Algorithmics
Final Exam #3 (P3)

Undergraduate 2^{nd} year - S3
EPITA

17 December 2019 - 9 : 30

Instructions (read it) :

- You must answer on **the answer sheets provided.**
  - No other sheet will be picked up. Keep your rough drafts.
  - Answer within the provided space. **Answers outside will not be marked**: Use your drafts!
  - Do not separate the sheets unless they can be re-stapled before handing in.
  - Penciled answers will not be marked.
- The presentation is negatively marked, which means that you are marked out of 20 points and the presentation points (maximum of 2) are taken off this grade.
- **Code:**
  - All code must be written in the language `Python` (no C, CAML, ALGO or anything else).
  - **Any `Python` code not indented will not be marked.**
  - All that you need (classes, types, routines) is indicated where needed!
  - You can write your own functions as long as they are documented (we have to know what they do). In any case, the last written function should be the one which answers the question.
- Duration : 2h

Exercise 1 (Implementation and questions – *2 points*)

Imagine a simple undirected graph G of 9 vertices represented by the following adjacency matrix:

	1	2	3	4	5	6	7	8	9
1		V	V						
2	V								
3	V			V	V				
4			V			V	V		
5			V						
6				V					
7				V				V	V
8							V		
9							V		

1. Give two graph properties that the transitive closure of G has.

2. Give the Depth-First Search postorder list of vertices of G. As usual, you will start from the vertex 1 and assume that the successors are met in increasing order.

Exercise 2 (Warshall - Union-Find – *4 points*)

Let G be the graph $< S, A >$ with vertices numbered from 0 to 13. The Warshall algorithm seen in lecture built the following matrix (no value = *false*, 1 = *true*) from G:

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

1. What are the connected components (vertex sets) of the graph G?

2. The algorithms `find` and `union` (non-optimized versions) are applied the list of edges A of G.
   Among the following vectors, which could correspond to the result?

P_1

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

P_2

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

P_3

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

P_4

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

Exercise 3 (I want to be tree – *5 points*)

Définition :

A **tree** is an **acyclic connected** graph.

Using **imperatively a depth-first search**, write the function `isTree` that tests whether a graph is a tree.

Exercise 4 (Distances and center – *6 points*)

Definitions

- The **distance** between two vertices in a graph is the number of edges in a **shortest path** connecting them.
- The **eccentricity** of a vertex x in $G =< S, A >$ is defined by:

$$\text{exc}(x) = \max_{y \in S}\{distance(x, y)\}$$

- The **radius** of a graph is the minimum eccentricity of its vertices. That is to say, the shortest distance a vertex can be from other points in the graph.
- The **center** of a graph is the set of vertices with eccentricity equal to the graph's radius (vertices of minimum eccentricity).

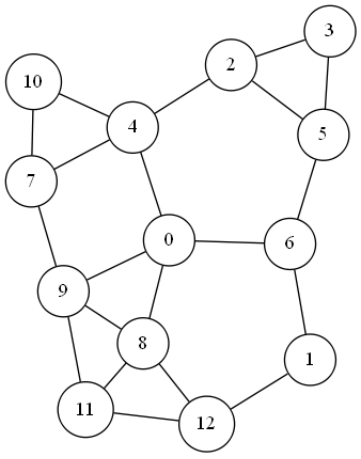
Write the function `center(`G`)` that returns the center of the graph G (a list).

In the graph G_6:

Vertices 0, 4 and 6 are of eccentricity 3.
Vertices 3 and 11 are of eccentricity 5.
Remaining vertices are of eccentricity 4.

The radius of G_6 is 3 and the vertices 0, 4 and 6 constitute its center.

```
>>> center(G6)
[0, 4, 6]
```



Graph G_6

Exercise 5 (What is this? – *3 points*)

```python
import Graph, Queue

def __aux(G, v, m, res, count):
    q = Queue()
    q.enqueue(v)
    m[v] = True
    while not q.isempty():
        cur = q.dequeue()
        res[cur] = count
        count += 1
        for succ in G.adjlists[cur]:
            if not m[succ]:
                m[succ] = True
                q.enqueue(succ)
    return count

def mystery(G, src):
    m = [False] * G.order
    res = [0] * G.order
    count = __aux(G, src, m, res, 0)
    for v in range(G.order):
        if not m[v]:
            count = __aux(G, v, m, res, count)
    NG = Graph(G.order, directed=G.directed)
    for v in range(G.order):
        for s in G.adjlists[v]:
            if G.directed or res[v] <= res[s]:
                NG.addedge(res[v], res[s])
    return NG
```

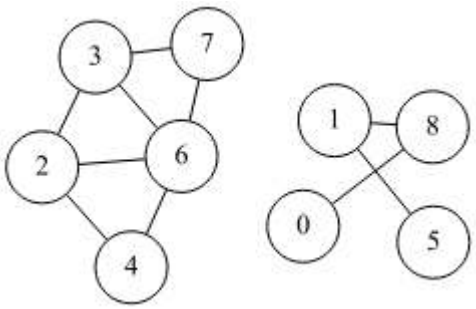


FIGURE 4 – Graph G_4

1. Draw the graph resulting of the call `mystery(`G_4`, 0)` (vertices are encountered in increasing order) where G_4 is the graph of the figure 4.

2. List the vertices in order of encounter during this run.

3. From a graph with k connected components, how many connected components the graph produced by the function `mystery` will have?

Appendix

Classes `Graph` and `Queue` are supposed imported.

Graphs

All exercises use the implementation with adjacency lists of graphs.
Graphs we manage cannot be empty.

```
class Graph:
    def __init__(self, order, directed = False):
        self.order = order
        self.directed = directed
        self.adjlists = []
        for i in range(order):
            self.adjlists.append([])
```

Queues

- `Queue()` returns a new queue
- q`.enqueue(`e`)` enqueues e in q
- q`.dequeue()` returns the first element of q, dequeued
- q`.isempty()` tests whether q is empty

Others

- `range`
- on lists:
  - `len(L)`
  - `L.append()`
  - `L.pop()`
  - `L.insert(index, elt)`

Your functions

You can write your own functions as long as they are **documented** (we have to know what they do).
In any case, the last written function should be the one which answers the question.