AG44 - Project 1

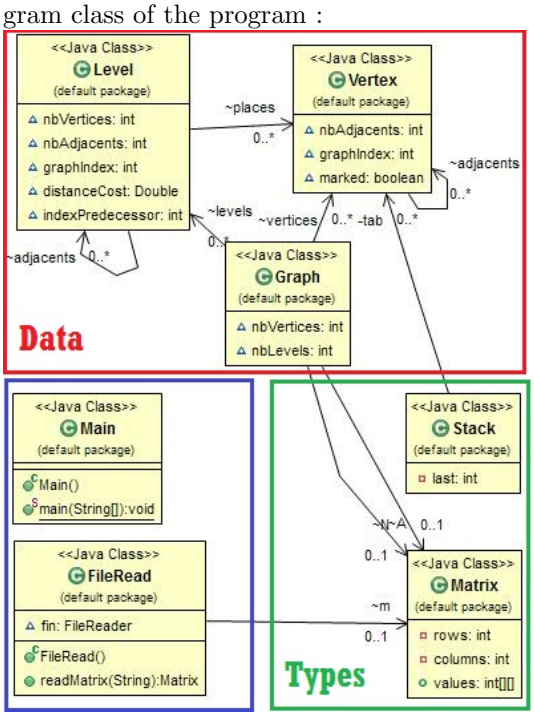
JAFFALI Hamza - 28 october 2014

Abstract

This document constitutes the report of the first project as a part of the UV AG44, a project named “Ariadne’s thread”. We consider a game, composed of differents places that can be reached by other places according to a matrix given by the “player“. We can also regroup places in order to form levels. We chose to represent this game by means of a graph, in which every place will be a vertex. This graph must be directed, because we suppose that if we can pass from a place to another, it does not necessarily mean that we can return back to the place we came from. There is always a path between the first and the last place, because it is supposed that one can always finish the game.

1 Data architecture

We choose in this project to use the Java programming language. So 7 classes have been defined accordingly : Graph, Vertex, Level, Matrix , Stack, FileRead and Main. Below is the diagram class of the program :



A graph contains a list of vertices (places) and levels. A level contains a list of vertices (places) and a list of adjacents levels. A vertex (place) contains also a list of adjacents vertices (places).

2 Listing game levels

In this project, the problem of listing the differents levels proper to the game leads us to resolve common issue of graph theory : determining the strongly connected components of the graph representing the game.
We chose here to implement the Kosaraju's algorithm, using the Depth-First Search algorithm to traverse the graph.
In fact, we first apply the DFS in order to determine the post-ordering traversal in the graph. Once this has been achieved, the graph is inverted, so we implement the complement of the graph, and then we apply it the DFS, following the order given by the post-ordering. Each time the DFS stops at a vertex, it means that we had traversed entirely a level, and we will start the DFS from an unvisited vertex, always according to the post-ordering, and this, until we traverse all the graph.
Until we determine all levels of the graph, these levels are stored in a ArrayList, which is an attribute of the graph.

3 Reduced matrix N

The reduced matrix N is a kind of cost adjacency matrix of levels in the game. $\forall (i,j) \in L$, the set of all levels, $N_{ij} = k$ if there is k direct passage (possibly $k = 0$) from level i to level j. N is a square matrix, which size is the number of levels in the game.
In ordrer to fill the matrix N, for each place in each level of the graph, if we can go to this place to another place not in the same level, thus, so we increment the coefficient in the matrix N which his line is the index of the level that reach, and his column is the index of the level that is reached.

4 Longest path

Our friend asked us to find the longest path between two specific levels : the starting level, containing the frist place (place 1), and the ending level, containing the last place (place n).
In order to compute the solution for this longest path, a recursive version of the Label-Correcting algorithm is used for so as to find the shortest path.
We start the algorithm by putting all distances at $+\infty$ for each level, except for the level 1. Then we call a recursive fuction called ***visit(list, level)***, with parameters an empty list and the first level of the game. The ***visit*** fuction is therefore defined as follows :

```
begin visit
  add level to list;
  for each successor u of level
  with u not in the list
    if d(s,u) = +infinity or
    d(s,u) < d(s,level) + cost(level-->u)
    then
      d(s,u) = d(s,level) + cost(level-->u);
    end if;
    visit (list, u);
  end for;
  remove level from list;
end visit;
```

We show by induction that when we ***visit*** a level u not in the list, the list contains the sequence of vertices of an acyclic path from s, the first level, to this level u. Therefore, the algorithm is convergent. In the worst case, ie when the graph is complete, the number we call the ***visit*** fuction is the number of acyclic paths starting from s, ie :

$$\sum_{k=1}^{|V|}(|V|-1)(|V|-2)...(|V|-(k-1))$$

Consequently, the complexity of the algorithm is $O(|V|!)$

5 Examples

```
 ----- Beginnig of display -----

Place's adjacency lists display :
Place 1 adjacent to : { 2 }
Place 2 adjacent to : { 3 4 }
Place 3 adjacent to : { 2 5 7 }
Place 4 adjacent to : { 5 8 }
Place 5 adjacent to : { 4 8 }
Place 6 adjacent to : { 2 8 }
Place 7 adjacent to : { 6 8 }
Place 8 adjacent to : { }

Levels display :
Level 1 : { 1 }
Level 2 : { 2 6 7 3 }
Level 3 : { 5 4 }
Level 4 : { 8 }

Level's adjacency lists display :
Level 1 adjacent to : { 2 }
Level 2 adjacent to : { 3 4 }
Level 3 adjacent to : { 4 }
Level 4 adjacent to : { }


Adjacency matrix of places :

0 1 0 0 0 0 0 0
0 0 1 1 0 0 0 0
0 1 0 0 1 0 1 0
0 0 0 0 1 0 0 1
0 0 0 1 0 0 0 1
0 1 0 0 0 0 0 1
0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0

Reduced matrix N of levels :

0 1 0 0
0 0 2 2
0 0 0 2
0 0 0 0

The longest path between Level 1
and Level 4 is [1 -> 2 -> 3 -> 4]
and it costs 5.

 ----- End of display -----
```

```
 ----- Beginnig of display -----

Place's adjacency lists display :
Place 1 adjacent to : { 2 3 }
Place 2 adjacent to : { 4 5 }
Place 3 adjacent to : { 2 4 8 }
Place 4 adjacent to : { 1 7 }
Place 5 adjacent to : { 6 }
Place 6 adjacent to : { 7 }
Place 7 adjacent to : { 5 10 }
Place 8 adjacent to : { 9 }
Place 9 adjacent to : { 8 10 }
Place 10 adjacent to : { 11 }
Place 11 adjacent to : { 12 }
Place 12 adjacent to : { 13 }
Place 13 adjacent to : { 10 }

Levels display :
Level 1 : { 1 4 2 3 }
Level 2 : { 8 9 }
Level 3 : { 7 6 5 }
Level 4 : { 10 13 12 11 }

Level's adjacency lists display :
Level 1 adjacent to : { 2 3 }
Level 2 adjacent to : { 4 }
Level 3 adjacent to : { 4 }
Level 4 adjacent to : { }


Adjacency matrix of places :

0 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0
0 1 0 1 0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0

Reduced matrix N of levels :

0 1 2 0
0 0 0 1
0 0 0 1
0 0 0 0

The longest path between Level 1
and Level 4 is [1 -> 3 -> 4] and
it costs 3.

 ----- End of display -----
```