PRACTICE SET – 3 **13-11-2024**

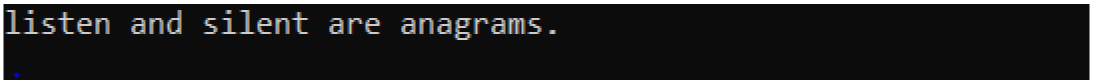
DSA

1. Anagram program

CODE:

```
import java.util.Arrays;

public class AnagramChecker {
  public static boolean isAnagram(String str1, String str2) {
    // Remove all white spaces and convert strings to lowercase
    str1 = str1.replaceAll("\\s", "").toLowerCase();
    str2 = str2.replaceAll("\\s", "").toLowerCase();

    // If lengths are different, they can't be anagrams
    if (str1.length() != str2.length()) {
      return false;
    }

    // Convert strings to char arrays and sort them
    char[] arr1 = str1.toCharArray();
    char[] arr2 = str2.toCharArray();
    Arrays.sort(arr1);
    Arrays.sort(arr2);

    // Compare sorted arrays
```

```
        return Arrays.equals(arr1, arr2);
    }

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

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

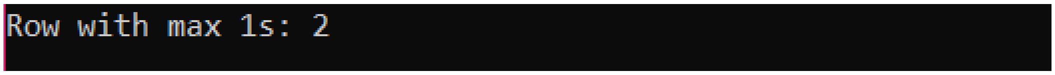


```
listen and silent are anagrams.
```

2. Row with max 1s'

CODE:

```
public class MaxOnesRow {
    public static int rowWithMaxOnes(int[][] matrix) {
        int maxRow = -1;
        int maxCount = 0;

        for (int i = 0; i < matrix.length; i++) {
            int count = countOnes(matrix[i]);
            if (count > maxCount) {
                maxCount = count;
                maxRow = i;
            }
        }

        return maxRow;
    }

    private static int countOnes(int[] row) {
        int left = 0;
        int right = row.length - 1;

        // Binary search for the first 1 in the row
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (row[mid] == 1 && (mid == 0 || row[mid - 1] == 0)) {
                return row.length - mid; // Number of 1s in the row
            } else if (row[mid] == 1) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }

        return 0; // No 1s in the row
    }

    public static void main(String[] args) {
        int[][] matrix = {
            {0, 0, 0, 1},
            {0, 1, 1, 1},
            {1, 1, 1, 1},
            {0, 0, 0, 0}
        };
        int maxRow = rowWithMaxOnes(matrix);
        System.out.println("Row with max 1s: " + maxRow);
    }
}
```



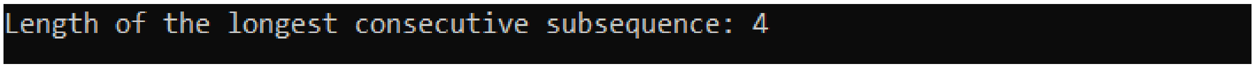
```
Row with max 1s: 2
```

3. Longest consequtive subsequence

CODE:

```
import java.util.HashSet;

public class LongestConsecutiveSubsequence {
    public static int findLongestConsecutiveSubsequence(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        int longestStreak = 0;
    // Add all elements to the set
    for (int num : nums) {
        set.add(num);
    }

    // Find the longest consecutive sequence
    for (int num : nums) {
        // Only start sequence if `num - 1` is not in the set
        if (!set.contains(num - 1)) {
            int currentNum = num;
            int currentStreak = 1;

            // Count consecutive numbers
            while (set.contains(currentNum + 1)) {
                currentNum += 1;
                currentStreak += 1;
            }

            // Update longest streak
            longestStreak = Math.max(longestStreak, currentStreak);
        }
    }

    return longestStreak;
}
```

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

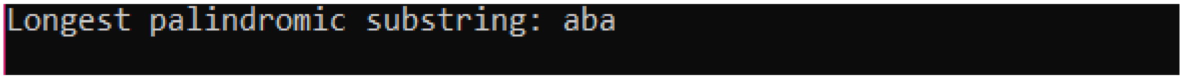


```
Length of the longest consecutive subsequence: 4
```

4. longest palindrome in a string

CODE:

```
public class LongestPalindrome {
    public static String longestPalindrome(String s) {
        if (s == null || s.length() < 1) return "";
        int start = 0, end = 0;

        for (int i = 0; i < s.length(); i++) {
            int len1 = expandAroundCenter(s, i, i);       // Odd-length palindromes
            int len2 = expandAroundCenter(s, i, i + 1);   // Even-length palindromes
            int len = Math.max(len1, len2);

            if (len > end - start) {
                start = i - (len - 1) / 2;
                end = i + len / 2;
            }
        }
        return s.substring(start, end + 1);
    }

    private 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 = "babad";
        System.out.println("Longest palindromic substring: " + longestPalindrome(s));
    }
}
```



```
Longest palindromic substring: aba
```

5. Rat in a maze problem

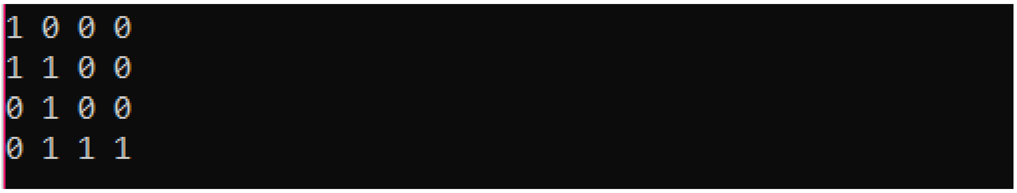
CODE:

```
public class RatInMaze {
    // Dimensions of the maze
    private static final int N = 4;
```

```
// Function to print the solution matrix
private static void printSolution(int[][] solution) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            System.out.print(solution[i][j] + " ");
        }
        System.out.println();
    }
}

// Utility function to check if x, y is valid index for N*N maze
private static boolean isSafe(int[][] maze, int x, int y) {
    return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
}

// Solves the maze problem using backtracking
private static boolean solveMaze(int[][] maze) {
    int[][] solution = new int[N][N]; // Initialize the solution matrix

    if (solveMazeUtil(maze, 0, 0, solution) == false) {
        System.out.println("Solution doesn't exist");
        return false;
    }

    printSolution(solution);
    return true;
```

```
}

// A recursive utility function to solve the Maze problem
private static boolean solveMazeUtil(int[][] maze, int x, int y, int[][] solution) {
    // If (x, y) is the goal, return true
    if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {
        solution[x][y] = 1;
        return true;
    }

    // Check if maze[x][y] is a valid move
    if (isSafe(maze, x, y)) {
        // Mark x, y as part of the solution path
        solution[x][y] = 1;

        // Move forward in x direction
        if (solveMazeUtil(maze, x + 1, y, solution)) {
            return true;
        }

        // If moving in x doesn't work, move down in y direction
        if (solveMazeUtil(maze, x, y + 1, solution)) {
            return true;
        }
```

```
            // If none of the above movements work, backtrack and unmark x, y as part of solution path
            solution[x][y] = 0;
            return false;
        }

        return false;
    }

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

        solveMaze(maze);
    }
}
```



```
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1
```