

**NAME : Anish Patial**

**UID : 22BCS15029**

**SECTION : FL\_IoT 601 'A'**

**Ap lab assignment**

1. [Set Matrix Zeroes](#)

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        const int m = matrix.size();
        const int n = matrix[0].size();
        bool shouldFillFirstRow = false;
        bool shouldFillFirstCol = false;

        for (int j = 0; j < n; ++j)
            if (matrix[0][j] == 0) {
                shouldFillFirstRow = true;
                break;
            }

        for (int i = 0; i < m; ++i)
            if (matrix[i][0] == 0) {
                shouldFillFirstCol = 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 (shouldFillFirstRow)
        for (int j = 0; j < n; ++j)
            matrix[0][j] = 0;
    if (shouldFillFirstCol)
        for (int i = 0; i < m; ++i)
            matrix[i][0] = 0;
}
};

```

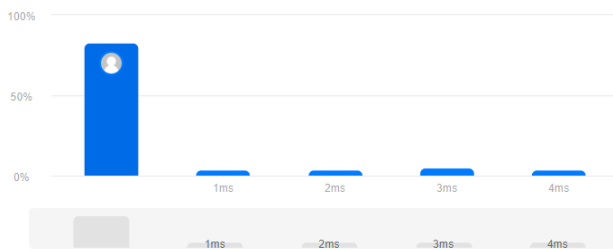
← All Submissions

C++ Auto

Accepted 202 / 202 testcases passed  
 Anish Patil submitted at Apr 06, 2025 23:11

Editorial Solution

Runtime 0 ms | Beats 100.00%  
 Memory 18.60 MB | Beats 53.62%



Code | C++

```
class Solution {
```

```

23         matrix[i][0] = 0;
24         matrix[0][j] = 0;
25     }
26     for (int i = 1; i < m; ++i)
27         for (int j = 1; j < n; ++j)
28             if (matrix[i][0] == 0 || matrix[0][j] == 0)
29                 matrix[i][j] = 0;
30     if (shouldFillFirstRow)
31         for (int j = 0; j < n; ++j)
32             matrix[0][j] = 0;
33     if (shouldFillFirstCol)
34         for (int i = 0; i < m; ++i)
35             matrix[i][0] = 0;
36 }
37 };
38

```

Saved

Testcase Test Result

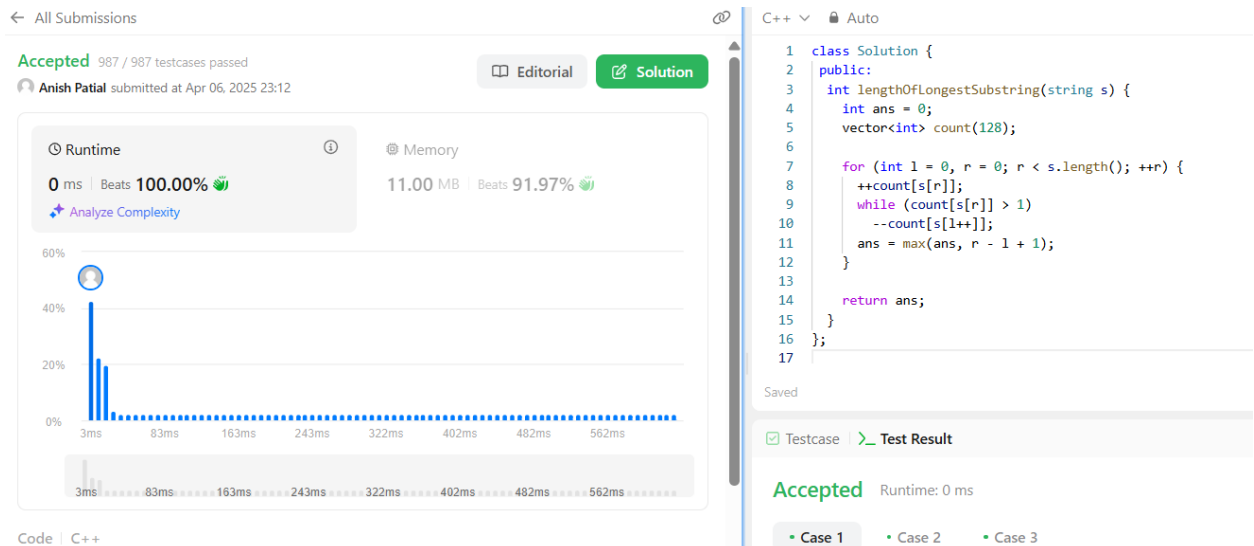
Accepted Runtime: 0 ms

Case 1 Case 2

Input

## 2. Longest Substring Without Repeating Characters

```
class Solution {  
  
public:  
  
    int lengthOfLongestSubstring(string s) {  
  
        int ans = 0;  
  
        vector<int> count(128);  
  
  
        for (int l = 0, r = 0; r < s.length(); ++r) {  
  
            ++count[s[r]];  
  
            while (count[s[r]] > 1)  
                --count[s[l++]];  
  
            ans = max(ans, r - l + 1);  
  
        }  
  
        return ans;  
  
    }  
  
};
```



### 3. [Reverse Linked List II](#)

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (left == 1)
            return reverseN(head, right);

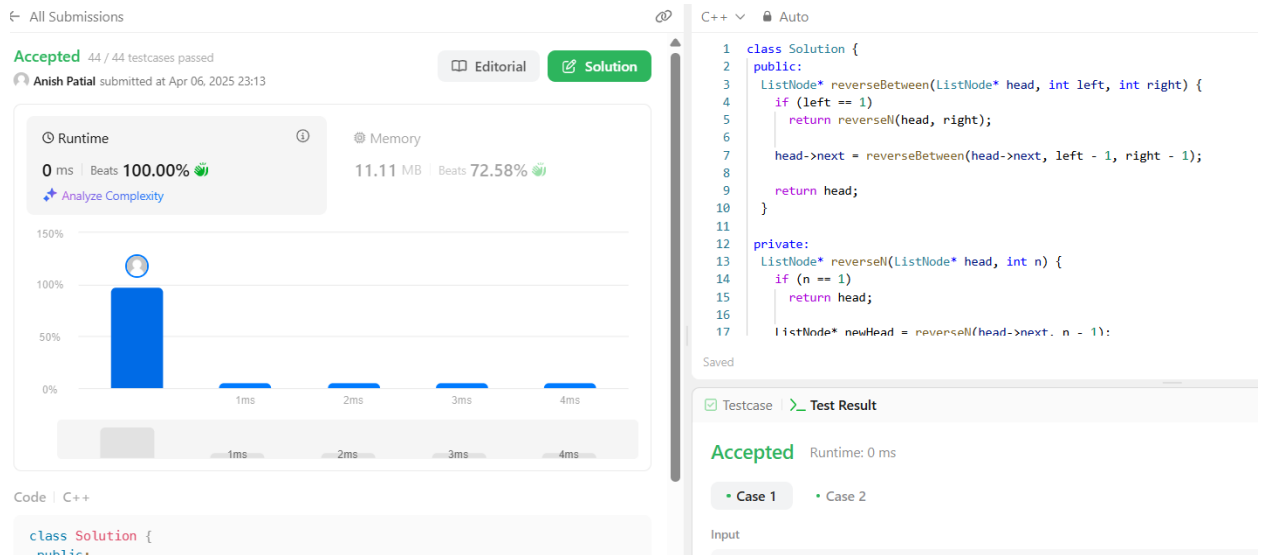
        head->next = reverseBetween(head->next, left - 1, right - 1);

        return head;
    }

private:
    ListNode* reverseN(ListNode* head, int n) {
        if (n == 1)
            return head;

        ListNode* newHead = reverseN(head->next, n - 1);
        ListNode* headNext = head->next;
        head->next = headNext->next;
        headNext->next = head;

        return newHead;
    }
};
```



#### 4. Linked List Cycle

```
class Solution {
public:
    bool hasCycle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;

        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast)
                return true;
        }

        return false;
    }
};
```

Accepted 29 / 29 testcases passed  
Anish Patil submitted at Apr 06, 2025 23:16

Editorial

Solution

Runtime

13 ms | Beats 17.73%

Analyze Complexity

Memory

11.81 MB | Beats 53.40%



Code | C++

```
1 class Solution {
2 public:
3     bool hasCycle(ListNode* head) {
4         ListNode* slow = head;
5         ListNode* fast = head;
6
7         while (fast != nullptr && fast->next != nullptr) {
8             slow = slow->next;
9             fast = fast->next->next;
10            if (slow == fast)
11                return true;
12        }
13
14        return false;
15    }
16 };
17
```

Saved

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

## 5. The Skyline Problem

```
class Solution {
public:
    vector<vector<int>> getSkyline(const vector<vector<int>>& buildings) {
        const int n = buildings.size();
        if (n == 0)
            return {};
        if (n == 1) {
            const int left = buildings[0][0];
            const int right = buildings[0][1];
            const int height = buildings[0][2];
            return {{left, height}, {right, 0}};
        }

        const vector<vector<int>> left =
            getSkyline({buildings.begin(), buildings.begin() + n / 2});
        const vector<vector<int>> right =
            getSkyline({buildings.begin() + n / 2, buildings.end()});
        return merge(left, right);
    }
};
```

private:

```
vector<vector<int>> merge(const vector<vector<int>>& left,  
                        const vector<vector<int>>& right) {
```

```
    vector<vector<int>> ans;
```

```
    int i = 0;
```

```
    int j = 0;
```

```
    int leftY = 0;
```

```
    int rightY = 0;
```

```
    while (i < left.size() && j < right.size())
```

```
        if (left[i][0] < right[j][0]) {
```

```
            leftY = left[i][1];
```

```
            addPoint(ans, left[i][0], max(left[i++][1], rightY));
```

```
        } else {
```

```
            rightY = right[j][1];
```

```
            addPoint(ans, right[j][0], max(right[j++][1], leftY));
```

```
        }
```

```
    while (i < left.size())
```

```
        addPoint(ans, left[i][0], left[i++][1]);
```

```
    while (j < right.size())
```

```
        addPoint(ans, right[j][0], right[j++][1]);
```

```
    return ans;
```

```
}
```

```
void addPoint(vector<vector<int>>& ans, int x, int y) {
```

```
    if (!ans.empty() && ans.back()[0] == x) {
```

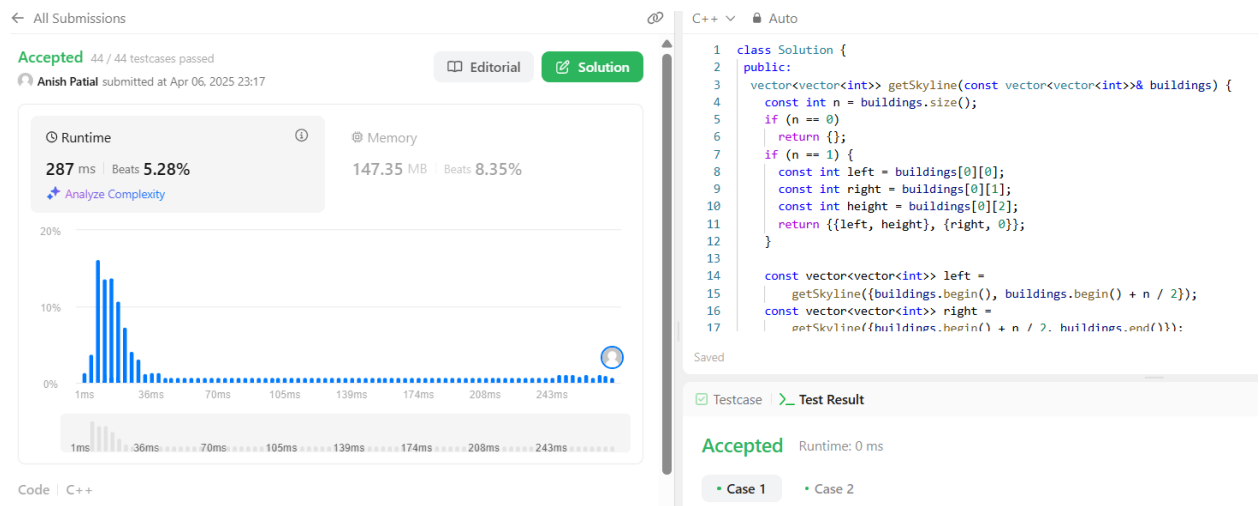
```
        ans.back()[1] = y;
```

```
    return;
```

```

    }
    if (!ans.empty() && ans.back()[1] == y)
        return;
    ans.push_back({x, y});
}
};

```



## 6. Longest Increasing Subsequence II

```

struct SegmentTreeNode {
    int lo;
    int hi;
    int maxLength;
    std::unique_ptr<SegmentTreeNode> left;
    std::unique_ptr<SegmentTreeNode> right;
    SegmentTreeNode(int lo, int hi, int maxLength,
        std::unique_ptr<SegmentTreeNode> left = nullptr,
        std::unique_ptr<SegmentTreeNode> right = nullptr)
        : lo(lo),
        hi(hi),
        maxLength(maxLength),
        left(std::move(left)),
        right(std::move(right)) {}
};

```

```

class SegmentTree {

```



```

public:
    explicit SegmentTree() : root(make_unique<SegmentTreeNode>(0, 1e5 + 1, 0)) {}

    void updateRange(int i, int j, int maxLength) {
        update(root, i, j, maxLength);
    }
    int queryRange(int i, int j) {
        return query(root, i, j);
    }

private:
    std::unique_ptr<SegmentTreeNode> root;

    void update(std::unique_ptr<SegmentTreeNode>& root, int i, int j,
                int maxLength) {
        if (root->lo == i && root->hi == j) {
            root->maxLength = maxLength;
            root->left = nullptr;
            root->right = nullptr;
            return;
        }
        const int mid = root->lo + (root->hi - root->lo) / 2;
        if (root->left == nullptr) {
            root->left = make_unique<SegmentTreeNode>(root->lo, mid, root->maxLength);
            root->right =
                make_unique<SegmentTreeNode>(mid + 1, root->hi, root->maxLength);
        }
        if (j <= mid)
            update(root->left, i, j, maxLength);
        else if (i > mid)
            update(root->right, i, j, maxLength);
        else {
            update(root->left, i, mid, maxLength);
            update(root->right, mid + 1, j, maxLength);
        }
        root->maxLength = merge(root->left->maxLength, root->right->maxLength);
    }

```

```

int query(std::unique_ptr<SegmentTreeNode>& root, int i, int j) {
    if (root->left == nullptr)
        return root->maxLength;
    if (root->lo == i && root->hi == j)
        return root->maxLength;
    const int mid = root->lo + (root->hi - root->lo) / 2;
    if (j <= mid)
        return query(root->left, i, j);
    if (i > mid)
        return query(root->right, i, j);
    return merge(query(root->left, i, mid), query(root->right, mid + 1, j));
}

```

```

int merge(int left, int right) const {
    return max(left, right);
};
};

```

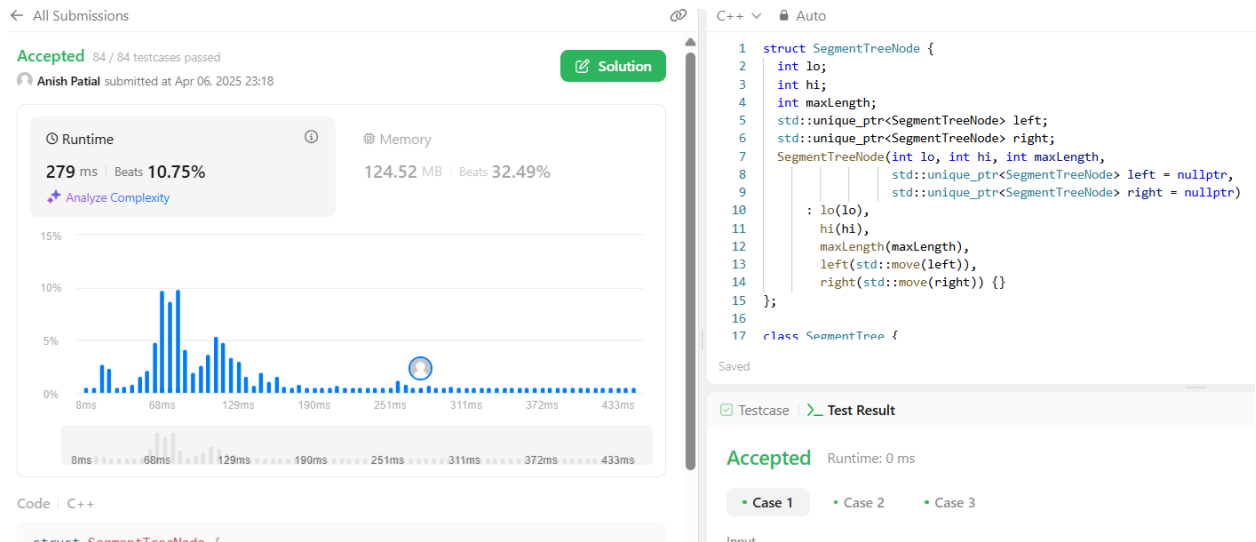
```

class Solution {
public:
    int lengthOfLIS(vector<int>& nums, int k) {
        int ans = 1;
        SegmentTree tree;

        for (const int num : nums) {
            const int left = max(1, num - k);
            const int right = num - 1;
            const int maxLength = tree.queryRange(left, right) + 1;
            ans = max(ans, maxLength);
            tree.updateRange(num, num, maxLength);
        }

        return ans;
    }
};

```



## 7. [Search a 2D Matrix II](#)

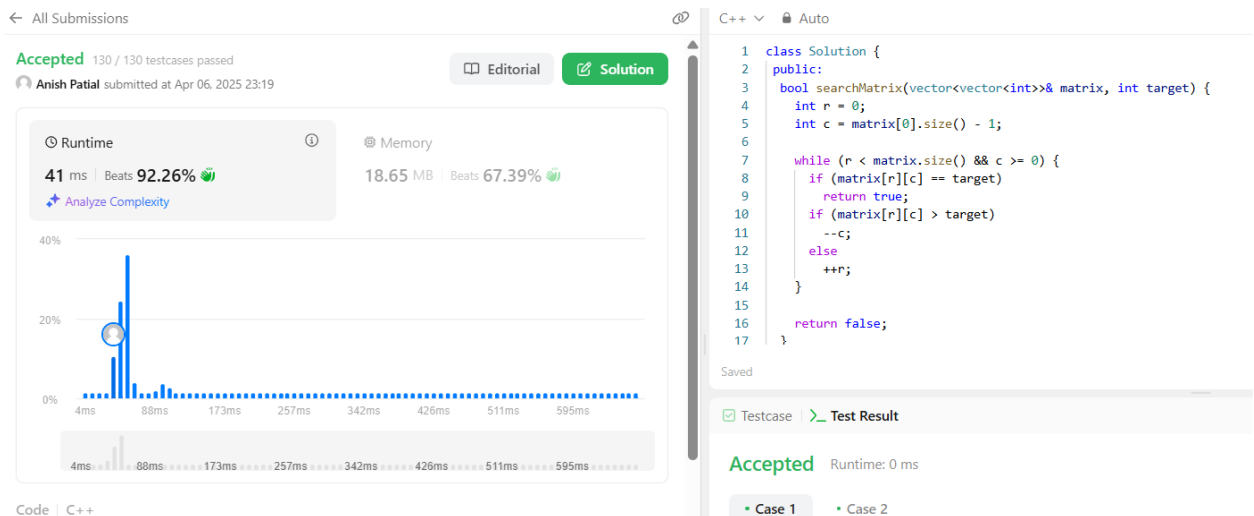
```

class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int r = 0;
        int c = matrix[0].size() - 1;

        while (r < matrix.size() && c >= 0) {
            if (matrix[r][c] == target)
                return true;
            if (matrix[r][c] > target)
                --c;
            else
                ++r;
        }

        return false;
    }
};

```



## 8. Word Break

```
class Solution {
```

```
public:
```

```
    bool wordBreak(string s, vector<string>& wordDict) {
        return wordBreak(s, {wordDict.begin(), wordDict.end()}, {});
    }

```

```
private:
```

```
    bool wordBreak(const string& s, const unordered_set<string>&& wordSet,
                   unordered_map<string, bool>&& mem) {
        if (wordSet.contains(s))
            return true;
        if (const auto it = mem.find(s); it != mem.cend())
            return it->second;
        for (int i = 1; i < s.length(); ++i) {
            const string& prefix = s.substr(0, i);
            const string& suffix = s.substr(i);
            if (wordSet.contains(prefix) &&

```

```

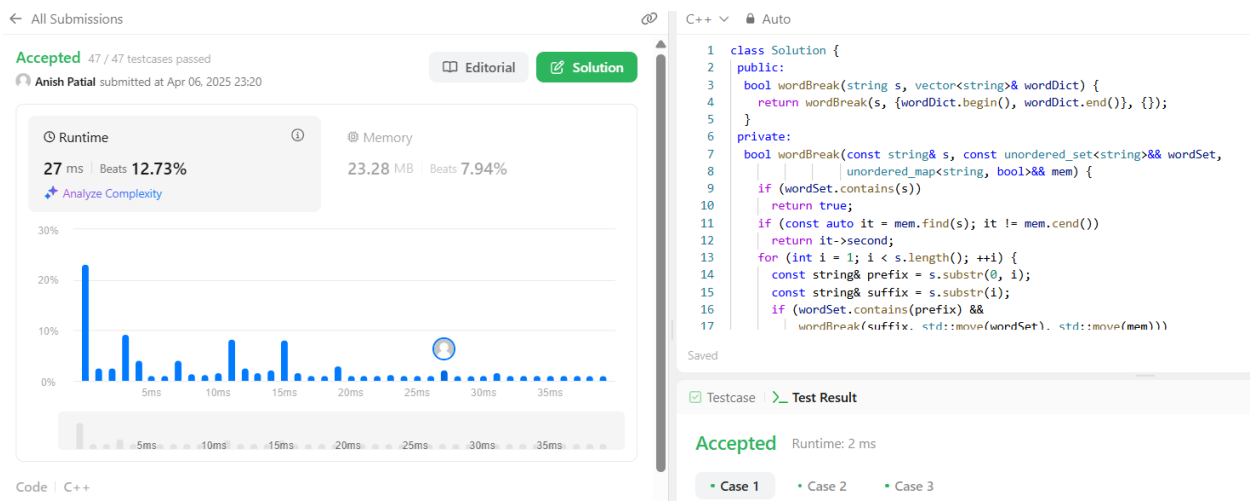
        wordBreak(suffix, std::move(wordSet), std::move(mem)))

    return mem[s] = true;
}

return mem[s] = false;
}

};

```



## 9. Longest Increasing Path in a Matrix

```

class Solution {
public:
    int longestIncreasingPath(vector<vector<int>>& matrix) {
        const int m = matrix.size();
        const int n = matrix[0].size();
        int ans = 0;
        vector<vector<int>> mem(m, vector<int>(n));

        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
                ans = max(ans, dfs(matrix, i, j, INT_MIN, mem));
    }
};

```

```

        return ans;
    }
private:
int dfs(const vector<vector<int>>& matrix, int i, int j, int prev,
        vector<vector<int>>& mem) {
    if (i < 0 || i == matrix.size() || j < 0 || j == matrix[0].size())
        return 0;
    if (matrix[i][j] <= prev)
        return 0;
    int& ans = mem[i][j];
    if (ans > 0)
        return ans;

    const int curr = matrix[i][j];
    return ans = 1 + max({dfs(matrix, i + 1, j, curr, mem),
                        dfs(matrix, i - 1, j, curr, mem),
                        dfs(matrix, i, j + 1, curr, mem),
                        dfs(matrix, i, j - 1, curr, mem)});
}
};

```

← All Submissions

Accepted 139 / 139 testcases passed  
 Anish Patil submitted at Apr 06, 2025 23:21

Editorial Solution

Runtime 16 ms | Beats 42.91%  
 Memory 21.90 MB | Beats 42.14%

Analyze Complexity

Code C++

```

1 class Solution {
2 public:
3     int longestIncreasingPath(vector<vector<int>>& matrix) {
4         const int m = matrix.size();
5         const int n = matrix[0].size();
6         int ans = 0;
7         vector<vector<int>> mem(m, vector<int>(n));
8
9         for (int i = 0; i < m; ++i)
10             for (int j = 0; j < n; ++j)
11                 ans = max(ans, dfs(matrix, i, j, INT_MIN, mem));
12         return ans;
13     }
14 private:
15     int dfs(const vector<vector<int>>& matrix, int i, int j, int prev,
16             vector<vector<int>>& mem) {
17         if (i < 0 || i == matrix.size() || j < 0 || j == matrix[0].size())

```

Saved

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

## 10. Trapping Rain Water

```
class Solution {
public:
    int trap(vector<int>& height) {
        const int n = height.size();
        int ans = 0;
        vector<int> l(n);
        vector<int> r(n);

        for (int i = 0; i < n; ++i)
            l[i] = i == 0 ? height[i] : max(height[i], l[i - 1]);

        for (int i = n - 1; i >= 0; --i)
            r[i] = i == n - 1 ? height[i] : max(height[i], r[i + 1]);

        for (int i = 0; i < n; ++i)
            ans += min(l[i], r[i]) - height[i];

        return ans;
    }
};
```

