JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & I.T

2020-21

Non-Linear Data Structures Lab
Project Report

IMPLEMENTATION OF DIJKSTRA'S ALGORITHM USING FENWICK TREE & MIN HEAP

Made By :

Sachin Kumar	17104008
Ayush Nagar	17104012
Akshara Nigam	17104018
Siddharth Aggarwal	17104031

Acknowledgement

It gives us a great sense of pleasure to present the report of the Non Linear Data Structures Lab Project undertaken during B.Tech Pre-Final Year. We owe special gratitude to Mr. Mohit Kumar, Department of Computer Science and IT, for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project.

Aim : Using Min Heap & Fenwick tree, to implement the shortest path algorithm (Dijkstra's Algorithm)

Working : The nodes are inserted using **add_edge()** function to the graph object. The graph shortest path from **0^{th}** node is calculated using the **dijkstra()** function. Each unvisited node is traversed and its next node is found using **getNextSelectedNode()** which first finds the unvisited neighbour of the last selected node. The Fenwick tree of each unselected node is made using the Fenwick tree class' object and the sum of the next probable node is found and stored in Min Heap.

When all the unvisited path nodes are added to the list of objects of Fenwick tree which is used to create min-heap using **build_heap_optimised()**. The minimum cost node is popped using min-heap and checked if the destination node is already visited or not. If the destination node is found already visited, it means the cycle is going to be formed and this edge needs to be discarded. The edge is popped from the min-heap, then either the heap gets empty or the popped edge destination is unvisited or all the nodes are found to be visited.
As soon as all the nodes get visited, the program terminates by printing the cost of each node traversal from 0^{th} node.

Setup/Execution: To run the code in you system type the command in your terminal or cmd step by step :

Step 1 : ***g++ final_code.cpp***

Step 2 : ***./a.out < test1-input.txt***

This will run the code for the test case 1, you could also provide your own test cases and save it as a *.txt* file.

Test file format

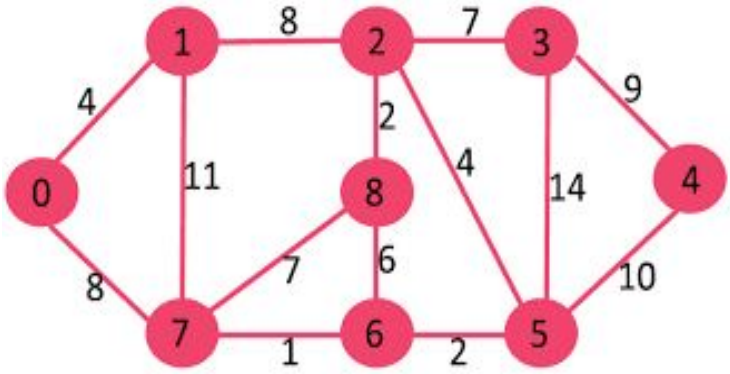
First-line must tell the number of nodes=>**n**
Next line must indicate the number of edges in bidirected graph=>**e**
Next e lines must be of format=> **source destination cost**

Complexity Analysis :

Fenwick Tree ⇒	(a) sum()	:	O(N)
	(b) update()	:	O(N)
	(c) copyNupdate()	:	O(N)
	(d) getParent()	:	O(1)
	(e) getNext()	:	O(1)
Min Heap ⇒	(a) build_heap_optimized()	:	O(N)
	(b) delete_peak()	:	O(logN)
	(c) down_heapify()	:	O(logN)
Main Code ⇒	(a) addEdge()	:	O(1)
(including Graph	(b) findUnvisitedNeighbour	:	O(N)
Min Heap &	(c) getPathCost()	:	O(N)
Fenwick Tree)	(d) pathByVisited()	:	O(N)
	(e) findNextSelectedNode()	:	O(N^2) (Amortized)
	(f) allVisited()	:	O(N)
	(g) dijkstra()	:	O(N)

Testing :

Test Case 1 :-



(Input)

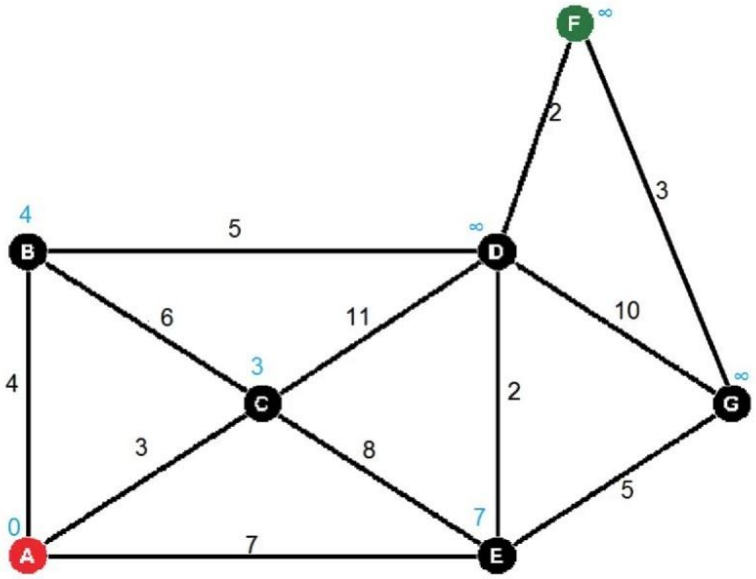
Expected Outcome by actual Dijkstra's algorithm:- 0 4 12 19 21 11 9 8 14

```
ayushnagar123@ayushnagar123-HP-Laptop-15-da0xxx:~/Desktop/Non_linear_DS$ ./a.out <tests/test1-input.txt
path for 7:- 0 8
path for 1:- 0 4
value= 4 source= 0 dest=1
path for 2:- 4 8
path for 7:- 4 11
value= 8 source= 0 dest=7
path for 8:- 8 7
path for 6:- 8 1
value= 9 source= 7 dest=6
path for 8:- 8 1 6
path for 5:- 8 1 2
value= 11 source= 6 dest=5
path for 4:- 8 1 2 10
path for 3:- 8 1 2 14
path for 2:- 8 1 2 4
value= 12 source= 1 dest=2
path for 3:- 4 8 7
path for 8:- 4 8 2
value= 14 source= 2 dest=8
value= 19 source= 2 dest=3
path for 4:- 4 8 7 9
value= 21 source= 5 dest=4

0 4 12 19 21 11 9 8 14
```

(Output)

Test Case 2 :-



(Input)

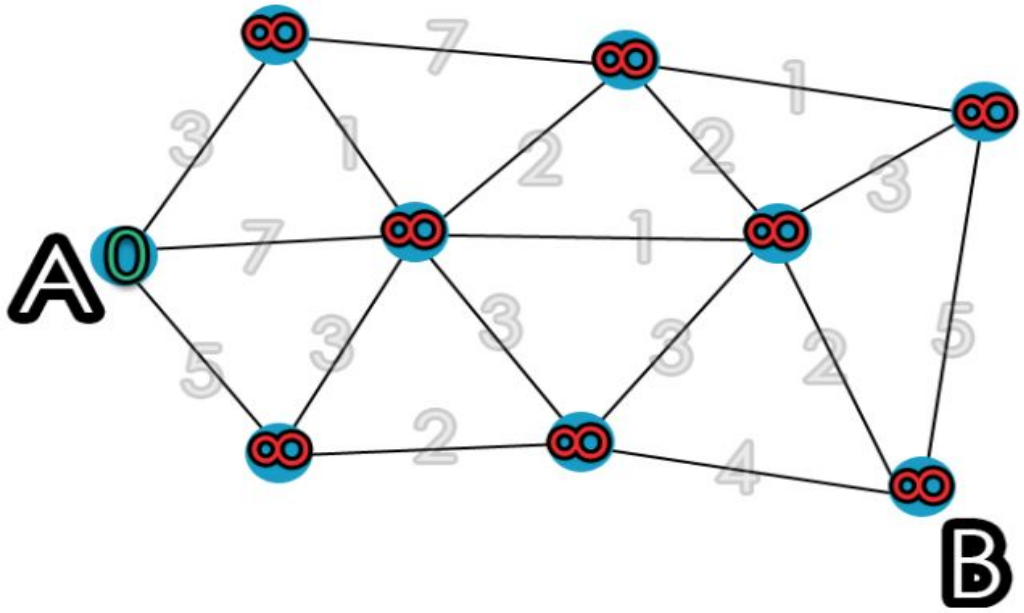
Expected Outcome by actual Dijkstra's algorithm:- 0 4 3 9 7 11 12

```
ayushnagar123@ayushnagar123-HP-Laptop-15-da0xxx:~/Desktop/Non_linear_DS$ ./a.out <tests/test2-input.txt
path for 4:- 0 7 x:~/Desktop/Non_li
path for 2:- 0 3
path for 1:- 0 4
value= 3 source= 0 dest=2
path for 4:- 3 8
path for 3:- 3 11
path for 1:- 3 6
value= 4 source= 0 dest=1
path for 3:- 4 5
value= 7 source= 0 dest=4
path for 6:- 7 5
path for 3:- 7 2
value= 9 source= 1 dest=3
path for 5:- 4 5 2
path for 6:- 4 5 10
value= 11 source= 3 dest=5
path for 6:- 4 5 2 3
value= 12 source= 4 dest=6

0 4 3 9 7 11 12
```

(Output)

Test Case 3 :-



(Input)

Expected Outcome by actual Dijkstra's algorithm:- 0 3 4 5 7 5 7 7

```
ayushnagar123@ayushnagar123-HP-Laptop-15-da0xxx:~/Desktop/Non_linear_DS$ ./a.out <tests/test3-input.txt
path for 3:- 0 5
path for 2:- 0 7
path for 1:- 0 3
value= 3 source= 0 dest=1
path for 2:- 3 1
path for 4:- 3 7
value= 4 source= 1 dest=2
path for 3:- 3 1 3
path for 5:- 3 1 3
path for 6:- 3 1 1
path for 4:- 3 1 2
value= 5 source= 0 dest=3
path for 5:- 5 2
value= 5 source= 2 dest=6
path for 8:- 3 1 1 2
path for 7:- 3 1 1 3
path for 5:- 3 1 1 3
path for 4:- 3 1 1 2
value= 6 source= 2 dest=4
path for 7:- 3 1 2 1
value= 7 source= 4 dest=7
path for 8:- 3 1 2 1 5
value= 7 source= 2 dest=5
path for 8:- 3 1 3 4
value= 7 source= 6 dest=8

0 3 4 5 6 7 5 7 7
```

(Output)

Limitations: Doesn't work well with graphs having isolated nodes. But some optimization may help the process to work with it.

Code Repository Github Link:

https://github.com/siddhu15798/Non_linear_DS