Olff Nathan
GI01
Dec. 2013

AG44 : Report

Ski resort

Goals of the project

The purpose of this project is to implement a shortest path algorithm on a graph and some of its subgraphs. The graph in this project represents a ski resort of many stations and many roads (pistes). The user will be able to select a source, a destination and get the shortest path between them using either any of the pistes, or only those he wants to use (for example : no black piste for beginners).

Data structures

I choose to represent the graph as an adjacency matrix because it is very useful to apply either Dijkstra's algorithm (better complexity) or Floyd Warshall's. Within the matrix, I store only the length of the shortest edge between the two vertices (because we can have many roads connecting directly A to B). In the *Graph* class, there is also a list of *Vertex* (used to get the vertex' name) and an list of *Edge* (used to store informations about the type of road).

A special constructor is generating a subgraph containing all the vertices from the original graph and only some of its edges.

To compute shortest path's algorithms, I used a class named *ShortestPath* (for Dikjstra) and *FloydWarshall*. Both have a reference to the graph it is linked to.

The *FloydWarshall* class contains the matrix that is traversed by the algorithm and an other matrix to keep track of the intermediate vertices (and therefore allow a path reconstruction).

The *ShortestPath* (using Dikjstra's algorithm) has 2 sets, an *openSet* and a *closedSet*. It also has an simple array used to get path reconstruction.

Algorithms

Depth-first search

Here it can be used in the second question, to get the accessible points from a specific place. Applied to a subgraph, it can give the accessible points using only some type of edges (with user's restrictions).

Description of the algorithm : We begin the traversal from a vertex sent in parameter. From there, we call recursively the DFS on each of its children (adjacency vertices). Each time a node is visited by the program, the vertex is marked as explored and added to a queue, so that we don't traverse the same node multiple times. Each time the DFS finished expanding a vertex, we push it on a stack S. Therefore we have the list of node traversed in a post-order relation.

Floyd Warshall's shortest path

Floyd Warshall is a pretty heavy algorithm with a complexity of $O(n^3)$. Why am I using it in my project ? Because once the calculation has been done once, the program can give the shortest path between any pair of two points, so it doesn't have to do the calculation several times.

In practice, the matrices resulted from our Floyd Warshall could be stored directly within the software because we can assume that the map won't change very often.

Dijkstra's algorithm

Dijkstra's algorithm compute all shortest path from one source to every other vertices on the graph.

The fact is that I already had implemented Dijkstra's for a previous personal project, so i wanted to use it again on the subgraphs. There is many possibilities in combining routes types, so it wouldn't be appropriate here to compute every combination with a Floyd Warshall.

To implement it, I created a priority queue for the *openSet* in order to get fastest access to the element with the lowest priority. In our example the priority is the distance from the source to a given vertex.

Questions

1. The problem described here is a shortest path problem. I choosed to implement **Floyd Warshall**'s algorithm in order to save computations if we want to find paths between several pairs of places.

2. Restricting the types of pistes is just like creating a subgraph of the graph with fewer edges and the same vertices. To get the accessible place from a given place, we only have to do a **depth-first** (or breadth-first) traversal of that subgraph. As I explained before, I already had a Dijkstra's algorithm implemented in C++, so I also give the user the possibility to get the accessible places and the shortest paths to go to each of them.