DSA Programming Practice – 7,8 – KAVYA U – CSBS(III yr) – 2026 BATCH – 22CB024

1. Minimum path sum :

Given a cost matrix cost[][] and a position (M, N) in cost[][], write a function that returns cost of minimum cost path to reach (M, N) from (0, 0). Each cell of the matrix represents a cost to traverse through that cell. The total cost of a path to reach (M, N) is the sum of all the costs on that path (including both source and destination). You can only traverse down, right and diagonally lower cells from a given cell, i.e., from a given cell (i, j), cells (i+1, j), (i, j+1), and (i+1, j+1) can be traversed.

Note: You may assume that all costs are positive integers.

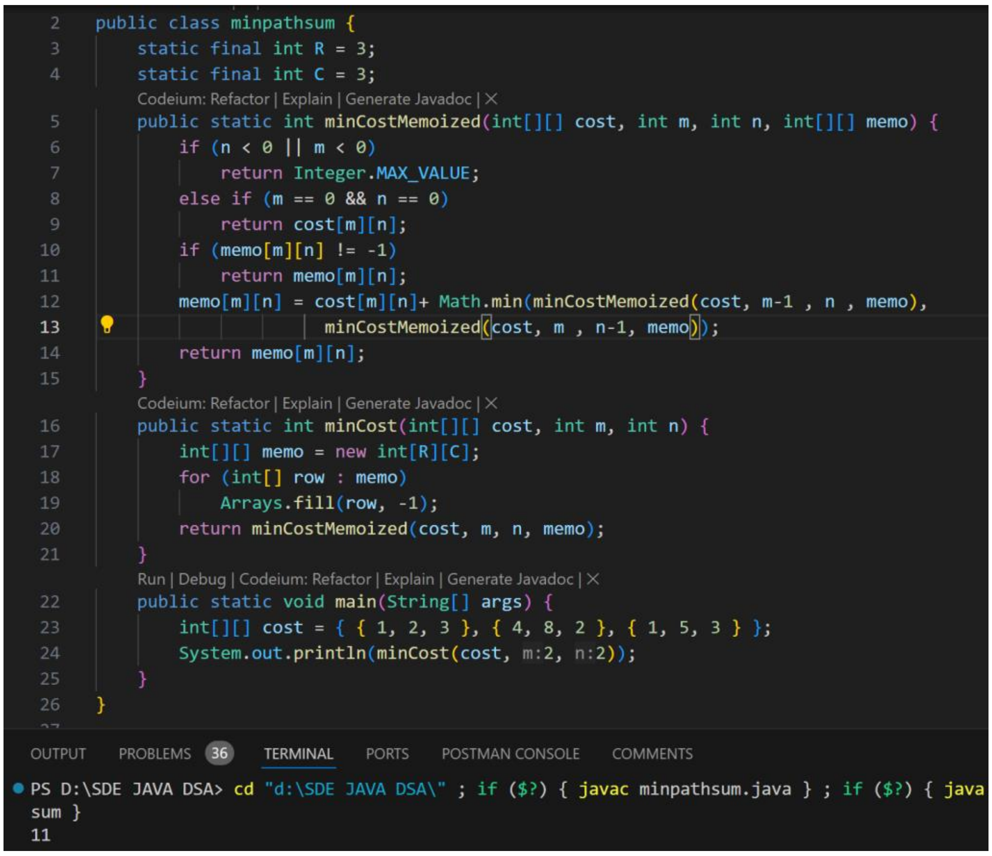
Example:

Input: [[1, 2, 3], [4, 8, 2], [1, 5, 3]]

Explanation : The path with minimum cost is highlighted in the following figure. The path is (0, 0) –> (0, 1) –> (1, 2) –> (2, 2). The cost of the path is 8 (1 + 2 + 2 + 3).

Output: 8

Code and Output:


Codeium: Refactor | Explain | Generate Javadoc | ✕
Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | ✕
OUTPUT
PROBLEMS 36
TERMINAL
PORTS
POSTMAN CONSOLE
COMMENTS

```java
public class minpathsum {
    static final int R = 3;
    static final int C = 3;
    public static int minCostMemoized(int[][] cost, int m, int n, int[][] memo) {
        if (n < 0 || m < 0)
            return Integer.MAX_VALUE;
        else if (m == 0 && n == 0)
            return cost[m][n];
        if (memo[m][n] != -1)
            return memo[m][n];
        memo[m][n] = cost[m][n]+ Math.min(minCostMemoized(cost, m-1 , n , memo),
                    minCostMemoized(cost, m , n-1, memo));
        return memo[m][n];
    }
    public static int minCost(int[][] cost, int m, int n) {
        int[][] memo = new int[R][C];
        for (int[] row : memo)
            Arrays.fill(row, -1);
        return minCostMemoized(cost, m, n, memo);
    }
    public static void main(String[] args) {
        int[][] cost = { { 1, 2, 3 }, { 4, 8, 2 }, { 1, 5, 3 } };
        System.out.println(minCost(cost, m:2, n:2));
    }
}
```

```
PS D:\SDE JAVA DSA> cd "d:\SDE JAVA DSA\" ; if ($?) { javac minpathsum.java } ; if ($?) { java
sum }
11
```

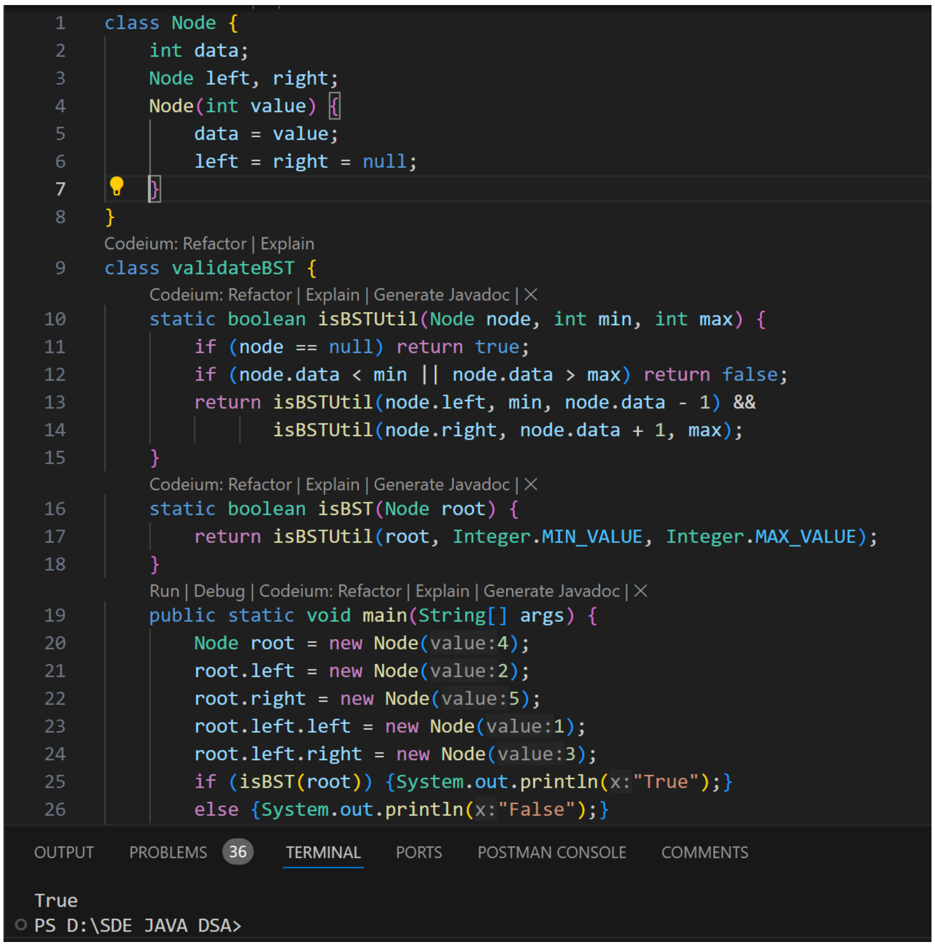
Time Complexity: O(M * N)

2. Validate binary search tree :

Given the root of a binary tree. Check whether it is a Binary Search Tree or not. A Binary Search Tree (BST) is a node-based binary tree data structure with the following properties.

- All keys in the left subtree are smaller than the root and all keys in the right subtree are greater.
- Both the left and right subtrees must also be binary search trees.
- Each key must be distinct.

Code and Output:



```java
class Node {
    int data;
    Node left, right;
    Node(int value) {
        data = value;
        left = right = null;
    }
}
Codeium: Refactor | Explain
class validateBST {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static boolean isBSTUtil(Node node, int min, int max) {
        if (node == null) return true;
        if (node.data < min || node.data > max) return false;
        return isBSTUtil(node.left, min, node.data - 1) &&
               isBSTUtil(node.right, node.data + 1, max);
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static boolean isBST(Node root) {
        return isBSTUtil(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        Node root = new Node(value:4);
        root.left = new Node(value:2);
        root.right = new Node(value:5);
        root.left.left = new Node(value:1);
        root.left.right = new Node(value:3);
        if (isBST(root)) {System.out.println(x:"True");}
        else {System.out.println(x:"False");}
```

```
True
PS D:\SDE JAVA DSA>
```

Time Complexity: O(N)

3. Word ladder:

Given a dictionary, and two words ‘start’ and ‘target’ (both of same length). Find length of the smallest chain from ‘start’ to ‘target’ if it exists, such that adjacent words in the chain only differ by one character and each word in the chain is a valid word i.e., it exists in the dictionary. It may be assumed that the ‘target’ word exists in dictionary and length of all dictionary words is same.

Input: Dictionary = {POON, PLEE, SAME, POIE, PLEA, PLIE, POIN}, start = TOON, target = PLEA

Output: 7

Explanation: TOON – POON – POIN – POIE – PLIE – PLEE – PLEA

Input: Dictionary = {ABCD, EBAD, EBCD, XYZA}, start = ABCV, target = EBAD

Output: 4

Explanation: ABCV – ABCD – EBCD – EBAD

Code and Output:

Time Complexity:

4. Word ladder -II :

Given a dictionary, and two words start and target (both of the same length). Find length of the smallest chain from start to target if it exists, such that adjacent words in the chain only differ by one character and each word in the chain is a valid word i.e., it exists in the dictionary. It may be assumed that the target word exists in the dictionary and the lengths of all the dictionary words are equal.

Input: Dictionary = {POON, PLEE, SAME, POIE, PLEA, PLIE, POIN}

start = “TOON”

target = “PLEA”

Output: 7

TOON -> POON –> POIN –> POIE –> PLIE –> PLEE –> PLEA

Code and Output:

Time Complexity:

5. Course schedule :

There are a total of N tasks, labeled from 0 to N-1. Some tasks may have prerequisites, for example to do task 0 you have to first complete task 1, which is expressed as a pair: [0, 1]. Given the total number of tasks N and a list of prerequisite pairs P, find if it is possible to finish all tasks.

Input: N = 4, P = 3, prerequisites = {{1,0},{2,1},{3,2}}

Output: Yes

Explanation: To do task 1 you should have completed task 0, and to do task 2 you should have finished task 1, and to do task 3 you should have finished task 2. So it is possible.

Input: N = 2, P = 2, prerequisites = {{1,0},{0,1}}

Output: No

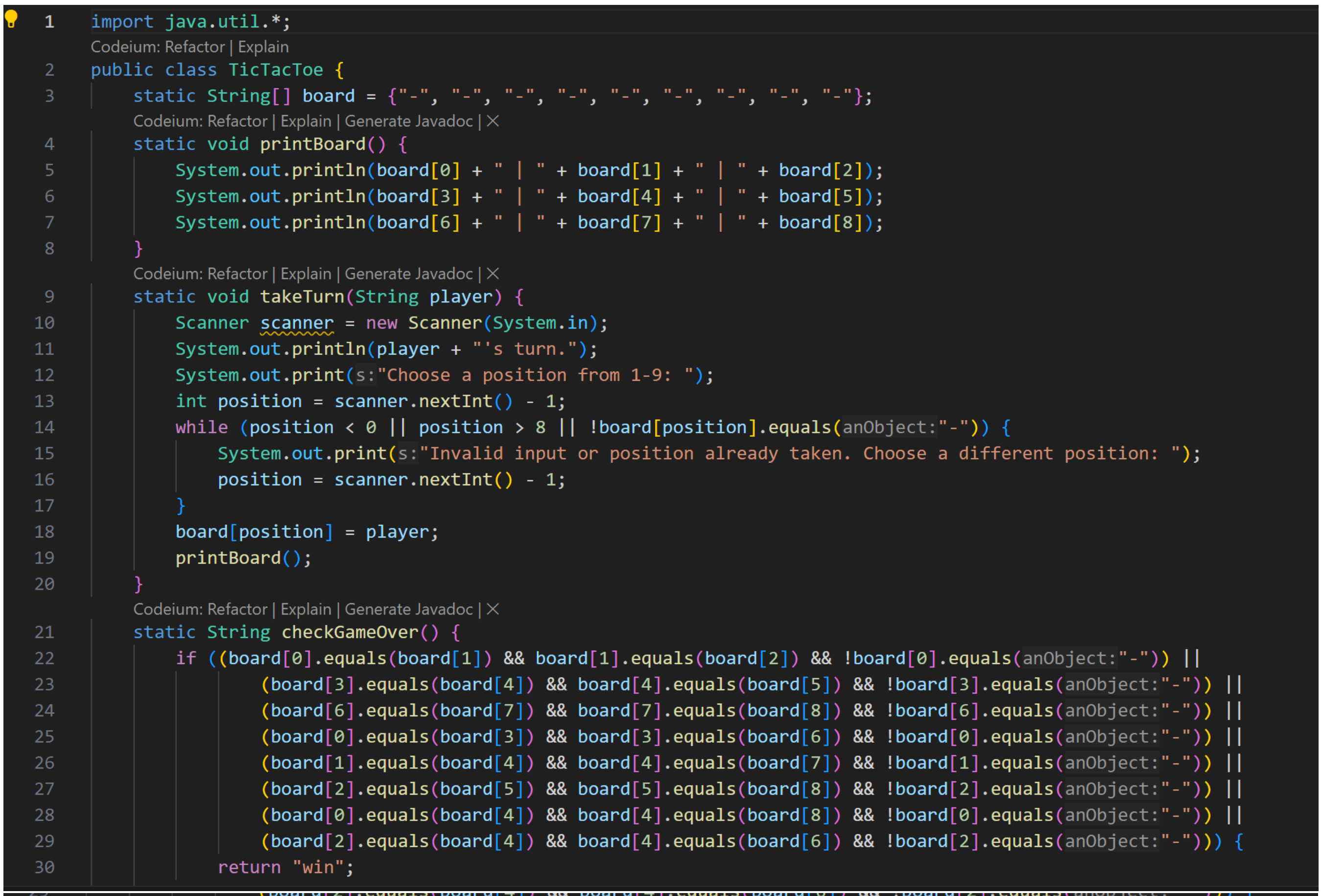
Explanation: To do task 1 you should have completed task 0, and to do task 0 you should have finished task 1. So it is impossible.

Code and Output:

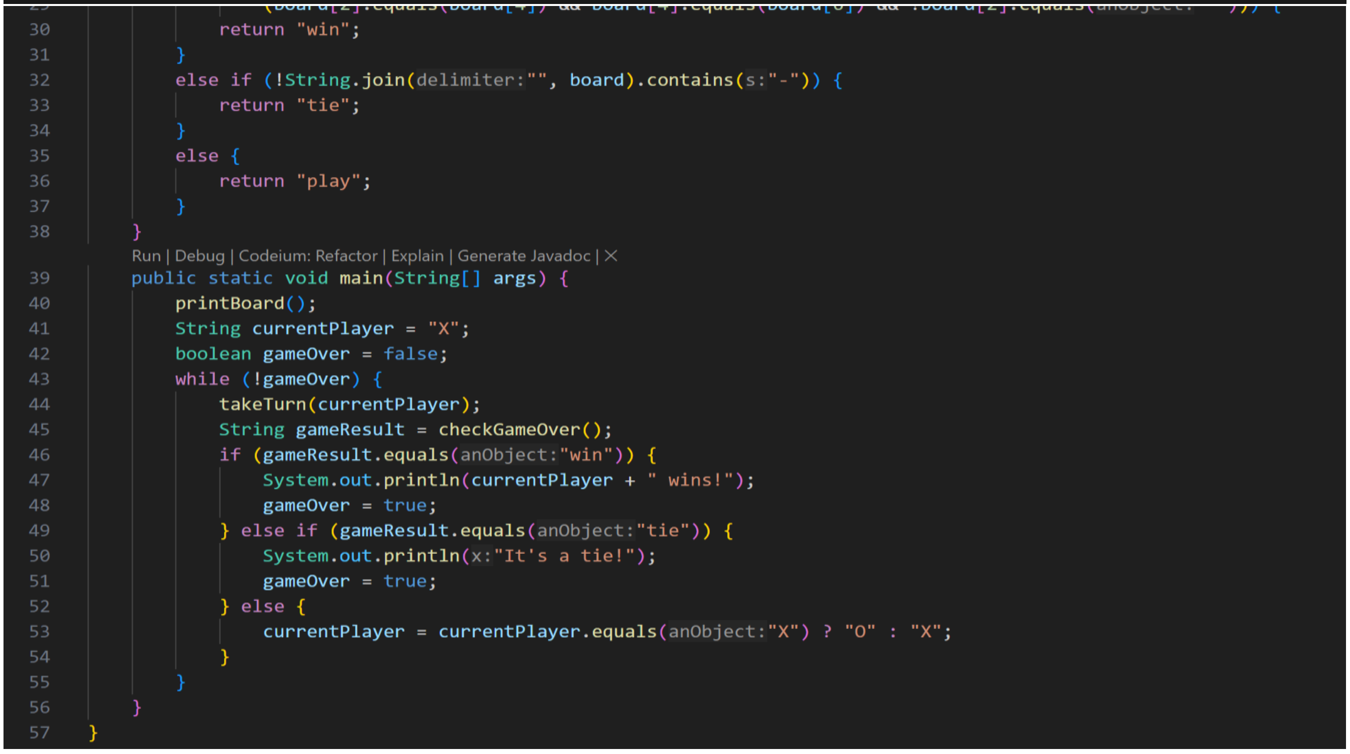
Time Complexity:

6. Design tic tac toe :

Code :

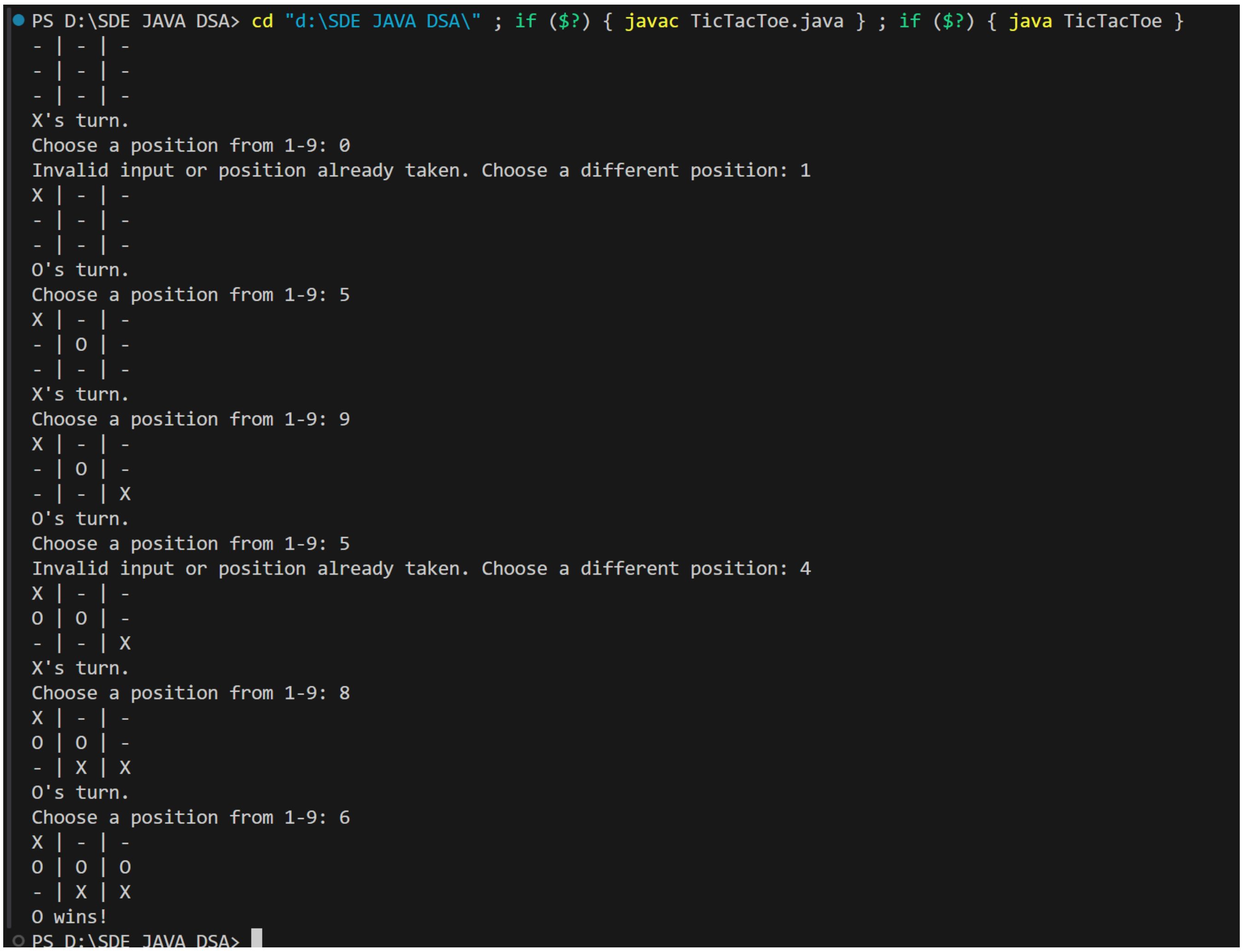

Codeium: Refactor | Explain
Codeium: Refactor | Explain | Generate Javadoc | ×

```java
import java.util.*;
public class TicTacToe {
    static String[] board = {"-", "-", "-", "-", "-", "-", "-", "-", "-"};
    static void printBoard() {
        System.out.println(board[0] + " | " + board[1] + " | " + board[2]);
        System.out.println(board[3] + " | " + board[4] + " | " + board[5]);
        System.out.println(board[6] + " | " + board[7] + " | " + board[8]);
    }
    static void takeTurn(String player) {
        Scanner scanner = new Scanner(System.in);
        System.out.println(player + "'s turn.");
        System.out.print(s:"Choose a position from 1-9: ");
        int position = scanner.nextInt() - 1;
        while (position < 0 || position > 8 || !board[position].equals(anObject:"-")) {
            System.out.print(s:"Invalid input or position already taken. Choose a different position: ");
            position = scanner.nextInt() - 1;
        }
        board[position] = player;
        printBoard();
    }
    static String checkGameOver() {
        if ((board[0].equals(board[1]) && board[1].equals(board[2]) && !board[0].equals(anObject:"-")) ||
                (board[3].equals(board[4]) && board[4].equals(board[5]) && !board[3].equals(anObject:"-")) ||
                (board[6].equals(board[7]) && board[7].equals(board[8]) && !board[6].equals(anObject:"-")) ||
                (board[0].equals(board[3]) && board[3].equals(board[6]) && !board[0].equals(anObject:"-")) ||
                (board[1].equals(board[4]) && board[4].equals(board[7]) && !board[1].equals(anObject:"-")) ||
                (board[2].equals(board[5]) && board[5].equals(board[8]) && !board[2].equals(anObject:"-")) ||
                (board[0].equals(board[4]) && board[4].equals(board[8]) && !board[0].equals(anObject:"-")) ||
                (board[2].equals(board[4]) && board[4].equals(board[6]) && !board[2].equals(anObject:"-"))) {
            return "win";
```


Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | ×

```java
            return "win";
        }
        else if (!String.join(delimiter:"", board).contains(s:"-")) {
            return "tie";
        }
        else {
            return "play";
        }
    }
    public static void main(String[] args) {
        printBoard();
        String currentPlayer = "X";
        boolean gameOver = false;
        while (!gameOver) {
            takeTurn(currentPlayer);
            String gameResult = checkGameOver();
            if (gameResult.equals(anObject:"win")) {
                System.out.println(currentPlayer + " wins!");
                gameOver = true;
            } else if (gameResult.equals(anObject:"tie")) {
                System.out.println(x:"It's a tie!");
                gameOver = true;
            } else {
                currentPlayer = currentPlayer.equals(anObject:"X") ? "O" : "X";
            }
        }
    }
}
```

Output:



```
PS D:\SDE JAVA DSA> cd "d:\SDE JAVA DSA\" ; if ($?) { javac TicTacToe.java } ; if ($?) { java TicTacToe }
- | - | -
- | - | -
- | - | -
X's turn.
Choose a position from 1-9: 0
Invalid input or position already taken. Choose a different position: 1
X | - | -
- | - | -
- | - | -
O's turn.
Choose a position from 1-9: 5
X | - | -
- | O | -
- | - | -
X's turn.
Choose a position from 1-9: 9
X | - | -
- | O | -
- | - | X
O's turn.
Choose a position from 1-9: 5
Invalid input or position already taken. Choose a different position: 4
X | - | -
O | O | -
- | - | X
X's turn.
Choose a position from 1-9: 8
X | - | -
O | O | -
- | X | X
O's turn.
Choose a position from 1-9: 6
X | - | -
O | O | O
- | X | X
O wins!
PS D:\SDE JAVA DSA>
```

Time Complexity: O(1)

7. Remove Linkedlist:

```java
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode ans = new ListNode(0, head);
        ListNode dummy = ans;

        while (dummy != null) {
            while (dummy.next != null && dummy.next.val == val) {
                dummy.next = dummy.next.next;
            }
            dummy = dummy.next;
        }

        return ans.next;
    }
}
```

Time Complexity : O(N)

8. Buy and Sell Stock – II:

```java
class Solution {
    public int maxProfit(int[] prices) {
        int profit = 0;

        for (int i = 1; i < prices.length; i++) {
            if (prices[i] > prices[i - 1]) {
                profit += prices[i] - prices[i - 1];
            }
        }

        return profit;
    }
}
```

Time Complexity : O(N)

9. Number of Islands:

```java
class Solution {
    public int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0) {
            return 0;
        }

        int row = grid.length;
        int col = grid[0].length;
        int count = 0;

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                if (grid[i][j] == '1') {
                    count++;
                    makeZero(i, j, row, col, grid);
                }
            }
        }
        return count;
    }

    private void makeZero(int i, int j, int row, int col, char[][] grid) {
        if (i < 0 || j < 0 || i >= row || j >= col || grid[i][j] == '0') {
            return;
        }

        grid[i][j] = '0';
        makeZero(i + 1, j, row, col, grid);
        makeZero(i - 1, j, row, col, grid);
        makeZero(i, j + 1, row, col, grid);
        makeZero(i, j - 1, row, col, grid);
    }
}
```

Time Complexity : O(M*N)

10. Group Anagrams:

```java
import java.util.*;

class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        Map<String, List<String>> anagramMap = new HashMap<>();

        for (String word : strs) {
            char[] charArray = word.toCharArray();
            Arrays.sort(charArray);
            String sortedWord = new String(charArray);

            anagramMap.putIfAbsent(sortedWord, new ArrayList<>());
            anagramMap.get(sortedWord).add(word);
        }

        return new ArrayList<>(anagramMap.values());
    }
}
```

Time Complexity : O(k*NlogN)

11. 3Sum Closest:

```java
import java.util.Arrays;
class Solution {
    public int threeSumClosest(int[] nums, int target) {
        Arrays.sort(nums);
        int closest = nums[0] + nums[1] + nums[2];
        for (int i = 0; i < nums.length - 2; i++) {
            int left = i + 1, right = nums.length - 1;
            while (left < right) {
                int currSum = nums[i] + nums[left] + nums[right];
                if (currSum == target) {
                    return target;
                }
                if (Math.abs(currSum - target) < Math.abs(closest - target)) {
                    closest = currSum;
                }
                if (currSum < target) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return closest;
    }
}
```

Time Complexity: O(N^2)

12. Jump Game – II:

```
class Solution {
    public int jump(int[] nums) {
        int near = 0, far = 0, jumps = 0;

        while (far < nums.length - 1) {
            int farthest = 0;
            for (int i = near; i <= far; i++) {
                farthest = Math.max(farthest, i + nums[i]);
            }
            near = far + 1;
            far = farthest;
            jumps++;
        }

        return jumps;
    }
}
```

Time Complexity : O(N)

13. Decode Ways:

```
class Solution {
    public int numDecodings(String s) {
        int strLen = s.length();
        int[] dp = new int[strLen + 1];
        dp[0] = 1;
        if (s.charAt(0) != '0') {
            dp[1] = 1;
        } else {
            return 0;
        }
        for (int i = 2; i <= strLen; ++i) {
            if (s.charAt(i - 1) != '0') {
                dp[i] += dp[i - 1];
            }
            if (s.charAt(i - 2) == '1' ||
                    (s.charAt(i - 2) == '2' && s.charAt(i - 1) <= '6')) {
                dp[i] += dp[i - 2];
            }
        }
        return dp[strLen];
    }
}
```

Time Complexity : O(N)

14. **Longest Substring Without Repeating Characters** :

```java
import java.util.HashSet;
public class Solution {
    public int lengthOfLongestSubstring(String s) {
        HashSet<Character> unique = new HashSet<>();
        int start = 0, end = 0, maxi = 0;
        while (start < s.length()) {
            if (!unique.contains(s.charAt(start))) {
                unique.add(s.charAt(start));
                maxi = Math.max(maxi, start - end + 1);
            } else {
                while (unique.contains(s.charAt(start))) {
                    unique.remove(s.charAt(end));
                    end++;
                }
                unique.add(s.charAt(start));
            }
            start++;
        }
        return maxi;
    }
    public static void main(String[] args) {
        Solution solution = new Solution();
        System.out.println(solution.lengthOfLongestSubstring("abcabcbb"));
        System.out.println(solution.lengthOfLongestSubstring("bbbbb"));
        System.out.println(solution.lengthOfLongestSubstring("pwwkew"));
    }
}
```

Time Complexity : O(N)