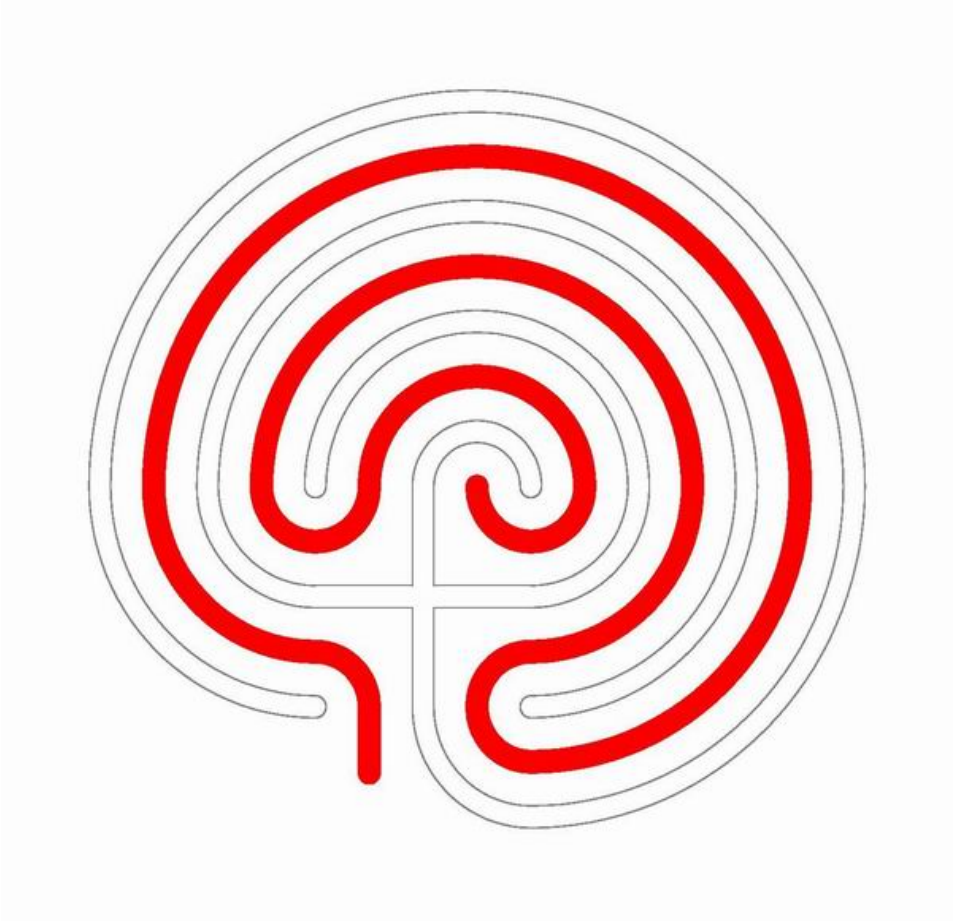
AG44 – Project 1

Ariadne's thread



SCHULZ Quentin
AG44T01GRB

GI01 A13
25/10/13

Resume of the project

→ Given a matrix illustrating a game map, give all strongly connected components, each representing a level, give the reduced matrix associated to these strongly connected components and try to find the longest path from the beginning to the end of the game.

Identification of the problem

→ The classical problem of graph theory to deal with separation of places into game levels is to find the strongly connected components in the square matrix Mi,j representing direct edges from place i to place j.

Data structures

Vertex
- children : List of pointers on a child (Vertex) - parents : List of pointers on a parent (Vertex) - _vertex_number : number of the vertex (int) - marked : state (if visited or not) (boolean)

Level
- lvl : List of pointers on a place (Vertex) - distance : distance from starting place (int) - marked : state (if visited or not) (boolean)

StronglyConnectedComponents
- levels : List of pointers on a level (Level) - reducedMatrix : List of list of edges (int)

Main algorithms

→ *Strongly connected components*
I directly used the algorithm provided in this course. We perform a Depth-first Search (DFS) and each visited vertex is marked. When we reach a dead-end, we push this vertex into a stack. Perform it until all vertices are marked.
While the stack is not empty we add to the list of levels the level which is the return of the method getLevel.
This method take a vertex and the stack as arguments. If the list of vertices in this level is empty, we pop each marked vertex in the stack until we find an unmarked one or the stack is empty. The first unmarked vertex is put in this list. If the list is not empty, we add the vertex passed by parameter to the list and we delete this vertex from the stack. We mark the last vertex added to the list and for each of its unmarked parents, we recursively call the getLevel method.

→ *Longest Path*
We call the createLongestPath method of the StronglyConnectedComponents class in which we pass the index minus 1 of each level (beginning and end) and a counter initialized at 0. By default, all distances are set to -1. We begin with checking if each index is possible and if the distance of the vertex is different from 0. If it's not, then we felt in the starting vertex therefore we don't have to continue.
If the count equals 0 then it means than we are in the first occurrence of the recurrence call, or in other words, at the starting place. In this case, the distance is set to 0 and the vertex is marked. Afterwards, for each of its children we recursively call the same method but with the index of the child and the count incremented.

If the count doesn't equal 0 then we must test if the distance of the vertex is inferior to the counter. If not, we stop here. Otherwise, we set the distance of the startLevel vertex to count and we mark it. Then if the startLevel is not the endLevel in the current occurrence of the method, we recursively call it for each of its unmarked child but with the index of this child and the count incremented.
We end the method by unmarking the current startLevel vertex.
This method only sets each vertex's distance. For finding the path, we just need to traverse the matrix from the end level (maximal distance) to the start level (distance 0). In this perspective, we are looking for the vertex whose distance is maximal distance minus 1. Then the vertex whose distance is maximal distance minus 1 minus 1 and this, until we found the vertex whose distance is 0. Each time we find the right vertex, we add it in a queue.
The answer for the provided matrix is 1→2→4 (strongly connected component number).

→ *Reduced matrix*

```
void createReducedMatrix()
        temp <= List of edges starting from a same level
        tempValue <= number of edges from a same level to a fixed other level
        for each levelH in levels
                for each levelV in levels
                        tempValue <= 0
                        if (levelH!=levelV)
                                for each vertexH in levelH
                                        for each vertexV in levelV
                                                if (vertexH is parent of vertexV)
                                                        tempValue <= tempValue+1
                                                endif
                                        done
                                done
                        endif
                        temp.add(tempValue)
                done
                reducedMatrix.add(temp)
        done
```

Base matrix

01. 0 1 1 0 0 0 0 0 0 0 0 0 0
02. 0 0 0 1 1 0 0 0 0 0 0 0 0
03. 0 1 0 1 0 0 0 1 0 0 0 0 0
04. 1 0 0 0 0 0 1 0 0 0 0 0 0
05. 0 0 0 0 0 1 0 0 0 0 0 0 0
06. 0 0 0 0 0 0 1 0 0 0 0 0 0
07. 0 0 0 0 1 0 0 0 0 1 0 0 0
08. 0 0 0 0 0 0 0 0 1 0 0 0 0
09. 0 0 0 0 0 0 0 1 0 1 0 0 0
10. 0 0 0 0 0 0 0 0 0 0 1 0 0
11. 0 0 0 0 0 0 0 0 0 0 0 1 0
12. 0 0 0 0 0 0 0 0 0 0 0 0 1
13. 0 0 0 0 0 0 0 0 0 1 0 0 0

Strongly connected components:

1. 1 2 3 4
2. 8 9
3. 5 6 7
4. 10 11 12 13

Reduced matrix N

1. 0 1 2 0
2. 0 0 0 1
3. 0 0 0 1
4. 0 0 0 0