Conception and Implementation of Spatial Analysis Methods

Homework No.10

Raptakis Giorgos
MN: 374030

Contents

Description of Algorithm

In a given graph we implement Dijkstra algorithm so as to find the shortest path from one start vertex (source) to all other vertices.

We use the graph of Lab 9, so for the implementation of Dijkstra algorithm, we store weighted graph in three classes Graph, Vertex, Edge and also we make use of the created functions.

To solve the shortest path we use priority que, `priority_queue<Vertex*, vector<Vertex*>, CompareVertices> pq;`

We create the function: `void Dijkstra(Vertex* StartVertex, Graph &g)` ,as arguments accepts the start vertex and the class vertex.

Every vertex contains a short distance which as initial value is infinitive, only the value of the source vertex is equal of zero. The value of a vertex, short distance change when the algorithm find a shorter value than the current

We push the source vertex to the priority que and give to short distance value equal of zero. Then we make a while loop that take as arguments the features of priority que.

In the first vertex of the que, we make a for- loop to its list of neighbor edge. In every edge we take its weight and the second vertex.

To find the second vertex of an edge we use the function `Vertex* neighbor_ofaVertex(Vertex* v, int Weigh)`, which has as arguments a vertex and a neighbor edge's weight and return the other vertex.

When we have the second vertex compare if its value of short distance is smaller than the cumulative of the shortest distance of first vertex and their weigh of the edge. If the statement is true, then the shortest distance of second vertex take the cumulative value as new shortest distance.

Then if the second vertex has not entered till that time to the priority que we push it so as to examine its neighbors also.

This procedure repeated for all vertices and its neighbor's edges till the que to get empty.

Finally, we print the result of Dijkstra algorithm `void printDijkstra(Graph &g)` with the use of for- loop print the shortest distance of every vertex.

Results

```
The vertex: 0 has shortest distance: 0
The vertex: 1 has shortest distance: 4
The vertex: 2 has shortest distance: 12
The vertex: 3 has shortest distance: 19
The vertex: 4 has shortest distance: 21
The vertex: 5 has shortest distance: 11
The vertex: 6 has shortest distance: 9
The vertex: 7 has shortest distance: 8
The vertex: 8 has shortest distance: 14
Press any key to continue . . .
```