Algorithmics
Midterm #4 (C4)

Undergraduate 2^{nd} year (S4)
Epita

5 March 2019 - 14 : 45

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 known what they do).
    In any case, the last written function should be the one which answers the question.
- ☐ Duration : 2h

IL FAUT MAINTENANT QUE J'ÉTUDIE LES PLANS DE LA MAISON POUR PRÉPARER MA VISITE...

Exercise 1 (Cut points, cut edges – *5 points*)

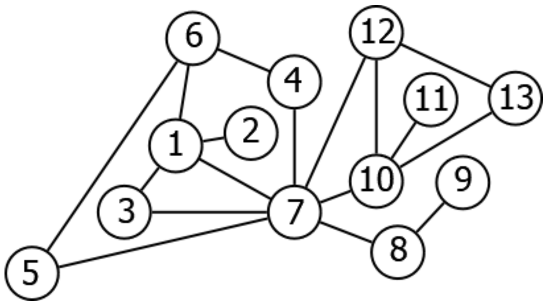


Figure 1: Graph G_1

1. Give the cut points of G_1.
2. Give the cut edges of G_1.
3. Give, using lists of edges, the biconnected components of G_1 (one component per line).
4. Give, for each vertex of the graph G_1, the table of *prefix* and *higher* values obtained during the depth-first search traversal of the graph G_1. You will use for *higher* the following formula:

$$higher(x) = min \begin{cases} prefix(x) & \\ higher(y) & \forall\ (x, y) \text{ discovery edge} \\ prefix(z) & \forall\ (x, z) \text{ back edge} \end{cases}$$

As usual, the root of the traversal is the vertex 1 and successors are processed in increasing order.

Exercise 2 (I want to be a tree – *8 points*)

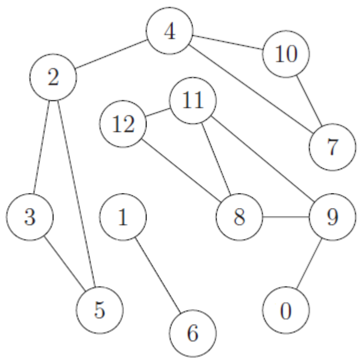


Figure 2: Not a tree yet

The aim of this exercise is to turn any graph into a *tree* while minimizing the number of modifications.

1. Give two definitions of a graph that is a *tree*

2. We perform a **depth-first search** of a graph:

   (a) What are the edges that can be removed **during the traversal** without increasing the number of connected components?

   (b) Apply to the graph in figure 2: give the list of the edges that will be removed during the traversal where vertices are choosen in increasing order.

3. During the depth-first search, we assign to each vertex the number the component it belongs to (from 1 to k, if there are k components):

   (a) How many edges have to be added to make the graph connected?

   (b) What are the edges to add **during the traversal** to make the graph connected?

   (c) Apply to the graph in figure 2: give the list of the edges to add to make it connected.

4. Write the function that

   - builds the connected component vector cc of the original graph ;
   - adds the edges to the graph to make it connected ;
   - removes "useless" edges from the graph (without increasing the number of connected components).

Exercise 3 (Condensation – *4 points*)

Let G be a digraph with k strongly connected components: C_0, C_1, ..., C_{k-1}. The *condensation* of G is the digraph $G_R = < S_R, A_R >$ defined by:

- $S_R = \{C_0, C_1, \cdots, C_{k-1}\}$
- $C_i \rightarrow C_j \in A_R \Leftrightarrow$ There exists at least an edge in G with its head in the strongly connected component C_i and its tail in the strongly connected component C_j.

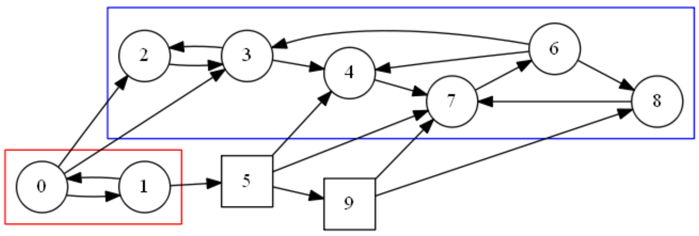


Figure 3: Digraph G_1

The aim here is to build the condensation of a graph G. We have the list of the strongly connected components of G (each component is a list of the vertices it contains).
For instance, the following list is the component list of the graph in figure 3:

```
scc = [[2, 3, 4, 6, 7, 8], [9], [5], [0, 1]]
```

Write the function `condensation(`G`, `scc`)` that builds the condensation G_R of a digraph G, with scc its component list. The function returns G_r and the vector of components: a vector that gives for each vertex the number the component it belongs to (the vertex in G_R).

For instance, with G_1 the graph in figure 3:

```
>>> (Gr, comp) = condensation(G1, scc)
>>> comp
[3, 3, 0, 0, 0, 2, 0, 0, 0, 1]
```

Exercise 4 (Digraphs and Mystery – *3 points*)

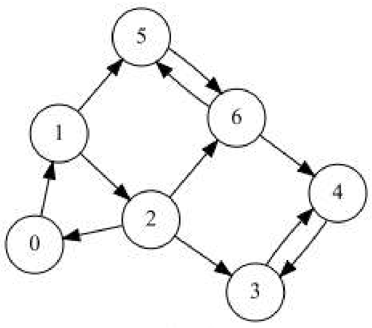


Figure 4: Digraph G_2

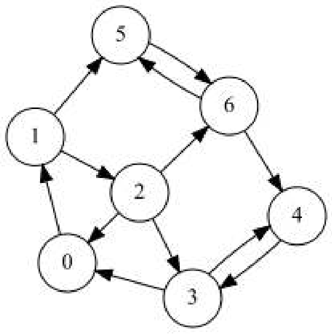


Figure 5: Digraph G_3

```
def __test(G, x, p, c):
    c += 1
    p[x] = c
    rx = p[x]
    for y in G.adjlists[x]:
        if p[y] == 0:
            (ry, c) = __test(G, y, p, c)
            if ry == -1:
                return (-1, c)
            rx = min(rx, ry)
        else:
            rx = min(rx, p[y])

    if rx == p[x]:
        if p[x] != 1:
            return (-1, c)

    return (rx, c)

def test(G):
    p = [0] * G.order
    c = 0
    (r, c) = __test(G, 0, p, c)
    return (r != -1) and (c == G.order)
```

We assume that the adjacency lists are sorted in increasing order in the graph in parameter.

1. Let G_2 and G_3 be the digraphs in figures 4 and 5. For each of the following calls:
   - how many calls of `__test` have been done?
   - what is the result returned by `test`?

   (a) `test(`G_2`)`
   (b) `test(`G_3`)`

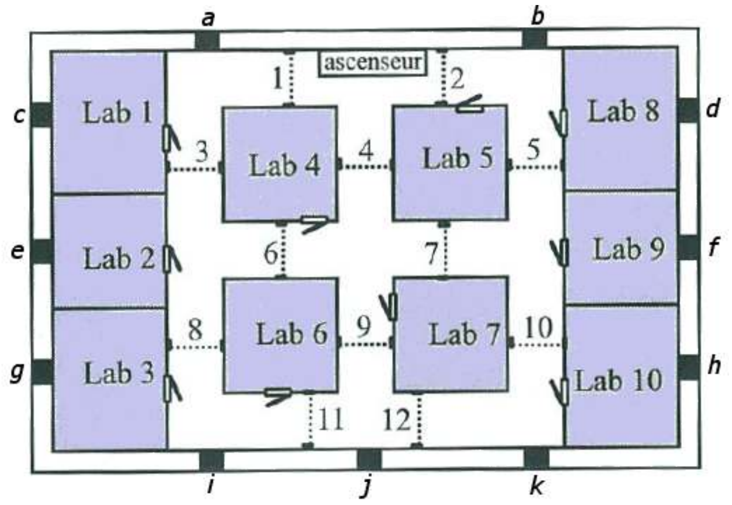
2. Let G be a digraph. What is the information returned by `test(`G`)`?

Exercise 5 (Saving Algernon – *Bonus*)

Algernon, the famous mouse, has been kidnapped by a research laboratory member, who brings it to his lab. But Algernon is clever and has managed to escape, taking advantage of the opening of the laboratory door. We must find Algernon: during the night it might have gone anywhere and now it can be in a lab or stuck in an air vent.

Here is a map of the research laboratory:

- 10 labs (Lab 1 to Lab 10), one for each researcher;
- the air vents, represented by the black squares (from a to k): a mouse can go in but cannot go out and each air vent leads to a dead end ;
- the elevator ("ascenseur"), which leads directly outside ;
- and, in dashed lines, the motion detectors (numbered from 1 to 12).



Motion detectors are rather archaic: simple mechanical indicators that count passage number (one cannot know when they have been triggered).

Here are the numbers of detections during the night:

detector #	1	2	3	4	5	6	7	8	9	10	11	12
nb detections	2	1	3	3	1	3	0	2	2	3	2	1

Which researcher is the kidnapper? Where is Algernon now?

1. (a) Which researcher is the kidnapper?

   (b) Where is Algernon now?

2. Justify (use the given map to illustrate your justification).

Appendix

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

Graphs

```python
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)

    def removeedge(self, src, dst):
        self.adjlists[src].remove(dst)
        if not self.directed and dst != src:
            self.adjlists[dst].remove(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

- 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.