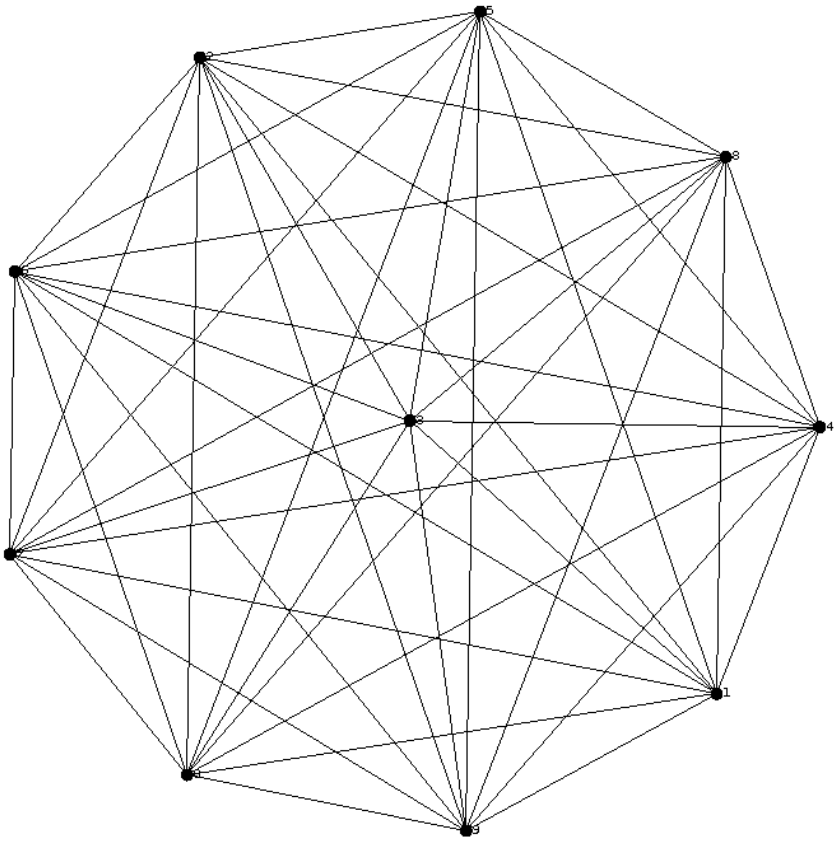
Graph Theory Project

By Lucas Salvato



INP TOULOUSE ENSEEIHT

Table of Contents

Introduction

This project was realized in second period during the Graph Theory class at Enseeiht. You will be able to find the whole program in my github(https://github.com/lonaxous/JaGraph). I have been working on that project for about 3 weeks being supervised by Mr.Jakllari Gentian. Let me introduce to you the subject.

The Subject

Write a program in the programming language of your choosing that :

1. Asks a user for the number of vertices and then can create either a cycle or complete graph.

2. Can remove a single vertex from a graph it has creates (vertices are identified by a number between 0 and n-1, with n the number of vertices).

3. Runs BFS on the graphs it has created and prints the BFS tree.

4. (Bonus) Assigns weights at random to the edges, runs Dijkstra and prints the shortest path tree.

The Choose Language

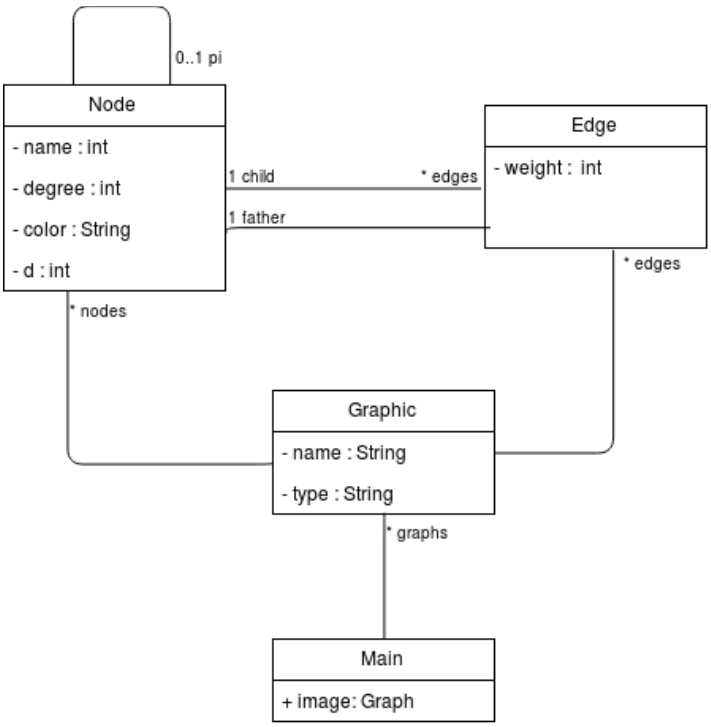
The programming language chosen is Java. Indeed, the subject is in correlation with object-oriented language. Moreover, I am very comfortable with Java that I have already used in others projects.

The program is composed by 4 classes :

- Node
- Edge
- Graphic, The Graph Class
- Main, The main program

Java Logo

Graphstream Logo



Class Diagram

Color, pi, d and weight are used in the BFS and Dijkstra algorithm.

The project uses the library GraphStream to print graphically graphs. That is why the Graph class does not use the name “Graph”.

Installing

Search the executable jar in this link :
https://github.com/lonaxous/JaGraph/raw/master/JaGraph.jar

Go to the repertory where you download this file, and use the command

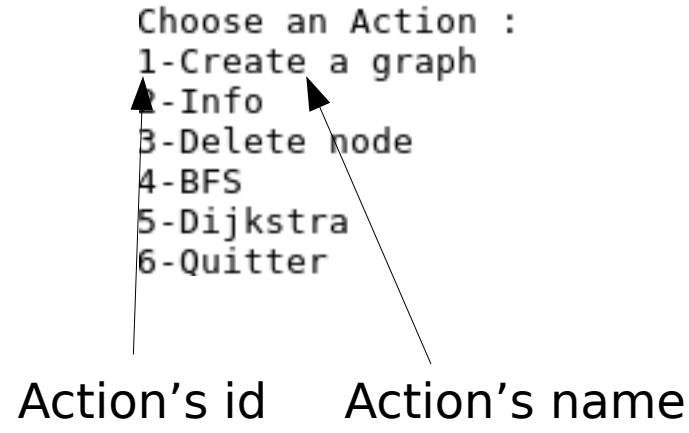
java -jar JaGraph.jar

Instruction For Use

During the program , you must choose action with a terminal menu. That is really easy.

You just need to write the action’s id on the terminal menu in order to select the action.

Every time the user will have to make a choice, that kind of menu will be proposed to him.



Menu Example

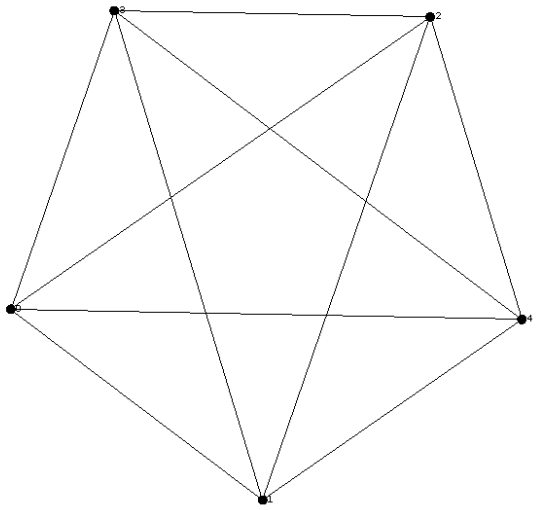
All The Application’s Actions

Create a Graph

In this application you can create two kinds of graphs with the number of nodes of your choice.

Complete

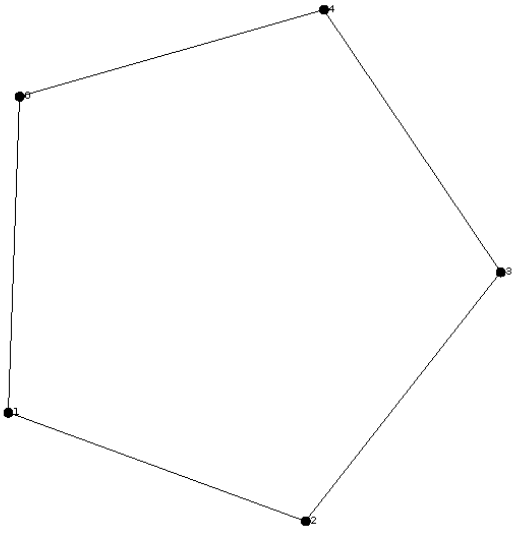
A complete graph is composed by vertices and line segments. Actually, for one hand every line segment joints two vertices. On the other hand, every pair of vertices is connected by a line segment.



Complete graph with 5 nodes

Cycle

Cycle or circular graph is a graph which consists of a single cycle. Every vertex has degree 2.

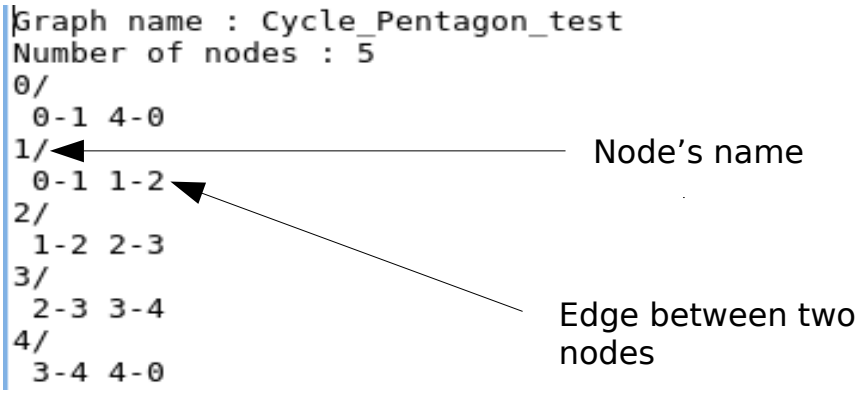


Cycle graph with 5 nodes

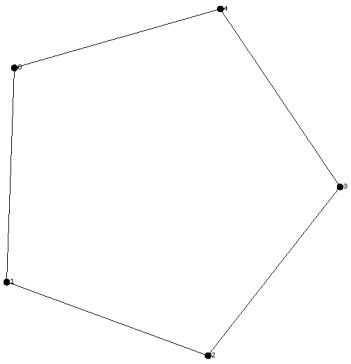
Info

You can print the informations of your graph choice. You are going to find on this panel informations like :

- The Graph’s name;
- The Number of nodes;
- The edges per nodes (“Node 1” - “Node 2”).



Information Panel example



Cycle graph with 5 nodes

Delete Node

In that program you can suppress in the graph of your choice on of its vertices. In a complete graph the suppression is really simple. You just need to suppress the node and its edges. However, in a circle graph the action is harder. Actually, it is necessary to suppress the node and its edges. Moreover, you will need to link the 2 neighbors of the suppressed nodes.

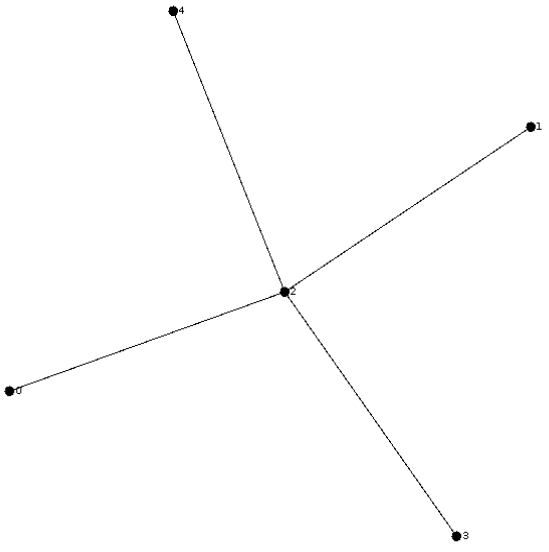
BFS(Breadth First Search)

BFS is a way to find all the vertices reachable from the a given source vertex. BFS goes throught a connected component of a given graph and defines a spanning tree.

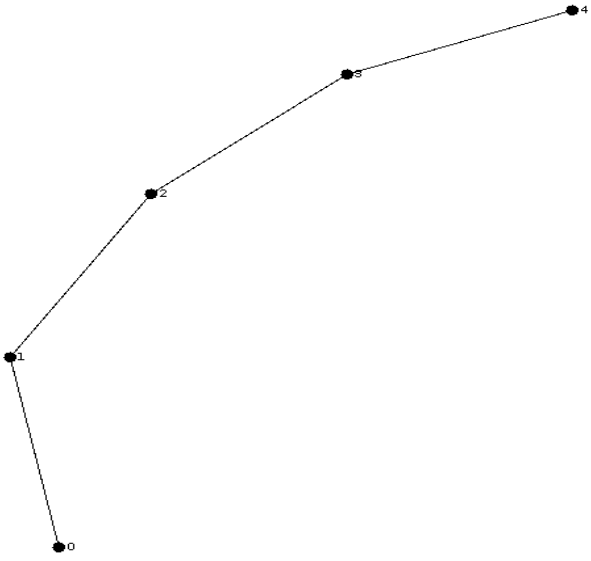
BFS starts with the chosen vertex, which is at level 0. In the first stage, we are going to visit all the vertices which are at one edge of distance. When we will visit there, we paint it as ”visited” the vertices adjacent to the start vertex – these vertices are placed into level 1. In the second stage, we are going to visit all the new vertices we can reach at the distance of two edges away from the source vertex. These new vertices (which are adjacent to level 1 vertices and not previously assigned to a level) are placed into level 2, and so on. The BFS will be finished when every vertex will have been visited.

```
public void bfs(int i, Graph image){
    ArrayList<Node> queue = new ArrayList<Node>();
    Node u; //Actual vertex
    Node s = nodes.get(i); //The source vertex
    Node v; //Neighbour vertex
    //initialise all the nodes
    for(int cpt=0;cpt<nodes.size();cpt++){
        if(i!=cpt){
            u = nodes.get(cpt);
            u.setColor("white");
            u.setD(1000);
            u.setPi(null);
        }
    }
    s.setColor("gray");
    s.setD(0);
    s.setPi(null);
    queue.add(s);
    //Terminate when all the nodes are visited
    while (queue.size()!=0){
        //Obtain the next vertex to visited
        u = queue.get(0);
        queue.remove(0);
        //Check his neighbour
        for (int j=0;j<u.edges.size();j++){
            v = u.edges.get(j).getNeighbour(u);
            //If we find a new node, add to the queue
            if(v.getColor()=="white"){
                v.setColor("gray");
                v.setD(u.getD()+1);
                v.setPi(u);
                queue.add(v);
            }
        }
        u.setColor("black");}
```

BFS algorithm



BFS on a complete graph, 2 is the source vertex



BFS on a cycle graph, 2 is the source vertex

Dijkstra

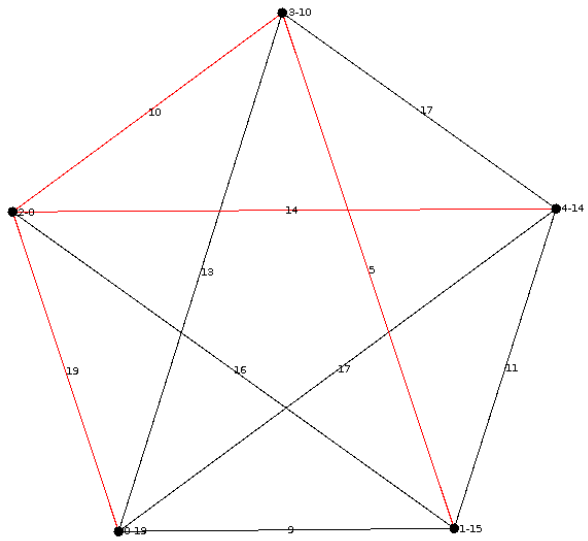
This algorithm enables us to find the shortest distances for a minimum costs. Here we are going to explain to you the use of this algorithm. Start at the ending vertex by marking it with a distance of 0. Indeed, it is 0 units from the end. Then, call this vertex your current vertex, and find this neighbor vertices. Identify all of the vertices that are connected to the current vertex with an edge. Now you can calculate their distance to the end by adding the weight of the edge to the mark on the current vertex. Mark each of these vertices with their corresponding distance, but change a vertex's mark if it is smaller than a previous mark. Each time you mark the starting vertex, keep tracking the path that results in that mark. Label the current vertex as visited by putting an X over it. Once a vertex is visited, we won't look at it again. Among the vertices you have just marked, find the one with the smallest mark, and make it your current vertex. Start again from step 2, finding the neighbor vertices. Once you have labeled the beginning vertex as visited, stop it: the distance of the shortest path is the mark of the starting vertex, and the shortest path is the path that resulted in that mark.

```
public void dijkstra(int i,Graph image){
    initialiseSource(i); //Dijkstra initialise
    ArrayList<Edge> es; //The edge list
    ArrayList<Node> Q = new ArrayList<Node>(); //The copy of the node's list
    Q.addAll(nodes);
    Node u;
    //Randomize the weight of edges
    for(int cpt=0;cpt<edges.size();cpt++){
        edges.get(cpt).setWeight((int)(Math.random() * 20));
    }
    //Check all the edges and relax
    while (Q.size() != 0){
        //Take the minimum weight edge
        u = extractMin(Q);
        Q.remove(u);
        es = u.getEdges();
        //Relax the neighbours edges
        for(int cpt=0;cpt<es.size();cpt++){
            relax(u,es.get(cpt).getNeighbour(u),es.get(cpt));
        }
    }
```

```
//Extract the minimum weight edge
public Node extractMin(ArrayList<Node> q){
    Node min= q.get(0);
    for(int cpt=0;cpt<q.size();cpt++){
        if (min.getD()>q.get(cpt).getD()){
            min = q.get(cpt);
        }
    }
    return min;
}

//Relax an edge
public void relax(Node u, Node v, Edge e){
    if (v.getD() > u.getD() + e.getWeight()){
        v.setD(u.getD()+e.getWeight());
        v.setPi(u);
    }
}
```

Dijkstra Algorithm



Dijkstra 2 is the source vertex (the shortest path in red)

Conclusion

As a conclusion we can say that the graph theory project has been really interesting for me. Indeed, practicing with these 2 algorithms presented in that report make me more comfortable with these tools. Actually, currently I think I am able to understand precisely their working principles.

I really enjoy to have the choice of the language to make that project. Using Java in a field I didn’t know at all allows me to gain experience with that language.

In the future I am sure I will link the knowledges learned with than project with new knowledges such as artificial intelligence.

Importants Links

Project github : https://github.com/lonaxous/JaGraph

Executable jar link : https://github.com/lonaxous/JaGraph/raw/master/JaGraph.jar

Graphstream library : http://graphstream-project.org/