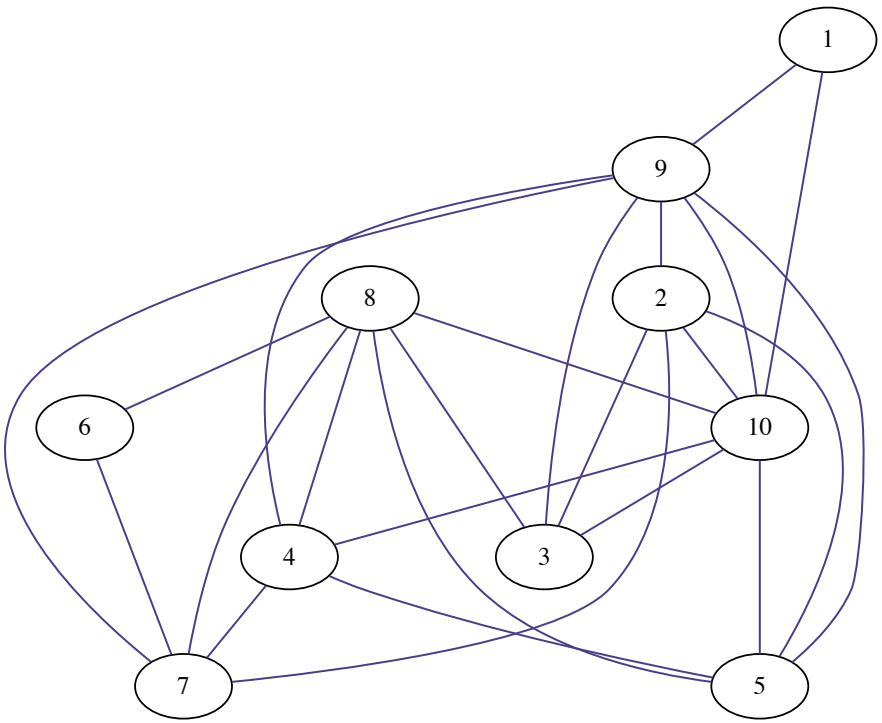
The dominating set problem



Mathematically speaking : if $G(V, E)$ is the graph. We look for a **subset of nodes** D such that **all nodes in the graph** are the neighbor of **at least one node** in D.

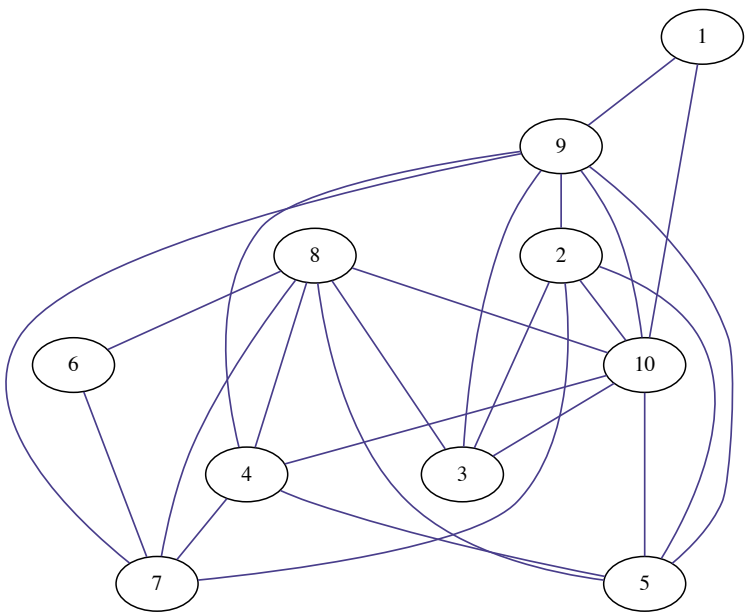
The dominating set problem

Mathematically speaking : if $G(V, E)$ is the graph. We look for a **subset of nodes** D such that **all nodes in the graph** are the neighbor of **at least one node** in D.



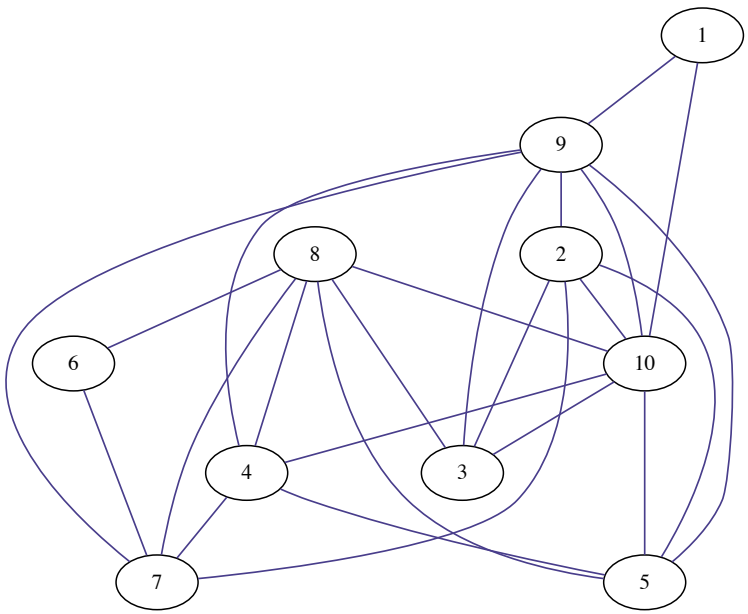
The dominating set problem

Mathematically speaking : if $G(V, E)$ is the graph. We look for a **subset of nodes** D such that **all nodes in the graph** are the neighbor of **at least one node** in D. And we want to pick the **smallest D** that "dominates" the network.



The dominating set problem

What is the most trivial dominating subset ?



Dominating set : example 1

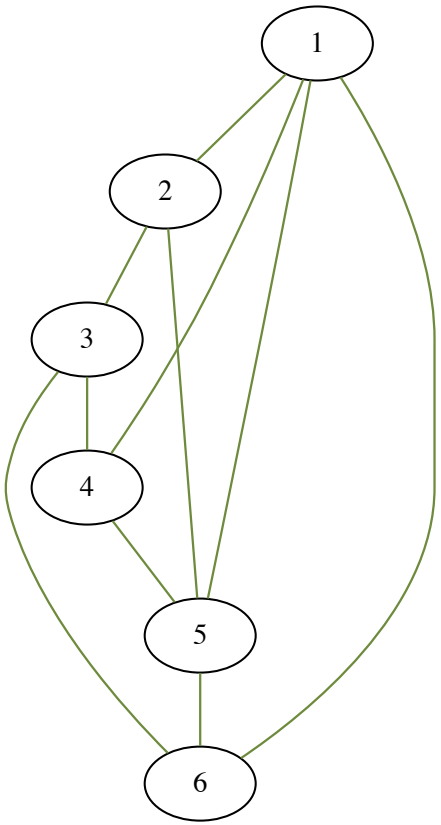


Figure – Some simple graph

Dominating set : example 1

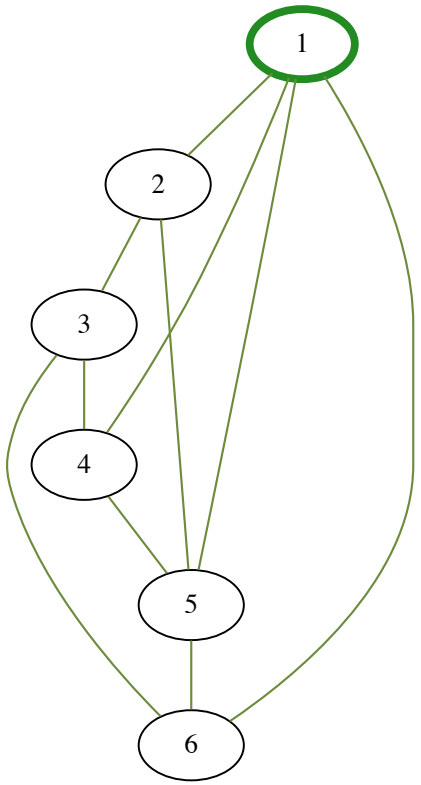


Figure – Is this a dominating subset ?

Dominating set : example 1

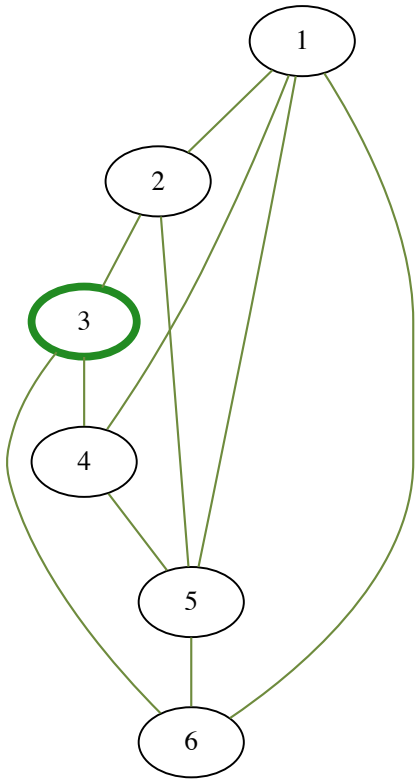


Figure – Is this a dominating subset ?

Dominating set : example 1

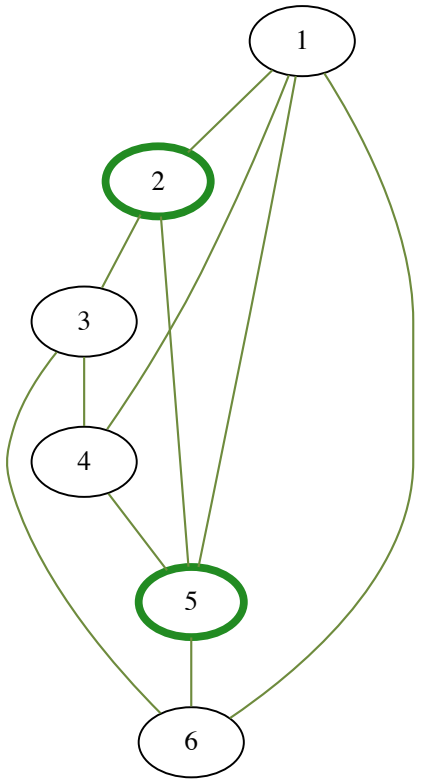


Figure – Is this a dominating subset ?

Dominating set : example 1

A **minimal dominating set** is a dominating set D such that removing any node from D prevents it from still being dominating.

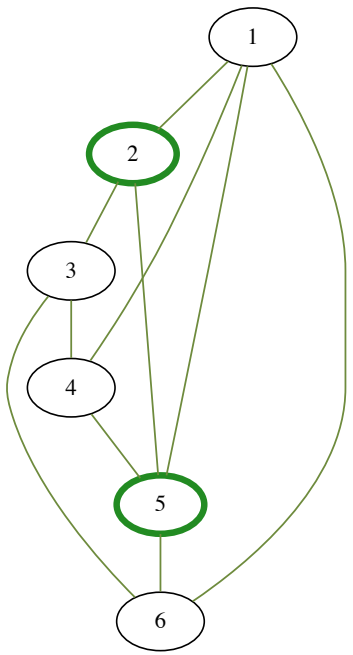


Figure – Concept of minimal dominating set.

Dominating set : example 1

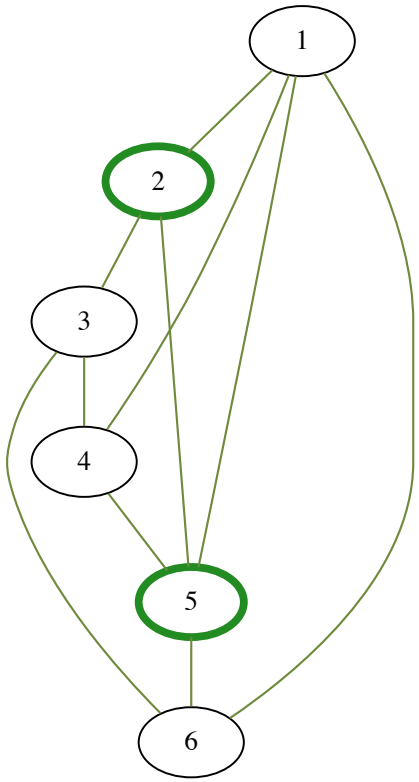
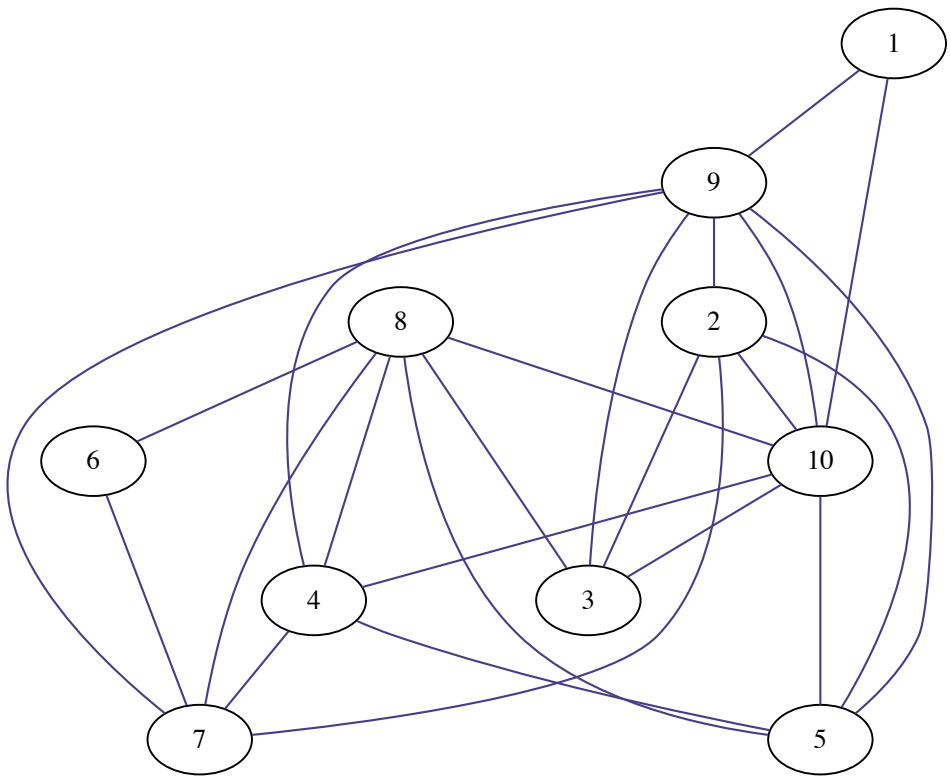


Figure – Is this a dominating subset ? Yes. Is it minimal ?

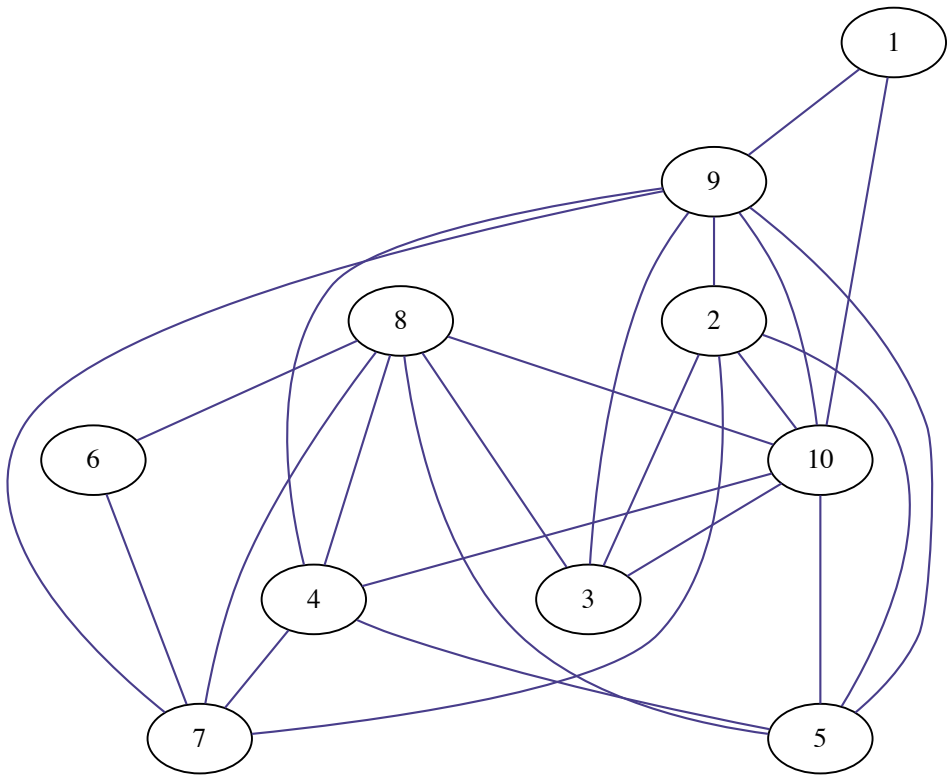
Dominating set : example 2

Please find a dominating set in this graph.



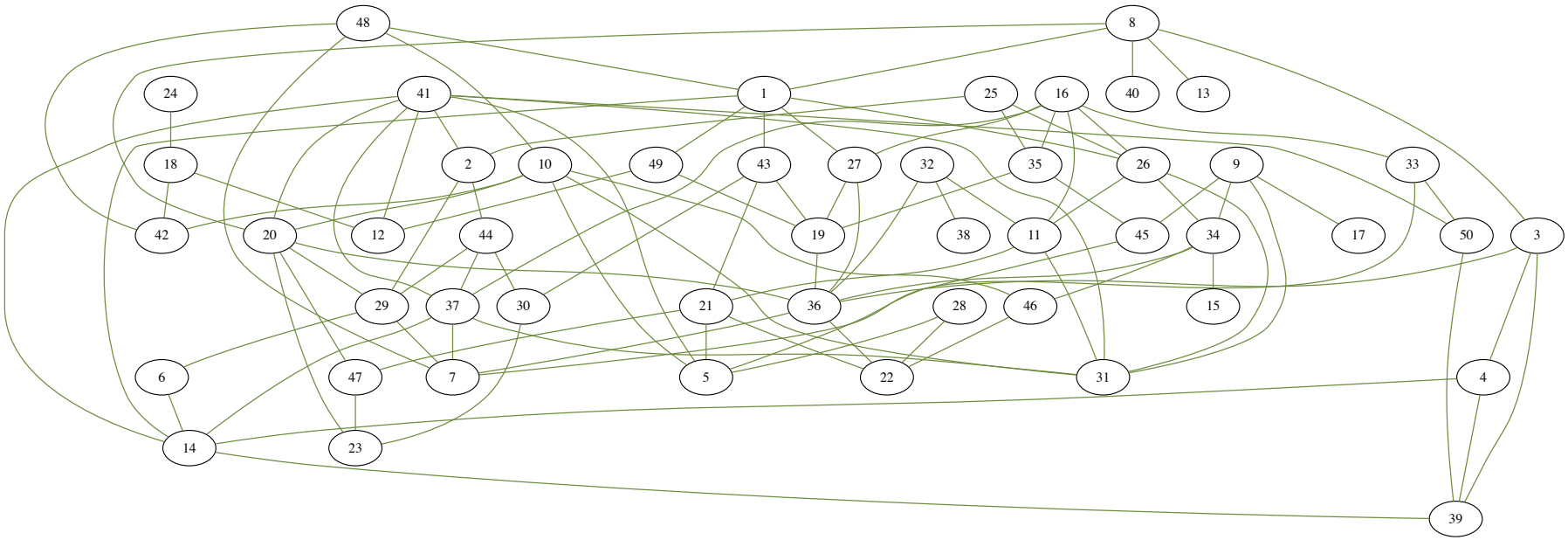
Dominating set : example 2

Please find a **minimal** dominating set in this graph.



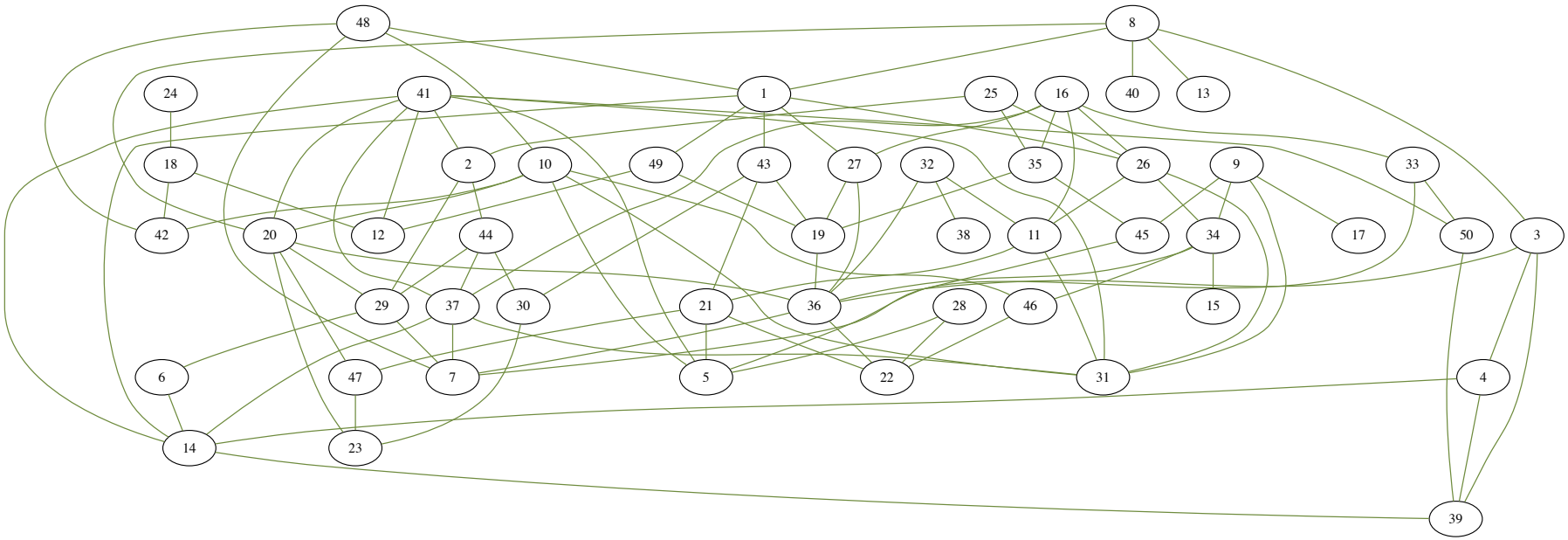
Dominating set : example 3

Please find a **minimal** dominating set in this graph.



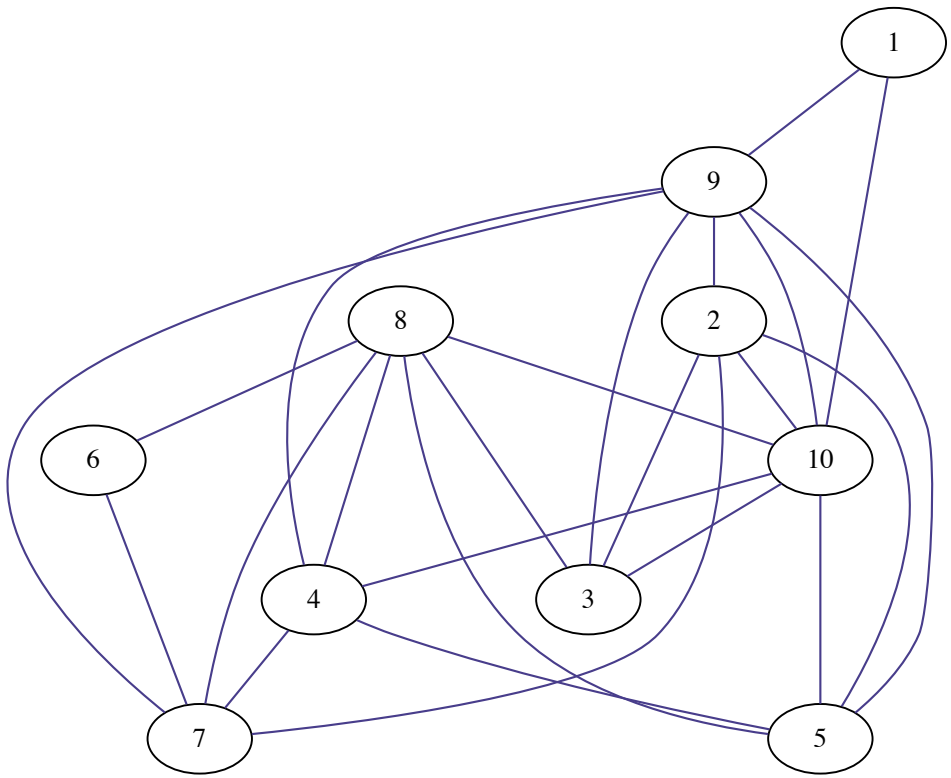
Dominating set : example 3

Is **minimal** the same thing as minimum ?



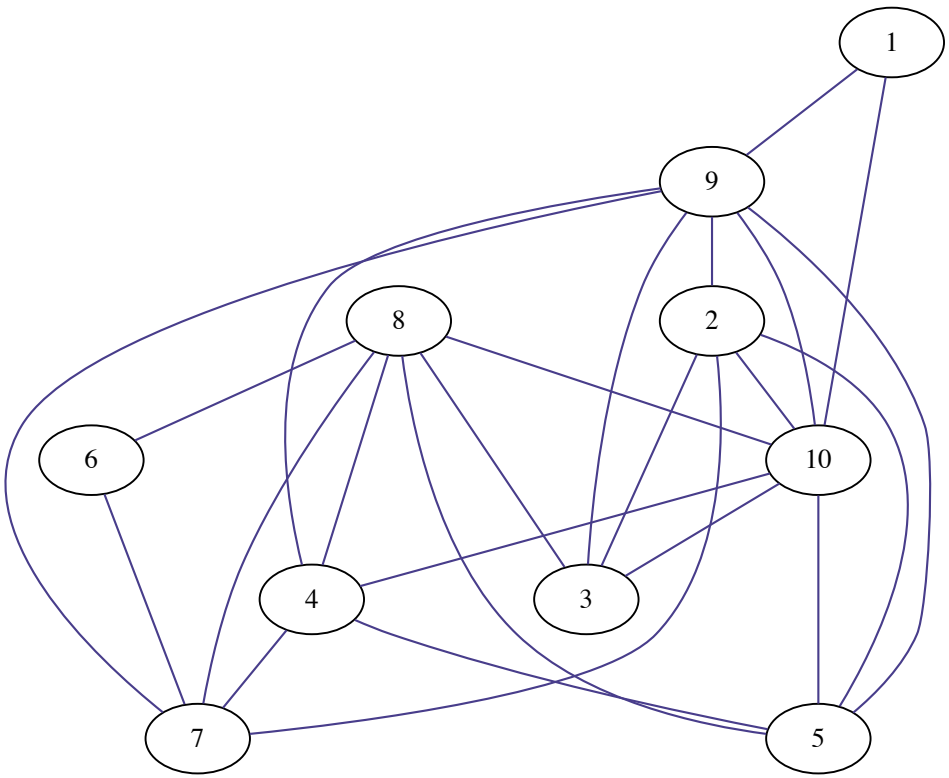
Dominating set : exhaustive search

What would be the **exhaustive search** in the case of the Dominating set problem ?



Dominating set : exhaustive search

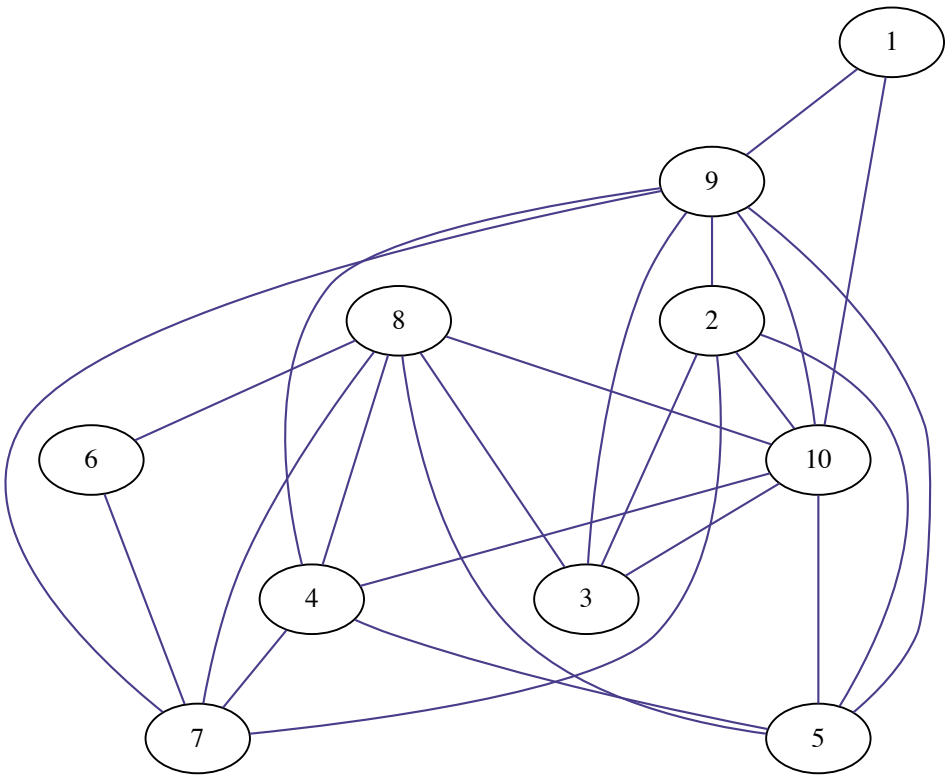
How many possibilities do have to try as a function of n ?



Dominating set : exhaustive search

How many possibilities do have to try as a function of n ?

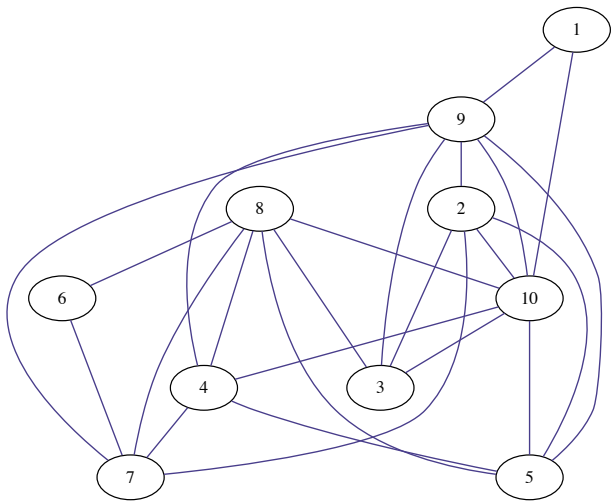
The number of subsets in $[1 : n]$ is :



Dominating set : exhaustive search

How many possibilities do have to try as a function of n ?
The number of subsets in $[1 : n]$ is :

$$2^n = \sum_{k=0}^{n} \binom{n}{k} \tag{1}$$



Heuristic

The exhaustive search is no possible. So what method should we use ?

Heuristic

Ok so the exhaustive search is no possible. So what method should we use ?
Let's build a **greedy algorithm** (heuristic).

Greedy algorithm

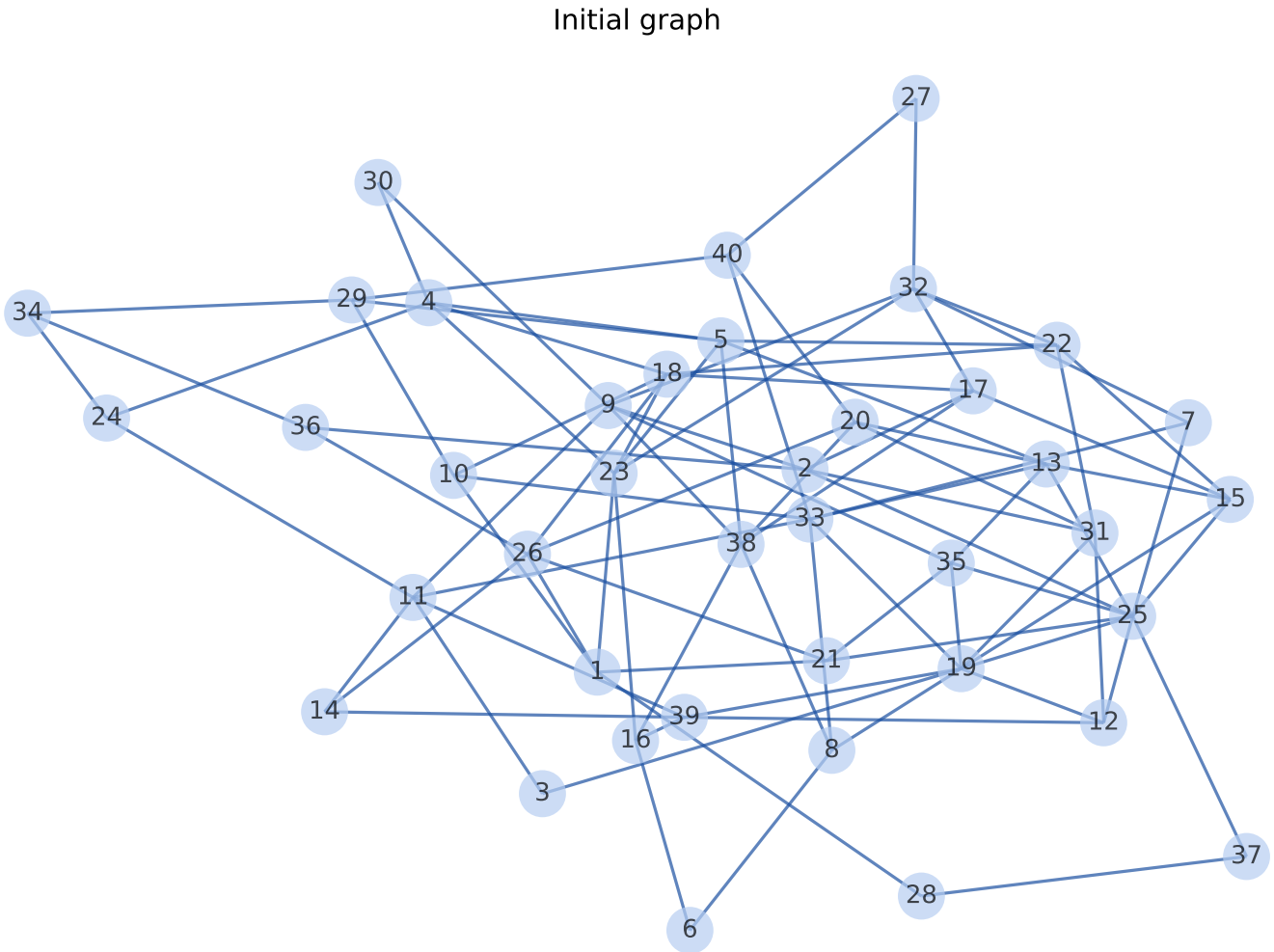
In a graph (unweighted), the **degree of a node** is its number of neighbors.

dominating set

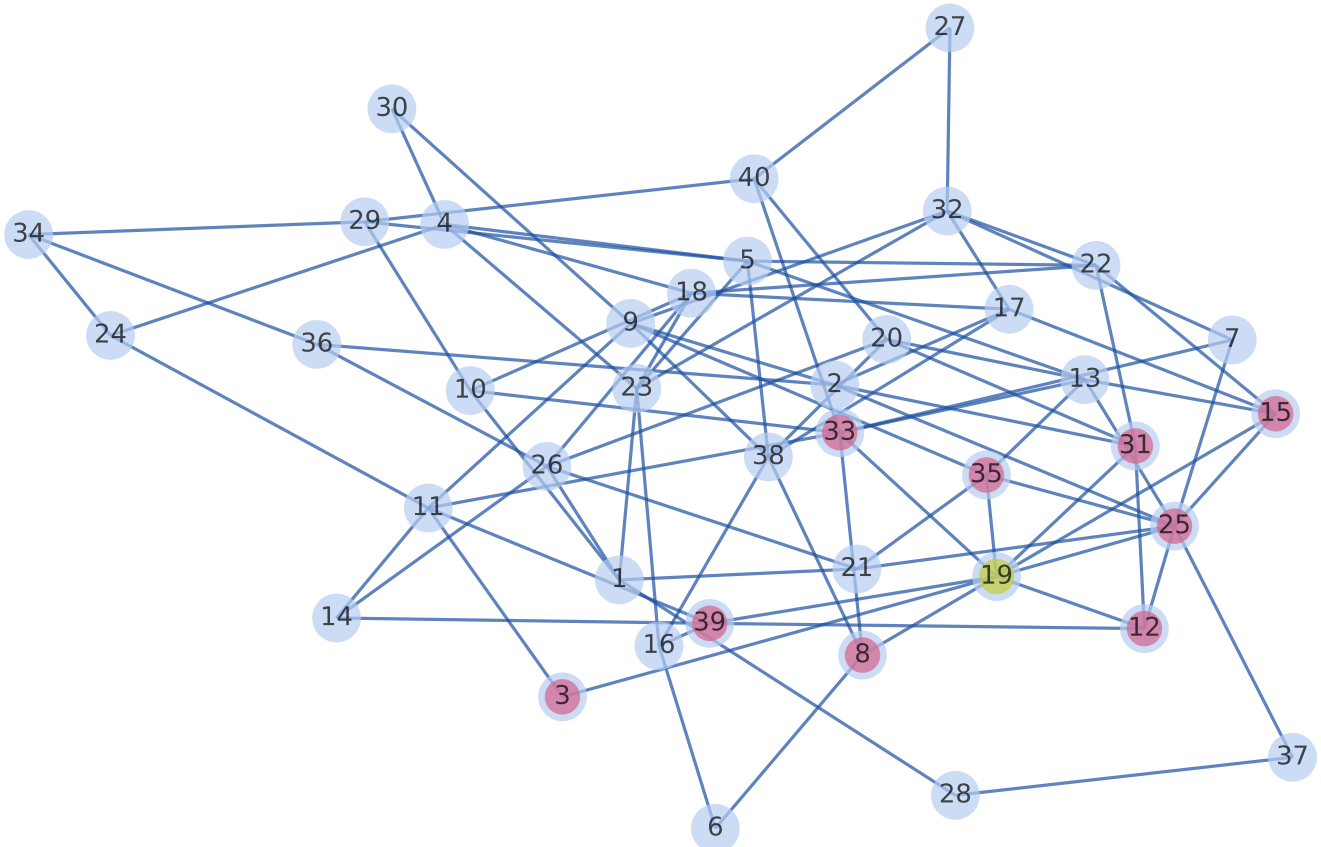
Exercice 4 : Greedy algorithm implementation

cd graphs/dominating_set and modify **greedy_standard.py** in order to apply the greedy algorithm :

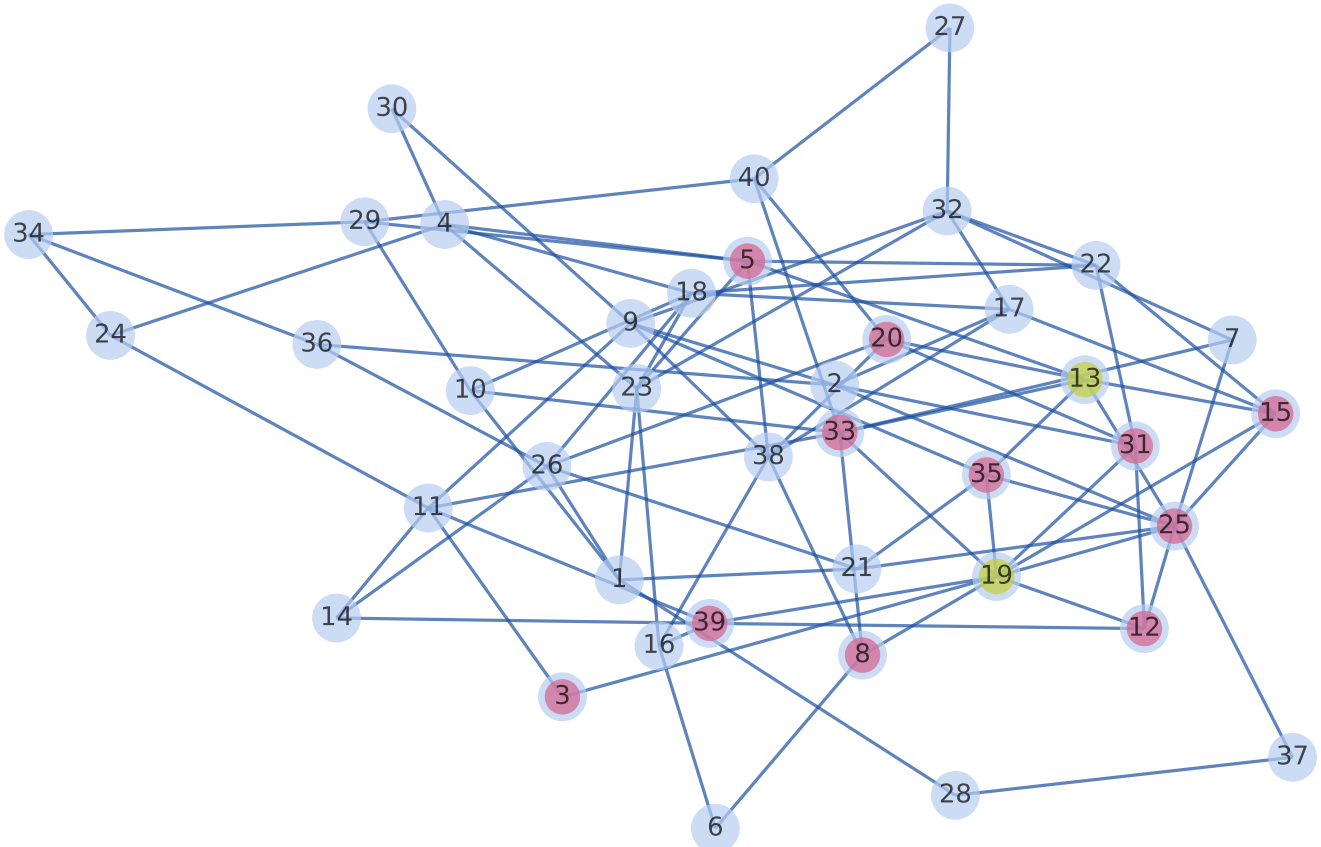
- sort nodes by degree
- progressively add the to the set until it's dominating


Initial graph
27
30
40
34
29
4
32
5
22
18
17
24
9
20
7
36
10
23
2
13
15
33
31
26
38
35
11
25
1
21
19
14
39
12
16
8
3
37
28
6

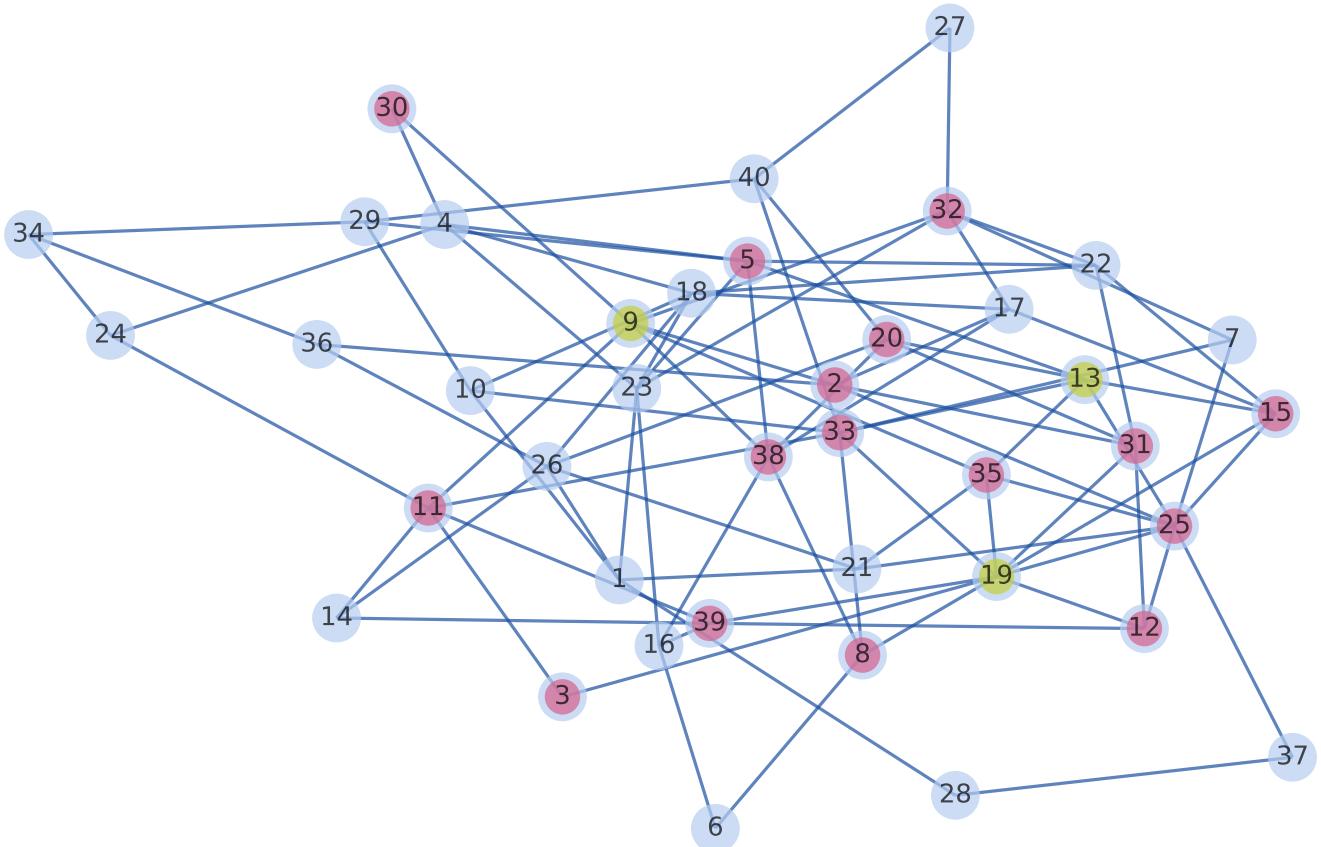
Subset size: 1
Algo step: 1



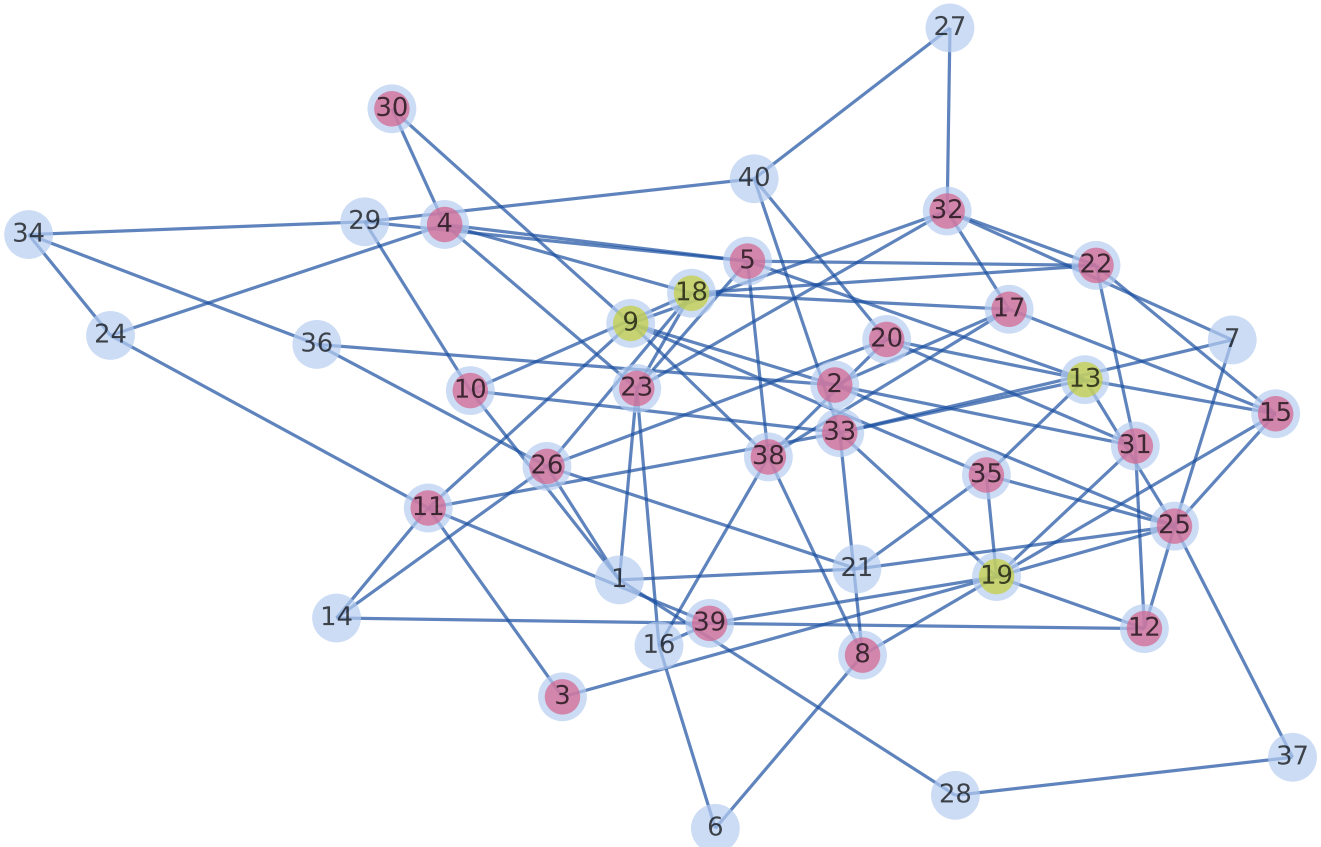
Subset size: 2
Algo step: 2



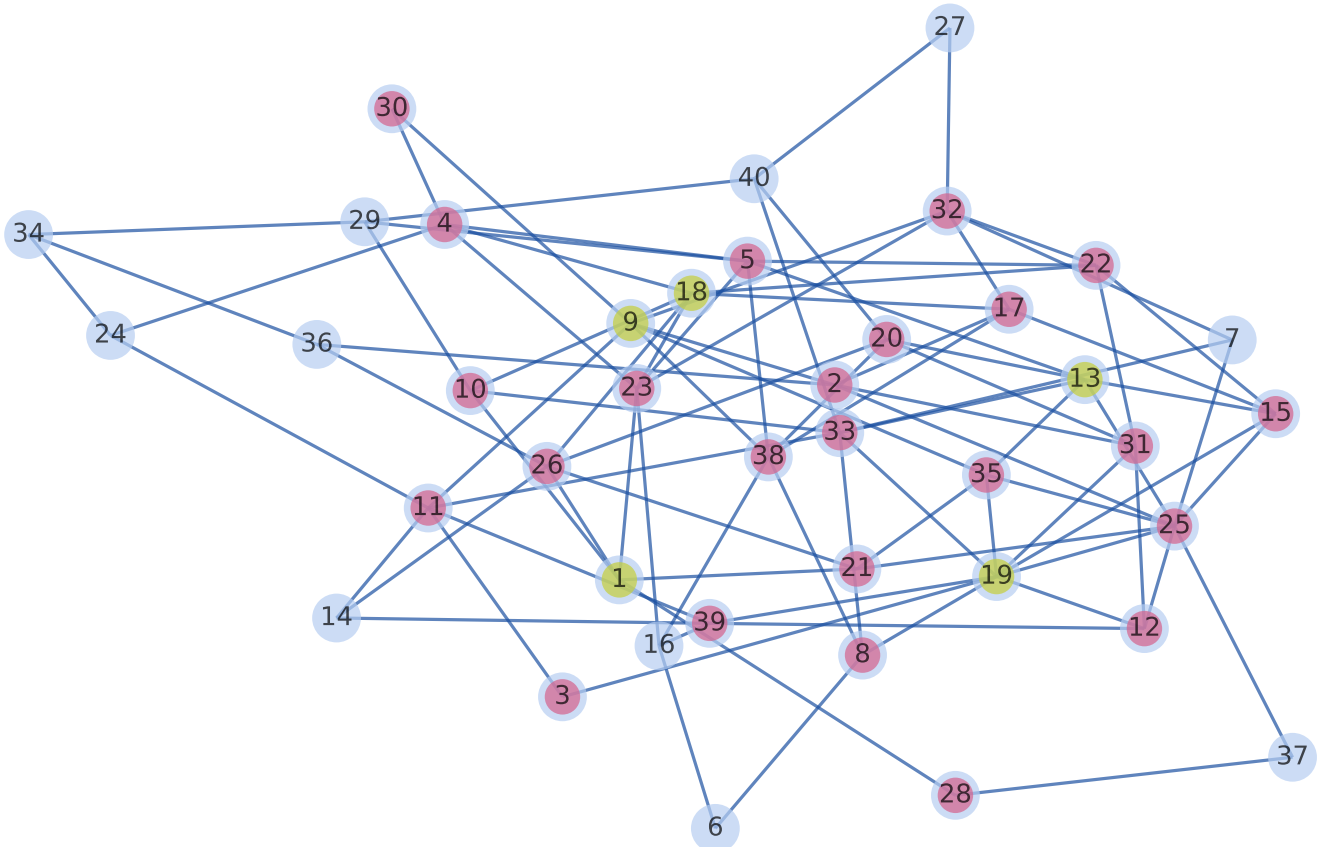
Subset size: 3
Algo step: 3



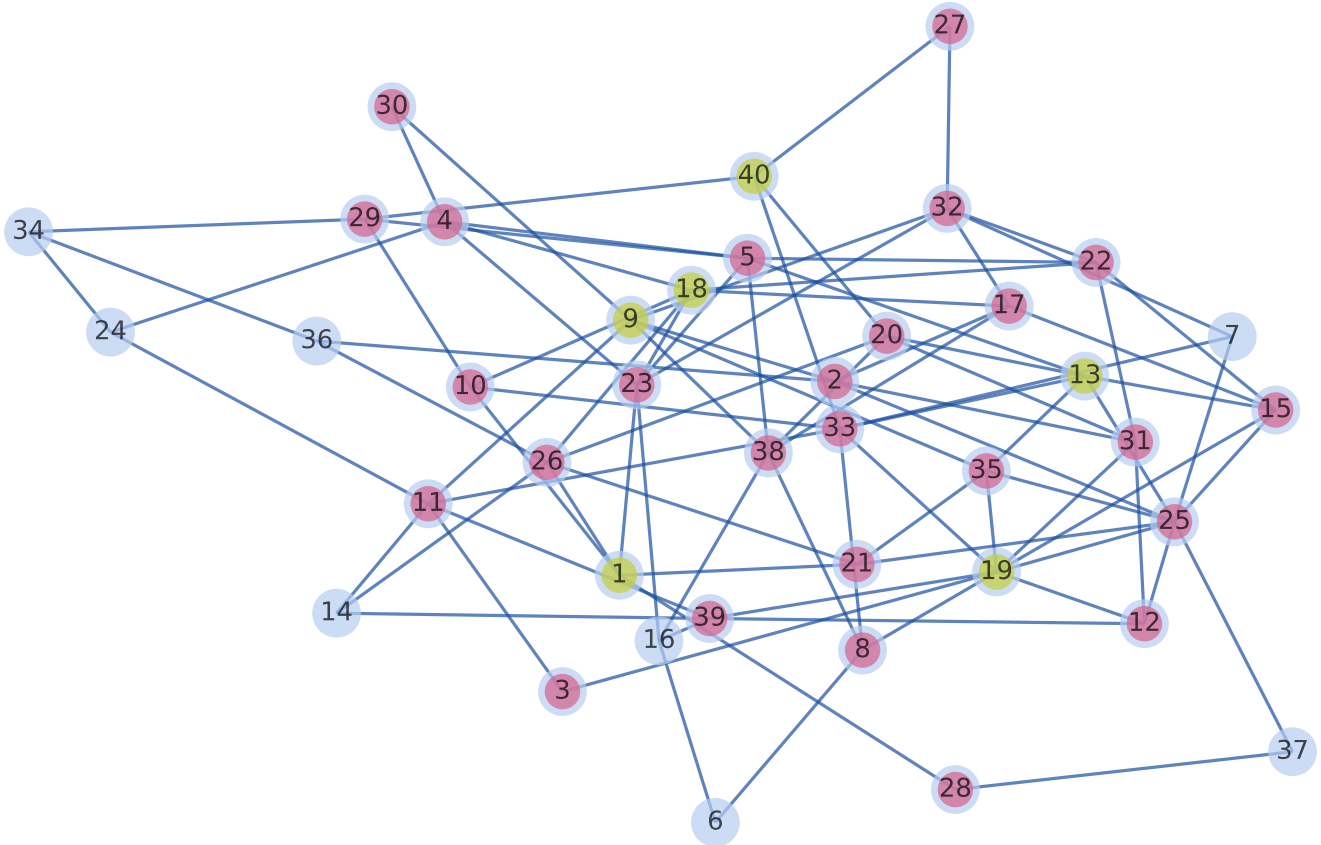
Subset size: 4
Algo step: 4



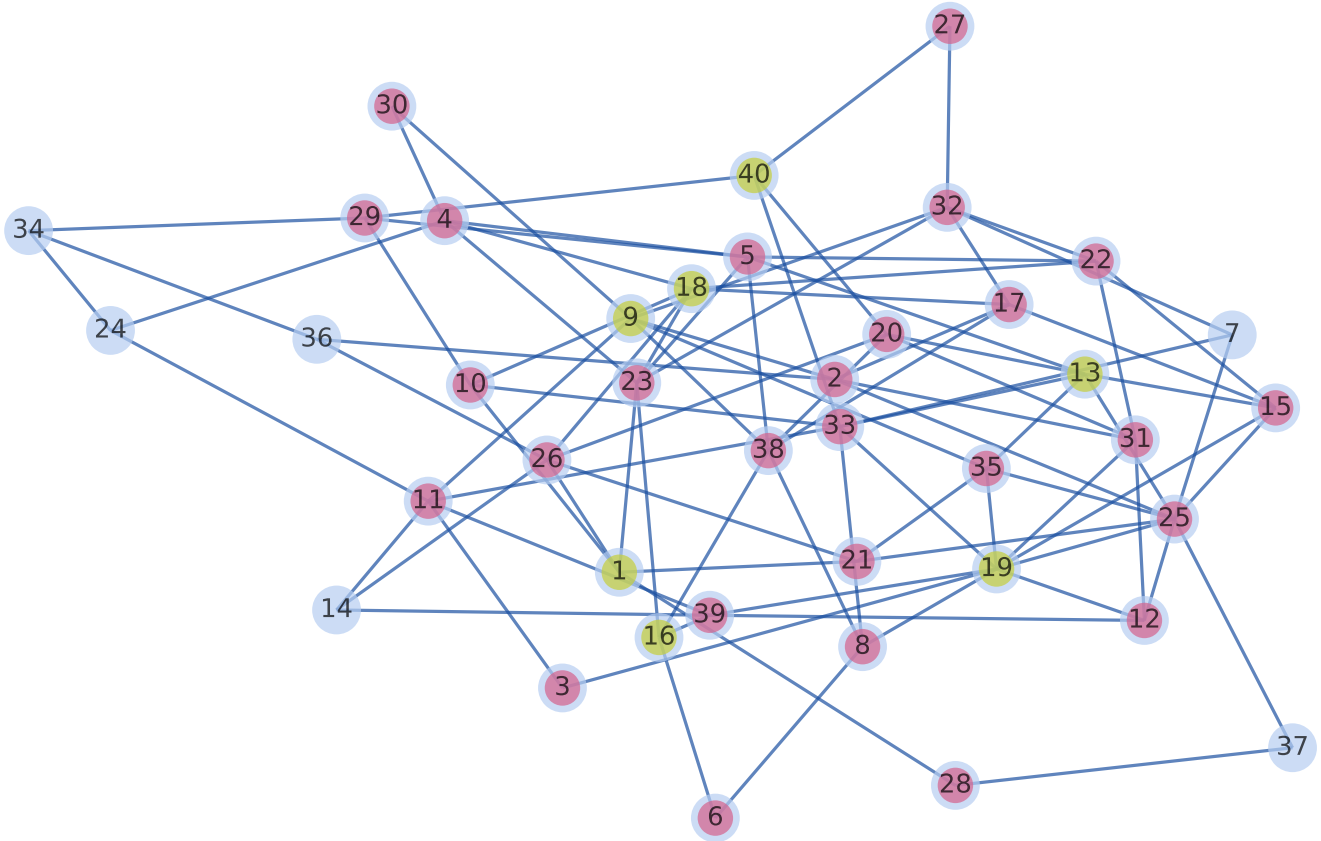
Subset size: 5
Algo step: 5



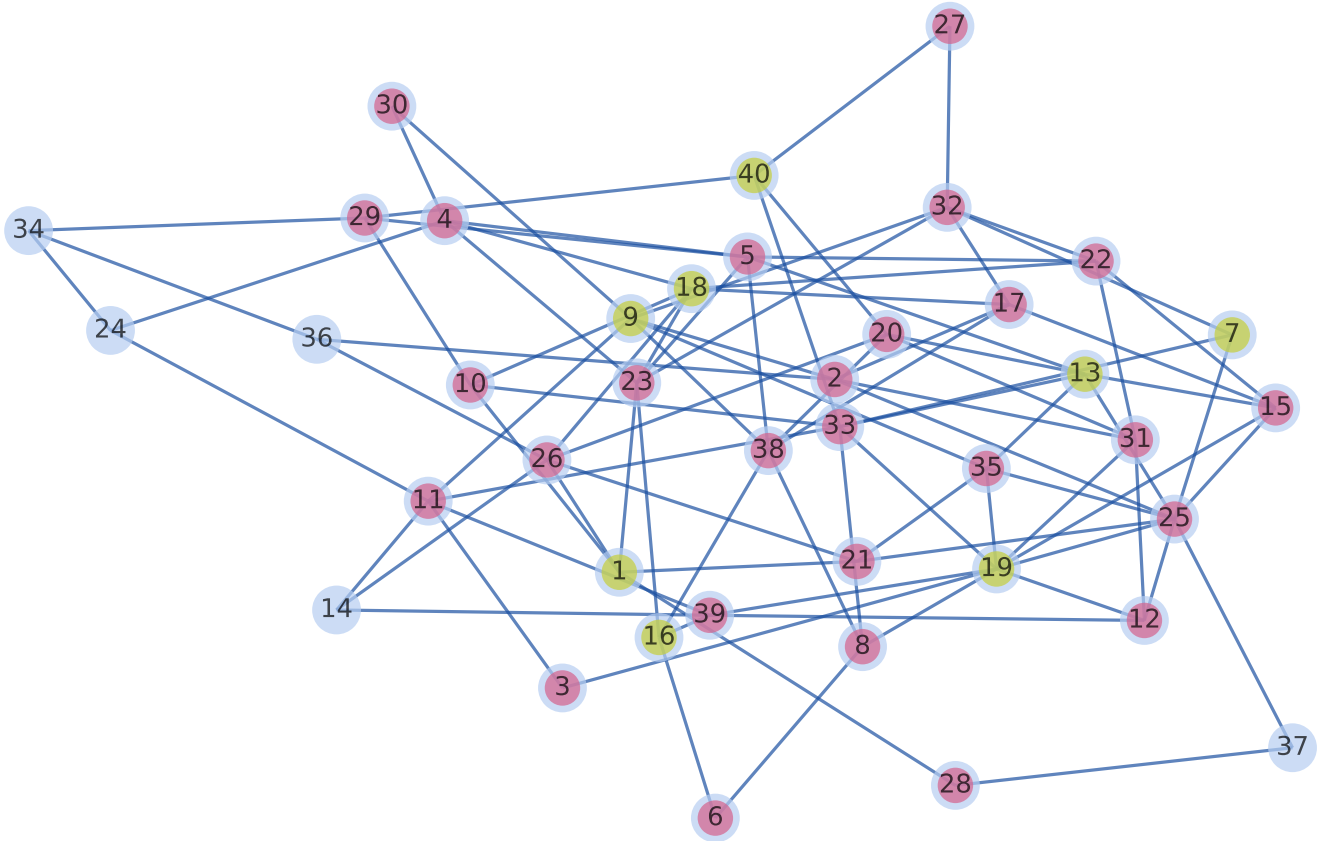
Subset size: 6
Algo step: 6



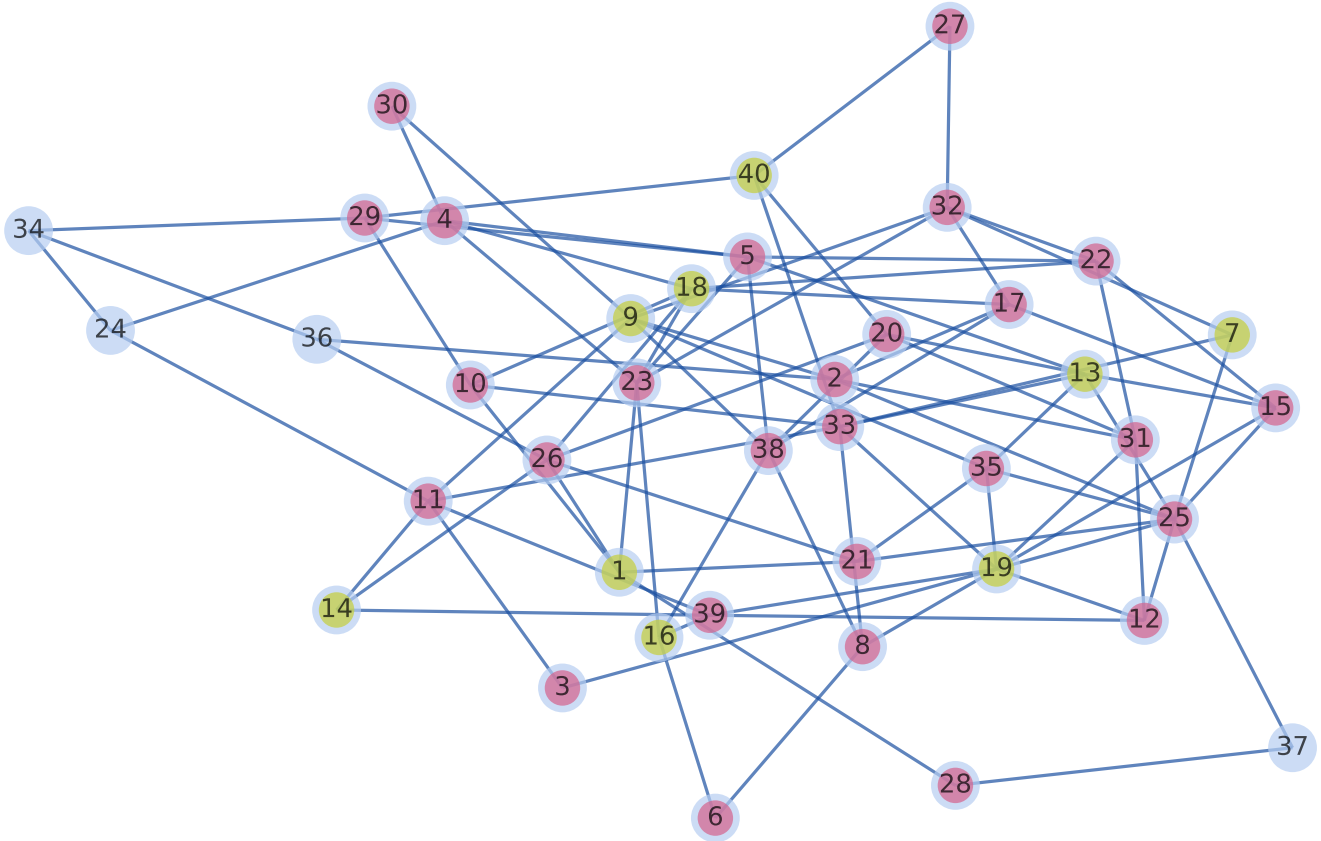
Subset size: 7
Algo step: 7



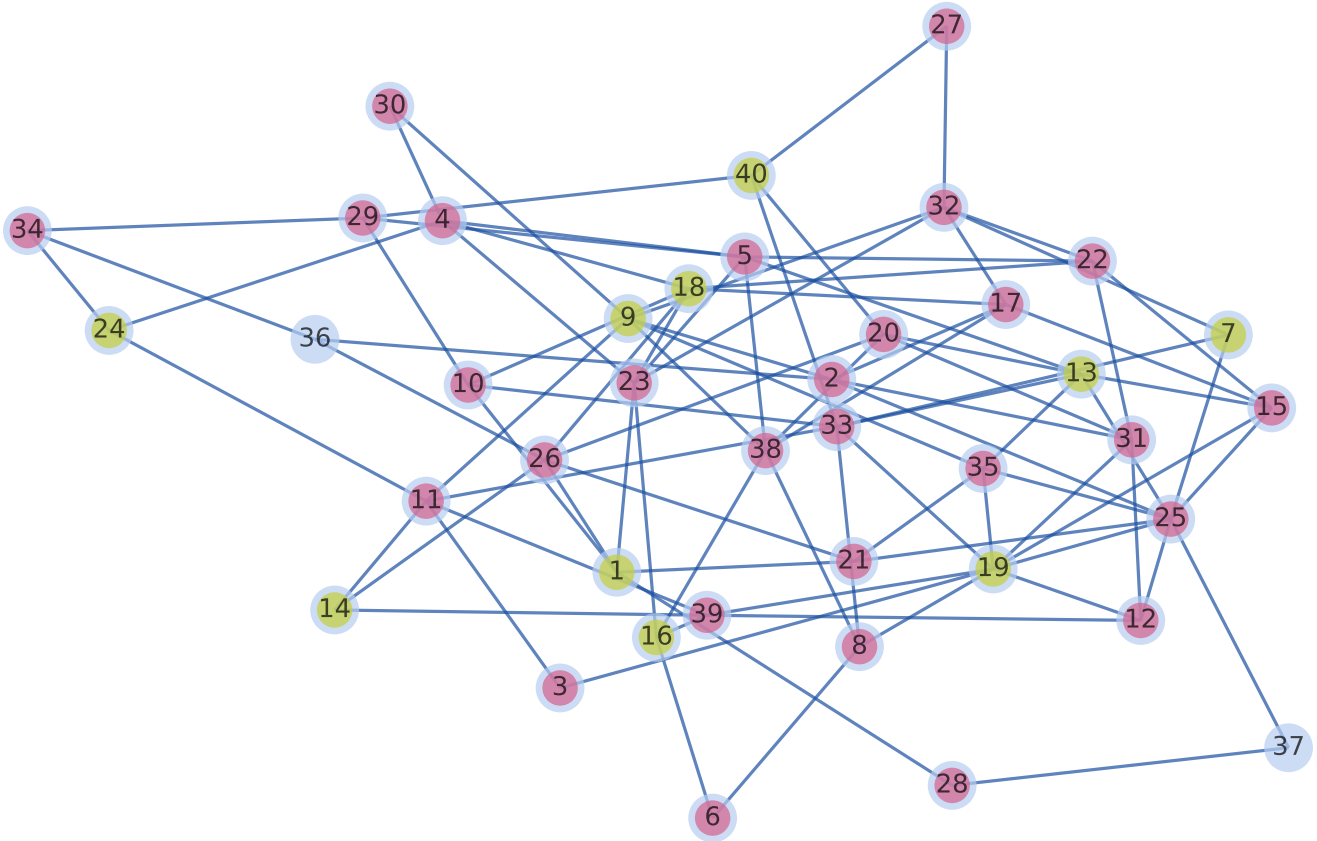
Subset size: 8
Algo step: 8



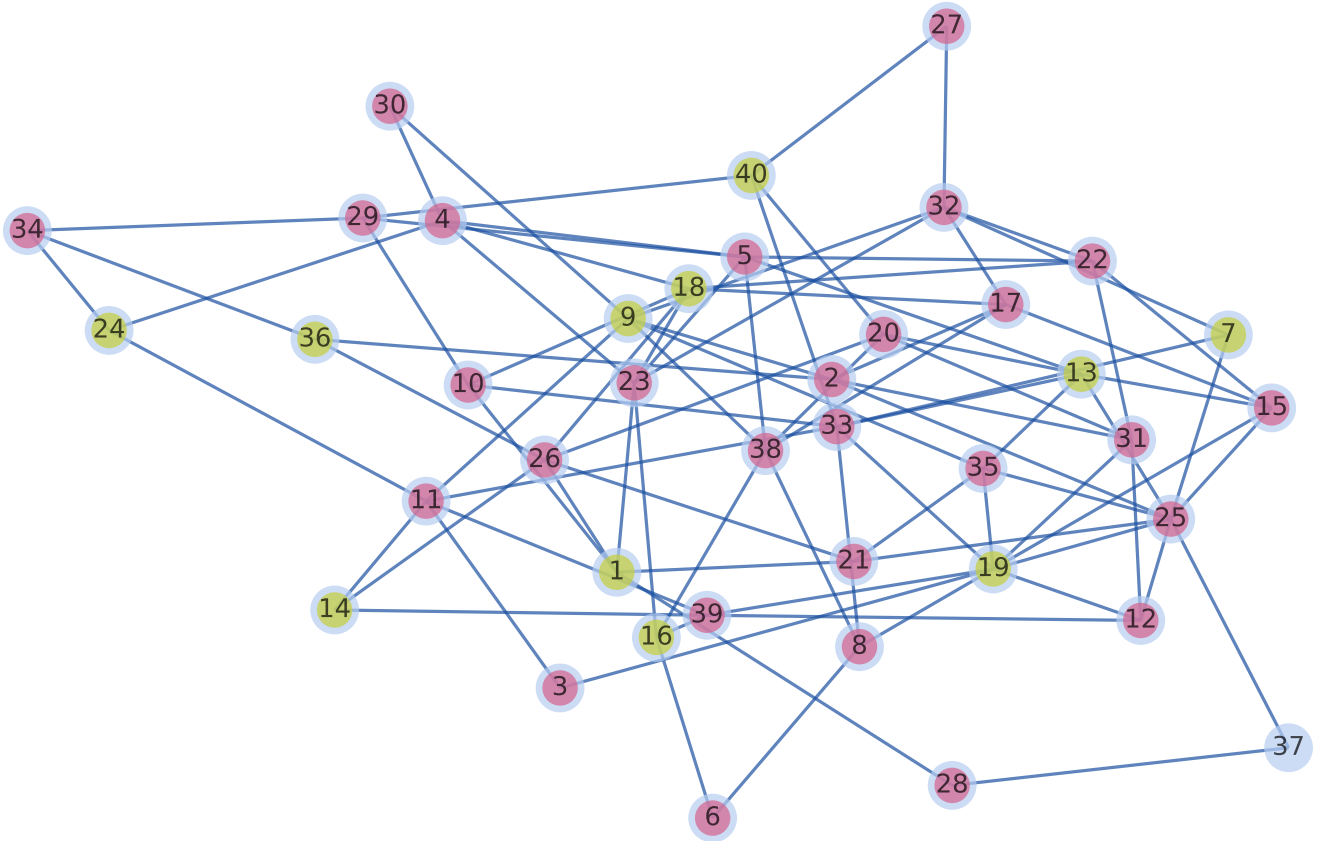
Subset size: 9
Algo step: 9



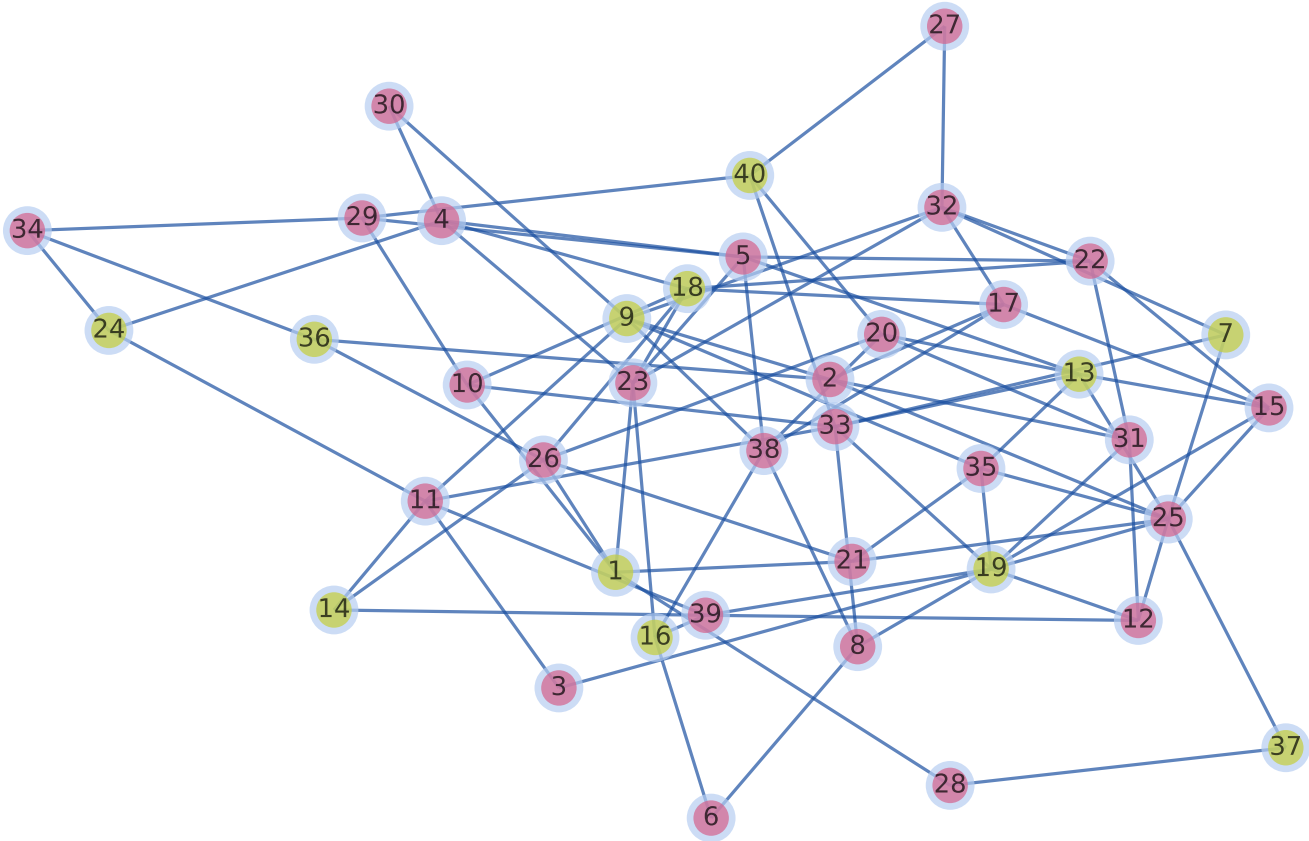
Subset size: 10
Algo step: 10



Subset size: 11
Algo step: 11



Subset size: 12
Algo step: 12



dominating set

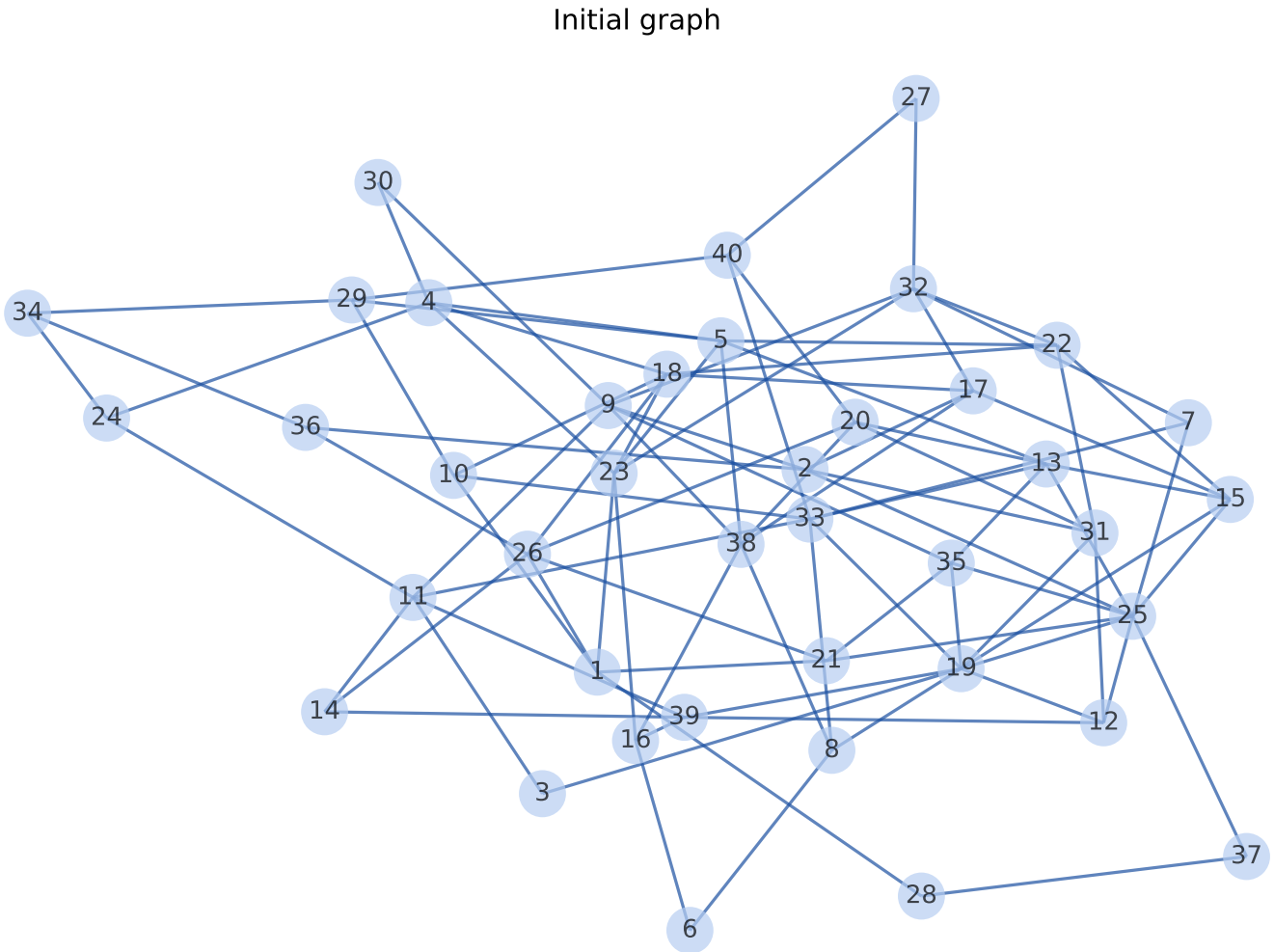
Exercice 4 : Greedy algorithm implementation
Running **main_dominating _set.py** and completing the code inside **dominating_set/greedy_standard.py**, generate new random instances of the problem and apply the algorithm to them. The parameters (like the number of nodes in the graph) might be changed in **params.py**.

Complexity

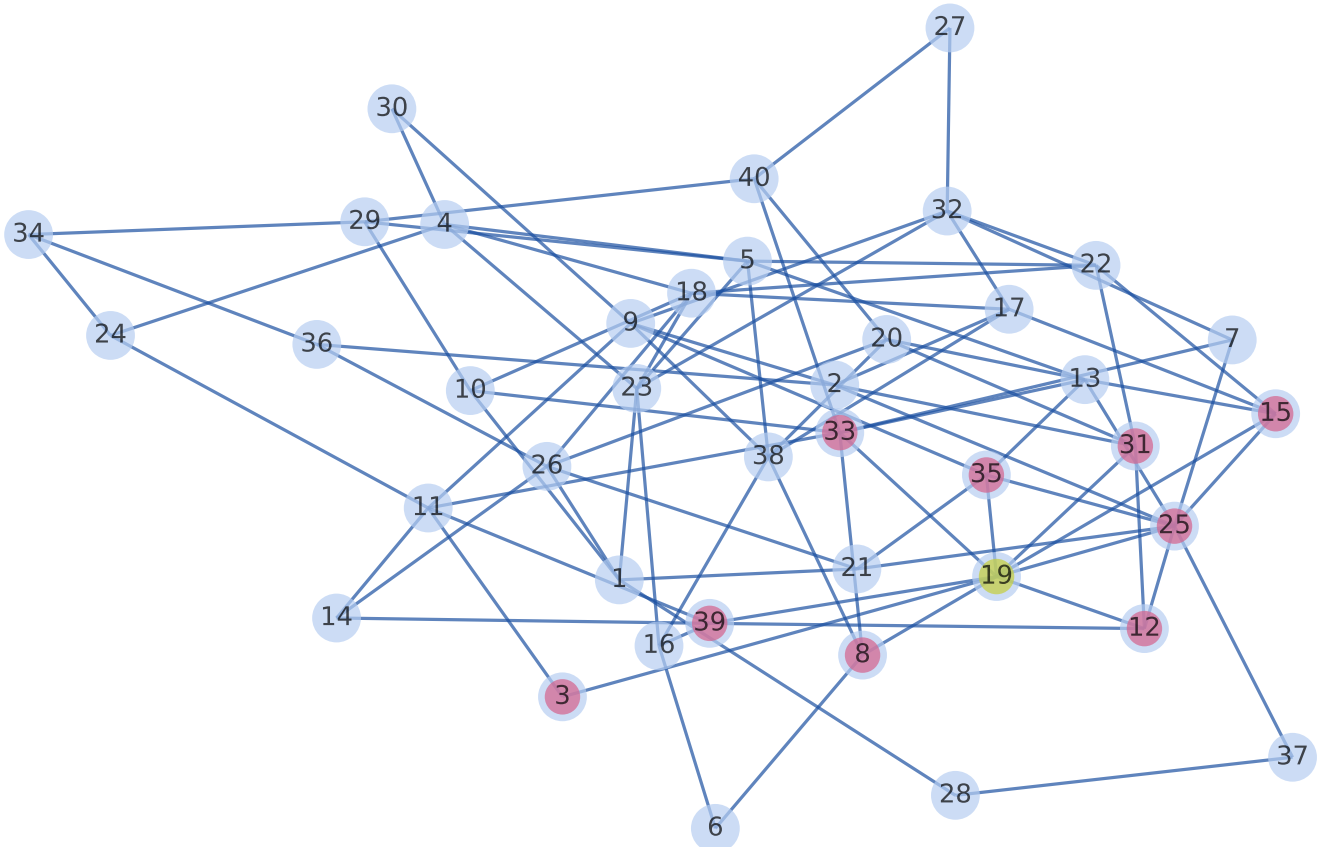
Exercice 5 : What is the complexity of the greedy algorithm ?

Variant

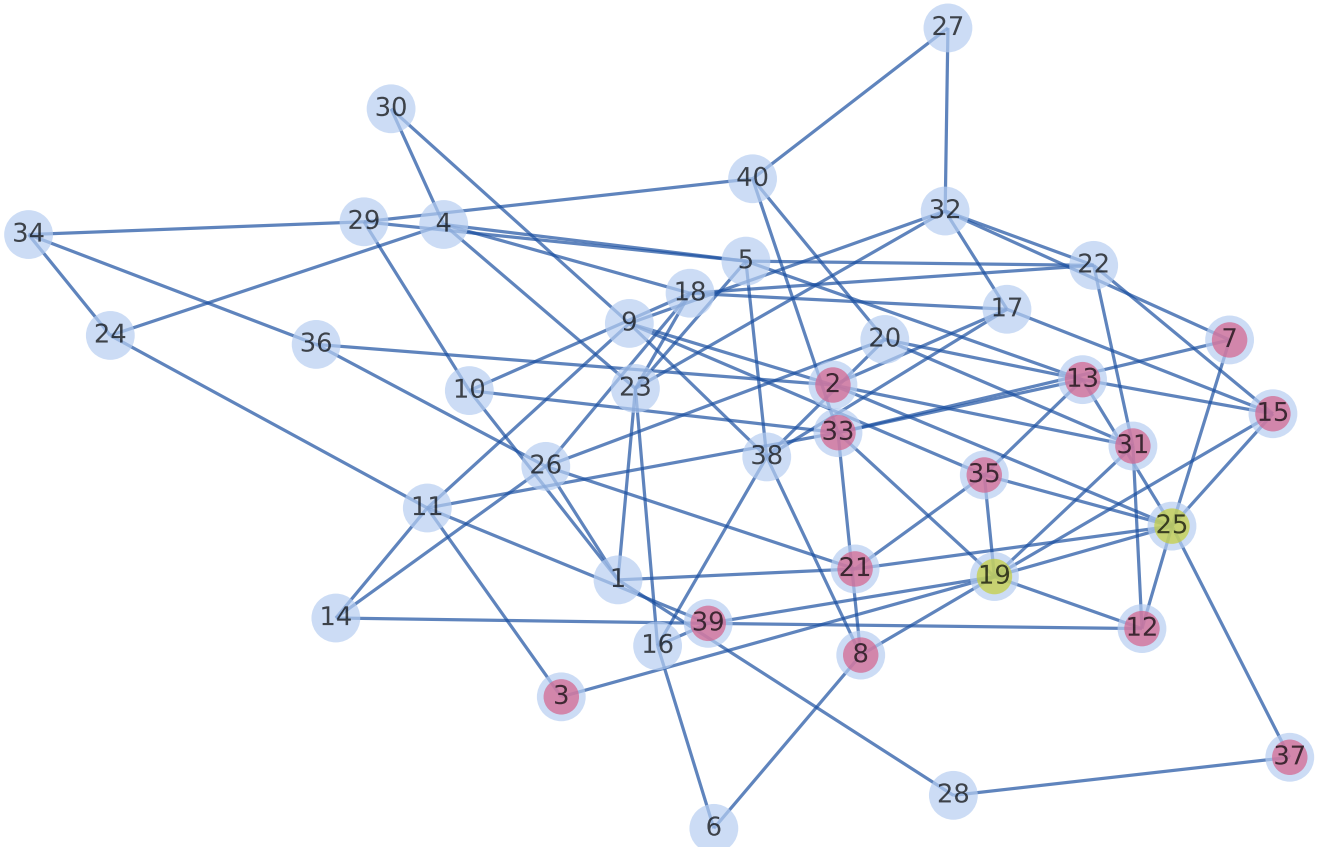
Exercice 6 : Try to see what happens using a variant of the heurstic, where we can add nodes that are already dominated to the set of selected nodes. Which method is faster ?
You can use **dominating_set/greedy_bis.py**


Initial graph
27
30
40
34
29
4
32
5
22
18
17
24
9
20
7
36
10
23
2
13
15
33
31
26
38
35
11
25
1
21
19
14
39
12
16
8
3
37
28
6

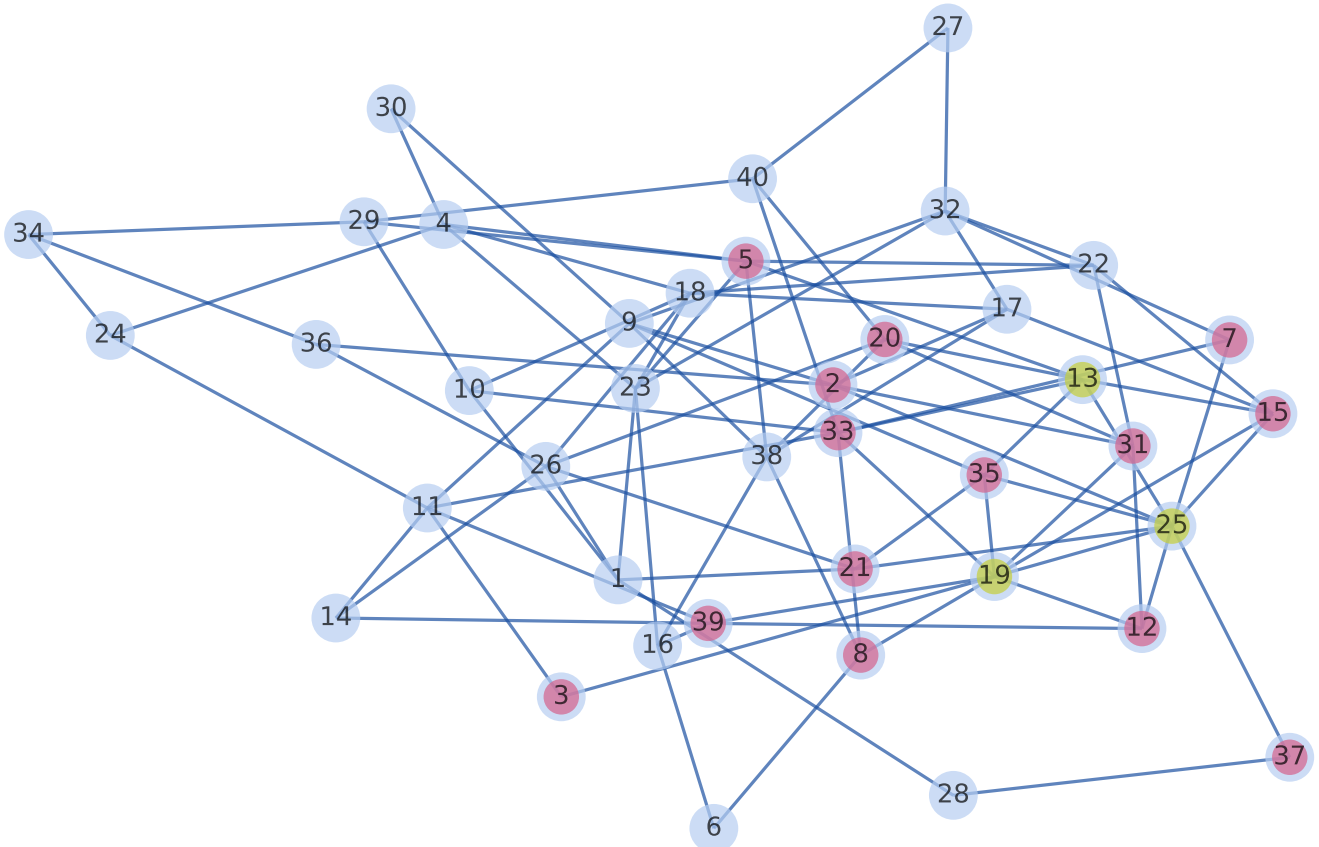
Subset size: 1
Algo step: 1



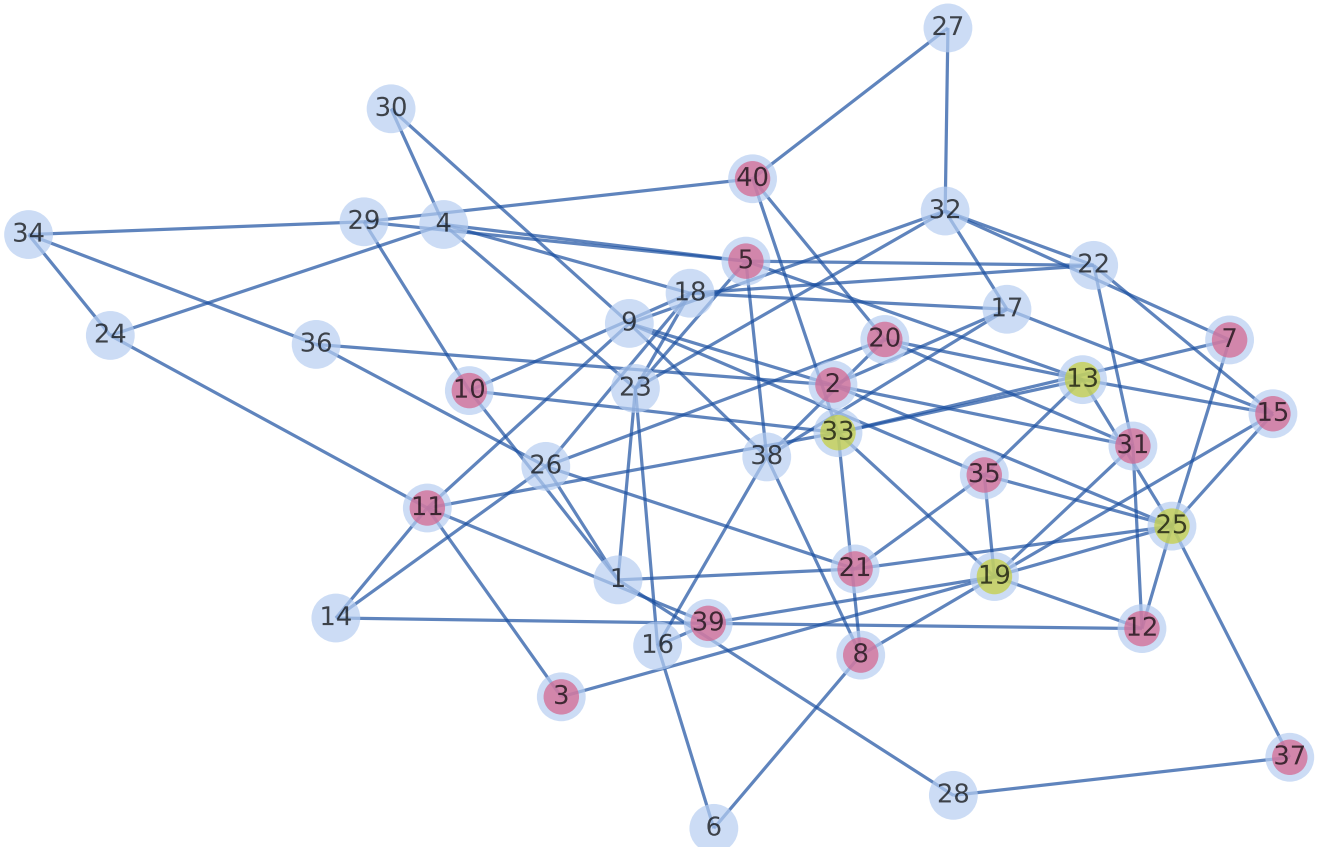
Subset size: 2
Algo step: 2



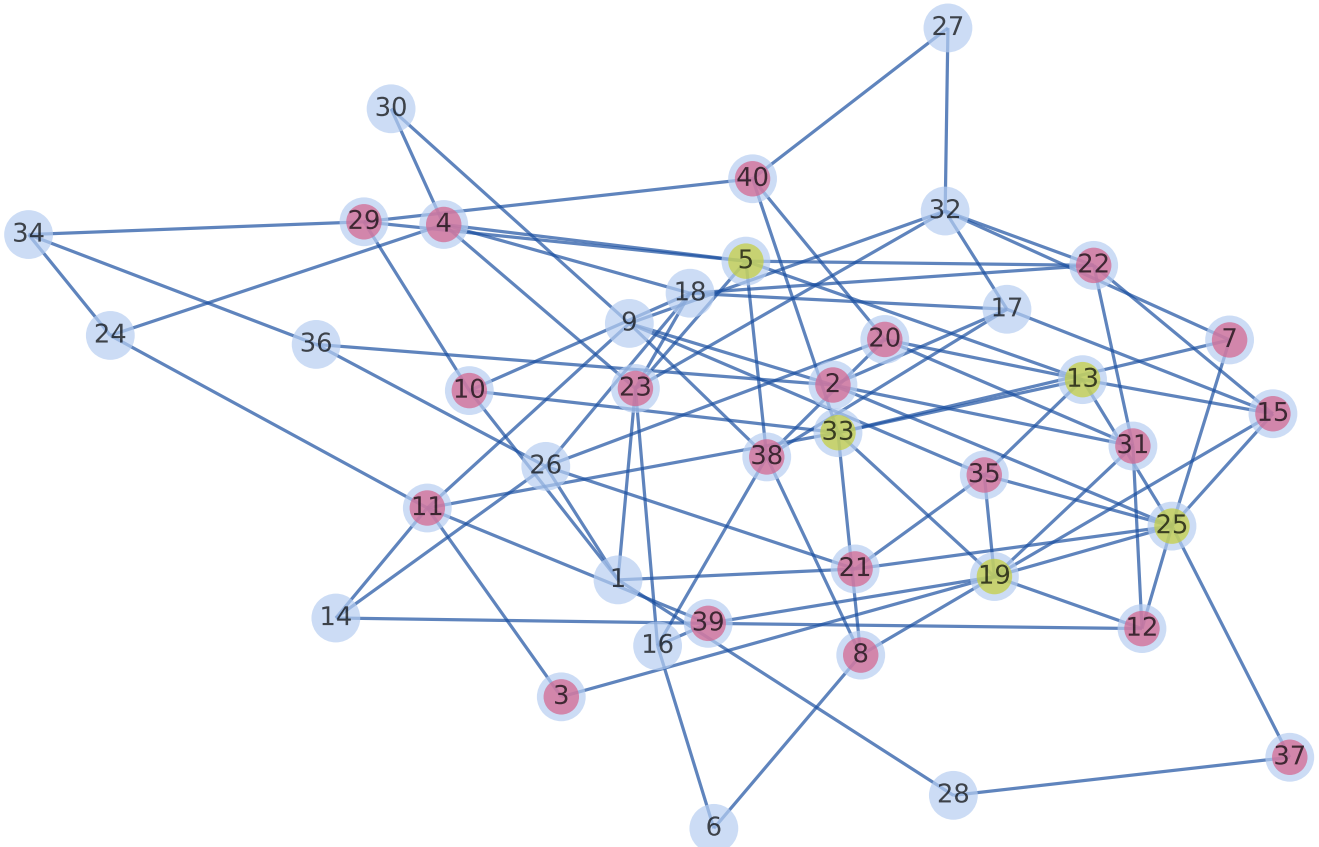
Subset size: 3
Algo step: 3



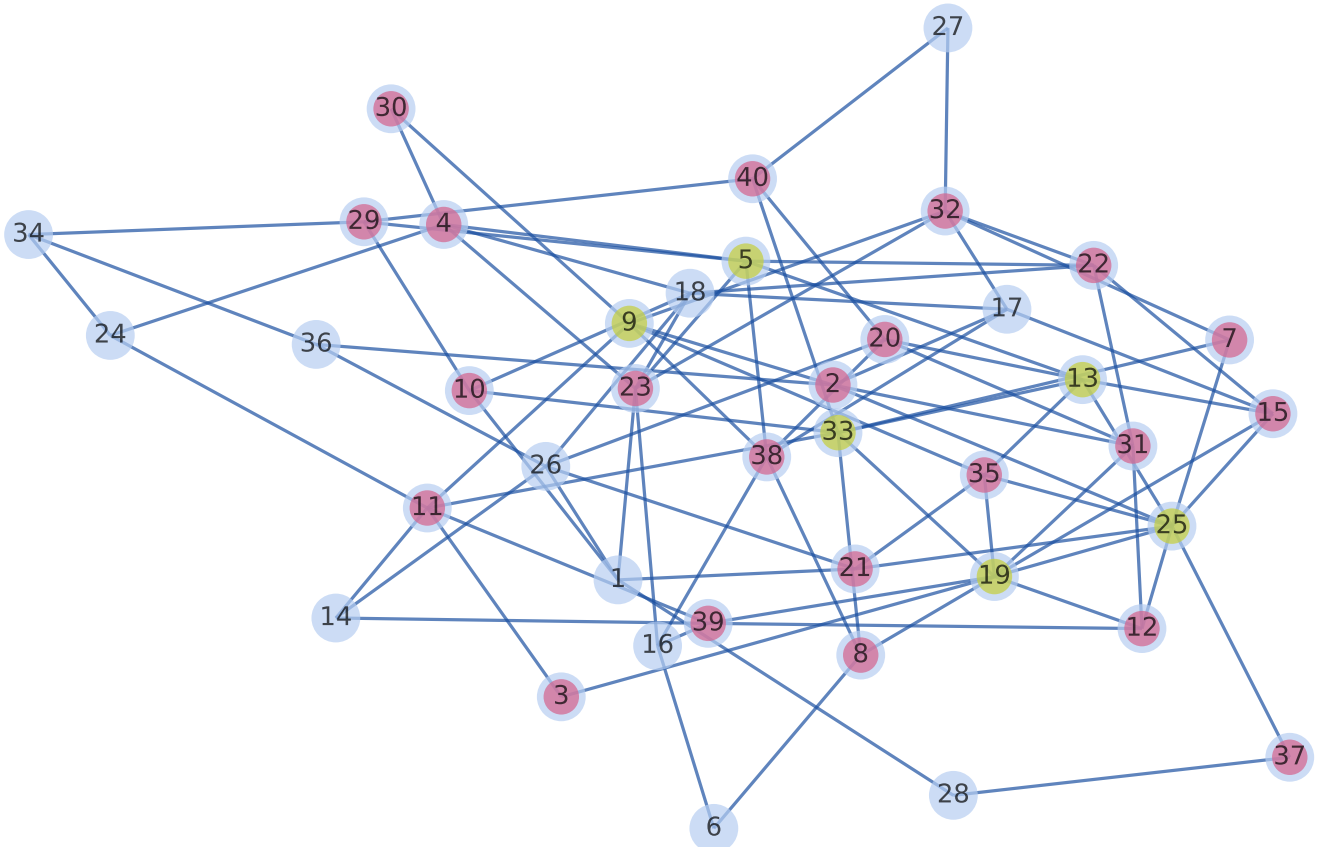
Subset size: 4
Algo step: 4



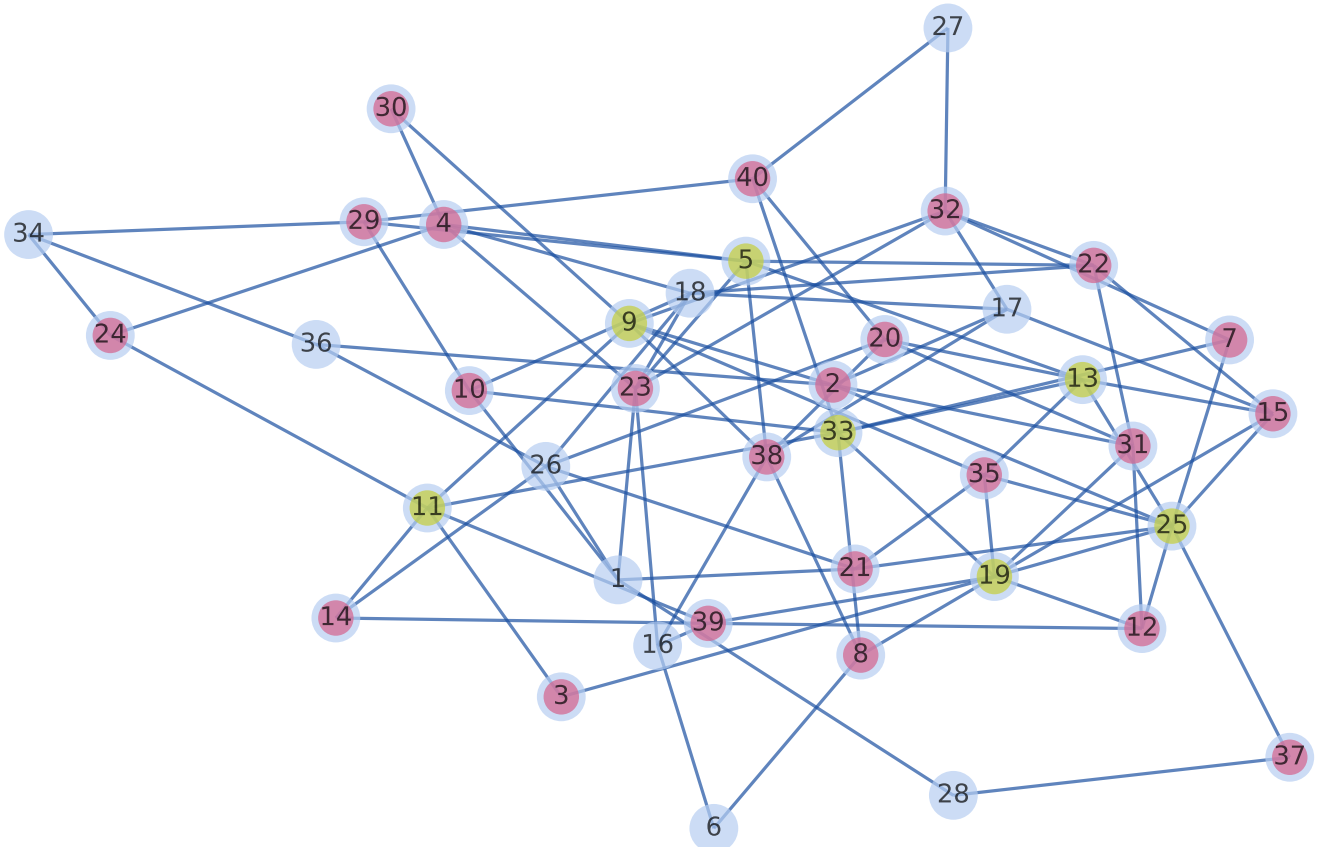
Subset size: 5
Algo step: 5



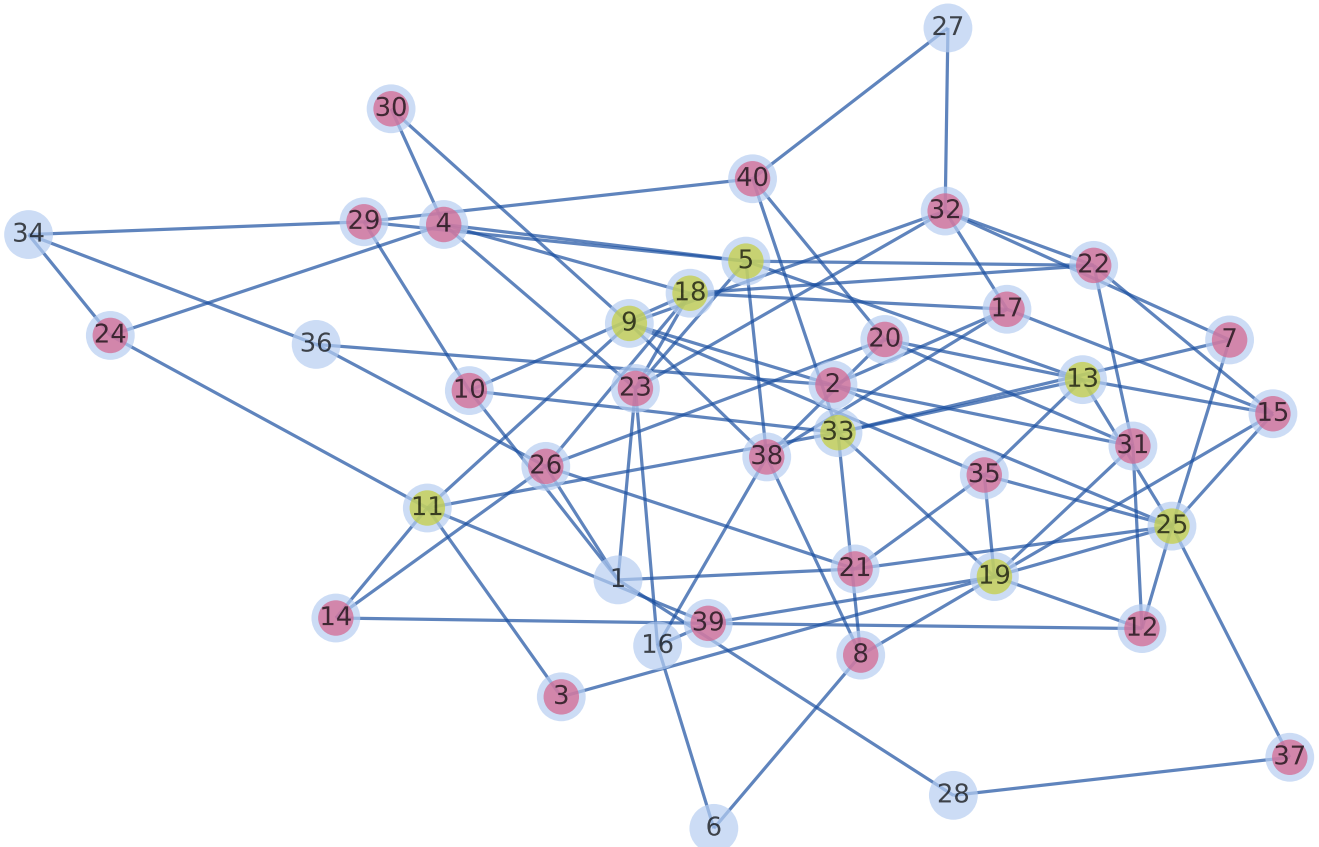
Subset size: 6
Algo step: 6



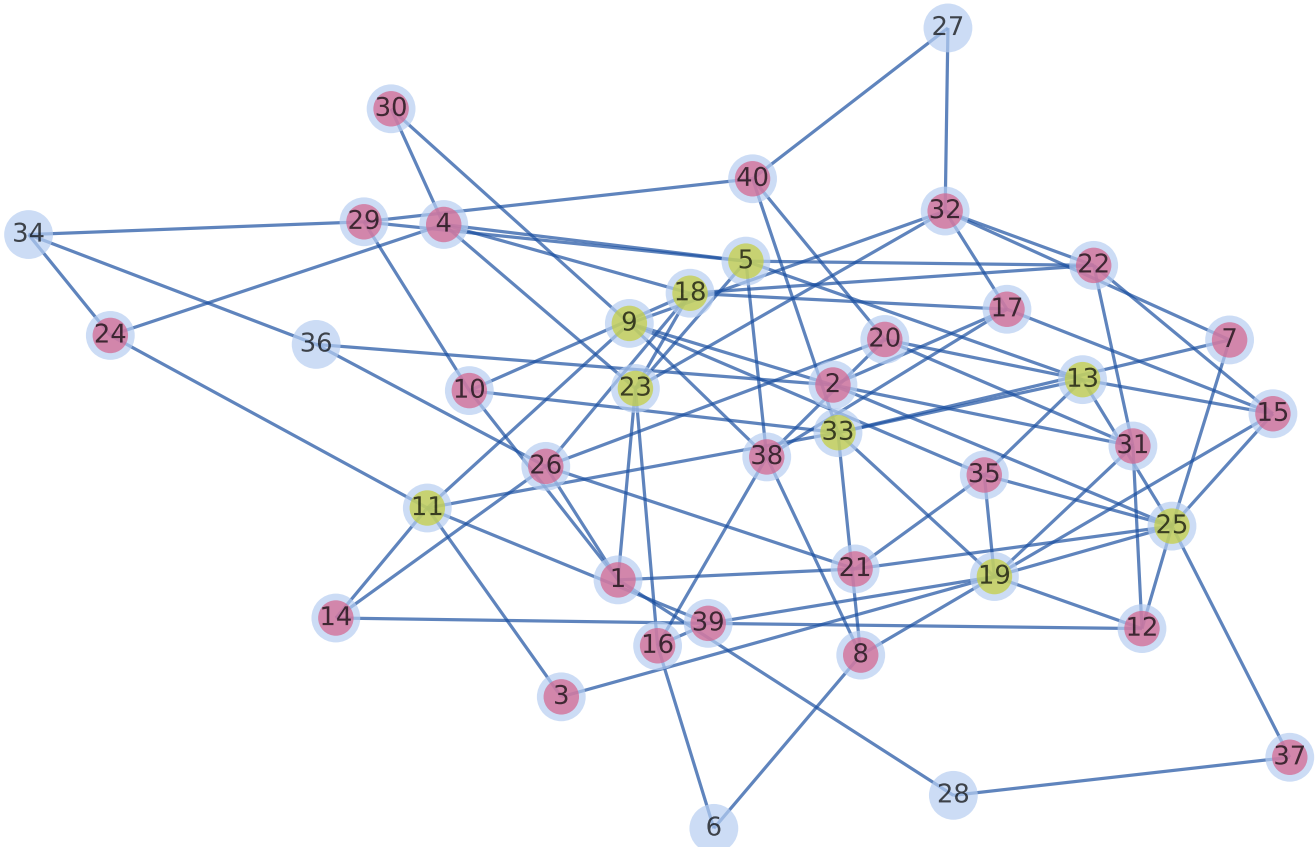
Subset size: 7
Algo step: 7



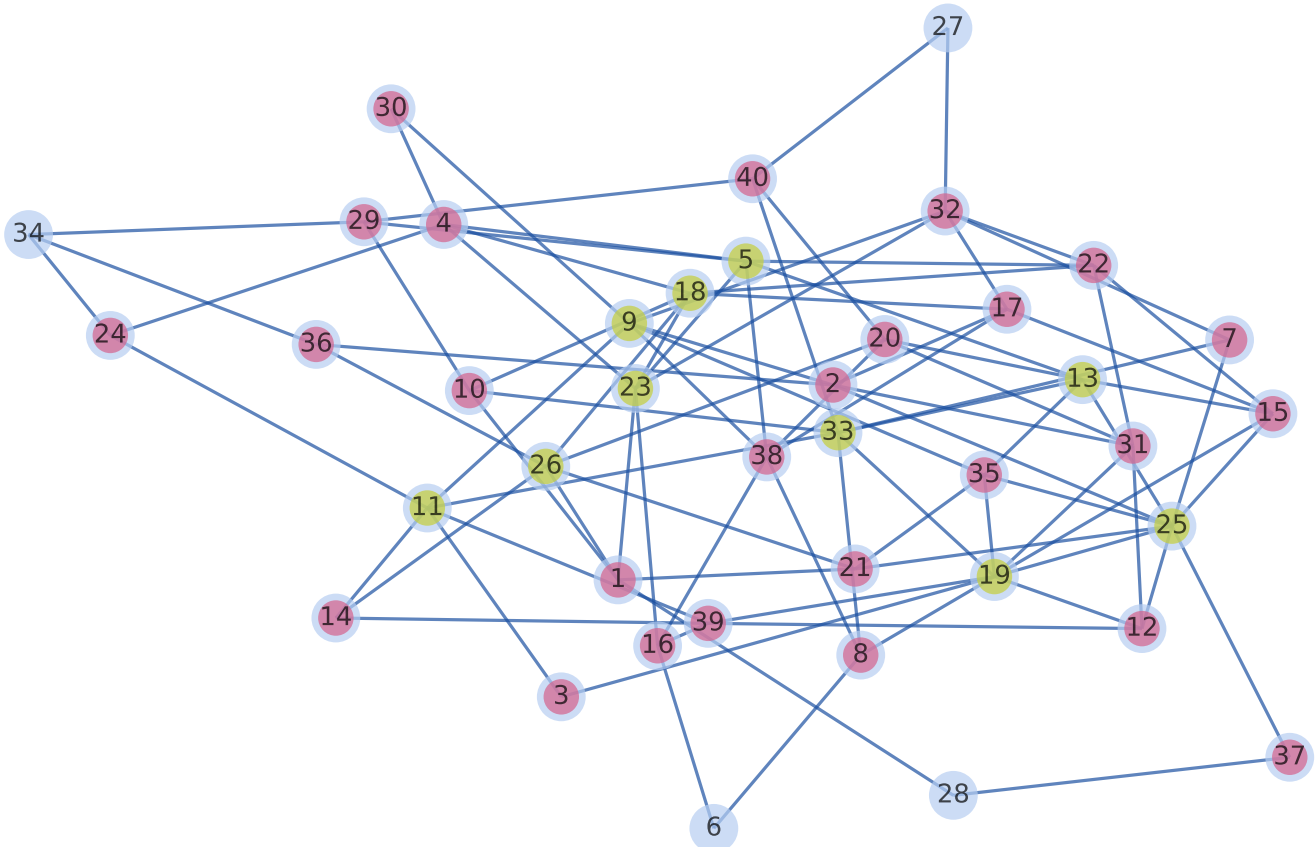
Subset size: 8
Algo step: 8



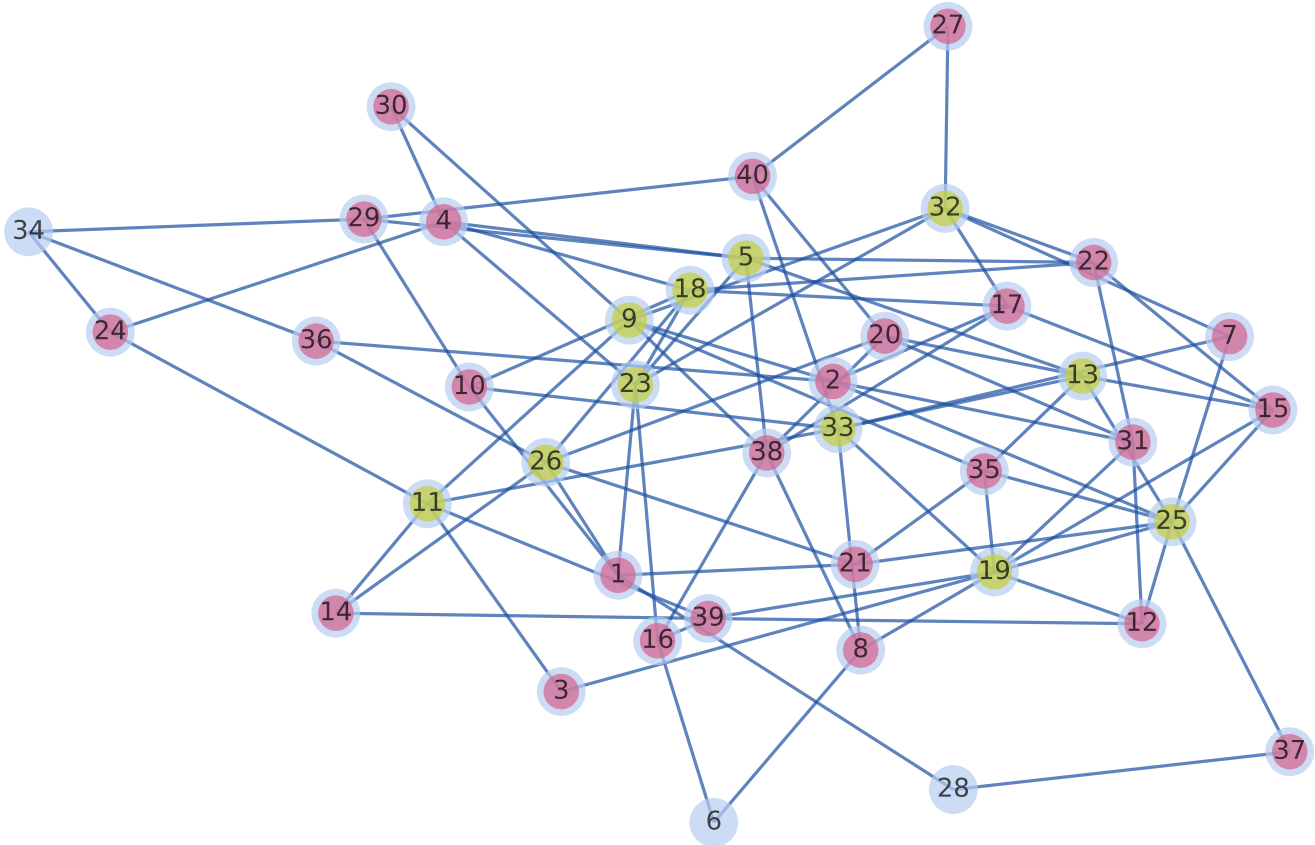
Subset size: 9
Algo step: 9



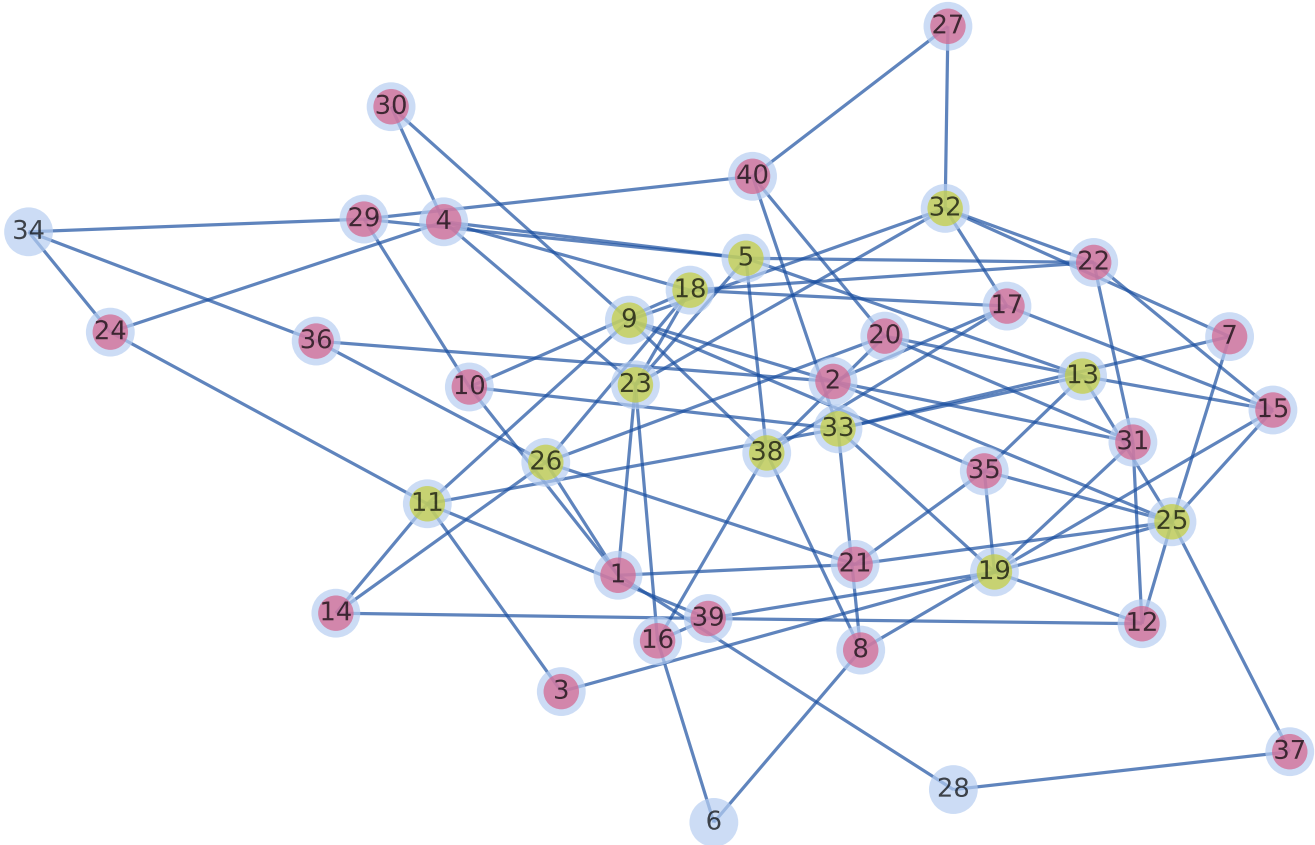
Subset size: 10
Algo step: 10



Subset size: 11
Algo step: 11



Subset size: 12
Algo step: 12

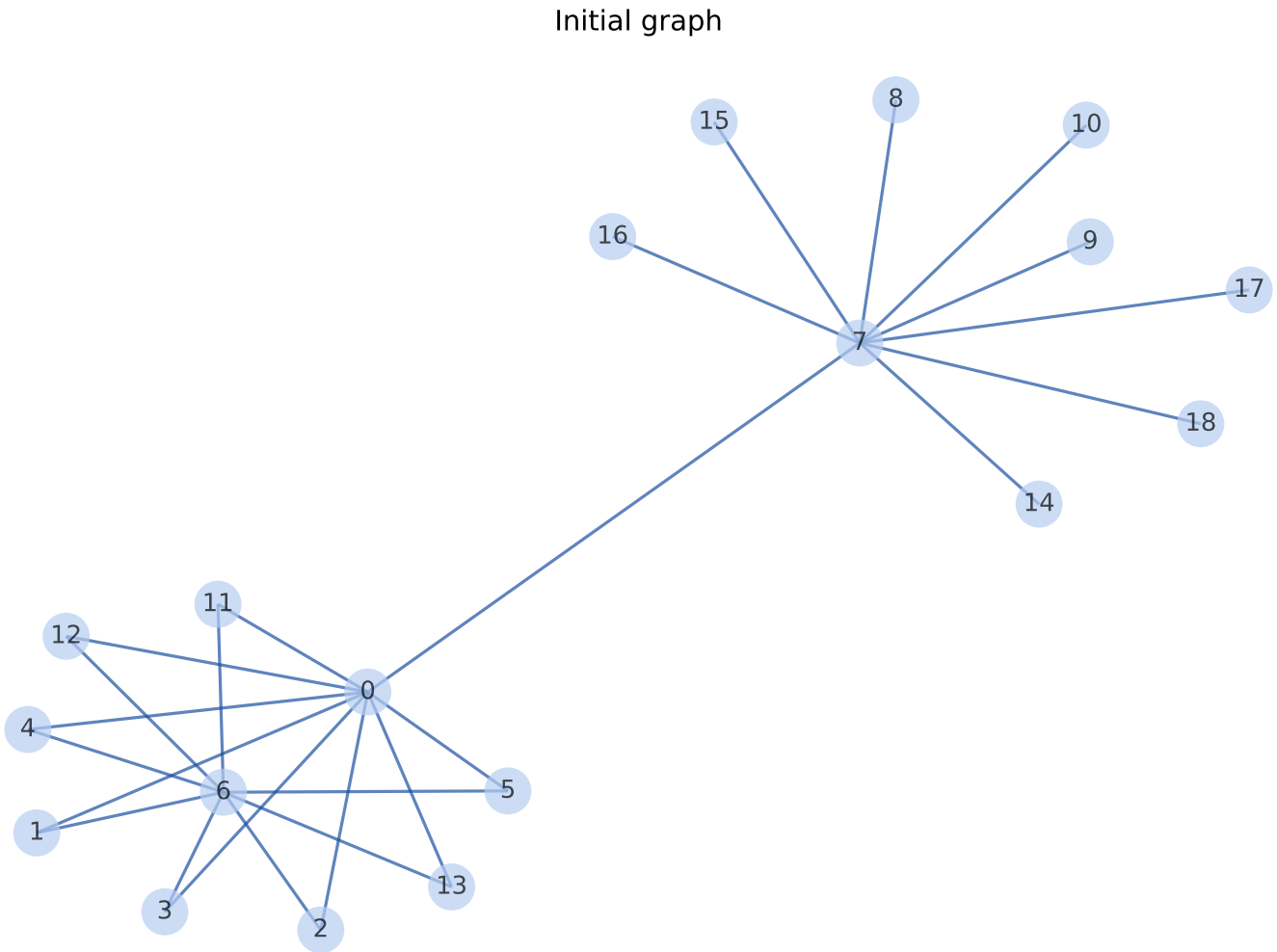


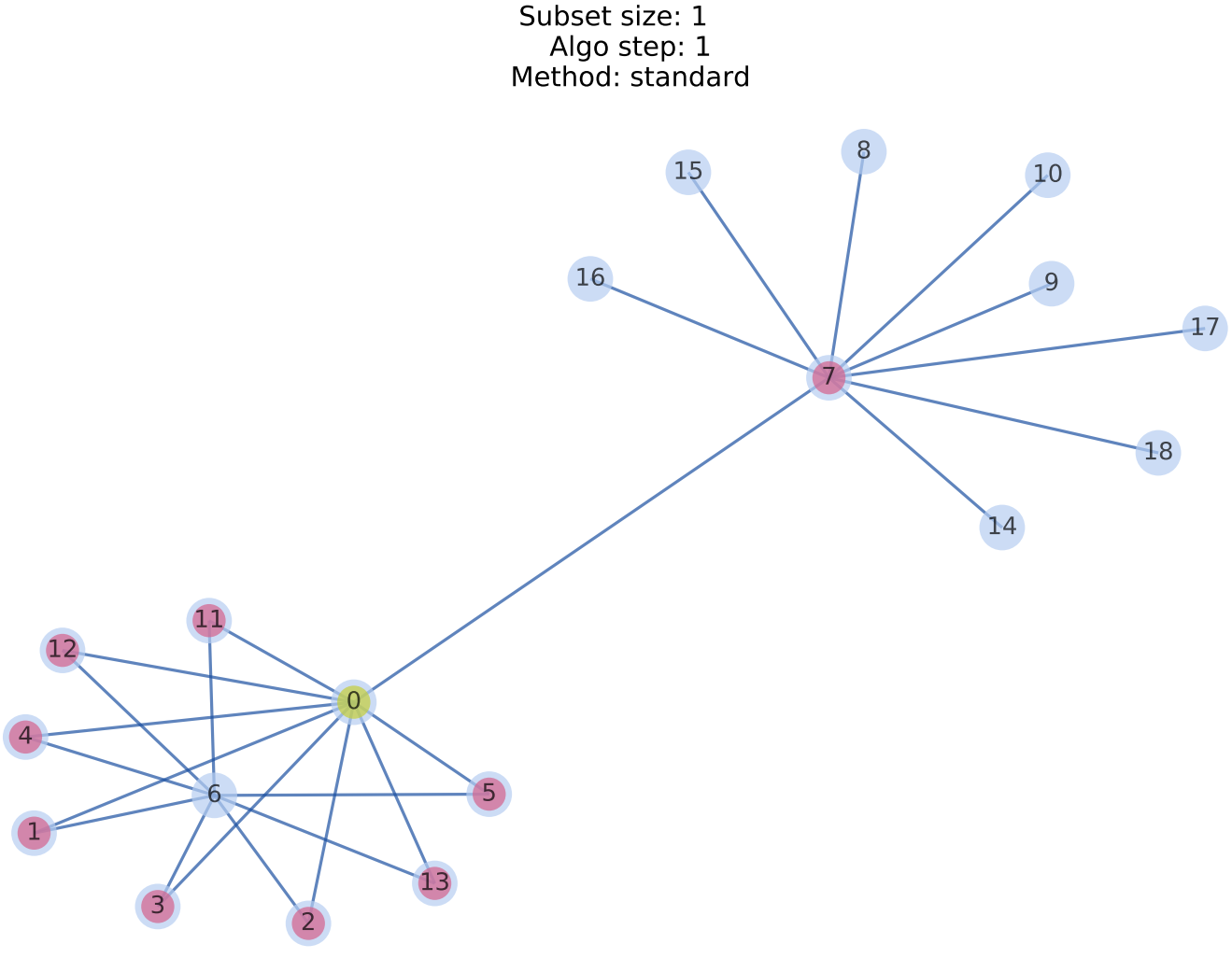
Variant 2

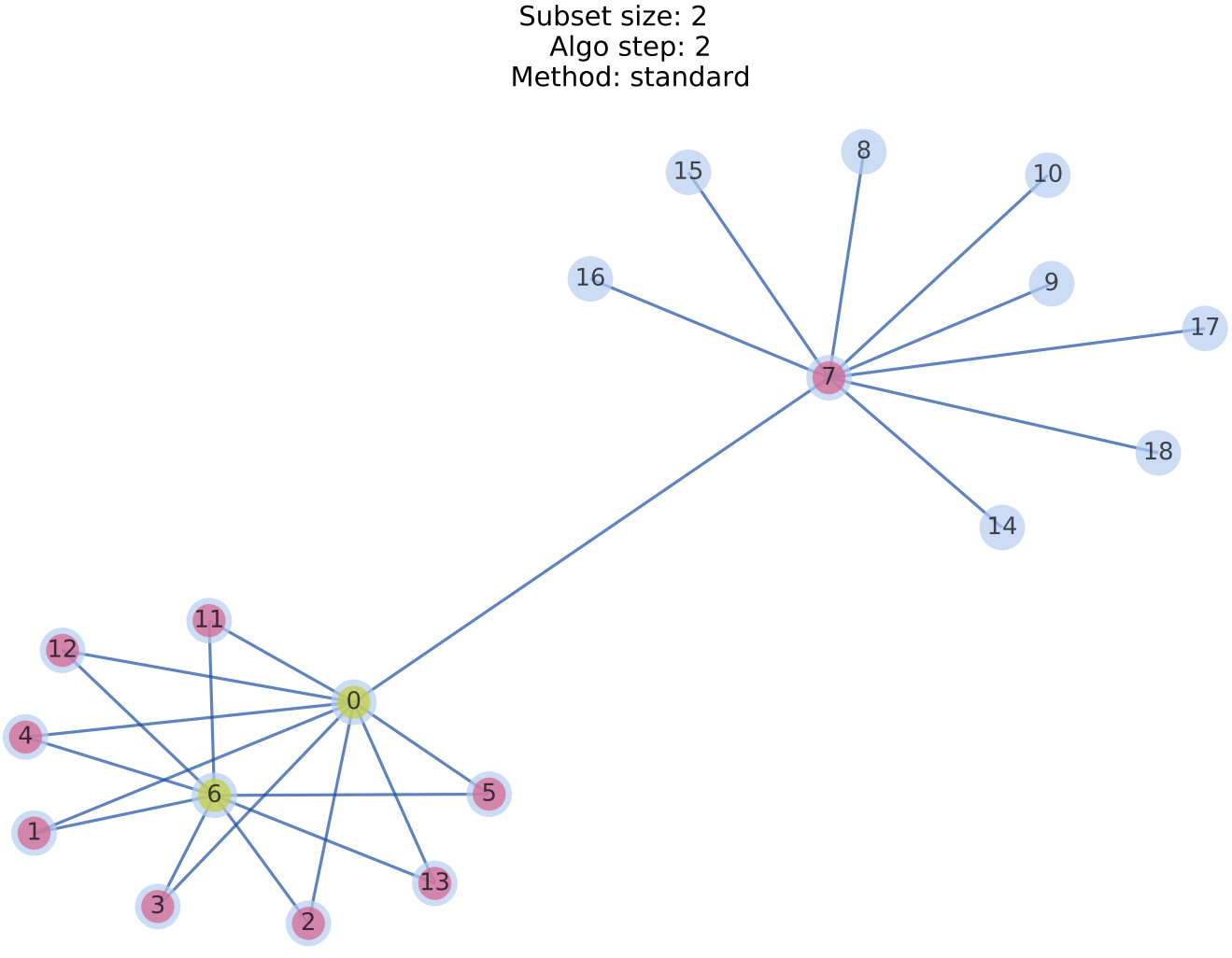
Exercice 7 : Implement of another variant where the degrees of the nodes are recomputed after each algorithm step.
You can use **dominating_set/greedy_ter.py**

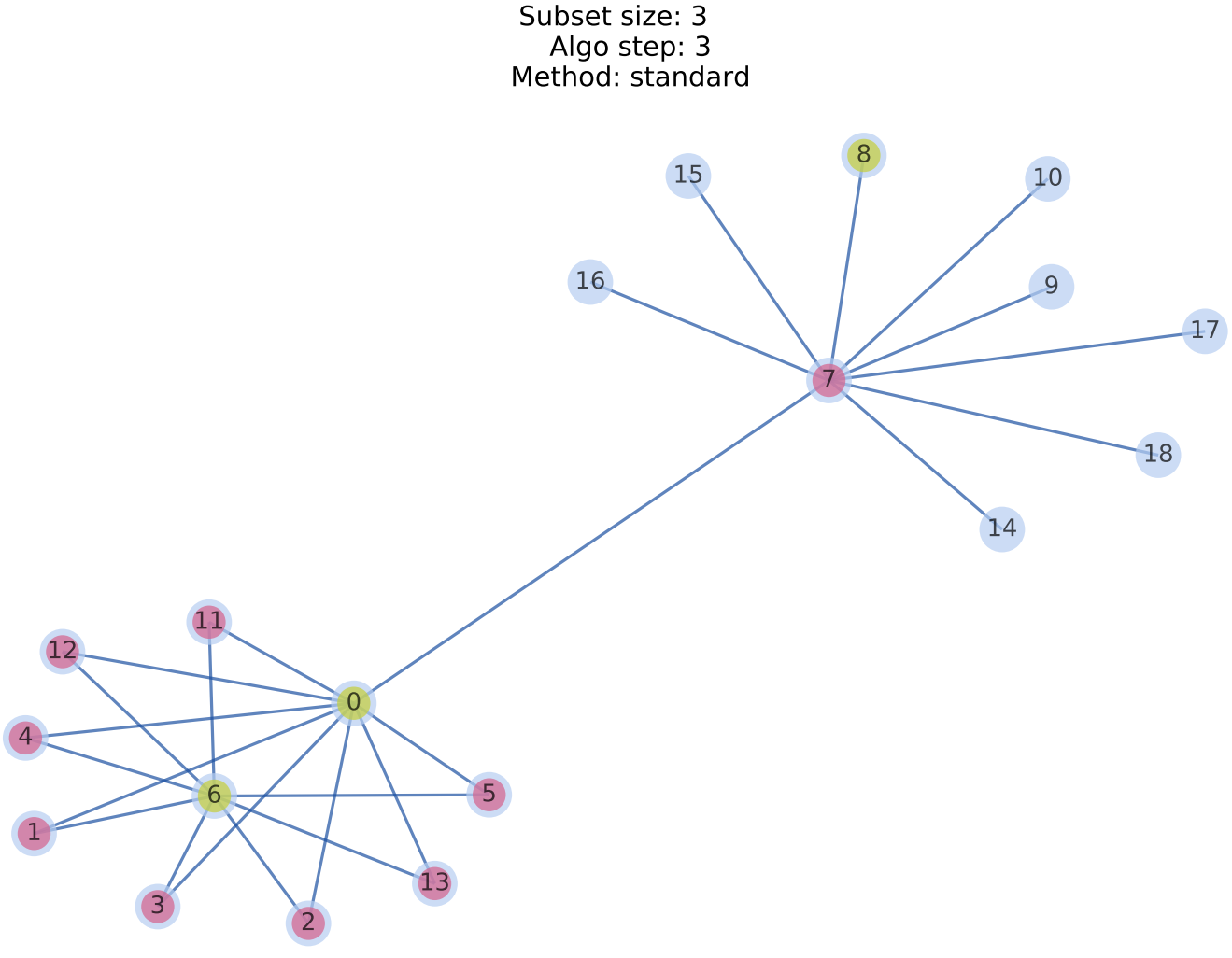
Different performances

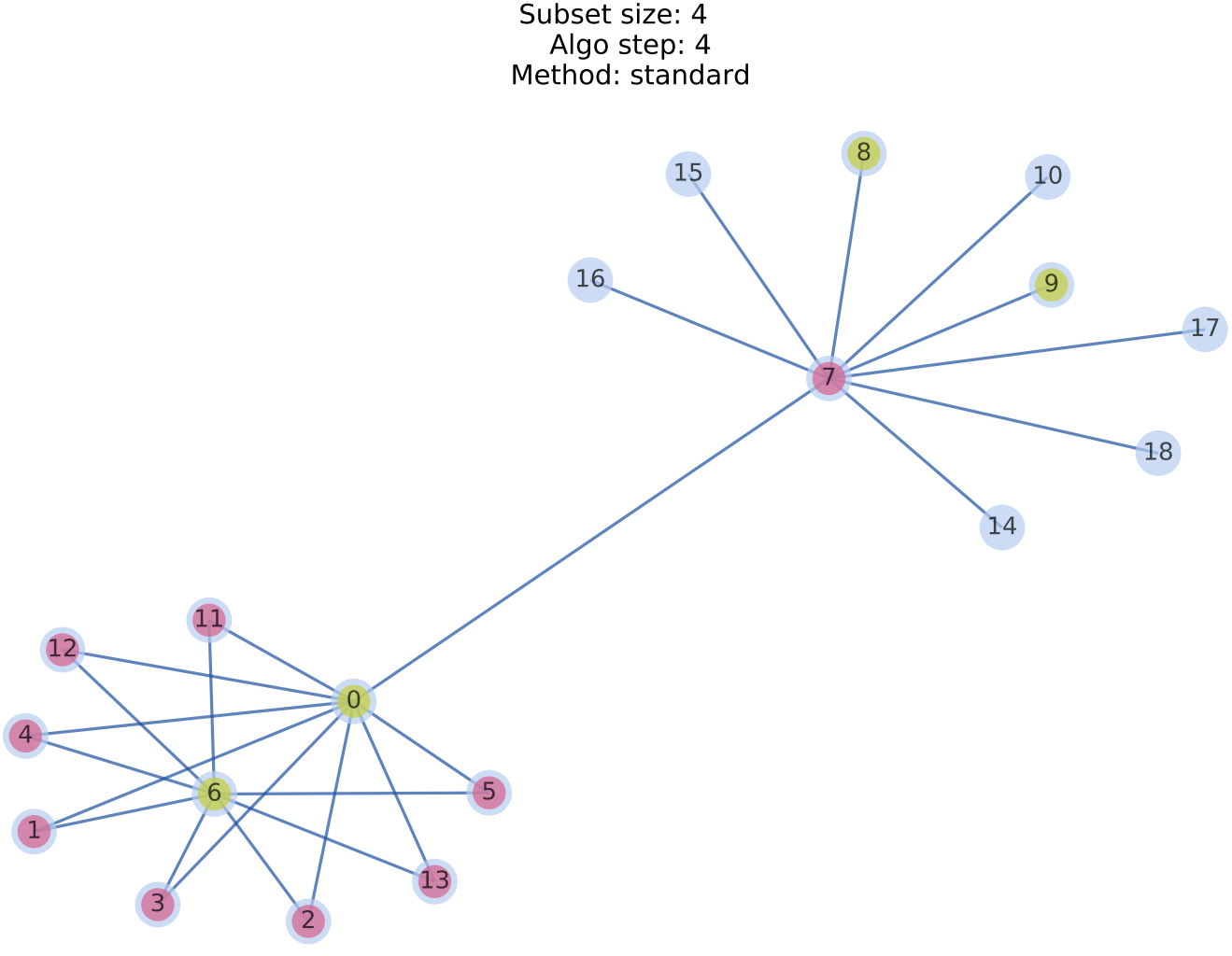
We have 3 variants of the algorithm, it seems that on most random cases "ter" works better (gives a smaller dominating set).
Exercice 8 : Can you find graph for which "standard" and "ter" are beaten by "bis" ?

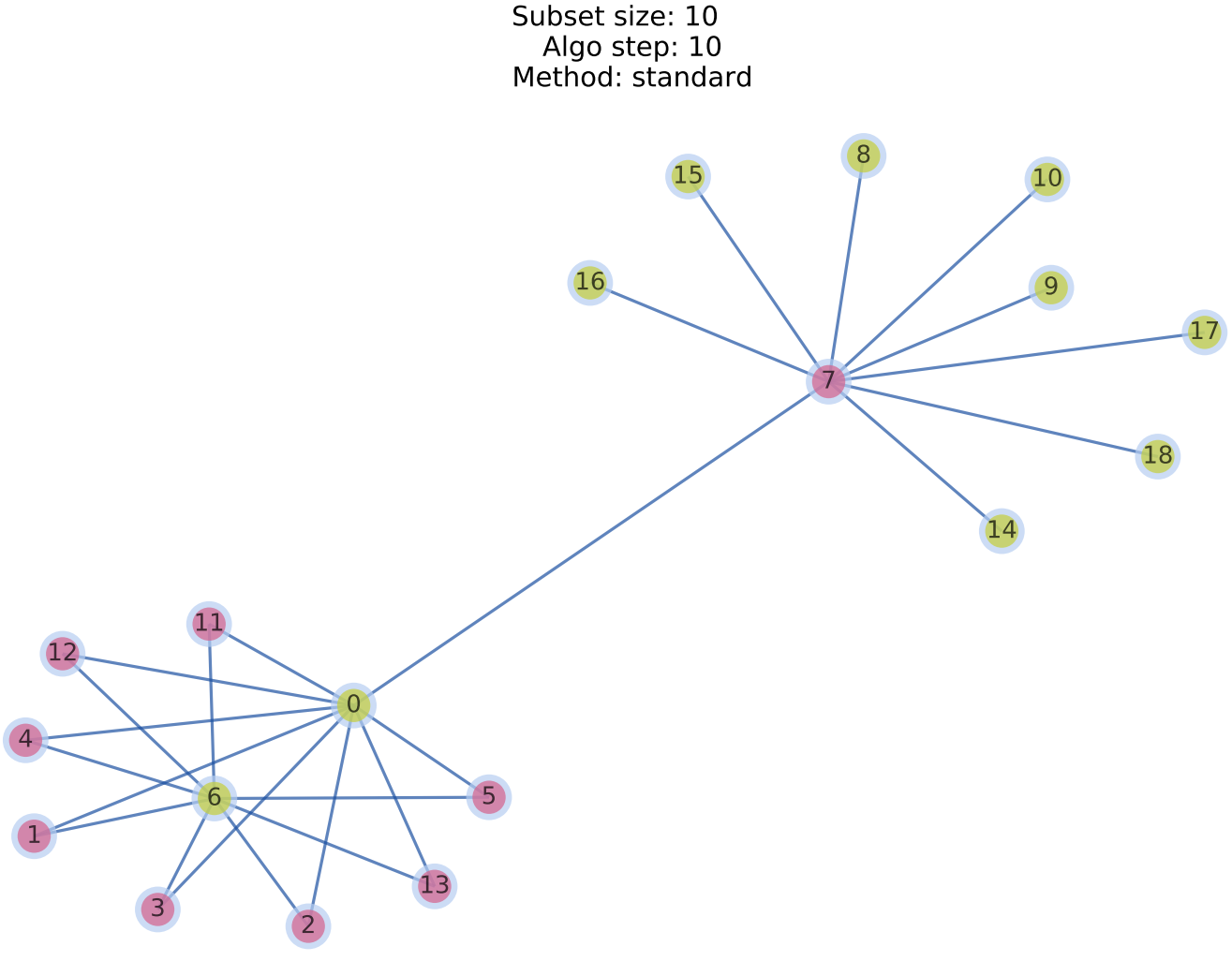

Initial graph
8
15
10
16
9
17
7
18
14
11
12
0
4
5
6
1
13
3
2

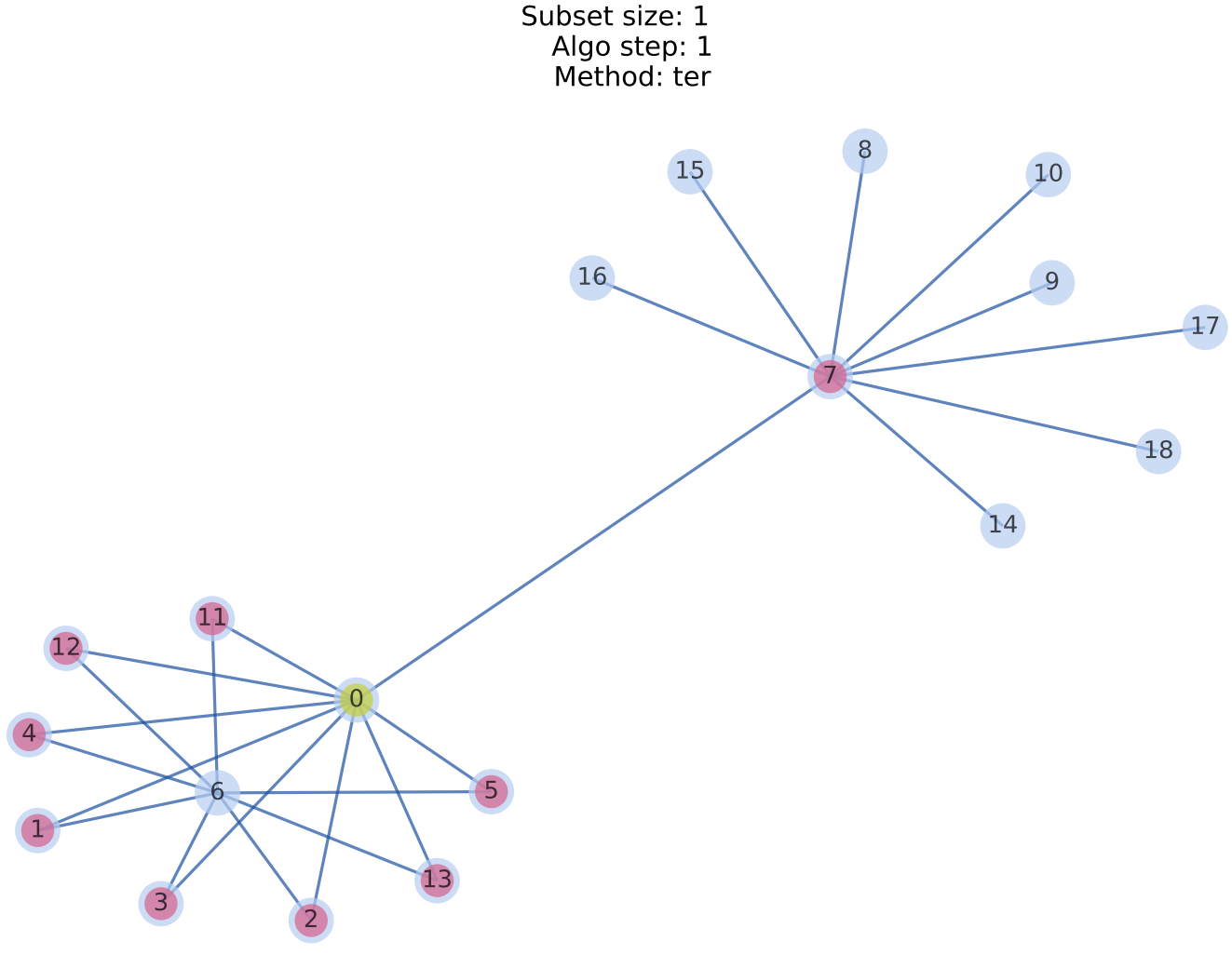

Subset size: 1
Algo step: 1
Method: standard
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2

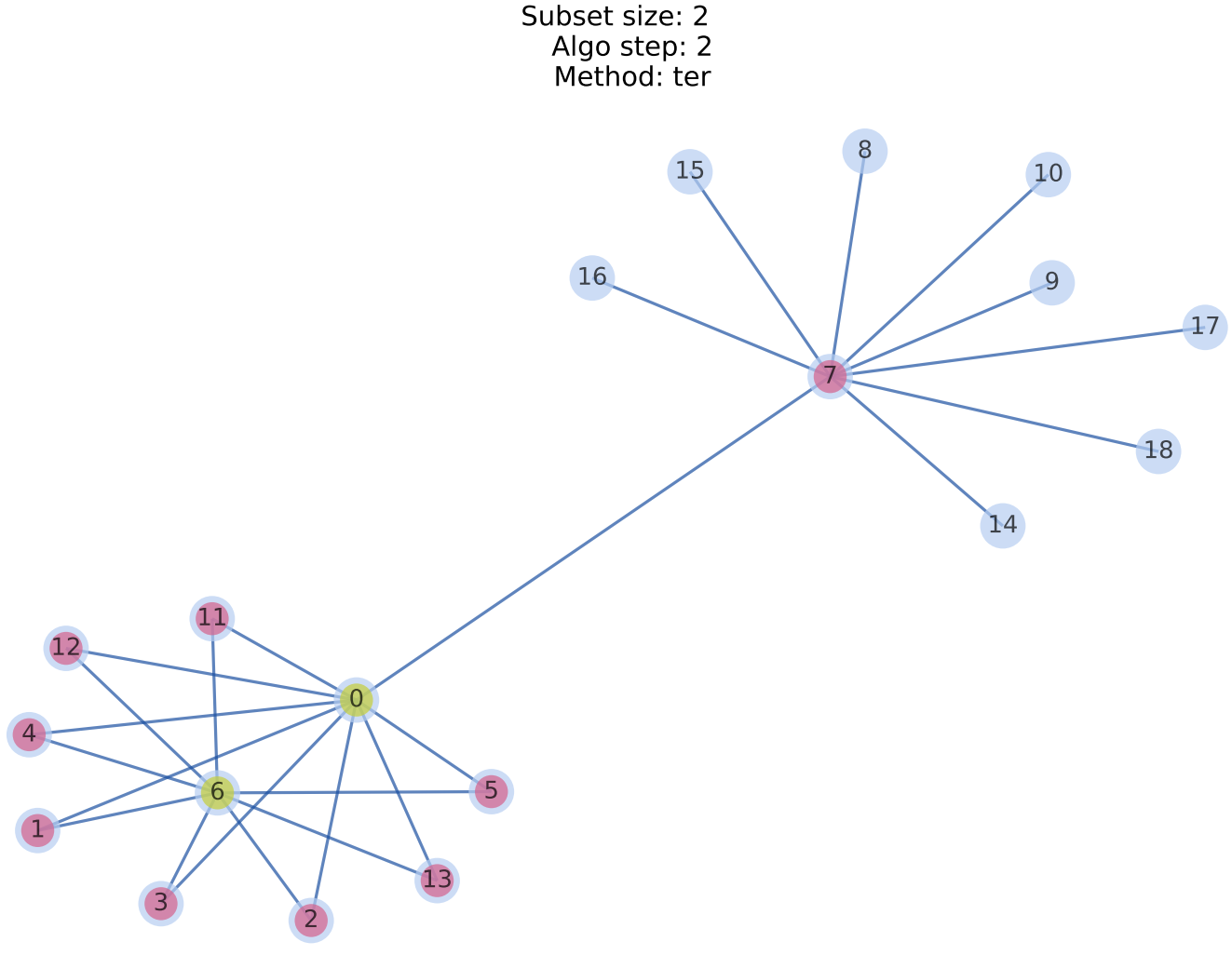

Subset size: 2
Algo step: 2
Method: standard
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2

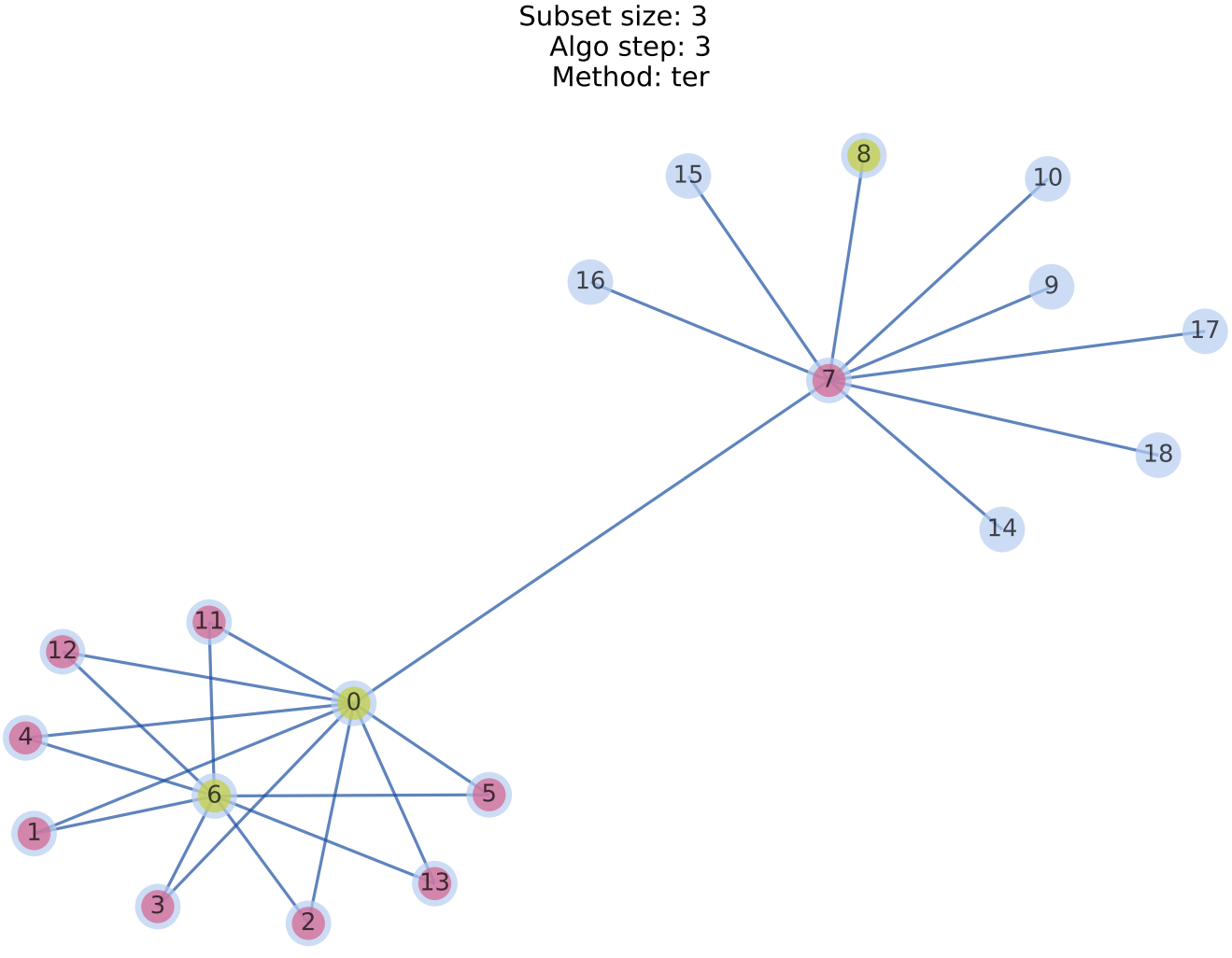

Subset size: 3
Algo step: 3
Method: standard
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2

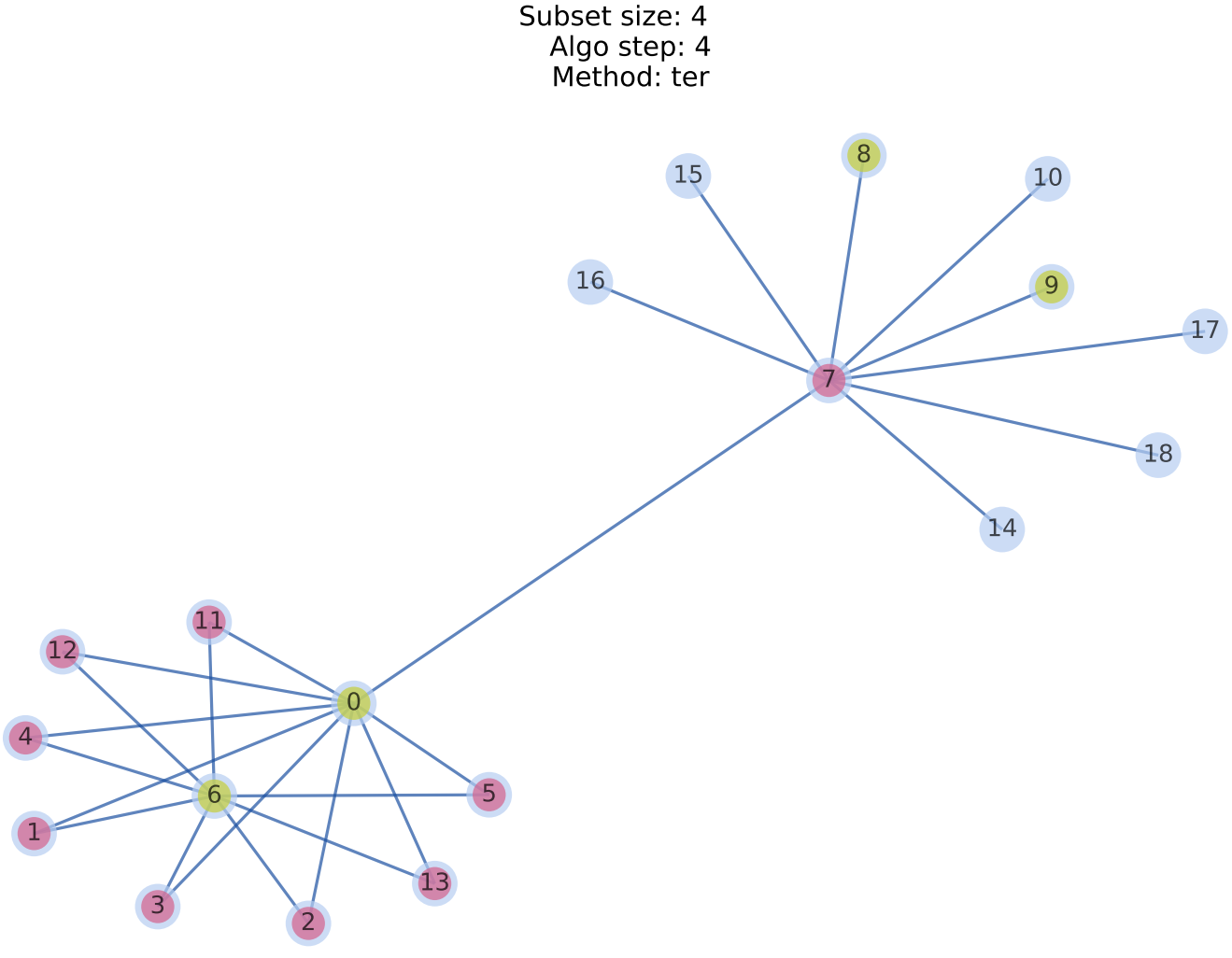

Subset size: 4
Algo step: 4
Method: standard
15
8
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2

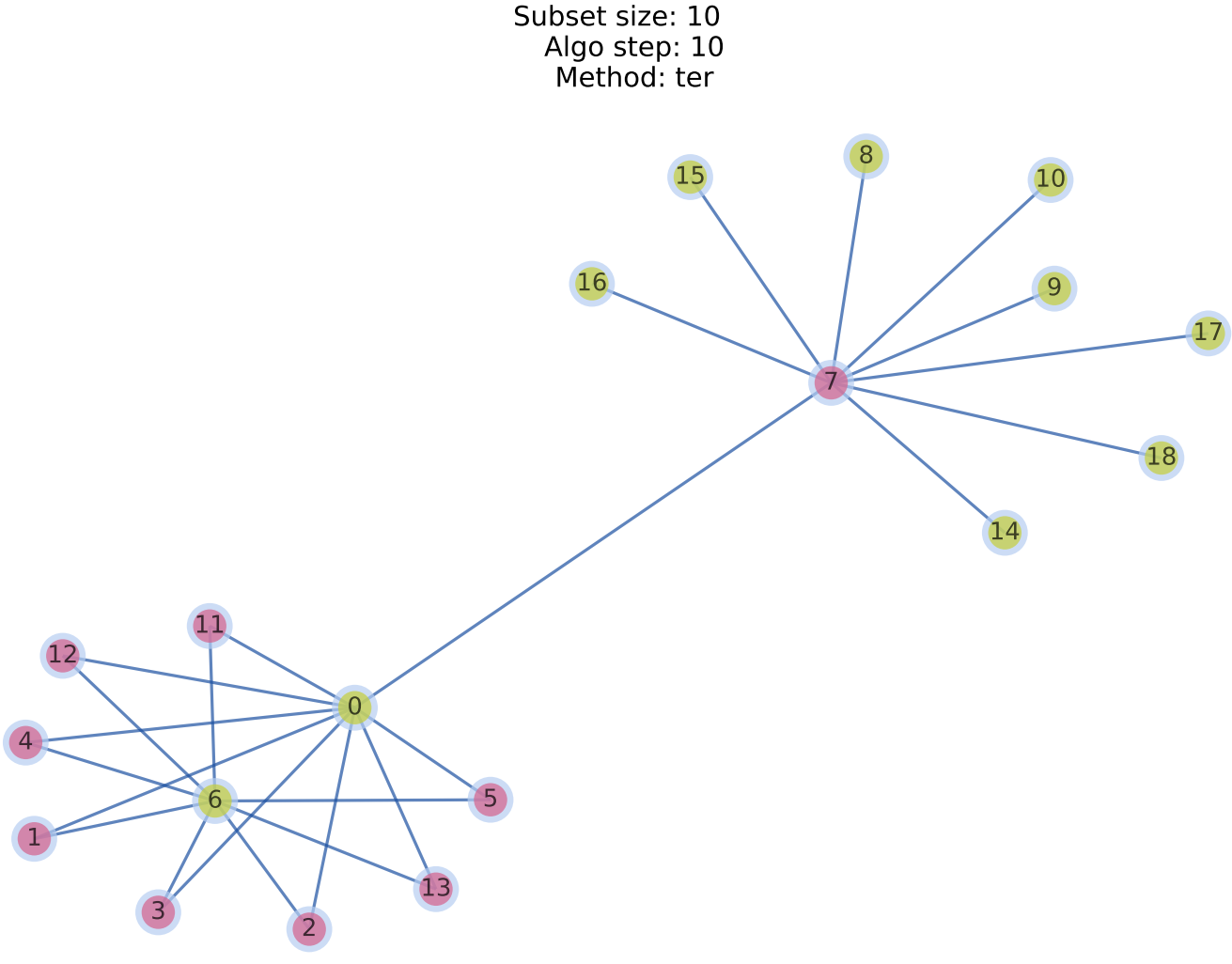

Subset size: 10
Algo step: 10
Method: standard
15
8
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2

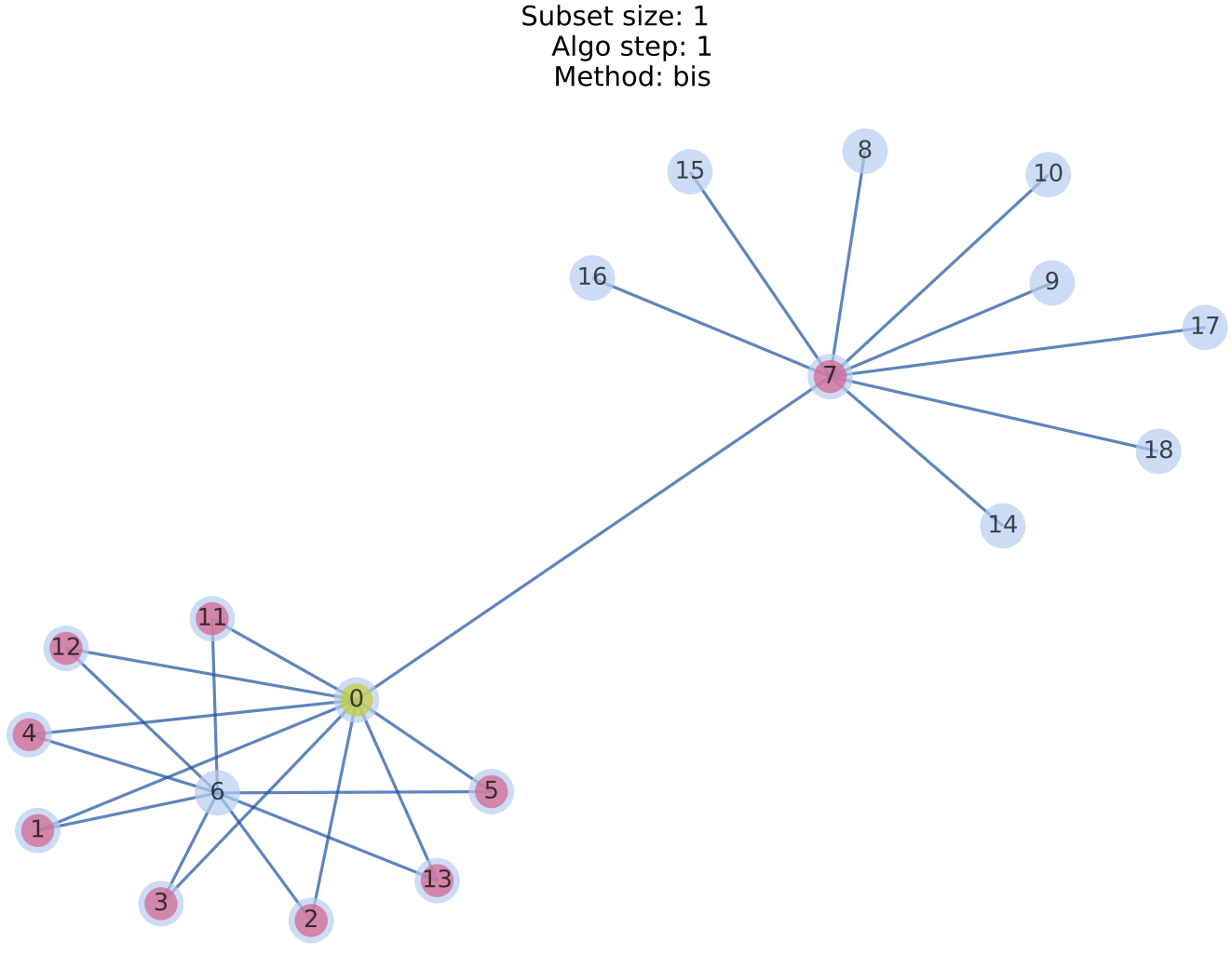

Subset size: 1
Algo step: 1
Method: ter
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2

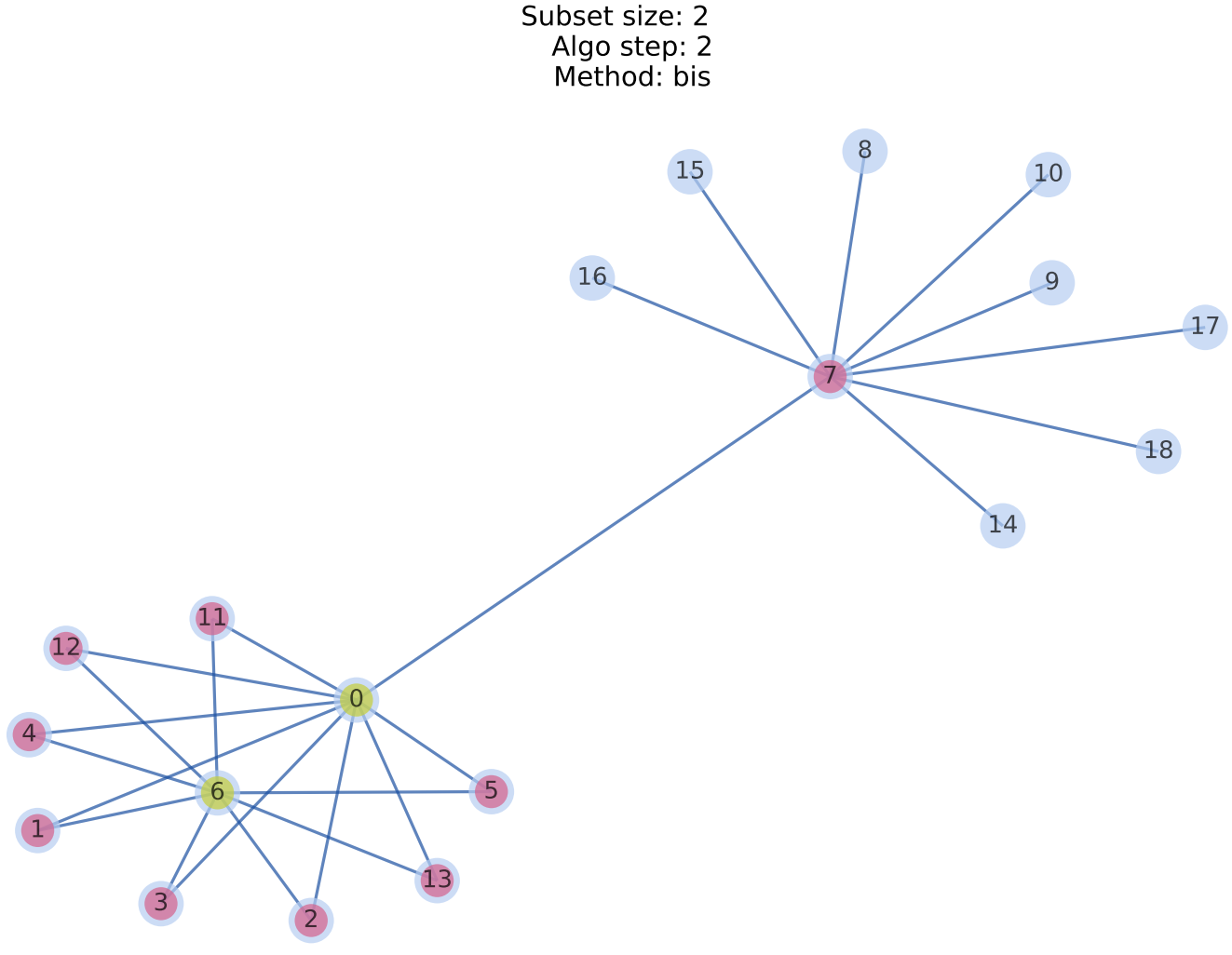

Subset size: 2
Algo step: 2
Method: ter
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2

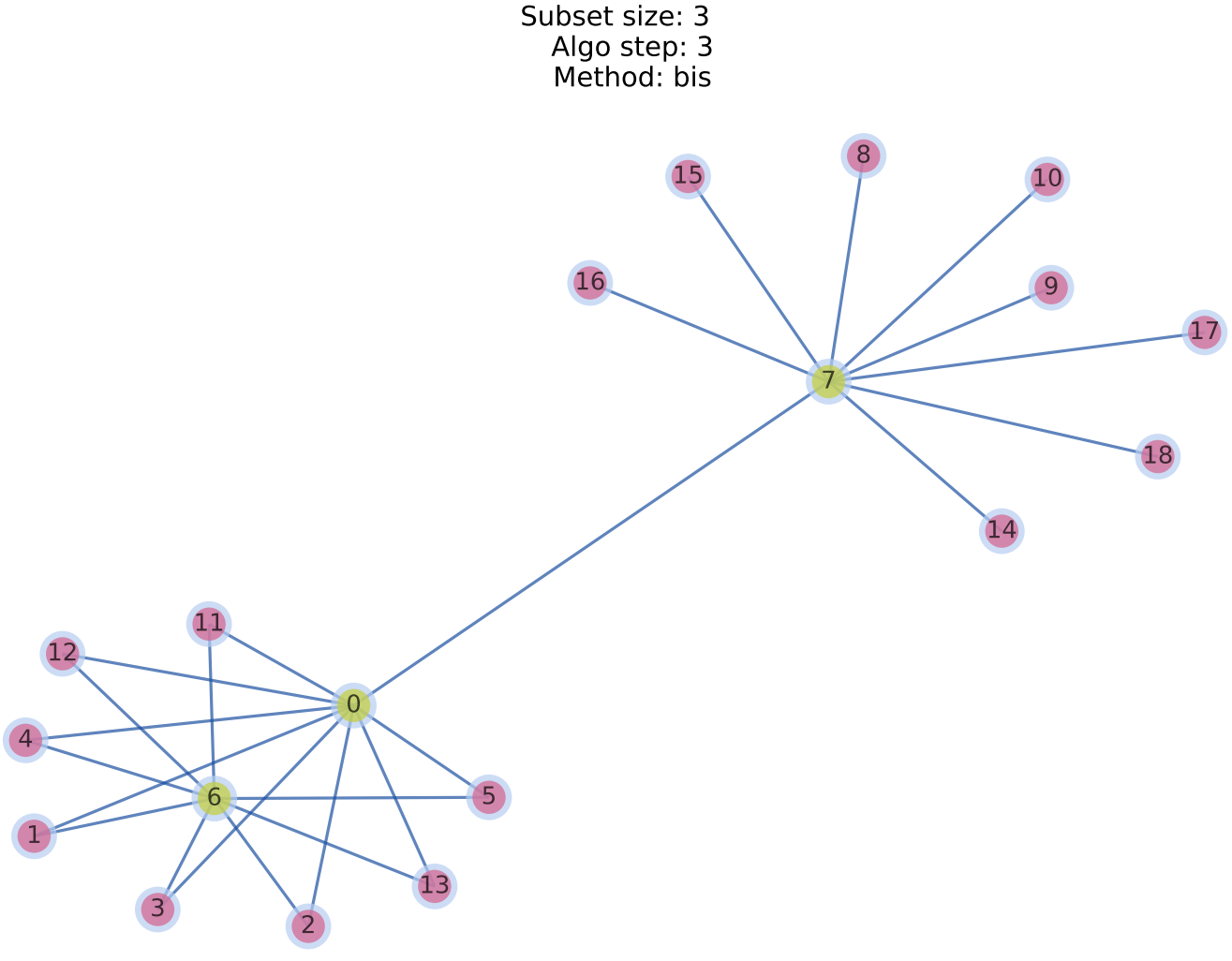

Subset size: 3
Algo step: 3
Method: ter
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2


Subset size: 4
Algo step: 4
Method: ter
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2


Subset size: 10
Algo step: 10
Method: ter
15
8
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2


Subset size: 1
Algo step: 1
Method: bis
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2


Subset size: 2
Algo step: 2
Method: bis
8
15
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2


Subset size: 3
Algo step: 3
Method: bis
15
8
10
16
9
17
7
18
14
11
12
0
4
6
5
1
13
3
2

Non optimal greedy algorithm

Exercice 9 : Find a graph for which "standard" gives a very bad solution.

Non optimal greedy algorithm

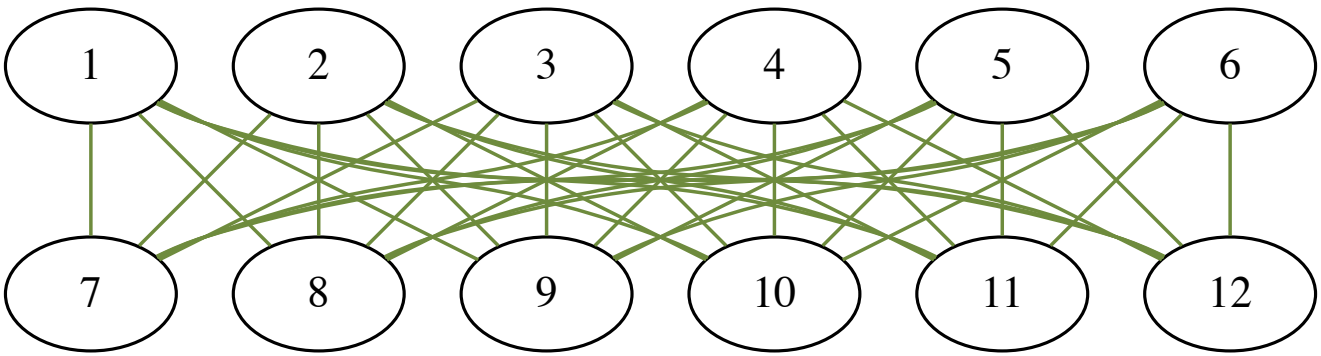


Figure – Complete bipartie graph

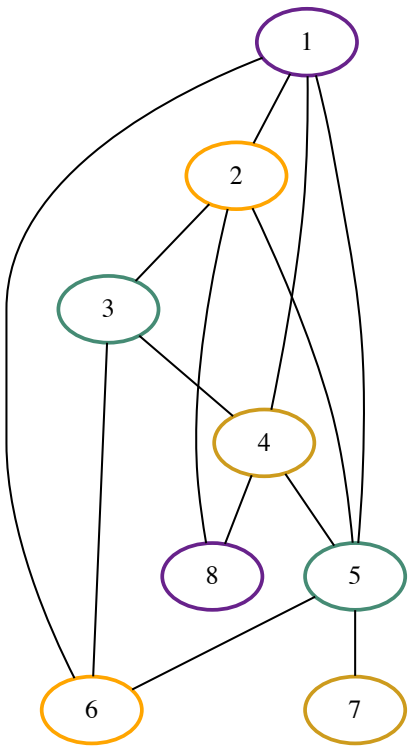
Networkx

- The library networkx has some functions that implement many graph algorithms.
- `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.dominating.dominating_set.html`
- See **dominating_nx.py**.

The coloring problem

Say you have a map with different countries. You need to assign a color to each country, so that two countries that have a common border are filled with a different color. We assume that we would like to use a small number of colors (the smaller, the better).
Exercice 10 : How could we formalize this problem with a graph ?

The coloring problem

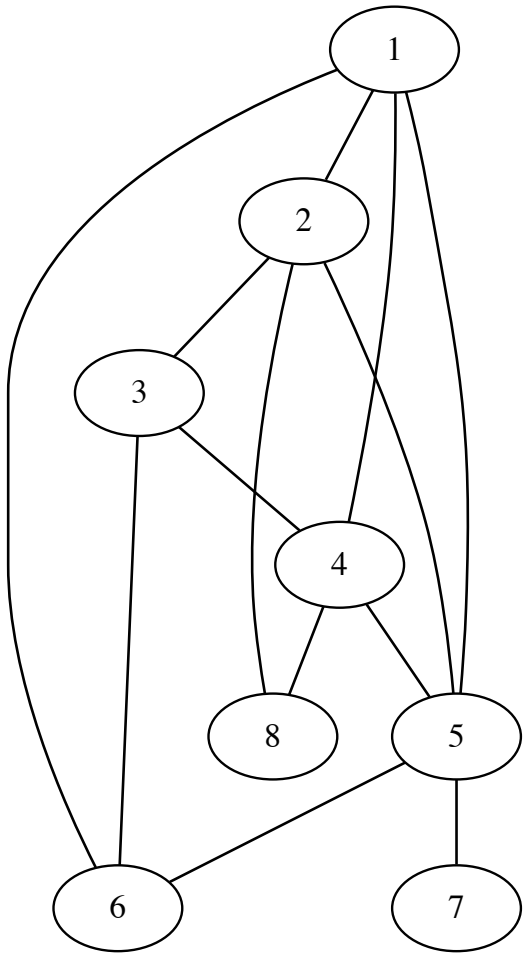


We want to find the smallest number of **fully disconnected subgraphs** in a graph.

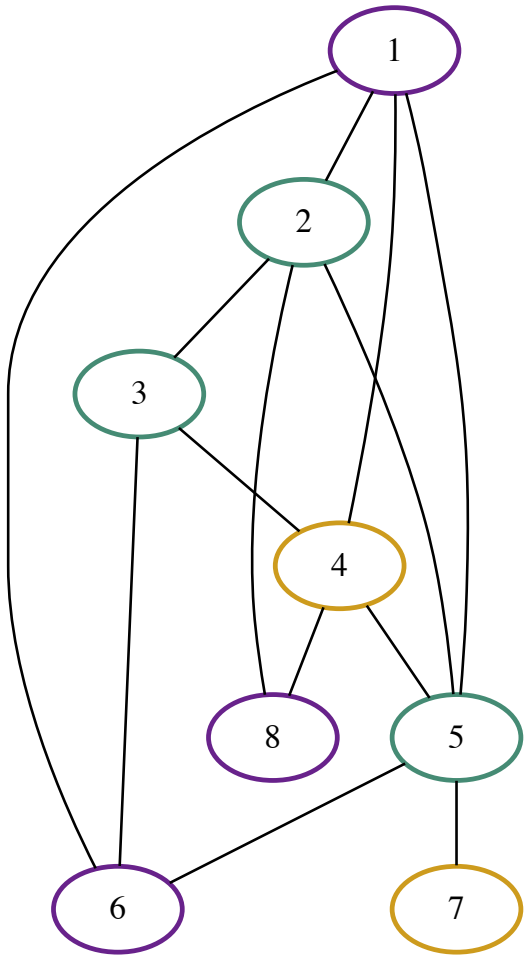
The coloring problem

We want to find the smallest number of **fully disconnected subgraph** in a graph.
Each subgraph will be associated with a color.

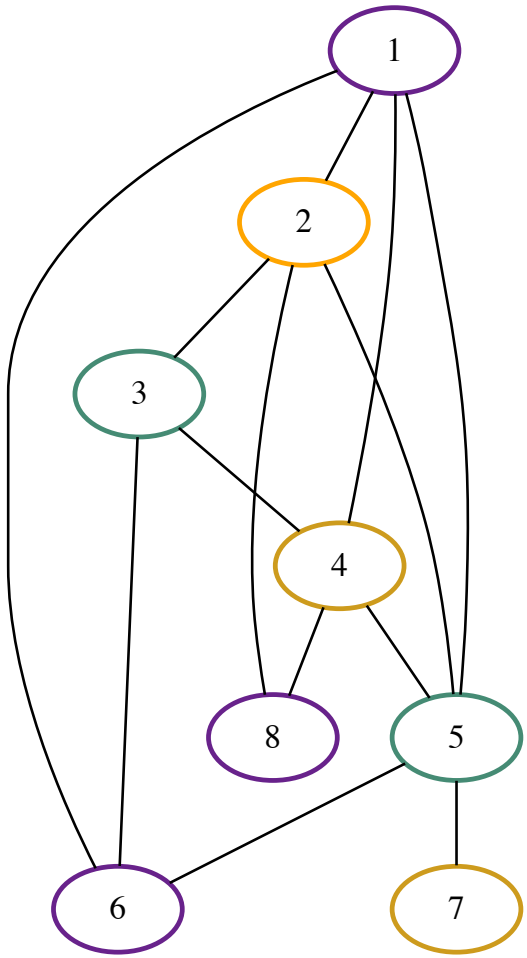
Coloring



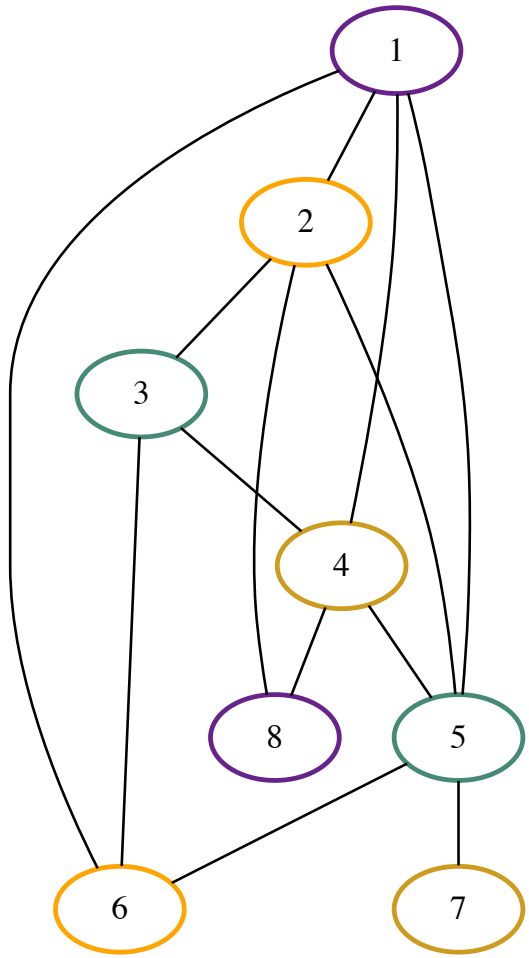
Is this a coloring ?



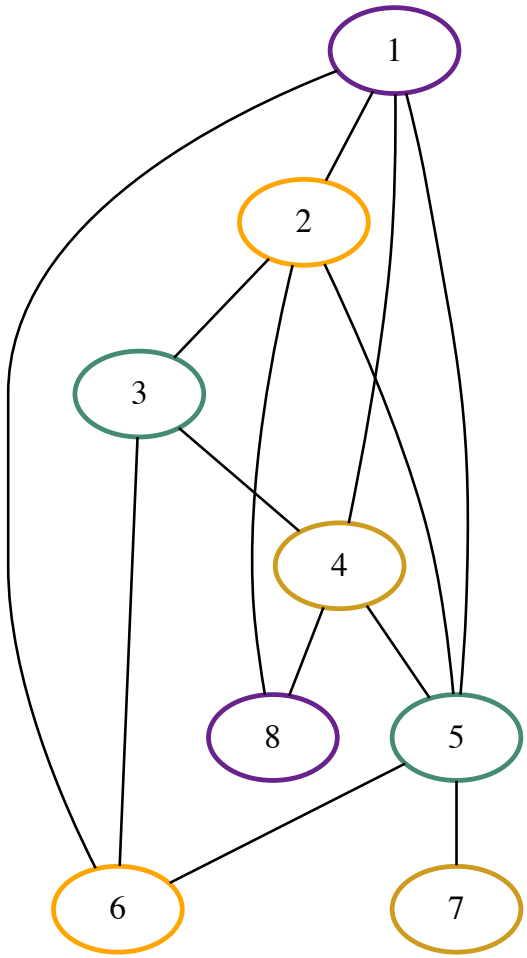
Is this a coloring ?



Is this a coloring ? yes



Could we have used only 3 colors ?



Coloring

- What would be a trivial coloring ?

Coloring

- What would be a trivial coloring ? assign a color to each node (very bad solution)
- Could you think of a heuristic ?

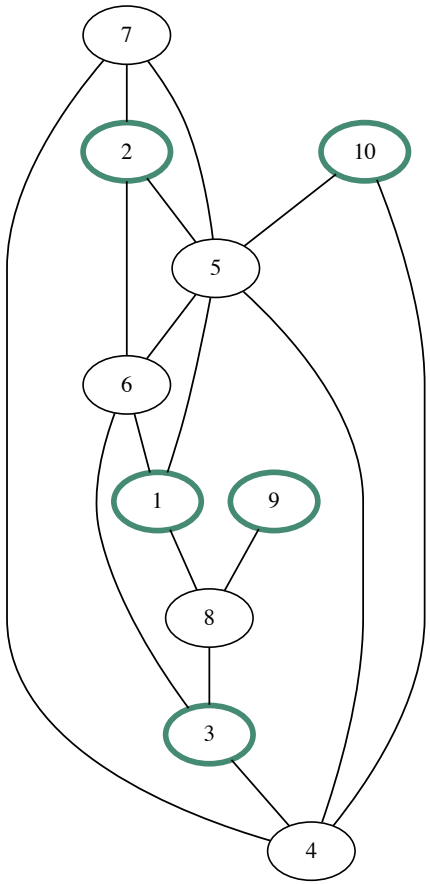
Other applications

- Planning activities (color : time in the day)
- Assigning frequencies (color : frequency)

Independent set

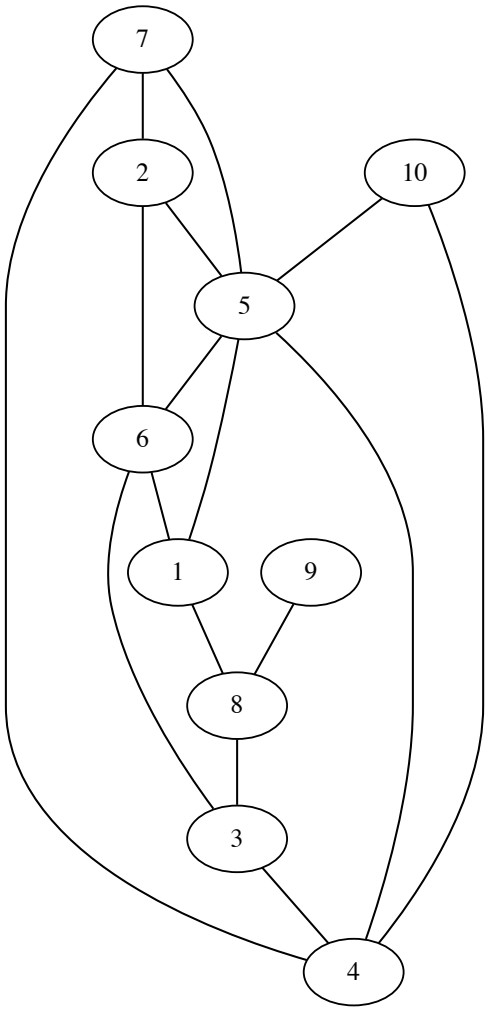
You have a group of people. Some people cannot work with each other. You want to build to largest possible team of people.
Exercice 11 : How could we formalize this with a graph ?

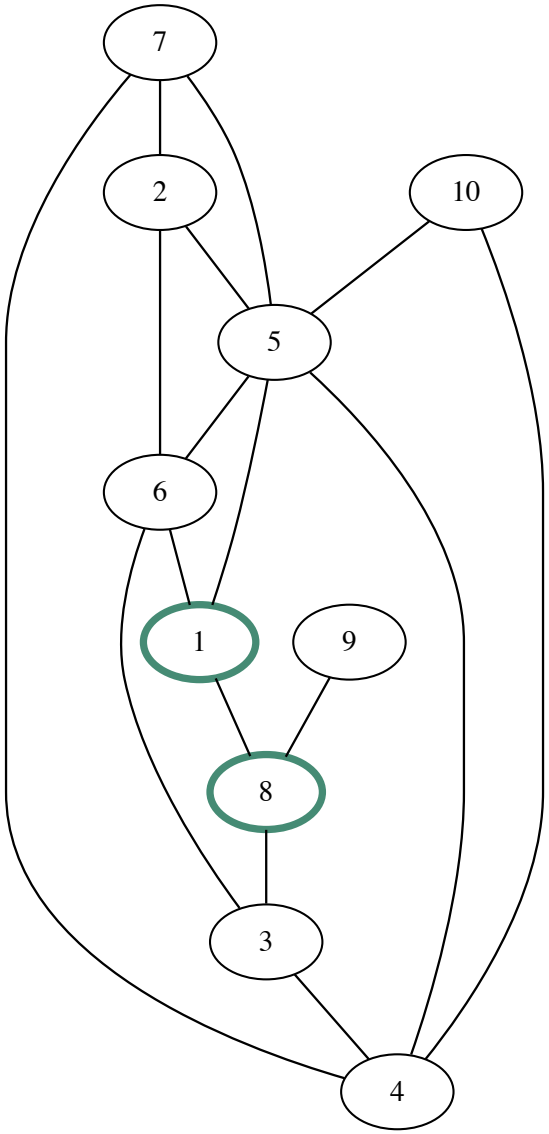
Independent Set

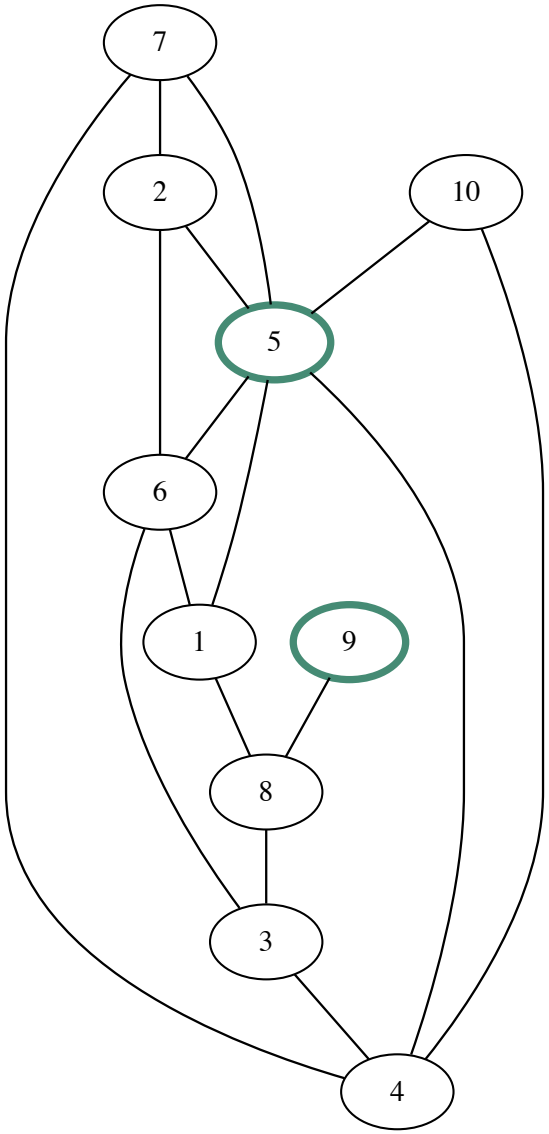


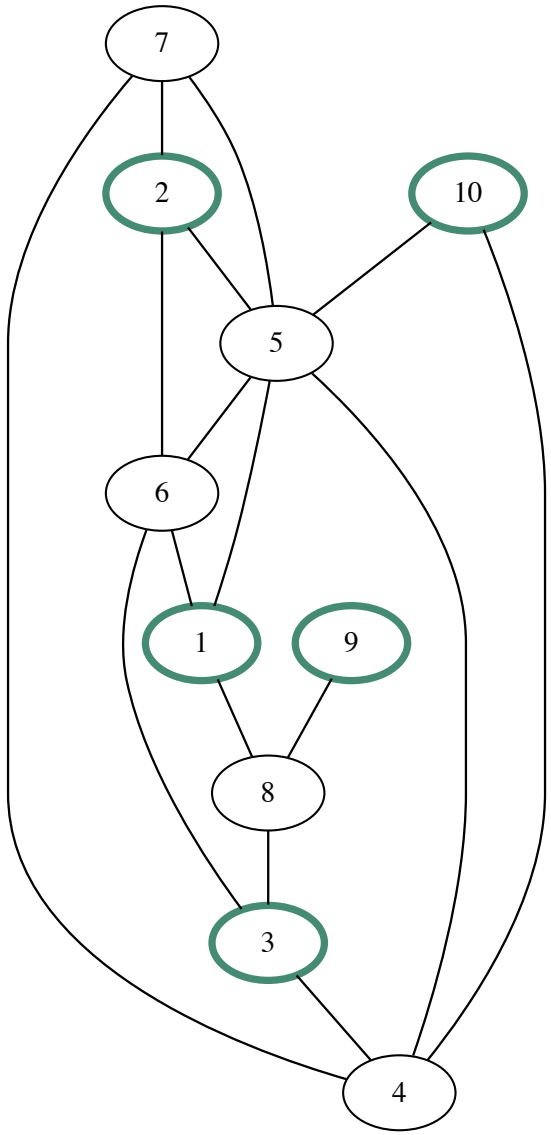
Assuming that an edge represents the fact that two persons cannot work with each other, we want to find **the largest disconnected subgraph**.

Independent set : what is a trivial independent set ?

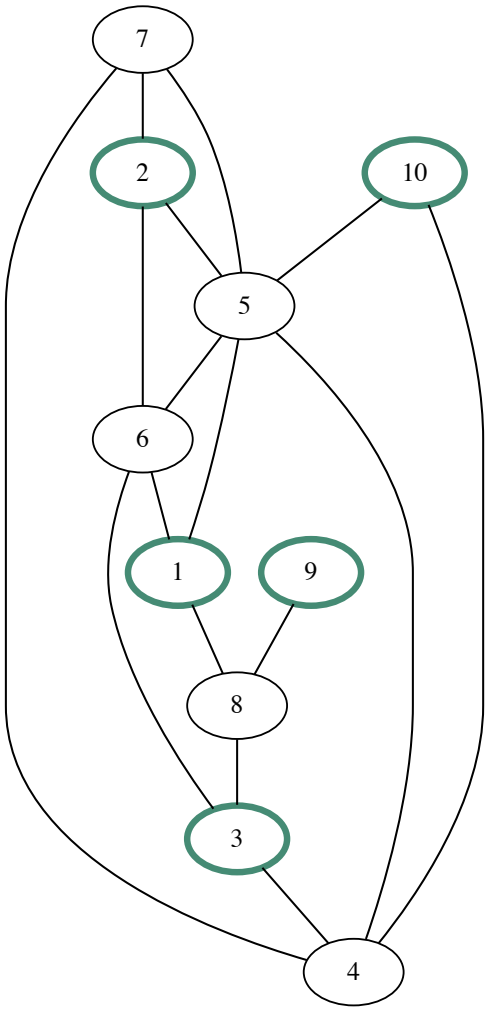



7
2
10
5
6
1
9
8
3
4


7
2
10
5
6
1
9
8
3
4


7
2
10
5
6
1
9
8
3
4

Maximal vs maximum independent set



Complexity

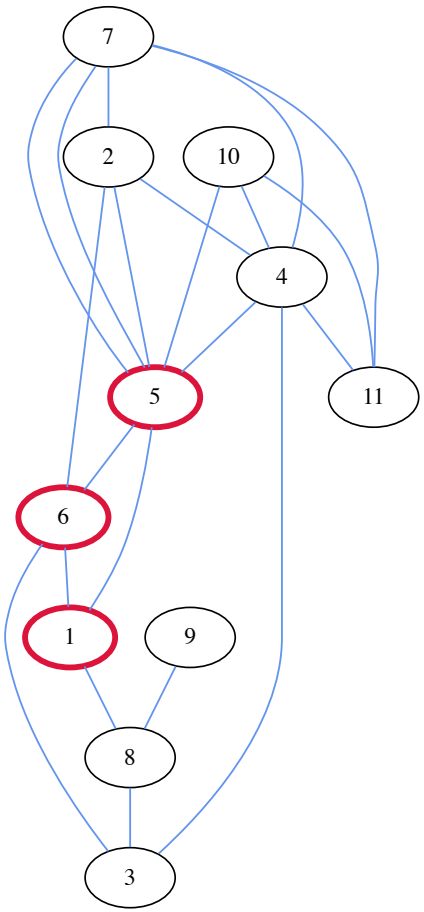
- The running time T of an algorithm A is its running time on the worst possible input (instance I) it can get (for a given size)
- The complexity $T(P)$ of a problem P is the running time of the best possible algorithm for that problem.

$$T(P) = \min_{A} \max_{I} T(P, A, I) \tag{2}$$

Equivalence between problems

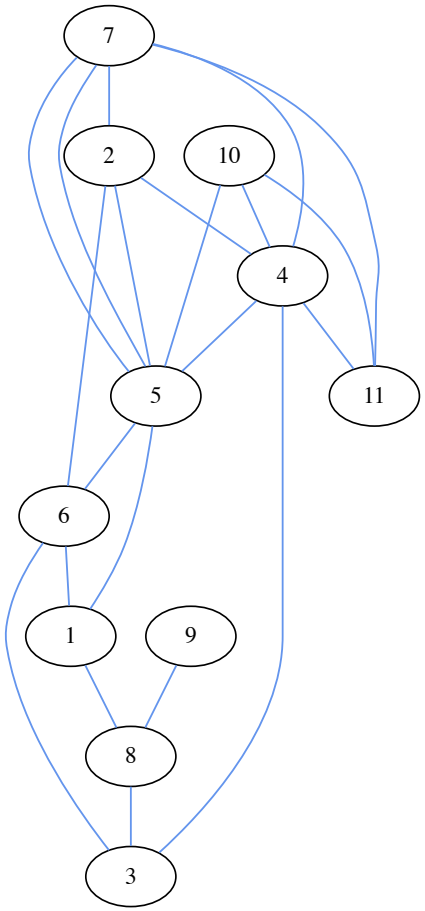
- Some problems have the same difficulty because they are "equivalent", in a specific way (polynomial reduction).
- On the contrary, some problems are strictly more complex than others.
- Hard problems : Maximum independent set, minimum coloring, smallest dominating set, TSP, etc.
- Easier problem : Shortest Path

Maximum clique problem

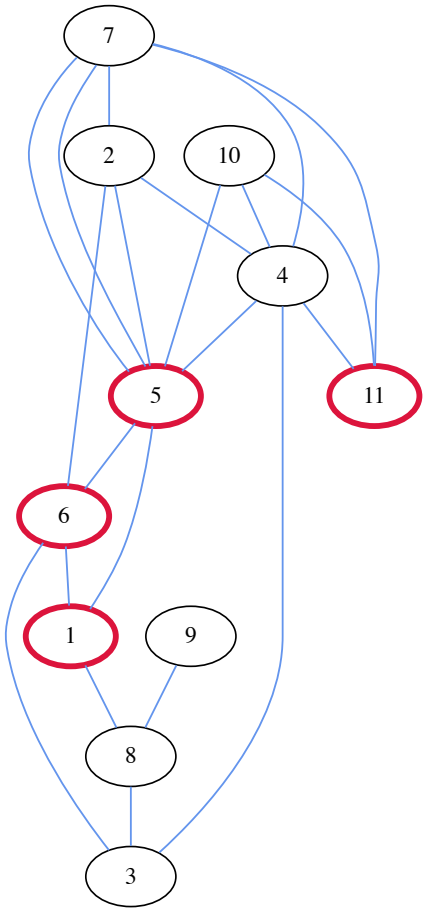


The **maxium clique** problem consists in finding the largest completely connected subgraph (the induced subgraph is **complete**)

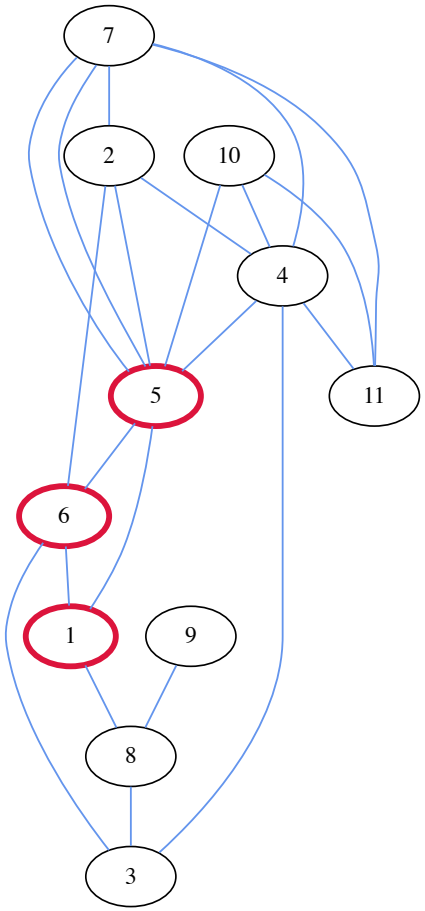
Maximum clique problem



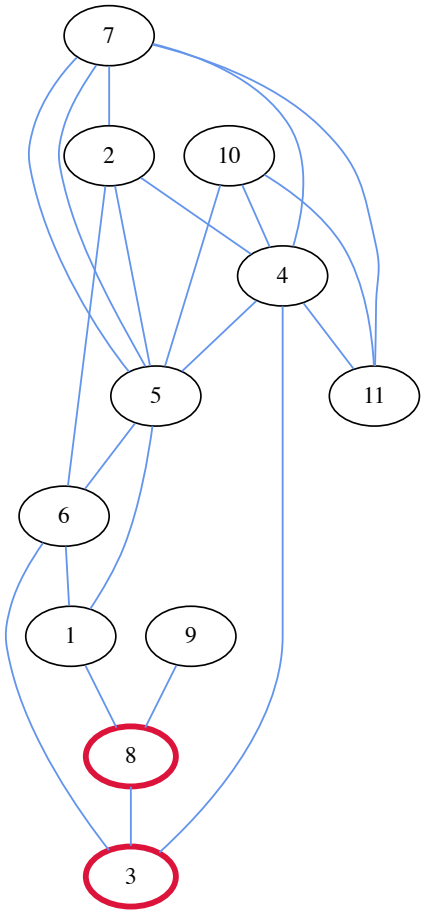
Maximum clique problem



Maximum clique problem



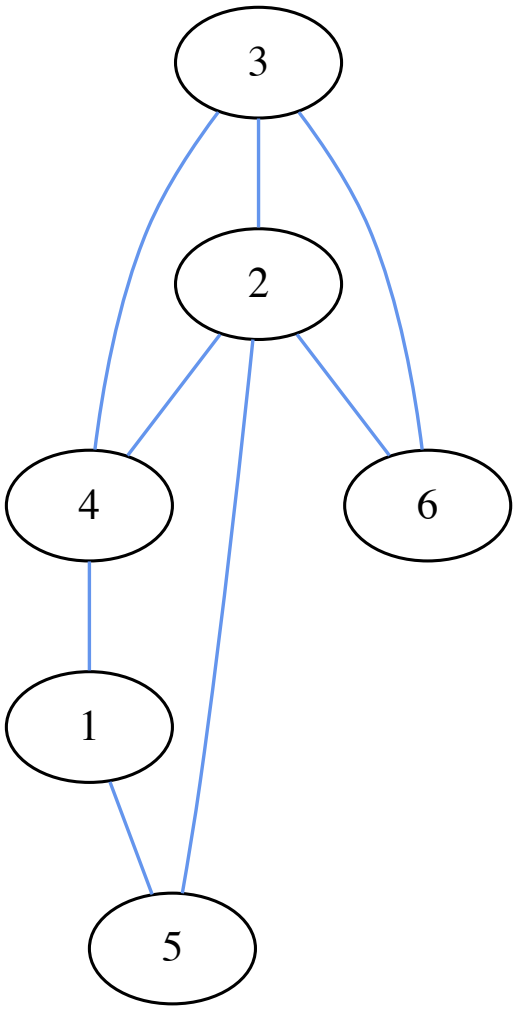
Maximum clique problem



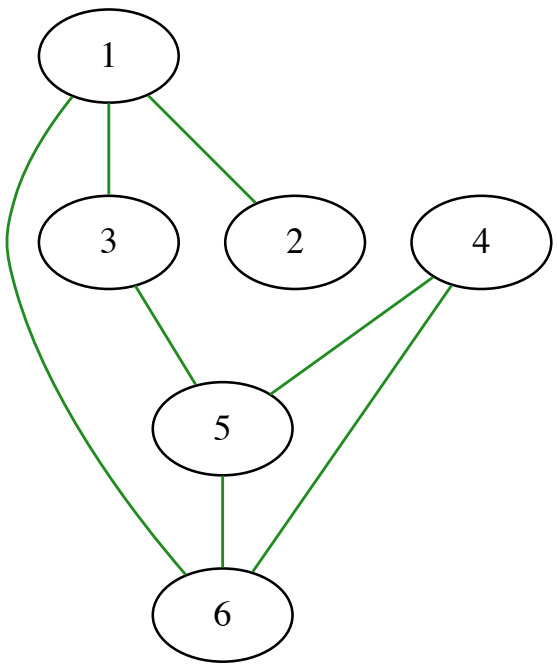
Equivalence between problems

Exercice 12 : Can you relate the maximum clique problem to another problem we saw before ?

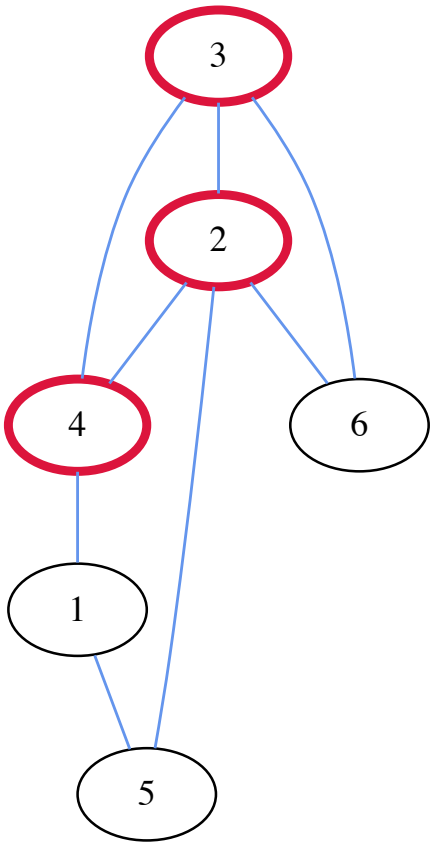
Linking problems



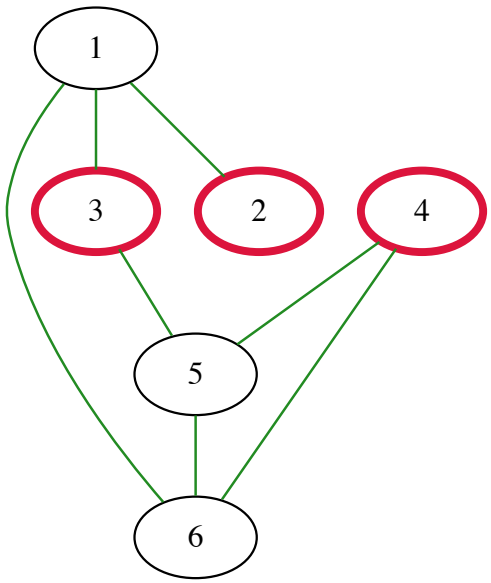
Linking problems



Linking problems



Linking problems



Polynomial-time reduction

To study a problem, it is sometimes useful to transform it into another.

Exercice 13 : Transformation

cd clique/ and use **complement_graph.py** in order to transform a graph into its **complement graph**.

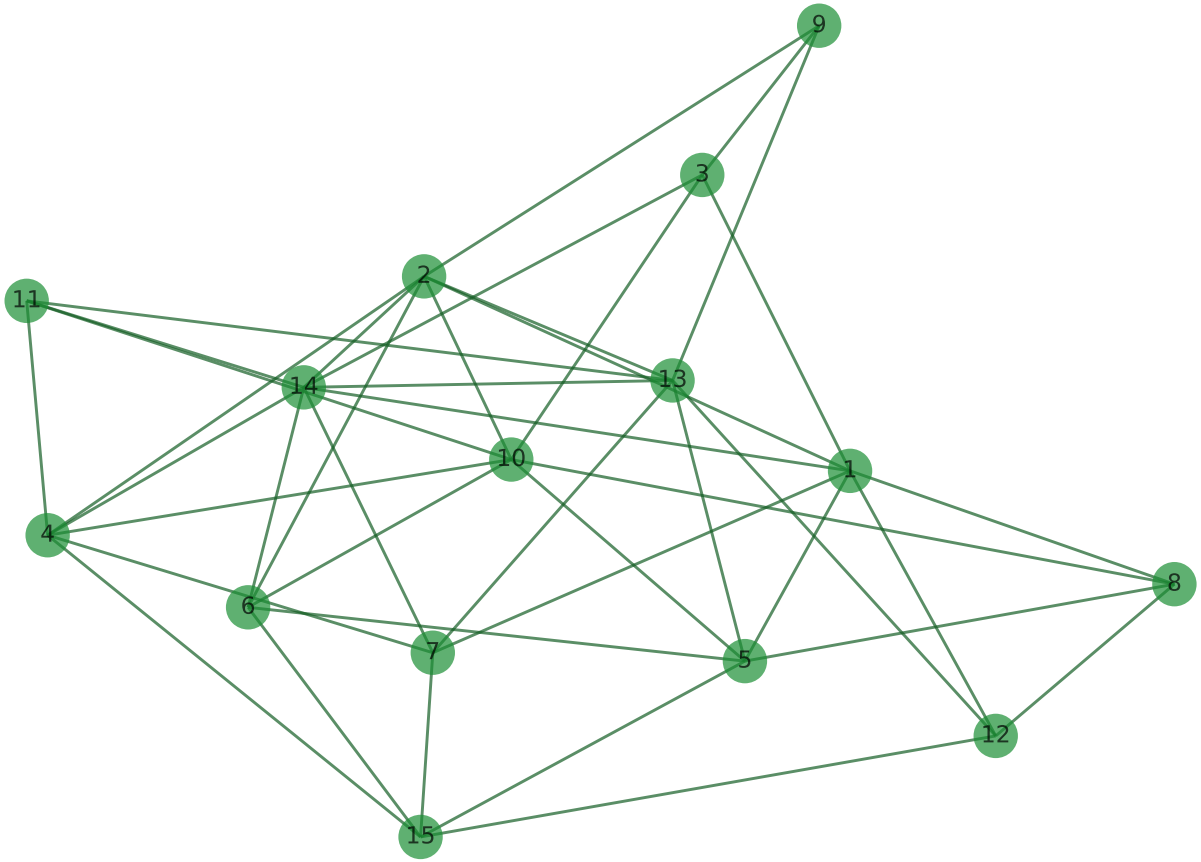
Exercice 13 : Transformation

cd clique/ and use **complement_graph.py** in order to transform a graph into its **complement graph**.

What is the complexity of this operation ?

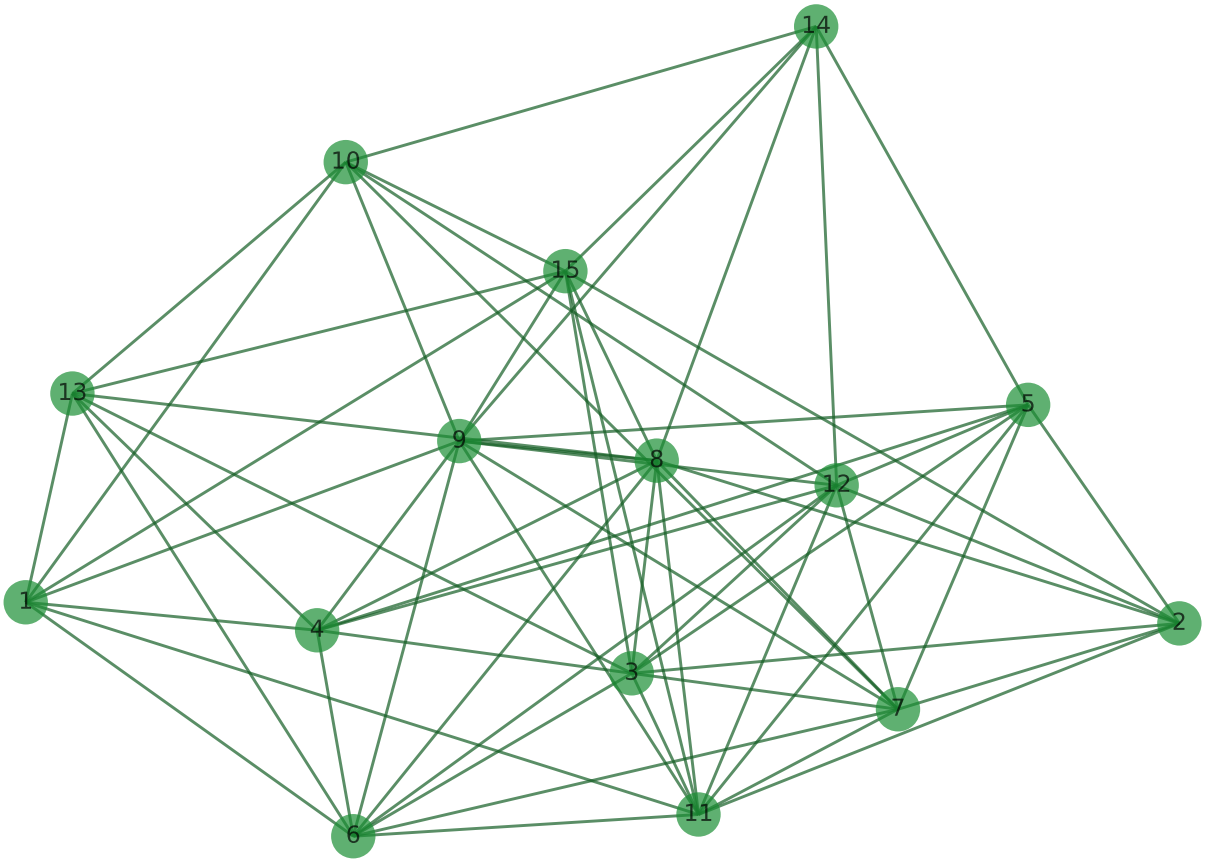
Complement graph

images/initial graph.pdf



Complement graph

images/complement graph.pdf



Dominating set to set covering

- This is another example of two problems that are equivalent.

Problems that are not equivalent

- Eulerian paths and hamiltonian paths

Classes of complexity

- Problems have been gathered under **classes of complexity**
- **P** : we can obtain a solution with polynomial complexity
- **NP** : we can verify a solution in polynomial time (doesn't mean we can find a solution in polynomial time)
- **NP hard** : if it is in P, all NP problems are in P.
- **NP complete** : NP and NP hard

P=NP ?