**ADVANCED PROGRAMMING-II**

**ASSIGNMENT-09**

**UID: 22BCS11558 SECTION:614-B**

**NAME-NIKHIL DATE:16-04-25**

**Q1. Set Matrix Zeroes:**

**Code:**

class Solution {

public:

    void setZeroes(vector<vector<int>>& matrix) {

        int m = matrix.size();

        int n = matrix[0].size();

        bool firstRowZero = false, firstColZero = false;

        for (int j = 0; j < n; j++) {

            if (matrix[0][j] == 0) {

                firstRowZero = true;

                break;

            }

        }

        for (int i = 0; i < m; i++) {

            if (matrix[i][0] == 0) {

                firstColZero = true;

                break;

            }

        }

        for (int i = 1; i < m; i++) {

            for (int j = 1; j < n; j++) {

                if (matrix[i][j] == 0) {

                    matrix[i][0] = 0;

                    matrix[0][j] = 0;

                }

            }

        }

        for (int i = 1; i < m; i++) {

            for (int j = 1; j < n; j++) {

                if (matrix[i][0] == 0 || matrix[0][j] == 0)

                    matrix[i][j] = 0;

            } }

        if (firstRowZero) {

            for (int j = 0; j < n; j++)

                matrix[0][j] = 0;

        }

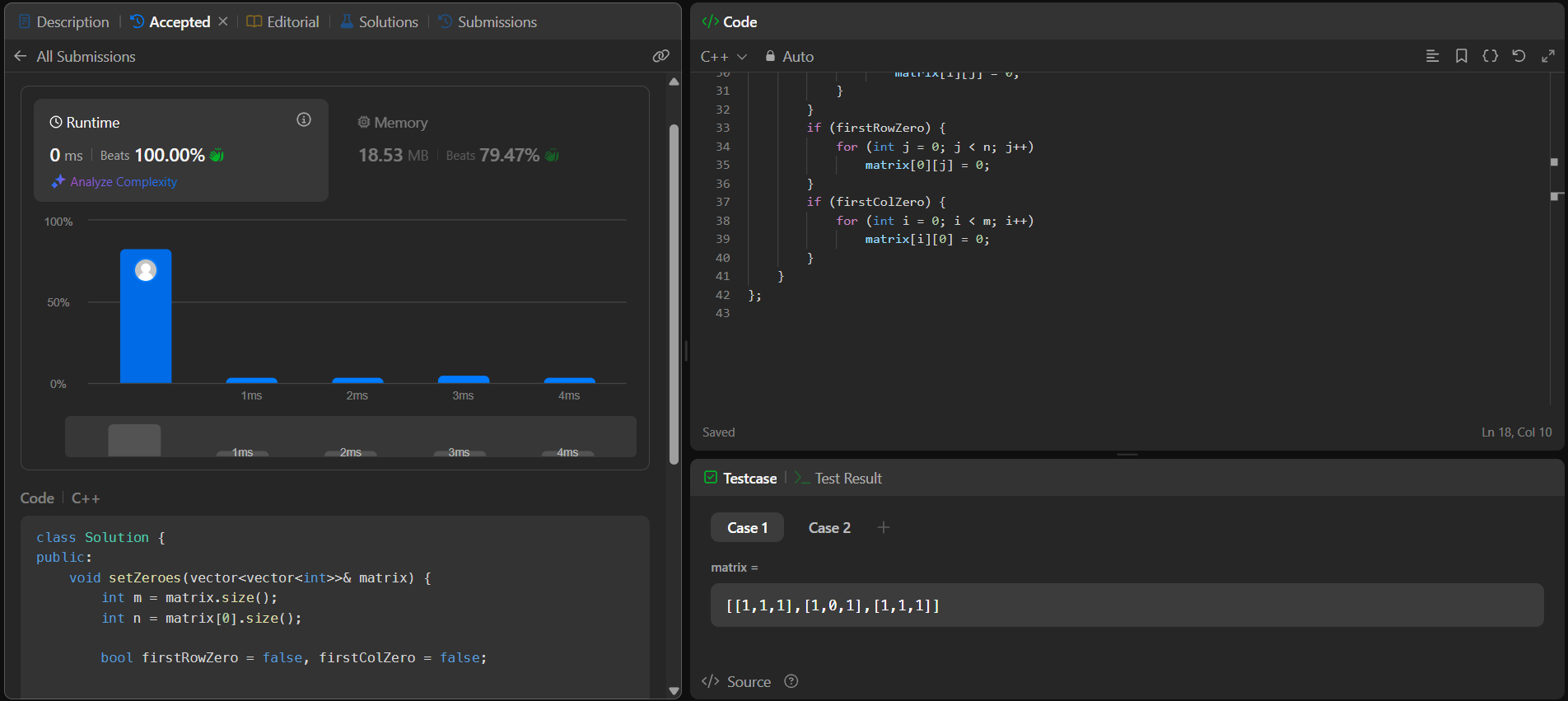
        if (firstColZero) {

            for (int i = 0; i < m; i++)

                matrix[i][0] = 0;

        }}};

**Screenshot:**



**Q2. Reverse Linked List II:**

**Code:**

class Solution {

public:

    ListNode\* reverseBetween(ListNode\* head, int left, int right) {

        if (!head || left == right) return head;

        ListNode dummy(0);

        dummy.next = head;

        ListNode\* prev = &dummy;

        for (int i = 1; i < left; ++i) {

            prev = prev->next;

        }

        ListNode\* start = prev->next;

        ListNode\* then = start->next;

        for (int i = 0; i < right - left; ++i) {

            start->next = then->next;

            then->next = prev->next;

            prev->next = then;

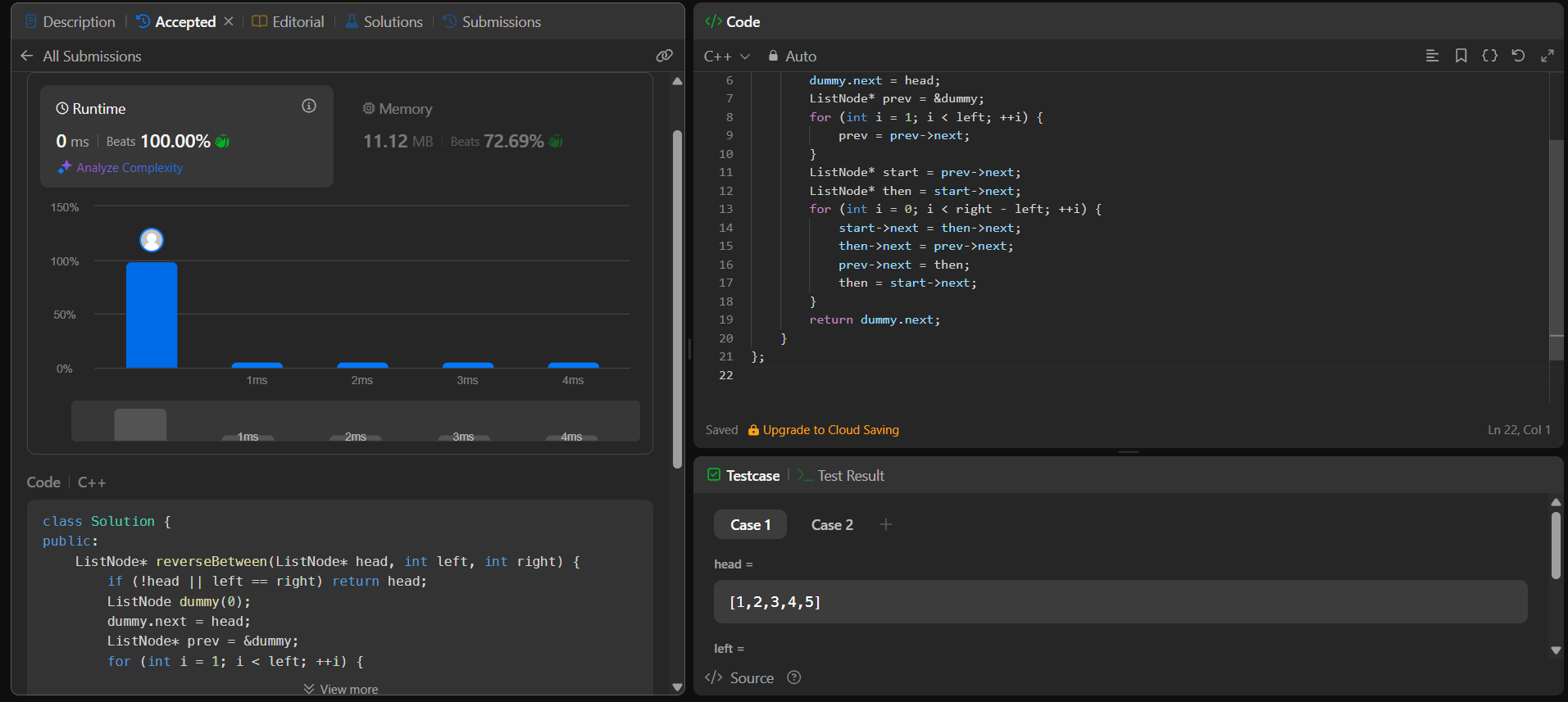
            then = start->next;

        }

        return dummy.next;

    }

};

**Screenshot:**

**Q3.** **Search a 2D Matrix II:**

**Code:**

class Solution {

public:

    bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int m = matrix.size(), n = matrix[0].size();

        int row = 0, col = n - 1;

        while (row < m && col >= 0) {

            if (matrix[row][col] == target) return true;

            else if (matrix[row][col] > target) col--;

            else row++;

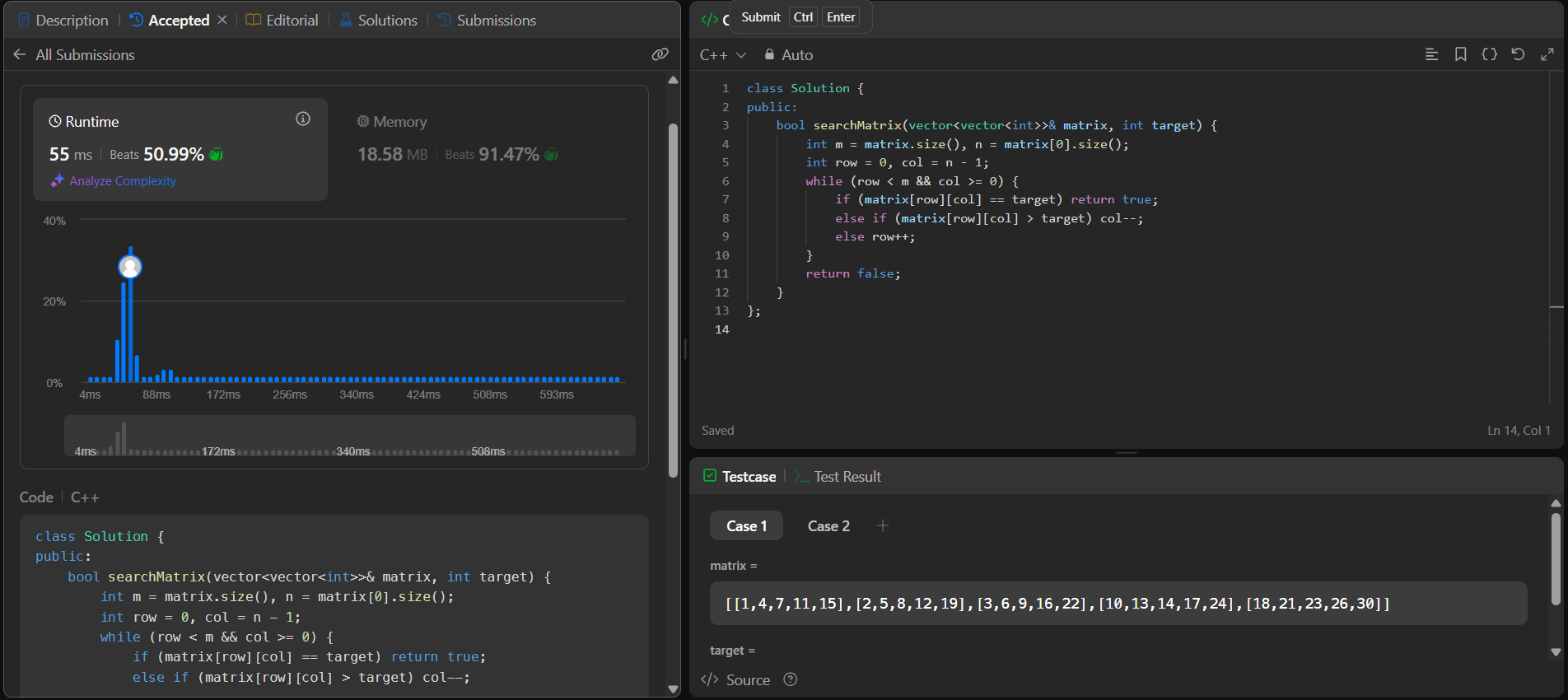
        }

        return false;

    }

};

**Screenshot:**



**Q4. Word Break:**

**Code:**

class Solution {

public:

    bool wordBreak(string s, vector<string>& wordDict) {

        unordered\_set<string> dict(wordDict.begin(), wordDict.end());

        vector<bool> dp(s.length() + 1, false);

        dp[0] = true;

        for (int i = 1; i <= s.length(); ++i) {

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

                if (dp[j] && dict.count(s.substr(j, i - j))) {

                    dp[i] = true;

                    break;

                }

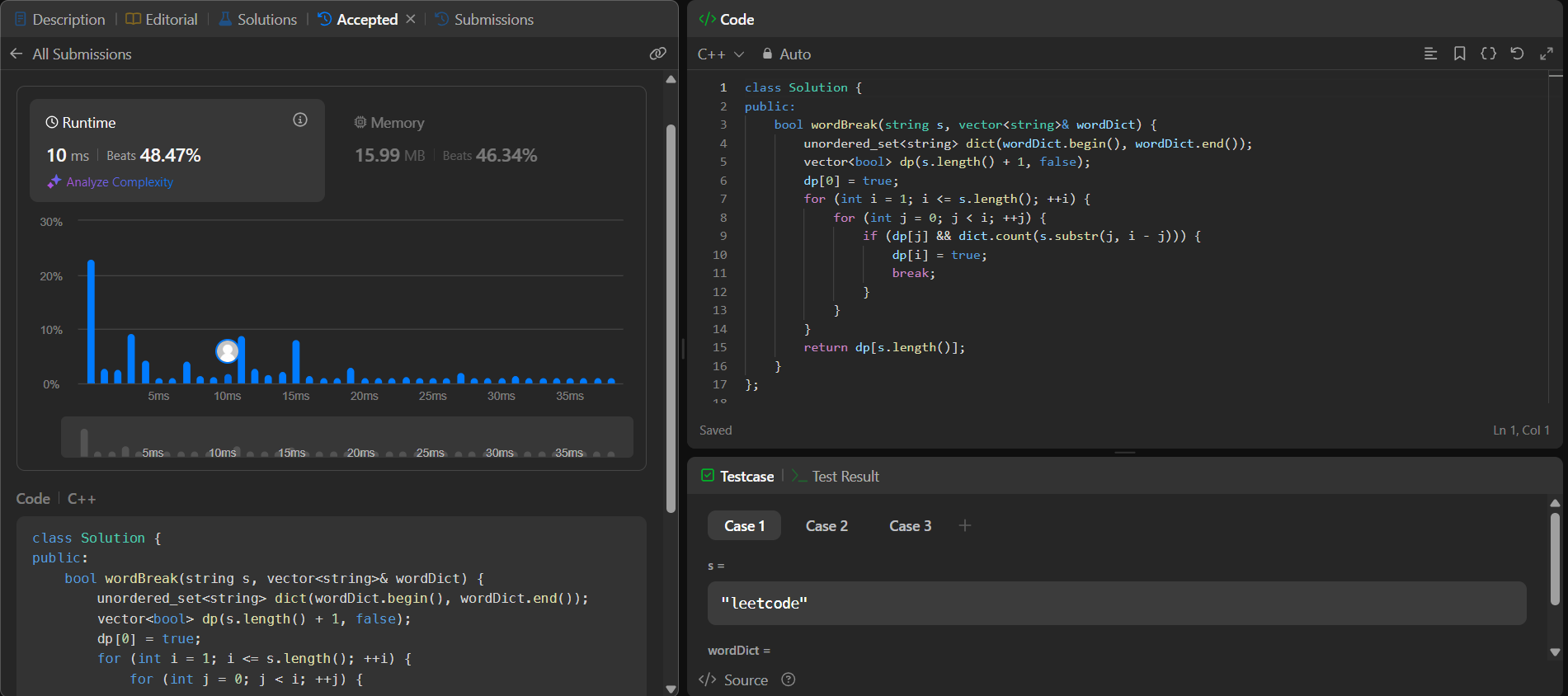
            }

        }

        return dp[s.length()];

    }};

**Screenshot:**



**Q5. Longest Increasing Path in a Matrix:**

**Code:**

class Solution {

public:

    int m, n;

    vector<vector<int>> dirs = {{0,1}, {1,0}, {0,-1}, {-1,0}};

    int dfs(int i, int j, vector<vector<int>>& matrix, vector<vector<int>>& dp) {

        if (dp[i][j] != 0) return dp[i][j];

        int maxLen = 1;

        for (auto& d : dirs) {

            int x = i + d[0], y = j + d[1];

            if (x >= 0 && x < m && y >= 0 && y < n && matrix[x][y] > matrix[i][j]) {

                maxLen = max(maxLen, 1 + dfs(x, y, matrix, dp));

            }

        }

        return dp[i][j] = maxLen;

    }

    int longestIncreasingPath(vector<vector<int>>& matrix) {

        m = matrix.size();

        n = matrix[0].size();

        vector<vector<int>> dp(m, vector<int>(n, 0));

        int result = 0;

        for (int i = 0; i < m; ++i) {

            for (int j = 0; j < n; ++j) {

                result = max(result, dfs(i, j, matrix, dp));

            }   }

        return result;

  }};

**Screenshot:**

