

## Project II: Graph Algorithms

Implement by: Sumedh Joglekar

Language Used: Java

### 1) Dijkstra's Algorithm :-

1. In order to find the shortest path in directed and undirected graphs for given source and with no negative edge involved, Dijkstra's algorithm can be used. I have implemented the same in java to find shortest path from given source.
2. Working of Dijkstra's :
  - a. Dijkstra's Algorithm can be implemented on both directed as well as undirected graphs.
  - b. First initialize all the nodes with MAX\_VALUE i.e. infinity except for source node.
  - c. Now add, source node to cloud and relax all the adjacent nodes of source also update the parent of all nodes explored to source node.
  - d. Store all the explored nodes in a heap using priority queue.
  - e. In the next iteration choose the closest node (node which has lesser edge value) and relax all the adjacent nodes of newly chosen node.
  - f. Every time after relaxation, update the parents of relaxed node.
  - g. Always select the node which has lesser edge value from priority queue and after each iteration heapify the queue of an edge.
  - h. Continue to do relaxing till the all the nodes are visited and added to the cloud.
3. Data Structures used :  
Priority Queue, Array List, List, Array
4. Run time :  
As the priority queue is used to store the edge values of all nodes and heapifying it after every iteration, run time of the algorithm can be given as –  
 $O(E \log V)$  {where E : Edges, V:Vertices}
5. Instructions to implement the code:  
Method A:
  - i) Extract the given file
  - ii) Open Command Prompt
  - iii) Goto the Path (using 'cd {Location of the file}\src\Dijkstras')
  - iv) While running from cmd, make sure path of txt file mentioned in the code is an absolute path.
  - v) Compile the java code by running (javac Dijkstra.java){Make sure JDK path variable added to environment paths}
  - vi) After Successful compilation, run the actual code : 'java Dijkstra'



```
Ca. Command Prompt
C:\Users\sunedh_4594>cd C:\Users\sunedh_4594\Documents\NetBeansProjects\AlgorithmProject\src\Dijkstras
```

```
C:\Users\sumedh_4594\Documents\NetBeansProjects\AlgorithmProject\src\Dijkstras>javac Dijkstra.java
C:\Users\sumedh_4594\Documents\NetBeansProjects\AlgorithmProject\src\Dijkstras>
```

```
C:\Users\sumedh_4594\Documents\NetBeansProjects\AlgorithmProject\src\Dijkstras>java Dijkstra
Taking Source as :- A
Given Sequence is :
A B 4
A H 8
B C 8
B H 11
C D 7
C F 4
C I 2
D E 9
D F 14
E F 10
F G 2
G H 1
G I 6
H I 7
Path and Distances of all Vertices from given source will be :-
A
A ----> B<4>
A ----> B<4> ----> C<12>
A ----> B<4> ----> C<12> ----> D<19>
A ----> H<8> ----> G<9> ----> F<11> ----> E<21>
A ----> H<8> ----> G<9> ----> F<11>
A ----> H<8> ----> G<9>
A ----> H<8>
A ----> B<4> ----> C<12> ----> I<14>
C:\Users\sumedh_4594\Documents\NetBeansProjects\AlgorithmProject\src\Dijkstras>
```

#### Method B:

- i) Extract the given project.
- ii) Import the extracted project in any IDE.
- iii) Select Package = Dijkstras and open Dijkstras.java file.
- iv) Update the path of the file at line number 82 and 128. PFB the Screenshot of the same.

```
81 // Taking absolute path of the file.
82 //public static final String x = "src\\Dijkstras\\Undirected2.txt";
83 //public static final String x = "src\\Dijkstras\\Undirected1.txt";
84 //public static final String x = "src\\Dijkstras\\Undirected.txt";
85 //public static final String x = "src\\Dijkstras\\Directed.txt";
86 //public static final String x = "src\\Dijkstras\\Directed1.txt";
87 public static final String x = "src\\Dijkstras\\Directed2.txt";
88 public static void main(String[] args) throws IOException { ...35 lines }
123 }
124 class Graph {
125     public static int flag ;
126     String str[];
127     String type = null;
128     int vertex_count = 0;
129     int edges_count = 0;
130     private final Map<String, Vertex> graph; // mapping of vertex names to Vertex ob;
131     public void readFile() // Method To read the file and check if Graph as Directed
132     {
133         //File file = new File("src\\Dijkstras\\Undirected2.txt");
134         //File file = new File("src\\Dijkstras\\Undirected1.txt");
135         //File file = new File("src\\Dijkstras\\Undirected.txt");
136         //File file = new File("src\\Dijkstras\\Directed1.txt");
137         //File file = new File("src\\Dijkstras\\Directed.txt");
138         File file = new File("src\\Dijkstras\\Directed2.txt");
139         Scanner sc;
```

- v) Once the path is updated, run the code.

## 6. Sample Input with Output:

Input 1(Directed):

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

Output :

```
run:
Taking Source as :- A
Given Sequence is :
A B 4
A H 8
B C 8
B H 11
C D 7
C F 4
C I 2
D E 9
D F 14
E F 10
F G 2
G H 1
G I 6
H I 7
Path and Distances of all Vertices from given source will be :-
A
A ---> B(4)
A ---> B(4) ---> C(12)
A ---> B(4) ---> C(12) ---> D(19)
A ---> B(4) ---> C(12) ---> D(19) ---> E(28)
A ---> B(4) ---> C(12) ---> F(16)
A ---> B(4) ---> C(12) ---> F(16) ---> G(18)
A ---> H(8)
A ---> B(4) ---> C(12) ---> I(14)
BUILD SUCCESSFUL (total time: 1 second)
```

Input2(Directed):

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

Output:

```
run:
Taking Source as :- B
Given Sequence is :
A B 2
A C 1
A D 4
B C 1
B E 10
B F 2
C A 9
C E 8
D C 2
E D 7
E G 1
F H 3
G E 4
G F 2
H G 1

Path and Distances of all Vertices from given source will be :-
B ---> C(1) ---> A(10)
B
B ---> C(1)
B ---> C(1) ---> A(10) ---> D(14)
B ---> C(1) ---> E(9)
B ---> F(2)
B ---> F(2) ---> H(5) ---> G(6)
B ---> F(2) ---> H(5)
BUILD SUCCESSFUL (total time: 0 seconds)
```

### Input3(Directed):

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

### Output3:

```
Output - Algorithm Project (run)  Dijkstra.java  Nuskais.java  Directed...
run:
Taking Source as :- A
Given Sequence is :
B C 11
A B 2
A C 9
B E 10
C A 2
D A 4
D C 3
E C 8
E D 4
E G 1
F B 2
G E 4
G F 2
G I 5
H F 3
H G 1
H I 10
Path and Distances of all Vertices from given source will be :-
A
A ---> B(2)
A ---> C(9)
A ---> B(2) ---> E(12) ---> D(16)
A ---> B(2) ---> E(12)
A ---> B(2) ---> E(12) ---> G(13) ---> F(15)
A ---> B(2) ---> E(12) ---> G(13)
H(node is not reachable from source)
A ---> B(2) ---> E(12) ---> G(13) ---> I(18)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Input 1:(Undirected)

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

Output:





```
run:
Taking Source as :- A
Given Sequence is :
A B 4
A C 2
B C 1
B D 9
B E 2
C D 1
C E 2
D E 8
D F 3
E F 3

Path and Distances of all Vertices from given source will be :-
A
A ----> C(2) ----> B(3)
A ----> C(2)
A ----> C(2) ----> D(3)
A ----> C(2) ----> E(4)
A ----> C(2) ----> D(3) ----> F(6)
BUILD SUCCESSFUL (total time: 4 seconds)
```

Input2 (Undirected):

1	6 8 U
2	A B 7
3	A C 9
4	A F 14
5	B D 15
6	C D 11
7	C F 2
8	D E 6
9	E F 9
0	A

Output:



```
run:
Taking Source as :- A
Given Sequence is :
A B 7
A C 9
A F 14
B D 15
C D 11
C F 2
D E 6
E F 9
Path and Distances of all Vertices from given source will be :-
A
A ----> B(7)
A ----> C(9)
A ----> C(9) ----> D(20)
A ----> C(9) ----> F(11) ----> E(20)
A ----> C(9) ----> F(11)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Input3:(Undirected)

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

Output:

```
run:
Taking Source as :- A
Given Sequence is :
A B 4
A H 8
B C 8
B H 11
C D 7
C F 4
C I 2
D E 9
D F 14
E F 10
F G 2
G H 1
G I 6
H I 7

Path and Distances of all Vertices from given source will be :-
A
A ----> B(4)
A ----> B(4) ----> C(12)
A ----> B(4) ----> C(12) ----> D(19)
A ----> H(8) ----> G(9) ----> F(11) ----> E(21)
A ----> H(8) ----> G(9) ----> F(11)
A ----> H(8) ----> G(9)
A ----> H(8)
A ----> B(4) ----> C(12) ----> I(14)
BUILD SUCCESSFUL (total time: 0 seconds)
```



## 2) Kruskal's Algorithm :

1. Kruskal algorithm is used to find the minimum possible weight between any two vertices in given undirected graph.
2. Working of Kruskal's Algorithm:
  - i) In order to find the minimum distance first we need to create a graph using vertices and weight of edges given in text file.
  - ii) Then we need to store the values of edges in increasing order of their weight.
  - iii) Check if any cycle is present or not.
  - iv) After that take out the edge with lowest weight and add it to the graph. Here we need to ignore the edges which are creating the cycles.
  - v) This needs to be done till all the vertices of graph are visited as the spanning tree is the one which has all the vertices similar to the original graph.
  - vi) At the termination of the algorithm, the tree forms a minimum spanning tree of the given graph.
3. Data Structures used :  
List, Array of list
4. Running time of an algorithm :  
As the sorting of edges are being done in linear time. The time complexity of an algorithm can be given as :-  $(E \log(V))$
5. Instructions to run the code:
  - i) Import the package Graph included in the zip file.
  - ii) Open Kruskal.java file as it contains the main function and run this particular java file.
  - iii) Input file path is present on the line numbers 19 - 24. Make sure input file is present in the same package else need to provide absolute path of the file.
  - iv) I have run the code for 5 different inputs and those files are already present in the src/graph/ folder of same package.

```
18 public void rf() {  
19     //File file = new File("src/graph/sample3.txt");  
20     File file = new File("src/graph/sample2.txt");  
21     //File file = new File("src/Undirected.txt");  
22     //File file = new File("src/Undirected1.txt");  
23     //File file = new File("src/Undirected2.txt");  
24     Scanner sc;
```

6. Sample output:

Input1:

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

Output:

```
run:
D ----> E ----> 2
D ----> F ----> 2
A ----> C ----> 3
F ----> G ----> 3
A ----> B ----> 4
B ----> D ----> 5
Total Weight 19
BUILD SUCCESSFUL (total time: 3 seconds)
```

Input2:

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

Output :

```
run:
A ----> B ----> 1
B ----> C ----> 1
C ----> D ----> 1
B ----> E ----> 2
D ----> F ----> 3
Total Weight 8
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

Input 3:

1	6 8 U
2	A B 7
3	A C 9
4	A F 14
5	B D 15
6	C D 11
7	C F 2
8	D E 6
9	E F 9





Output :

```
run:
C ----> F ----> 2
D ----> E ----> 6
A ----> B ----> 7
A ----> C ----> 9
E ----> F ----> 9
Total Weight 33
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

Input 4:

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

Output:







```
run:
G ----> H ----> 1
C ----> I ----> 2
F ----> G ----> 2
A ----> B ----> 4
C ----> F ----> 4
C ----> D ----> 7
A ----> H ----> 8
D ----> E ----> 9
Total Weight 37
BUILD SUCCESSFUL (total time: 0 seconds)
```

Input5:

Source	History
1	4 5 U
2	A B 1
3	A C 4
4	A D 3
5	B D 2
6	C D 5

Output:



```
run:
A ----> B ----> 1
B ----> D ----> 2
A ----> C ----> 4
Total Weight 7
BUILD SUCCESSFUL (total time: 0 seconds)
```