NAME: BHARATH V

DEPT: CSBS

1.Anagram Problem

```
import java.util.HashMap;
public class AnagramCheck {
   public static boolean areAnagrams(String str1, String str2) {
      if (str1.length() != str2.length()) {
         return false;
      }
      HashMap<Character, Integer> map1 = new HashMap<>();
      HashMap<Character, Integer> map2 = new HashMap<>();
      for (char c : str1.toCharArray()) {
         map1.put(c, map1.getOrDefault(c, 0) + 1);
      }
      for (char c : str2.toCharArray()) {
         map2.put(c, map2.getOrDefault(c, 0) + 1);
      }
      return map1.equals(map2);
   }

   public static void main(String[] args) {
      String str1 = "listen";
      String str2 = "silent";
      if (areAnagrams(str1, str2)) {
```

```
            System.out.println(str1 + " and " + str2 + " are anagrams.");
        } else {
            System.out.println(str1 + " and " + str2 + " are not anagrams.");
        }
    }
}
```

OUTPUT

Listen and silent are anagrams

Complexity

Time: O(n)

Space: O(1)

2.Row with Max 1’s

```
public class RowWithMaxOnes {
    public static int rowWithMaxOnes(int[][] matrix) {
        int rowCount = matrix.length;
        int colCount = matrix[0].length;
        int maxRowIndex = -1;
        int maxOnesCount = -1;
        int j = colCount - 1;

        for (int i = 0; i < rowCount; i++) {
            while (j >= 0 && matrix[i][j] == 1) {
                j--;
                if (maxOnesCount < colCount - j - 1) {
                    maxOnesCount = colCount - j - 1;
```

```
                maxRowIndex = i;
            }
        }
    }

    return maxRowIndex;
}

public static void main(String[] args) {
    int[][] matrix = {
        {0, 0, 1, 1},
        {0, 1, 1, 1},
        {1, 1, 1, 1},
        {0, 0, 0, 1}
    };

    int result = rowWithMaxOnes(matrix);
    if (result == -1) {
        System.out.println("No 1's in the matrix.");
    } else {
        System.out.println("Row with maximum 1's is: " + result);
    }
}
}
```

OUTPUT

Row with maximum 1’s is 2

Complexity

Time: O(n+m)

Space: O(1)

3.Longest Consecutive subsequences

```
import java.util.HashSet;
public class LongestConsecutiveSubsequence {
    public static int longestConsecutive(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : nums) {
            set.add(num);
        }

        int longestStreak = 0;
        for (int num : set) {
            if (!set.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;
                while (set.contains(currentNum + 1)) {
                    currentNum++;
                    currentStreak++;
                }
                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }
```

```
        return longestStreak;
    }

    public static void main(String[] args) {
        int[] nums = {100, 4, 200, 1, 3, 2};
        int result = longestConsecutive(nums);
        System.out.println("Length of the longest consecutive subsequence: " +
result);
    }
}
```

OUTPUT

```
Length of the longest consecutive subsequence: 4
```

Complexity

Time: O(n)

Space: O(n)

4.Longest Palindrome in a String

```
public class Solution4 {
    static String longestPalSubstr(String s) {
        int n = s.length();
        if (n == 0) return "";

        int start = 0, maxLen = 1;
        for (int i = 0; i < n; i++) {
            int len1 = expandAroundCenter(s, i, i);
```

```
            int len2 = expandAroundCenter(s, i, i + 1);
            int len = Math.max(len1, len2);
            if (len > maxLen) {
                maxLen = len;
                start = i - (len - 1) / 2;
            }
        }

        return s.substring(start, start + maxLen);
    }

    static int expandAroundCenter(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }

        return right - left - 1;
    }

    public static void main(String[] args) {
        String s = "forgeeksskeegfor";
        System.out.println(longestPalSubstr(s));
    }
}
```

OUTPUT:

Geeksskeeg

Complexity;

Time:O(n^2)

Space:O(n)

5.Rat in a maze problem

```
public class RatInMaze {
   static void printSolution(int[][] sol) {
      for (int i = 0; i < sol.length; i++) {
         for (int j = 0; j < sol[i].length; j++) {
            System.out.print(sol[i][j] + " ");
         }
         System.out.println();
      }
   }
   static boolean solveMaze(int[][] maze, int x, int y, int[][] sol) {
      int N = maze.length;

      if (x == N - 1 && y == N - 1) {
         sol[x][y] = 1;
         return true;
      }
      if (isSafe(maze, x, y)) {
         // Mark x, y as part of the solution path
```

```
        sol[x][y] = 1;
        if (solveMaze(maze, x + 1, y, sol)) {
            return true;
        }
        if (solveMaze(maze, x, y + 1, sol)) {
            return true;
        }
        if (solveMaze(maze, x - 1, y, sol)) {
            return true;
        }
        if (solveMaze(maze, x, y - 1, sol)) {
            return true;
        }
        sol[x][y] = 0;
        return false;
    }

    return false;
}
static boolean isSafe(int[][] maze, int x, int y) {
    int N = maze.length;
    return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
}

public static void main(String[] args) {
    int[][] maze = {
```

```
            {1, 0, 0, 0},
            {1, 1, 0, 1},
            {0, 1, 0, 0},
            {1, 1, 1, 1}
        };

        int N = maze.length;
        int[][] sol = new int[N][N];
        if (solveMaze(maze, 0, 0, sol)) {
            printSolution(sol);
        } else {
            System.out.println("No path found");
        }
    }
}
```

Complexity:

Time: $O(n^2)$

Space: $O(n^2)$