Algorithmics
Final Exam #3 (P3)

Undergraduate 2^{nd} year (S3)
EPITA

18 December 2018 - 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 (Warshall - Union-Find – *3 points*)

Let G be the graph $< S, A >$ with vertices numbered from 0 to 9. 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
0	1		1			1	1			
1		1		1				1	1	
2	1		1			1	1			
3		1		1				1	1	
4					1					1
5	1		1			1	1			
6	1		1			1	1			
7		1		1				1	1	
8		1		1				1	1	
9					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
2	-1	-1	1	1	2	2	1	1	4

P_2

0	1	2	3	4	5	6	7	8	9
2	-1	-1	1	-1	2	2	1	1	4

P_3

0	1	2	3	4	5	6	7	8	9
5	-1	-1	1	-1	2	5	1	3	4

P_4

0	1	2	3	4	5	6	7	8	9
5	-1	-1	1	-1	2	5	1	3	-1

Exercise 2 (In the depth of the spanning forest – *2 points*)

Build the spanning forest of the graph `G` for the depth first traversal from vertex 1. Vertices are encountered in increasing order. *Add to the forest the various kinds of edges met during the traversal, with an explicit legend.*

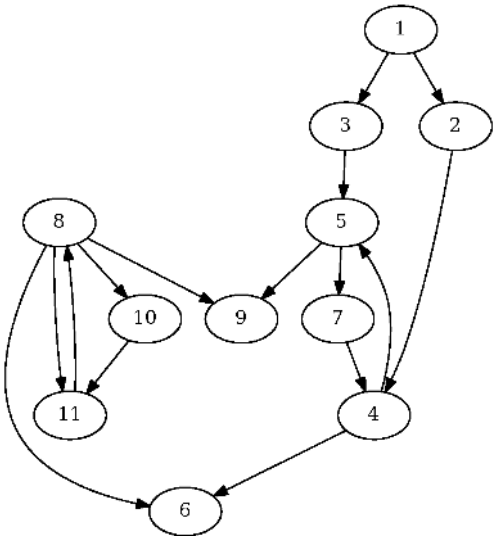

Figure 1: A digraph

Exercise 3 (Components – *3 points*)

Write the function `components` that determines the connected components of a grah.

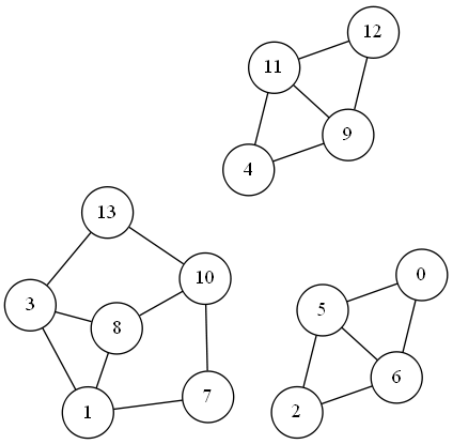

Figure 2: Graph `G1`

The function returns the pair:

- Number of connected components;
- The component vector: it for each vertex the number the component it belongs to.

Application example, with `G1` the graph in figure 2:

```
>>> components(G1)
(3, [1, 2, 1, 2, 3, 1, 1, 2, 2, 3, 2, 3, 3, 2])
```

Exercise 4 (Diameter – 5 *points*)

Définitions :

- The **distance** between two vertices in a graph is the number of edges in a **shortest path** connecting them.
- The **diameter** of a graph is the **highest distance** between any pair of vertices.

When the graph is a tree, computing the diameter is simple:

- From any vertex s_0, find a vertex s_1 whose distance from s_0 is maximal.
- From s_1, find a vertex s_2 whose distance from s_0 is maximal.
- The distance between s_1 and s_2 is the graph diameter.

Write the function `diameter` that computes the diameter of a graph that is a tree.

Exercise 5 (Euler – 6 *points*)

A graph (undirected) is *Eulerian* if contains an *Eulerian path*, that is a path which visits every edge exactly once, or an *Eulerian cycle* which starts and ends on the same vertex.

Euler's theorem states that such a path (Eulerian path) exists if, and only if, the graph is **connected**, and there are **zero or two nodes of odd degree**. If there is no nodes of odd degree at all, the path is an Eulerian cycle.

Using **imperatively a depth-first search**, write a function that tests whether a **simple** graph is *Eulerian*, i.e. it contains an Eulerian path or cycle.

A little help: How to know the degree of a vertex with adjacency list implementation?

Exercise 6 (What is this? – *3 points*)

```
def __what(G, x, d):
    lc = 0
    for y in G.adjlists[x]:
        if d[y] == -1:
            d[y] = d[x] + 1
            lc = max(lc, __what(G, y, d))
        else:
            if d[y] + 1 < d[x]:
                lc = max(lc, d[x] - d[y] + 1)
    return lc

def what(G):
    d = [-1] * G.order
    lc = 0
    for x in range(G.order):
        if d[x] == -1:
            d[x] = 0
            lc = max(lc, __what(G, x, d))
    return (lc, d)
```

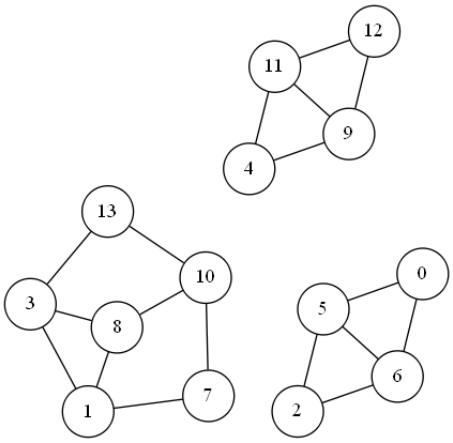

Figure 4 – Graph G_4

1. What is the result returned by `what(`G_4`)` with G_4 the above graph whose adjacency lists are sorted in increasing order.

2. What does the array `d` represent?

3. **Bonus:** What does the value `lc` represent?

Appendix

Classes `Graph` and `Queue` are supposed imported. Graphs we manage cannot be empty.

Graphs

All exercises use the implementation with adjacency lists of graphs.

```
class Graph:
    def __init__(self, order, directed = False):
        self.order = order
        self.directed = directed
        self.adjlists = []
        for i in range(order):
            self.adjlists.append([])
    def addedge(self, src, dst):
        self.adjlists[src].append(dst)
        if not self.directed and dst != src:
            self.adjlists[dst].append(src)
```

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

Functions you can use:

- on lists: `len`
- `range`.

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.

Reminder:

Integers can not be "passed as reference" in Python...