Graphs (Graphes)
Applications

Exercise 3.1 (Connect me)

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

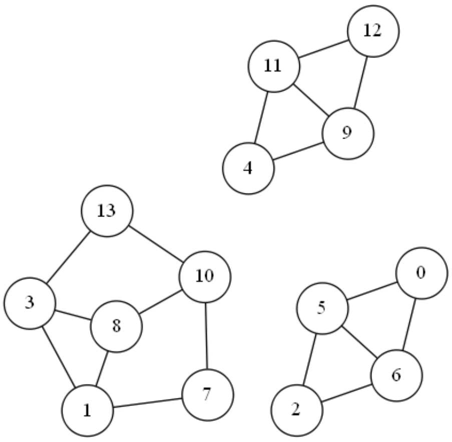

Figure 1: Graph `G_3cc`

`files/graph_3comp.gra`

The function returns the pair:

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

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

```
>>> components(G_3cc)
(3, [1, 2, 1, 2, 3, 1, 1, 2, 2, 3, 2, 3, 3, 2])
```

Exercise 3.2 (That's the way)

1. How to find a path (a chain) between two vertices in a graph? Give at least two different methods and compare them.
2. Write a function that searches for a path between two vertices. If a path is found, it has to be returned (a vertex list).

Exercise 3.3 (Coloring – *Final S3# - 2019*)

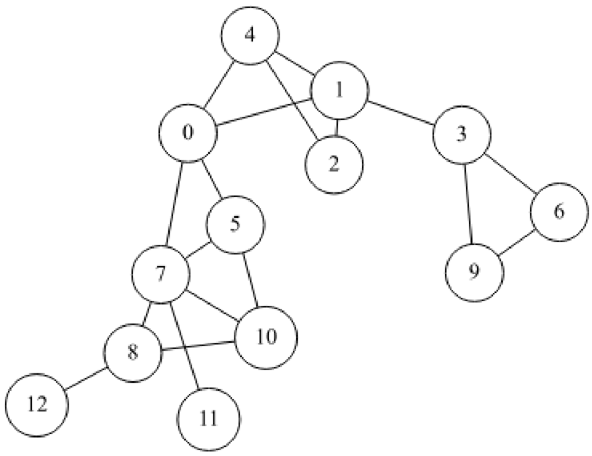

Figure 2: Bipartite graph?

`files/graph_sb.gra`

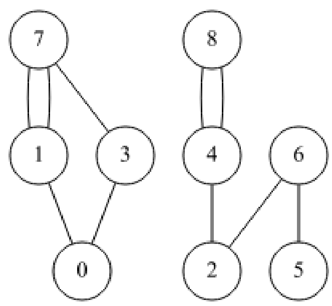

Figure 3: Bipartite graph?

`files/graph_2cc_multi.gra`

A bipartite graph is a graph $G =< S, A >$ where vertices can be partitioned into two sets S_1 et S_2, such that $\{u, v\} \in A$ implies either $u \in S_1$ and $v \in S_2$, or $u \in S_2$ and $v \in S_1$. That is, no edge connects vertices in the same set.

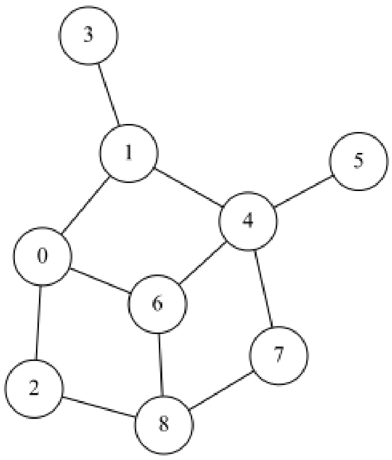

Figure 4: Bipartite graph?

`files/graph_bip_test2.gra`

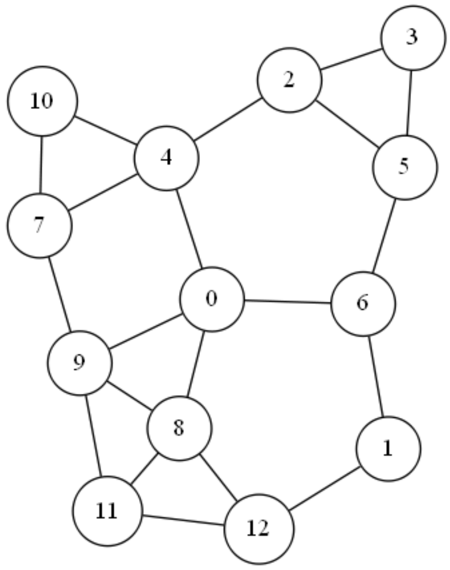

Figure 5: Bipartite graph? (`Gc`)

`files/graph_center.gra`

1. Are the graphs of figures 2 to 5 bipartite? For each bipartite graph give the two sets S_1 and S_2.
2. Write a function that tests whether a graph is bipartite.

Exercise 3.4 (Center of the world – *Final S3 - 2019*)

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 connected graph G (a list).

In the graph Gc (fig 5):

- 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 Gc is 3 and the vertices 0, 4 and 6 constitute its center.

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

Exercise 3.5 (I want to be tree – *Final S3 - 2019*)

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 3.6 (Compilation, cooking. . .)

1. *Scheduling, a simple example:*

   The following statements have to be executed with one processor:

   0 read(a)
   1 b ← a + d
   2 c ← 2 * a
   3 d ← e + 1
   4 read(e)
   5 f ← h + c / e
   6 g ← d * h
   7 h ← e - 5
   8 i ← h - f

   What are the possible orders of running?

   How to represent this problem with a graph?

   Each solution is called a *topological sort*.

2. What property should have the graph so that a topological sort exists?

3. When the graph is drawn lining up the vertices in a topological order, what can be observed?

4. (a) Let $suffix$ be the array of the last encounter of the vertices: the suffix order during the depth-first tarversal.
   Prove that for any pair of different vertices $u, v \in S$, if there is an arc in G from u to v, and if G has the property of question 2, then $suffix[v] < suffix[u]$.

   (b) Deduce an algorithm that finds a solution of topological order in a graph (Here, we assumed that a solution exists.)

   (c) What has to be changed in the algorithm if we want it to check if a solution exists?

   (d) Write a Python function that returns a topological order as a vertex list.

What about cooking?