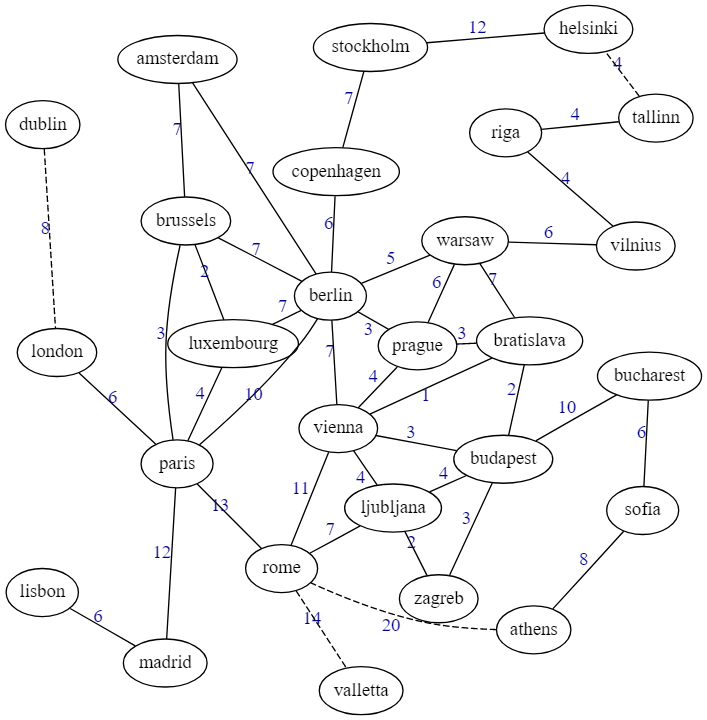
|  |  |
| --- | --- |
| **Algorithms & Data Structures** | |
| **Project 2** | |
| **Name** | Nilanjan Mhatre |
| **Student Id** | 801045013 |

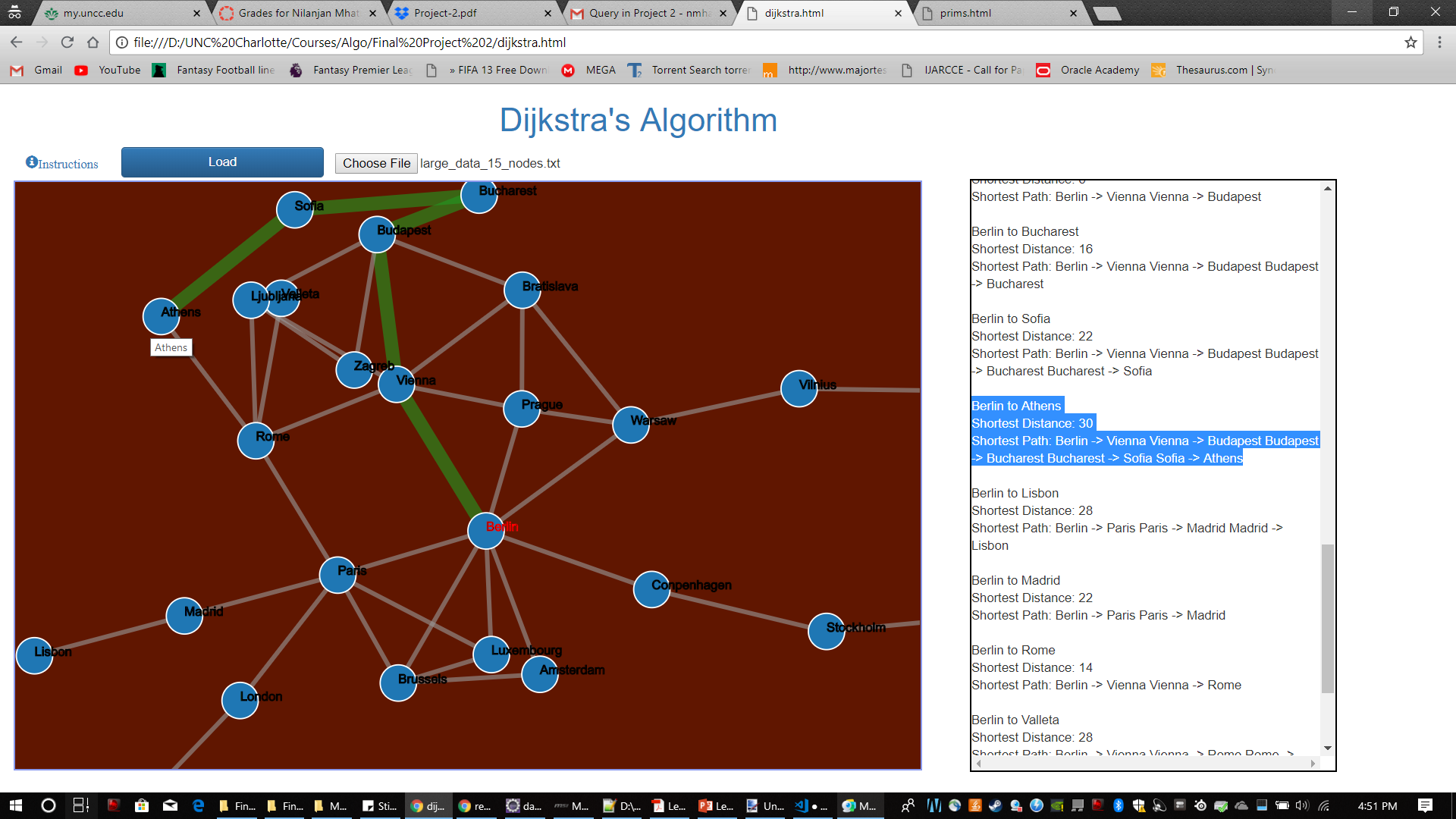
**Problem 1: Find Shortest path in undirected graphs.**

**Dijkstra’s Algorithm**

Input Sample 1: large\_data\_27\_nodes.txt



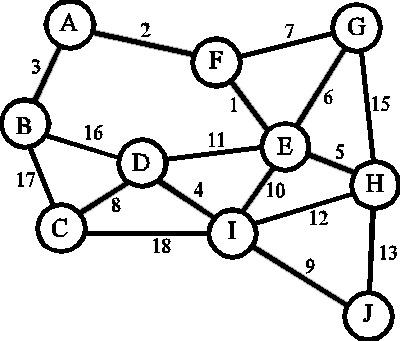
Output:



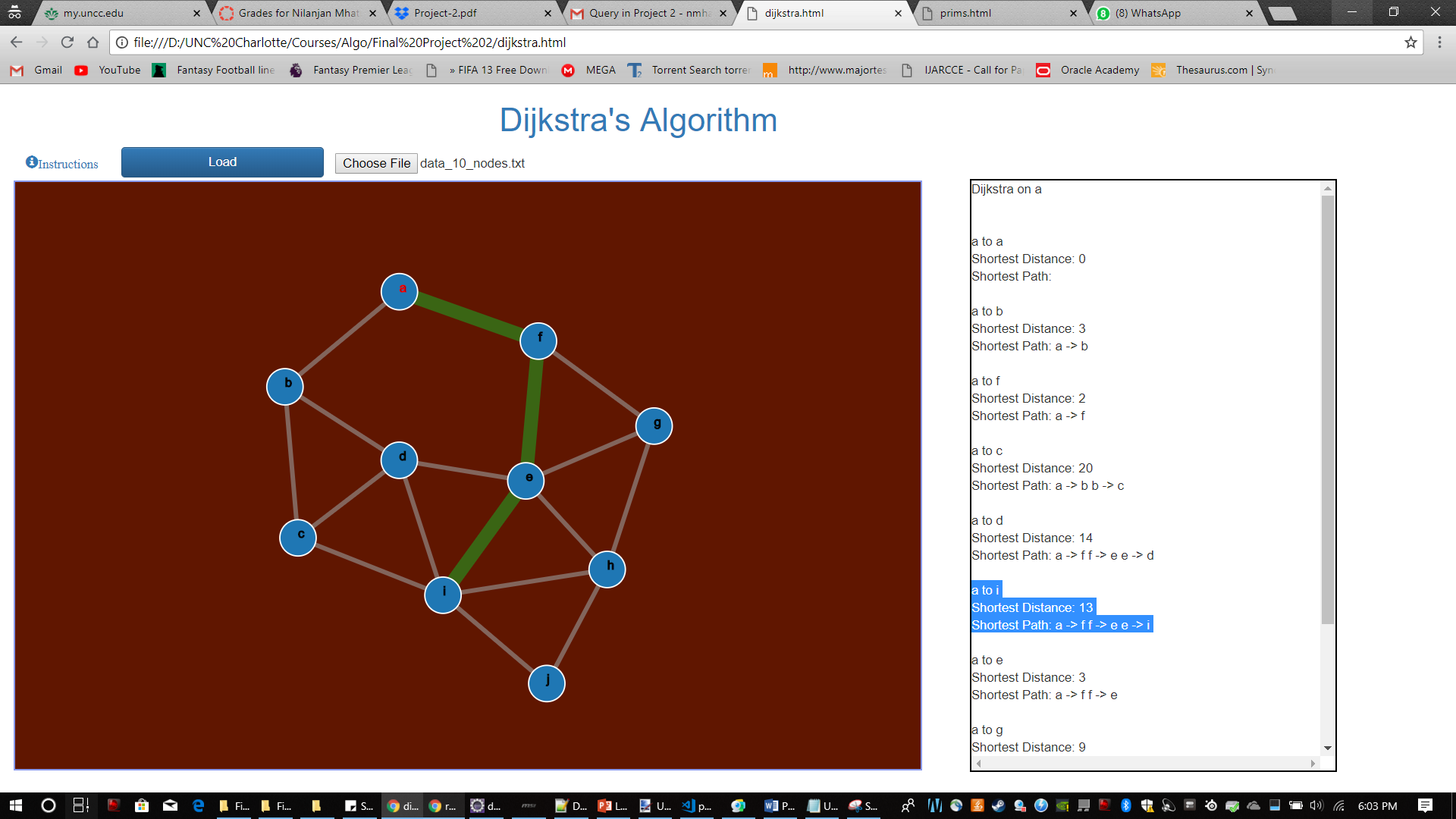
Dijkstra on Berlin

Berlin to Amsterdam  
Shortest Distance: 7  
Shortest Path: Berlin -> Amsterdam   
  
Berlin to Brussels  
Shortest Distance: 7  
Shortest Path: Berlin -> Brussels   
  
Berlin to Berlin  
Shortest Distance: 0  
Shortest Path:   
  
Berlin to Dublin  
Shortest Distance: 24  
Shortest Path: Berlin -> Paris Paris -> London London -> Dublin   
  
Berlin to London  
Shortest Distance: 16  
Shortest Path: Berlin -> Paris Paris -> London   
  
Berlin to Stockholm  
Shortest Distance: 13  
Shortest Path: Berlin -> Conpenhagen Conpenhagen -> Stockholm   
  
Berlin to Helsinki  
Shortest Distance: 23  
Shortest Path: Berlin -> Warsaw Warsaw -> Vilnius Vilnius -> Riga Riga -> Tallin Tallin -> Helsinki   
  
Berlin to Conpenhagen  
Shortest Distance: 6  
Shortest Path: Berlin -> Conpenhagen   
  
Berlin to Tallin  
Shortest Distance: 19  
Shortest Path: Berlin -> Warsaw Warsaw -> Vilnius Vilnius -> Riga Riga -> Tallin   
  
Berlin to Riga  
Shortest Distance: 15  
Shortest Path: Berlin -> Warsaw Warsaw -> Vilnius Vilnius -> Riga   
  
Berlin to Vilnius  
Shortest Distance: 11  
Shortest Path: Berlin -> Warsaw Warsaw -> Vilnius   
  
Berlin to Luxembourg  
Shortest Distance: 7  
Shortest Path: Berlin -> Luxembourg   
  
Berlin to Paris  
Shortest Distance: 10  
Shortest Path: Berlin -> Paris   
  
Berlin to Warsaw  
Shortest Distance: 5  
Shortest Path: Berlin -> Warsaw   
  
Berlin to Prague  
Shortest Distance: 3  
Shortest Path: Berlin -> Prague   
  
Berlin to Vienna  
Shortest Distance: 3  
Shortest Path: Berlin -> Vienna   
  
Berlin to Bratislava  
Shortest Distance: 4  
Shortest Path: Berlin -> Vienna Vienna -> Bratislava   
  
Berlin to Budapest  
Shortest Distance: 6  
Shortest Path: Berlin -> Vienna Vienna -> Budapest   
  
Berlin to Bucharest  
Shortest Distance: 16  
Shortest Path: Berlin -> Vienna Vienna -> Budapest Budapest -> Bucharest   
  
Berlin to Sofia  
Shortest Distance: 22  
Shortest Path: Berlin -> Vienna Vienna -> Budapest Budapest -> Bucharest Bucharest -> Sofia   
  
Berlin to Athens  
Shortest Distance: 30  
Shortest Path: Berlin -> Vienna Vienna -> Budapest Budapest -> Bucharest Bucharest -> Sofia Sofia -> Athens   
  
Berlin to Lisbon  
Shortest Distance: 28  
Shortest Path: Berlin -> Paris Paris -> Madrid Madrid -> Lisbon   
  
Berlin to Madrid  
Shortest Distance: 22  
Shortest Path: Berlin -> Paris Paris -> Madrid   
  
Berlin to Rome  
Shortest Distance: 14  
Shortest Path: Berlin -> Vienna Vienna -> Rome   
  
Berlin to Valleta  
Shortest Distance: 28  
Shortest Path: Berlin -> Vienna Vienna -> Rome Rome -> Valleta   
  
Berlin to Ljubljana  
Shortest Distance: 7  
Shortest Path: Berlin -> Vienna Vienna -> Ljubljana   
  
Berlin to Zagreb  
Shortest Distance: 9  
Shortest Path: Berlin -> Vienna Vienna -> Budapest Budapest -> Zagreb

**Input Sample 2: - data\_10\_nodes.txt**



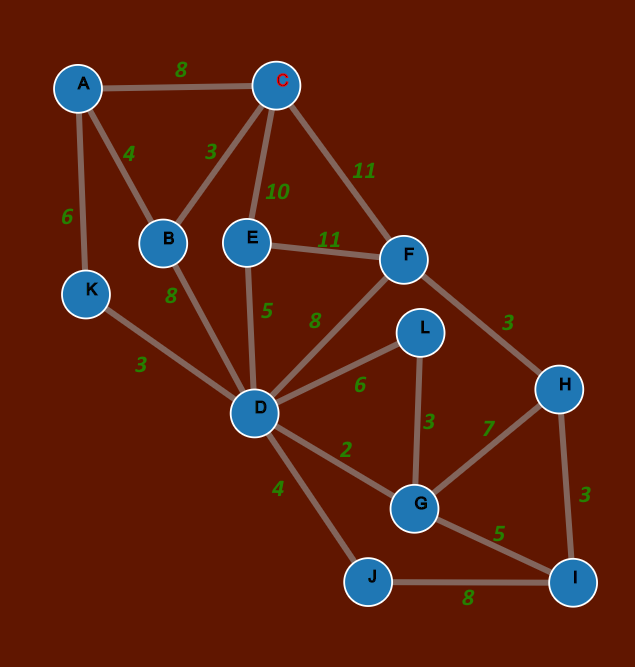
Output:



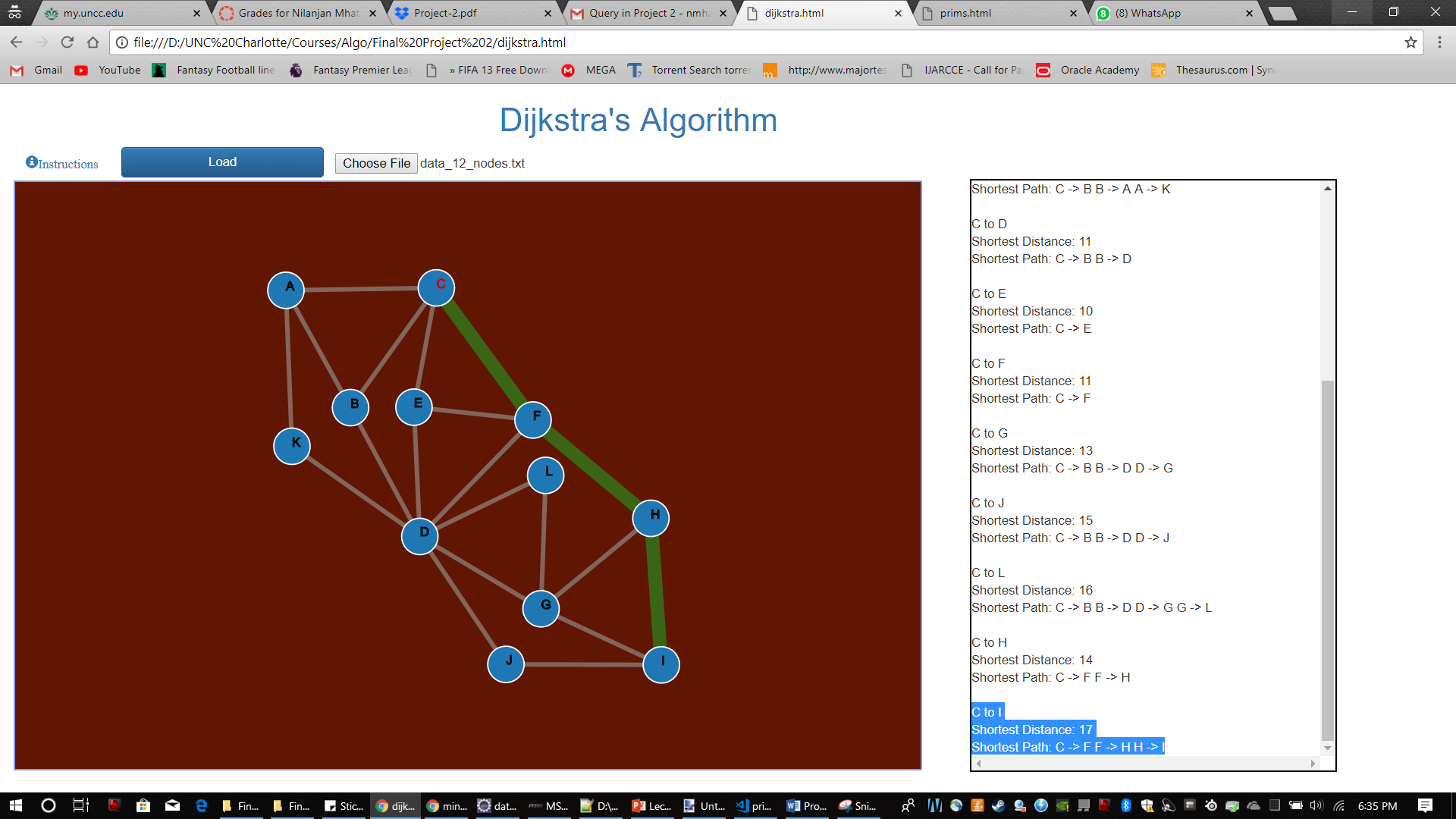
Dijkstra on a

a to a  
Shortest Distance: 0  
Shortest Path:   
  
a to b  
Shortest Distance: 3  
Shortest Path: a -> b   
  
a to f  
Shortest Distance: 2  
Shortest Path: a -> f   
  
a to c  
Shortest Distance: 20  
Shortest Path: a -> b b -> c   
  
a to d  
Shortest Distance: 14  
Shortest Path: a -> f f -> e e -> d   
  
a to i  
Shortest Distance: 13  
Shortest Path: a -> f f -> e e -> i   
  
a to e  
Shortest Distance: 3  
Shortest Path: a -> f f -> e   
  
a to g  
Shortest Distance: 9  
Shortest Path: a -> f f -> g   
  
a to h  
Shortest Distance: 8  
Shortest Path: a -> f f -> e e -> h   
  
a to j  
Shortest Distance: 21  
Shortest Path: a -> f f -> e e -> h h -> j

**Input Sample 3**: - **data\_12\_nodes.txt**



Output:-



Dijkstra on C

C to A  
Shortest Distance: 7  
Shortest Path: C -> B B -> A   
  
C to B  
Shortest Distance: 3  
Shortest Path: C -> B   
  
C to C  
Shortest Distance: 0  
Shortest Path:   
  
C to K  
Shortest Distance: 13  
Shortest Path: C -> B B -> A A -> K   
  
C to D  
Shortest Distance: 11  
Shortest Path: C -> B B -> D   
  
C to E  
Shortest Distance: 10  
Shortest Path: C -> E   
  
C to F  
Shortest Distance: 11  
Shortest Path: C -> F   
  
C to G  
Shortest Distance: 13  
Shortest Path: C -> B B -> D D -> G   
  
C to J  
Shortest Distance: 15  
Shortest Path: C -> B B -> D D -> J   
  
C to L  
Shortest Distance: 16  
Shortest Path: C -> B B -> D D -> G G -> L   
  
C to H  
Shortest Distance: 14  
Shortest Path: C -> F F -> H   
  
C to I  
Shortest Distance: 17  
Shortest Path: C -> F F -> H H -> I

Runtime Analysis (Dijkstra Algorithm): -

Priority Queue is used that uses heap structure.

Each vertex insertion and the corresponding heapify operation

takes ‘O(log n)’ time. ‘Total = n log n’

Each vertex removal and the corresponding heapify operation

takes ‘O(log n)’ time. ‘Total = n log n’

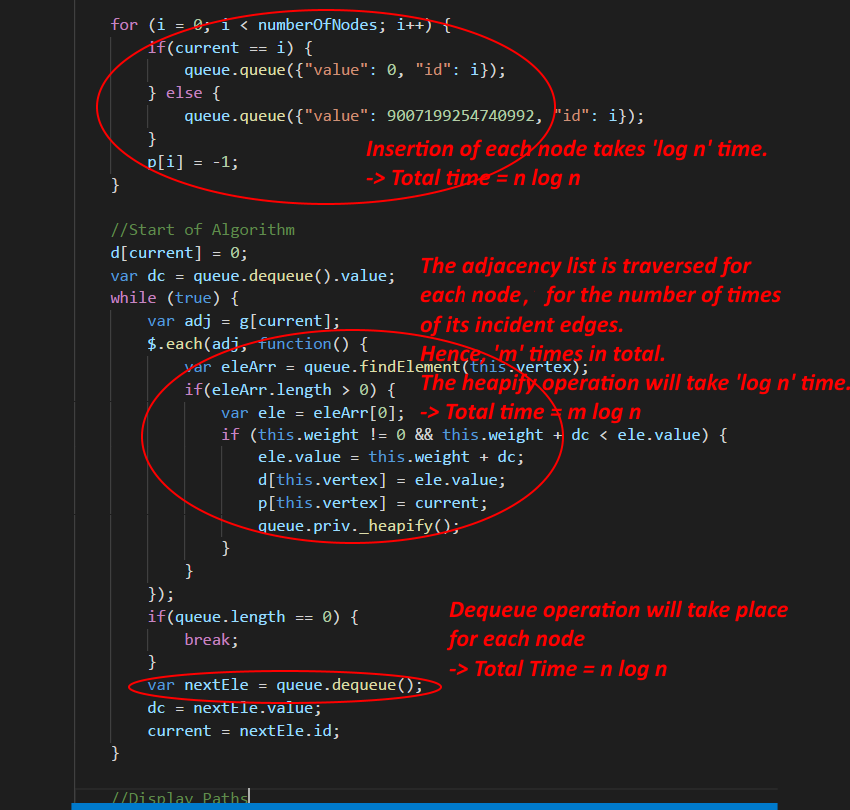
Each vertex modified along with heapify operation takes ‘O(log n)’ time

The modification is done for each edge, for total ‘m’ edges.

Total = m log n

The total time = n log n + m log n + n log n

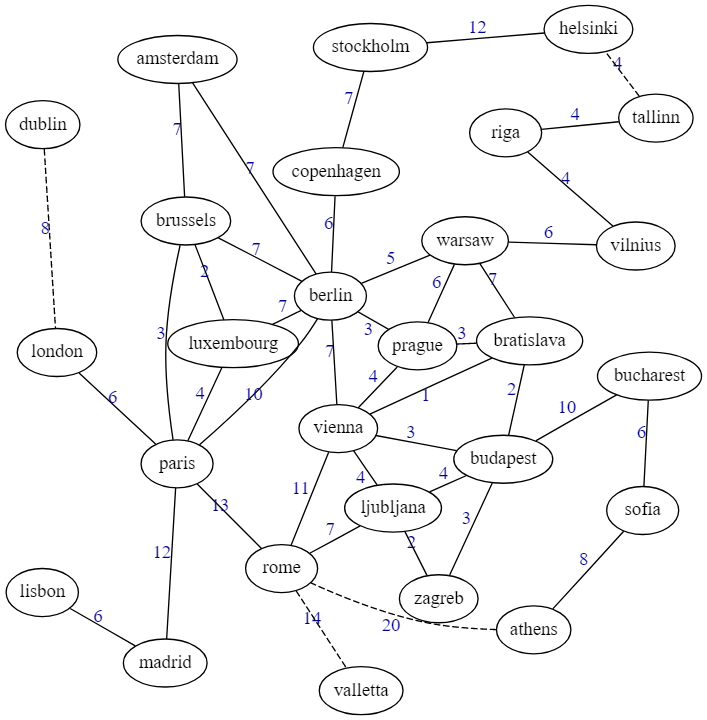
= O((m+n) log n)



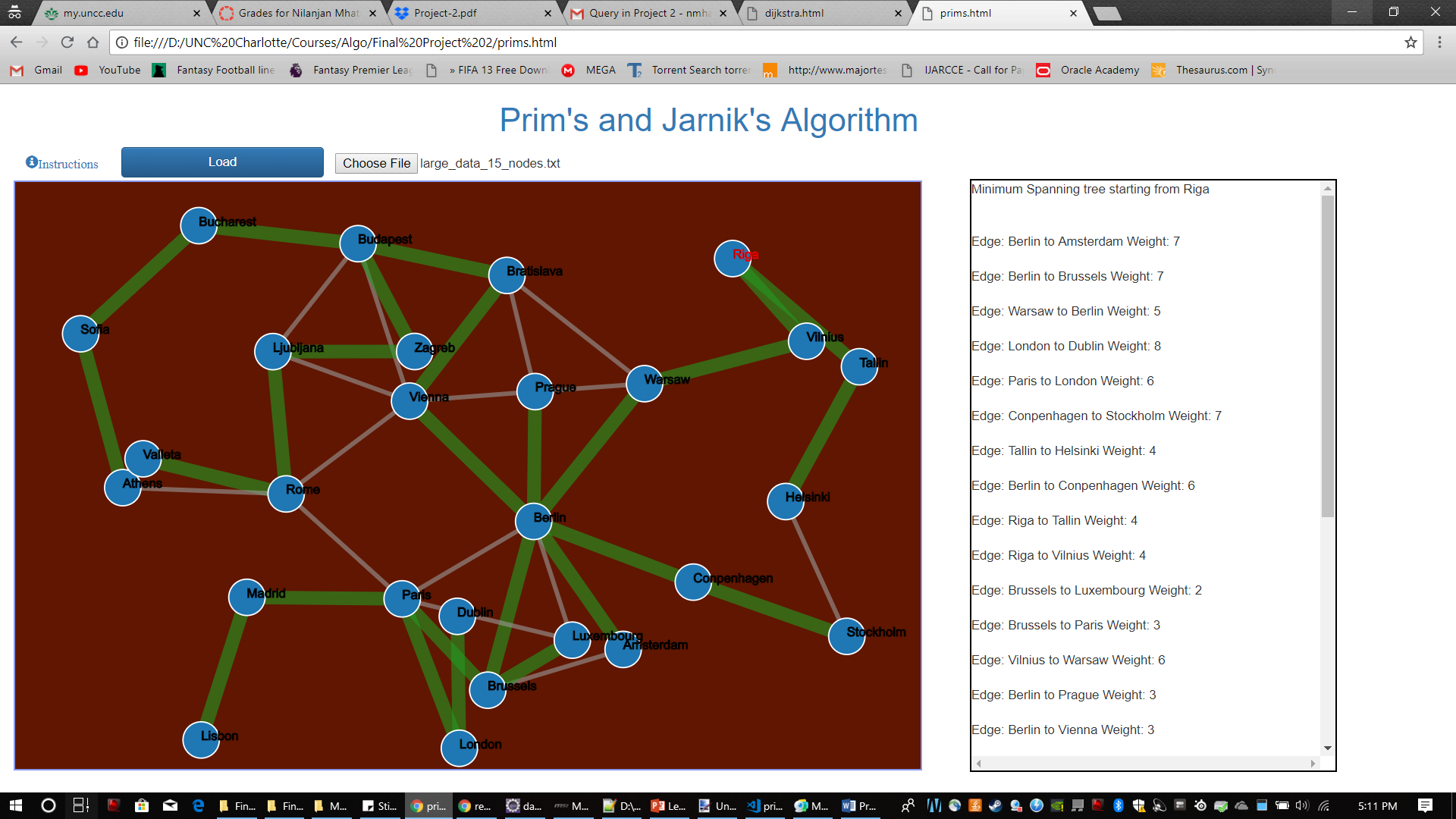
**Problem 2: Find the Minimum spanning tree (MST).**

**Prim’s and Jarnik’s Algorithm**

**Input Sample 1: large\_data\_27\_nodes.txt**



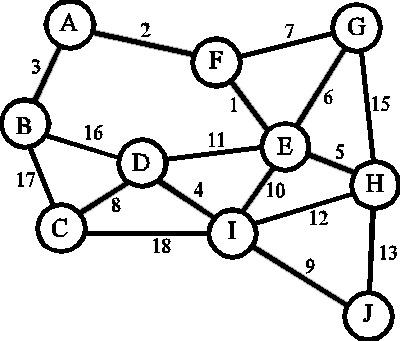
Output:



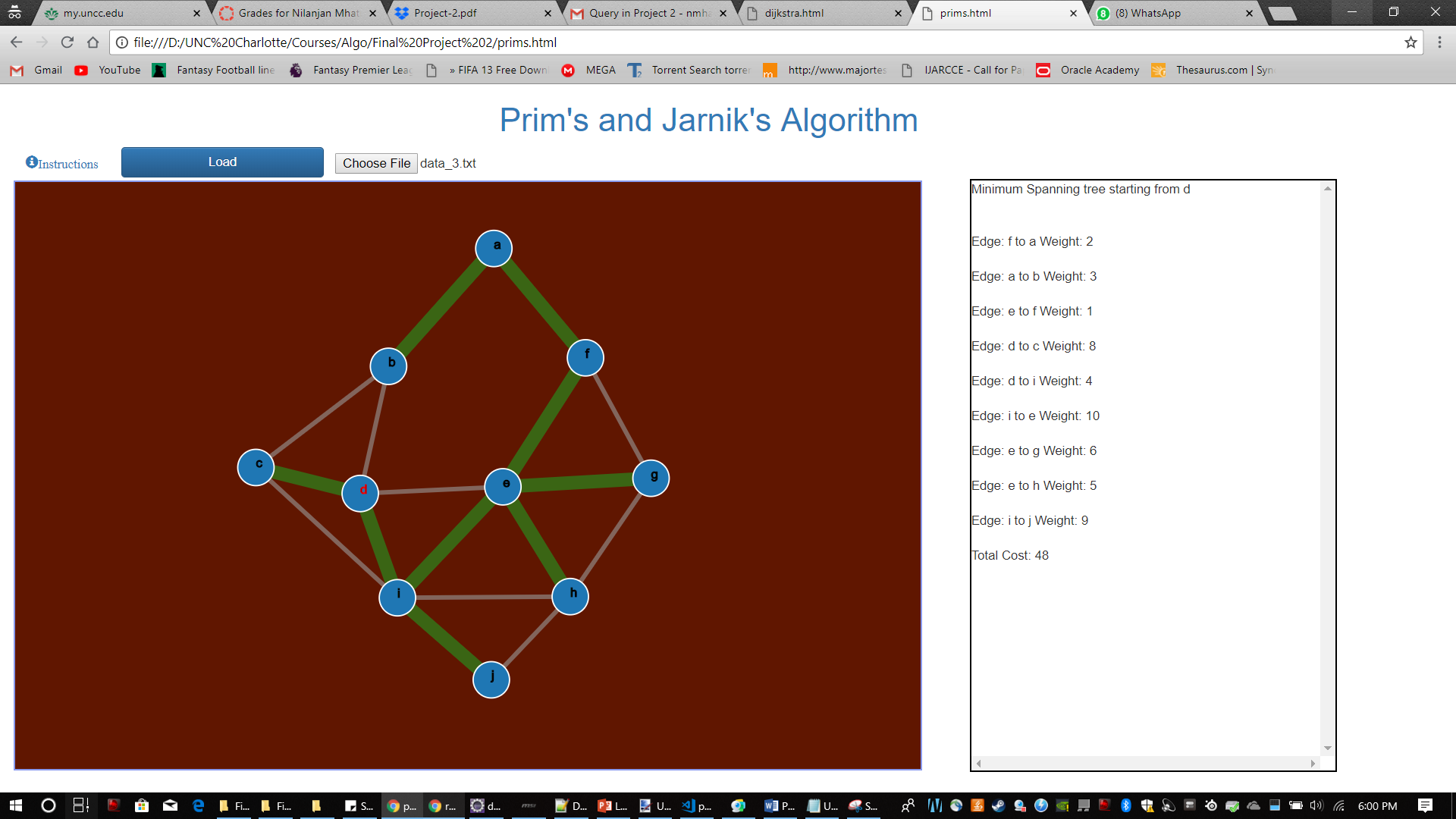
Minimum Spanning tree starting from Riga

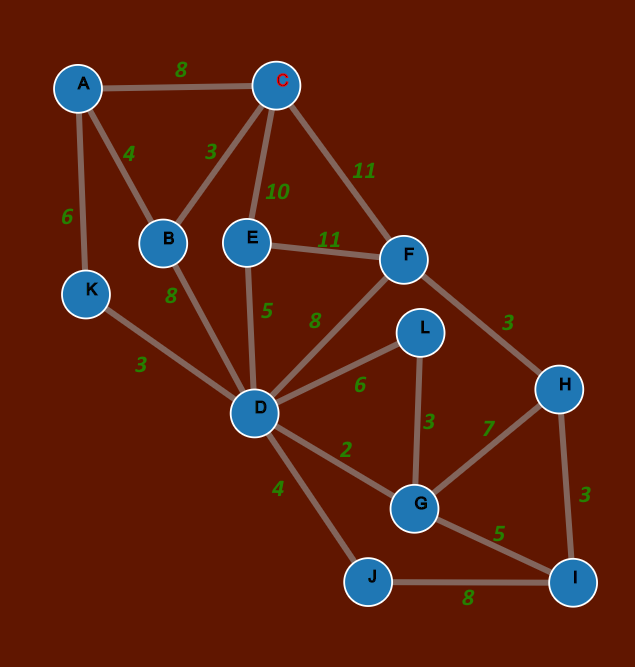
Edge: Berlin to Amsterdam Weight: 7  
  
Edge: Berlin to Brussels Weight: 7  
  
Edge: Warsaw to Berlin Weight: 5  
  
Edge: London to Dublin Weight: 8  
  
Edge: Paris to London Weight: 6  
  
Edge: Conpenhagen to Stockholm Weight: 7  
  
Edge: Tallin to Helsinki Weight: 4  
  
Edge: Berlin to Conpenhagen Weight: 6  
  
Edge: Riga to Tallin Weight: 4  
  
Edge: Riga to Vilnius Weight: 4  
  
Edge: Brussels to Luxembourg Weight: 2  
  
Edge: Brussels to Paris Weight: 3  
  
Edge: Vilnius to Warsaw Weight: 6  
  
Edge: Berlin to Prague Weight: 3  
  
Edge: Berlin to Vienna Weight: 3  
  
Edge: Vienna to Bratislava Weight: 1  
  
Edge: Bratislava to Budapest Weight: 2  
  
Edge: Budapest to Bucharest Weight: 10  
  
Edge: Bucharest to Sofia Weight: 6  
  
Edge: Sofia to Athens Weight: 8  
  
Edge: Madrid to Lisbon Weight: 6  
  
Edge: Paris to Madrid Weight: 12  
  
Edge: Ljubljana to Rome Weight: 7  
  
Edge: Rome to Valleta Weight: 14  
  
Edge: Zagreb to Ljubljana Weight: 2  
  
Edge: Budapest to Zagreb Weight: 3  
  
Total Cost: 146

**Input Sample 2: - data\_10\_nodes.txt**

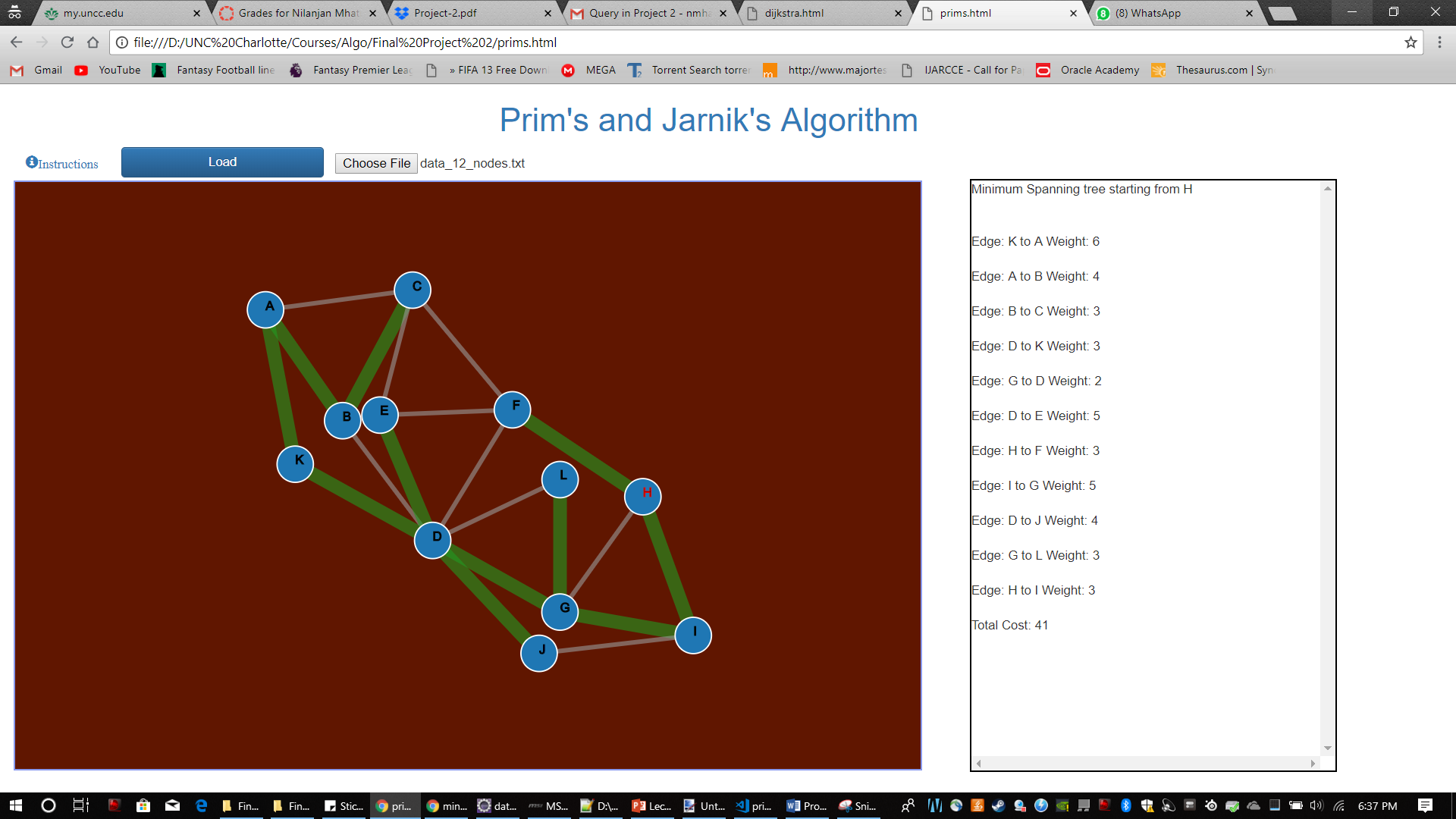


Output:



**Input Sample 3**: - **data\_12\_nodes.txt** 

Output:-



Runtime Analysis (Prim's and Jarnik’s Algorithm): -

Priority Queue is used that uses heap structure.

Each vertex insertion and the corresponding heapify operation

takes ‘O(log n)’ time. ‘Total = n log n’

Each vertex removal and the corresponding heapify operation

takes ‘O(log n)’ time. ‘Total = n log n’

Each vertex modified along with heapify operation takes ‘O(log n)’ time

Each vertex will be modified for maximum of its incident edges, i.e. for total ‘m’ edges.

Total = m log n

The total time = n log n + m log n + n log n

= O((m+n) log n)

