KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

Narayanguda, Hyderabad – 500029

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL
COMPETETETIVE PROGRAMMING LAB

B.Tech III YEAR II SEM (RKR21 Regulations)

ACADEMIC YEAR 2024-25

INDEX

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

Department of Computer Science and Engineering

Vision of the Institution:

- To be the fountain head in producing highly skilled, globally competent engineers.
- Producing quality graduates trained in the latest software technologies and related tools and striving to make India a world leader in software products and services.

Mission of the Institution:

- To provide a learning environment that inculcates problem solving skills, professional, ethical responsibilities, lifelong learning through multi modal platforms and prepare students to become successful professionals.
- To establish Industry Institute Interaction to make students ready for the industry.
- To provide exposure to students on latest hardware and software tools.
- To promote research based projects/activities in the emerging areas of technology convergence.
- To encourage and enable students to not merely seek jobs from the industry but also to create new enterprises
- To induce a spirit of nationalism which will enable the student to develop, understand India's challenges and to encourage them to develop effective solutions.
- To support the faculty to accelerate their learning curve to deliver excellent service to students

Department of Computer Science and Engineering

Vision of the Department:

To be among the region's premier teaching and research Computer Science and Engineering departments producing globally competent and socially responsible graduates in the most conducive academic environment.

Mission of the Department:

- To provide faculty with state of the art facilities for continuous professional development and research, both in foundational aspects and of relevance to emerging computing trends.
- To impart skills that transform students to develop technical solutions for societal needs and inculcate entrepreneurial talents.
- To inculcate an ability in students to pursue the advancement of knowledge in various specializations of Computer Science and Engineering and make them industry-ready.
- To engage in collaborative research with academia and industry and generate adequate resources for research activities for seamless transfer of knowledge resulting in sponsored projects and consultancy.
- To cultivate responsibility through sharing of knowledge and innovative computing solutions that benefit the society-at-large.
- To collaborate with academia, industry and community to set high standards in academic excellence and in fulfilling societal responsibilities.

Department of Computer Science and Engineering

PROGRAM OUTCOMES (POs)

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Department of Computer Science and Engineering

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: An ability to analyze the common business functions to design and develop appropriate Computer Science solutions for social upliftments.

PSO2: Shall have expertise on the evolving technologies like Python, Machine Learning, Deep Learning, Internet of Things (IOT), Data Science, Full stack development, Social Networks, Cyber Security, Big Data, Mobile Apps, CRM, ERP eetc.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will have successful careers in computer related engineering fields or will be able to successfully pursue advanced higher education degrees.

PEO2: Graduates will try and provide solutions to challenging problems in their profession by applying computer engineering principles.

PEO3: Graduates will engage in life-long learning and professional development by rapidly adapting changing work environment.

PEO4: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

Department of Computer Science and Engineering

Course Outcomes and CO-PO/PSO Mapping

At the end of the course a student will be able Course Outcomes (COs):

	Course Outcome(CO)
CO1	Design and implement solutions for arrays.
CO2	Design and implement solutions for different trees.
CO3	Design solutions for graph applications.
CO4	Design solutions for compression techniques.
CO5	Design solutions for disjoint set applications.

CO-PO-PSO MAPPING:

CP LAB	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	H	H	H	H			M					L	H	M
CO2	H	H	H	H			M					L	M	M
CO3	H	H	H	H			M					M	M	M
CO4	H	H	H	H			M	M				L	M	M
CO5	H	H	H	H			M					L	M	M

Experiment-1

a) Write a JAVA Program to find Subarrays with K Different integers

Given an array A of positive integers, call a (contiguous, not necessarily distinct) subarray of A good if the number of different integers in that subarray is exactly K. (For example, [1,2,3,1,2] has 3 different integers: 1, 2, and 3.)

Return the number of good subarrays of A.

Example 1:

Input: A = [1,2,1,2,3], K = 2

Output: 7

Explanation: Subarrays formed with exactly 2 different integers: [1,2], [2,1], [1,2], [2,3], [1,2,1], [2,1,2], [1,2,1,2]

Example 2:

Input: A = [1,2,1,3,4], K = 3

Output: 3

Explanation: Subarrays formed with exactly 3 different integers: [1,2,1,3], [2,1,3],

[1,3,4]. **Note:**

1 <= A.length <= 20000

1 <= A[i] <= A.length

1 <= K <= A.length

PROGRAM

```
import java.util.*;
class Solution {
  public int subarraysWithKDistinct(int[] A, int K) {
    return atMostK(A, K) - atMostK(A, K - 1);
  }
  int atMostK(int[] A, int K) {
    int i = 0, res = 0;
    Map < Integer, Integer > count = new HashMap < > ();
    for (int j = 0; j < A.length; ++j) {
      if (count.getOrDefault(A[j], 0) == 0) K--;
      count.put(A[j], count.getOrDefault(A[j], 0) + 1);
```

```
      while (K < 0) {
        count.put(A[i], count.get(A[i]) - 1);
        if (count.get(A[i]) == 0) K++;
        i++;
      }
      res += j - i + 1;
    }
    return res;
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int k = sc.nextInt();
    int arr[] = new int[n];
    for (int i = 0; i < n; i++)
      arr[i] = sc.nextInt();
    System.out.println(new Solution().subarraysWithKDistinct(arr, k));
  }
}
```

b) Write a JAVA Program to find shortest sub array with sum at least K

Return the length of the shortest, non-empty, contiguous subarray of A with sum at least K. If there is no non-empty subarray with sum at least K, return -1.

Example 1:

Input: A = [1], K = 1

Output: 1

Example 2:

Input: A = [1,2], K = 4

Output: -1

Example 3:

Input: A = [2,-1,2], K = 3

Output: 3

Note:

1 <= A.length <= 50000

-10 ^ 5 <= A[i] <= 10 ^ 5

1 <= K <= 10 ^ 9

PROGRAM

```
import java.util.*;
class ShortestSubarray {
  public int shortestSubarray(int[] A, int K) {
    int N = A.length;
    long[] P = new long[N + 1];
    for (int i = 0; i < N; ++i)
      P[i + 1] = P[i] + (long) A[i];

    // Want smallest y-x with P[y] - P[x] >= K
    int ans = N + 1; // N+1 is impossible
    Deque < Integer > monoq = new LinkedList(); //opt(y) candidates, as indices of P
    for (int y = 0; y < P.length; ++y) {
      // Want opt(y) = largest x with P[x] <= P[y] - K;
```

```
      while (!monoq.isEmpty() && P[y] <= P[monoq.getLast()])
        monoq.removeLast();
      while (!monoq.isEmpty() && P[y] >= P[monoq.getFirst()] + K)
      ans = Math.min(ans, y - monoq.removeFirst());
      monoq.addLast(y);
    }
    return ans < N + 1 ? ans : -1;
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int k = sc.nextInt();
    int arr[] = new int[n];
    for (int i = 0; i < n; i++)
      arr[i] = sc.nextInt();
    System.out.println(new ShortestSubarray().shortestSubarray(arr, k));
  }
}
```

c) Write a JAVA Program to implement Fenwick Tree

Malika taught a new fun time program practice for Engineering Students. As a part of this she has given set of numbers, and asked the students to find the gross value of numbers between indices S1 and S2 (S1<=S2), inclusive. Now it's your task to create a method sumRange(S1,S2) which return the sum of numbers between the indices S1 and S2, both are inclusive.

Input Format:

Line-1: An integer n, size of the array(set of numbers).

Line-2: n space separated integers.

Line-3: Two integers s1 and s2, index positions where s1<=s2.

Output Format:

An integer, sum of integers between indices(s1, s2).

Sample Input-1:

8

1 2 13 4 25 16 17 8

2 6

Sample Output-1:

75

Constraints:

1 <= nums.length <= 3 * 104

-100 <= nums[i] <= 100

0 <= index < nums.length

-100 <= val <= 100

0 <= left <= right < nums.length

At most 3 * 104 calls will be made to update and sumRange.

PROGRAM

```
import java.util.*;
class NumArray {
int[] nums;
  int[] BIT;
```

```
int n;
public NumArray(int[] nums) {
  this.nums = nums;
  n = nums.length;
  BIT = new int[n + 1];
  for (int i = 0; i < n; i++)
  init(i, nums[i]);

}
public void init(int i, int val) {
  i++;
  while (i <= n) {
    BIT[i] += val;
    i += (i & -i);
  }
}
public int getSum(int i) {
  int sum = 0;
  i++;
  while (i > 0) {
    sum += BIT[i];
    i -= (i & -i);
  }
  return sum;
}
public int sumRange(int i, int j) {
  return getSum(j) - getSum(i - 1);
}

public static void main(String args[]) {
  Scanner scan = new Scanner(System.in);
  int n = scan.nextInt();
  int[] nums = new int[n];
  for (int i = 0; i < n; i++) {
```

```
      nums[i] = scan.nextInt();
    }
    NumArray na = new NumArray(nums);
    // call sumrange(s1,s2)
    int s1 = scan.nextInt();
    int s2 = scan.nextInt();
    System.out.println(na.sumRange(s1, s2));

  }
}
```

d) Write a JAVA Program to implement a segment tree with its operations In Hyderabad

after a long pandemic gap, the Telangana Youth festival Is Organized at HITEX.

In HITEX, there are a lot of programs planned. During the festival in order to maintain the rules of Pandemic, they put a constraint that one person can only attend any one of the programs in one day according to planned days. Now it's your aim to implement the "Solution" class in such a way that you need to return the maximum number of programs you can attend according to given constraints.

Explanation: You have a list of programs 'p' and days 'd', where you can attend only one program on one day. Programs [p] = [first day, last day], p is the program's first day and the last day.

Input Format:

Line-1: An integer N, number of programs.

Line-2: N comma separated pairs, each pair(f_day, l_day) is separated by

space. **Output Format:**

An integer, the maximum number of programs you can attend.

Sample Input-1:

4

1 2,2 4,2 3,2 2

Sample Output-1:

4

Sample Input-2:

6

1 5,2 3,2 4,2 2,3 4,3 5

Sample Output-2:

5

PROGRAM

```
import java.util.*;
import java.io.*;
// Segment Tree
class Solution {
  class SegmentTreeNode {
```

```
  int start, end;
  SegmentTreeNode left, right;
  int val;

  public SegmentTreeNode(int start, int end) {
    this.start = start;
    this.end = end;
    left = null;
    right = null;
    val = 0;
  }
}
SegmentTreeNode root;
public int maxEvents(int[][] events) {
  if (events == null || events.length == 0) return 0;
  Arrays.sort(events, (a, b) -> {
    if (a[1] == b[1])
      return a[0] - b[0];
    return a[1] - b[1];
  });

  int lastDay = events[events.length - 1][1];
  int firstDay = Integer.MAX_VALUE;
  for (int i = 0; i < events.length; i++) {
    firstDay = Math.min(firstDay, events[i][0]);
  }
  root = buildSegmentTree(firstDay, lastDay);
  int count = 0;
  for (int[] event: events) {
    int earliestDay = query(root, event[0], event[1]);
    if (earliestDay != Integer.MAX_VALUE) {
    count++;
      update(root, earliestDay);
    }
```

```
  }
  return count;
}
private SegmentTreeNode buildSegmentTree(int start, int end) {
  if (start > end)
    return null;
  SegmentTreeNode node = new SegmentTreeNode(start, end);
  node.val = start;
  if (start != end) {
    int mid = start + (end - start) / 2;
    node.left = buildSegmentTree(start, mid);
    node.right = buildSegmentTree(mid + 1, end);
  }
  return node;
}
private void update(SegmentTreeNode curr, int lastDay) {
  if (curr.start == curr.end) {
    curr.val = Integer.MAX_VALUE;
  } else {
    int mid = curr.start + (curr.end - curr.start) / 2;
    if (mid >= lastDay) {
      update(curr.left, lastDay);
    } else {
      update(curr.right, lastDay);
    }
    curr.val = Math.min(curr.left.val, curr.right.val);
  }
}
private int query(SegmentTreeNode curr, int left, int right) {
  if (curr.start == left && curr.end == right) {
    return curr.val;
  }
  int mid = curr.start + (curr.end - curr.start) / 2;
  if (mid >= right) {
```

```
      return query(curr.left, left, right);
    } else if (mid < left) {
      return query(curr.right, left, right);
    } else
      return Math.min(query(curr.left, left, mid), query(curr.right, mid + 1, right));
  }
}
public class MaxEvents {
  public static void main(String args[]) throws IOException {
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    scan.nextLine();
    String str[] = scan.nextLine().split(",");
    int nums[][] = new int[n][2]; // declaring 10000 records to read from the input.txt file
    for (int i = 0; i < n; i++) {
      String val[] = str[i].split(" ");
      nums[i][0] = Integer.parseInt(val[0]);
      nums[i][1] = Integer.parseInt(val[1]);
    } // reading input into nums array

    Solution fna = new Solution(); // Fenwick Tree Approch Class Object Creation
    long result = fna.maxEvents(nums); // call Fenwick sumRange()
    System.out.println(result);
  }
}
```

e) Write a JAVA Program to implement TREAP with its operations

Given an integer array nums, return the number of reverse pairs in the array. A reverse pair is a pair (i, j) where 0 <= i < j < nums.length and nums[i] > 2 * nums[j].

Example 1:

Input: nums = [1,3,2,3,1]

Output: 2

Example 2:

Input: nums = [2,4,3,5,1]

Output: 3

Constraints:

1 <= nums.length <= 5 * 104

-2^31 <= nums[i] <= 2^31 – 1

PROGRAM

```
import java.util.*;
class Solution {
  class Pair < U, V > {
    U left;V right;
    Pair(U _left, V _right) {
      left = _left;
      right = _right;
    }
  }
  class Item {
    Double key;
    Double priority;
    long cnt;
    Item left, right;
    Item() {
      left = right = null;
      key = null;
```

```
    priority = null;
    cnt = 1L;
  }
}
long cnt(Item item) {
  if (item == null) return 0;
  return item.cnt;
}
void updateCnt(Item item) {
  if (item != null)
    item.cnt = 1 + cnt(item.left) + cnt(item.right);
}
Item[] split(Item item, double key) {
  Item[] ret = null;
  if (item == null) {
    ret = new Item[] {
    null,
      null
    };
  } else if (item.key < key) {
    ret = split(item.right, key);
    item.right = ret[0];
    ret = new Item[] {
      item,
      ret[1]
    };
  } else {
    ret = split(item.left, key);
    item.left = ret[1];
    ret = new Item[] {
      ret[0], item
    };
  }
  updateCnt(item);
```

```
  return ret;
}
// suppose key in l strictly < key in r
Item merge(Item l, Item r) {
  Item ret = null;
  if (l == null || r == null) ret = (l != null) ? l : r;
  else if (l.priority > r.priority) {
    l.right = merge(l.right, r);
    ret = l;
  } else {
    r.left = merge(l, r.left);
    ret = r;
  }
  updateCnt(ret);
  return ret;
}
Item insert(Item root, Item item) {
  Item ret = null;
  if (root == null) {
    ret = item;
  } else if (root.priority < item.priority) {
    Item[] res = split(root, item.key);
    item.left = res[0];
    item.right = res[1]; //System.out.println(cnt(res[0]) + " " + cnt(res[1]));  ret = item;
  } else {
    if (root.key > item.key) {
      ret = insert(root.left, item);
      root.left = ret;
      ret = root;
    } else {
      ret = insert(root.right, item);
      root.right = ret;
      ret = root;
    }
```

```
  }
  updateCnt(ret);
  return ret;
}
Pair < Item, Long > searchNoGreaterThan(Item root, double key) {
  Item[] res = split(root, key);
  long ret = cnt(res[0]);
  return new Pair < > (merge(res[0], res[1]), ret);
}
void printTreap(Item root) {
  System.out.println("total num: " + cnt(root));
  if (root == null) return;
  Queue < Item > q = new LinkedList < > ();
  q.add(root);
  int blank = 1;
  while (!q.isEmpty()) {
    int size = q.size();
    for (int i = 0; i < size; i++) {
      Item node = q.poll();
      System.out.print("<" + node.key + "," + node.priority + ">");
      for (int j = 0; j < blank; j++) {
        System.out.print(" ");
      }
      if (node.left != null) q.add(node.left);
      if (node.right != null) q.add(node.right);
    }
    System.out.println();
    blank <<= 1;
  }
  System.out.println(" ------------------");
}
public int reversePairs(int[] nums) {
  if (nums == null || nums.length == 0) return 0;
  int LEN = nums.length;
```

```
    Random rand = new Random();
    Item root = new Item();
    root.priority = rand.nextDouble();
    root.key = nums[LEN - 1] + 0.0;
    int ans = 0;
    for (int i = LEN - 2; i >= 0; i--) {
      Pair < Item, Long > ret = searchNoGreaterThan(root, (nums[i] + 0.0) / 2);
      ans += ret.right;
      root = ret.left;
      Item e = new Item();
      e.priority = rand.nextDouble();
      e.key = nums[i] + 0.0;
      root = insert(root, e);
    }
    return ans;
  }
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int arr[] = new int[n];
    for (int i = 0; i < n; i++)
    arr[i] = sc.nextInt();
    System.out.println(new Solution().reversePairs(arr));
  }
}
```

f) Write a JAVA Program to find a permutation of the vertices (*topological order*) which corresponds to the order defined by all edges of the graph

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

For example, a topological sorting of the following graph is “5 4 2 3 1 0”. There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is “4 5 2 3 1 0”. The first vertex in topological sorting is always a vertex with in degree as 0 (a vertex with no incoming edges).

PROGRAM

```
import java.io.*;
import java.util.*;
class TopologicalSort {
  private int V;
  // Adjacency List as ArrayList of ArrayList's
  private ArrayList < ArrayList < Integer > > adj;
  TopologicalSort(int v) {
    V = v;
    adj = new ArrayList < ArrayList < Integer > > (v);
    for (int i = 0; i < v; ++i)
      adj.add(new ArrayList < Integer > ());
  }
  // Function to add an edge into the graph
  void addEdge(int v, int w) {
  adj.get(v).add(w);
  }
  // A recursive function used by topologicalSort
  void topologicalSortUtil(int v, boolean visited[], Stack < Integer > stack) {
    // Mark the current node as visited.
    visited[v] = true;
    Integer i;
    // Recur for all the vertices adjacent to this vertex
    Iterator < Integer > it = adj.get(v).iterator();
    while (it.hasNext()) {
```

```
      i = it.next();
      if (!visited[i])
        topologicalSortUtil(i, visited, stack);
    }
    // Push current vertex to stack which stores result
    stack.push(v);
  }
  void topologicalSort() {
    Stack < Integer > stack = new Stack < Integer > ();
    // Mark all the vertices as not visited
    boolean visited[] = new boolean[V];
    for (int i = 0; i < V; i++)
      visited[i] = false;
    // Call the recursive helper function to store Topological Sort starting from all vertices one by one
    for (int i = 0; i < V; i++)
      if (visited[i] == false)
        topologicalSortUtil(i, visited, stack);
    // Print contents of stack
    while (stack.empty() == false)
      System.out.print(stack.pop() + " ");
  }
  // Driver code
  public static void main(String args[]) {
    TopologicalSort g = new TopologicalSort(6);
    g.addEdge(5, 2);
    g.addEdge(5, 0);
    g.addEdge(4, 0);
    g.addEdge(4, 1);
    g.addEdge(2, 3);
    g.addEdge(3, 1);
    System.out.println("Following is a Topological " + "sort of the given graph");
    g.topologicalSort();
  }
}
```

g) Write a JAVA Program to find all the articulation points of a graph

We are given an undirected graph. An articulation point (or cut vertex) is defined as a vertex which, when removed along with associated edges, makes the graph disconnected (or more precisely, increases the number of connected components in the graph). The task is to find all articulation points in the given graph.

PROGRAM

```
import java.util.*;
class Graph {
static int time;
  static void addEdge(ArrayList < ArrayList < Integer > > adj, int u, int v) {
    adj.get(u).add(v);
    adj.get(v).add(u);
  }
  static void APUtil(ArrayList < ArrayList < Integer > > adj, int u, boolean visited[], int disc[], int low[],
int parent, boolean isAP[]) {
    // Count of children in DFS Tree
    int children = 0;
    // Mark the current node as visited
    visited[u] = true;
    // Initialize discovery time and low value
    disc[u] = low[u] = ++time;
    // Go through all vertices adjacent to this
    for (Integer v: adj.get(u)) {
      // If v is not visited yet, then make it a child of u
      // in DFS tree and recur for it
      if (!visited[v]) {
        children++;
        APUtil(adj, v, visited, disc, low, u, isAP);
        // Check if the subtree rooted with v has a connection to one of the ancestors of u low[u] =
Math.min(low[u], low[v]);
        // If u is not root and low value of one of its child is more than discovery value of u.
        if (parent != -1 && low[v] >= disc[u])
          isAP[u] = true;
      }
      // Update low value of u for parent function calls.
```

```
    else if (v != parent)
      low[u] = Math.min(low[u], disc[v]);
  }
  // If u is root of DFS tree and has two or more children.
  if (parent == -1 && children > 1)
    isAP[u] = true;
}
static void AP(ArrayList < ArrayList < Integer > > adj, int V) {
  boolean[] visited = new boolean[V];
  int[] disc = new int[V];
  int[] low = new int[V];
  boolean[] isAP = new boolean[V];
  int time = 0, par = -1;
  // Adding this loop so that the code works even if we are given disconnected graph
  for (int u = 0; u < V; u++)
    if (visited[u] == false)
      APUtil(adj, u, visited, disc, low, par, isAP);
  for (int u = 0; u < V; u++)
    if (isAP[u] == true)
      System.out.print(u + " ");
  System.out.println();
}
public static void main(String[] args) {
  // Creating first example graph
  int V = 5;
  ArrayList < ArrayList < Integer > > adj1 = new ArrayList < ArrayList < Integer > > (V);
  for (int i = 0; i < V; i++)
    adj1.add(new ArrayList < Integer > ());
  addEdge(adj1, 1, 0);
  addEdge(adj1, 0, 2);
  addEdge(adj1, 2, 1);
  addEdge(adj1, 0, 3);
  addEdge(adj1, 3, 4);
  System.out.println("Articulation points in the graph");
```

```
    AP(adj1, V);
  }
}
```

Output

```
Articulation points in the graph
0 3
```

h) Write a JAVA Program to check whether the permutation of a string forms a palindrome

Given a string s, return true if a permutation of the string could form a

palindrome. **Example 1:**

Input: s = "code"

Output: false

Example 2:

Input: s = "aab"

Output: true

Example 3:

Input: s = "carerac"

Output: true

PROGRAM

```
import java.util.*;
class PermutationPalindrome {
  public boolean canPermutePalindrome(String s) {
    Integer bitMask = 0;
    for (int i = 0; i < s.length(); i++)
    bitMask ^= 1 << (s.charAt(i) - 'a' + 1);
    return Integer.bitCount(bitMask) <= 1;
  }
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    System.out.println(new PermutationPalindrome().canPermutePalindrome(sc.next()));
  }
}
```

i) Write a JAVA Program to return all index pairs [i,j] given a text string and words (a list), so that the substring text[i]…text[j] is in the list of words

Given a text string and words (a list of strings), return all index pairs [i, j] so that the substring text[i]...text[j] is in the list of words.

Note:

- All strings contains only lowercase English letters.
- It's guaranteed that all strings in words are different.
- Return the pairs [i,j] in sorted order (i.e. sort them by their first coordinate in case of ties sort them by their second coordinate).

Example 1:

Input: text = "thestoryofleetcodeandme", words =

["story","fleet","leetcode"] Output: [[3,7],[9,13],[10,17]]

Example 2:

Input: text = "ababa", words = ["aba","ab"]

Output: [[0,1],[0,2],[2,3],[2,4]]

Explanation:

Notice that matches can overlap, see "aba" is found in [0,2] and [2,4].

PROGRAM

```
import java.util.*;
class Solution {
  public int[][] indexPairs(String text, String[] words) {
    List < int[] > indexPairsList = new ArrayList < int[] > ();
    for (String word: words) {
      int wordLength = word.length();
      int curIndex = 0;
      while (curIndex >= 0) {
        curIndex = text.indexOf(word, curIndex);
        if (curIndex >= 0) {
```

```
          indexPairsList.add(new int[] {
            curIndex,
            curIndex + wordLength - 1
          });
          curIndex++;
        }
      }
    }
    Collections.sort(indexPairsList, new Comparator < int[] > () {
      public int compare(int[] array1, int[] array2) {
        return ((array1[0] != array2[0]) ? (array1[0] - array2[0]) : (array1[1] - array2[1]));
      }
    });
    int length = indexPairsList.size();
    int[][] indexPairs = new int[length][2];
    for (int i = 0; i < length; i++) {
      int[] indexPair = indexPairsList.get(i);
      indexPairs[i][0] = indexPair[0];
      indexPairs[i][1] = indexPair[1];
    }
    return indexPairs;
  }
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    String text = sc.nextLine();
    String[] words = sc.nextLine().split(" ");
    int[][] result = new Solution().indexPairs(text, words);
    for (int[] res: result)
    System.out.print(Arrays.toString(res));
  }
}
```

j) Write a JAVA Program to find the lowest common ancestor of a binary tree Given a

binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).”

Given the following binary tree: root = [3,5,1,6,2,0,8,null,null,7,4]

Example 1:

Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

Output: 3

Explanation: The LCA of nodes 5 and 1 is 3.

Example 2:

Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4

Output: 5

Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

Note:

All of the nodes' values will be unique.

p and q are different and both values will exist in the binary tree.

PROGRAM

```
import java.util.*;

public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      String[] arr= sc.nextLine().split(" ");
      String[] persons = sc.nextLine().split(" ");

      List<Integer> v = new ArrayList<>();
      int n=arr.length;
      for (int i = 0; i < n; i++) {
         v.add(Integer.parseInt(arr[i]));
      }
```

```
        Node root = new Node(v.get(0));
        Node P1 = new Node(Integer.parseInt(persons[0]));
        Node P2 = new Node(Integer.parseInt(persons[1]));
        Queue<Node> q = new LinkedList<>();
        Queue<Node> q2 = new LinkedList<>();

        q.add(root);

        int j = 1;
        while (j < n && !q.isEmpty()) {
            Node temp = q.poll();
            if (v.get(j) != -1) {
                temp.left = new Node(v.get(j));
                q.add(temp.left);
            }

            j++;

            if (j < n && v.get(j) != -1) {
                temp.right = new Node(v.get(j));
                q.add(temp.right);
            }

            j++;
        }
        Node res=new Solution().lowestCommonAscendant(root, P1, P2);
        System.out.println(res.data);
    }
}

class Node {
    public int data;
    public Node left;
    public Node right;
    public Node(int value) {
        data = value;
        left = null;
        right = null;
    }
}

class Solution {
    Node lowestCommonAscendant(Node root,Node P1, Node P2){
        if(root==null)
        {
            return null;
        }
        if(P1.data ==root.data || P2.data==root.data)
        {
            return root;
        }
```

```
        Node left=lowestCommonAscendant(root.left,P1,P2);
        Node right=lowestCommonAscendant(root.right,P1,P2);

        if(left==null)
        {
           return right;
        }
        else if(right ==null)
        {
           return left;
        }
        else
        {
           return root;
        }
    }
}
```

k) Write a JAVA Program to find the Longest Increasing Path in a Matrix.

Given an integer matrix, find the length of the longest increasing path. From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

Example 1:
Input: nums = 3 3
9 9 4
6 6 8
2 1 1

Output: 4

Explanation: The longest increasing path is [1, 2, 6, 9].

Example 2:
Input: nums =3 3
3 4 5
3 2 6
2 2 1

Output: 4

Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

PROGRAM

```
import java.util.*;
public class Solution {
  private int[] dx = new int[] { 0, 0, -1, 1 };
  private int[] dy = new int[] { 1, -1, 0, 0 };
  public int longestIncreasingPath(int[][] matrix) {
    if (matrix == null || matrix.length == 0) {
    return 0;
    }
    int longest = 0;
    int m = matrix.length;
    int n = matrix[0].length;
    // longest path starting from position (i,j)
    int[][] dp = new int[m][n];
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        longest = Math.max(longest, dfs(i, j, matrix, dp));
      }
```

```
    }
    return longest;
  }
  private int dfs(int row, int col, int[][] matrix, int[][] dp) {
    if (dp[row][col] > 0) {
      return dp[row][col];
    }
    int m = matrix.length;
    int n = matrix[0].length;
    int currentLongest = 0;
    for (int c = 0; c < 4; c++) {
      int i = row + dx[c];
      int j = col + dy[c];
      if (i >= 0 && i < m && j >= 0 && j < n && matrix[row][col] < matrix[i][j]) {
        currentLongest = Math.max(currentLongest, dfs(i, j, matrix, dp));
      }
    }
    dp[row][col] = 1 + currentLongest;
    return dp[row][col];
  }
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    int m = sc.nextInt();
    int n = sc.nextInt();
    int matrix[][] = new int[m][n];
    for (int i = 0; i < m; i++)
      for (int j = 0; j < n; j++)
        matrix[i][j] = sc.nextInt();
    System.out.println(new Solution().longestIncreasingPath(matrix));
  }
}
```

2) Develop a java program to find the Lexicographically smallest equivalent string.

Given strings A and B of the same length, we say A[i] and B[i] are equivalent characters. For example, if A = "abc" and B = "cde", then we have 'a' == 'c', 'b' == 'd', 'c' == 'e'. Equivalent characters follow the usual rules of any equivalence relation:

- Reflexivity: 'a' == 'a'
- Symmetry: 'a' == 'b' implies 'b' == 'a'
- Transitivity: 'a' == 'b' and 'b' == 'c' implies 'a' == 'c'

For example, given the equivalency information from A and B above, S = "eed", "acd", and "aab" are equivalent strings, and "aab" is the lexicographically smallest equivalent string of S. Return the lexicographically smallest equivalent string of S by using the equivalency information from A and B.

Example 1:

Input: A = "parker", B = "morris", S = "parser"

Output: "makkek"

Explanation: Based on the equivalency information in A and B, we can group their characters as [m,p], [a,o], [k,r,s], [e,i]. The characters in each group are equivalent and sorted in lexicographical order. So the answer is "makkek".

Example 2:

Input: A = "hello", B = "world", S = "hold"

Output: "hdld"

Explanation: Based on the equivalency information in A and B, we can group their characters as [h,w], [d,e,o], [l,r]. So only the second letter 'o' in S is changed to 'd', the answer is "hdld".

Example 3:

Input: A = "leetcode", B = "programs", S = "sourcecode"

Output: "aauaaaaada"

Explanation: We group the equivalent characters in A and B as [a,o,e,r,s,c], [l,p], [g,t] and [d,m], thus all letters in S except 'u' and 'd' are transformed to 'a', the answer is "aauaaaaada".

Note:

String A, B and S consist of only lowercase English letters from 'a' -

'z'. String A and B are of the same length.

PROGRAM

```
import java.util.*;
class Solution {
  public String smallestEquivalentString(String A, String B, String S) {
    int[] graph = new int[26];
    for (int i = 0; i < 26; i++) {
      graph[i] = i;
    }
    for (int i = 0; i < A.length(); i++) {
      int a = A.charAt(i) - 'a';
      int b = B.charAt(i) - 'a';
      int end1 = find(graph, b);
      int end2 = find(graph, a);
      if (end1 < end2) {
      graph[end2] = end1;
      } else {
        graph[end1] = end2;
      }
    }
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < S.length(); i++) {
      char c = S.charAt(i);
      sb.append((char)('a' + find(graph, c - 'a')));
    }
    return sb.toString();
  }
  private int find(int[] graph, int idx) {
    while (graph[idx] != idx) {
      idx = graph[idx];
    }
    return idx;
  }
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
```

```
    String A = sc.next();
    String B = sc.next();
    String substr = sc.next();
    System.out.println(new Solution().smallestEquivalentString(A,B,substr));
  }
}
```