11/29/2020

Dev Tejwani

Barath Dandu

COP 4530 Final Project Report

Dijkstra's Algorithm

Aim/Purpose:

The purpose of the assignment is to implement an undirected weighted Graph ADT and performing Dijkstra's Algorithm to find the shortest path between two vertices. The graph will implement methods that add and remove vertices, add and remove edges, and calculate the shortest path.

Introduction:

Dijkstra's algorithm computes the length of the shortest path from the source to each of the remaining vertices in the graph. The algorithm is considered a Greedy algorithm as it uses problem solving methods based on actions to see if there is a better long-term strategy. Dijkstra's algorithm uses the greedy approach to solve the single source shortest problem. It repeatedly selects from the unselected vertices, vertex v nearest to source s and declares the distance to be the actual shortest distance from s to v. The edges of v are then checked to see if their destination can be reached by v followed by the relevant outgoing edges. Dijkstra's algorithm works by computing the shortest path from source to vertices among the k closes vertices to the source. It works when it is a directed-weighted graph, and the edges are non-negative. Modern day applications of the algorithm include traffic information systems, mapping such as map quest, google maps and routing systems, epidemiology.

Implementation:

For this project, the problem was broken down into different functions. As shown:

```
void addVertex(std::string label);
void removeVertex(std::string label);
void addEdge(std::string label1, std::string label2, unsigned long weight);
void removeEdge(std::string label1, std::string label2);
unsigned long shortestPath(std::string startLabel, std::string endLabel,
std::vector<std::string> &path);
```

```
void printShortestPath(std::map<std::string, std::string> startLabel, std::string
endLabel, std::vector<std::string> &path);
```

The addVertex function Creates and adds a vertex to the graph with label. No two vertices should have the same label.

The removeVertex function Removes the vertex with label from the graph. Also removes the edges between that vertex and the other vertices of the graph.

The addEdge function Adds an edge of value weight to the graph between the vertex with label1 and the vertex with label2. A vertex with label1 and a vertex with label2 must both exist, there must not already bean edge between those vertices, and a vertex cannot have an edge to itself.

The removeEdge function Removes the edge from the graph between the vertex with label1 and the vertex with label2. A vertex with label1 and a vertex with label2 must both exist and there must be an edge between those vertices.

The shortestPath function Calculates the shortest path between the vertex with startLabel and the vertex with endLabel using Dijkstra's Algorithm. A vector is passed into the method that stores the shortest path between the vertices. The return value is the sum of the edges between the start and end vertices on the shortest path.

We used an adjacency list to implement the algorithm in our assignment, and it can be described as collection of unordered list used to represent a finite graph. To implement this, another .hpp class was created named Edge.

It is better to use an adjacency list than an adjacency matrix when the graphs are not fully connected as we are only using memory or nodes that actually exist on the graph.

Efficiency:

The efficiency of the algorithm can be calculated by using the Big-O Notation. Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity. Big O is a member of a family of notations invented by Paul Bachmann, Edmund Landau, and others, collectively called Bachmann–Landau notation or asymptotic notation.

The efficiency can be calculated by Time = $|V|T_{EXTRACT-MIN} + |E|T_{DECREASE-KEY.}$ using this the time efficiency was calculated as, $O(|V|^2)$ for arrays, $O((|V|+|E|) \log |V|)$ for Binary heap.

Disadvantages:

Dijkstra's Algorithm is a great algorithm and has many uses however, it has some limitations too. The biggest limitation is that it cannot handle negative edges, and this makes it so that it always cannot find the right shortest path as it always sees negative edges as the shortest path. The other disadvantage is that it does a blind search to find the shortest path which leads to it wasting a lot of time.

Conclusion:

This project helped us learn one of the most important algorithms used in daily life and it showed us how to work in teams and how to coordinate in a professional environment. We have successfully implemented Dijkstra's Algorithm to implement an undirected weighted Graph ADT and to find the shortest path between two vertices. The project has passed all 18 assertions and test cases. This algorithm can truly be called the greedy algorithm as it chooses the nearest possible vertex known as local optimum.