Code:

prac4N.java > prac4N

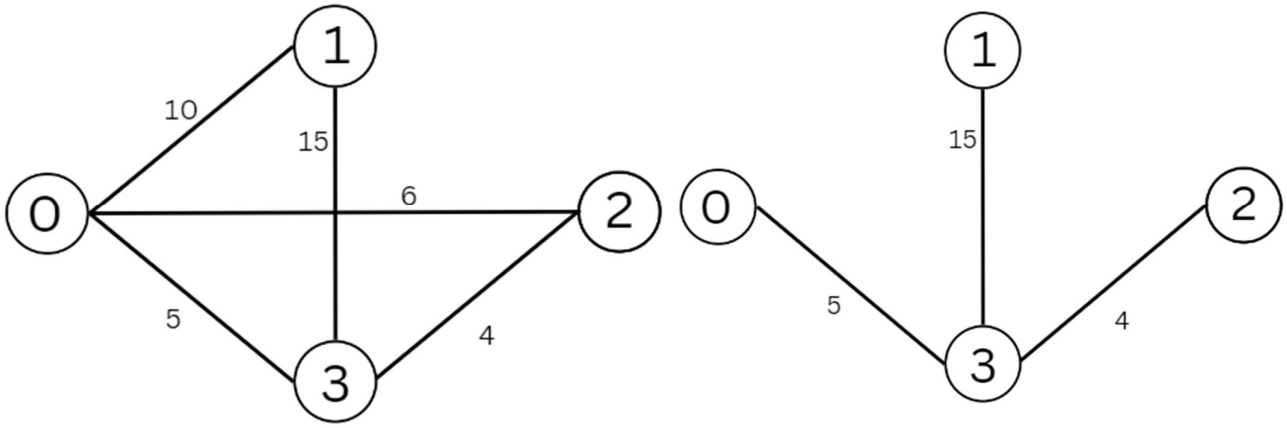
```java
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class prac4N {

    static class Edge {
        int src, dest, weight;

        public Edge(int src, int dest, int weight)
        {
            this.src = src;
            this.dest = dest;
            this.weight = weight;
        }
    }

    static class Subset {
        int parent, rank;

        public Subset(int parent, int rank)
        {
            this.parent = parent;
            this.rank = rank;
        }
    }
    Run | Debug
    public static void main(String[] args)
    {
        int V = 4;
        List<Edge> graphEdges = new ArrayList<Edge>(
            List.of(new Edge(src:0, dest:1, weight:10), new Edge(src:0, dest:2, weight:6),
                    new Edge(src:0, dest:3, weight:5), new Edge(src:1, dest:3, weight:15),
                    new Edge(src:2, dest:3, weight:4)));

        graphEdges.sort(new Comparator<Edge>() {
            @Override public int compare(Edge o1, Edge o2)
            {
                return o1.weight - o2.weight;
            }
        });

        kruskals(V, graphEdges);
    }

    private static void kruskals(int V, List<Edge> edges)
    {
        int j = 0;
        int noOfEdges = 0;

        Subset subsets[] = new Subset[V];

        Edge results[] = new Edge[V];

        for (int i = 0; i < V; i++) {
            subsets[i] = new Subset(i, rank:0);
        }

        while (noOfEdges < V - 1) {

            Edge nextEdge = edges.get(j);
```

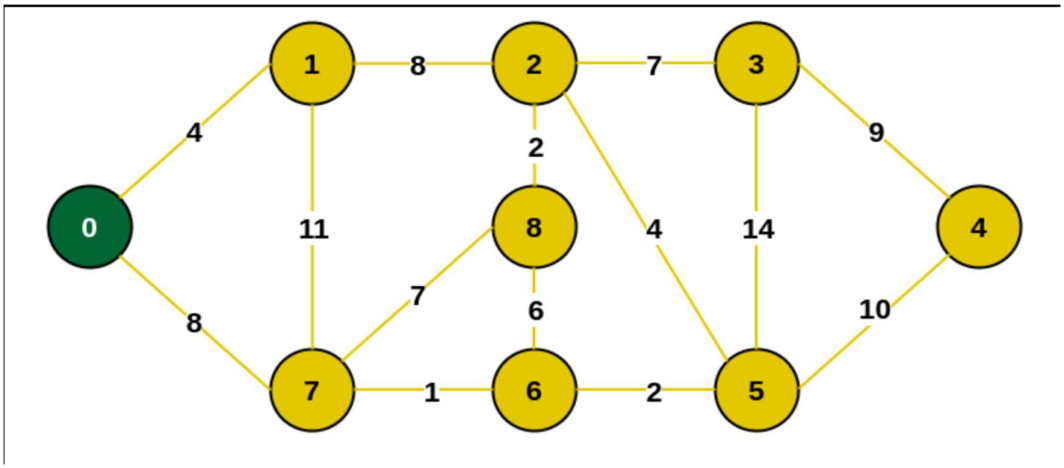
```
            int x = findRoot(subsets, nextEdge.src);
            int y = findRoot(subsets, nextEdge.dest);

            if (x != y) {
                results[noOfEdges] = nextEdge;
                union(subsets, x, y);
                noOfEdges++;
            }

            j++;
        }

        System.out.println(
            x:"Following are the edges of the constructed MST:");
        int minCost = 0;
        for (int i = 0; i < noOfEdges; i++) {
            System.out.println(results[i].src + " -- "
                               + results[i].dest + " == "
                               + results[i].weight);
            minCost += results[i].weight;
        }
        System.out.println("Total cost of MST: " + minCost);
    }

    private static void union(Subset[] subsets, int x,
                              int y)
    {
        int rootX = findRoot(subsets, x);
        int rootY = findRoot(subsets, y);

        if (subsets[rootY].rank < subsets[rootX].rank) {
            subsets[rootY].parent = rootX;
        }
        else if (subsets[rootX].rank
                 < subsets[rootY].rank) {
            subsets[rootX].parent = rootY;
        }
        else {
            subsets[rootY].parent = rootX;
            subsets[rootX].rank++;
        }
    }

    private static int findRoot(Subset[] subsets, int i)
    {
        if (subsets[i].parent == i)
            return subsets[i].parent;

        subsets[i].parent
            = findRoot(subsets, subsets[i].parent);
        return subsets[i].parent;
    }
}
```

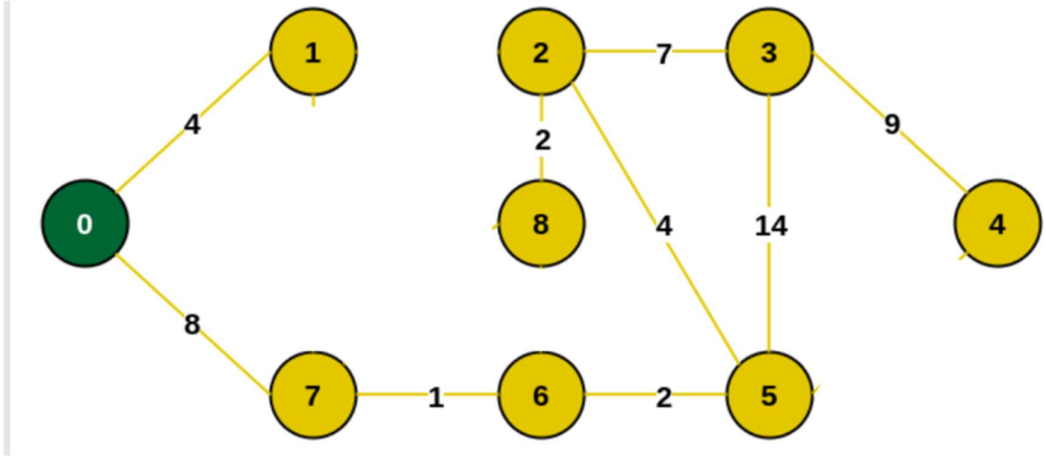
Output:

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

N


1
10
15
6
0
2
5
4
3
1
15
0
2
5
4
3


1
8
2
7
3
4
2
9
0
11
8
4
14
4
7
6
10
8
7
1
6
2
5


1
2
7
3
4
2
9
0
8
4
14
4
8
7
1
6
2
5

Code:

prac4.java > prac4 > kruskalMST()

```java
import java.util.*;
class Edge implements Comparable<Edge> {
    int src, dest, weight;
    public int compareTo(Edge otherEdge) {
        return this.weight - otherEdge.weight;
    }
};
class prac4 {
    int V, E;
    Edge[] edges;
    prac4(int v, int e) {
        V = v;
        E = e;
        edges = new Edge[E];
        for (int i = 0; i < e; ++i) {
            edges[i] = new Edge();
        }
    }
    int find(int[] parent, int i) {
        if (parent[i] == -1) {
            return i;
        }
        return find(parent, parent[i]);
    }
    void union(int[] parent, int x, int y) {
        int xset = find(parent, x);
        int yset = find(parent, y);
        parent[xset] = yset;
    }
    void kruskalMST() {
        Edge[] result = new Edge[V - 1];
        int[] parent = new int[V];
        Arrays.fill(parent, -1);
        Arrays.sort(edges);
        int e = 0;
        int i = 0;
        while (e < V - 1 && i < E) {
            Edge nextEdge = edges[i++];
            int x = find(parent, nextEdge.src);
            int y = find(parent, nextEdge.dest);
            if (x != y) {
                result[e++] = nextEdge;
                union(parent, x, y);
            }
        }
        System.out.println(x:"Edges in Minimum Spanning Tree:");
        int minimumCost = 0;
        for (i = 0; i < e; ++i) {
            System.out.println(result[i].src + " - " + result[i].dest + " : " + result[i].weight);
            minimumCost += result[i].weight;
        }
        System.out.println("Minimum Cost: " + minimumCost);
    }
    Run | Debug
    public static void main(String[] args) {
        int V = 9;
        int E = 14;
        prac4 graph = new prac4(V, E);

        graph.edges[0].src = 0;
        graph.edges[0].dest = 1;
        graph.edges[0].weight = 4;
```

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

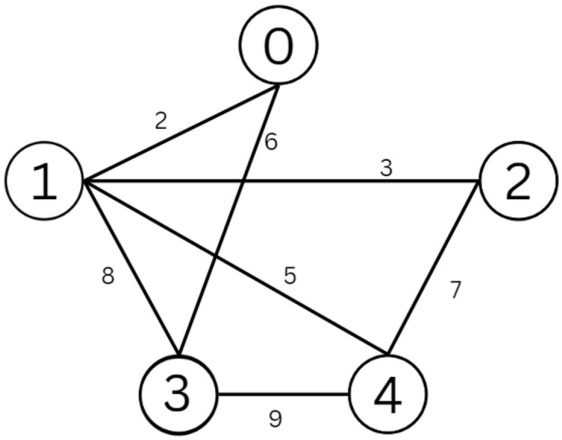
Code:-

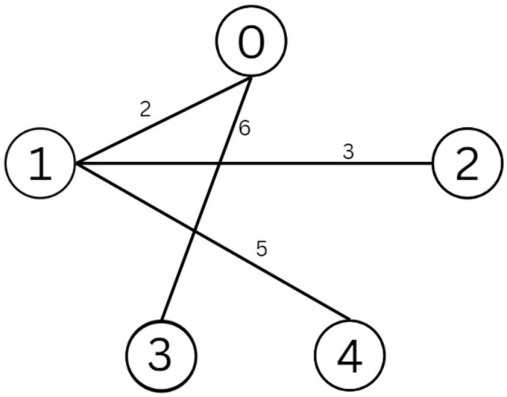
prac5.java > prac5 > main(String[])

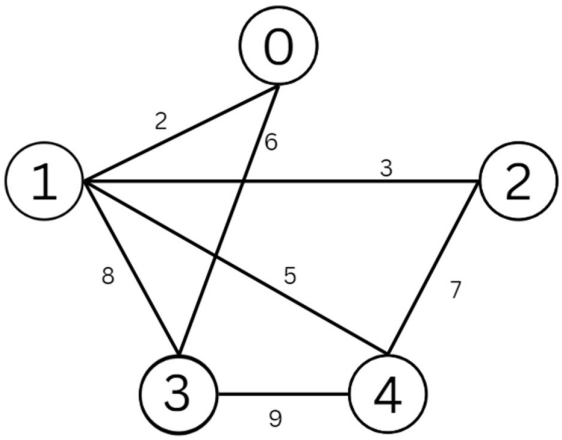
```java
import java.util.Arrays;

class prac5 {
    private static final int V = 5; // Number of vertices

    int minKey(int key[], boolean mstSet[]) {
        int min = Integer.MAX_VALUE;
        int minIndex = -1;
        for (int v = 0; v < V; v++) {
            if (!mstSet[v] && key[v] < min) {
                min = key[v];
                minIndex = v;
            }
        }
        return minIndex;
    }
    void printMST(int parent[], int graph[][]) {
        System.out.println(x:"Edge \tWeight");
        for (int i = 1; i < V; i++) {
            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
        }
    }
    void primMST(int graph[][]) {
        int parent[] = new int[V]; // Array to store constructed MST
        int key[] = new int[V]; // Key values used to pick the minimum weight edge
        boolean mstSet[] = new boolean[V]; // To represent set of vertices included in MST

        Arrays.fill(key, Integer.MAX_VALUE);
        Arrays.fill(mstSet, val:false);

        // Start from the first vertex
        key[0] = 0;
        parent[0] = -1;

        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, mstSet);
            mstSet[u] = true;
            for (int v = 0; v < V; v++) {
                if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
            }
        }
        printMST(parent, graph);
    }
    Run | Debug
    public static void main(String[] args) {
        prac5 mst = new prac5();

        int graph[][] = new int[][]{
            {0, 2, 0, 6, 0},
            {2, 0, 3, 8, 5},
            {0, 3, 0, 0, 7},
            {6, 8, 0, 0, 9},
            {0, 5, 7, 9, 0}
        };

        mst.primMST(graph);
    }
}
```

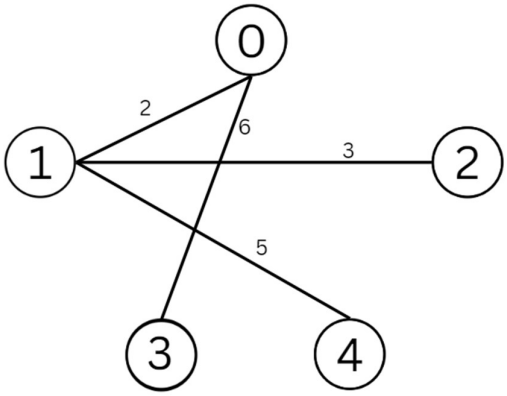
Output:

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


0
1
2
3
4
2
6
3
8
5
7
9


0
1
2
3
4
2
6
3
5


0
1
2
3
4
2
6
3
8
5
7
9


0
1
2
3
4
2
6
3
5

Code:-

prac5.java > prac5 > main(String[])

```java
import java.util.Arrays;

class prac5 {
    private static final int V = 5; // Number of vertices

    int minKey(int key[], boolean mstSet[]) {
        int min = Integer.MAX_VALUE;
        int minIndex = -1;
        for (int v = 0; v < V; v++) {
            if (!mstSet[v] && key[v] < min) {
                min = key[v];
                minIndex = v;
            }
        }
        return minIndex;
    }
    void printMST(int parent[], int graph[][]) {
        System.out.println(x:"Edge \tWeight");
        for (int i = 1; i < V; i++) {
            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
        }
    }
    void primMST(int graph[][]) {
        int parent[] = new int[V]; // Array to store constructed MST
        int key[] = new int[V]; // Key values used to pick the minimum weight edge
        boolean mstSet[] = new boolean[V]; // To represent set of vertices included in MST

        Arrays.fill(key, Integer.MAX_VALUE);
        Arrays.fill(mstSet, val:false);

        // Start from the first vertex
        key[0] = 0;
        parent[0] = -1;

        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, mstSet);
            mstSet[u] = true;
            for (int v = 0; v < V; v++) {
                if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
            }
        }
        printMST(parent, graph);
    }
    Run | Debug
    public static void main(String[] args) {
        prac5 mst = new prac5();

        int graph[][] = new int[][]{
            {0, 2, 0, 6, 0},
            {2, 0, 3, 8, 5},
            {0, 3, 0, 0, 7},
            {6, 8, 0, 0, 9},
            {0, 5, 7, 9, 0}
        };

        mst.primMST(graph);
    }
}
```

Output:

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

Code:-

prac6.java > prac6

```java
import java.util.PriorityQueue;
import java.util.HashMap;
import java.util.Map;

class HuffmanNode implements Comparable<HuffmanNode> {
    char data;
    int frequency;
    HuffmanNode left, right;

    public HuffmanNode(char data, int frequency) {
        this.data = data;
        this.frequency = frequency;
        left = right = null;
    }

    public int compareTo(HuffmanNode node) {
        return this.frequency - node.frequency;
    }
}

public class prac6 {
    public static void printHuffmanCodes(HuffmanNode root, String code) {
        if (root == null) return;

        if (root.left == null && root.right == null) {
            System.out.println(root.data + ": " + code);
        }

        printHuffmanCodes(root.left, code + "0");
        printHuffmanCodes(root.right, code + "1");
    }

    public static void buildHuffmanTree(String text) {
        if (text == null || text.length() == 0) return;

        Map<Character, Integer> frequencyMap = new HashMap<>();
        for (char c : text.toCharArray()) {
            frequencyMap.put(c, frequencyMap.getOrDefault(c, defaultValue:0) + 1);
        }

        PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();
        for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {
            minHeap.offer(new HuffmanNode(entry.getKey(), entry.getValue()));
        }

        while (minHeap.size() > 1) {
            HuffmanNode left = minHeap.poll();
            HuffmanNode right = minHeap.poll();

            HuffmanNode newNode = new HuffmanNode(data:'\0', left.frequency + right.frequency);
            newNode.left = left;
            newNode.right = right;

            minHeap.offer(newNode);
        }

        HuffmanNode root = minHeap.peek();
        printHuffmanCodes(root, code:"");
    }

    Run | Debug
    public static void main(String[] args) {
        String text = "My name is akanksh bodakhe";
        buildHuffmanTree(text);
    }
}
```

Output:-

```
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:-

prac6.java > prac6 > main(String[])

```java
import java.util.PriorityQueue;
import java.util.HashMap;
import java.util.Map;

class HuffmanNode implements Comparable<HuffmanNode> {
    char data;
    int frequency;
    HuffmanNode left, right;

    public HuffmanNode(char data, int frequency) {
        this.data = data;
        this.frequency = frequency;
        left = right = null;
    }

    public int compareTo(HuffmanNode node) {
        return this.frequency - node.frequency;
    }
}

public class prac6 {
    public static void printHuffmanCodes(HuffmanNode root, String code) {
        if (root == null) return;

        if (root.left == null && root.right == null) {
            System.out.println(root.data + ": " + code);
        }

        printHuffmanCodes(root.left, code + "0");
        printHuffmanCodes(root.right, code + "1");
    }

    public static void buildHuffmanTree(String text) {
        if (text == null || text.length() == 0) return;

        Map<Character, Integer> frequencyMap = new HashMap<>();
        for (char c : text.toCharArray()) {
            frequencyMap.put(c, frequencyMap.getOrDefault(c, defaultValue:0) + 1);
        }

        PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();
        for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {
            minHeap.offer(new HuffmanNode(entry.getKey(), entry.getValue()));
        }

        while (minHeap.size() > 1) {
            HuffmanNode left = minHeap.poll();
            HuffmanNode right = minHeap.poll();

            HuffmanNode newNode = new HuffmanNode(data:'\0', left.frequency + right.frequency);
            newNode.left = left;
            newNode.right = right;

            minHeap.offer(newNode);
        }

        HuffmanNode root = minHeap.peek();
        printHuffmanCodes(root, code:"");
    }

    Run | Debug
    public static void main(String[] args) {
        String text = "Hii! I am Nisarga Telang";
        buildHuffmanTree(text);
    }
}
```

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:-

prac7.java > prac7 > main(String[])

```java
import java.util.*;

class Graph {
    private int V; // Number of vertices
    private List<List<Node>> adj;

    public Graph(int V) {
        this.V = V;
        adj = new ArrayList<>(V);
        for (int i = 0; i < V; i++) {
            adj.add(new ArrayList<>());
        }
    }

    public void addEdge(int source, int destination, int weight) {
        Node node = new Node(destination, weight);
        adj.get(source).add(node); // Add edge to the adjacency list
    }

    public void dijkstra(int source) {
        PriorityQueue<Node> minHeap = new PriorityQueue<>(V, Comparator.comparingInt(node -> node.weight));
        int[] dist = new int[V];
        Arrays.fill(dist, Integer.MAX_VALUE);

        // Initialize the source vertex
        dist[source] = 0;
        minHeap.add(new Node(source, weight:0));

        while (!minHeap.isEmpty()) {
            int u = minHeap.poll().vertex;

            for (Node neighbor : adj.get(u)) {
                int v = neighbor.vertex;
                int weight = neighbor.weight;

                // Relaxation step
                if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                    dist[v] = dist[u] + weight;
                    minHeap.add(new Node(v, dist[v]));
                }
            }
        }

        // Print the shortest distances
        System.out.println("Shortest distances from source vertex " + source + ":");
        for (int i = 0; i < V; i++) {
            System.out.println("Vertex " + i + ": " + dist[i]);
        }
    }

    private static class Node {
        int vertex;
        int weight;

        Node(int vertex, int weight) {
            this.vertex = vertex;
            this.weight = weight;
        }
    }
}

```

```java
public class prac7 {
    Run | Debug
    public static void main(String[] args) {
        int V = 6; // Number of vertices
        Graph graph = new Graph(V);

        // Add edges with weights
        graph.addEdge(source:0, destination:1, weight:2);
        graph.addEdge(source:0, destination:2, weight:4);
        graph.addEdge(source:1, destination:2, weight:1);
        graph.addEdge(source:1, destination:3, weight:7);
        graph.addEdge(source:2, destination:4, weight:3);
        graph.addEdge(source:3, destination:4, weight:2);
        graph.addEdge(source:3, destination:5, weight:1);
        graph.addEdge(source:4, destination:5, weight:5);

        int sourceVertex = 0; // Source vertex for Dijkstra's algorithm

        graph.dijkstra(sourceVertex);
    }
}

```

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:-

prac7.java > prac7 > main(String[])

```java
import java.util.*;

class Graph {
    private int V; // Number of vertices
    private List<List<Node>> adj;

    public Graph(int V) {
        this.V = V;
        adj = new ArrayList<>(V);
        for (int i = 0; i < V; i++) {
            adj.add(new ArrayList<>());
        }
    }

    public void addEdge(int source, int destination, int weight) {
        Node node = new Node(destination, weight);
        adj.get(source).add(node); // Add edge to the adjacency list
    }

    public void dijkstra(int source) {
        PriorityQueue<Node> minHeap = new PriorityQueue<>(V, Comparator.comparingInt(node -> node.weight));
        int[] dist = new int[V];
        Arrays.fill(dist, Integer.MAX_VALUE);

        // Initialize the source vertex
        dist[source] = 0;
        minHeap.add(new Node(source, weight:0));

        while (!minHeap.isEmpty()) {
            int u = minHeap.poll().vertex;

            for (Node neighbor : adj.get(u)) {
                int v = neighbor.vertex;
                int weight = neighbor.weight;

                // Relaxation step
                if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                    dist[v] = dist[u] + weight;
                    minHeap.add(new Node(v, dist[v]));
                }
            }
        }

        // Print the shortest distances
        System.out.println("Shortest distances from source vertex " + source + ":");
        for (int i = 0; i < V; i++) {
            System.out.println("Vertex " + i + ": " + dist[i]);
        }
    }

    private static class Node {
        int vertex;
        int weight;

        Node(int vertex, int weight) {
            this.vertex = vertex;
            this.weight = weight;
        }
    }
}

```

```java
public class prac7 {
    Run | Debug
    public static void main(String[] args) {
        int V = 6; // Number of vertices
        Graph graph = new Graph(V);

        // Add edges with weights
        graph.addEdge(source:0, destination:1, weight:2);
        graph.addEdge(source:0, destination:2, weight:4);
        graph.addEdge(source:1, destination:2, weight:1);
        graph.addEdge(source:1, destination:3, weight:7);
        graph.addEdge(source:2, destination:4, weight:3);
        graph.addEdge(source:3, destination:4, weight:2);
        graph.addEdge(source:3, destination:5, weight:1);
        graph.addEdge(source:4, destination:5, weight:5);

        int sourceVertex = 0; // Source vertex for Dijkstra's algorithm

        graph.dijkstra(sourceVertex);
    }
}

```

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:-

prac8.java > prac8 > findLCS(String, String)

```java
public class prac8 {
    public static String findLCS(String str1, String str2) {
        int m = str1.length();
        int n = str2.length();

        // Create a 2D array to store the LCS lengths
        int[][] dp = new int[m + 1][n + 1];

        // Build the LCS matrix using dynamic programming
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }

        // Reconstruct the LCS from the dp matrix
        int i = m;
        int j = n;
        StringBuilder lcs = new StringBuilder();
        while (i > 0 && j > 0) {
            if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
                lcs.insert(offset:0, str1.charAt(i - 1));
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j - 1]) {
                i--;
            } else {
                j--;
            }
        }

        return lcs.toString();
    }

    Run | Debug
    public static void main(String[] args) {
        String str1 = "Akanksh";
        String str2 = "Bodakhe";

        String lcs = findLCS(str1, str2);
        System.out.println("Longest Common Subsequence: " + lcs);
    }
}
```

Output:

```
Longest Common Subsequence: akh
```

Code:-

prac8.java > prac8 > main(String[])

```java
public class prac8 {
    public static String findLCS(String str1, String str2) {
        int m = str1.length();
        int n = str2.length();

        // Create a 2D array to store the LCS lengths
        int[][] dp = new int[m + 1][n + 1];

        // Build the LCS matrix using dynamic programming
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }

        // Reconstruct the LCS from the dp matrix
        int i = m;
        int j = n;
        StringBuilder lcs = new StringBuilder();
        while (i > 0 && j > 0) {
            if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
                lcs.insert(offset:0, str1.charAt(i - 1));
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j - 1]) {
                i--;
            } else {
                j--;
            }
        }

        return lcs.toString();
    }

    Run | Debug
    public static void main(String[] args) {
        String str1 = "Nisarga";
        String str2 = "Telang";

        String lcs = findLCS(str1, str2);
        System.out.println("Longest Common Subsequence: " + lcs);
    }
}
```

Output:-

```
Longest Common Subsequence: ag
```

Code:-

prac9.java > prac9

```java

class prac9 {

    class Edge {
        int src, dest, weight;
        Edge() { src = dest = weight = 0; }
    };

    int V, E;
    Edge edge[];

    prac9(int v, int e)
    {
        V = v;
        E = e;
        edge = new Edge[e];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
    }

    void BellmanFord(prac9 graph, int src)
    {
        int V = graph.V, E = graph.E;
        int dist[] = new int[V];

        for (int i = 0; i < V; ++i)
            dist[i] = Integer.MAX_VALUE;
        dist[src] = 0;

        // have at-most |V| - 1 edges
        for (int i = 1; i < V; ++i) {
            for (int j = 0; j < E; ++j) {
                int u = graph.edge[j].src;
                int v = graph.edge[j].dest;
                int weight = graph.edge[j].weight;
                if (dist[u] != Integer.MAX_VALUE
                    && dist[u] + weight < dist[v])
                    dist[v] = dist[u] + weight;
            }
        }

        for (int j = 0; j < E; ++j) {
            int u = graph.edge[j].src;
            int v = graph.edge[j].dest;
            int weight = graph.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE
                && dist[u] + weight < dist[v]) {
                System.out.println(
                    x:"Graph contains negative weight cycle");
                return;
            }
        }
        printArr(dist, V);
    }

    void printArr(int dist[], int V)
    {
        System.out.println(x:"Vertex Distance from Source");
        for (int i = 0; i < V; ++i)
            System.out.println(i + "\t\t" + dist[i]);
    }
```

```
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     }
105 }
```

Output:-

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

Code:-

prac9.java > prac9

```java

class prac9 {

    class Edge {
        int src, dest, weight;
        Edge() { src = dest = weight = 0; }
    };

    int V, E;
    Edge edge[];

    prac9(int v, int e)
    {
        V = v;
        E = e;
        edge = new Edge[e];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
    }

    void BellmanFord(prac9 graph, int src)
    {
        int V = graph.V, E = graph.E;
        int dist[] = new int[V];

        for (int i = 0; i < V; ++i)
            dist[i] = Integer.MAX_VALUE;
        dist[src] = 0;

        // have at-most |V| - 1 edges
        for (int i = 1; i < V; ++i) {
            for (int j = 0; j < E; ++j) {
                int u = graph.edge[j].src;
                int v = graph.edge[j].dest;
                int weight = graph.edge[j].weight;
                if (dist[u] != Integer.MAX_VALUE
                    && dist[u] + weight < dist[v])
                    dist[v] = dist[u] + weight;
            }
        }

        for (int j = 0; j < E; ++j) {
            int u = graph.edge[j].src;
            int v = graph.edge[j].dest;
            int weight = graph.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE
                && dist[u] + weight < dist[v]) {
                System.out.println(
                    x:"Graph contains negative weight cycle");
                return;
            }
        }
        printArr(dist, V);
    }

    void printArr(int dist[], int V)
    {
        System.out.println(x:"Vertex Distance from Source");
        for (int i = 0; i < V; ++i)
            System.out.println(i + "\t\t" + dist[i]);
    }
```

```
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     }
105 }
```

Output:-

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

Code:-

prac10.java > prac10 > floydWarshall(int[][])

```java
class prac10 {
    final static int INF = 99999, V = 4;
    void floydWarshall(int dist[][])
    {
        int i, j, k;
        for (k = 0; k < V; k++) {
            // Pick all vertices as source one by one
            for (i = 0; i < V; i++) {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; j++) {
                    if (dist[i][k] + dist[k][j]
                        < dist[i][j])
                        dist[i][j]
                            = dist[i][k] + dist[k][j];
                }
            }
        }

        // Print the shortest distance matrix
        printSolution(dist);
    }

    void printSolution(int dist[][])
    {
        System.out.println(
            "The following matrix shows the shortest "
            + "distances between every pair of vertices");
        for (int i = 0; i < V; ++i) {
            for (int j = 0; j < V; ++j) {
                if (dist[i][j] == INF)
                    System.out.print(s:"INF ");
                else
                    System.out.print(dist[i][j] + " ");
            }
            System.out.println();
        }
    }

    // Driver's code
    Run | Debug
    public static void main(String[] args)
    {
        int graph[][] = { { 0, 3, INF, 7 },
                          { 8, 0, 2, INF },
                          { 5, INF, 0, 1 },
                          { 2, INF, INF, 0 } };
        prac10 a = new prac10();

        // Function call
        a.floydWarshall(graph);
    }
}
```

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:-

prac10.java > prac10 > floydWarshall(int[][])

```java
class prac10 {
    final static int INF = 99999, V = 4;
    void floydWarshall(int dist[][])
    {
        int i, j, k;
        for (k = 0; k < V; k++) {
            // Pick all vertices as source one by one
            for (i = 0; i < V; i++) {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; j++) {
                    if (dist[i][k] + dist[k][j]
                        < dist[i][j])
                        dist[i][j]
                            = dist[i][k] + dist[k][j];
                }
            }
        }

        // Print the shortest distance matrix
        printSolution(dist);
    }

    void printSolution(int dist[][])
    {
        System.out.println(
            "The following matrix shows the shortest "
            + "distances between every pair of vertices");
        for (int i = 0; i < V; ++i) {
            for (int j = 0; j < V; ++j) {
                if (dist[i][j] == INF)
                    System.out.print(s:"INF ");
                else
                    System.out.print(dist[i][j] + " ");
            }
            System.out.println();
        }
    }

    // Driver's code
    Run | Debug
    public static void main(String[] args)
    {
        int graph[][] = { { 0, 3, INF, 7 },
                          { 8, 0, 2, INF },
                          { 5, INF, 0, 1 },
                          { 2, INF, INF, 0 } };
        prac10 a = new prac10();

        // Function call
        a.floydWarshall(graph);
    }
}
```

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:-

prac11.java > prac11 > main(String[])

```java
// BFS algorithm in Java

import java.util.*;

public class prac11 {
  private int V;
  private LinkedList<Integer> adj[];

  // Create a graph
  prac11(int v) {
    V = v;
    adj = new LinkedList[v];
    for (int i = 0; i < v; ++i)
      adj[i] = new LinkedList();
  }

  // Add edges to the graph
  void addEdge(int v, int w) {
    adj[v].add(w);
  }

  // BFS algorithm
  void BFS(int s) {

    boolean visited[] = new boolean[V];

    LinkedList<Integer> queue = new LinkedList();

    visited[s] = true;
    queue.add(s);

    while (queue.size() != 0) {
      s = queue.poll();
      System.out.print(s + " ");

      Iterator<Integer> i = adj[s].listIterator();
      while (i.hasNext()) {
        int n = i.next();
        if (!visited[n]) {
          visited[n] = true;
          queue.add(n);
        }
      }
    }
  }

  Run | Debug
  public static void main(String args[]) {
    prac11 g = new prac11(v:4);

    g.addEdge(v:0, w:1);
    g.addEdge(v:0, w:2);
    g.addEdge(v:1, w:2);
    g.addEdge(v:2, w:0);
    g.addEdge(v:2, w:3);
    g.addEdge(v:3, w:3);
    System.out.println("Following is Breadth First Traversal " + "(starting from vertex 2)");

    g.BFS(s:2);
  }
}
```

Output:-

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

Code:-

prac11.java > prac11 > main(String[])

```java
// BFS algorithm in Java

import java.util.*;

public class prac11 {
  private int V;
  private LinkedList<Integer> adj[];

  // Create a graph
  prac11(int v) {
    V = v;
    adj = new LinkedList[v];
    for (int i = 0; i < v; ++i)
      adj[i] = new LinkedList();
  }

  // Add edges to the graph
  void addEdge(int v, int w) {
    adj[v].add(w);
  }

  // BFS algorithm
  void BFS(int s) {

    boolean visited[] = new boolean[V];

    LinkedList<Integer> queue = new LinkedList();

    visited[s] = true;
    queue.add(s);

    while (queue.size() != 0) {
      s = queue.poll();
      System.out.print(s + " ");

      Iterator<Integer> i = adj[s].listIterator();
      while (i.hasNext()) {
        int n = i.next();
        if (!visited[n]) {
          visited[n] = true;
          queue.add(n);
        }
      }
    }
  }

  Run | Debug
  public static void main(String args[]) {
    prac11 g = new prac11(v:4);

    g.addEdge(v:0, w:1);
    g.addEdge(v:0, w:2);
    g.addEdge(v:1, w:2);
    g.addEdge(v:2, w:0);
    g.addEdge(v:2, w:3);
    g.addEdge(v:3, w:3);
    System.out.println("Following is Breadth First Traversal " + "(starting from vertex 2)");

    g.BFS(s:2);
  }
}
```

Output:-

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

Code:-

prac12.java > ...

```java
import java.util.*;
class prac12 {
    private int V;
    private LinkedList<Integer> adj[];
    @SuppressWarnings("unchecked") prac12(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }
    void addEdge(int v, int w)
    {
        // Add w to v's list.
        adj[v].add(w);
    }
    void DFSUtil(int v, boolean visited[])
    {
        visited[v] = true;
        System.out.print(v + " ");

        Iterator<Integer> i = adj[v].listIterator();
        while (i.hasNext()) {
            int n = i.next();
            if (!visited[n])
                DFSUtil(n, visited);
        }
    }

    void DFS(int v)
    {
        boolean visited[] = new boolean[V];
        DFSUtil(v, visited);
    }

    // Driver Code
    public static void main(String args[])
    {
        prac12 g = new prac12(v:4);

        g.addEdge(v:0, w:1);
        g.addEdge(v:0, w:2);
        g.addEdge(v:1, w:2);
        g.addEdge(v:2, w:0);
        g.addEdge(v:2, w:3);
        g.addEdge(v:3, w:3);

        System.out.println(
            "Following is Depth First Traversal "
            + "(starting from vertex 2)");

        // Function call
        g.DFS(v:2);
    }
}
```

Output:-

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

Code:-

prac12.java > ...

```java
import java.util.*;
class prac12 {
    private int V;
    private LinkedList<Integer> adj[];
    @SuppressWarnings("unchecked") prac12(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }
    void addEdge(int v, int w)
    {
        // Add w to v's list.
        adj[v].add(w);
    }
    void DFSUtil(int v, boolean visited[])
    {
        visited[v] = true;
        System.out.print(v + " ");

        Iterator<Integer> i = adj[v].listIterator();
        while (i.hasNext()) {
            int n = i.next();
            if (!visited[n])
                DFSUtil(n, visited);
        }
    }

    void DFS(int v)
    {
        boolean visited[] = new boolean[V];
        DFSUtil(v, visited);
    }

    // Driver Code
    public static void main(String args[])
    {
        prac12 g = new prac12(v:4);

        g.addEdge(v:0, w:1);
        g.addEdge(v:0, w:2);
        g.addEdge(v:1, w:2);
        g.addEdge(v:2, w:0);
        g.addEdge(v:2, w:3);
        g.addEdge(v:3, w:3);

        System.out.println(
            "Following is Depth First Traversal "
            + "(starting from vertex 2)");

        // Function call
        g.DFS(v:2);
    }
}
```

Output:-

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

Code:-

HTML:-

prac8N.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Fruit Stock</title>
    <link rel="stylesheet" href="prac8N.css">
</head>
<body>
    <div class="navbar">
        <a href="#">Home</a>
        <div class="search-container">
            <input type="text" id="searchInput" placeholder="Search for fruits...">
            <button onclick="searchFruits()">Search</button>
        </div>
    </div>

    <h2 style="text-align: center"> Fruit Stock </h2>
    <table text-align="center">
        <thead>
            <tr>
                <th>Fruit Name</th>
                <th>Quantity</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Orange</td>
                <td>33</td>
            </tr>
            <tr>
                <td>Apple</td>
                <td>43</td>
            </tr>
            <tr>
                <td>Pineapple</td>
                <td>10</td>
            </tr>
            <tr>
                <td>Guava</td>
                <td>23</td>
            </tr>
            <tr>
                <td>Dragon Fruit</td>
                <td>22</td>
            </tr>
            <tr>
                <td>Banana</td>
                <td>15</td>
            </tr>
            <tr>
                <td>Pomogrenate</td>
                <td>32</td>
            </tr>
            <tr>
                <td>Papaya</td>
                <td>34</td>
            </tr>
            <tr>
                <td>Cherry</td>
                <td>23</td>
            </tr>
```

```
            <tr>
                <td>Kiwi</td>
                <td>34</td>
            </tr>
            <tr>
                <td>Custurd Apple</td>
                <td>36</td>
            </tr>
            <tr>
                <td>Pear</td>
                <td>45</td>
            </tr>
            <tr>
                <td>Grapes</td>
                <td>16</td>
            </tr>
            <tr>
                <td>Mango</td>
                <td>10</td>
            </tr>
        </tbody>
    </table>
<script src="prac8N.js"></script>
</body>
</html>
```

CSS:-

prac8N.css > .navbar

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0
}

.navbar {
    background-color: #333;
    overflow: hidden;
}

.navbar a {
    float: left;
    font-size: 16px;
    color: white;
    text-align: center;
    padding: 16px;
    text-decoration:none ;
}

.navbar a:hover {
    background-color: #ddd;
    color: black;
}

.search-container {
    float: right;
    margin-right: 20px;
}

input[type=text] {
    padding: 5px;
    margin-top: 8px;
    border: none;
    font-size: 16px;
}
```

```css
button {
    padding: 6px 10px;
    background-color: #4CAF50;
    color: white;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: #45a049;
}

table {
    border-collapse: collapse;
    width: 25%;
    align-items: center;
    margin: 20px auto;
}

th, td {
    border: 1px outset #604f4f;
    padding: 5px;
    word-spacing: normal;
    text-align: center;

}

tr:nth-child(even) {
    background-color: #e8cece;
}
tr:nth-child(odd) {
    background-color: #ecdcdc;
}

th {
    background-color: #333;
    color: white;
}
```

Javascript:-

JS prac8N.js > searchFruits

```javascript
function searchFruits() {
    const input = document.getElementById('searchInput');
    const filter = input.value.toUpperCase();
    const table = document.querySelector('table');
    const rows = table.querySelectorAll('tbody tr');

    rows.forEach((row) => {
        const nameCell = row.cells[0];
        if (nameCell) {
            const name = nameCell.textContent || nameCell.innerText;
            if (name.toUpperCase().includes(filter)) {
                row.style.display = '';
            } else {
                row.style.display = 'none';
            }
        }
    });
}
```

Code:-

HTML:

prac8.html > html > body > table > tbody > tr > td

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student Search</title>
    <link rel="stylesheet" href="prac8.css">
</head>
<body>
    <div class="navbar">
        <a href="#">Home</a>
        <div class="search-container">
            <input type="text" id="searchInput" placeholder="Search for students...">
            <button onclick="searchStudents()">Search</button>
        </div>
    </div>

    <h2>Student List</h2>
    <table>
        <thead>
            <tr>
                <th>Roll Number</th>
                <th>Name</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>1</td>
                <td>Aastha Dongare</td>
            </tr>
            <tr>
                <td>2</td>
                <td>Sakshi Choudhary</td>
            </tr>
            <tr>
                <td>3</td>
                <td>Manjiri Dangre</td>
            </tr>
            <tr>
                <td>4</td>
                <td>Gayatri Deshkar</td>
            </tr>
            <tr>
                <td>5</td>
                <td>Vedanti Dhole</td>
            </tr>
            <tr>
                <td>6</td>
                <td>Shraddha Giri</td>
            </tr>
            <tr>
                <td>7</td>
                <td>Pranoti Gulhane</td>
            </tr>
            <tr>
                <td>8</td>
                <td>Sapna Joshi</td>
            </tr>
            <tr>
                <td>9</td>
```

```
            <td>Kaynat Sheikh</td>
        </tr>
        <tr>
            <td>10</td>
            <td>Vishakha Kurzekar</td>
        </tr>
        <tr>
            <td>11</td>
            <td>Tejaswini Mankar</td>
        </tr>
        <tr>
            <td>12</td>
            <td>Marufa Mansuri</td>
        </tr>
        <tr>
            <td>13</td>
            <td>Kashmini Meshram</td>
        </tr>
        <tr>
            <td>14</td>
            <td>Miheera Jadhav</td>
        </tr>
        <tr>
            <td>15</td>
            <td>Priya Nandanwar</td>
        </tr>
        <tr>
            <td>16</td>
            <td>Sakshi Nimje</td>
        </tr>
        <tr>
            <td>17</td>
            <td>Swarali Prayagi</td>
        </tr>
        <tr>
            <td>18</td>
            <td>Sujata Randhaye</td>
        </tr>
        <tr>
            <td>19</td>
            <td>Isha Rangari</td>
        </tr>
        <tr>
            <td>20</td>
            <td>Vedika Raut</td>
        </tr>
        <tr>
            <td>21</td>
            <td>Sakina Ali</td>
        </tr>
        <tr>
            <td>22</td>
            <td>Chetna Salve</td>
        </tr>
        <tr>
            <td>23</td>
            <td>Rohini Shambharkar</td>
        </tr>
        <tr>
            <td>24</td>
            <td>Priyanka Talokar</td>
        </tr>
```

```
                <td>24</td>
                <td>Priyanka Talokar</td>
            </tr>
            <tr>
                <td>25</td>
                <td>Nisarga Telang</td>
            </tr>
            <tr>
                <td>26</td>
                <td>Kinjal Tiwari</td>
            </tr>
            <tr>
                <td>27</td>
                <td>Mrudula Wankar</td>
            </tr>
            <tr>
                <td>28</td>
                <td>Abhishek Badukale</td>
            </tr>
            <tr>
                <td>29</td>
                <td>Pratik Bagdi</td>
            </tr>
            <tr>
                <td>30</td>
                <td>Jay Bele</td>
            </tr>

        </tbody>
    </table>
    <script src="prac8.js"></script>
</body>
</html>
```

CSS:-

prac8.css > th

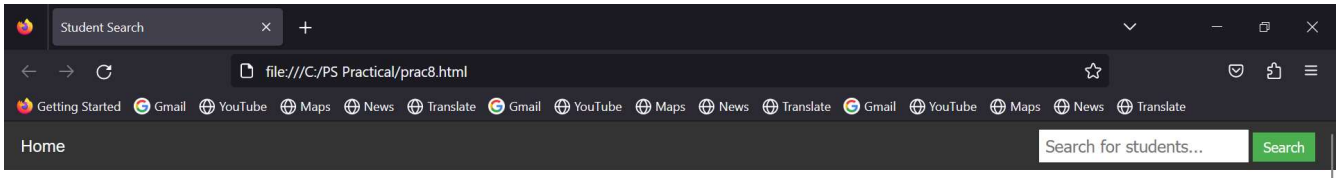
```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

.navbar {
    background-color: #333;
    overflow: hidden;
}

.navbar a {
    float: left;
    font-size: 16px;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}

.navbar a:hover {
    background-color: #ddd;
    color: black;
}

.search-container {
    float: right;
    margin-right: 20px;
}

input[type=text] {
    padding: 6px;
    margin-top: 8px;
    border: none;
    font-size: 16px;
}
```

```css

button {
    padding: 6px 10px;
    background-color: #4CAF50;
    color: white;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: #45a049;
}

table {
    border-collapse: collapse;
    width: 80%;
    margin: 20px auto;
}

th, td {
    border: 1px solid #ddd;
    padding: 8px;
    text-align: left;
}

tr:nth-child(even) {
    background-color: #f2f2f2;
}

th {
    background-color: #333;
    color: white;
}
```

Javascript:-

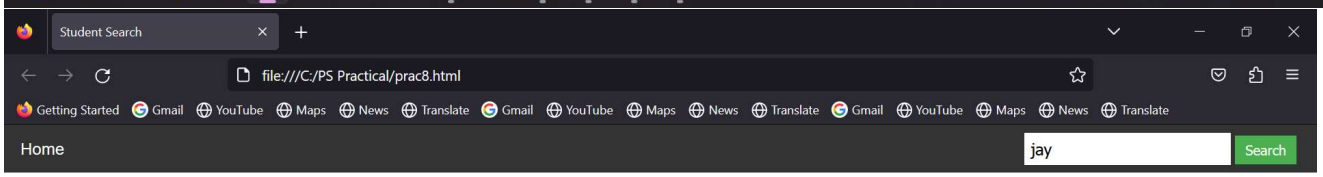
prac8.js > searchStudents

```javascript
function searchStudents() {
    const input = document.getElementById('searchInput');
    const filter = input.value.toUpperCase();
    const table = document.querySelector('table');
    const rows = table.querySelectorAll('tbody tr');

    rows.forEach((row) => {
        const nameCell = row.cells[1]; // Assuming the student names are in the second column (index 1)
        if (nameCell) {
            const name = nameCell.textContent || nameCell.innerText;
            if (name.toUpperCase().includes(filter)) {
                row.style.display = '';
            } else {
                row.style.display = 'none';
            }
        }
    });
}
```



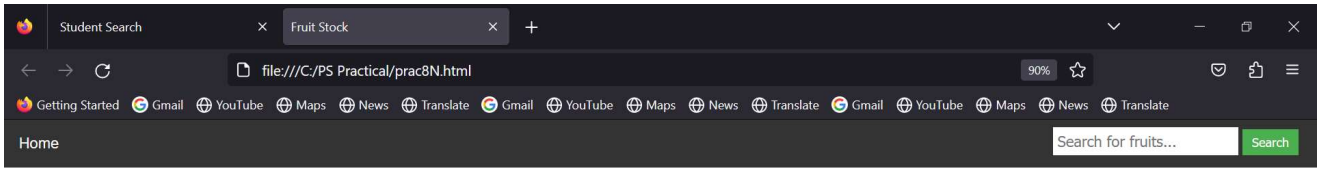
Student List

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



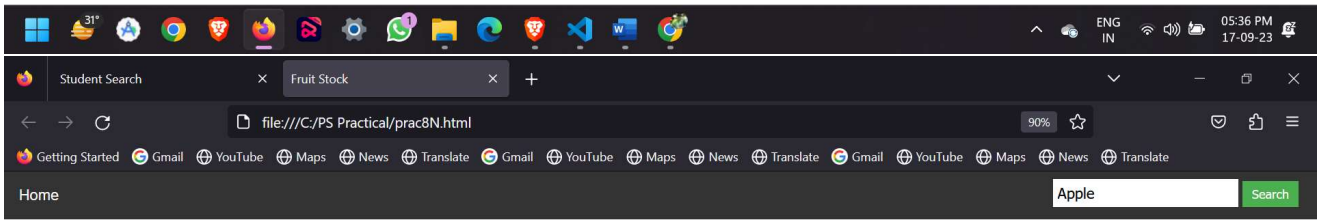
Student List

Roll Number	Name
30	Jay Bele



Fruit Stock

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



Fruit Stock

Fruit Name	Quantity
Apple	43
Pineapple	10
Custurd Apple	36