Graphs (Graphes)
Implementations and traversals

1 Representations / Implementations

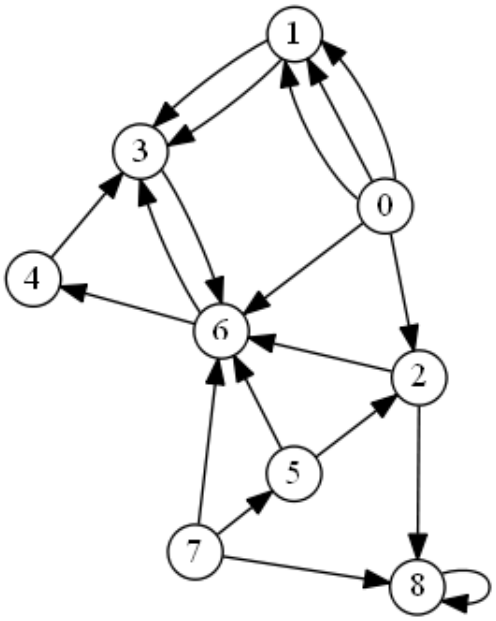


Figure 1: Digraph (Graphe orienté) G'_1

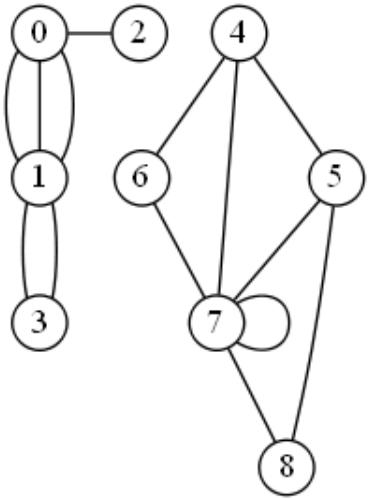


Figure 2: Graph (Graphe non orienté) G'_2

Exercise 1.1 (`GraphMat`: Adjacency Matrix)

This first implementation uses adjacency matrices.

1. With this implementation, what differences exist between a directed graph (or *digraph*) and a "undirected" one, a weighted graph and a none one, a simple graph and a mutigraph?
2. Give the matrix representations of the graphs in figures 1 and 2.
3. We want to use the same type to implement directed and undirected graphs, simple graphs and multigraphs. What should the implementation contain ?

Exercise 1.2 (`Graph`: Adjacency Lists)

1. What is the other way to represent/implement graphs?
2. With this representation, what differences exist between a directed graph and a "undirected" one, a simple graph and a mutigraph ?
3. Give the representations of the graphs in figures 1 and 2.
4. We want to use the same type to implement any kind of graphs: directed or not, simple and multigraphs. What should the implementation contain ?

Exercise 1.3 (`dot`)

Write the functions that return the `dot` representation of a graph for both implementations.

Examples:

- Digraph G'_1 (figure 1)

```
>>> print(todot(G1))
digraph {
  0 -> 1
  0 -> 2
  0 -> 6
  1 -> 3
  2 -> 6
  2 -> 8
  3 -> 6
  4 -> 3
  5 -> 2
  5 -> 6
  6 -> 3
  6 -> 4
  7 -> 5
  7 -> 6
  7 -> 8
  8 -> 8
}
```

- Graph G'_2 (figure 2)

```
    >>> print(todot(G2))
    graph {
      1 -- 0
      1 -- 0
      1 -- 0
      2 -- 0
      3 -- 1
      3 -- 1
      5 -- 4
      6 -- 4
      7 -- 4
      7 -- 5
      7 -- 6
      7 -- 7
      8 -- 5
      8 -- 7
    }
```

Exercise 1.4 (Load)

Our file format `GRA` is a text file, composed as follow:

- a first line containing 0 or 1: 0 for (undirected) graphs, 1 for digraphs
- a second line with a single integer representing the order of the graph
- a series of lines representing each edge: 2 vertex numbers split by a space

See the provided files: `digraph1.gra graph2.gra`.

Write the functions that build a graph from a `".gra"` file in both implementations.

Notes: Unless indicated otherwise, thereafter we will essentially use simple graphs. The examples used here will be the graph G_1 (simple digraph from G'_1) and G_2 (simple graph from G'_2).

---

2 Traversals

Exercise 2.1 (Breadth-first search (Parcours largeur))

1. Draw the spanning forests associated with the breadth-first searches of the graphs G_1 and G_2 from vertex 0, then from vertex 7 (vertices are chosen in increasing order).
2. Give the principle of the breadth-first search algorithm. Compare with the traversal of a general tree.
3. How can we store the spanning forest?
4. Write in both implementations the breadth-first search functions. The functions have to give the spanning forests.

Exercise 2.2 (Depth-first search (Parcours profondeur))

1. Draw the spanning forests associated with the depth-first searches of the graphs G_1 and G_2 from vertex 0 then from vertex 7 (vertices are chosen in increasing order).
2. Give the principle of the depth-first search algorithm. Compare with the traversal of a general tree.
3. **Graphs (undirected)**
   (a) What are the different kinds of arcs (egdes) met during the traversal?
   Add and name the missing arcs to the spanning forest of the depth-first search of G_2 obtained in question 1.
   (b) What has to be added to the depth-first search to classify arcs ?
   (c) Write the depth-first search function, when the graph is undirected and in matrix implementation. Add, during the traversal, the kinds of met arcs.
4. **Digraphs**
   (a) What are the different kinds of arcs (egdes) met during the depth-first search?
   Add and name the missing arcs to the spanning forest of the depth-first search of G_1 obtained in question 1.. How distinguish the different arcs?
   (b) We assign to each vertex a prefix value (first encounter) and a suffix value (last encounter). Write the conditions to classify arcs with these values (using an unique counter).
   (c) Write the depth-first search function of a digraph represented with adjacency lists. Add, during the traversal, the kinds of met arcs.
5. **Bonus**
   The depth-first search can be iterative.
   Give the principle and write the traversal for a digraph in adjacency list implementation.

3 Applications

Exercise 3.1 (Path – *Final S3# 2017*)

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 3.2 (Bipartite graph (Graphes bipartis) – *Final S3 - 2017*)

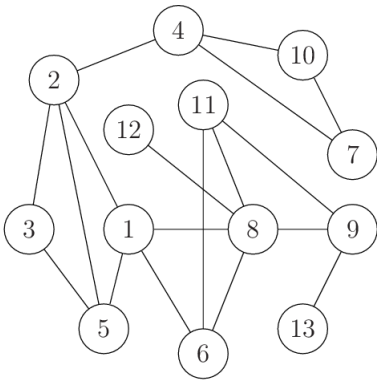


Figure 3: Graph G_3

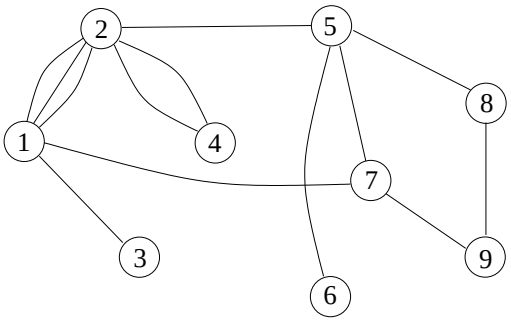


Figure 4: Graph G_4

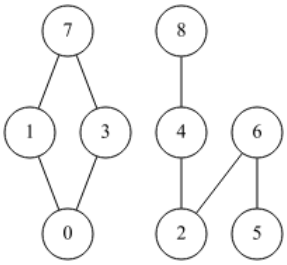


Figure 5: Graph G_5

A bipartite graph is a graph (multigraph) $G =< S, A >$ 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.

1. Are the graphs of figures 3 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 3.3 (Acyclic – *Final S3 - 2017*)

Définition :
An *acyclic* digraph is a graph without circuits.

Write the function `is_acyclic` that tests whether a digraph represented with adjacency lists is *acyclic*.

Exercise 3.4 (I want to be tree – *Final S3# - 2018*)

Définition :

A **tree** is an **acyclic connected** graph.

Write the function `isTree` that tests whether a graph is a tree.

Exercise 3.5 (Distances and center – *Final S3# - 2018*)

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).

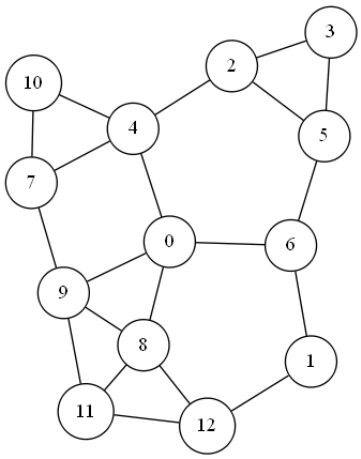
Write the function `center(G)` that returns the center of the graph G (a list).

In the graph G_6:

Vertices 0, 4 and 6 are of eccentricity 3.
Vertices 3 and 10 are of eccentricity 5.
Remaining vertices are of eccentricity 4.

The radius of G_6 is 3 and the vertices 0, 4 and 6 constitute its center.

```
>>> center(G6)
[0, 4, 6]
```



Graph G_6

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*.

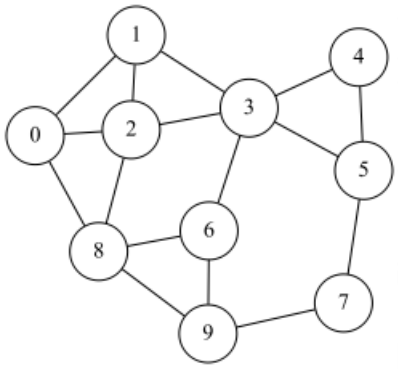
2. What property should have the graph so that a topological sort exists?
3. When the graph is drawn lining up the vertices in a topological order, what can be observed?
4. (a) Let $suffix$ be the array of the last encounter of the vertices: the suffix order during the depth-first tarversal.
   Prove 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?

Exercise 3.7 (What is this? – *Final S3# - 2018*)

Consider the following function:

```
1  def buildGraph(G, s, n):
2      map = [None] * G.order
3      dist = [-1] * G.order
4      NG = graph.Graph(1)
5      dist[s] = 0
6      map[s] = 0
7      q = queue.Queue()
8      q.enqueue(s)
9      while not q.isempty():
10         s = q.dequeue()
11         for adj in G.adjlists[s]:
12             if (dist[adj] == -1) and (dist[s] < n):
13                 dist[adj] = dist[s] + 1
14                 map[adj] = NG.order
15                 NG.addvertex()
16                 q.enqueue(adj)
17             if dist[adj] != -1:
18                 NG.addedge(map[s], map[adj])
19     return (NG, dist, map)
```



Graph G_7

1. This algorithm is called with `build_graph(`G_7`, 4, 2)` (G_7 the graph in figure 5).
   (a) Fill the array `dist`.
   (b) Fill the array `map`.
   (c) Draw the built graph (NG).
2. `build_graph(`G`, `s`, `n`)` is called with G any non-empty graph, s a vertex of G, and n a positive integer.
   (a) During the execution, what does the array `dist` represent?
   (b) During the execution, what is the array `map` used for?
   (c) After the execution, what does the graph NG represent?