Dijkstra's Shortest Paths From Source Vertex to all Vertices

Problem Statement

Given a directed graph and a source vertex, write a method to find shortest paths from a source vertex to all other vertices in the graph.

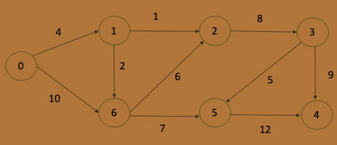
The function will have the following signature:

```
vector<int> dijkstra(const Graph& graph, int src)
```

We have defined the following Graph class and Edge struct for you:

```
struct Edge {
        int src, dest, weight;
        Edge(int _src, int _dest, int _weight) {src = _src; dest = _dest; weight = _weight;}
};

class Graph {
public:
        int numVertices;
        vector<vector<pair<int, int>>> adjList;
    // e.g. 0: {1, 12}, {2, 4} -> the weight from vertex 0 to 1 is 12
    //                            and the weight from vertex 0 to 2 is 4

        Graph(const vector<Edge>& edges, int vertices) {
                numVertices = vertices;

                adjList.resize(vertices);

                for (auto &edge : edges) {
                        adjList[edge.src].push_back(make_pair(edge.dest, edge.weight));
                }
        }
};
```

Examples



Input:

```
7 0
[[0,1,4],[1,2,1],[2,3,8],[3,4,9],[3,5,5],[5,4,12],[6,5,7],[0,6,10],[1,6,2],[6,2,6]]
```

Output:

```
Vertex Distance from Source:
0       0
1       4
2       5
3       13
4       22
5       13
6       6
```

Test Cases

- As for the input first line, the first item is the number of vertices in the graph. The second item is the source vertex.
- The second line of input is a list of edges in the graph.
- The output indicates the shortest path from the source vertex to all vertices in the graph.

Note

- You are only required to return a `vector<int>` that contains the shortest path from the source vertex to all vertices in the directed graph.
- You should not print anything out in the `dijkstra` method. We will print out the items you return in the main method.

Author: Lisha Zhou, **Date Created:** 17 Jun 2020, **Last Modified:** 17 Jun 2020

Sample Input:

```
7 0
[[0,1,4],[1,2,1],[2,3,8],[3,4,9],[3,5,5],[5,4,12],[6,5,7],[0,6,10],[1,6,2],[6,2,6]]
```

Sample Output:

```
Vertex Distance from Source:
0       0
1       4
2       5
3       13
4       22
5       13
6       6
```

Write a program, test using stdin → stdout

Correct answer from **93** learners
Total **61%** of tries are correct

Good job.

Now you have access to the Forum of Solutions where you can discuss your solution with others.

```
[+] Test #1. Wowza!
[+] Test #2. You are a genius!
[+] Test #3. Wowza!
[+] Test #4. Good work!

4 of 4 test(s) passed.
```

```
#include <unordered_set>
#include <limits>
#include <vector>
#include <queue>
#include <utility>
typedef pair<int,int> pairs;

vector<int> dijkstra(const Graph& graph, int src) {
    //Containers
    unordered_set<int> checkedSet;
    unordered_set<int> uncheckedSet;
    vector<int> distance;
    vector<int> predecessor;
    priority_queue<pairs, vector<pairs>, greater<pairs>> distanceQueue;

    //Add vertices to the unchecked list
    for (int i = 0; i < graph.adjList.size(); i++) {
        uncheckedSet.insert(i);
    }

    //Set all the distances to infinity, source distance to 0, and all predecessors to -1
    distance.resize(graph.adjList.size(), INT32_MAX);
    distance[src] = 0;
    predecessor.resize(graph.adjList.size(), -1);

    //Create the pairs (distance, vertex) and add to queue
    for (int i = 0; i < graph.adjList.size(); i++) {
        pairs dvPair = make_pair(distance[i], i);
        distanceQueue.push(dvPair);
    }

    //Dijkstra's Algorithm
    while (!uncheckedSet.empty()) {
        //Vertex with the shortest distance from the source
        pairs dvPair = distanceQueue.top();
        int vertex = dvPair.second;
        int vertexDistance = dvPair.first;
        //Go through the adjList for the vertex and do the relaxation
        for (pairs toVertexAndWeight : graph.adjList[vertex]) {
            int oldDistance = distance[toVertexAndWeight.first];
            int newDistance = vertexDistance + toVertexAndWeight.second;
            if (newDistance < oldDistance) {
                distance[toVertexAndWeight.first] = newDistance;
                predecessor[toVertexAndWeight.first] = vertex;
            }
        }
        //Remove vertex from unchecked set and insert it into the checked set
        uncheckedSet.erase(vertex);
        checkedSet.insert(vertex);
        //Recreate the distance queue??
        distanceQueue = priority_queue<pairs, vector<pairs>, greater<pairs>>();
        for (int i = 0; i < graph.adjList.size(); i++) {
            if (checkedSet.find(i) == checkedSet.end()) {
                pairs dvPair = make_pair(distance[i], i);
                distanceQueue.push(dvPair);
            }
        }
    }

    return distance;
}

/*
while (!distanceQueue.empty()) {
        pairs dv = distanceQueue.top();
        cout << "Vertex:  " << dv.second << "    Distance: " << dv.first << endl;
        distanceQueue.pop();
    }
*/


```

Next step | Solve again

Your submissions You got: **1 point** out of 1

1 1 Step 1 Next step

Comments 7 Solutions

GO Leave a comment

No discussions. Feel free to start one.