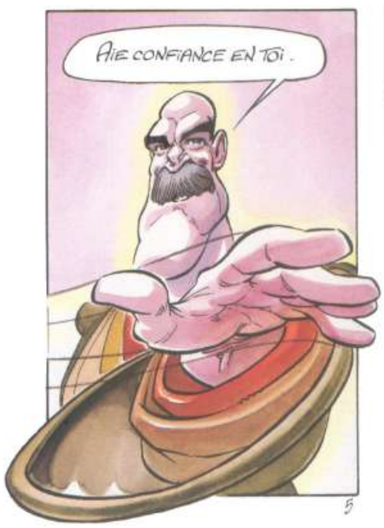
Algorithmics
Final Exam #3 (P3)

Undergraduate 2^{nd} year - S3#
Epita

May 12, 2021 - 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 – *4 points*)

Let G_1 be the graph $< S, A >$ with vertices numbered from 0 to 8.
The algorithms `find` and `union` (non-optimized versions) applied the list of edges A of G_1 allowed to build the following vector P.

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

1. What are the connected components (vertex sets) of the graph G_1?

2. Give the adjacency matrix of the transitive closure of G_1 (no value = *false*, 1 = *true*).

3. This time the optimized versions of find (with path compression) and union (by rank) are applied to the list of edges A of G_1. Among the following vectors, which could correspond to the result?

	0	1	2	3	4	5	6	7	8
P_1	8	8	4	7	-3	8	4	-2	-4

	0	1	2	3	4	5	6	7	8
P_2	1	-4	6	7	6	1	-3	-2	2

	0	1	2	3	4	5	6	7	8
P_3	8	8	6	7	6	8	-3	-3	-4

	0	1	2	3	4	5	6	7	8
P_4	-4	0	4	7	-3	0	4	-2	0

Exercise 2 (Get Back – *4 points*)

In some problems, we use a mark vector in which each vertex can have *3 values* during a traversal:

- A value (`None` for instance) for the unmarked vertices
- A value for the first meeting (for instance 1)
- A value for the second meeting (for instance 2)

Using **imperatively a mark vector *with 3 values*, no matter which ones**, write the function `acyclic(`G`)` that checks whether the digraph G is acyclic.

Exercise 3 (Density – *6 points*)

An undirected simple graph (without multiple links nor loops) is called *dense* when the number of edges (p) is large compared to the number of vertices (n).

For this exercise we define the *density* of a graph by the measure p/n.

1. For a simple connected graph:
   (a) **The less dense**: give the minimal value of p as a function of n. What type of graph can have these measures?
   (b) **The most dense**: give the maximal value of p as a function of n. What type of graph can have these measures?
2. Write the function `density_components` that returns the list of the *density* of the connected components of a simple undirected graph.

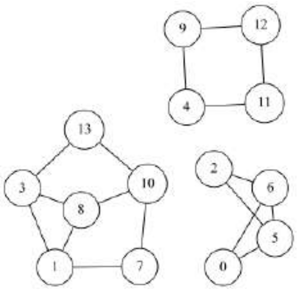


Figure 1: Graph `G_3cc`

Application example, with `G_3cc` *the graph in figure 1:*

```
>>> density_components(G_3cc)
[1.25, 1.3333333333333333, 1.0]
```

- The first component has 4 vertices, 5 edges
- The second component has 6 vertices, 8 edges
- The third component has 4 vertices, 4 edges

Exercise 4 (Levels – *6 points*)

Definitions:

- The **distance** ($distance(x, y)$) between two vertices x and y 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:

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

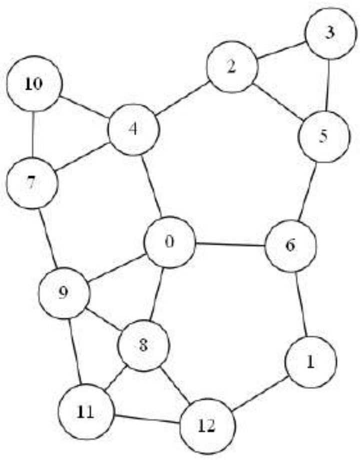


Figure 2: `Gc`

Write the function `levels(G)` that returns a list L of length $exc(src) + 1$ in which each value $L[i]$ contains the vertices at a distance i from src in G.

Application example on the graph `Gc` in figure 2 (the order in the sub-lists is not important):

```
>>> levels(Gc, 0)
[[0], [4, 6, 8, 9], [2, 7, 10, 1, 5, 11, 12], [3]]
```

Appendix

Classes `Graph` and `Queue` are assumed to be imported.

Graphs

All exercises use the implementation with adjacency lists of graphs.
Graphs we manage cannot be empty. There is neither multiple edges nor loops.

```
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

Others

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

  and any operator…

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.