GATE TECHNICAL TRAINING – DSA CODING PRACTICE PROBLEMS 2026

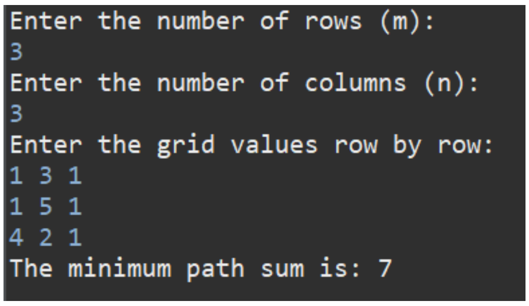
DATE: 19-11-2024 **NAME: S Yathissh – CSBS**

1. MINIMUM PATH

SUMPROGRAM:

```java
package dsaPracticeProblems;
import java.util.Scanner;

class MinimumPathSum {
    public static int minPathSum(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        for (int j = 1; j < n; j++) {
            grid[0][j] += grid[0][j - 1];
        }

        for (int i = 1; i < m; i++) {
            grid[i][0] += grid[i - 1][0];
        }

        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                grid[i][j] += Math.min(grid[i - 1][j], grid[i][j - 1]);
            }
        }

        return grid[m - 1][n - 1];
    }

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

        System.out.println("Enter the number of rows (m):");
        int m = scanner.nextInt();
        System.out.println("Enter the number of columns (n):");
        int n = scanner.nextInt();

        int[][] grid = new int[m][n];
        System.out.println("Enter the grid values row by row:");

        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }

        int result = minPathSum(grid);

        System.out.println("The minimum path sum is: " + result);
        scanner.close();
    }
}
```

OUTPUT:



```
Enter the number of rows (m):
3
Enter the number of columns (n):
3
Enter the grid values row by row:
1 3 1
1 5 1
4 2 1
The minimum path sum is: 7
```

TIME COMPLEXITY: O(m*n)

2. **VALIDATE BINARY**

SEARCH TREEPROGRAM:

```java
package dsaPracticeProblems;
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode() {}

    TreeNode(int val) {
        this.val = val;
    }

    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}

class Solution {
    public boolean isValidBST(TreeNode root) {
        if (root == null) return true;

        Stack<TreeNode> st = new Stack<>();
        TreeNode previous = null;

        while (root != null || !st.isEmpty()) {
            while (root != null) {
                st.push(root);
                root = root.left;
            }

            root = st.pop();
            if (previous != null && root.val <= previous.val) return false;

            previous = root;
            root = root.right;
        }
        return true;
    }
}

public class ValidateBST {
    public static void main(String[] args) {
        String[] values = {"2", "1", "3"};

        TreeNode root = buildTree(values);

        Solution solution = new Solution();
        boolean isValid = solution.isValidBST(root);

        System.out.println("Is the binary tree a valid BST? " + isValid);
    }

    private static TreeNode buildTree(String[] values) {
        if (values.length == 0 || values[0].equals("null")) return null;

        TreeNode root = new TreeNode(Integer.parseInt(values[0]));
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        int i = 1;

        while (i < values.length) {
            TreeNode current = queue.poll();

            if (i < values.length && !values[i].equals("null")) {
                TreeNode leftChild = new TreeNode(Integer.parseInt(values[i]));
                current.left = leftChild;
                queue.add(leftChild);
            }
            i++;

            if (i < values.length && !values[i].equals("null")) {
                TreeNode rightChild = new TreeNode(Integer.parseInt(values[i]));
                current.right = rightChild;
                queue.add(rightChild);
            }
            i++;
        }

        return root;
    }
}
```

OUTPUT:

```
Is the binary tree a valid BST? true
```

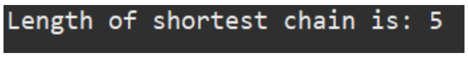
TIME COMPLEXITY: O(n)

3. WORD LADDER

PROGRAM:

```
package dsaPracticeProblems;import
java.util.*;

class WordLadder {

static int shortestChainLen(String start, String target, Set<String> D)
{

    if(start == target)return 0;

  if (!D.contains(target))return 0;

  int level = 0, wordlength = start.length();

  Queue<String> Q = new LinkedList<>();
  Q.add(start);

  while (!Q.isEmpty())
  {

          ++level;

          int sizeofQ = Q.size();

          for (int i = 0; i < sizeofQ; ++i)
          {

                  char []word = Q.peek().toCharArray();Q.remove();

                  for (int pos = 0; pos < wordlength; ++pos)
                  {

                          char orig_char = word[pos];

                          for (char c = 'a'; c <= 'z'; ++c)
                          {
                                  word[pos] = c;

                                  if (String.valueOf(word).equals(target))return level +
                                          1;

                                  if (!D.contains(String.valueOf(word)))continue;
                                  D.remove(String.valueOf(word));

                                  Q.add(String.valueOf(word));
                          }

                          word[pos] = orig_char;
                  }
          }
  }

  return 0;
}

public static void main(String[] args)
{
  Set<String> D = new HashSet<String>();
  D.add("hot");
  D.add("dot");
  D.add("dog");
  D.add("lot");
  D.add("log");
  D.add("cog");
  String start = "hit";
  String target = "cog";
  System.out.print("Length of shortest chain is: " + shortestChainLen(start,
target, D));
}
}
```

OUTPUT:



```
Length of shortest chain is: 5
```

TIME COMPLEXITY: O(n*l)

4. WORD LADDER -II

PROGRAM:

OUTPUT:

TIME COMPLEXITY:

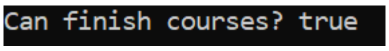
5. COURSE

SCHEDULEPROGRAM:

```
package dsaPracticeProblems;
import java.util.*;

class Solution1 {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        List<List<Integer>> graph = new ArrayList<>();
        for (int i = 0; i < numCourses; i++) {
            graph.add(new ArrayList<>());
        }

        for (int[] prereq : prerequisites) {
            graph.get(prereq[1]).add(prereq[0]);
        }

        int[] visited = new int[numCourses];
        for (int i = 0; i < numCourses; i++) {
            if (visited[i] == 0) {
```

```java
                if (hasCycle(i, graph, visited)) {
                    return false;
                }
            }
        }

        return true;
    }

    private boolean hasCycle(int node, List<List<Integer>> graph, int[] visited) {
        if (visited[node] == 1) {
            return true;
        }

        if (visited[node] == 2) {
            return false;
        }

        visited[node] = 1;

        for (int neighbor : graph.get(node)) {
            if (hasCycle(neighbor, graph, visited)) {
                return true;
            }
        }

        visited[node] = 2;

        return false;
    }
}

public class CourseSchedule {
    public static void main(String[] args) {
        Solution1 solution = new Solution1();

        int numCourses1 = 2;
        int[][] prerequisites1 = {{1, 0}};
        System.out.println("Can finish courses? " +
solution.canFinish(numCourses1, prerequisites1));
 }
}
```

OUTPUT:



```
Can finish courses? true
```

TIME COMPLEXITY: O(V+E)

6. **DESIGN TIC TAC**

TOEPROGRAM:

OUTPUT:

TIME COMPLEXITY:

7. NEXT PERMUTATION

PROGRAM:

```
package dsaPracticeProblems;import
java.util.*;

public class NextPermutation {
        static void nextPermutation(int[] arr) {List<int[]> res
                = new ArrayList<>();

                permutations(res, arr, 0); Collections.sort(res,
                Arrays::compare);

                for (int i = 0; i < res.size(); i++) {

                        if (Arrays.equals(res.get(i), arr)) {if (i < res.size()
                                - 1) {
                                        int[] nextPerm = res.get(i + 1); for(int j = 0; j <
                                        arr.length; j++)
                                                arr[j] = nextPerm[j];
                                }

                                if (i == res.size() - 1) { int[] nextPerm =
                                        res.get(0);
                                        for(int j = 0; j < arr.length; j++)arr[j] =
                                                nextPerm[j];
                                }

                                break;
                        }
                }
        }

        static void permutations(List<int[]> res, int[] arr, int idx) {if (idx == arr.length - 1) {
                        res.add(arr.clone());return;
                }
                for (int i = idx; i < arr.length; i++) {swap(arr,   idx,
                        i); permutations(res, arr, idx + 1); swap(arr, idx,
                        i);
                }
        }

        static void swap(int[] arr, int i, int j) {int temp = arr[i];
                arr[i] = arr[j];arr[j] =
                temp;
        }

        public static void main(String[] args) { Scanner scanner =
             new Scanner(System.in);
             System.out.println("Enter the number of elements: ");int n =
             scanner.nextInt();

             int[] arr = new int[n];
             System.out.println("Enter the elements in the array: ");
```

```
       for(int i = 0; i<n ; i++) {
             arr[i] = scanner.nextInt();
       }
       nextPermutation(arr);

       System.out.println("The next permutation is: ");
        for (int num : arr)
            System.out.print(num + " ");
    }
}
```

OUTPUT:

```
Enter the number of elements:
6
Enter the elements in the array:
2 4 1 7 5 0
The next permutation is:
2 4 5 0 1 7
```

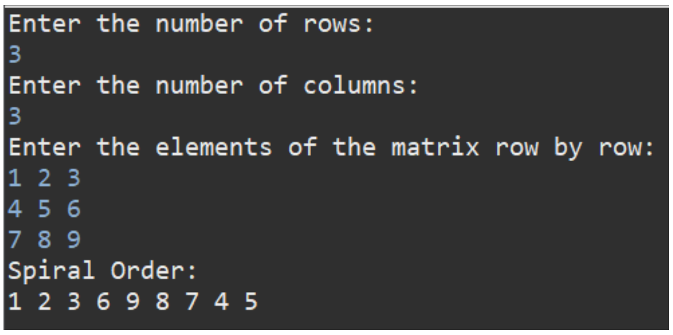
TIME COMPLEXITY: O(n!.n)

8. SPIRAL MATRIX

PROGRAM:

```
package dsaPracticeProblems;
import java.util.*;

public class SpiralMatrix {
  public List<Integer> spiralOrder(int[][] matrix) {
    if (matrix.length == 0)
      return new ArrayList<>();

    final int m = matrix.length;
    final int n = matrix[0].length;
    List<Integer> ans = new ArrayList<>();
    int r1 = 0;
    int c1  =  0;
    int r2 = m - 1;
    int c2 = n - 1;

    while (ans.size() < m * n) {
      for (int j = c1; j <= c2 && ans.size() < m * n; ++j)
        ans.add(matrix[r1][j]);
      for (int i = r1 + 1; i <= r2 - 1 && ans.size() < m * n; ++i)
        ans.add(matrix[i][c2]);
      for (int j = c2; j >= c1 && ans.size() < m * n; --j)
        ans.add(matrix[r2][j]);
      for (int i = r2 - 1; i >= r1 + 1 && ans.size() < m * n; --i)
        ans.add(matrix[i][c1]);
      ++r1;
      ++c1;
      --r2;
      --c2;
    }
    return ans;
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the number of rows:");
    int m = scanner.nextInt();
    System.out.println("Enter the number of columns:");
    int n = scanner.nextInt();

    int[][] matrix = new int[m][n];
    System.out.println("Enter the elements of the matrix row by row:");

    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        matrix[i][j] = scanner.nextInt();
      }
    }

    SpiralMatrix solution = new SpiralMatrix();
    List<Integer> result = solution.spiralOrder(matrix);

    System.out.println("Spiral Order:");
    for (int num : result) {
      System.out.print(num + " ");
    }
  }
}
```

OUTPUT:



```
Enter the number of rows:
3
Enter the number of columns:
3
Enter the elements of the matrix row by row:
1 2 3
4 5 6
7 8 9
Spiral Order:
1 2 3 6 9 8 7 4 5
```

TIME COMPLEXITY: O(mn)

9. LONGEST SUBSTRING WITHOUT REPEATING CHARACTERSPROGRAM:

```
package dsaPracticeProblems;
import java.util.*;

public class LongestSubstring {
 static int longestUniqueSubstr(String s) {
     int n = s.length();
     int res = 0;
     for (int i = 0; i < n; i++) {
         boolean[] visited = new boolean[256];

         for (int j = i; j < n; j++) {
             if (visited[s.charAt(j)]) {
                 break;
             }
               else {
                 res = Math.max(res, j - i + 1);
                 visited[s.charAt(j)] = true;
             }
         }
     }
     return res;
 }

 public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the String: ");
        String s = scanner.nextLine();

     System.out.println("The length of the longest substring is
"+longestUniqueSubstr(s));
 }
}
```

OUTPUT:

```
Enter the String:
aabccc
The length of the longest substring is 3
```

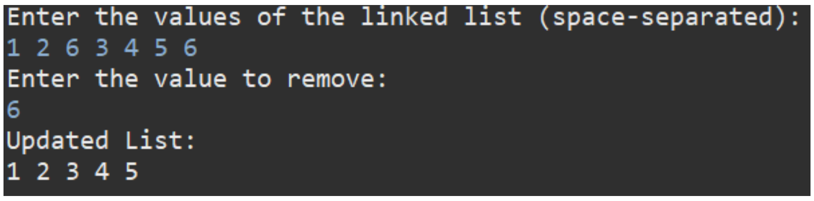
TIME COMPLEXITY: O(n^2)

10. REMOVE LINKED LIST

ELEMENTSPROGRAM:

```
package dsaPracticeProblems;
import java.util.Scanner;

class ListNode {
    int val;
    ListNode next;
    ListNode() {}
    ListNode(int val) { this.val = val; }
    ListNode(int val, ListNode next) { this.val = val; this.next = next; }
}

class RemoveLinkedListElement {
    public ListNode removeElements(ListNode head, int val) {
        ListNode res = new ListNode(0, head);
        ListNode temp = res;

        while (temp != null) {
                while (temp.next != null && temp.next.val == val) {temp.next =
                        temp.next.next;
                }
                temp = temp.next;
        }
        return res.next;
}

public void printList(ListNode head) {

        while (head != null) {
                System.out.print(head.val);head =
                head.next;
                if (head != null) {
                        System.out.print(" ");
                }
        }

}

public ListNode createList(int[] values) { if (values.length
        == 0) return null; ListNode head = new
        ListNode(values[0]);ListNode current = head;
        for (int i = 1; i < values.length; i++) { current.next = new
                ListNode(values[i]);current = current.next;
        }
        return head;
}

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

        System.out.println("Enter the values of the linked list (space-separated):
");
        String[] input = scanner.nextLine().split(" ");int[] values = new
        int[input.length];
        for (int i = 0; i < input.length; i++) { values[i] =
                Integer.parseInt(input[i]);
        }

        System.out.println("Enter the value to remove: ");int val =
        scanner.nextInt();

        RemoveLinkedListElement solution = new RemoveLinkedListElement();ListNode head
        = solution.createList(values);

        ListNode updatedList = solution.removeElements(head, val);

        System.out.println("Updated List:");
        solution.printList(updatedList);

        scanner.close();
}
}
```

OUTPUT:



```
Enter the values of the linked list (space-separated):
1 2 6 3 4 5 6
Enter the value to remove:
6
Updated List:
1 2 3 4 5
```

TIME COMPLEXITY: O(n)

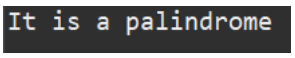
11. PALINDROME

LINKED LISTPROGRAM:

```java
package dsaPracticeProblems;
import java.util.Stack;

class Node {
 int data;
 Node next;
 Node(int d) {
     data = d;
     next = null;
 }
}

class PalindromeLinkedList {
 static boolean isPalindrome(Node head) {
     Node currNode  =  head;
     Stack<Integer> s = new Stack<>();

     while (currNode != null) {
         s.push(currNode.data);
         currNode = currNode.next;
     }

     while (head != null) {
         int c = s.pop();

         if (head.data != c) {
             return false;
         }

         head = head.next;
     }

     return true;
 }

 public static void main(String[] args) {
     Node head = new Node(1);
     head.next =  new  Node(2);
     head.next.next = new Node(3);
     head.next.next.next = new Node(2);
     head.next.next.next.next = new Node(1);

     boolean result = isPalindrome(head);

     if (result)
```

```
         System.out.println("It is a palindrome");
     else
         System.out.println("It is not a palindrome");
 }
}
```

OUTPUT:



```
It is a palindrome
```

TIME COMPLEXITY: O(n)