

README
GRAPH THEORY IN PYTHON
graph_gen.py

Édouard Lumet <edouard.lumet@etu.enseeiht.fr>

11 février 2018

Index

Preamble	2
User's guide	3
Requirements	3
Features	3
Programming design	6
Graph modelling	6
BFS algorithm	6
Go further...	8
Dijkstra's algorithm	8
<i>pygraphviz</i> only	8

Preamble

Thanks for reading me!

The aim of this project is having a better understanding of graph theory and algorithms used in this field. To do that we have to write a program in the programming language of our choosing that asks a user for the number of vertices. Then there are few possibilities : create a cycle graph, create a complete graph, remove one vertex from the graph, run BFS algorithm on the graph and print the BFS tree.

I chose Python for this program because it is a "user-friendly" programming language that avoids non-graph problem such as pointer issues. Indeed the real questions were most *How to model a graph ?* or *How to implement algorithm like BFS ?*

In this documentation a user's guide is provided and a more technical section describes the design of the program with a few explanation about the choices made.

User's guide

Requirements

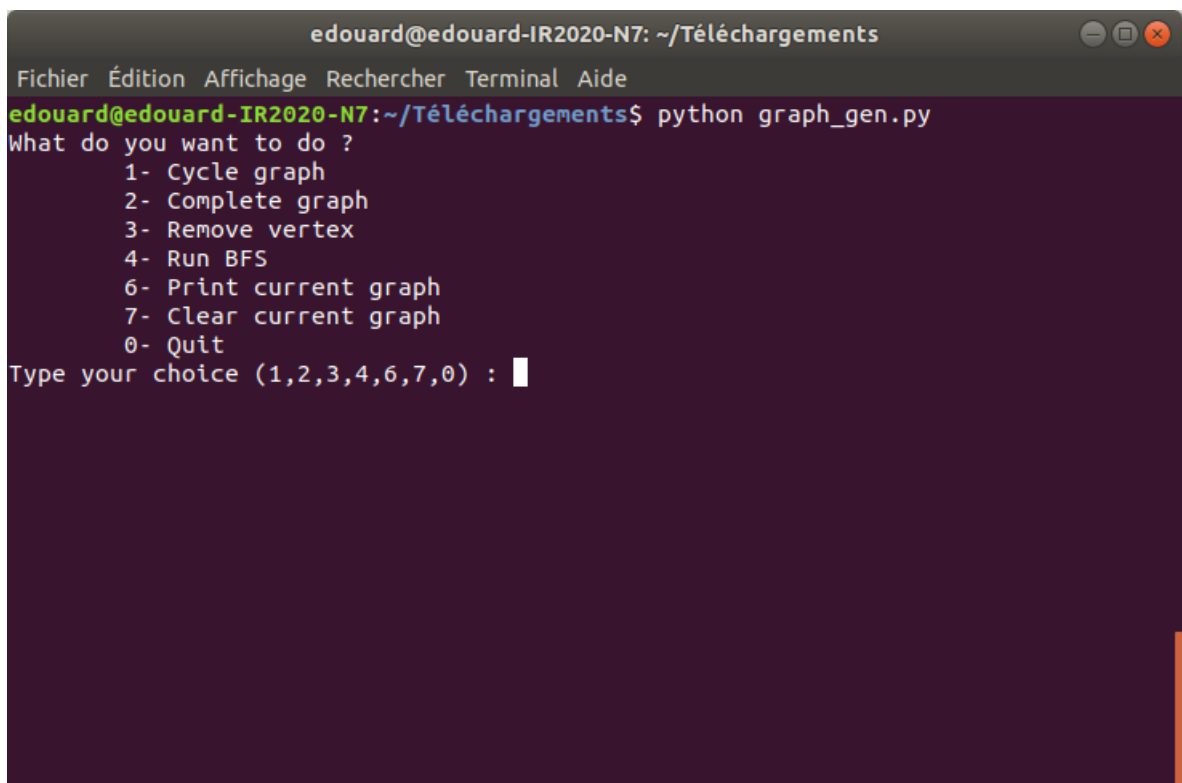
Python is an interpreted programming language so there is no need to compile the code. You just need Python 2.7 (default) installed and the *pygraphviz* library (which requires Python 2.7 only). On Linux platform with **apt**, type the following commands in a prompt :

```
sudo apt install python-dev graphviz libgraphviz-dev pkg-config
sudo pip install pygraphviz
```

And that's all you need !

Features

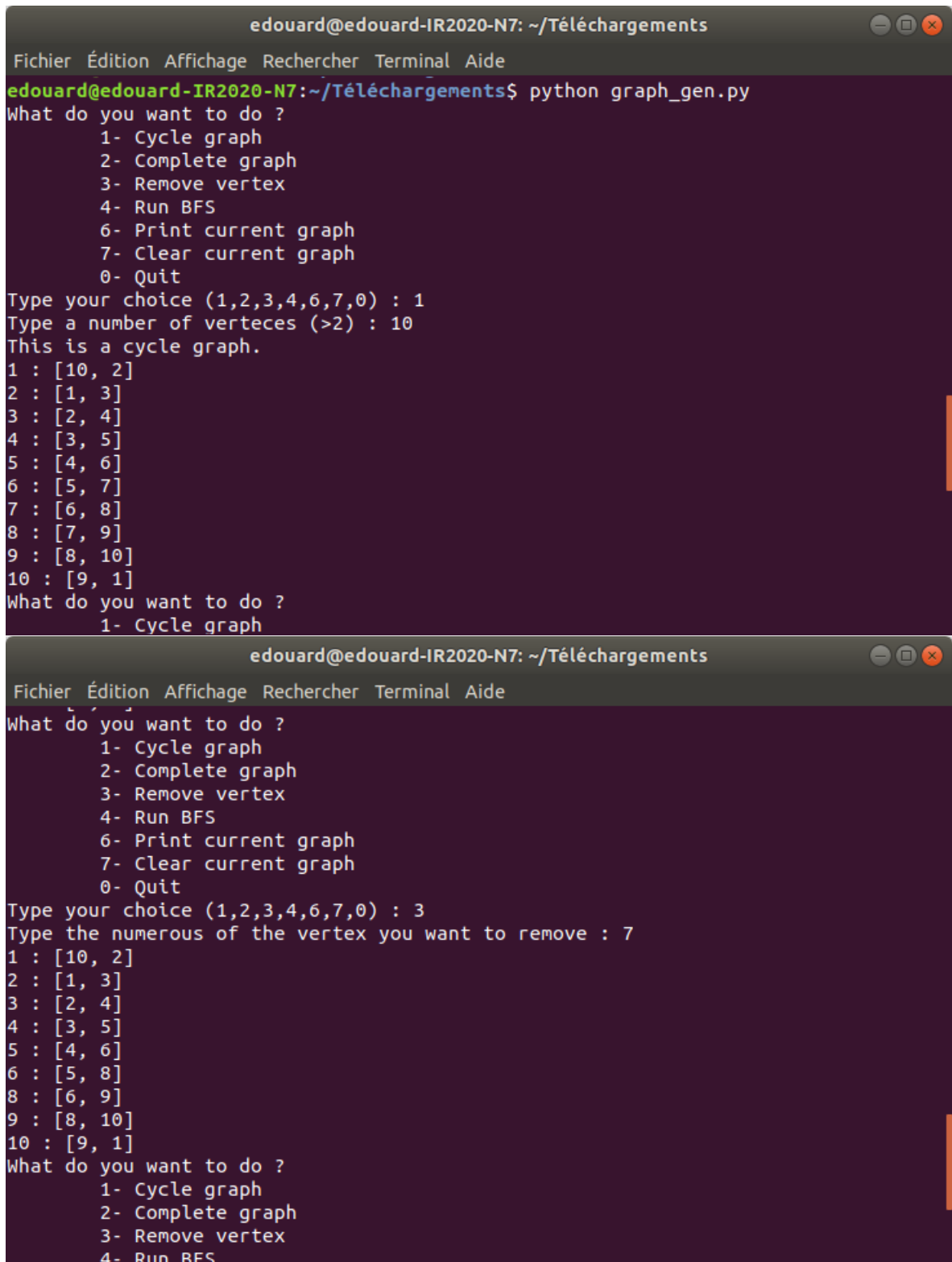
The following figure shows the menu that represents the different features offered by the program.

A screenshot of a terminal window titled "edouard@edouard-IR2020-N7: ~/Téléchargements". The terminal shows the command "python graph_gen.py" being executed. The output is a menu with the following options: "1- Cycle graph", "2- Complete graph", "3- Remove vertex", "4- Run BFS", "6- Print current graph", "7- Clear current graph", and "0- Quit". The prompt "Type your choice (1,2,3,4,6,7,0) :" is followed by a cursor. The terminal window has a menu bar with "Fichier", "Édition", "Affichage", "Rechercher", "Terminal", and "Aide".

To start, type 1 or 2 to generate either a cycle or a complete graph. Then you can remove one vertex, run BFS algorithm or print the graph. When you generate a graph it is text-printed. The same when you remove a vertex. Choosing option 6 will create a file named *graph.png* in the same directory as the *graph_gen.py* program. Running BFS will also create a file name *BFS_tree.png* so you will be able to view the BFS tree.

Before generating a new graph you MUST select option 7 to clear the graph. However you can save it by copying the *graph.dot* file but the feature that allows to manage graphs only with *pygraphviz* is not developed yet.

Here you have an exemple of what you can do and how to use the program. It's not very hard.



```
edouard@edouard-IR2020-N7: ~/Téléchargements
Fichier Édition Affichage Rechercher Terminal Aide
edouard@edouard-IR2020-N7:~/Téléchargements$ python graph_gen.py
What do you want to do ?
  1- Cycle graph
  2- Complete graph
  3- Remove vertex
  4- Run BFS
  6- Print current graph
  7- Clear current graph
  0- Quit
Type your choice (1,2,3,4,6,7,0) : 1
Type a number of verteces (>2) : 10
This is a cycle graph.
1 : [10, 2]
2 : [1, 3]
3 : [2, 4]
4 : [3, 5]
5 : [4, 6]
6 : [5, 7]
7 : [6, 8]
8 : [7, 9]
9 : [8, 10]
10 : [9, 1]
What do you want to do ?
  1- Cycle graph

edouard@edouard-IR2020-N7: ~/Téléchargements
Fichier Édition Affichage Rechercher Terminal Aide
What do you want to do ?
  1- Cycle graph
  2- Complete graph
  3- Remove vertex
  4- Run BFS
  6- Print current graph
  7- Clear current graph
  0- Quit
Type your choice (1,2,3,4,6,7,0) : 3
Type the numerous of the vertex you want to remove : 7
1 : [10, 2]
2 : [1, 3]
3 : [2, 4]
4 : [3, 5]
5 : [4, 6]
6 : [5, 8]
8 : [6, 9]
9 : [8, 10]
10 : [9, 1]
What do you want to do ?
  1- Cycle graph
  2- Complete graph
  3- Remove vertex
  4- Run BFS
```

```
edouard@edouard-IR2020-N7: ~/Téléchargements
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
5 : [4, 6]
6 : [5, 8]
8 : [6, 9]
9 : [8, 10]
10 : [9, 1]
What do you want to do ?
  1- Cycle graph
  2- Complete graph
  3- Remove vertex
  4- Run BFS
  6- Print current graph
  7- Clear current graph
  0- Quit
Type your choice (1,2,3,4,6,7,0) : 4
Type the numerous of the source vertex to run BFS : 6
What do you want to do ?
  1- Cycle graph
  2- Complete graph
  3- Remove vertex
  4- Run BFS
  6- Print current graph
  7- Clear current graph
  0- Quit
Type your choice (1,2,3,4,6,7,0) : █
```

Programming design

Graph modelling

In this program, dictionary is used to model graphs because its structure is quite interesting. We define it as follows : the key is an integer and represents one vertex, the value is a list and represents the neighbors of this vertex (adjacency list).

When creating a graph we just have to iterate from the first vertex to the number given by the user to create the dictionary. For a cycle graph neighbors are the previous vertex and the next vertex except for the first and the last vertices which are neighbors. For a complete graph adjacency list is the list of all the vertices except the vertex itself so it is easy to build it.

When removing a vertex in a complete graph we just have to remove the *key :value* from the dictionary then to cross all the vertices and to remove the vertex in every adjacency list. In a cycle graph we have to modify both adjacency lists of the previous and the next vertices. For instance when removing B which is between A and C we have to modify A next neighbor for C and to modify C previous neighbor for A. Then just remove B from the dictionary.

Therefore we see that dictionaries are useful and easy to use in graph theory. You can even find an example in the Python doc : <https://www.python.org/doc/essays/graphs/>

BFS algorithm

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  from collections import deque
4
5  # This function runs BFS on a graph
6  def bfs(vertex):
7      """
8          Call this function with the numerous of a vertex to run BFS algorithm
9          on the graph you generated from this vertex.
10     """
11     queue = deque([]) # LIFO, see line 3
12     queue.append(vertex)
13     VISITED[vertex] = True
14     while queue != deque([]):
15         vertex = queue.popleft()
16         BFS_TREE.add_node(vertex)
17         for neighbor in GRAPH[vertex]:
18             if not VISITED[neighbor]:
19                 queue.append(neighbor)
20                 VISITED[neighbor] = True
21     return BFS_TREE

```

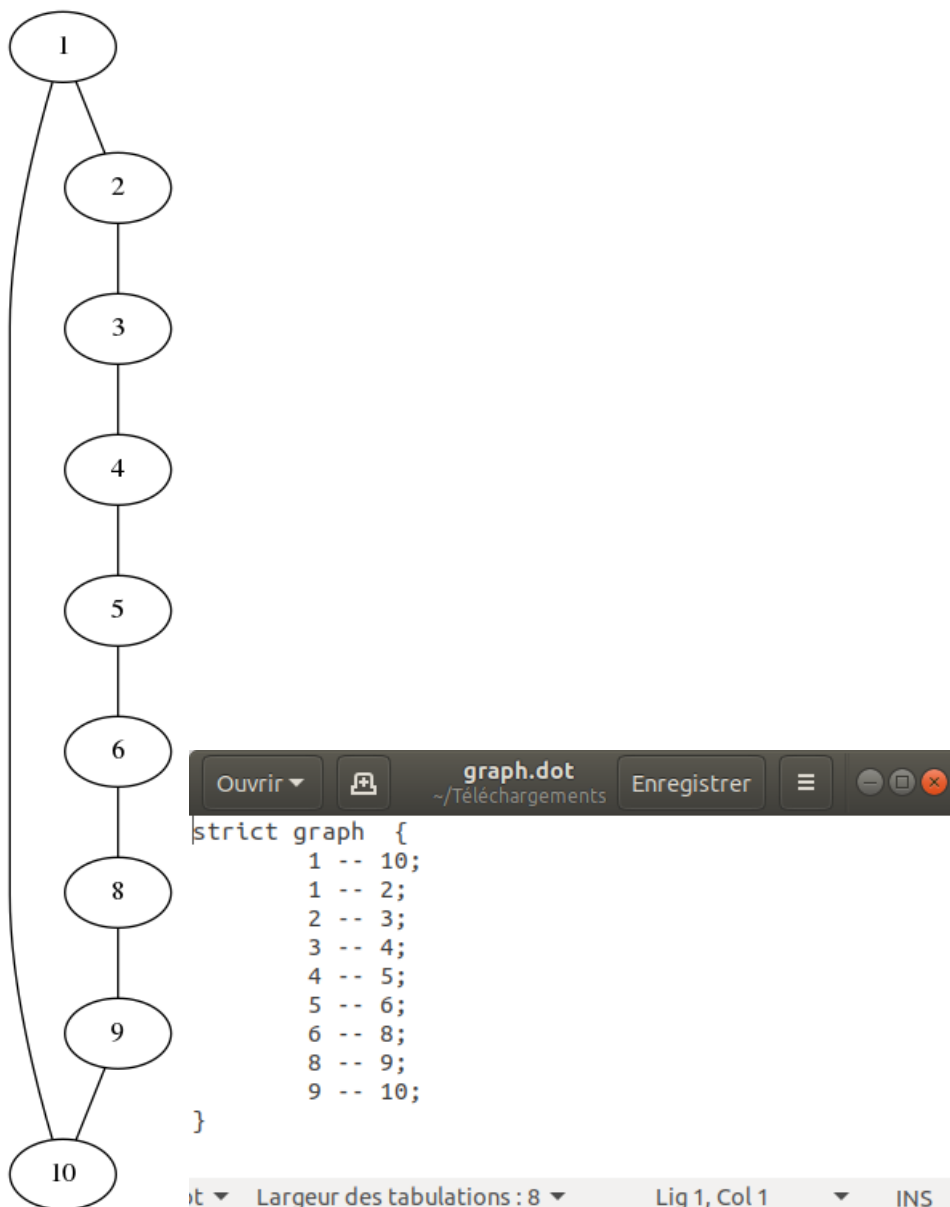
This algorithm is quite simple. It is based on a queue that can be a list or a real LIFO (*deque* from *collections*). First we have to add the source vertex in the queue and to mark

it. This is why we use a global dictionary called VISITED that has vertices as keys and booleans as values (True means the vertex is marked/visited). Then while the queue is not empty we dequeue it and add the vertex in the BFS tree. After that for each vertex in the adjacency list the vertex recently added we add the non visited yet neighbors in the queue and mark them.

It is very easy to implement BFS in Python.

Quick *pygraphviz* overview

pygraphviz is a library that permits to draw a graph. Some functions like `ADD_NODE()`, `ADD_EDGE()`, `WRITE()` and `DRAW()` are specific to this library. They can add a single node or an edge then write it in a dot script. It can also draw it in a png file.



Go further...

Dijkstra's algorithm

An other feature to implement could be the Dijkstra's algorithm. It requires positive weights on the edges. This algorithm is the most efficient and the quickest to find the shortest path from one vertex to another vertex in a graph.

First we have to put the distance of each vertex to infinity except the source vertex which has a distance of 0. We also use the VISITED dictionary. Then while all the vertices are not visited we select the neighbor with the least distance and mark it. For each of its neighbors we sum up its distance and the cost between the neighbor and itself. If the sum is lower than the distance of the neighbor, this sum will be the new distance of the neighbor and we add it in the tree.

The least distance can be computed with Prim's algorithm which is a minimum spanning tree algorithm.

pygraphviz only

A last interesting feature to implement could be the use of *pygraphviz* only for graph modelling. No more need to use a dictionary to model a graph and you can write a custom dot script and load it from the program.