Graphs (Graphes)
Implementations and traversals

1 Representations / Implementations

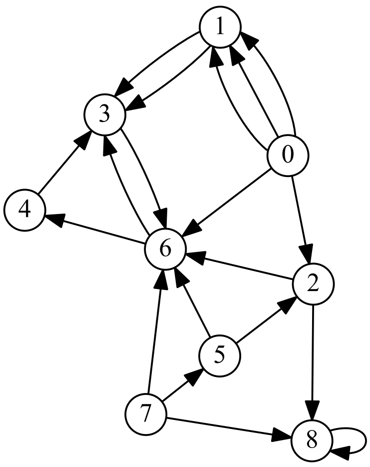

Figure 1: Digraph (Graphe orienté) G_1'

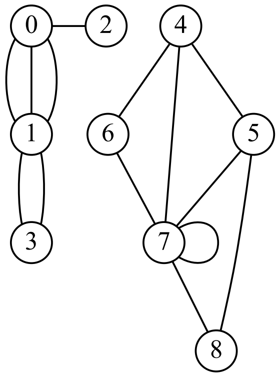

Figure 2: Graph (Graphe non orienté) G_2'

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 (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.

Exercise 1.4 (Degrees)

1. Write a function that fills two vectors *in* and *out* that represent respectively indegrees and outdegrees of all vertices of a digraph represented by adjacency lists.
2. **The degree** of a graph is the maximum value of its vertex degrees.

   The indegree of a digraph is the maximum value of its vertex indegrees.

   The outdegree of a digraph is the maximum value of its vertex outdegrees.

   Write a function that computes the indegree and the outdegree of a digraph represented by an adjacency matrix.

Exercise 1.5 (`dot`)

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

Examples:

- Graph 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
}
```

Bonus:
Write the functions that build a graph from a `".dot"` file (simplified).

Notes: 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 traversal)

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 traversal)

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, when the graph is directed and 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 (Distance from start)

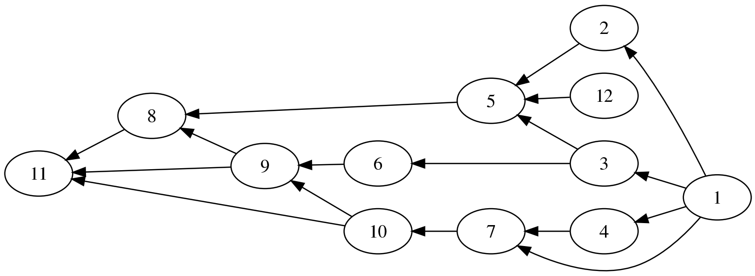

Figure 3: A graph

The aim here is to find the vertices that are at a distance in a range $[d_{min}, d_{max}]$ from a starting vertex. The result will be displayed: if possible one line per level.

1. Calculate the distances from vertex 1 in the graph of figure 3 (all the distances, without range).
2. Write the procedure `distances(`G`, `src`, `$dmin$`, `$dmax$`)` that displays vertices that are at a distance from $dmin$ to $dmax$ from the vertex src in the graph G (use the matrix implementation).

Exercise 3.3 (I want to be tree – *Final S3 2016*)

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

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

Exercise 3.4 (Diameter – *Final S3 2016*)

Définitions :

- The **distance** between two vertices in a graph is the number of edges in a **shortest path** connecting them.
- The **diameter** of a graph is the **highest distance** between any pair of vertices.

When the graph is a tree, computing the diameter is simple:

- From any vertex s_0, find a vertex s_1 whose distance from s_0 is maximal.
- From s_1, find a vertex s_2 whose distance from s_0 is maximal.
- The distance between s_1 and s_2 is the graph diameter.

Write the function `diameter` that computes the diameter of a graph that is a tree.

What if the graph is not a tree?

- Does the previous method give the diameter?
- If so, justify. Otherwise what should be done to find the diameter?

Exercise 3.5 (Compilation, cooking. . .)

1. *Scheduling, a simple example:*

   The following statements have to be executed with one processor:

   ① read(a)
   ② b ← a + d
   ③ c ← 2 * a
   ④ d ← e + 1
   ⑤ read(e)
   ⑥ f ← h + c / e
   ⑦ g ← d * h
   ⑧ h ← e - 5
   ⑨ 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.

5. **Bonus:** Write a function that tests whether a vertex list represents a topological order of a given digraph.

What about cooking?