Project 2 Report

Joey Parker

University of North Carolina at Charlotte

ITCS 6114-81

Dr. Dewan Ahmed

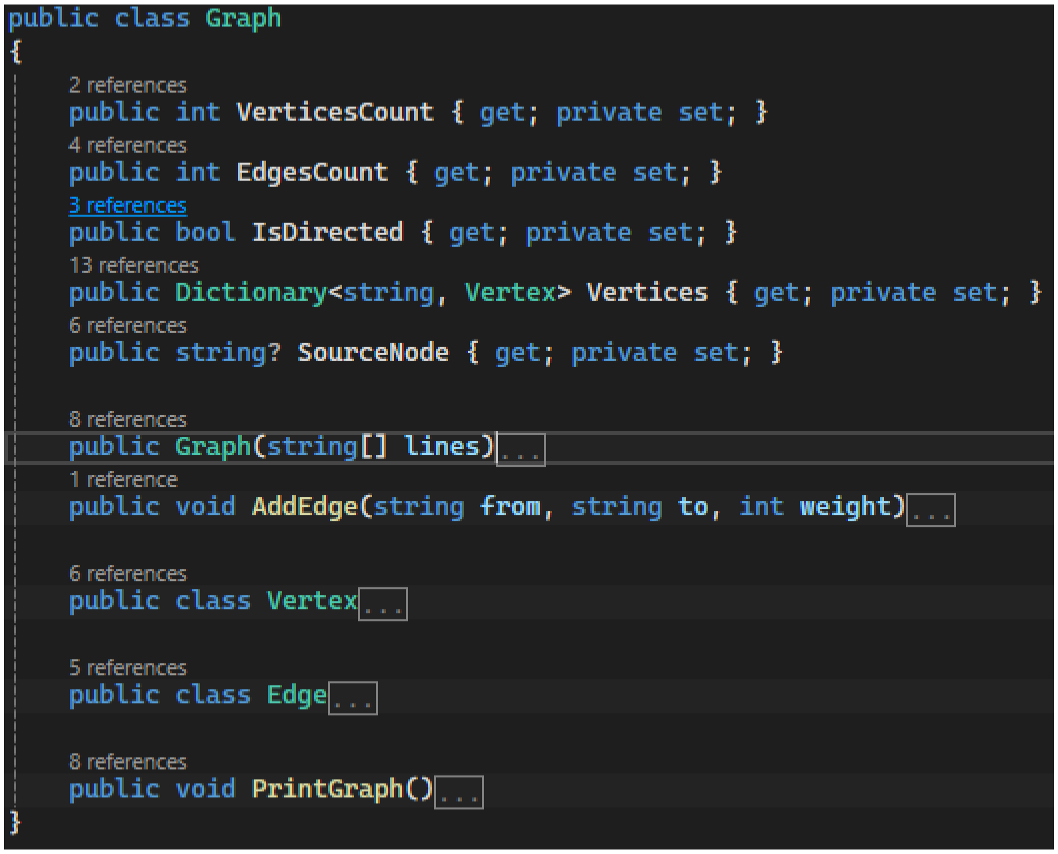
July 28, 2024

Project GitHub Repository: https://github.com/Ninjajkl/ITCS-6114-Project-2/

Table of Contents

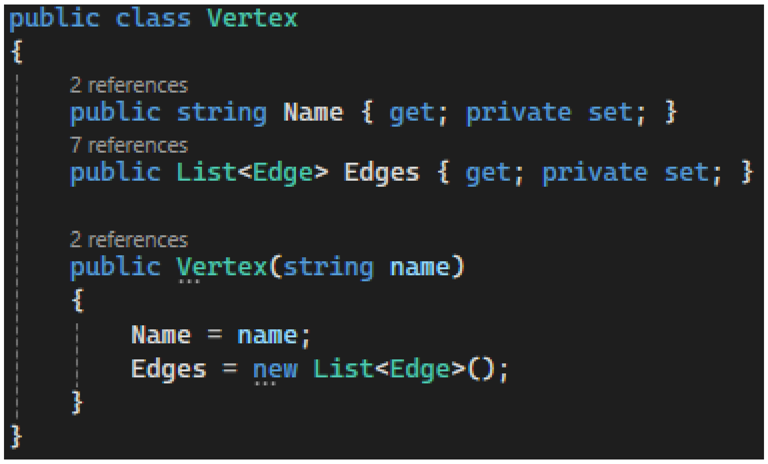
1 Graph Data Structure

My graph structure has five members. The Vertices Count (int), the Edges Count (int), whether the graph is directed or undirected (bool), the name of the Source node/vertex (string), and a <string, Vertex> Directory called Vertices that holds all the vertices in the graph, given their string name.



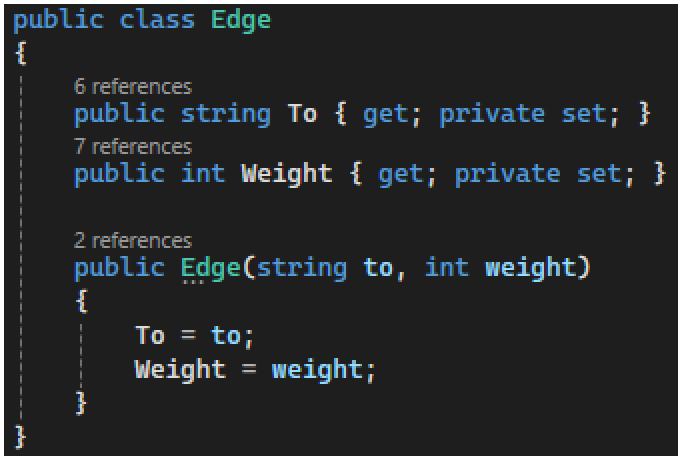
```
public class Graph
{
    2 references
    public int VerticesCount { get; private set; }
    4 references
    public int EdgesCount { get; private set; }
    3 references
    public bool IsDirected { get; private set; }
    13 references
    public Dictionary<string, Vertex> Vertices { get; private set; }
    6 references
    public string? SourceNode { get; private set; }

    8 references
    public Graph(string[] lines)...
    1 reference
    public void AddEdge(string from, string to, int weight)...

    6 references
    public class Vertex...

    5 references
    public class Edge...

    8 references
    public void PrintGraph()...
}
```

Each Vertex contains its Name (string) and a list of Edges originating at the Vertex.



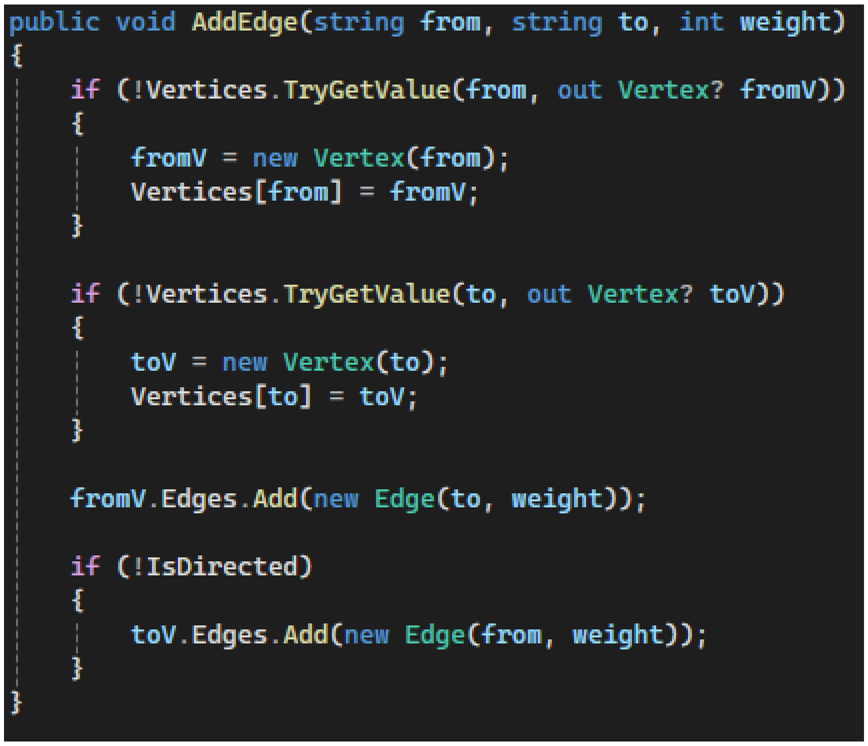
```
public class Vertex
{
    2 references
    public string Name { get; private set; }
    7 references
    public List<Edge> Edges { get; private set; }

    2 references
    public Vertex(string name)
    {
        Name = name;
        Edges = new List<Edge>();
    }
}
```

Each Edge contains the To Vertex Name (string) and the Weight of the Edge (int).



```
public class Edge
{
    6 references
    public string To { get; private set; }
    7 references
    public int Weight { get; private set; }

    2 references
    public Edge(string to, int weight)
    {
        To = to;
        Weight = weight;
    }
}
```

To make the graph possible for undirected edges (as the edges are only stored by one of the connected vertices), each time we add an edge to a vertex, we also add the edge again to the connected vertex. This does not happen for directed graph generation.



```
public void AddEdge(string from, string to, int weight)
{
    if (!Vertices.TryGetValue(from, out Vertex? fromV))
    {
        fromV = new Vertex(from);
        Vertices[from] = fromV;
    }

    if (!Vertices.TryGetValue(to, out Vertex? toV))
    {
        toV = new Vertex(to);
        Vertices[to] = toV;
    }

    fromV.Edges.Add(new Edge(to, weight));

    if (!IsDirected)
    {
        toV.Edges.Add(new Edge(from, weight));
    }
}
```

2 Note on Input Format + Instructions to Run

The input format is the same as the given file format, but the last line depicting the Source Node/Vertex is Not Optional.

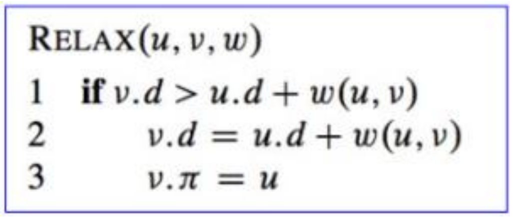
To run the program, follow the instructions in the README.md

3 Shortest Path

3.1 Pseudocode

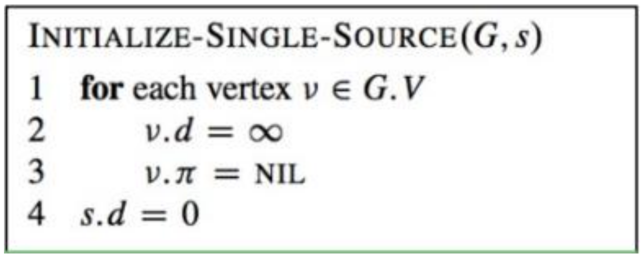
The Pseudocode for Dijkstra's algorithm is given in 3 parts: Relax, Initialize-Single-Source, and the main body.

The pseudocode for Relax is as follows (taken from the lecture slides):



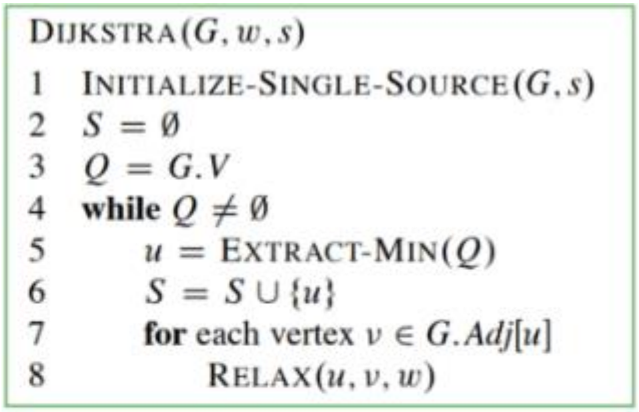
```
RELAX($u, v, w$)
if $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$
```

The pseudocode for Initialize-Single-Source is as follows (taken from the lecture slides):



```
INITIALIZE-SINGLE-SOURCE($G, s$)
for each vertex $v \in G.V$
    $v.d = \infty$
    $v.\pi = \text{NIL}$
$s.d = 0$
```

The pseudocode for the main body is as follows (taken from the lecture slides):



```
DIJKSTRA($G, w, s$)
INITIALIZE-SINGLE-SOURCE($G, s$)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        RELAX($u, v, w$)
```

3.2 Pseudocode Runtime

V will be the number of vertices. E will be the number of edges. The Initialize-Single-Source operation takes time O(V). The time to build the binary min-heap Q is O(V log V). The Extract-Min operation for a min-heap priority queue is O(log V). The Relax operation takes time O(log V), and there can be at most O(E) such operations.

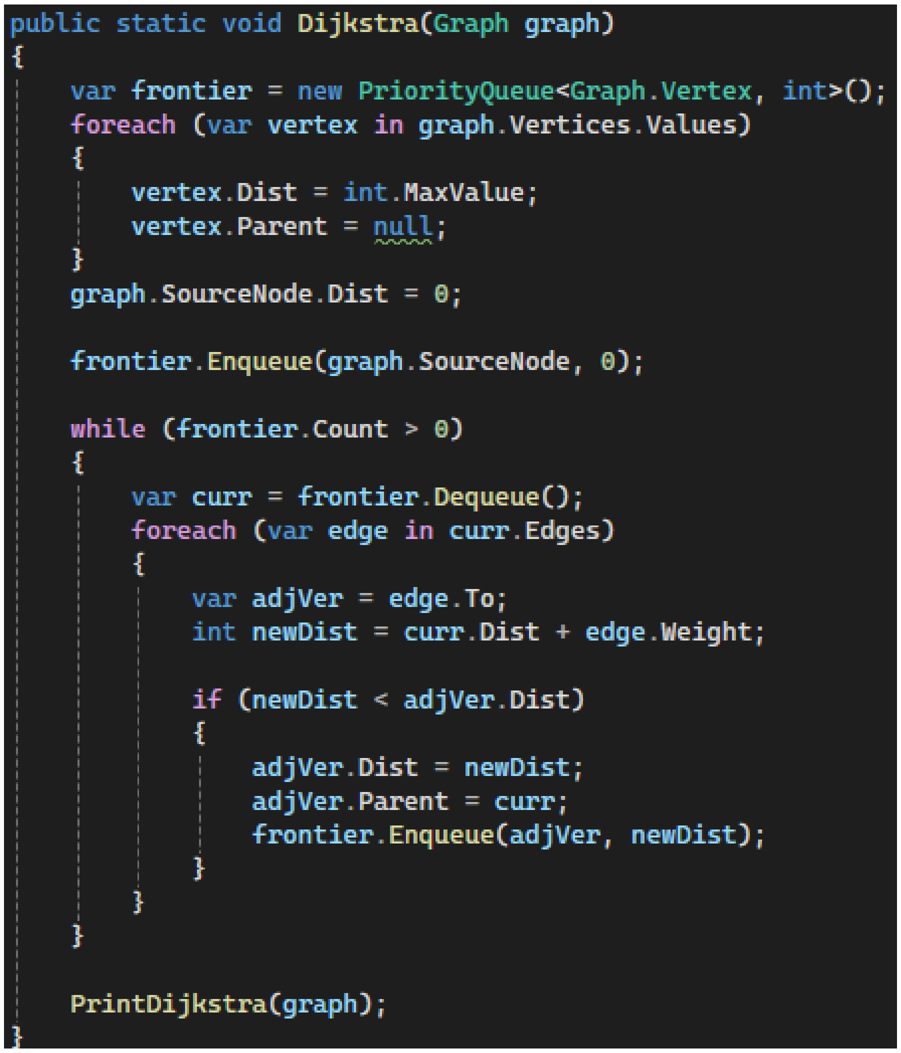
The time complexity of Dijkstra's when using a min-heap priority queue is therefore O(V log V + E log V) or O((V + E) log V). If all vertices are reachable from the source it is O(E log V).

3.3 Runtime of Actual Implementation

My implementation of Dijkstra's algorithm used the min-heap priority queue data structure for the frontier. As mentioned above, this gives my implementation of Dijkstra's Algorithm an O((V + E) log V) runtime.

3.4 Implementation Code

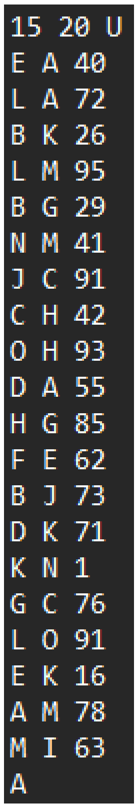
Here is the code for my Implementation of Dijkstra's



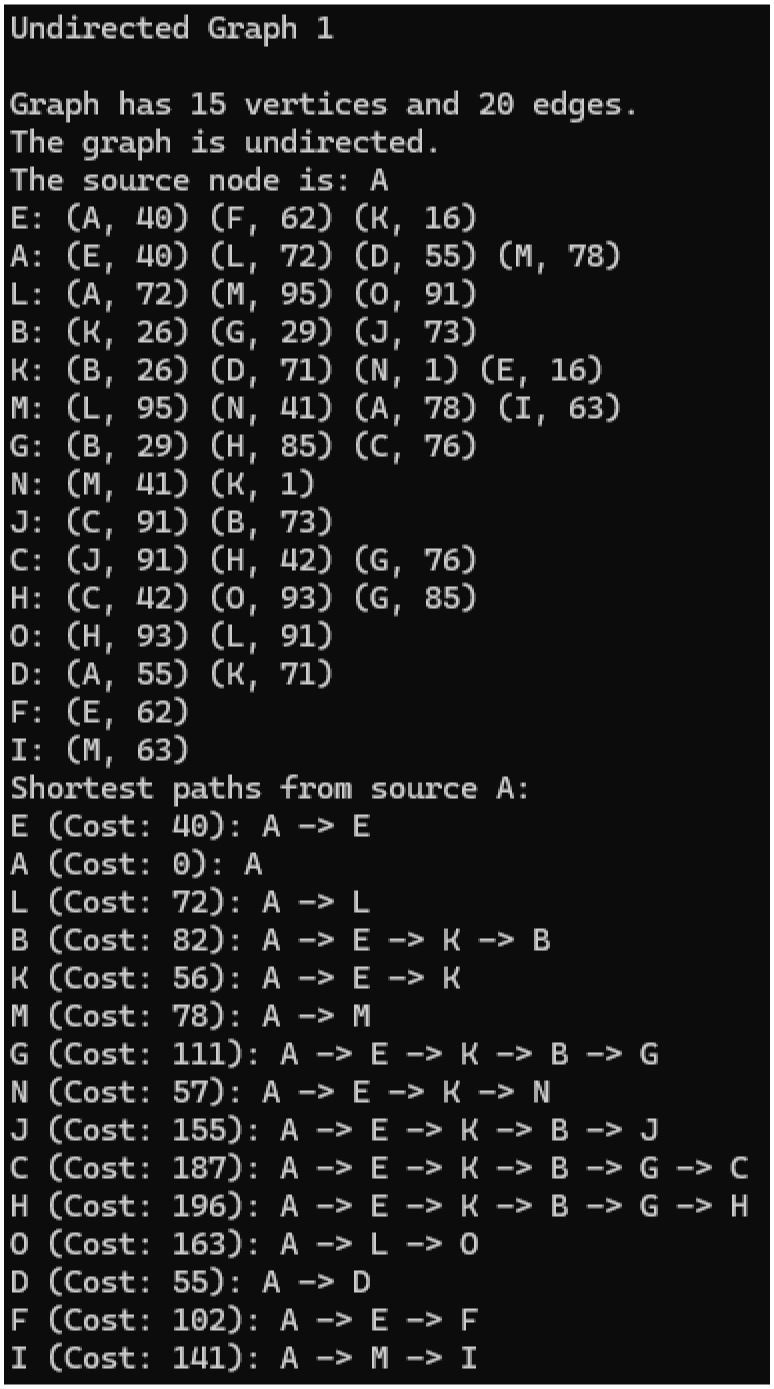
```
public static void Dijkstra(Graph graph)
{
    var frontier = new PriorityQueue<Graph.Vertex, int>();
    foreach (var vertex in graph.Vertices.Values)
    {
        vertex.Dist = int.MaxValue;
        vertex.Parent = null;
    }
    graph.SourceNode.Dist = 0;

    frontier.Enqueue(graph.SourceNode, 0);

    while (frontier.Count > 0)
    {
        var curr = frontier.Dequeue();
        foreach (var edge in curr.Edges)
        {
            var adjVer = edge.To;
            int newDist = curr.Dist + edge.Weight;

            if (newDist < adjVer.Dist)
            {
                adjVer.Dist = newDist;
                adjVer.Parent = curr;
                frontier.Enqueue(adjVer, newDist);
            }
        }
    }

    PrintDijkstra(graph);
}
```

3.5 Sample Graphs

For this section I will have the Input on the left, and the Output for that Graph on the right. The first two graphs are undirected, and the last two are directed.

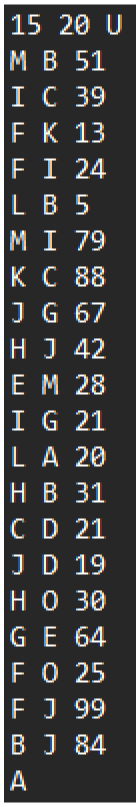
1. 

```
15 20 U
E A 40
L A 72
B K 26
L M 95
B G 29
N M 41
J C 91
C H 42
O H 93
D A 55
H G 85
F E 62
B J 73
D K 71
K N 1
G C 76
L O 91
E K 16
A M 78
M I 63
A
```

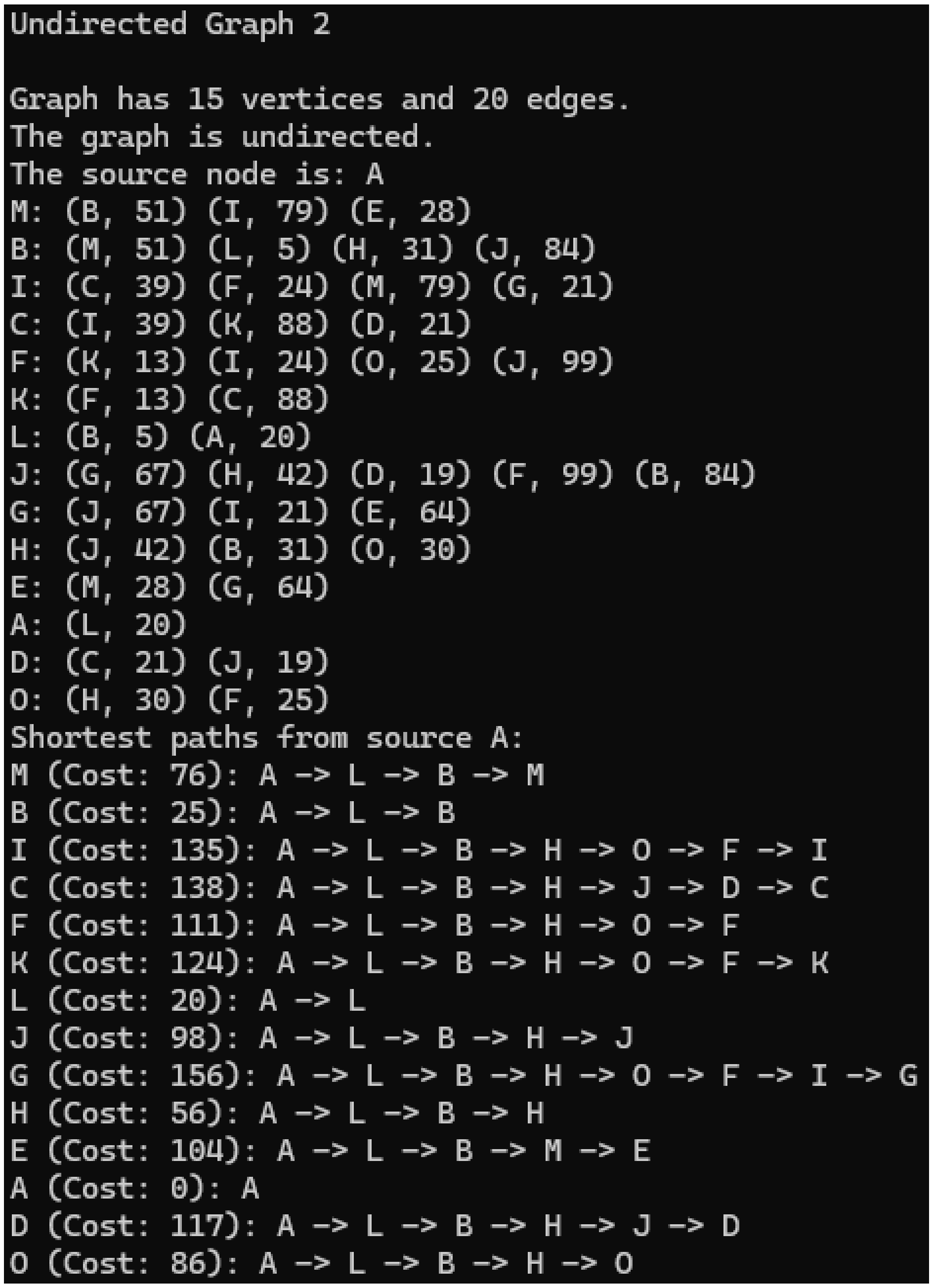


```
Undirected Graph 1

Graph has 15 vertices and 20 edges.
The graph is undirected.
The source node is: A
E: (A, 40) (F, 62) (K, 16)
A: (E, 40) (L, 72) (D, 55) (M, 78)
L: (A, 72) (M, 95) (O, 91)
B: (K, 26) (G, 29) (J, 73)
K: (B, 26) (D, 71) (N, 1) (E, 16)
M: (L, 95) (N, 41) (A, 78) (I, 63)
G: (B, 29) (H, 85) (C, 76)
N: (M, 41) (K, 1)
J: (C, 91) (B, 73)
C: (J, 91) (H, 42) (G, 76)
H: (C, 42) (O, 93) (G, 85)
O: (H, 93) (L, 91)
D: (A, 55) (K, 71)
F: (E, 62)
I: (M, 63)
Shortest paths from source A:
E (Cost: 40): A -> E
A (Cost: 0): A
L (Cost: 72): A -> L
B (Cost: 82): A -> E -> K -> B
K (Cost: 56): A -> E -> K
M (Cost: 78): A -> M
G (Cost: 111): A -> E -> K -> B -> G
N (Cost: 57): A -> E -> K -> N
J (Cost: 155): A -> E -> K -> B -> J
C (Cost: 187): A -> E -> K -> B -> G -> C
H (Cost: 196): A -> E -> K -> B -> G -> H
O (Cost: 163): A -> L -> O
D (Cost: 55): A -> D
F (Cost: 102): A -> E -> F
I (Cost: 141): A -> M -> I
```

2.

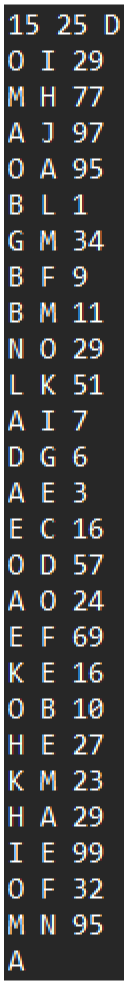


```
15 20 U
M B 51
I C 39
F K 13
F I 24
L B 5
M I 79
K C 88
J G 67
H J 42
E M 28
I G 21
L A 20
H B 31
C D 21
J D 19
H O 30
G E 64
F O 25
F J 99
B J 84
A
```

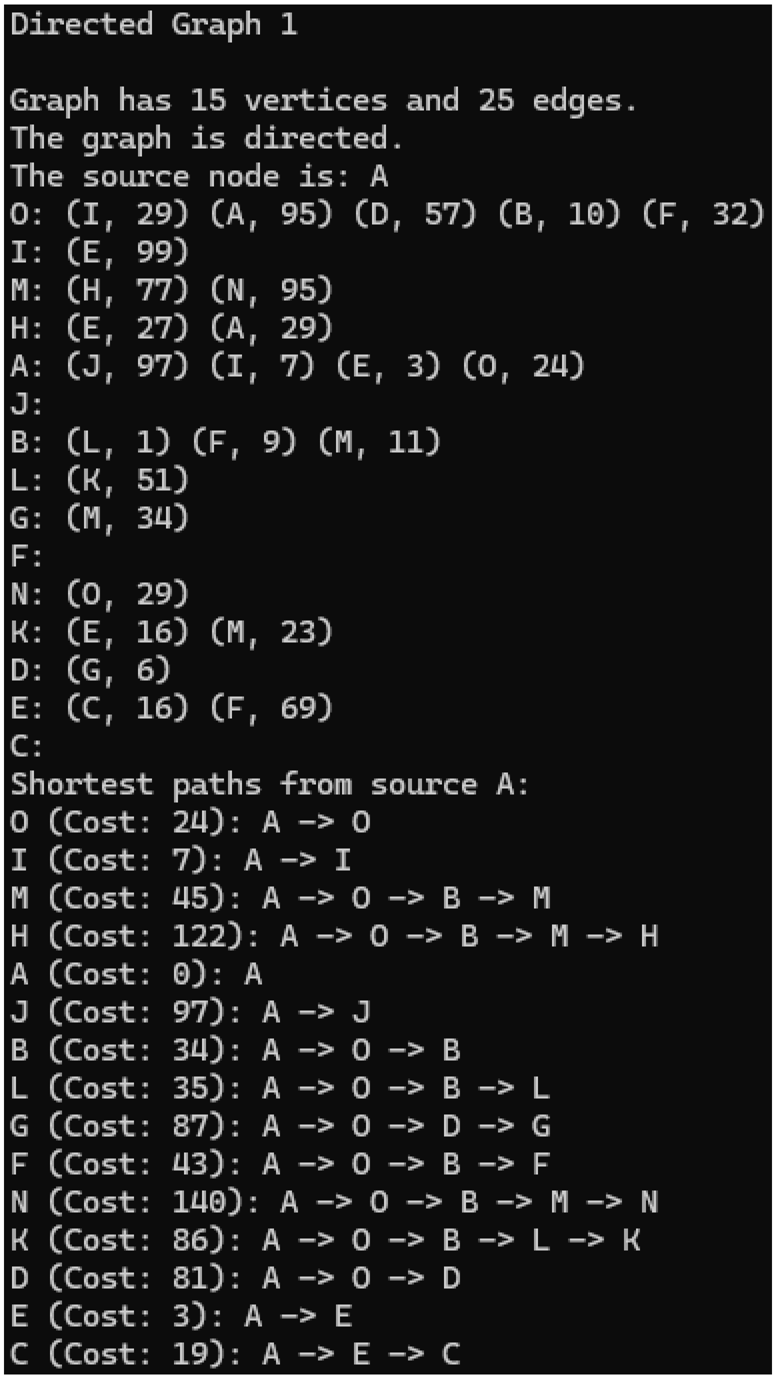


```
Undirected Graph 2

Graph has 15 vertices and 20 edges.
The graph is undirected.
The source node is: A
M: (B, 51) (I, 79) (E, 28)
B: (M, 51) (L, 5) (H, 31) (J, 84)
I: (C, 39) (F, 24) (M, 79) (G, 21)
C: (I, 39) (K, 88) (D, 21)
F: (K, 13) (I, 24) (O, 25) (J, 99)
K: (F, 13) (C, 88)
L: (B, 5) (A, 20)
J: (G, 67) (H, 42) (D, 19) (F, 99) (B, 84)
G: (J, 67) (I, 21) (E, 64)
H: (J, 42) (B, 31) (O, 30)
E: (M, 28) (G, 64)
A: (L, 20)
D: (C, 21) (J, 19)
O: (H, 30) (F, 25)
Shortest paths from source A:
M (Cost: 76): A -> L -> B -> M
B (Cost: 25): A -> L -> B
I (Cost: 135): A -> L -> B -> H -> O -> F -> I
C (Cost: 138): A -> L -> B -> H -> J -> D -> C
F (Cost: 111): A -> L -> B -> H -> O -> F
K (Cost: 124): A -> L -> B -> H -> O -> F -> K
L (Cost: 20): A -> L
J (Cost: 98): A -> L -> B -> H -> J
G (Cost: 156): A -> L -> B -> H -> O -> F -> I -> G
H (Cost: 56): A -> L -> B -> H
E (Cost: 104): A -> L -> B -> M -> E
A (Cost: 0): A
D (Cost: 117): A -> L -> B -> H -> J -> D
O (Cost: 86): A -> L -> B -> H -> O
```

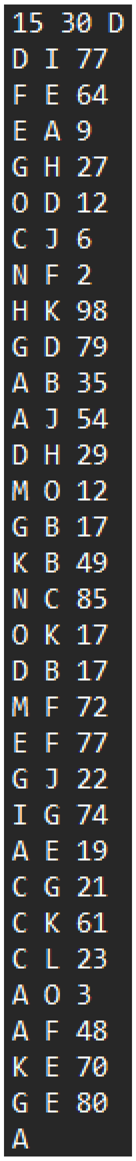
3.



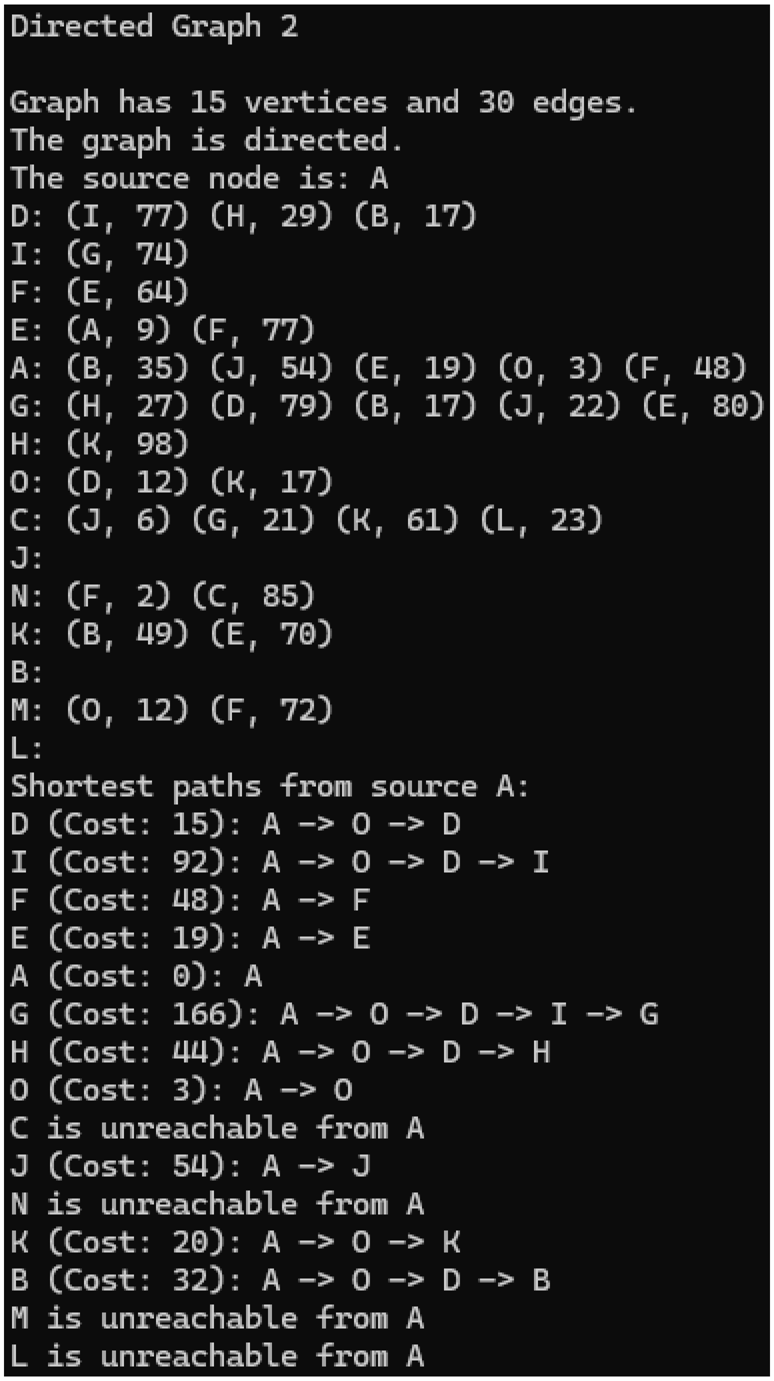
```
15 25 D
O I 29
M H 77
A J 97
O A 95
B L 1
G M 34
B F 9
B M 11
N O 29
L K 51
A I 7
D G 6
A E 3
E C 16
O D 57
A O 24
E F 69
K E 16
O B 10
H E 27
K M 23
H A 29
I E 99
O F 32
M N 95
A
```



```
Directed Graph 1

Graph has 15 vertices and 25 edges.
The graph is directed.
The source node is: A
O: (I, 29) (A, 95) (D, 57) (B, 10) (F, 32)
I: (E, 99)
M: (H, 77) (N, 95)
H: (E, 27) (A, 29)
A: (J, 97) (I, 7) (E, 3) (O, 24)
J:
B: (L, 1) (F, 9) (M, 11)
L: (K, 51)
G: (M, 34)
F:
N: (O, 29)
K: (E, 16) (M, 23)
D: (G, 6)
E: (C, 16) (F, 69)
C:
Shortest paths from source A:
O (Cost: 24): A -> O
I (Cost: 7): A -> I
M (Cost: 45): A -> O -> B -> M
H (Cost: 122): A -> O -> B -> M -> H
A (Cost: 0): A
J (Cost: 97): A -> J
B (Cost: 34): A -> O -> B
L (Cost: 35): A -> O -> B -> L
G (Cost: 87): A -> O -> D -> G
F (Cost: 43): A -> O -> B -> F
N (Cost: 140): A -> O -> B -> M -> N
K (Cost: 86): A -> O -> B -> L -> K
D (Cost: 81): A -> O -> D
E (Cost: 3): A -> E
C (Cost: 19): A -> E -> C
```

4. 

```
15 30 D
D I 77
F E 64
E A 9
G H 27
O D 12
C J 6
N F 2
H K 98
G D 79
A B 35
A J 54
D H 29
M O 12
G B 17
K B 49
N C 85
O K 17
D B 17
M F 72
E F 77
G J 22
I G 74
A E 19
C G 21
C K 61
C L 23
A O 3
A F 48
K E 70
G E 80
A
```

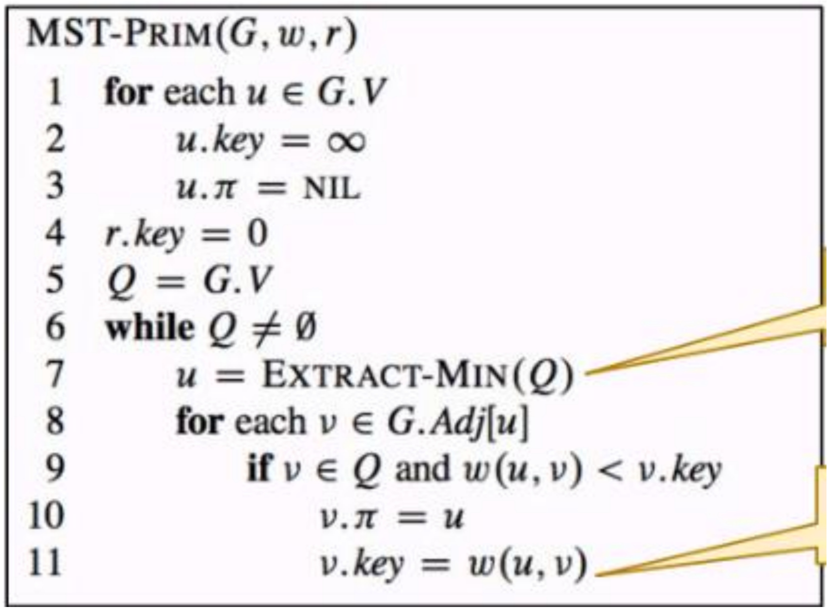


```
Directed Graph 2

Graph has 15 vertices and 30 edges.
The graph is directed.
The source node is: A
D: (I, 77) (H, 29) (B, 17)
I: (G, 74)
F: (E, 64)
E: (A, 9) (F, 77)
A: (B, 35) (J, 54) (E, 19) (O, 3) (F, 48)
G: (H, 27) (D, 79) (B, 17) (J, 22) (E, 80)
H: (K, 98)
O: (D, 12) (K, 17)
C: (J, 6) (G, 21) (K, 61) (L, 23)
J:
N: (F, 2) (C, 85)
K: (B, 49) (E, 70)
B:
M: (O, 12) (F, 72)
L:
Shortest paths from source A:
D (Cost: 15): A -> O -> D
I (Cost: 92): A -> O -> D -> I
F (Cost: 48): A -> F
E (Cost: 19): A -> E
A (Cost: 0): A
G (Cost: 166): A -> O -> D -> I -> G
H (Cost: 44): A -> O -> D -> H
O (Cost: 3): A -> O
C is unreachable from A
J (Cost: 54): A -> J
N is unreachable from A
K (Cost: 20): A -> O -> K
B (Cost: 32): A -> O -> D -> B
M is unreachable from A
L is unreachable from A
```

4 Minimum Spanning Tree

4.1 Pseudocode

The Pseudocode for Prim’s algorithm is given in just one part (taken from the lecture slides):



```
MST-PRIM($G, w, r$)
for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = $ NIL
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = $ EXTRACT-MIN($Q$)
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$
```

4.2 Pseudocode Runtime

V will be the number of vertices. E will be the number of edges. The Initialization operation takes time O(V). The time to build the binary min-heap Q is O(V log V). The Extract-Min operation for a min-heap priority queue is O(log V).

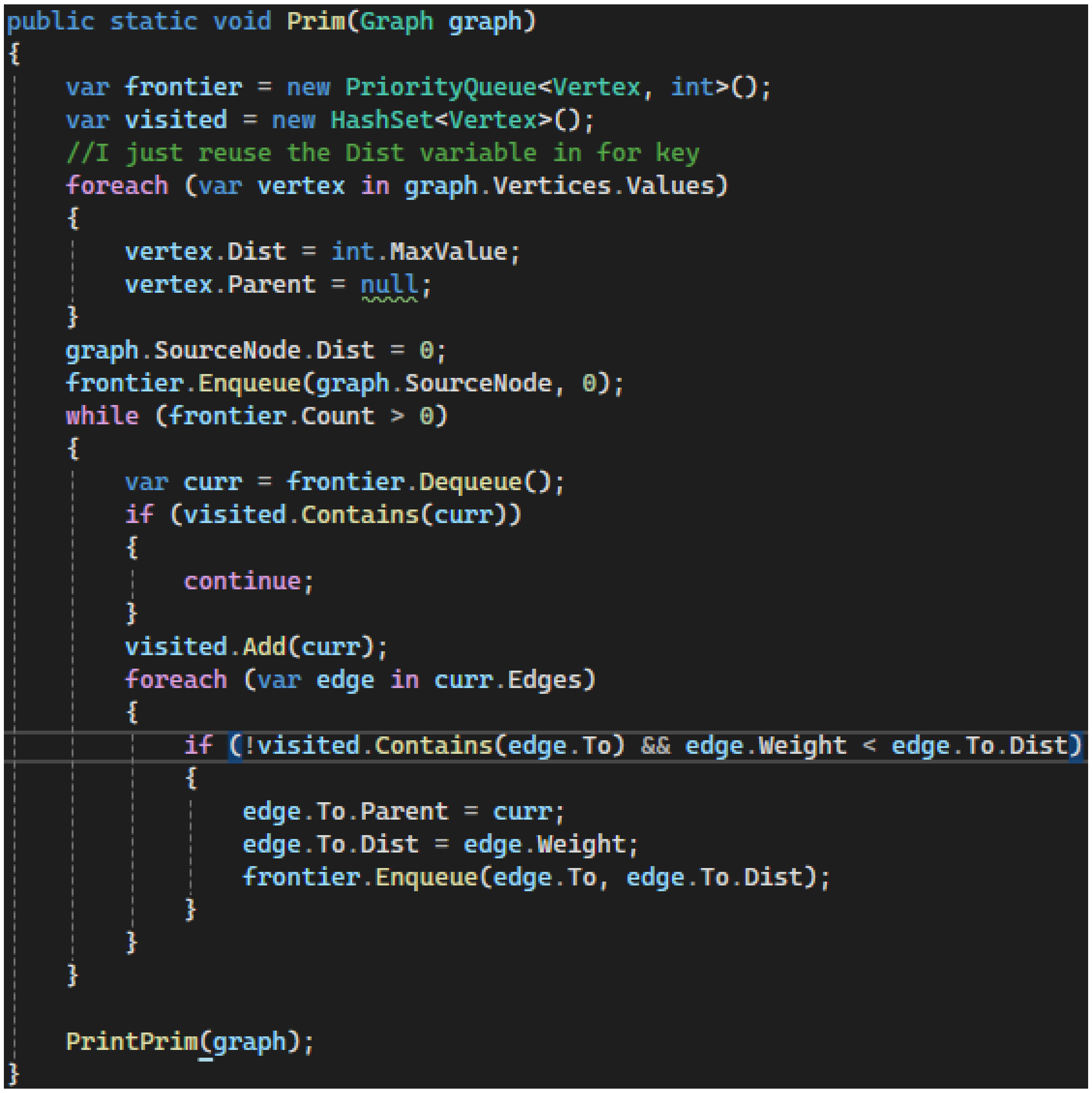
Updating the priority of a vertex's neighbors has a few sub-complexities within it. It first involves checking each edge connected to the vertex, which is worst case E edges. Considering each edge and changing its values takes O(log V) time. Since this happens at most E times, the total time complexity for this section is O(E log V)

The time complexity of Prim's when using a min-heap priority queue is therefore O(V log V + E log V) or O((V + E) log V).

4.3 Runtime of Actual Implementation

My implementation of Prim's algorithm used the min-heap priority queue data structure for the frontier. As mentioned above, this gives my implementation of Prim's Algorithm an O((V + E) log V) runtime.

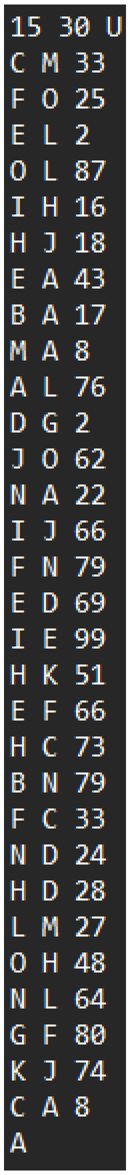
4.4 Implementation Code



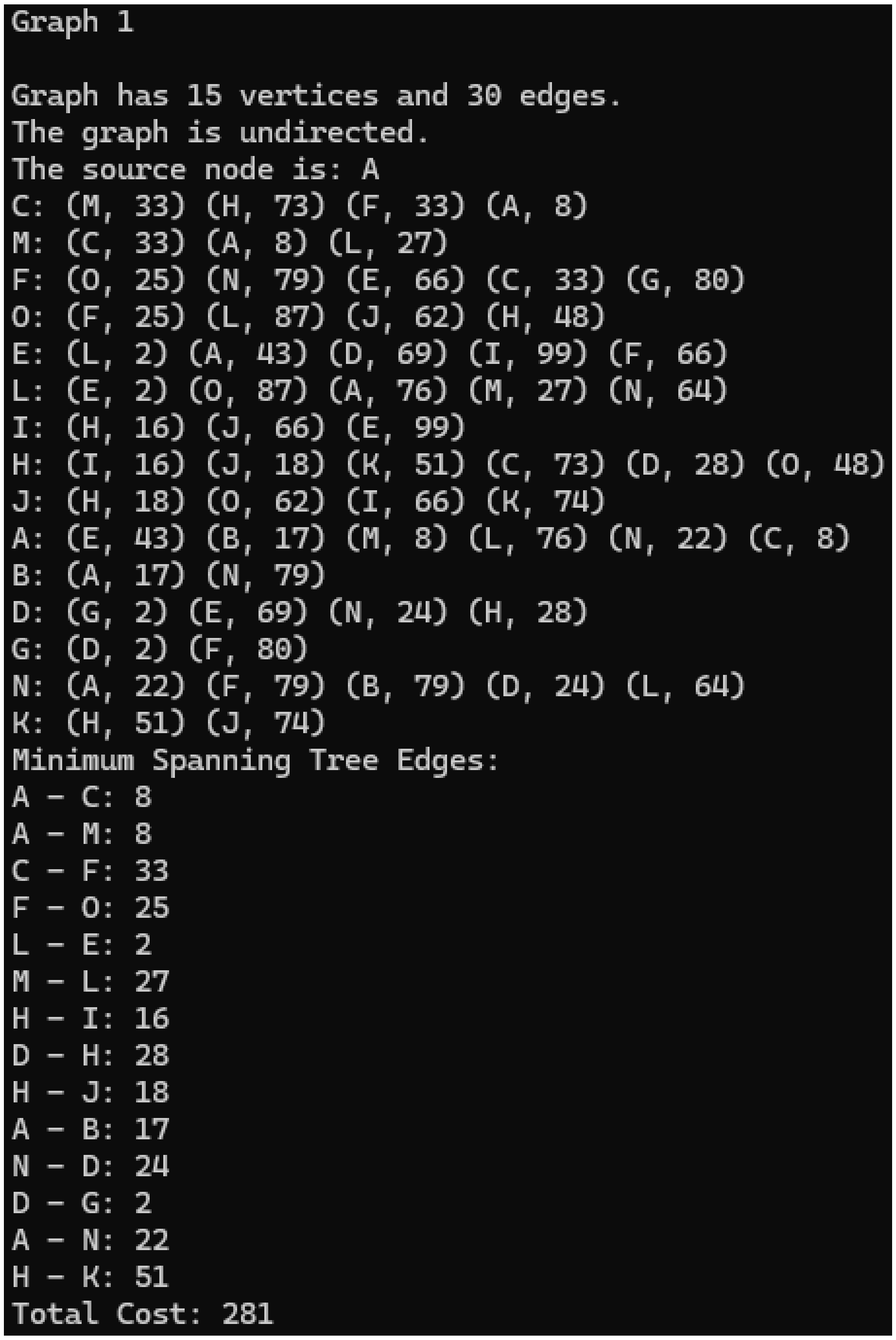
```
public static void Prim(Graph graph)
{
    var frontier = new PriorityQueue<Vertex, int>();
    var visited = new HashSet<Vertex>();
    //I just reuse the Dist variable in for key
    foreach (var vertex in graph.Vertices.Values)
    {
        vertex.Dist = int.MaxValue;
        vertex.Parent = null;
    }
    graph.SourceNode.Dist = 0;
    frontier.Enqueue(graph.SourceNode, 0);
    while (frontier.Count > 0)
    {
        var curr = frontier.Dequeue();
        if (visited.Contains(curr))
        {
            continue;
        }
        visited.Add(curr);
        foreach (var edge in curr.Edges)
        {
            if (!visited.Contains(edge.To) && edge.Weight < edge.To.Dist)
            {
                edge.To.Parent = curr;
                edge.To.Dist = edge.Weight;
                frontier.Enqueue(edge.To, edge.To.Dist);
            }
        }
    }

    PrintPrim(graph);
}
```

4.5 Sample Graphs

For this section I will have the Input on the left, and the Output for that Graph on the right.

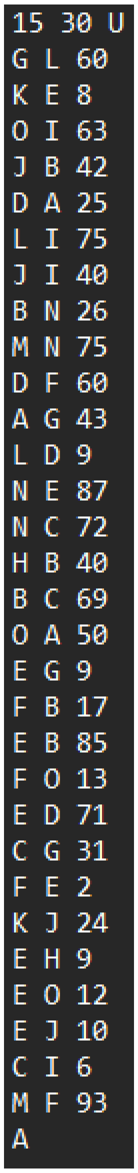
1. 

```
15 30 U
C M 33
F O 25
E L 2
O L 87
I H 16
H J 18
E A 43
B A 17
M A 8
A L 76
D G 2
J O 62
N A 22
I J 66
F N 79
E D 69
I E 99
H K 51
E F 66
H C 73
B N 79
F C 33
N D 24
H D 28
L M 27
O H 48
N L 64
G F 80
K J 74
C A 8
A
```

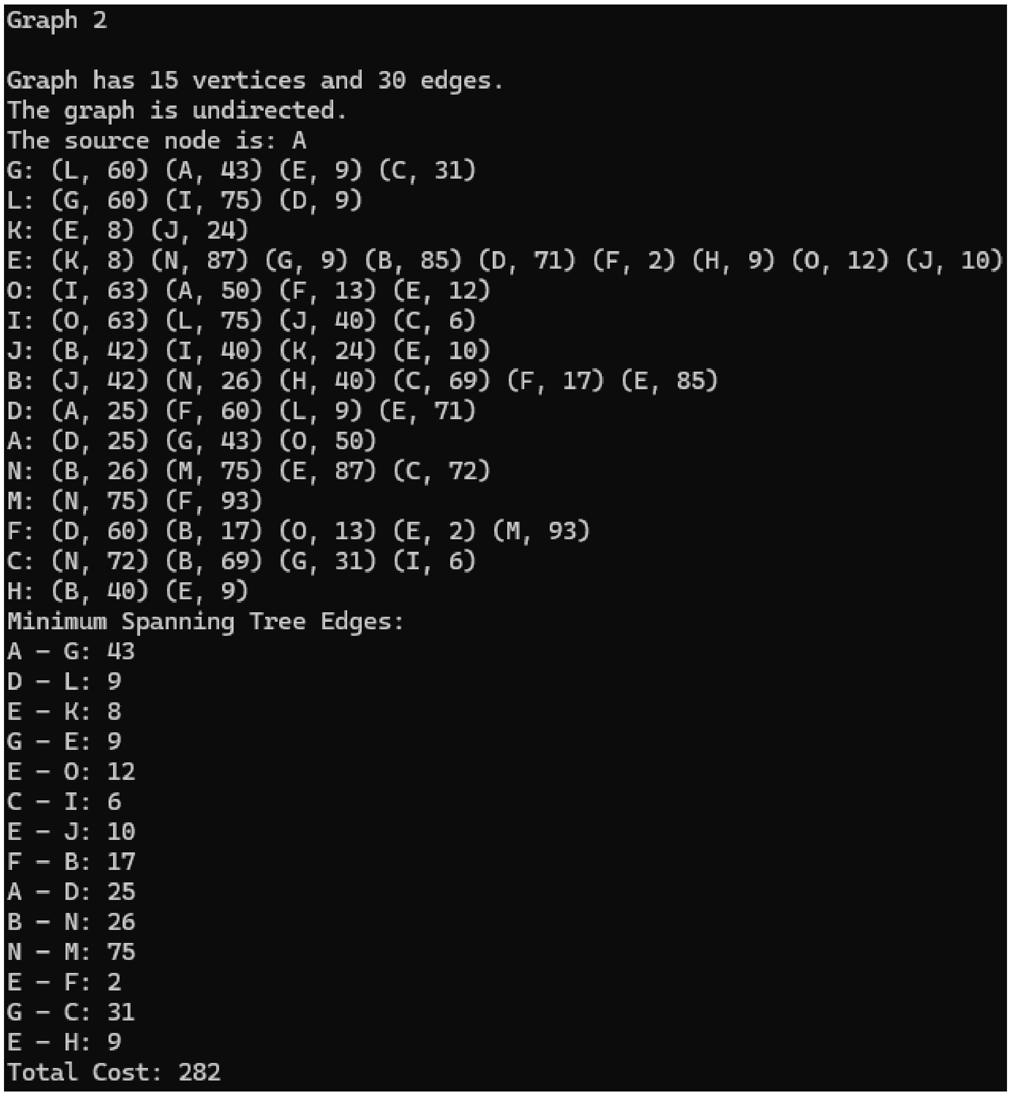


```
Graph 1

Graph has 15 vertices and 30 edges.
The graph is undirected.
The source node is: A
C: (M, 33) (H, 73) (F, 33) (A, 8)
M: (C, 33) (A, 8) (L, 27)
F: (O, 25) (N, 79) (E, 66) (C, 33) (G, 80)
O: (F, 25) (L, 87) (J, 62) (H, 48)
E: (L, 2) (A, 43) (D, 69) (I, 99) (F, 66)
L: (E, 2) (O, 87) (A, 76) (M, 27) (N, 64)
I: (H, 16) (J, 66) (E, 99)
H: (I, 16) (J, 18) (K, 51) (C, 73) (D, 28) (O, 48)
J: (H, 18) (O, 62) (I, 66) (K, 74)
A: (E, 43) (B, 17) (M, 8) (L, 76) (N, 22) (C, 8)
B: (A, 17) (N, 79)
D: (G, 2) (E, 69) (N, 24) (H, 28)
G: (D, 2) (F, 80)
N: (A, 22) (F, 79) (B, 79) (D, 24) (L, 64)
K: (H, 51) (J, 74)
Minimum Spanning Tree Edges:
A - C: 8
A - M: 8
C - F: 33
F - O: 25
L - E: 2
M - L: 27
H - I: 16
D - H: 28
H - J: 18
A - B: 17
N - D: 24
D - G: 2
A - N: 22
H - K: 51
Total Cost: 281
```

2.

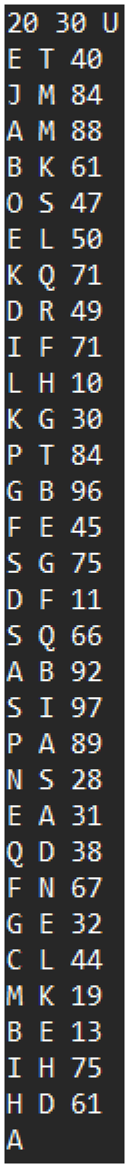


```
15 30 U
G L 60
K E 8
O I 63
J B 42
D A 25
L I 75
J I 40
B N 26
M N 75
D F 60
A G 43
L D 9
N E 87
N C 72
H B 40
B C 69
O A 50
E G 9
F B 17
E B 85
F O 13
E D 71
C G 31
F E 2
K J 24
E H 9
E O 12
E J 10
C I 6
M F 93
A
```

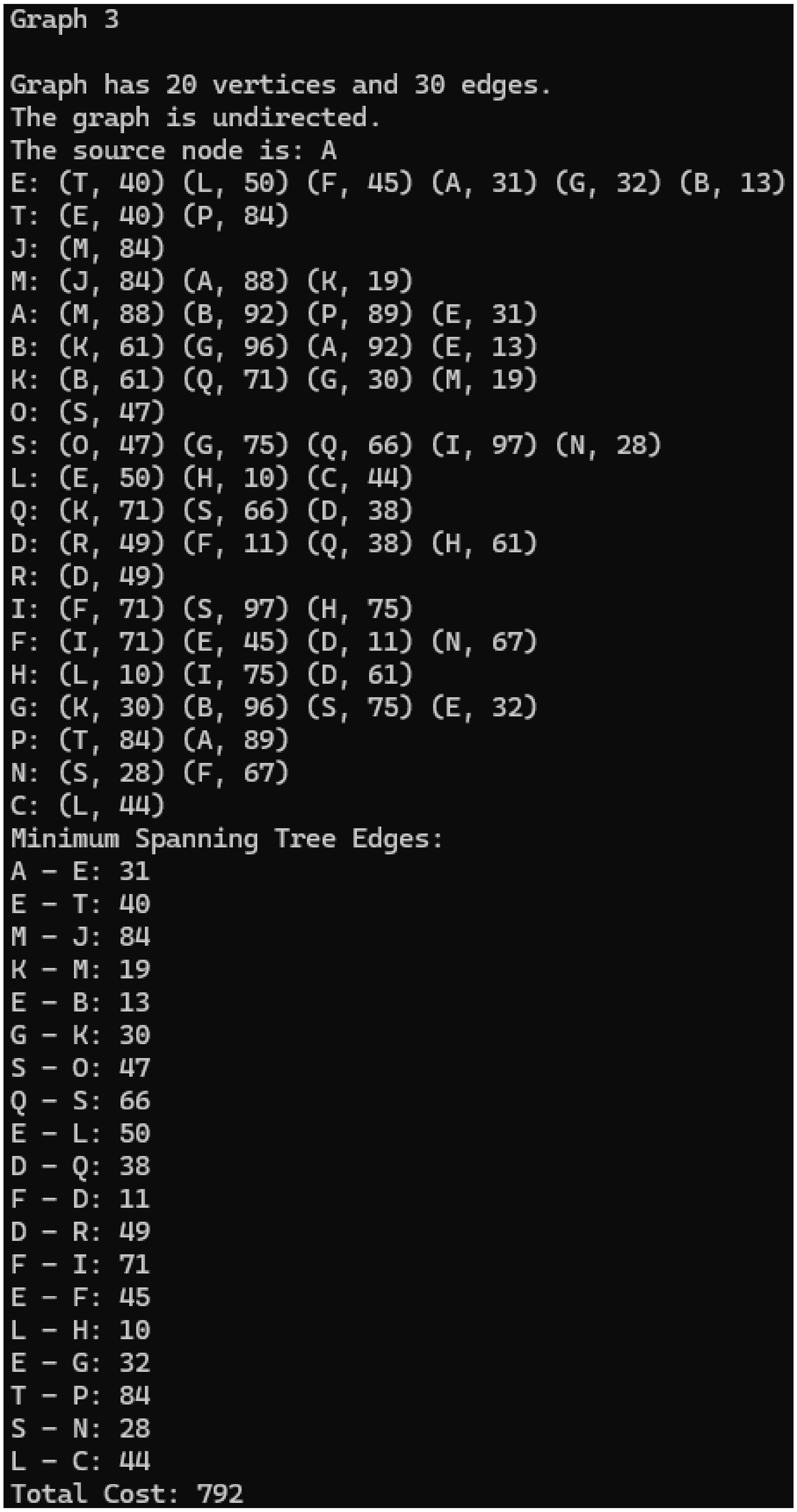


```
Graph 2

Graph has 15 vertices and 30 edges.
The graph is undirected.
The source node is: A
G: (L, 60) (A, 43) (E, 9) (C, 31)
L: (G, 60) (I, 75) (D, 9)
K: (E, 8) (J, 24)
E: (K, 8) (N, 87) (G, 9) (B, 85) (D, 71) (F, 2) (H, 9) (O, 12) (J, 10)
O: (I, 63) (A, 50) (F, 13) (E, 12)
I: (O, 63) (L, 75) (J, 40) (C, 6)
J: (B, 42) (I, 40) (K, 24) (E, 10)
B: (J, 42) (N, 26) (H, 40) (C, 69) (F, 17) (E, 85)
D: (A, 25) (F, 60) (L, 9) (E, 71)
A: (D, 25) (G, 43) (O, 50)
N: (B, 26) (M, 75) (E, 87) (C, 72)
M: (N, 75) (F, 93)
F: (D, 60) (B, 17) (O, 13) (E, 2) (M, 93)
C: (N, 72) (B, 69) (G, 31) (I, 6)
H: (B, 40) (E, 9)
Minimum Spanning Tree Edges:
A - G: 43
D - L: 9
E - K: 8
G - E: 9
E - O: 12
C - I: 6
E - J: 10
F - B: 17
A - D: 25
B - N: 26
N - M: 75
E - F: 2
G - C: 31
E - H: 9
Total Cost: 282
```

3.

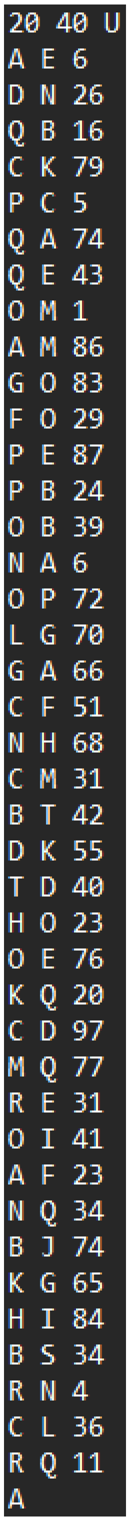


```
20 30 U
E T 40
J M 84
A M 88
B K 61
O S 47
E L 50
K Q 71
D R 49
I F 71
L H 10
K G 30
P T 84
G B 96
F E 45
S G 75
D F 11
S Q 66
A B 92
S I 97
P A 89
N S 28
E A 31
Q D 38
F N 67
G E 32
C L 44
M K 19
B E 13
I H 75
H D 61
A
```

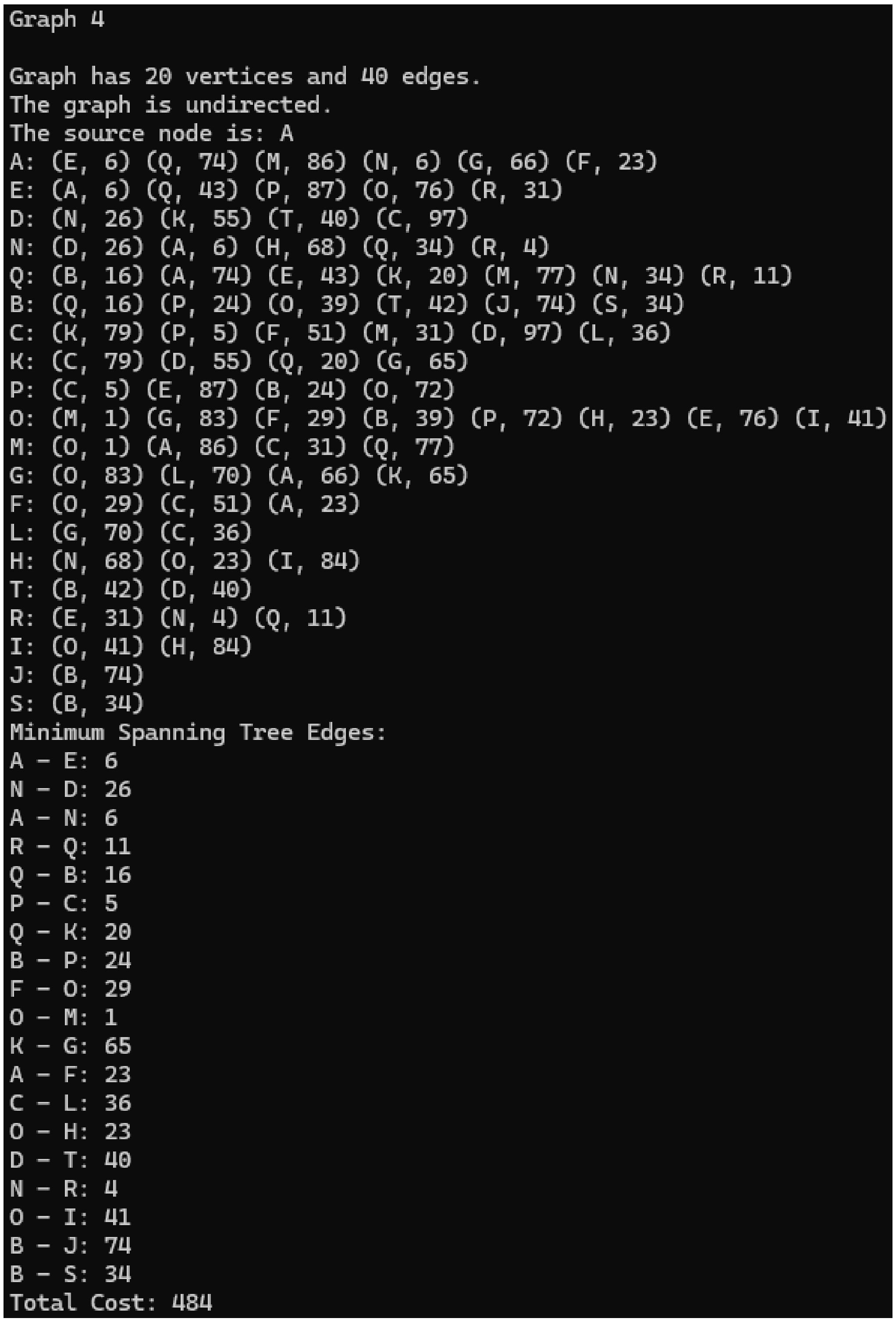


```
Graph 3

Graph has 20 vertices and 30 edges.
The graph is undirected.
The source node is: A
E: (T, 40) (L, 50) (F, 45) (A, 31) (G, 32) (B, 13)
T: (E, 40) (P, 84)
J: (M, 84)
M: (J, 84) (A, 88) (K, 19)
A: (M, 88) (B, 92) (P, 89) (E, 31)
B: (K, 61) (G, 96) (A, 92) (E, 13)
K: (B, 61) (Q, 71) (G, 30) (M, 19)
O: (S, 47)
S: (O, 47) (G, 75) (Q, 66) (I, 97) (N, 28)
L: (E, 50) (H, 10) (C, 44)
Q: (K, 71) (S, 66) (D, 38)
D: (R, 49) (F, 11) (Q, 38) (H, 61)
R: (D, 49)
I: (F, 71) (S, 97) (H, 75)
F: (I, 71) (E, 45) (D, 11) (N, 67)
H: (L, 10) (I, 75) (D, 61)
G: (K, 30) (B, 96) (S, 75) (E, 32)
P: (T, 84) (A, 89)
N: (S, 28) (F, 67)
C: (L, 44)
Minimum Spanning Tree Edges:
A - E: 31
E - T: 40
M - J: 84
K - M: 19
E - B: 13
G - K: 30
S - O: 47
Q - S: 66
E - L: 50
D - Q: 38
F - D: 11
D - R: 49
F - I: 71
E - F: 45
L - H: 10
E - G: 32
T - P: 84
S - N: 28
L - C: 44
Total Cost: 792
```

4.



```
20 40 U
A E 6
D N 26
Q B 16
C K 79
P C 5
Q A 74
Q E 43
O M 1
A M 86
G O 83
F O 29
P E 87
P B 24
O B 39
N A 6
O P 72
L G 70
G A 66
C F 51
N H 68
C M 31
B T 42
D K 55
T D 40
H O 23
O E 76
K Q 20
C D 97
M Q 77
R E 31
O I 41
A F 23
N Q 34
B J 74
K G 65
H I 84
B S 34
R N 4
C L 36
R Q 11
A
```



```
Graph 4

Graph has 20 vertices and 40 edges.
The graph is undirected.
The source node is: A
A: (E, 6) (Q, 74) (M, 86) (N, 6) (G, 66) (F, 23)
E: (A, 6) (Q, 43) (P, 87) (O, 76) (R, 31)
D: (N, 26) (K, 55) (T, 40) (C, 97)
N: (D, 26) (A, 6) (H, 68) (Q, 34) (R, 4)
Q: (B, 16) (A, 74) (E, 43) (K, 20) (M, 77) (N, 34) (R, 11)
B: (Q, 16) (P, 24) (O, 39) (T, 42) (J, 74) (S, 34)
C: (K, 79) (P, 5) (F, 51) (M, 31) (D, 97) (L, 36)
K: (C, 79) (D, 55) (Q, 20) (G, 65)
P: (C, 5) (E, 87) (B, 24) (O, 72)
O: (M, 1) (G, 83) (F, 29) (B, 39) (P, 72) (H, 23) (E, 76) (I, 41)
M: (O, 1) (A, 86) (C, 31) (Q, 77)
G: (O, 83) (L, 70) (A, 66) (K, 65)
F: (O, 29) (C, 51) (A, 23)
L: (G, 70) (C, 36)
H: (N, 68) (O, 23) (I, 84)
T: (B, 42) (D, 40)
R: (E, 31) (N, 4) (Q, 11)
I: (O, 41) (H, 84)
J: (B, 74)
S: (B, 34)
Minimum Spanning Tree Edges:
A - E: 6
N - D: 26
A - N: 6
R - Q: 11
Q - B: 16
P - C: 5
Q - K: 20
B - P: 24
F - O: 29
O - M: 1
K - G: 65
A - F: 23
C - L: 36
O - H: 23
D - T: 40
N - R: 4
O - I: 41
B - J: 74
B - S: 34
Total Cost: 484
```