

Code:

```
J prac4N.java > ᐃ prac4N
 1 import java.util.ArrayList;
 2 import java.util.Comparator;
 3 import java.util.List;
 4
 5 public class prac4N {
 6
 7     static class Edge {
 8         int src, dest, weight;
 9
10         public Edge(int src, int dest, int weight)
11         {
12             this.src = src;
13             this.dest = dest;
14             this.weight = weight;
15         }
16     }
17
18     static class Subset {
19         int parent, rank;
20
21         public Subset(int parent, int rank)
22         {
23             this.parent = parent;
24             this.rank = rank;
25         }
26     }
27
28     public static void main(String[] args)
29     {
30         int V = 4;
31         List<Edge> graphEdges = new ArrayList<Edge>(
32             List.of(new Edge(src:0, dest:1, weight:10), new Edge(src:0, dest:2, weight:6),
33                   new Edge(src:0, dest:3, weight:5), new Edge(src:1, dest:3, weight:15),
34                   new Edge(src:2, dest:3, weight:4)));
35
36         graphEdges.sort(new Comparator<Edge>() {
37             @Override public int compare(Edge o1, Edge o2)
38             {
39                 return o1.weight - o2.weight;
40             }
41         });
42
43         kruskals(V, graphEdges);
44     }
45
46     private static void kruskals(int V, List<Edge> edges)
47     {
48         int j = 0;
49         int noOfEdges = 0;
50
51         Subset subsets[] = new Subset[V];
52
53         Edge results[] = new Edge[V];
54
55         for (int i = 0; i < V; i++) {
56             subsets[i] = new Subset(i, rank:0);
57         }
58
59         while (noOfEdges < V - 1) {
60             Edge nextEdge = edges.get(j);
```

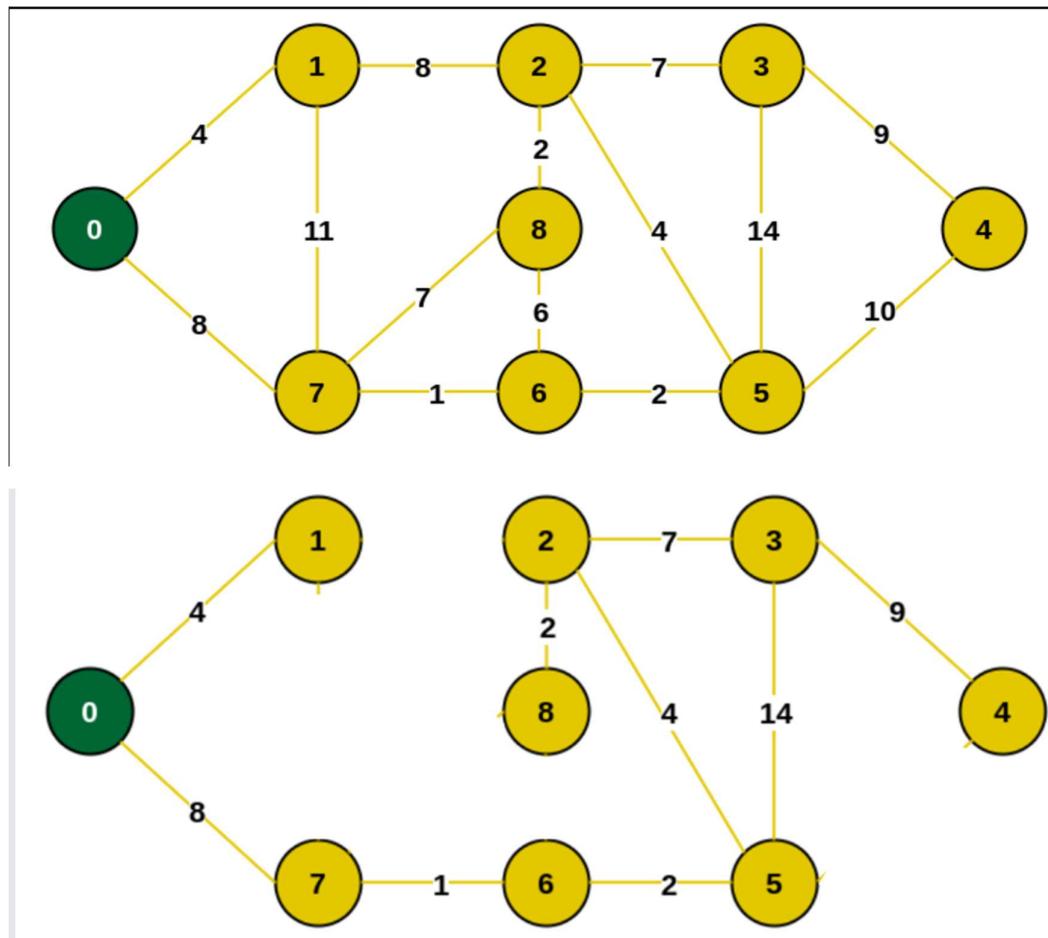
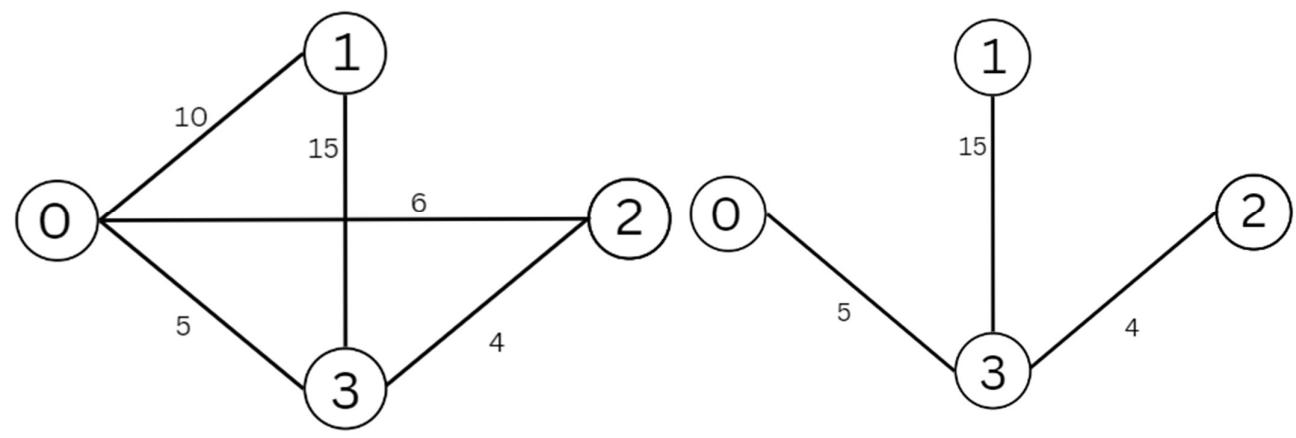
```

61     int x = findRoot(subsets, nextEdge.src);
62     int y = findRoot(subsets, nextEdge.dest);
63
64     if (x != y) {
65         results[noOfEdges] = nextEdge;
66         union(subsets, x, y);
67         noOfEdges++;
68     }
69
70     j++;
71 }
72
73 System.out.println(
74     x:"Following are the edges of the constructed MST:");
75 int minCost = 0;
76 for (int i = 0; i < noOfEdges; i++) {
77     System.out.println(results[i].src + " -- "
78     | | | | + results[i].dest + " == "
79     | | | | + results[i].weight);
80     minCost += results[i].weight;
81 }
82 System.out.println("Total cost of MST: " + minCost);
83 }
84
85 private static void union(Subset[] subsets, int x,
86 | | | | | | | int y)
87 {
88     int rootX = findRoot(subsets, x);
89     int rootY = findRoot(subsets, y);
90
91     if (subsets[rootY].rank < subsets[rootX].rank) {
92         subsets[rootY].parent = rootX;
93     }
94     else if (subsets[rootX].rank
95     | | | | < subsets[rootY].rank) {
96         subsets[rootX].parent = rootY;
97     }
98     else {
99         subsets[rootY].parent = rootX;
100        subsets[rootX].rank++;
101    }
102 }
103
104 private static int findRoot(Subset[] subsets, int i)
105 {
106     if (subsets[i].parent == i)
107         return subsets[i].parent;
108
109     subsets[i].parent
110     | = findRoot(subsets, subsets[i].parent);
111     return subsets[i].parent;
112 }
113 }
```

Output:

Following are the edges of the constructed MST:
 2 -- 3 == 4
 0 -- 3 == 5
 0 -- 1 == 10
 Total cost of MST: 19

N



Code:

```
J prac4.java > 🛡️ prac4 > ⚙️ kruskalMST()
1  import java.util.*;
2  class Edge implements Comparable<Edge> {
3      int src, dest, weight;
4      public int compareTo(Edge otherEdge) {
5          return this.weight - otherEdge.weight;
6      }
7  };
8  class prac4 {
9      int V, E;
10     Edge[] edges;
11     prac4(int v, int e) {
12         V = v;
13         E = e;
14         edges = new Edge[E];
15         for (int i = 0; i < e; ++i) {
16             edges[i] = new Edge();
17         }
18     }
19     int find(int[] parent, int i) {
20         if (parent[i] == -1) {
21             return i;
22         }
23         return find(parent, parent[i]);
24     }
25     void union(int[] parent, int x, int y) {
26         int xset = find(parent, x);
27         int yset = find(parent, y);
28         parent[xset] = yset;
29     }
30     void kruskalMST() {
31         Edge[] result = new Edge[V - 1];
32         int[] parent = new int[V];
33         Arrays.fill(parent, -1);
34         Arrays.sort(edges);
35         int e = 0;
36         int i = 0;
37         while (e < V - 1 && i < E) {
38             Edge nextEdge = edges[i++];
39             int x = find(parent, nextEdge.src);
40             int y = find(parent, nextEdge.dest);
41             if (x != y) {
42                 result[e++] = nextEdge;
43                 union(parent, x, y);
44             }
45         }
46         System.out.println("Edges in Minimum Spanning Tree:");
47         int minimumCost = 0;
48         for (i = 0; i < e; ++i) {
49             System.out.println(result[i].src + " - " + result[i].dest + " : " + result[i].weight);
50             minimumCost += result[i].weight;
51         }
52         System.out.println("Minimum Cost: " + minimumCost);
53     }
}
Run | Debug
54  public static void main(String[] args) {
55      int V = 9;
56      int E = 14;
57      prac4 graph = new prac4(V, E);
58
59      graph.edges[0].src = 0;
60      graph.edges[0].dest = 1;
61      graph.edges[0].weight = 4;
```

```

62     graph.edges[1].src = 0;
63     graph.edges[1].dest = 7;
64     graph.edges[1].weight = 8;
65
66     graph.edges[2].src = 1;
67     graph.edges[2].dest = 7;
68     graph.edges[2].weight = 11;
69
70     graph.edges[3].src = 1;
71     graph.edges[3].dest = 2;
72     graph.edges[3].weight = 8;
73
74     graph.edges[4].src = 7;
75     graph.edges[4].dest = 8;
76     graph.edges[4].weight = 7;
77
78     graph.edges[5].src = 7;
79     graph.edges[5].dest = 6;
80     graph.edges[5].weight = 1;
81
82     graph.edges[6].src = 6;
83     graph.edges[6].dest = 8;
84     graph.edges[6].weight = 6;
85
86     graph.edges[7].src = 6;
87     graph.edges[7].dest = 5;
88     graph.edges[7].weight = 2;
89
90     graph.edges[8].src = 8;
91     graph.edges[8].dest = 2;
92     graph.edges[8].weight = 2;
93
94     graph.edges[9].src = 2;
95     graph.edges[9].dest = 3;
96     graph.edges[9].weight = 7;
97
98     graph.edges[10].src = 2;
99     graph.edges[10].dest = 5;
100    graph.edges[10].weight = 4;
101
102    graph.edges[11].src = 3;
103    graph.edges[11].dest = 5;
104    graph.edges[11].weight = 14;
105
106    graph.edges[12].src = 3;
107    graph.edges[12].dest = 4;
108    graph.edges[12].weight = 9;
109
110    graph.edges[13].src = 4;
111    graph.edges[13].dest = 5;
112    graph.edges[13].weight = 10;
113
114    graph.kruskalMST();
115
116 }
117 }
118 }
```

Output

```

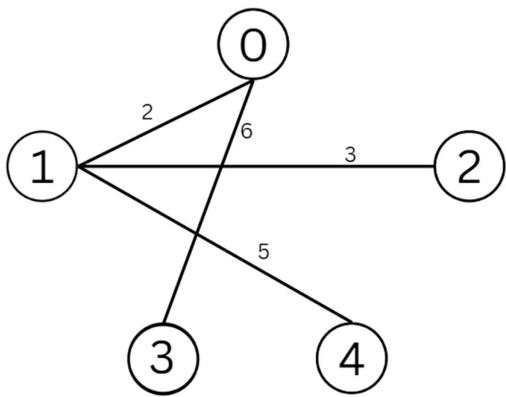
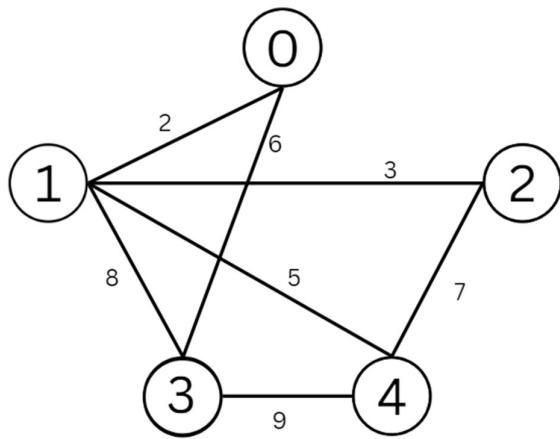
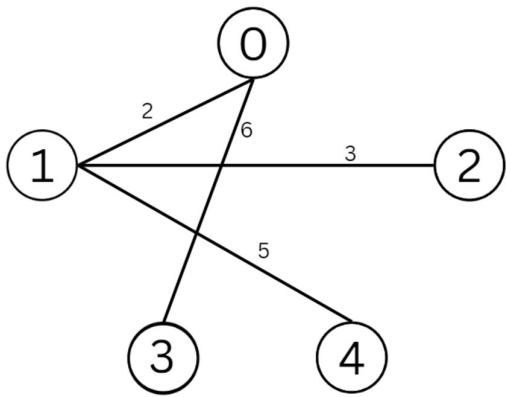
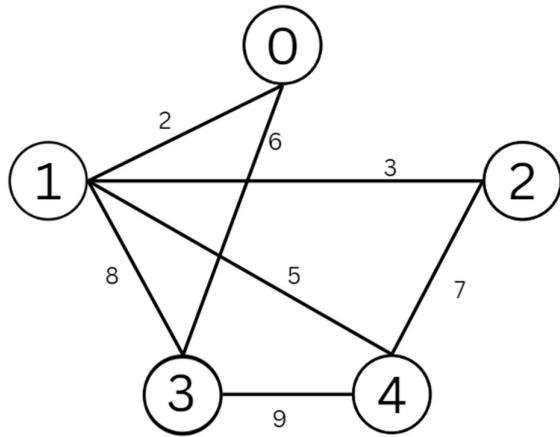
Edges in Minimum Spanning Tree:
7 - 6 : 1
6 - 5 : 2
8 - 2 : 2
0 - 1 : 4
2 - 5 : 4
2 - 3 : 7
0 - 7 : 8
3 - 4 : 9
Minimum Cost: 37
```

Code:-

```
J prac5.java > pracs5 > main(String[])
  1  import java.util.Arrays;
●  2
  3 class pracs5 {
  4     private static final int V = 5; // Number of vertices
  5
  6     int minKey(int key[], boolean mstSet[]) {
  7         int min = Integer.MAX_VALUE;
  8         int minIndex = -1;
  9         for (int v = 0; v < V; v++) {
10             if (!mstSet[v] && key[v] < min) {
11                 min = key[v];
12                 minIndex = v;
13             }
14         }
15         return minIndex;
16     }
17     void printMST(int parent[], int graph[][])
18     System.out.println("Edge \tWeight");
19     for (int i = 1; i < V; i++) {
20         System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
21     }
22 }
23 void primMST(int graph[][])
24     int parent[] = new int[V]; // Array to store constructed MST
25     int key[] = new int[V]; // Key values used to pick the minimum weight edge
26     boolean mstSet[] = new boolean[V]; // To represent set of vertices included in MST
27
28     Arrays.fill(key, Integer.MAX_VALUE);
29     Arrays.fill(mstSet, val:false);
30
31     // Start from the first vertex
32     key[0] = 0;
33     parent[0] = -1;
34
35     for (int count = 0; count < V - 1; count++) {
36         int u = minKey(key, mstSet);
37         mstSet[u] = true;
38         for (int v = 0; v < V; v++) {
39             if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
40                 parent[v] = u;
41                 key[v] = graph[u][v];
42             }
43         }
44     }
45     printMST(parent, graph);
46 }
Run | Debug
47 public static void main(String[] args) {
48     pracs5 mst = new pracs5();
49
50     int graph[][] = new int[][]{
51         {0, 2, 0, 6, 0},
52         {2, 0, 3, 8, 5},
53         {0, 3, 0, 0, 7},
54         {6, 8, 0, 0, 9},
55         {0, 5, 7, 9, 0}
56     };
57
58     mst.primMST(graph);
59 }
60 }
```

Output:

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```



Code:-

```
J prac5.java > pracs5 > main(String[])
  1  import java.util.Arrays;
●  2
  3 class prac5 {
  4     private static final int V = 5; // Number of vertices
  5
  6     int minKey(int key[], boolean mstSet[]) {
  7         int min = Integer.MAX_VALUE;
  8         int minIndex = -1;
  9         for (int v = 0; v < V; v++) {
10             if (!mstSet[v] && key[v] < min) {
11                 min = key[v];
12                 minIndex = v;
13             }
14         }
15         return minIndex;
16     }
17     void printMST(int parent[], int graph[][])
18     System.out.println("Edge \tWeight");
19     for (int i = 1; i < V; i++) {
20         System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
21     }
22 }
23 void primMST(int graph[][])
24     int parent[] = new int[V]; // Array to store constructed MST
25     int key[] = new int[V]; // Key values used to pick the minimum weight edge
26     boolean mstSet[] = new boolean[V]; // To represent set of vertices included in MST
27
28     Arrays.fill(key, Integer.MAX_VALUE);
29     Arrays.fill(mstSet, val:false);
30
31     // Start from the first vertex
32     key[0] = 0;
33     parent[0] = -1;
34
35     for (int count = 0; count < V - 1; count++) {
36         int u = minKey(key, mstSet);
37         mstSet[u] = true;
38         for (int v = 0; v < V; v++) {
39             if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
40                 parent[v] = u;
41                 key[v] = graph[u][v];
42             }
43         }
44     }
45     printMST(parent, graph);
46 }
Run | Debug
47 public static void main(String[] args) {
48     prac5 mst = new prac5();
49
50     int graph[][] = new int[][]{
51         {0, 2, 0, 6, 0},
52         {2, 0, 3, 8, 5},
53         {0, 3, 0, 0, 7},
54         {6, 8, 0, 0, 9},
55         {0, 5, 7, 9, 0}
56     };
57
58     mst.primMST(graph);
59 }
60 }
```

Output:

```
Edge    Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

Code:-

```
J prac6.java > prac6
 1  import java.util.PriorityQueue;
 2  import java.util.HashMap;
 3  import java.util.Map;
 4
 5  class HuffmanNode implements Comparable<HuffmanNode> {
 6      char data;
 7      int frequency;
 8      HuffmanNode left, right;
 9
10     public HuffmanNode(char data, int frequency) {
11         this.data = data;
12         this.frequency = frequency;
13         left = right = null;
14     }
15
16     public int compareTo(HuffmanNode node) {
17         return this.frequency - node.frequency;
18     }
19 }
20
21 public class prac6 {
22     public static void printHuffmanCodes(HuffmanNode root, String code) {
23         if (root == null) return;
24
25         if (root.left == null && root.right == null) {
26             System.out.println(root.data + ":" + code);
27         }
28
29         printHuffmanCodes(root.left, code + "0");
30         printHuffmanCodes(root.right, code + "1");
31     }
32
33     public static void buildHuffmanTree(String text) {
34         if (text == null || text.length() == 0) return;
35
36         Map<Character, Integer> frequencyMap = new HashMap<>();
37         for (char c : text.toCharArray()) {
38             frequencyMap.put(c, frequencyMap.getOrDefault(c, defaultValue:0) + 1);
39         }
40
41         PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();
42         for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {
43             minHeap.offer(new HuffmanNode(entry.getKey(), entry.getValue()));
44         }
45
46         while (minHeap.size() > 1) {
47             HuffmanNode left = minHeap.poll();
48             HuffmanNode right = minHeap.poll();
49
50             HuffmanNode newNode = new HuffmanNode(data: '\0', left.frequency + right.frequency);
51             newNode.left = left;
52             newNode.right = right;
53
54             minHeap.offer(newNode);
55         }
56
57         HuffmanNode root = minHeap.peek();
58         printHuffmanCodes(root, code:"");
59     }
60
61     public static void main(String[] args) {
62         String text = "My name is akanksh bodakhe";
63         buildHuffmanTree(text);
64     }
65 }
```

Output:-

```
Input data (in decimal)
h: 000
s: 001
y: 0100
i: 01010
o: 01011
k: 011
a: 100
n: 1010
M: 10110
m: 10111
: 110
e: 1110
b: 11110
d: 11111
```

Code:-

```
J prac6.java > prac6 > main(String[])
1 import java.util.PriorityQueue;
2 import java.util.HashMap;
3 import java.util.Map;
4
5 class HuffmanNode implements Comparable<HuffmanNode> {
6     char data;
7     int frequency;
8     HuffmanNode left, right;
9
10    public HuffmanNode(char data, int frequency) {
11        this.data = data;
12        this.frequency = frequency;
13        left = right = null;
14    }
15
16    public int compareTo(HuffmanNode node) {
17        return this.frequency - node.frequency;
18    }
19}
20
21 public class prac6 {
22     public static void printHuffmanCodes(HuffmanNode root, String code) {
23         if (root == null) return;
24
25         if (root.left == null && root.right == null) {
26             System.out.println(root.data + ": " + code);
27         }
28
29         printHuffmanCodes(root.left, code + "0");
30         printHuffmanCodes(root.right, code + "1");
31     }
32
33     public static void buildHuffmanTree(String text) {
34         if (text == null || text.length() == 0) return;
35
36         Map<Character, Integer> frequencyMap = new HashMap<>();
37         for (char c : text.toCharArray()) {
38             frequencyMap.put(c, frequencyMap.getOrDefault(c, defaultValue:0) + 1);
39         }
40
41         PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();
42         for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {
43             minHeap.offer(new HuffmanNode(entry.getKey(), entry.getValue()));
44         }
45
46         while (minHeap.size() > 1) {
47             HuffmanNode left = minHeap.poll();
48             HuffmanNode right = minHeap.poll();
49
50             HuffmanNode newNode = new HuffmanNode(data:'\0', left.frequency + right.frequency);
51             newNode.left = left;
52             newNode.right = right;
53
54             minHeap.offer(newNode);
55         }
56
57         HuffmanNode root = minHeap.peek();
58         printHuffmanCodes(root, code:@"");
59     }
60
61     public static void main(String[] args) {
62         String text = "Hi! I am Nisarga Telang";
63         buildHuffmanTree(text);
64     }
65 }
66
```

Output:-

```
r: 0000
l: 0001
!: 0010
T: 0011
m: 0100
N: 0101
i: 011
s: 1000
H: 10010
n: 10011
a: 101
g: 1100
e: 11010
I: 11011
: 111
```

Code:-

```
J prac7.java > 🏷 prac7 > ⚙ main(String[])
1  import java.util.*;
2
3  class Graph {
4      private int V; // Number of vertices
5      private List<List<Node>> adj;
6
7      public Graph(int V) {
8          this.V = V;
9          adj = new ArrayList<>(V);
10         for (int i = 0; i < V; i++) {
11             adj.add(new ArrayList<>());
12         }
13     }
14
15     public void addEdge(int source, int destination, int weight) {
16         Node node = new Node(destination, weight);
17         adj.get(source).add(node); // Add edge to the adjacency list
18     }
19
20     public void dijkstra(int source) {
21         PriorityQueue<Node> minHeap = new PriorityQueue<>(V, Comparator.comparingInt(node -> node.weight));
22         int[] dist = new int[V];
23         Arrays.fill(dist, Integer.MAX_VALUE);
24
25         // Initialize the source vertex
26         dist[source] = 0;
27         minHeap.add(new Node(source, weight:0));
28
29         while (!minHeap.isEmpty()) {
30             int u = minHeap.poll().vertex;
31
32             for (Node neighbor : adj.get(u)) {
33                 int v = neighbor.vertex;
34                 int weight = neighbor.weight;
35
36                 // Relaxation step
37                 if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
38                     dist[v] = dist[u] + weight;
39                     minHeap.add(new Node(v, dist[v]));
40                 }
41             }
42         }
43
44         // Print the shortest distances
45         System.out.println("Shortest distances from source vertex " + source + ":");
46         for (int i = 0; i < V; i++) {
47             System.out.println("Vertex " + i + ": " + dist[i]);
48         }
49     }
50
51     private static class Node {
52         int vertex;
53         int weight;
54
55         Node(int vertex, int weight) {
56             this.vertex = vertex;
57             this.weight = weight;
58         }
59     }
60 }
61
```

```
62 ✓ public class prac7 {  
63   Run | Debug  
64     public static void main(String[] args) {  
65       int V = 6; // Number of vertices  
66       Graph graph = new Graph(V);  
67  
68       // Add edges with weights  
69       graph.addEdge(source:0, destination:1, weight:2);  
70       graph.addEdge(source:0, destination:2, weight:4);  
71       graph.addEdge(source:1, destination:2, weight:1);  
72       graph.addEdge(source:1, destination:3, weight:7);  
73       graph.addEdge(source:2, destination:4, weight:3);  
74       graph.addEdge(source:3, destination:4, weight:2);  
75       graph.addEdge(source:3, destination:5, weight:1);  
76       graph.addEdge(source:4, destination:5, weight:5);  
77  
78       int sourceVertex = 0; // Source vertex for Dijkstra's algorithm  
79  
80       graph.dijkstra(sourceVertex);  
81     }  
82
```

Output:-

```
Shortest distances from source vertex 0:  
Vertex 0: 0  
Vertex 1: 2  
Vertex 2: 3  
Vertex 3: 9  
Vertex 4: 6  
Vertex 5: 10
```

Code:-

```
J prac7.java > 🏷 prac7 > ⚙ main(String[])
1  import java.util.*;
2
3  class Graph {
4      private int V; // Number of vertices
5      private List<List<Node>> adj;
6
7      public Graph(int V) {
8          this.V = V;
9          adj = new ArrayList<>(V);
10         for (int i = 0; i < V; i++) {
11             adj.add(new ArrayList<>());
12         }
13     }
14
15     public void addEdge(int source, int destination, int weight) {
16         Node node = new Node(destination, weight);
17         adj.get(source).add(node); // Add edge to the adjacency list
18     }
19
20     public void dijkstra(int source) {
21         PriorityQueue<Node> minHeap = new PriorityQueue<>(V, Comparator.comparingInt(node -> node.weight));
22         int[] dist = new int[V];
23         Arrays.fill(dist, Integer.MAX_VALUE);
24
25         // Initialize the source vertex
26         dist[source] = 0;
27         minHeap.add(new Node(source, weight:0));
28
29         while (!minHeap.isEmpty()) {
30             int u = minHeap.poll().vertex;
31
32             for (Node neighbor : adj.get(u)) {
33                 int v = neighbor.vertex;
34                 int weight = neighbor.weight;
35
36                 // Relaxation step
37                 if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
38                     dist[v] = dist[u] + weight;
39                     minHeap.add(new Node(v, dist[v]));
40                 }
41             }
42         }
43
44         // Print the shortest distances
45         System.out.println("Shortest distances from source vertex " + source + ":");
46         for (int i = 0; i < V; i++) {
47             System.out.println("Vertex " + i + ": " + dist[i]);
48         }
49     }
50
51     private static class Node {
52         int vertex;
53         int weight;
54
55         Node(int vertex, int weight) {
56             this.vertex = vertex;
57             this.weight = weight;
58         }
59     }
60 }
61
```

```
62 ✓ public class prac7 {  
63   Run | Debug  
64     public static void main(String[] args) {  
65       int V = 6; // Number of vertices  
66       Graph graph = new Graph(V);  
67  
68       // Add edges with weights  
69       graph.addEdge(source:0, destination:1, weight:2);  
70       graph.addEdge(source:0, destination:2, weight:4);  
71       graph.addEdge(source:1, destination:2, weight:1);  
72       graph.addEdge(source:1, destination:3, weight:7);  
73       graph.addEdge(source:2, destination:4, weight:3);  
74       graph.addEdge(source:3, destination:4, weight:2);  
75       graph.addEdge(source:3, destination:5, weight:1);  
76       graph.addEdge(source:4, destination:5, weight:5);  
77  
78       int sourceVertex = 0; // Source vertex for Dijkstra's algorithm  
79  
80       graph.dijkstra(sourceVertex);  
81     }  
82
```

Output:-

```
Shortest distances from source vertex 0:  
Vertex 0: 0  
Vertex 1: 2  
Vertex 2: 3  
Vertex 3: 9  
Vertex 4: 6  
Vertex 5: 10
```

Code:-

```
J prac8.java > 🏃 prac8 > ⚙ findLCS(String, String)
1  public class prac8 {
2      public static String findLCS(String str1, String str2) {
3          int m = str1.length();
4          int n = str2.length();
5
6          // Create a 2D array to store the LCS lengths
7          int[][] dp = new int[m + 1][n + 1];
8
9          // Build the LCS matrix using dynamic programming
10         for (int i = 1; i <= m; i++) {
11             for (int j = 1; j <= n; j++) {
12                 if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
13                     dp[i][j] = dp[i - 1][j - 1] + 1;
14                 } else {
15                     dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
16                 }
17             }
18         }
19
20         // Reconstruct the LCS from the dp matrix
21         int i = m;
22         int j = n;
23         StringBuilder lcs = new StringBuilder();
24         while (i > 0 && j > 0) {
25             if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
26                 lcs.insert(offset:0, str1.charAt(i - 1));
27                 i--;
28                 j--;
29             } else if (dp[i - 1][j] > dp[i][j - 1]) {
30                 i--;
31             } else {
32                 j--;
33             }
34         }
35
36         return lcs.toString();
37     }
38
39     Run | Debug
40     public static void main(String[] args) {
41         String str1 = "Akanksh";
42         String str2 = "Bodakhe";
43
44         String lcs = findLCS(str1, str2);
45         System.out.println("Longest Common Subsequence: " + lcs);
46     }
47
48 }
```

Output:

Longest Common Subsequence: akh

Code:-

```
J prac8.java > pr8 prac8 > main(String[])
1  public class prac8 {
2      public static String findLCS(String str1, String str2) {
3          int m = str1.length();
4          int n = str2.length();
5
6          // Create a 2D array to store the LCS lengths
7          int[][] dp = new int[m + 1][n + 1];
8
9          // Build the LCS matrix using dynamic programming
10         for (int i = 1; i <= m; i++) {
11             for (int j = 1; j <= n; j++) {
12                 if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
13                     dp[i][j] = dp[i - 1][j - 1] + 1;
14                 } else {
15                     dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
16                 }
17             }
18         }
19
20         // Reconstruct the LCS from the dp matrix
21         int i = m;
22         int j = n;
23         StringBuilder lcs = new StringBuilder();
24         while (i > 0 && j > 0) {
25             if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
26                 lcs.insert(offset:0, str1.charAt(i - 1));
27                 i--;
28                 j--;
29             } else if (dp[i - 1][j] > dp[i][j - 1]) {
30                 i--;
31             } else {
32                 j--;
33             }
34         }
35
36         return lcs.toString();
37     }
38
39     Run | Debug
40     public static void main(String[] args) {
41         String str1 = "Nisarga";
42         String str2 = "Telang";
43
44         String lcs = findLCS(str1, str2);
45         System.out.println("Longest Common Subsequence: " + lcs);
46     }
47
48 }
```

Output:-

```
Longest Common Subsequence: ag
```

Code:-

```
J prat9.java > ᐃ prat9
1
2  class prat9 {
3
4      class Edge {
5          int src, dest, weight;
6          Edge() { src = dest = weight = 0; }
7      };
8
9      int V, E;
10     Edge edge[];
11
12     prat9(int v, int e)
13     {
14         V = v;
15         E = e;
16         edge = new Edge[e];
17         for (int i = 0; i < e; ++i)
18             edge[i] = new Edge();
19     }
20
21     void BellmanFord(prat9 graph, int src)
22     {
23         int V = graph.V, E = graph.E;
24         int dist[] = new int[V];
25
26         for (int i = 0; i < V; ++i)
27             dist[i] = Integer.MAX_VALUE;
28         dist[src] = 0;
29
30         // have at-most |V| - 1 edges
31         for (int i = 1; i < V; ++i) {
32             for (int j = 0; j < E; ++j) {
33                 int u = graph.edge[j].src;
34                 int v = graph.edge[j].dest;
35                 int weight = graph.edge[j].weight;
36                 if (dist[u] != Integer.MAX_VALUE
37                     && dist[u] + weight < dist[v])
38                     dist[v] = dist[u] + weight;
39             }
40         }
41
42         for (int j = 0; j < E; ++j) {
43             int u = graph.edge[j].src;
44             int v = graph.edge[j].dest;
45             int weight = graph.edge[j].weight;
46             if (dist[u] != Integer.MAX_VALUE
47                 && dist[u] + weight < dist[v]) {
48                 System.out.println(
49                     x:"Graph contains negative weight cycle");
50                 return;
51             }
52         }
53         printArr(dist, V);
54     }
55
56     void printArr(int dist[], int V)
57     {
58         System.out.println(x:"Vertex Distance from Source");
59         for (int i = 0; i < V; ++i)
60             System.out.println(i + "\t\t" + dist[i]);
61     }
}
```

```

62
63 // Driver's code
Run | Debug
64 public static void main(String[] args)
65 {
66     int V = 5; // Number of vertices in graph
67     int E = 8; // Number of edges in graph
68
69     prac9 graph = new prac9(V, E);
70
71     graph.edge[0].src = 0;
72     graph.edge[0].dest = 1;
73     graph.edge[0].weight = -1;
74
75     graph.edge[1].src = 0;
76     graph.edge[1].dest = 2;
77     graph.edge[1].weight = 4;
78
79     graph.edge[2].src = 1;
80     graph.edge[2].dest = 2;
81     graph.edge[2].weight = 3;
82
83     graph.edge[3].src = 1;
84     graph.edge[3].dest = 3;
85     graph.edge[3].weight = 2;
86
87     graph.edge[4].src = 1;
88     graph.edge[4].dest = 4;
89     graph.edge[4].weight = 2;
90
91     graph.edge[5].src = 3;
92     graph.edge[5].dest = 2;
93     graph.edge[5].weight = 5;
94
95     graph.edge[6].src = 3;
96     graph.edge[6].dest = 1;
97     graph.edge[6].weight = 1;
98
99     graph.edge[7].src = 4;
100    graph.edge[7].dest = 3;
101    graph.edge[7].weight = -3;
102
103    graph.BellmanFord(graph, src:0);
104 }

```

Output:-

```

Vertex Distance from Source
0          0
1          -1
2          2
3          -2
4          1   -

```

Code:-

```
J prat9.java > ᐃ prat9
1
2  class prat9 {
3
4      class Edge {
5          int src, dest, weight;
6          Edge() { src = dest = weight = 0; }
7      };
8
9      int V, E;
10     Edge edge[];
11
12     prat9(int v, int e)
13     {
14         V = v;
15         E = e;
16         edge = new Edge[e];
17         for (int i = 0; i < e; ++i)
18             edge[i] = new Edge();
19     }
20
21     void BellmanFord(prat9 graph, int src)
22     {
23         int V = graph.V, E = graph.E;
24         int dist[] = new int[V];
25
26         for (int i = 0; i < V; ++i)
27             dist[i] = Integer.MAX_VALUE;
28         dist[src] = 0;
29
30         // have at-most |V| - 1 edges
31         for (int i = 1; i < V; ++i) {
32             for (int j = 0; j < E; ++j) {
33                 int u = graph.edge[j].src;
34                 int v = graph.edge[j].dest;
35                 int weight = graph.edge[j].weight;
36                 if (dist[u] != Integer.MAX_VALUE
37                     && dist[u] + weight < dist[v])
38                     dist[v] = dist[u] + weight;
39             }
40         }
41
42         for (int j = 0; j < E; ++j) {
43             int u = graph.edge[j].src;
44             int v = graph.edge[j].dest;
45             int weight = graph.edge[j].weight;
46             if (dist[u] != Integer.MAX_VALUE
47                 && dist[u] + weight < dist[v]) {
48                 System.out.println(
49                     x:"Graph contains negative weight cycle");
50                 return;
51             }
52         }
53         printArr(dist, V);
54     }
55
56     void printArr(int dist[], int V)
57     {
58         System.out.println(x:"Vertex Distance from Source");
59         for (int i = 0; i < V; ++i)
60             System.out.println(i + "\t\t" + dist[i]);
61     }
}
```

```

62
63 // Driver's code
Run | Debug
64 public static void main(String[] args)
65 {
66     int V = 5; // Number of vertices in graph
67     int E = 8; // Number of edges in graph
68
69     prac9 graph = new prac9(V, E);
70
71     graph.edge[0].src = 0;
72     graph.edge[0].dest = 1;
73     graph.edge[0].weight = -1;
74
75     graph.edge[1].src = 0;
76     graph.edge[1].dest = 2;
77     graph.edge[1].weight = 4;
78
79     graph.edge[2].src = 1;
80     graph.edge[2].dest = 2;
81     graph.edge[2].weight = 3;
82
83     graph.edge[3].src = 1;
84     graph.edge[3].dest = 3;
85     graph.edge[3].weight = 2;
86
87     graph.edge[4].src = 1;
88     graph.edge[4].dest = 4;
89     graph.edge[4].weight = 2;
90
91     graph.edge[5].src = 3;
92     graph.edge[5].dest = 2;
93     graph.edge[5].weight = 5;
94
95     graph.edge[6].src = 3;
96     graph.edge[6].dest = 1;
97     graph.edge[6].weight = 1;
98
99     graph.edge[7].src = 4;
100    graph.edge[7].dest = 3;
101    graph.edge[7].weight = -3;
102
103    graph.BellmanFord(graph, src:0);
104 }

```

Output:-

```

Vertex Distance from Source
0          0
1          -1
2          2
3          -2
4          1   -

```

Code:-

```
J prac10.java > floydWarshall(int[][])
1  class prac10 {
2      final static int INF = 99999, V = 4;
3      void floydWarshall(int dist[][])
4      {
5          int i, j, k;
6          for (k = 0; k < V; k++) {
7              // Pick all vertices as source one by one
8              for (i = 0; i < V; i++) {
9                  // Pick all vertices as destination for the
10                 // above picked source
11                 for (j = 0; j < V; j++) {
12                     if (dist[i][k] + dist[k][j]
13                         < dist[i][j])
14                         dist[i][j]
15                         = dist[i][k] + dist[k][j];
16                 }
17             }
18         }
19
20         // Print the shortest distance matrix
21         printSolution(dist);
22     }
23
24     void printSolution(int dist[][])
25     {
26         System.out.println(
27             "The following matrix shows the shortest "
28             + "distances between every pair of vertices");
29         for (int i = 0; i < V; ++i) {
30             for (int j = 0; j < V; ++j) {
31                 if (dist[i][j] == INF)
32                     System.out.print("INF ");
33                 else
34                     System.out.print(dist[i][j] + " ");
35             }
36             System.out.println();
37         }
38     }
39
40     // Driver's code
Run | Debug
41     public static void main(String[] args)
42     {
43         int graph[][] = { { 0, 3, INF, 7 },
44                           { 8, 0, 2, INF },
45                           { 5, INF, 0, 1 },
46                           { 2, INF, INF, 0 } };
47         prac10 a = new prac10();
48
49         // Function call
50         a.floydWarshall(graph);
51     }
52 }
```

Output:-

```
The following matrix shows the shortest distances between every pair of vertices
0 3 5 6
5 0 2 3
3 6 0 1
2 5 7 0
```

Code:-

```
J prac10.java > ⌂ prac10 > ⌂ floydWarshall(int[][])
1  class prac10 {
2      final static int INF = 99999, V = 4;
3      void floydWarshall(int dist[][])
4      {
5          int i, j, k;
6          for (k = 0; k < V; k++) {
7              // Pick all vertices as source one by one
8              for (i = 0; i < V; i++) {
9                  // Pick all vertices as destination for the
10                 // above picked source
11                 for (j = 0; j < V; j++) {
12                     if (dist[i][k] + dist[k][j]
13                         < dist[i][j])
14                         dist[i][j]
15                         = dist[i][k] + dist[k][j];
16                 }
17             }
18         }
19
20         // Print the shortest distance matrix
21         printSolution(dist);
22     }
23
24     void printSolution(int dist[][])
25     {
26         System.out.println(
27             "The following matrix shows the shortest "
28             + "distances between every pair of vertices");
29         for (int i = 0; i < V; ++i) {
30             for (int j = 0; j < V; ++j) {
31                 if (dist[i][j] == INF)
32                     System.out.print("INF ");
33                 else
34                     System.out.print(dist[i][j] + " ");
35             }
36             System.out.println();
37         }
38     }
39
40     // Driver's code
Run | Debug
41     public static void main(String[] args)
42     {
43         int graph[][] = { { 0, 3, INF, 7 },
44                           { 8, 0, 2, INF },
45                           { 5, INF, 0, 1 },
46                           { 2, INF, INF, 0 } };
47         prac10 a = new prac10();
48
49         // Function call
50         a.floydWarshall(graph);
51     }
52 }
```

Output:-

```
The following matrix shows the shortest distances between every pair of vertices
0 3 5 6
5 0 2 3
3 6 0 1
2 5 7 0
```

Code:-

```
J prac11.java > ᐃ prac11 > ⌂ main(String[])
1 // BFS algorithm in Java
2
3 import java.util.*;
4
5 public class prac11 {
6     private int v;
7     private LinkedList<Integer> adj[];
8
9     // Create a graph
10    prac11(int v) {
11        v = v;
12        adj = new LinkedList[v];
13        for (int i = 0; i < v; ++i)
14            adj[i] = new LinkedList();
15    }
16
17    // Add edges to the graph
18    void addEdge(int v, int w) {
19        adj[v].add(w);
20    }
21
22    // BFS algorithm
23    void BFS(int s) {
24
25        boolean visited[] = new boolean[v];
26
27        LinkedList<Integer> queue = new LinkedList();
28
29        visited[s] = true;
30        queue.add(s);
31
32        while (queue.size() != 0) {
33            s = queue.poll();
34            System.out.print(s + " ");
35
36            Iterator<Integer> i = adj[s].listIterator();
37            while (i.hasNext()) {
38                int n = i.next();
39                if (!visited[n]) {
40                    visited[n] = true;
41                    queue.add(n);
42                }
43            }
44        }
45    }
46
Run | Debug
47    public static void main(String args[]) {
48        prac11 g = new prac11(v:4);
49
50        g.addEdge(v:0, w:1);
51        g.addEdge(v:0, w:2);
52        g.addEdge(v:1, w:2);
53        g.addEdge(v:2, w:0);
54        g.addEdge(v:2, w:3);
55        g.addEdge(v:3, w:3);
56        System.out.println("Following is Breadth First Traversal " + "(starting from vertex 2)");
57
58        g.BFS(s:2);
59    }
60 }
```

Output:-

```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```

Code:-

```
J prac11.java > ᐃ prac11 > ⚙ main(String[])
1 // BFS algorithm in Java
2
3 import java.util.*;
4
5 public class prac11 {
6     private int v;
7     private LinkedList<Integer> adj[];
8
9     // Create a graph
10    prac11(int v) {
11        v = v;
12        adj = new LinkedList[v];
13        for (int i = 0; i < v; ++i)
14            adj[i] = new LinkedList();
15    }
16
17    // Add edges to the graph
18    void addEdge(int v, int w) {
19        adj[v].add(w);
20    }
21
22    // BFS algorithm
23    void BFS(int s) {
24
25        boolean visited[] = new boolean[v];
26
27        LinkedList<Integer> queue = new LinkedList();
28
29        visited[s] = true;
30        queue.add(s);
31
32        while (queue.size() != 0) {
33            s = queue.poll();
34            System.out.print(s + " ");
35
36            Iterator<Integer> i = adj[s].listIterator();
37            while (i.hasNext()) {
38                int n = i.next();
39                if (!visited[n]) {
40                    visited[n] = true;
41                    queue.add(n);
42                }
43            }
44        }
45    }
46
Run | Debug
47    public static void main(String args[]) {
48        prac11 g = new prac11(v:4);
49
50        g.addEdge(v:0, w:1);
51        g.addEdge(v:0, w:2);
52        g.addEdge(v:1, w:2);
53        g.addEdge(v:2, w:0);
54        g.addEdge(v:2, w:3);
55        g.addEdge(v:3, w:3);
56        System.out.println("Following is Breadth First Traversal " + "(starting from vertex 2)");
57
58        g.BFS(s:2);
59    }
60 }
```

Output:-

```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```

Code:-

```
J prac12.java > ...
1 import java.util.*;
2 class prac12 {
3     private int v;
4     private LinkedList<Integer> adj[];
5     @SuppressWarnings("unchecked") prac12(int v)
6     {
7         v = v;
8         adj = new LinkedList[v];
9         for (int i = 0; i < v; ++i)
10            adj[i] = new LinkedList();
11    }
12    void addEdge(int v, int w)
13    {
14        // Add w to v's list.
15        adj[v].add(w);
16    }
17    void DFSUtil(int v, boolean visited[])
18    {
19        visited[v] = true;
20        System.out.print(v + " ");
21
22        Iterator<Integer> i = adj[v].listIterator();
23        while (i.hasNext()) {
24            int n = i.next();
25            if (!visited[n])
26                DFSUtil(n, visited);
27        }
28    }
29
30    void DFS(int v)
31    {
32        boolean visited[] = new boolean[V];
33        DFSUtil(v, visited);
34    }
35
36    // Driver Code
37    public static void main(String args[])
38    {
39        prac12 g = new prac12(V);
40
41        g.addEdge(v:0, w:1);
42        g.addEdge(v:0, w:2);
43        g.addEdge(v:1, w:2);
44        g.addEdge(v:2, w:0);
45        g.addEdge(v:2, w:3);
46        g.addEdge(v:3, w:3);
47
48        System.out.println(
49            "Following is Depth First Traversal "
50            + "(starting from vertex 2)");
51
52        // Function call
53        g.DFS(v:2);
54    }
55 }
```

Output:-

Following is Depth First Traversal (starting from vertex 2)
2 0 1 3

Code:-

```
J prac12.java > ...
1 import java.util.*;
2 class prac12 {
3     private int v;
4     private LinkedList<Integer> adj[];
5     @SuppressWarnings("unchecked") prac12(int v)
6     {
7         v = v;
8         adj = new LinkedList[v];
9         for (int i = 0; i < v; ++i)
10            adj[i] = new LinkedList();
11    }
12    void addEdge(int v, int w)
13    {
14        // Add w to v's list.
15        adj[v].add(w);
16    }
17    void DFSUtil(int v, boolean visited[])
18    {
19        visited[v] = true;
20        System.out.print(v + " ");
21
22        Iterator<Integer> i = adj[v].listIterator();
23        while (i.hasNext()) {
24            int n = i.next();
25            if (!visited[n])
26                DFSUtil(n, visited);
27        }
28    }
29
30    void DFS(int v)
31    {
32        boolean visited[] = new boolean[V];
33        DFSUtil(v, visited);
34    }
35
36    // Driver Code
37    public static void main(String args[])
38    {
39        prac12 g = new prac12(V);
40
41        g.addEdge(v:0, w:1);
42        g.addEdge(v:0, w:2);
43        g.addEdge(v:1, w:2);
44        g.addEdge(v:2, w:0);
45        g.addEdge(v:2, w:3);
46        g.addEdge(v:3, w:3);
47
48        System.out.println(
49            "Following is Depth First Traversal "
50            + "(starting from vertex 2)");
51
52        // Function call
53        g.DFS(v:2);
54    }
55 }
```

Output:-

Following is Depth First Traversal (starting from vertex 2)
2 0 1 3

Code:-

HTML:-

```
<prac8N.html> html
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Fruit Stock</title>
7       <link rel="stylesheet" href="prac8N.css">
8   </head>
9   <body>
10      <div class="navbar">
11          <a href="#">Home</a>
12          <div class="search-container">
13              <input type="text" id="searchInput" placeholder="Search for fruits...">
14              <button onclick="searchFruits()">Search</button>
15          </div>
16      </div>
17
18      <h2 style="text-align: center"> Fruit Stock </h2>
19      <table text-align="center">
20          <thead>
21              <tr>
22                  <th>Fruit Name</th>
23                  <th>Quantity</th>
24              </tr>
25          </thead>
26          <tbody>
27              <tr>
28                  <td>Orange</td>
29                  <td>33</td>
30              </tr>
31              <tr>
32                  <td>Apple</td>
33                  <td>43</td>
34              </tr>
35              <tr>
36                  <td>Pineapple</td>
37                  <td>10</td>
38              </tr>
39              <tr>
40                  <td>Guava</td>
41                  <td>23</td>
42              </tr>
43              <tr>
44                  <td>Dragon Fruit</td>
45                  <td>22</td>
46              </tr>
47              <tr>
48                  <td>Banana</td>
49                  <td>15</td>
50              </tr>
51              <tr>
52                  <td>Pomogrenate</td>
53                  <td>32</td>
54              </tr>
55              <tr>
56                  <td>Papaya</td>
57                  <td>34</td>
58              </tr>
59              <tr>
60                  <td>Cherry</td>
61                  <td>23</td>
62              </tr>
```

```

63      <tr>
64          <td>Kiwi</td>
65          <td>34</td>
66      </tr>
67      <tr>
68          <td>Custurd Apple</td>
69          <td>36</td>
70      </tr>
71      <tr>
72          <td>Pear</td>
73          <td>45</td>
74      </tr>
75      <tr>
76          <td>Grapes</td>
77          <td>16</td>
78      </tr>
79      <tr>
80          <td>Mango</td>
81          <td>10</td>
82      </tr>
83  </tbody>
84 </table>
85 <script src="prac8N.js"></script>
86 </body>
87 </html>

```

CSS:-

```

# prac8N.css > 4 .navbar
1  body {
2      font-family: Arial, sans-serif;
3      margin: 0;
4      padding: 0
5  }
6
7  .navbar {
8      background-color: ■#333;
9      overflow: hidden;
10 }
11
12 .navbar a {
13     float: left;
14     font-size: 16px;
15     color: □white;
16     text-align: center;
17     padding: 16px;
18     text-decoration:none ;
19 }
20
21 .navbar a:hover {
22     background-color: □#ddd;
23     color: ■black;
24 }
25
26 .search-container {
27     float: right;
28     margin-right: 20px;
29 }
30
31 input[type=text] {
32     padding: 5px;
33     margin-top: 8px;
34     border: none;
35     font-size: 16px;
36 }

```

```

37
38 < button {
39   padding: 6px 10px;
40   background-color: #4CAF50;
41   color: white;
42   border: none;
43   cursor: pointer;
44 }
45
46 < button:hover {
47   background-color: #45a049;
48 }
49
50 < table {
51   border-collapse: collapse;
52   width: 25%;
53   align-items: center;
54   margin: 20px auto;
55 }
56
57 < th, td {
58   border: 1px outset #604f4f;
59   padding: 5px;
60   word-spacing: normal;
61   text-align: center;
62 }
63
64
65 < tr:nth-child(even) {
66   background-color: #e8cece;
67 }
68 < tr:nth-child(odd) {
69   background-color: #ecdcdc;
70 }
71
72 < th {
73   background-color: #333;
74   color: white;
75 }

```

Javascript:-

```

js prac8N.js > ⌂ searchFruits
1 function searchFruits() {
2   const input = document.getElementById('searchInput');
3   const filter = input.value.toUpperCase();
4   const table = document.querySelector('table');
5   const rows = table.querySelectorAll('tbody tr');
6
7   rows.forEach((row) => {
8     const nameCell = row.cells[0];
9     if (nameCell) {
10       const name = nameCell.textContent || nameCell.innerText;
11       if (name.toUpperCase().includes(filter)) {
12         row.style.display = '';
13       } else {
14         row.style.display = 'none';
15       }
16     }
17   });
18 }

```

Code:-**HTML:**

```
<> prac8.html > html > body > table > tbody > tr > td
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Student Search</title>
7       <link rel="stylesheet" href="prac8.css">
8   </head>
9   <body>
10      <div class="navbar">
11          <a href="#">Home</a>
12          <div class="search-container">
13              <input type="text" id="searchInput" placeholder="Search for students...">
14              <button onclick="searchStudents()">Search</button>
15          </div>
16      </div>
17
18      <h2>student List</h2>
19      <table>
20          <thead>
21              <tr>
22                  <th>Roll Number</th>
23                  <th>Name</th>
24              </tr>
25          </thead>
26          <tbody>
27              <tr>
28                  <td>1</td>
29                  <td>Aastha Dongare</td>
30              </tr>
31              <tr>
32                  <td>2</td>
33                  <td>Sakshi Choudhary</td>
34              </tr>
35              <tr>
36                  <td>3</td>
37                  <td>Manjiri Dangre</td>
38              </tr>
39              <tr>
40                  <td>4</td>
41                  <td>Gayatri Deshkar</td>
42              </tr>
43              <tr>
44                  <td>5</td>
45                  <td>Vedanti Dhole</td>
46              </tr>
47              <tr>
48                  <td>6</td>
49                  <td>Shraddha Giri</td>
50              </tr>
51              <tr>
52                  <td>7</td>
53                  <td>Pranoti Gulhane</td>
54              </tr>
55              <tr>
56                  <td>8</td>
57                  <td>Sapna Joshi</td>
58              </tr>
59              <tr>
60                  <td>9</td>
```

```
61 |             <td>Kaynat Sheikh</td>
62 |         </tr>
63 |         <tr>
64 |             <td>10</td>
65 |             <td>Vishakha Kurzekar</td>
66 |         </tr>
67 |         <tr>
68 |             <td>11</td>
69 |             <td>Tejaswini Mankar</td>
70 |         </tr>
71 |         <tr>
72 |             <td>12</td>
73 |             <td>Marufa Mansuri</td>
74 |         </tr>
75 |         <tr>
76 |             <td>13</td>
77 |             <td>Kashmini Meshram</td>
78 |         </tr>
79 |         <tr>
80 |             <td>14</td>
81 |             <td>Miheera Jadhav</td>
82 |         </tr>
83 |         <tr>
84 |             <td>15</td>
85 |             <td>Priya Nandanwar</td>
86 |         </tr>
87 |         <tr>
88 |             <td>16</td>
89 |             <td>Sakshi Nimje</td>
90 |         </tr>
91 |         <tr>
92 |             <td>17</td>
93 |             <td>Swarali Prayagi</td>
94 |         </tr>
95 |         <tr>
96 |             <td>18</td>
97 |             <td>Sujata Randhaye</td>
98 |         </tr>
99 |         <tr>
100 |             <td>19</td>
101 |             <td>Isha Rangari</td>
102 |         </tr>
103 |         <tr>
104 |             <td>20</td>
105 |             <td>Vedika Raut</td>
106 |         </tr>
107 |         <tr>
108 |             <td>21</td>
109 |             <td>Sakina Ali</td>
110 |         </tr>
111 |         <tr>
112 |             <td>22</td>
113 |             <td>Chetna Salve</td>
114 |         </tr>
115 |         <tr>
116 |             <td>23</td>
117 |             <td>Rohini Shambharkar</td>
118 |         </tr>
119 |         <tr>
120 |             <td>24</td>
121 |             <td>Priyanka Talokar</td>
122 |         </tr>
```

```

120     <td>24</td>
121     <td>Priyanka Talokar</td>
122   </tr>
123   <tr>
124     <td>25</td>
125     <td>Nisarga Telang</td>
126   </tr>
127   <tr>
128     <td>26</td>
129     <td>Kinjal Tiwari</td>
130   </tr>
131   <tr>
132     <td>27</td>
133     <td>Mrudula Wankar</td>
134   </tr>
135   <tr>
136     <td>28</td>
137     <td>Abhishek Badukale</td>
138   </tr>
139   <tr>
140     <td>29</td>
141     <td>Pratik Bagdi</td>
142   </tr>
143   <tr>
144     <td>30</td>
145     <td>Jay Bele</td>
146   </tr>
147
148   </tbody>
149 </table>
150 <script src="prac8.js"></script>
151 </body>
152 </html>
153

```

CSS:-

```

# prac8.css > 8 th
1  body {
2    font-family: Arial, sans-serif;
3    margin: 0;
4    padding: 0;
5  }
6
7  .navbar {
8    background-color: ■ #333;
9    overflow: hidden;
10 }
11
12 .navbar a {
13   float: left;
14   font-size: 16px;
15   color: □white;
16   text-align: center;
17   padding: 14px 16px;
18   text-decoration: none;
19 }
20
21 .navbar a:hover {
22   background-color: □ #ddd;
23   color: ■black;
24 }
25
26 .search-container {
27   float: right;
28   margin-right: 20px;
29 }
30
31 input[type=text] {
32   padding: 6px;
33   margin-top: 8px;
34   border: none;
35   font-size: 16px;
36 }

```

```

37
38 button {
39   padding: 6px 10px;
40   background-color: #4CAF50;
41   color: white;
42   border: none;
43   cursor: pointer;
44 }
45
46 button:hover {
47   background-color: #45a049;
48 }
49
50 table {
51   border-collapse: collapse;
52   width: 80%;
53   margin: 20px auto;
54 }
55
56 th, td {
57   border: 1px solid #ddd;
58   padding: 8px;
59   text-align: left;
60 }
61
62 tr:nth-child(even) {
63   background-color: #f2f2f2;
64 }
65
66 th {
67   background-color: #333;
68   color: white;
69 }

```

Javascript:-

```

JS prac8.js > ⚏ searchStudents
1  function searchStudents() {
2    const input = document.getElementById('searchInput');
3    const filter = input.value.toUpperCase();
4    const table = document.querySelector('table');
5    const rows = table.querySelectorAll('tbody tr');
6
7    rows.forEach((row) => {
8      const nameCell = row.cells[1]; // Assuming the student names are in the second column (index 1)
9      if (nameCell) {
10        const name = nameCell.textContent || nameCell.innerText;
11        if (name.toUpperCase().includes(filter)) {
12          row.style.display = '';
13        } else {
14          row.style.display = 'none';
15        }
16      }
17    });
18 }

```

A screenshot of a Firefox browser window titled "Student Search". The address bar shows "file:///C:/PS Practical/prac8.html". The page content is a table titled "Student List" with two columns: "Roll Number" and "Name". The table contains 12 rows, each with a roll number from 1 to 12 and a corresponding name. The names listed are Aastha Dongare, Sakshi Choudhary, Manjiri Dangre, Gayatri Deskar, Vedanti Dhole, Shraddha Giri, Pranoti Gulhane, Sapna Joshi, Kaynat Sheikh, Vishakha Kurzekar, Tejaswini Mankar, and Marufa Mansuri.

Roll Number	Name
1	Aastha Dongare
2	Sakshi Choudhary
3	Manjiri Dangre
4	Gayatri Deskar
5	Vedanti Dhole
6	Shraddha Giri
7	Pranoti Gulhane
8	Sapna Joshi
9	Kaynat Sheikh
10	Vishakha Kurzekar
11	Tejaswini Mankar
12	Marufa Mansuri

A screenshot of a Firefox browser window titled "Student Search". The address bar shows "file:///C:/PS Practical/prac8.html". The page content is a table titled "Student List" with two columns: "Roll Number" and "Name". The table contains 1 row with roll number 30 and name Jay Bele. The search term "jay" is visible in the search bar.

Roll Number	Name
30	Jay Bele



A screenshot of a Firefox browser window titled "Fruit Stock". The address bar shows "file:///C/PS Practical/prac8N.html". The page content includes a search bar with "Search for fruits..." and a green "Search" button. Below is a table titled "Fruit Stock" with 15 rows of fruit names and their quantities.

Fruit Name	Quantity
Orange	33
Apple	43
Pineapple	10
Guava	23
Dragon Fruit	22
Banana	15
Pomogrenate	32
Papaya	34
Cherry	23
Kiwi	34
Custurd Apple	36
Pear	45
Grapes	16
Mango	10

A screenshot of a Firefox browser window titled "Fruit Stock". The address bar shows "file:///C/PS Practical/prac8N.html". The search bar contains "Apple" and the green "Search" button is visible. Below is a table titled "Fruit Stock" with 3 rows of fruit names and their quantities.

Fruit Name	Quantity
Apple	43
Pineapple	10
Custurd Apple	36

