
Contents

1	Misc	1	2.6 SparseTable	15
1.1	Contest	1	2.7 EulerTour	15
1.1.1	Fast I/O	1		
1.2	Tools	2		
1.2.1	<random>	2		
1.3	Algorithms	3	3 Math	16
1.3.1	Manacher Algo	3	3.1 Number Theory	16
1.3.2	TreeDiameter	4	3.1.1 Linear Sieve	16
1.3.3	GCD On Path (max, min etc)	5	3.1.2 Get Factors and SPF Fucn	16
1.3.4	SLiding Window	6	3.2 Combinatorics	17
1.3.5	NegCycle IN Directed Graph	7	3.2.1 Comb	17
1.3.6	UpDown Algo IN Tree	8		
1.3.7	KMP	9	4 UTILITY	18
1.3.8	Palindrome Subsequence	10		
1.3.9	Longest Increasing Subsequence	11		
2	Data Structures	11	1. Misc	
2.1	Fenwick Tree	11	1.1. Contest	
2.2	Segment Tree (SIMPLE)	11	1.1.1. Fast I/O	
2.3	Lazy Segment Tree (SIMPLE)	12	<pre>public class fast_io { public static PrintWriter out = new PrintWriter(new BufferedOutputStream(System.out)); static FASTIO in = new FASTIO(); public static void main(String[] args) throws IOException { int cp = in.nextInt(); while (cp-- > 0) { solve(); } out.close(); } }</pre>	
2.4	Binary Lifting (1 based)	13		
2.5	DSU	14		

```

11    }
12    static void solve() {}
13    static class FASTIO {
14        BufferedReader br;
15        StringTokenizer st;
16        public FASTIO() {
17            br = new BufferedReader(
18                new InputStreamReader(System.in)
19            );
20        }
21        String next() {
22            while (st == null || !st.hasMoreElements()) {
23                try {
24                    st = new StringTokenizer(br.readLine());
25                } catch (IOException e) {
26                    e.printStackTrace();
27                }
28            }
29            return st.nextToken();
30        }
31        int nextInt() {
32            return Integer.parseInt(next());
33        }
34        long nextLong() {
35            return Long.parseLong(next());
36        }
37        double nextDouble() {
38            return Double.parseDouble(next());
39        }
40        String nextLine() {
41            String str = "";
42            try {
43                st = null;

```

```

44            str = br.readLine();
45        } catch (IOException e) {
46            e.printStackTrace();
47        }
48        return str;
49    }
50}
51

```

1.2. Tools

1.2.1. <random>

```

1 import java.util.Random;
2
3 class random {
4     static final Random rng = new Random();
5
6     static int randInt(int l, int r) {
7         return l + rng.nextInt(r - l + 1);
8     }
9
10    static long randLong(long l, long r) {
11        return l + (Math.abs(rng.nextLong()) % (r - l
12            → + 1));
13    }
14    // use inside the main
15    // int a = randInt(1, 10);
16    // long b = randLong(100, 1000);
17
18    // ----- RANDOM (CP TEMPLATE) -----
19    // mt19937_64
→     rng(chrono::steady_clock::now().time_since_epoch().count());

```

```

20 // inline int rnd(int l, int r) {
21 //     return uniform_int_distribution<int>(l,
22 //         r)(rng);
23 //}
24
25 // inline long long rndll(long long l, long long r) {
26 //     return uniform_int_distribution<long long>(l,
27 //         r)(rng);
28 //}

```

1.3. Algorithms

1.3.1. Manacher Algo

```

1 public class Manacher {
2     public static void main(String[] args) {
3         String s = "aabaac";
4         manacher m = new manacher(s);
5         System.out.println(m.getLongestString());
6     }
7 }
8 class manacher {
9     String s, t;
10    int[] p;
11    public manacher(String s) {
12        this.s = s; build();
13    }
14    public void build() {
15        StringBuilder sb = new StringBuilder("#");
16        for (char ch : s.toCharArray())
17            sb.append(ch).append('#');
18        t = sb.toString();
19        int n = t.length();

```

```

20     p = new int[n];
21     int l = 0, r = 0;
22     for (int i = 0; i < n; i++) {
23         int mirror = l + r - i;
24         if (i < r)
25             p[i] = Math.min(r - i, p[mirror]);
26         while (i + p[i] + 1 < n && i - p[i] - 1 >= 0
27             && t.charAt(i + p[i] + 1) == t.charAt(i - p[i]
28                 - 1))
29             p[i]++;
30         if (i + p[i] > r) {
31             l = i - p[i];
32             r = i + p[i];
33         }
34     }
35     public boolean isPal(int l, int r) {
36         int center = l + r + 1;
37         int length = r - l + 1;
38         return p[center] >= length;
39     }
40     // Returns the length of the longest palindrome
41     public int getLongest() {
42         int maxLen = 0;
43         for (int x : p)
44             if (x > maxLen)
45                 maxLen = x;
46         return maxLen;
47     }
48     // Returns the actual longest palindromic substring
49     public String getLongestString() {
50         int maxLen = 0, center = 0;

```

```

51   for (int i = 0; i < p.length; i++) {
52     if (p[i] > maxLen) {
53       maxLen = p[i];
54       center = i;
55     }
56   }
57   // Map back to original string
58   int start = (center - maxLen) / 2;
59   return s.substring(start, start + maxLen);
60 }
61 }
```

1.3.2. TreeDiameter

```

1  public class TreeDiameter {
2    public static void main(String[] args) {
3      solve(); // out.close();
4    }
5    private static void solve() {
6      int n = in.nextInt();
7      List<List<Integer>> edges = new ArrayList<>();
8      for (int i = 0; i <= n; i++) {
9        edges.add(new ArrayList<>());
10     }
11     for (int i = 0; i < n - 1; i++) {
12       int u = in.nextInt();
13       int v = in.nextInt();
14       edges.get(u).add(v);
15       edges.get(v).add(u);
16     }
17     int[] distX = new int[n + 1];
18     int[] distY = new int[n + 1];
19     Arrays.fill(distX, -1);
```

```

20   Arrays.fill(distY, -1);
21   int x = 1;
22   // First DFS from a random node to find a
23   // farthest node
24   dfs(x, edges, -1, distX);
25   int y = farthestNode(n, distX);
26   // Second DFS from farthest node to
27   // find the farthest node from it
28   dfs(y, edges, -1, distY);
29   int z = farthestNode(n, distY);
30   // Print the diameter of the tree
31   System.out.println(distY[z]);
32 }
33 private static void dfs(int curr,
34   List<List<Integer>> edges, int parent, int[]
35   level) {
36   if (parent == -1)
37     level[curr] = 0;
38   else
39     level[curr] = level[parent] + 1;
40   for (int neighbor : edges.get(curr)) {
41     if (neighbor != parent)
42       dfs(neighbor, edges, curr, level);
43   }
44   // Find the farthest node from a given node
45   private static int farthestNode(int n, int[] dist)
46   {
47     int farthest = 0;
48     for (int i = 0; i <= n; i++) {
49       if (dist[i] > dist[farthest])
50         farthest = i;
```

```

49         }
50     }
51 }
52 }
```

1.3.3. GCD On Path (max, min etc)

```

1 import java.util.*;
2 public class GCDOnPath {
3     static final int MAX_LOG = 20;
4     static final int N = (int) 2e5 + 1;
5     static int[][] parent = new int[N][MAX_LOG];
6     static int[][] gcdVal = new int[N][MAX_LOG];
7     static int[] depth = new int[N];
8     static int[] arr = new int[N];
9     static List<List<Integer>> adj;
10
11    public static void main(String[] args) { }
12
13    private static void solve() {
14        dfs(1, 0);
15    }
16    private static void dfs(int node, int par) {
17        depth[node] = depth[par] + 1;
18        parent[node][0] = par;
19        gcdVal[node][0] = gcd(arr[node], arr[par]);
20        for (int j = 1; j < MAX_LOG; j++) {
21            int midParent = parent[node][j - 1];
22            parent[node][j] = parent[midParent][j - 1];
23            gcdVal[node][j] = gcd(gcdVal[node][j - 1],
24                gcdVal[midParent][j - 1]);
25        }
26        for (int child : adj.get(node)) {
```

```

27             if (child != par) {
28                 dfs(child, node);
29             }
30         }
31     }
32     private static int getGCDOnPath(int a, int b) {
33         int g = gcd(arr[a], arr[b]);
34         if (depth[a] < depth[b]) {
35             int temp = a;
36             a = b; b = temp;
37         }
38         int diff = depth[a] - depth[b];
39         for (int j = MAX_LOG - 1; j >= 0; j--) {
40             if (((1 << j) & diff) != 0) {
41                 g = gcd(g, gcdVal[a][j]);
42                 a = parent[a][j];
43             }
44         }
45
46         if (a == b)
47             return g;
48         for (int j = MAX_LOG - 1; j >= 0; j--) {
49             if (parent[a][j] != parent[b][j]) {
50                 g = gcd(g, gcd(gcdVal[a][j], gcdVal[b][j]));
51                 a = parent[a][j];
52                 b = parent[b][j];
53             }
54         }
55         g = gcd(g, gcd(gcdVal[a][0], gcdVal[b][0]));
56         return g;
57     }
58     private static int gcd(int a, int b) {
59         return b == 0 ? a : gcd(b, a % b);
```

```
60  }
61 }
```

1.3.4. Sliding Window

```
1 import java.util.*;
2 // Dequeue Optimization-->
3
4 public class SlidingWindow {
5     public static void main(String[] args) { }
6     // Function to find the minimum in each subarray of
7     // size k
8     private static List<Integer> sliding_wind_min(int[]
9         arr, int k) {
10        List<Integer> ans = new ArrayList<>();
11        int n = arr.length;
12        Deque<Integer> deque = new LinkedList<>();
13        for (int i = 0; i < n; i++) {
14            // Remove elements out of the current window
15            if (!deque.isEmpty() && deque.getFirst() < i - k
16                + 1) {
17                deque.pollFirst();
18            }
19            // Remove elements from the deque that are
20            // greater than or equal to the current
21            // element
22            while (!deque.isEmpty() && arr[deque.getLast()]
23                >= arr[i]) {
24                deque.pollLast();
25            }
26            // Add the current element index to the deque
27            deque.offerLast(i);
28            // Once the first window is fully traversed,
```

```
25             // start adding results
26             if (i >= k - 1)
27                 ans.add(arr[deque.getFirst()]);
28         }
29         return ans;
30     }
31     // code to find the sliding window maximum of size
32     // k.
33     public int[] maxSlidingWindow(int[] nums, int k) {
34         int n = nums.length;
35         int[] ans = new int[n + 1 - k];
36         TreeMap<Integer, Integer> map = new TreeMap<>();
37         int l = 0;
38         for (int r = 0; r < n; r++) {
39             map.put(nums[r], map.getOrDefault(nums[r], 0) +
40                     1);
41             if (r - l + 1 == k) {
42                 ans[l] = map.lastKey();
43                 int val = nums[l];
44                 if (map.get(val) == 1) {
45                     map.remove(val);
46                 } else {
47                     map.put(val, map.get(val) - 1);
48                 }
49                 l++;
50             }
51         }
52         return ans;
53     }
54     // max num is sliding window of size k.
55     public int[] maxSlidingWindow2(int[] nums, int k) {
56         int n = nums.length;
57         int[] ans = new int[n - k + 1];
```

```

55     int idx = 0;
56     Deque<Integer> deque = new LinkedList<>();
57     for (int i = 0; i < n; i++) {
58         if (!deque.isEmpty() && deque.getFirst() < i - k
59             + 1) {
60             deque.pollFirst();
61         }
62         while (!deque.isEmpty() && nums[deque.getLast()]
63             <= nums[i]) {
64             deque.pollLast();
65         }
66         deque.offerLast(i);
67         if (i >= k - 1)
68             ans[idx++] = nums[deque.getFirst()];
69     }
70     return ans;
71 } // Function to find the sliding window Median.
72 public double[] medianSlidingWindow(int[] nums, int
73                                     k) {
74     TreeSet<Integer> minSet = new TreeSet<>(
75         (a, b) -> nums[a] == nums[b] ? a - b
76             : Integer.compare(nums[a], nums[b]));
77     TreeSet<Integer> maxSet = new TreeSet<>(
78         (a, b) -> nums[a] == nums[b] ? a - b
79             : Integer.compare(nums[a], nums[b]));
80
81     double[] ans = new double[nums.length - k + 1];
82     for (int i = 0; i < nums.length; i++) {
83         minSet.add(i); // add the index in the low
84         maxSet.add(minSet.pollLast());
85         // add the last of minSet to max.
86         if (minSet.size() < maxSet.size())
87             ans[i] = ((double) nums[minSet.last()]
88             + nums[maxSet.first()]) / 2;
89         else
90             ans[i] = ((double) nums[minSet.last()])
91                 + nums[maxSet.first()]);
92         if (i >= k - 1) {
93             if (k % 2 == 0)
94                 ans[i - k + 1] = ((double)
95                     nums[minSet.last()]
96                     + nums[maxSet.first()]) / 2;
97             else
98                 ans[i - k + 1] = (double)
99                     nums[minSet.last()];
100            if (!minSet.remove(i - k + 1))
101                maxSet.remove(i - k + 1);
102        }
103    }
104    return ans;
105 }
```

```

// if low < high add the first from the high
// to the low set.
minSet.add(maxSet.pollFirst());
if (i >= k - 1) {
    if (k % 2 == 0)
        ans[i - k + 1] = ((double)
            nums[minSet.last()]
            + nums[maxSet.first()]) / 2;
    else
        ans[i - k + 1] = (double)
            nums[minSet.last()];
    if (!minSet.remove(i - k + 1))
        maxSet.remove(i - k + 1);
}
return ans;
}
```

1.3.5. NegCycle IN Directed Graph

```

public class negCycleDetectDG {
    public static void main(String[] args) throws
        IOException { }
    static void solve() {
        int[] parent = new int[n + 1];
        long[] dist = new long[n + 1];
        Arrays.fill(dist, INF);
        Arrays.fill(parent, -1);
        dist[1] = 0;
        int startNode = -1;
        // Run Bellman-Ford n times
        for (int i = 0; i < n; i++) {
```

```

12     startNode = -1;
13     for (long[] e : edges) {
14         int u = (int) e[0], v = (int) e[1];
15         long w = e[2];
16         if (dist[u] + w < dist[v]) {
17             dist[v] = dist[u] + w;
18             parent[v] = u;
19             startNode = v;
20         }
21     }
22     if (startNode == -1) {
23         out.println("NO");
24         return;
25     } // To ensure we are inside the cycle
26     for (int i = 0; i < n; i++) {
27         startNode = parent[startNode];
28     }
29     List<Integer> cycle = new ArrayList<>();
30     int v = startNode;
31     do {
32         cycle.add(v);
33         v = parent[v];
34     } while (v != startNode);
35     cycle.add(startNode);
36     Collections.reverse(cycle);
37     out.println("YES");
38     for (int node : cycle) {
39         out.print(node + " ");
40     }
41     out.println();
42 }
43 }
```

```
44 }
```

1.3.6. UpDown Algo IN Tree

```

1 // here we are finding the max dist in subTree in
2 // down1[node]
3 // and second max leaf dist in down2[node], also the
4 // up[node]
5 //= max dist not in the subTree. and the heavy[node]
6 // give in
7 //subTree frim which node the max distance means
8 // down1[node] is comming .
9 public class UpDownDist {
10    public static void main(String[] args) { }
11    static List<List<Integer>> adj;
12    static int[] depth, parent, down1, down2, heavy, up;
13    static void solve() {
14        depth = new int[n + 1];
15        down1 = new int[n + 1];
16        down2 = new int[n + 1];
17        heavy = new int[n + 1];
18        up = new int[n + 1];
19        parent = new int[n + 1];
20        dfsDepth(1, -1);
21        dfsDown(1, -1);
22        up[1] = 0;
23        dfsUp(1, -1);
24        long ans = -INF;
25        for (int node = 1; node <= n; node++) {
26            long curr = k * (long) Math.max(down1[node],
27                up[node]) - c * (long) depth[node];
28            ans = Math.max(ans, curr);
29        }
30    }
31 }
```

```

26     out.println(ans);
27 }
28 static void dfsUp(int node, int par) {
29     for (int adjNode : adj.get(node)) {
30         if (adjNode == par)
31             continue;
32         int curr = (heavy[node] == adjNode ?
33             down2[node] : down1[node]);
34         up[adjNode] = 1 + Math.max(up[node], curr);
35         dfsUp(adjNode, node);
36     }
37 }
38 static void dfsDown(int node, int p) {
39     down1[node] = down2[node] = 0;
40     heavy[node] = -1;
41     for (int adjNode : adj.get(node)) {
42         if (adjNode == p)
43             continue;
44         dfsDown(adjNode, node);
45         int curr = 1 + down1[adjNode];
46         if (curr > down1[node]) {
47             down2[node] = down1[node];
48             down1[node] = curr;
49             heavy[node] = adjNode;
50         } else if (curr > down2[node]) {
51             down2[node] = curr;
52         }
53     }
54 }
55 static void dfsDepth(int node, int p) {
56     parent[node] = p;
57     for (int adjNode : adj.get(node)) {
58         if (adjNode == p)

```

```

59         continue;
60         depth[adjNode] = 1 + depth[node];
61         dfsDepth(adjNode, node);
62     }
63 }
64 }
```

1.3.7. KMP

```

1 import java.util.*;
2 public class KMP {
3     private int n, m;
4     private String text, pattern;
5     private int[] LPS;
6     public KMP(String text, String pattern) {
7         this.text = text;
8         this.pattern = pattern;
9         this.n = text.length();
10        this.m = pattern.length();
11        this.LPS = new int[m];
12        generateLPS();
13    }
14    private void generateLPS() {
15        int len = 0;
16        int i = 1;
17        while (i < m) {
18            if (pattern.charAt(i) == pattern.charAt(len))
19                LPS[i++] = ++len;
20            else {
21                if (len != 0)
22                    len = LPS[len - 1];
23                else
24                    LPS[i++] = 0;

```

```

25     }
26   }
27 }
28
29 public List<int[]> countOccurrences() {
30   List<int[]> result = new ArrayList<>();
31   int i = 0, j = 0;
32   while (i < n) {
33     if (text.charAt(i) == pattern.charAt(j)) {
34       i++;
35       j++;
36     }
37     if (j == m) {
38       int start = i - m;
39       int end = i - 1;
40       result.add(new int[] { start, end });
41       j = LPS[j - 1];
42     } else if (i < n && text.charAt(i) !=
43       pattern.charAt(j)) {
44       if (j != 0) {
45         j = LPS[j - 1];
46       } else {
47         i++;
48       }
49     }
50   }
51   return result;
52 }

```

1.3.8. Palindrome Subsequence

```

1 public class palSubsequence {
2

```

```

3   public static void main(String[] args) {
4     solve();
5   }
6   public static void solve() {
7     for (int gap = 0; gap < n; gap++) {
8       for (int i = 0, j = gap; j < n; i++, j++) {
9         if (gap == 0) {
10           // single char is a palindrome
11           dp[i][j] = 1;
12         } else if (gap == 1) {
13           // if both char are same then 3 else 2
14           if (s.charAt(i) == s.charAt(j)) {
15             dp[i][j] = 3;
16           } else {
17             dp[i][j] = 2;
18           }
19         } else {
20           // the we have two cases
21           if (s.charAt(i) == s.charAt(j)) {
22             dp[i][j] = dp[i][j - 1] + dp[i + 1][j] +
23             1;
24           } else {
25             dp[i][j] = dp[i][j - 1] + dp[i + 1][j] -
26             dp[i + 1][j - 1];
27           }
28         }
29       }
30     }
31   }
32   // println(dp[0][n - 1]);
33 }

```

1.3.9. Longest Increasing Subsequence

```

1 import java.util.*;
2 public class lis {
3     public static void main(String[] args) {
4         // int[] arr = new int[n];
5         List<Long> dp = new ArrayList<>();
6         for (long x : arr) {
7             // Find the position to replace or extend
8             int pos = Collections.binarySearch(dp, x);
9             if (pos < 0)
10                 pos = -(pos + 1); // If not found, get
11                 ↑ insertion point
12             // If pos is within dp, replace the
13                 ↑ element
14             if (pos < dp.size()) {
15                 dp.set(pos, x);
16             } else {
17                 // Else, extend the subsequence
18                 dp.add(x);
19             }
20         }
21     }
22 }
23 
```

```

2     static int[] fTree;
3     public static void main(String[] args) {
4         // int[] arr = new int[n + 1]; // 1-based
5         // preProcess(arr);
6     } // 1-based indexing
7     static void preProcess(int[] arr) {
8         int n = arr.length - 1;
9         fTree = new int[n + 1];
10        for (int i = 1; i <= n; i++) {
11            update(i, arr[i]);
12        }
13    }
14    static int query(int l, int r) { return prefixSum(r)
15        - prefixSum(l - 1); }
16    static int prefixSum(int idx) {
17        int sum = 0;
18        while (idx > 0) {
19            sum += fTree[idx]; idx -= (idx & -idx);
20        }
21        return sum;
22    }
23    static void update(int idx, int delta) {
24        while (idx < fTree.length) {
25            fTree[idx] += delta; idx += (idx & -idx);
26        }
27    }

```

2. Data Structures

2.1. Fenwick Tree

```

1 public class FT { 
```

2.2. Segment Tree (SIMPLE)

```

1
2 public class SegTreeSimple { }
3 class SegmentTree { 
```

```

4  private int[] tree; private int n;
5  public SegmentTree(int[] arr) {
6      this.n = arr.length; this.tree = new int[4 * n];
7      build(arr, 0, 0, n - 1);
8  }
9  private void build(int[] arr,int node,int start,int
10     end) {
11      if (start == end) {
12          tree[node] = arr[start]; return;
13      }
14      int mid = (start + end) / 2;
15      build(arr, 2 * node + 1, start, mid);
16      build(arr, 2 * node + 2, mid + 1, end);
17      tree[node] = tree[2 * node + 1] + tree[2 * node +
18          2];
19  }
20  public void update(int index, int value) {
21      update(0, 0, n - 1, index, value);
22  }
23  private void update(int node,int st,int en,int
24      id,int val) {
25      if (st == en) {
26          tree[node] = val; return;
27      }
28      int mid = (st + en) / 2;
29      if (id <= mid) {
30          update(2 * node + 1, st, mid, id, val);
31      } else {
32          update(2 * node + 2, mid + 1, en, id, val);
33      }
34      tree[node] = tree[2 * node + 1] + tree[2 * node +
35          2];
36  }
37  public int query(int left, int right) {
38      return query(0, 0, n - 1, left, right);
39  }
40  private int KthOne(int node,int start,int end,int k)
41      {
42          if (start == end) return start;
43          int leftCount = tree[2 * node + 1];
44          if (k < leftCount) {
45              return KthOne(2*node+1,start,(start+end)/2,k);
46          } else {
47              return
48                  KthOne(2*node+2,(start+end)/2+1,end,k-leftCount);
49          }
50      }
51  public int findKthOne(int k) {
52      return KthOne(0, 0, n - 1, k);
53  }
54  private int query(int node,int start,int end,int
55      l,int r) {
56      if (r < start || l > end) return 0;// outside
57      if (l <= start && end <= r) return tree[node];//
58          inside
59      int mid = (start + end) / 2;
60      int leftSum = query(2 * node + 1, start, mid, l,
61          r);
62      int rightSum = query(2 * node + 2, mid + 1, end,
63          l, r);
64      return leftSum + rightSum;
65  }

```

```

2  private int n;
3  private long[] st;
4  private long[] lazy;
5  public void init(int _n) {
6      this.n = _n; st = new long[4 * n];
7      lazy = new long[4 * n];
8  }
9  private long combine(long a, long b) { return a + b;
10 }
11 private void push(int start, int end, int node) {
12     if (lazy[node] != 0) {
13         st[node] += (end - start + 1) * lazy[node];
14         if (start != end) {
15             lazy[2 * node + 1] += lazy[node];
16             lazy[2 * node + 2] += lazy[node];
17         } lazy[node] = 0;
18     }
19 }
20 private void build(int start,int end,int node,long[]
21   v) {
22     if (start == end)
23         st[node] = v[start]; return;
24     int mid = (start + end) / 2;
25     build(start, mid, 2 * node + 1, v);
26     build(mid + 1, end, 2 * node + 2, v);
27     st[node] = combine(st[2 * node + 1], st[2 * node +
28       2]);
29 }
30 private long query(int start,int end,int l,int r,int
31   node) {
32     push(start, end, node);
33     if (start > r || end < l) return 0;
34     if (start >= l && end <= r) return st[node];
35     int mid = (start + end) / 2;
36     long q1 = query(start, mid, l, r, 2 * node + 1);
37     long q2 = query(mid + 1, end, l, r, 2 * node + 2);
38     return combine(q1, q2);
39 }
40 private void update(int sta,int en,int node,int l,
41   int r,long val) {
42     push(sta, en, node);
43     if (sta > r || en < l) return;
44     if (sta >= l && en <= r) {
45         lazy[node] = val; push(sta, en, node); return;
46     }
47     int mid = (sta + en) / 2;
48     update(sta, mid, 2 * node + 1, l, r, val);
49     update(mid + 1, en, 2 * node + 2, l, r, val);
50     st[node] = combine(st[2 * node + 1], st[2 * node +
51       2]);
52 }
53 public void build(long[] v) { build(0, n - 1, 0, v);
54 }
55 public long query(int l, int r) { return query(0, n
56   - 1, l, r, 0); }
57 public void update(int l, int r, long x) { update(0,
58   n - 1, 0, l, r, x); }
59 }
60
61 // parent[node][i] = parent[parent[node][i - 1]][i -
62   1];
63 // This means that the  $2^i$  th parent of the node is

```

```

3 // 2i - 1 th parent of the node ka 2^i-1 th parent
4 public class BinaryLifting {
5     private static final int MAX_LOG = 20;
6     private static void solve() {
7         int[][] par = new int[n + 1][MAX_LOG];
8         dfs(1, 0, adj, par);
9     }
10    private static void dfs(int node, int parent,
11        List<List<Integer>> adj, int[][] par) {
12        par[node][0] = parent;
13        for (int j = 1; j < MAX_LOG; j++) {
14            par[node][j] = par[par[node][j - 1]][j - 1];
15        }
16        for (int adjNode : adj.get(node)) {
17            if (adjNode != parent)
18                dfs(adjNode, node, adj, par);
19        }
20    }
21    static int Kthparent(int node, int k, int[][] par) {
22        for (int i = MAX_LOG - 1; i >= 0; i--) {
23            if (((1 << i) & k) != 0) {
24                node = par[node][i];
25                if (node == 0) return 0;
26            }
27        }
28        return node;
29    }
30}

```

2.5. DSU

```

1 public class DSU {
2     private int[] parent, rank, size;

```

```

3     int component;
4     public DSU(int n) {
5         parent = new int[n]; rank = new int[n];
6         size = new int[n]; //
7         for (int i = 0; i < n; i++) {
8             parent[i] = i; size[i] = 1; //
9         } component = n;
10    }
11    public int find(int x) {
12        if (parent[x] != x) parent[x] = find(parent[x]);
13        return parent[x];
14    }
15    public boolean union(int u, int v) {
16        int rootU = find(u), rootV = find(v);
17        if (rootU == rootV)
18            return false;
19        component--;
20        if (rank[rootU] > rank[rootV]) {
21            parent[rootV] = rootU;
22            size[rootU] += size[rootV]; //
23        } else if (rank[rootU] < rank[rootV]) {
24            parent[rootU] = rootV;
25            size[rootV] += size[rootU]; //
26        } else {
27            parent[rootV] = rootU; rank[rootU]++;
28            size[rootU] += size[rootV]; //
29        }
30        return true;
31    }
32    public int getComp() { return component; }
33    public int getSize(int x) { return size[find(x)]; }
34}

```

35

2.6. SparseTable

```

1 public class SparseTable {
2     int[][] st;
3     int[] log;
4     public SparseTable(int[] arr) {
5         int n = arr.length;
6         int K = 32 - Integer.numberOfLeadingZeros(n);
7         st = new int[n][K];
8         log = new int[n + 1];
9         log[1] = 0;
10        for (int i = 2; i <= n; i++) {
11            log[i] = log[i / 2] + 1;
12        }
13        for (int i = 0; i < n; i++) {
14            st[i][0] = arr[i];
15        }
16        for (int j = 1; j < K; j++) {
17            for (int i = 0; i + (1 << j) <= n; i++) {
18                st[i][j] = Math.min(st[i][j - 1],
19                    st[i + (1 << (j - 1))][j - 1]);
20            }
21        }
22    }
23    public int query(int l, int r) {
24        int len = r - l + 1;
25        int j = log[len];
26        return Math.min(st[l][j], st[r - (1 << j) +
27            1][j]);
28    }
29 }
```

2.7. EulerTour

```

1 public class euler_tour {
2     static List<Integer>[] adj;
3     static int time = 0;
4     public static void main(String[] args)
5         throws IOException {solve();}
6     static void solve() {
7         long[] euler = new long[2 * n];
8         int[] inTime = new int[n + 1];
9         int[] outTime = new int[n + 1];
10        dfs(1, -1, inTime, outTime);
11        for (int i = 1; i <= n; i++) {
12            euler[inTime[i]] = v[i - 1];
13            euler[outTime[i]] = -v[i - 1];
14        }
15        SegTree seg = new SegTree(); seg.init(2 * n);
16        seg.build(0, 2 * n - 1, 0, euler);
17        while (q-- > 0) {
18            int type = in.nextInt();
19            if (type == 1) {
20                int s = in.nextInt(), x = in.nextLong();
21                seg.update(0, 2 * n - 1, inTime[s], 0, x);
22                seg.update(0, 2 * n - 1, outTime[s], 0, -x);
23            } else {
24                int s = in.nextInt();
25                out.println(seg.query(0, 2 * n - 1, 0,
26                    inTime[s], 0));
27            }
28        }
29    }
30    private static void dfs(int node, int parent, int[]
31        inTime, int[] outTime) {
```

```

30     inTime[node] = time++;
31     for (int adjNode : adj[node]) {
32         if (adjNode != parent)
33             dfs(adjNode, node, inTime, outTime);
34     }
35     outTime[node] = time++;
36 }
37 }
```

3. Math

3.1. Number Theory

3.1.1. Linear Sieve

```

1 public class prime_sieve {
2     static final int MAXN = 1_000_000;
3     static boolean[] isPrime = new boolean[MAXN];
4     public static void main(String[] args) { }
5     static void sieve() {
6         Arrays.fill(isPrime, true);
7         isPrime[0] = false;
8         isPrime[1] = false;
9         for (int i = 2; (long) i * i < MAXN; i++) {
10             if (isPrime[i]) {
11                 for (int j = i * i; j < MAXN; j += i)
12                     isPrime[j] = false;
13             }
14         }
15     }
16 }
```

3.1.2. Get Factors and SPF Fucn

```

1 import java.util.*;
2
3 public class allfactor {
4     public static void main(String[] args) { }
5     static int N = 100000;
6     static int[] spf = new int[N + 1];
// store the smallest prime factor of i in
→ spf[i].
7     static void spf() {
8         for (int i = 2; i <= N; i++) {
9             spf[i] = i;
10        }
// Sieve of Eratosthenes modified to find
→ smallest prime factor
11        for (int i = 2; i * i <= N; i++) {
12            if (spf[i] == i) { // If i is prime
13                for (int j = i * i; j <= N; j += i) {
14                    if (spf[j] == j)
// Mark spf[j] with the
→ smallest prime factor
15                        spf[j] = i;
16                }
17            }
18        }
19    }
20
21    static List<Integer> allFactors(int n) {
22        List<Integer> fac = new ArrayList<>();
23        fac.add(1);
24        while (n > 1) {
25            int p = spf[n];
26            List<Integer> cur = new ArrayList<>();
27            cur.add(p);
28            n /= p;
29            fac.addAll(cur);
30        }
31    }
32 }
```

```

29         cur.add(1);
30         while (n % p == 0) {
31             n /= p;
32             cur.add(cur.size() - 1) * p);
33         }
34         List<Integer> next = new ArrayList<>();
35         for (int x : fac)
36             for (int y : cur)
37                 next.add(x * y);
38         fac = next;
39     }
40     return fac;
41 }
42 }
```

3.2. Combinatorics

3.2.1. Comb

```

1 public class Main {
2     static final long MOD = 998244353L;
3     static final long INF = (long) 1e18;
4     static class Combinatorics {
5         final int MOD;
6         long[] fact, invFact;
7         public Combinatorics(int maxN, int mod) {
8             this.MOD = mod;
9             fact = new long[maxN + 1];
10            invFact = new long[maxN + 1];
11            precompute(maxN);
12        }
13        void precompute(int maxN) {
14            fact[0] = 1;
15            for (int i = 1; i <= maxN; i++) {
```

```

16                fact[i] = (i * fact[i - 1]) % MOD;
17            }
18            invFact[maxN] = modPow(fact[maxN], MOD - 2); // 
19            // Fermats little theorem
20            for (int i = maxN - 1; i >= 0; i--) {
21                invFact[i] = (invFact[i + 1] * (i + 1)) % MOD;
22            }
23            // NCK : no of ways to choose the k elements
24            // from n distinct element without caring order.
25            long nCk(int n, int k) {
26                if (k > n || k < 0)
27                    return 0;
28                return (((fact[n] * invFact[k]) % MOD) *
29                    invFact[n - k]) % MOD;
30            }
31            // NPK : no. of ways to arrange k elements out of
32            // n,
33            // where order matters
34            long nPk(int n, int k) {
35                if (k > n || k < 0)
36                    return 0;
37                return (fact[n] * invFact[n - k]) % MOD;
38            }
39            long factorial(int n) {
40                return fact[n];
41            }
42            // stars and bars formula C (n + k - 1, n) --> no.
43            // of ways to distribute n
44            // identical stars into k bins
45            long starsAndBars(int n_stars, int k_bins) {
46                if (n_stars == 0)
47                    return 1;
```

```

44     return 1;
45     if (k_bins == 0)
46         return 0;
47     return nCk(n_stars + k_bins - 1, n_stars);
48 }
49 long modPow(long a, long b) {
50     long res = 1;
51     while (b > 0) {
52         if ((b & 1) == 1)
53             res = (res * a) % MOD;
54         b >>= 1;
55         a = (a * a) % MOD;
56     }
57     return res;
58 }
59 }
60 }
```

4. UTILITY

```

1 public class Utility {
2     public static void main(String[] args) {    }
3     { int[][] prefix = new int[n + 2][m + 2];
4         for (int i = 1; i <= n; i++) {
5             for (int j = 1; j <= m; j++) {
6                 int g = (s[i - 1][j - 1] == 1) ? 1 : 0;
7                 prefix[i][j] = prefix[i - 1][j] +
8                     prefix[i][j - 1] - prefix[i - 1][j - 1] + g;
9             }
10        }
11        int totalG = 0;
12        for (int i = 0; i < n; i++) {
13            for (int j = 0; j < m; j++) {
```

```

14             totalG += (s[i][j] == 1) ? 1 : 0;    }
15     }
16     for (int i = 0; i < n; i++) {
17         for (int j = 0; j < m; j++) {
18             // checking for the sum of 2k * 2k grid.
19             int r1 = Math.max(0, i - k + 1); // top row
20             int r2 = Math.min(n, i + k); // bottom row
21             // (exclusive)
22             int c1 = Math.max(0, j - k + 1); // left col
23             int c2 = Math.min(m, j + k); // right col
24             // (exclusive)
25             // Number of 1s in the rec. (r1, c1) to
26             // (r2-1, c2-1)
27             int count = prefix[r2][c2] -
28                 prefix[r2][c1] - prefix[r1][c2] +
29                 prefix[r1][c1];
30         }
31     }
32     static class Pair {
33         int first, second;
34         Pair(int first, int second) {
35             this.first = first;
36             this.second = second;
37         }
38         @Override
39         public boolean equals(Object obj) {
40             if (obj == this)
41                 return true;
42             if (!(obj instanceof Pair))
43                 return false;
44             Pair pair = (Pair) obj;
```

```

42     return pair.first == this.first && pair.second
43     ↵   == this.second;
44 }
45 @Override
46 public int hashCode() {
47     return Objects.hash(first, second);
48 }
49 // Returns min. swaps required to sort arr[] in asc
50 ↵ order
51 static int minSwaps(int[] arr) {
52     int n = arr.length;
53     int[][] paired = new int[n][2];
54     for (int i = 0; i < n; i++) {
55         paired[i][0] = arr[i]; paired[i][1] = i;
56     }
57     Arrays.sort(paired, (a, b) ->
58     ↵     Integer.compare(a[0], b[0]));
59     boolean[] visited = new boolean[n];
60     int swaps = 0;
61     for (int i = 0; i < n; i++) {
62         if (visited[i] || paired[i][1] == i) continue;
63         int cycleSize = 0, j = i;
64         while (!visited[j]) {
65             visited[j] = true;
66             j = paired[j][1]; cycleSize++;
67         }
68         if (cycleSize > 1) swaps += (cycleSize - 1);
69     }
70     return swaps;
71 }
72 private static long maxSubarraySum(long[] a, int
73     ↵ left, int right) {
74     long curr = 0, maxSum = 0;
75     for (int i = left; i <= right; i++) {
76         curr += a[i];
77         maxSum = Math.max(maxSum, curr);
78         if (curr < 0) curr = 0;
79     }
80     return maxSum;
81 }
82 private static long minSubarraySum(long[] a, int
83     ↵ left, int right) {
84     long curr = 0, maxSum = 0;
85     for (int i = left; i <= right; i++) {
86         curr -= a[i];
87         maxSum = Math.max(maxSum, curr);
88         if (curr < 0) curr = 0;
89     }
90     return -maxSum;
91 }
92 static long modInverse(long a, long mod) {
93     return modPow(a, mod - 2, mod);
94 }
95 static long modDiv(long x, long y, long mod) {
96     // x * y^(MOD-2) % MOD
97     return (x * modPow(y, mod - 2, mod)) % mod;
98 }
99 static long modPow(long base, long exp, long mod) {
100    long result = 1;
101    base = base % mod;
102    while (exp > 0) {
103        if ((exp & 1) == 1)
104            result = (result * base) % mod;
105        base = (base * base) % mod;
106    }
107 }
```

```

102     exp >>= 1;
103 }
104 return result;
105 }
106 static long modMul(long a, long b, long mod) {
107     long result = 0;
108     a %= mod;
109     b %= mod;
110     while (b > 0) {
111         if ((b & 1) == 1) {
112             result = (result + a) % mod;
113         }
114         a = (a << 1) % mod; // a = (a * 2) % mod
115         b >>= 1; // b = b / 2
116     }
117     return result;
118 }
119 static void derangement() {
120     int k = 4;
121     int[] derangements = new int[k + 1];
122     derangements[0] = 1; // D(0) =
123     if (k > 0)
124         derangements[1] = 0; // D(1) =
125     for (int i = 2; i <= k; i++) {
126         derangements[i] = (i - 1) *
127             (derangements[i - 1] + derangements[i - 2]);
128     }
129 }
130 private static void addAllPrimFact(int x,
131     → HashMap<Integer, Integer> map) {
132     int i = 2;
133     while (i * i <= x) {

```

```

133         while (x % i == 0) {
134             map.put(i, map.getOrDefault(i, 0) + 1);
135             x /= i;
136         }
137         i++;
138     }
139     if (x > 1) map.put(x, map.getOrDefault(x, 0) + 1);
140 } // Find primes in range
141 public static List<Boolean> segmentedSieve(long L,
142     → long R) {
143     long lim = (long) Math.sqrt(R);
144     boolean[] mark = new boolean[(int) (lim + 1)];
145     List<Long> primes = new ArrayList<>();
146     for (long i = 2; i <= lim; i++) {
147         if (!mark[(int) i]) {
148             primes.add(i);
149             for (long j = i * i; j <= lim; j += i) {
150                 mark[(int) j] = true;
151             }
152         }
153     }
154     List<Boolean> isPrime = new ArrayList<>();
155     for (int i = 0; i <= R - L; i++) {
156         isPrime.add(true);
157     }
158     for (long prime : primes) {
159         long start = Math.max(prime * prime, (L + prime
160             → - 1) / prime * prime);
161         for (long j = start; j <= R; j += prime) {
162             isPrime.set((int) (j - L), false);
163         }
164     }

```

```

163     if (L == 1)
164         isPrime.set(0, false);
165     return isPrime;
166 }
167 // int bit = (num >> i) & 1;
168 int flipBit(int n, int j) {
169     return n ^ (1 << j);
170 }
171 // mex calculate for the arr of permutation
172 // long mex = (n * (n + 1) / 2) - sum;
173 public static int findMSB(long n) {
174     int msb = 0;
175     while (n > 1) {
176         n >>= 1;
177         msb++;
178     }
179     return 1 << msb;
180 }
181 public static long gcd(long a, long b) {
182     if (a == 0)
183         return b;
184     return gcd(b % a, a);
185 }
186 public static void factor(long n) {
187     long count = 0;
188     for (int i = 1; i * i <= n; i++) {
189         if (n % i == 0) {
190             // i -> is the one factor
191             count++;
192             if (i != n / i) {
193                 // n / i -> is the other factor
194                 count++;
195             }
196         }
197     }
198 }
199 public static long MahantaDist(long x1, long y1,
200                                long x2, long y2) {
201     return Math.abs(x1 - x2) + Math.abs(y1 - y2);
202 }
203 public static long numberOfDivisors(long num) {
204     long total = 1;
205     for (long i = 2; i * i <= num; i++) {
206         if (num % i == 0) {
207             int e = 0;
208             while (num % i == 0) {
209                 e++;
210                 num /= i;
211             }
212             total *= (e + 1);
213         }
214     }
215     if (num > 1) {
216         total *= 2;
217     }
218     return total;
219 }
220 public static long sumOfDivisors(long num) {
221     long total = 1;
222     for (long i = 2; i * i <= num; i++) {
223         if (num % i == 0) {
224             int e = 0;
225             while (num % i == 0) {
226                 e++;
227                 num /= i;
228             }
229             total += (i * (Math.pow(i, e) - 1)) / (i - 1);
230         }
231     }
232     return total;
233 }

```

```

227     }
228     long sum = 0, pow = 1;
229     while (e-- >= 0) {
230         sum += pow;
231         pow *= i;
232     }
233     total *= sum;
234 }
235 if (num > 1) {
236     total *= (1 + num);
237 }
238 return total;
239 }
240 public static long lcm(long a, long b) {
241     return Math.abs(a * b) / gcd(a, b);
242 }
// This is used when we use Pair inside the map
244 Map<Pair, Integer> map = new HashMap<>();
245 static class Pair {
246     long first, second;
247     Pair(long first, long second) {
248         this.first = first;
249         this.second = second;
250     }
251     @Override
252     public boolean equals(Object o) {
253         if (this == o)
254             return true;
255         if (o == null || getClass() != o.getClass())
256             return false;
257         Pair pair = (Pair) o;
258     }

```

```

259         return first == pair.first && second ==
260             pair.second;
261     }
262     @Override
263     public int hashCode() {
264         return (int) (31 * first + second);
265     }
266     static int[] dijkstra(List<List<Pair>> graph, int
267         → src, int n) {
268         PriorityQueue<Pair> pq = new PriorityQueue<>();
269         int[] dist = new int[n];
270         Arrays.fill(dist, Integer.MAX_VALUE);
271         dist[src] = 0;
272         pq.add(new Pair(src, 0));
273         while (!pq.isEmpty()) {
274             Pair p = pq.poll();
275             int u = p.node;
276             if (p.weight > dist[u]) continue;
277             for (Pair neighbor : graph.get(u)) {
278                 int v = neighbor.node;
279                 int weight = neighbor.weight;
280                 if (dist[u] + weight < dist[v]) {
281                     dist[v] = dist[u] + weight;
282                     pq.add(new Pair(v, dist[v]));
283                 }
284             }
285         }
286         return dist;
287     }
288     public static int[] bellmanFord(int n, int[][] []
289         → edges, int src) {

```

```

288     int[] dist = new int[n + 1];
289     Arrays.fill(dist, (int) 1e9);
290     dist[src] = 0;
291     // Relax all edges (n - 1) times
292     for (int i = 1; i <= n - 1; i++) {
293         boolean any = false;
294         for (int[] edge : edges) {
295             int u = edge[0], v = edge[1], wt = edge[2];
296             if (dist[u] != (int) 1e9 && dist[u] + wt <
297                 → dist[v]) {
298                 dist[v] = dist[u] + wt;
299                 any = true;
300             }
301             if (!any) break;
302             if (i == n - 1) return new int[] {};
303         }
304         return dist;
305     }
306     // static final int INF = 1_000_000_000;
307     static void floydWarshall(int[][] dist, int n) {
308         for (int k = 0; k < n; k++) {
309             for (int i = 0; i < n; i++) {
310                 for (int j = 0; j < n; j++) {
311                     if (dist[i][k] < INF && dist[k][j] < INF)
312                         dist[i][j] = Math.min(dist[i][j],
313                                     → dist[i][k] + dist[k][j]);
314                 }
315             }
316             for (int i = 0; i < n; i++) {
317                 if (dist[i][i] < 0) {
318                     // negative cycle
319                 }
320             }
321         }
322     }
323     // TOPOSORT and all that stuff toposort + cycle
324     → detection
325     public static boolean dfs(int node, int[] used,
326                               List<List<Integer>> adj, List<Integer> ans) {
327         used[node] = 1; // in recursion stack
328         for (int adjNode : adj.get(node)) {
329             if (used[adjNode] == 1) {
330                 return false; // detected a cycle
331             } else if (used[adjNode] == 0) {
332                 // not visited
333                 if (!dfs(adjNode, used, adj, ans))
334                     return false;
335             }
336         }
337         used[node] = 2; // visited but out of stack
338         ans.add(node);
339         return true;
340     }
341     // DFS cycle detection (Recomended)
342     public static boolean dfsCycleDG(int node,
343                                     List<List<Integer>> adj,
344                                     boolean[] visited, boolean[] onStack) {
345         visited[node] = true;
346         onStack[node] = true;
347         for (int neighbor : adj.get(node)) {
348             if (!visited[neighbor]) {
349                 if (dfsCycleDG(neighbor, adj, visited,
350                               → onStack))

```

```

346         return true;
347     } else if (onStack[neighbor]) {
348         return true; // Cycle detected
349     }
350     onStack[node] = false;
351     return false;
352 }
// BFS Cycle Detection (Kahn's Algorithm)
354 public static boolean hasCycle(int n,
355     ↪ List<List<Integer>> adj) {
356     int[] inDegree = new int[n];
357     for (int u = 0; u < n; u++) {
358         for (int v : adj.get(u))
359             inDegree[v]++;
360     }
361     Queue<Integer> q = new LinkedList<>();
362     for (int i = 0; i < n; i++) {
363         if (inDegree[i] == 0)
364             q.add(i);
365     }
366     int count = 0;
367     while (!q.isEmpty()) {
368         int u = q.poll();
369         count++;
370         for (int v : adj.get(u)) {
371             if (--inDegree[v] == 0)
372                 q.add(v);
373         }
374     } return count != n; // If count < n, there is a
375     ↪ cycle
376 }

377 // DFS-Based Topological Sort
378 public static List<Integer> topoSortDfs(int n,
379     ↪ List<List<Integer>> adj) {
380     boolean[] visited = new boolean[n];
381     List<Integer> topo = new ArrayList<>();
382     for (int i = 0; i < n; i++) {
383         if (!visited[i])
384             dfsTopo(i, adj, visited, topo);
385     }
386     Collections.reverse(topo);
387     return topo;
388 }
389 public static void dfsTopo(int node,
390     ↪ List<List<Integer>> adj,
391     boolean[] visited, List<Integer> topo) {
392     visited[node] = true;
393     for (int neighbor : adj.get(node)) {
394         if (!visited[neighbor])
395             dfsTopo(neighbor, adj, visited, topo);
396     }
397     topo.add(node);
398 }
399 public static List<Integer> topoSortBFS(int n,
400     ↪ List<List<Integer>> adj) {
401     int[] inDegree = new int[n];
402     for (int u = 0; u < n; u++) {
403         for (int v : adj.get(u)) {
404             inDegree[v]++;
405         }
406     }
407     Queue<Integer> q = new LinkedList<>();
408     for (int i = 0; i < n; i++) {
409

```

```

405     if (inDegree[i] == 0) q.add(i);
406 }
407 List<Integer> topo = new ArrayList<>();
408 while (!q.isEmpty()) {
409     int u = q.poll();
410     topo.add(u);
411     for (int v : adj.get(u)) {
412         if (--inDegree[v] == 0) q.add(v);
413     }
414 }
415 return topo.size() == n ? topo : new
416 → ArrayList<>();
417 } // MST using DSU (Krushkal ALgorythm)
418 int n;// Nodes int m; // Edges
419 Edge[] edges = new Edge[m];
420 for (int i = 0; i < m; i++) {
421     int u = in.nextInt();
422     int v = in.nextInt();
423     int w = in.nextInt();
424     edges[i] = new Edge(u, v, w);
425 }
426 Arrays.sort(edges); // Sort edges by weight
427 DSU dsu = new DSU(n);
428 long mstWeight = 0;
429 ArrayList<Edge> mstEdges = new ArrayList<>();
430 for (Edge e : edges) {
431     if (dsu.union(e.u, e.v)) { // If u and v are in
432         → different sets
433         mstWeight += e.w;
434         mstEdges.add(e);
435     }
436 }
437 static class Edge implements Comparable<Edge> {
438     int u, v, w;
439     Edge(int u, int v, int w) {
440         this.u = u; this.v = v; this.w = w;
441     }
442     public int compareTo(Edge o) {
443         return Integer.compare(this.w, o.w);
444     }
445 } // MST using PriorityQueue Prims Algorythm
446 static long primsMST(int n, List<List<int[]>> adj) {
447     boolean[] visited = new boolean[n + 1];
448     PriorityQueue<int[]> pq = new PriorityQueue<>((x, y)
449     → -> (x[1] - y[1]));
450     pq.add(new int[] { 1, 0 }); // Start from node 1
451     long mstWeight = 0;
452     while (!pq.isEmpty()) {
453         int[] curr = pq.poll();
454         int u = curr[0], w = curr[1];
455         if (visited[u])
456             continue;
457         visited[u] = true;
458         mstWeight += w;
459         for (int[] v : adj.get(u)) {
460             if (!visited[v[0]]) {
461                 pq.add(new int[] { v[0], v[1] });
462             }
463         }
464     }
465     return mstWeight;
466 }

```