Graph Algorithms and Related Data Structures

REPORT

PROJECT 2

ITCS 6114 - Algorithms and Data Structures

DEPARTMENT OF COMPUTER SCIENCE

SUBMITTED TO
Dewan T. Ahmed, Ph.D.

SUBMITTED BY:

Mukesh Dasari	801208218
Yogesh Narigapalli	801193142

Table of Contents

INTRODUCTION:

This report demonstrates the finding Single source shortest path of any given weighted directed or undirected graph and finding Minimum spanning tree of any given weighted undirected graph. For finding the shortest path from single source node we have used "Dijkstra's Algorithm" which uses greedy method of optimization. In this algorithm, the graph G = (V, E) where V is number of vertices and E is number of edges in the graph must be connected and weight should not have a negative value. And for finding the cost of minimum spanning tree we have used "Prim's Algorithm" which is also uses greedy method for finding the optimal solution. In this algorithm, we start from one random vertex of graph G = (V, E) and then keep on adding the edge with least value from the current vertex to the spanning tree. This algorithm considers a graph must be undirected and connected.

Project Structure:

```
|_ input_sources/
   |_ Dijkstra_graph1.txt
   |_ Dijkstra_graph2.txt
   |_ Dijkstra_graph3.txt
   |_ Dijkstra_graph4.txt
   |_ Prim_graph1.txt
   |_ Prim_graph2.txt
   |_ Prim_graph3.txt
   |_ Prim_graph4.txt
|_ dijkstra.cpp
|_ prims.cpp
|_ Makefile
|_ Report.pdf
```

Execution Information:

1. We have run 4 graphs on both Dijkstra's algorithm and Prim's algorithm.
2. In each test program calculates shortest distance in case of Dijkstra's and MST in case of Prim's algorithm along with execution time in milliseconds.
3. If in given graph input, starting point is not given which is the last line of the input file then default A is considered.
4. Vertices of graph inputs should be a character only [A-Z].

Running code:

The project uses Makefile to compile, run and test the algorithms. The inputs are taken from input_sources directory in which there are 4 different graphs or each algorithm. Below are the instructions for running the project.

1. **`make`** – Compiles both `dijkstra.cpp` and `prims.cpp` and creates executable files `dijkstra` and `prims`.
5. **`make test`** – It runs both executables using all test files from input_sources/ and prints output on the console of each graph.
6. **`make clean`** – It removes object files and executables.

SINGLE SOURCE SHORTEST PATH:

Description:

In a given graph G = (V, E) where V is the total number of vertices and E is the total number of edges with having a weight, a single source shortest path P can be defined as the sum of the weights of the edges which consist of E_0, E_1, …, E_k is,

$w(P) = \sum_{i=0}^{k-1} w(E_i)$

A distance of vertex v from a staring vertex s if is a shortest path between them then it is denoted as d(s, v). And if there exist no edge between any two vertices then the d(s, v) = ∞.

Algorithm:

Here we have implemented **Dijkstra's Algorithm** to calculate the shortest path from a source in a given graph. It uses the greedy approach which means at end we get the optimal solution. For using this algorithm, the graph should a connected graph and its edges should not have a negative weight. Also, this algorithm works on both directed and undirected graph.

Staring from the source, it grows set of vertices until all the vertices are covered. At each vertex v inside the set, we find the vertex u of not visited set which has least distance d(u). And then we add u in visited set and process goes on for its adjacent vertices. It uses Edge Relaxation technique in which suppose we have added v vertex in visited set and u is in not visited set then if current d(u) is greater than d(v) + w(v, u) then we update it with d(v) + w(v, u) value.

Pseudo Code:

```
Algorithm Dijkstra(G, w, s):
      INITIALIZE-SINGLE-SOURCE (G, s)
      S = 0
      Q = G.V
      while Q != 0 do
            u = EXTRACT-MIN(Q)
            S = S ∪ {u}
      for each vertex v ∈ G.Adj[u] do
            RELAX (u, v, w)

Algorithm  INITIALIZE-SINGLE-SOURCE (G, s)
      for each vertex v ∈G.V do
            d[v] = ∞
            π[v] = NIL
            d[s] = 0

Algorithm RELAX(u, v, w):
      If(v.d > u.d + w(u, v))
```

```
v.d = u.d + w(u, v)
v.π = u
```

Data Structure & Complexity Analysis:

To implement Dijkstra's Algorithm, we have used 1D vector for storing vertices, distances, parents and 2D vectors for storing graph. We have used priority queue which takes O(m log(n)) where m is number of edges and n is number of vertices. The EXTRACT_MIN(Q) takes O(log(n)) time, and the relaxation of vertices takes O(log(m)) time. So, the overall runtime of algorithm is O((n log(n) + m log(n)). To conclude the time complexity of Dijkstra's Algorithm is (m log(n)) considering it is a connected graph.

Input/Output:

Input:

Graph 1

```
6 10 U
A B 1
A C 2
B C 1
B D 3
B E 2
C D 1
C E 2
D E 4
D F 3
E F 3
A
```

Graph 2

```
6 18 U
A B 10
A C 20
B A 10
B D 50
B E 33
C A 20
C D 20
C E 33
D B 50
D C 33
D E 20
D F 2
E B 10
E C 33
E D 20
E F 1
F D 2
F E 1
A
```

Graph 3

```
9 28 D
A B 4
A H 8
B A 4
B C 8
B H 11
C B 8
C D 7
C F 4
C I 2
D C 7
D E 9
D F 14
E D 9
E F 10
F C 4
F D 14
F E 10
F G 2
G F 2
G H 1
G I 6
H A 8
H B 11
H G 1
H I 7
I C 2
I G 6
I H 7
```

Graph 4

```
9 16 U
A B 4
A C 14
A H 8
B C 8
B H 11
C D 7
C F 13
C I 2
D E 9
D A 11
D F 14
E F 10
F G 2
G H 5
G I 6
H I 3
A
```

Output:

```
./dijkstra

***********************************************
               DIJKSTRA'S ALGORITHM
***********************************************
Graph 1
Vertex : 6      Edge : 10
Shortest Distance : 1           Path : A -> B
Shortest Distance : 2           Path : A -> C
Shortest Distance : 3           Path : A -> C -> D
Shortest Distance : 3           Path : A -> B -> E
Shortest Distance : 6           Path : A -> B -> E -> F
Time(ms) : 0.0739
***********************************************
Graph 2
Vertex : 6      Edge : 18
Shortest Distance : 10          Path : A -> B
Shortest Distance : 20          Path : A -> C
Shortest Distance : 23          Path : A -> B -> E -> F -> D
Shortest Distance : 20          Path : A -> B -> E
Shortest Distance : 21          Path : A -> B -> E -> F
Time(ms) : 0.0186
***********************************************
Graph 3
Vertex : 9      Edge : 28
Shortest Distance : 4           Path : A -> B
Shortest Distance : 12          Path : A -> B -> C
Shortest Distance : 19          Path : A -> B -> C -> D
Shortest Distance : 21          Path : A -> H -> G -> F -> E
Shortest Distance : 11          Path : A -> H -> G -> F
Shortest Distance : 9           Path : A -> H -> G
Shortest Distance : 8           Path : A -> H
Shortest Distance : 14          Path : A -> B -> C -> I
Time(ms) : 0.0833
***********************************************
Graph 4
Vertex : 9      Edge : 16
Shortest Distance : 4           Path : A -> B
Shortest Distance : 12          Path : A -> B -> C
Shortest Distance : 11          Path : A -> D
Shortest Distance : 20          Path : A -> D -> E
Shortest Distance : 15          Path : A -> H -> G -> F
Shortest Distance : 13          Path : A -> H -> G
Shortest Distance : 8           Path : A -> H
Shortest Distance : 11          Path : A -> H -> I
Time(ms) : 0.1112
***********************************************
***********************************************
```

MINIMUM SPANNING TREE:

Description:

In a given graph G = (V, E) where V is the total number of vertices and E is the total number of edges with having a weight, a minimum spanning tree is calculating the smallest weight among all spanning trees of given graph G. Mathematically it is represented as,

w(G) = $\sum_{(u,v)\in G} w(u, v)$

To calculate the minimum spanning tree there are two most popular algorithms, Prim's Algorithm and Kruskal's Algorithm.

Algorithm:

Here we have implemented **Prim's Algorithm** to calculate the minimum spanning tree (MST) starting from source s. To yield the optimal solution it uses greedy approach. It works same as that of Dijkstra's Algorithm to find shortest path in graph. Prim's algorithms considers that the graph G is connected and with undirected edges. It starts from one vertex v and then keep adding edges with minimum weights and eventually covers all the vertices. visited vertices always make a single tree and which is the property of Prim's Algorithm.

Pseudo Code:

```
Algorithm MST_PRIM(G, w, s):
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    s.key = 0
    Q = G.V
    While Q != 0
        u = EXTRACT_MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q AND w(u, v) < v.key
                v.key = w(u, v)
                v.π = u
```

Data Structure & Complexity Analysis:

To implement Prim's Algorithm, we have used 1D vector for storing vertices, distances, parents and 2D vectors for storing graph. We have used priority queue which takes O(m log(n)) where m is number of edges and n is number of vertices. The EXTRACT_MIN(Q) takes O(log(n)) time as used min priority queue. So, the overall runtime of algorithm is O((n log(n) + m log(n)). To conclude the time complexity of Prim's Algorithm is (m log(n)) considering it is a connected graph.

Input/Output:

Input:

Graph 1

```
1   5 7 U
2   A B 2
3   A D 6
4   B C 3
5   B D 8
6   B E 5
7   C E 7
8   D E 9
9   A
```

Graph 2

```
1   6 9 U
2   A B 2
3   A C 3
4   B C 5
5   B D 3
6   B E 4
7   C E 2
8   D E 4
9   D F 3
10  E F 2
11  A
```

Graph 3

```
1   9 16 U
2   A B 4
3   A H 8
4   A E 2
5   A C 9
6   B C 8
7   B H 11
8   C D 7
9   C F 4
10  C I 2
11  D E 9
12  D F 14
13  E F 10
14  F G 2
15  G H 1
16  G I 6
17  H I 7
18  A
```

Graph 4

```
1   7 12 U
2   A B 1
3   A D 5
4   A E 8
5   A G 1
6   B C 2
7   B G 2
8   C A 3
9   C D 9
10  D E 7
11  E F 2
12  F G 1
13  F A 2
14  A
```

Output

```
./prims

************************************************
                PRIM'S ALGORITHM
************************************************
Graph 1
Vertex : 5      Edge : 7
Edge : A -> B   Weight : 2
Edge : B -> C   Weight : 3
Edge : A -> D   Weight : 6
Edge : B -> E   Weight : 5
MST Cost : 16
Time(ms) : 0.0653
************************************************
Graph 2
Vertex : 6      Edge : 9
Edge : A -> B   Weight : 2
Edge : A -> C   Weight : 3
Edge : B -> D   Weight : 3
Edge : C -> E   Weight : 2
Edge : E -> F   Weight : 2
MST Cost : 12
Time(ms) : 0.0428
************************************************
Graph 3
Vertex : 9      Edge : 16
Edge : A -> B   Weight : 4
Edge : A -> C   Weight : 9
Edge : E -> D   Weight : 9
Edge : A -> E   Weight : 2
Edge : E -> F   Weight : 10
Edge : H -> G   Weight : 1
Edge : G -> H   Weight : 1
Edge : G -> I   Weight : 6
MST Cost : 42
Time(ms) : 0.0987
************************************************
Graph 4
Vertex : 7      Edge : 12
Edge : A -> B   Weight : 1
Edge : B -> C   Weight : 2
Edge : A -> D   Weight : 5
Edge : F -> E   Weight : 2
Edge : G -> F   Weight : 1
Edge : A -> G   Weight : 1
MST Cost : 12
Time(ms) : 0.0639
************************************************
```