Algorithmics
Midterm #4 (C4)

Undergraduate 2nd year (S4)
Epita

6 March 2018 - 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

JE CROIS QUE QUELQU'UN M'A DEMANDÉ QUAND COMMENCERONT NOS AVENTURES...
ELLES COMMENCERONT DÈS QUE VOUS AUREZ PRIS LA PEINE DE TOURNER CETTE PAGE!... JE VOUS REMERCIE POUR VOTRE ATTENTION!

Exercise 1 (Cut points, cut edges – *3 points*)

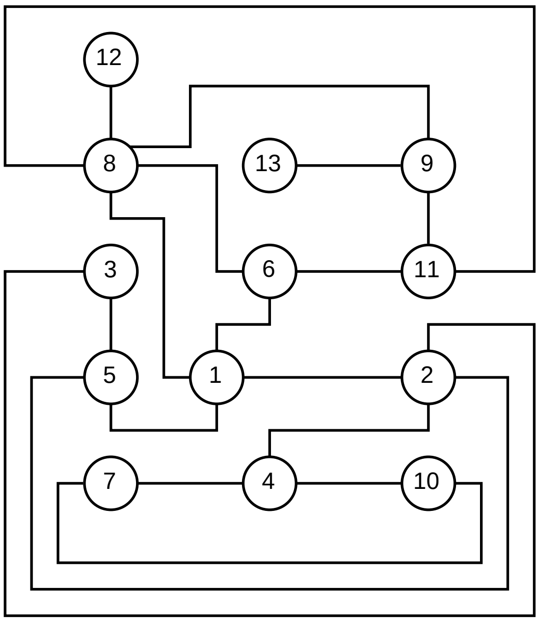

Figure 1: Graph G_1

1. Build the spanning forest of the graph G_1 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.
2. Give the cut points of G_1.
3. Give the cut edges of G_1.

Exercise 2 (SCC and condensation – *5 points*)

Let G be a digraph with k strongly connected components: C_1, C_2, …, C_k. The *reduced digraph* or *condensation* of G is the digraph $G_R = < S_R, A_R >$ defined by:

- $S_R = \{C_1, C_2, \cdots, C_k\}$
- $C_i \to C_j \in A_R \Leftrightarrow$ 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.

1. **Digraph → condensation**

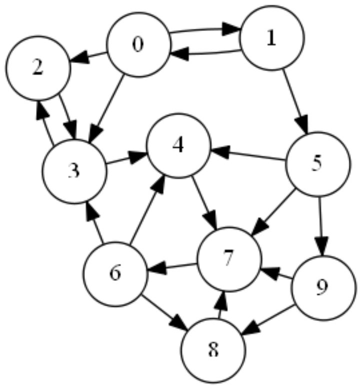

Figure 2: Digraph G_2

   (a) Give and numerate the strongly connected components of the digraph G_2 (figure 2).
   (b) Draw the condensation of G_2 (use numbers from question 1).
   (c) Can the addition of a single edge make the digraph strongly connected? Justify.

2. **Condensation → digraph**

   G_3 is the digraph whose connected components are in the array *comp* below and whose condensation is in the figure 3.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
comp	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	6	6

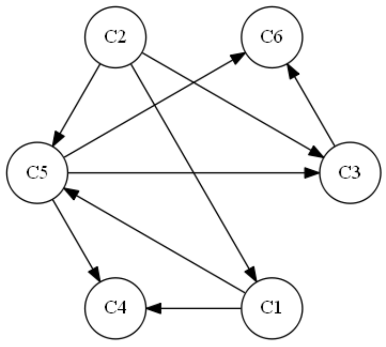

Figure 3: Condensation of G_3

   (a) Which vertices are not reachable from the vertex 0?

   (b) Among the following paths, which ones can not exist in G_3?

   - $3 \rightsquigarrow 7$
   - $4 \rightsquigarrow 21$
   - $18 \rightsquigarrow 2$
   - $11 \rightsquigarrow 15$

   (c) Give the minimum number of edges that must be added to G_3 in order to make it strongly connected.

Exercise 3 ("Global Connectivity Indicators" – *6 points*)

Some indicators measure the degree of fragmentation of a graph into connected components separated from each others. Let N be the number of vertices of a graph, k its number of connected components and $n_1, \cdots, n_k$ the numbers of vertices in each component ($n_1 + n_2 + ... + n_k = N$). We can define two "connectivity indexes" whose values are between 0 (empty graph) and 1 (connected graph).

- "Simple connectivity index": $IC_1 = (N - k)/(N - 1)$
- "Weighted connectivity index": $IC_2 = (n_1^2 + n_2^2 + ... + n_k^2)/(N^2)$

For instance for the graph in figure 4:

- $IC_1 = (14 - 3)/(14 - 1) = 0,9166..$
- $IC_2 = (4^2 + 4^2 + 6^2)/14^2 = 0,3469..$

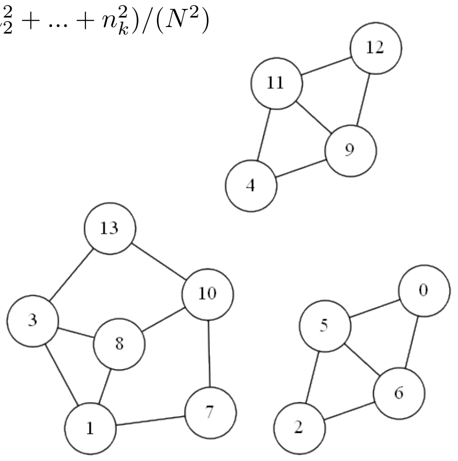

Figure 4: 3-component graph

1. In terms of probabilities, what information does the "weighted connectivity index" give us?

2. Write the functions computing both "connectivity indexes" (simple and weighted) of a graph (represented by adjacency lists) using a **depth-first traversal**.

Exercise 4 (Strongly Connected? – *7 points*)

We want to write a function that tests the strong connectivity of a digraph. We will use a simplified version of Tarjan's algorithm.

Tarjan's principle reminder:

We use a depth-first search in which vertices are marked in preoder .

For each vertex x we compute the value $return(x)$. It is the preorder encounter of a vertex met before x and in the same strongly connected component as x. If such a vertex does not exist, it is the preorder encounter of x. The value of $return(x)$ is defined by:

$$return(x) = min\{prefix[x], return(y), prefix[z]\}$$

This minimum is computes for

- each vertex y which is a descendant of x in the spanning forest,
- each vertex z such that $x \to z$ is a back edge, or a cross edge when the root of the strongly connected component of z is an ancestor of x.

Once the traversal from x is complete, we test if the return value of x is always the same as its prefix order value. If so, the vertex x is then a *component root*.

Questions:

1. In case of a strongly connected digraph, what is (are) the property(ies) of the first component root met during the traversal?
2. Write the function `is_strong`(G) that tests whether the digraph G is strongly connected.

Appendix

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

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

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.