leetcode.com/problems/number-of-provinces/

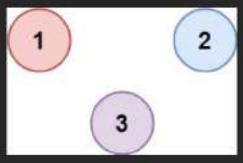
Gmail YouTube Maps Disney+ Hotstar -... Tata Play All Bookmarks

Problem List Run Submit Premium

Description | Editorial | Solutions | Submissions

Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]
Output: 2

Example 2:



Input: isConnected = [[1,0,0],[0,1,0],[0,0,1]]
Output: 3

Constraints:

- `1 <= n <= 200`
- `n == isConnected.length`
- `n == isConnected[i].length`
- `isConnected[i][j]` is `1` or `0`.
- `isConnected[i][i] == 1`

10.3K 132 121 Online

Code

C++ Auto

```
            dfs(isConnected, visited, j);
        }
    }
}

int findCircleNum(vector<vector<int>>& isConnected) {
    int n = isConnected.size();
    vector<bool> visited(n, false);
    int provinces = 0;

    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            ++provinces;
            dfs(isConnected, visited, i);
        }
    }

    return provinces;
}
};
```

Saved Ln 26, Col 3

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

leetcode.com/problems/binary-tree-maximum-path-sum/

Gmail YouTube Maps Disney+ Hotstar -... Tata Play All Bookmarks

Problem List Run Submit Premium

Description | Editorial | Solutions | Submissions

124. Binary Tree Maximum Path Sum

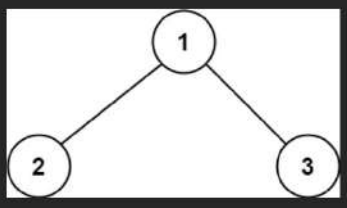
Hard Topics Companies

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return *the maximum **path sum** of any **non-empty** path*.

Example 1:



Input: root = [1,2,3]
Output: 6

17.4K 237 186 Online

Code

C++ Auto

```
8   *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9   *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10  * };
11  */
12  class Solution {
13  public:
14      int maxSum = INT_MIN;
15
16      int dfs(TreeNode* node) {
17          if (!node) return 0;
18
19          int left = max(0, dfs(node->left));
20          int right = max(0, dfs(node->right));
21
22          maxSum = max(maxSum, left + right + node->val);
23
24          return node->val + max(left, right);
25      }
26
27      int maxPathSum(TreeNode* root) {
28          dfs(root);
```

Saved Ln 31, Col 3

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

leetcode.com/problems/surrounded-regions/

Gmail | YouTube | Maps | Disney+ Hotstar -... | Tata Play | All Bookmarks

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

130. Surrounded Regions

Medium | Topics | Companies

You are given an `m x n` matrix `board` containing **letters** `'X'` and `'O'`, **capture regions** that are **surrounded**:

- **Connect**: A cell is connected to adjacent cells horizontally or vertically.
- **Region**: To form a region **connect every** `'O'` cell.
- **Surround**: The region is surrounded with `'X'` cells if you can **connect the region** with `'X'` cells and none of the region cells are on the edge of the `board`.

To capture a **surrounded region**, replace all `'O'`s with `'X'`s **in-place** within the original board. You do not need to return anything.

Example 1:

Input: `board = [["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]`

Output: `[["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]`

Explanation:

9.1K | 216 | 122 Online

Code

C++ | Auto

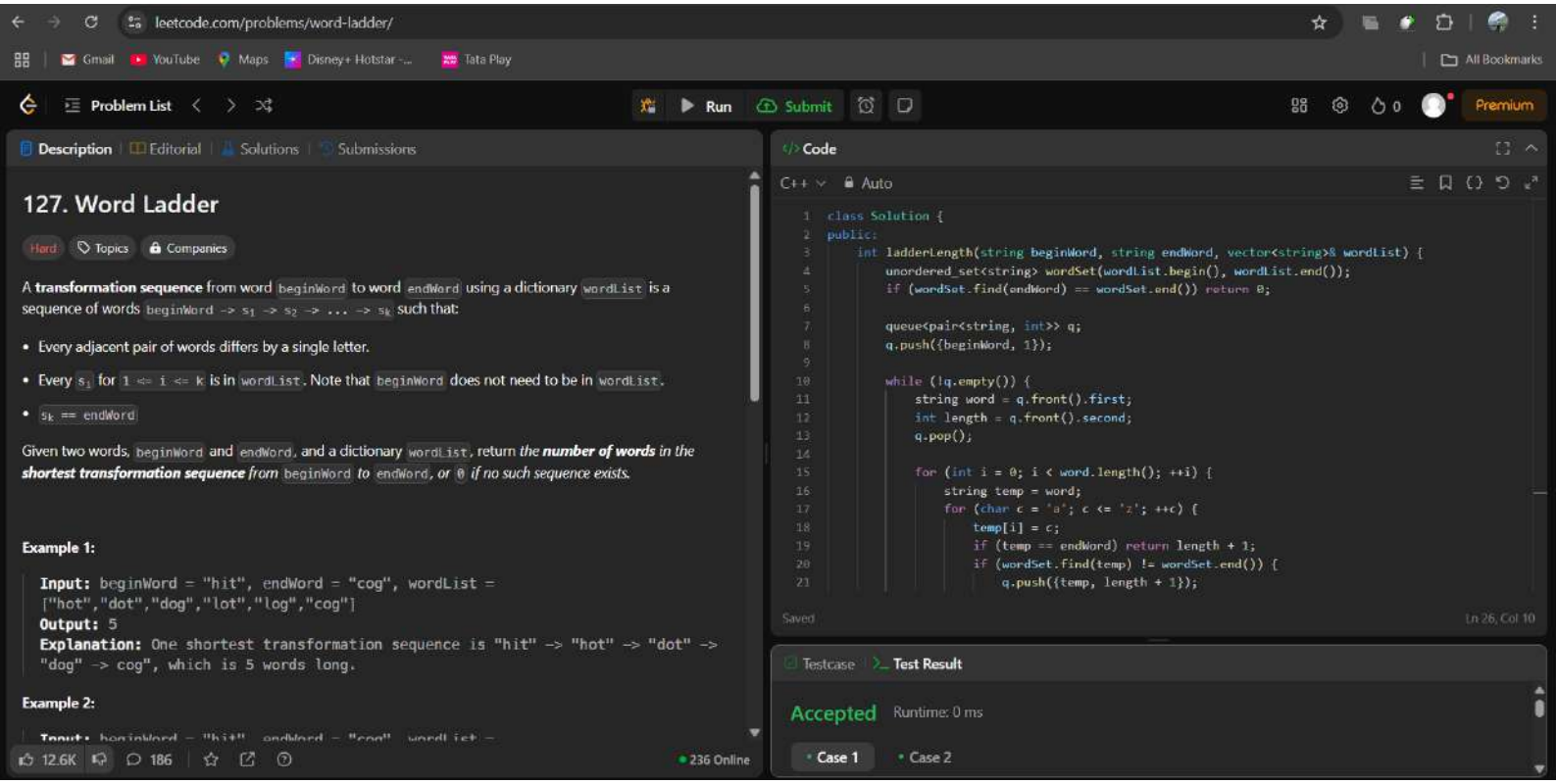
```cpp
class Solution {
public:
    void dfs(vector<vector<char>>& board, int i, int j) {
        int m = board.size(), n = board[0].size();
        if (i < 0 || i >= m || j < 0 || j >= n || board[i][j] != 'O') return;

        board[i][j] = '#';

        dfs(board, i + 1, j);
        dfs(board, i - 1, j);
        dfs(board, i, j + 1);
        dfs(board, i, j - 1);
    }

    void solve(vector<vector<char>>& board) {
        int m = board.size();
        if (m == 0) return;
        int n = board[0].size();

        for (int i = 0; i < m; ++i) {
            if (board[i][0] == 'O') dfs(board, i, 0);
```

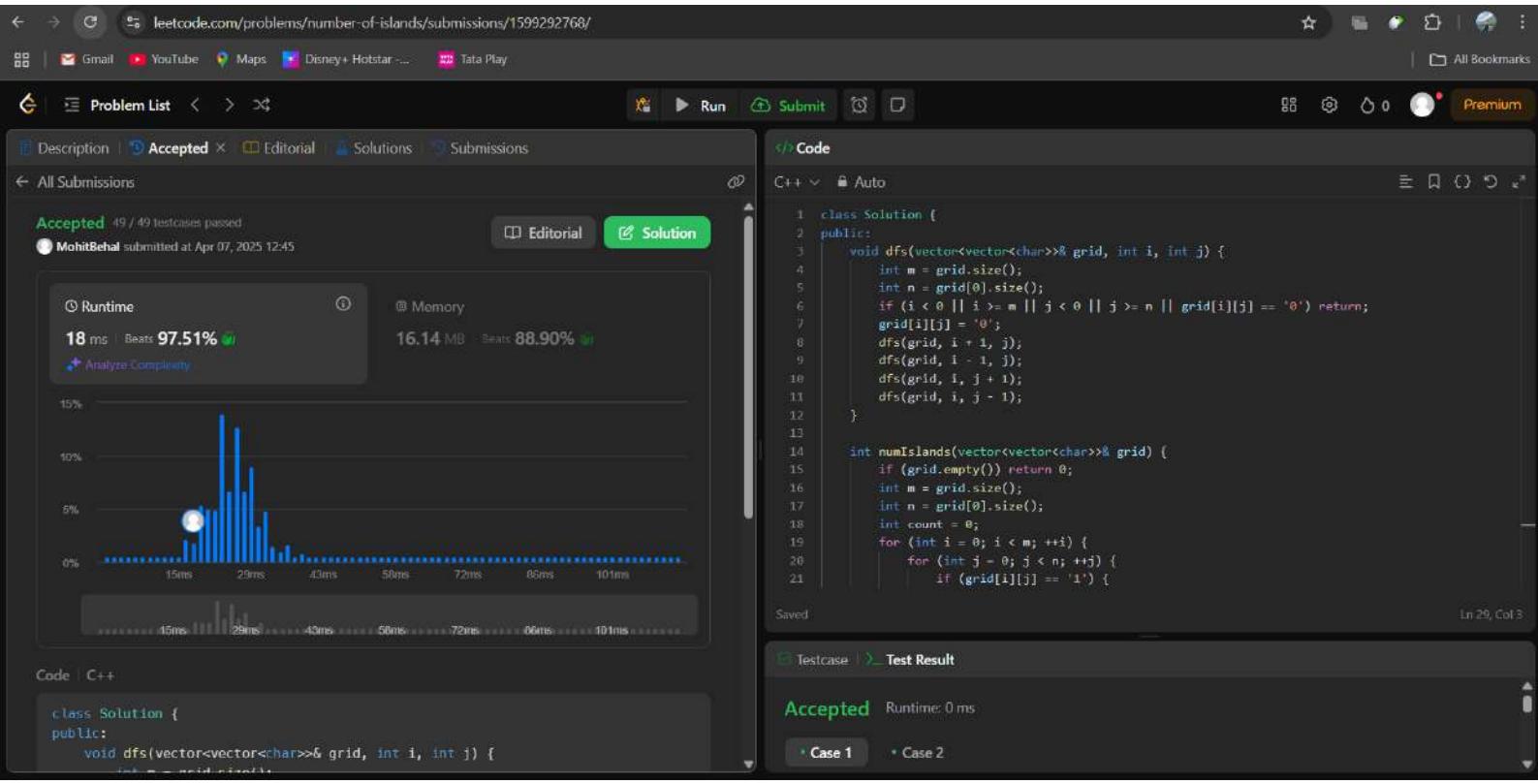
Saved | Ln 13, Col 6

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2


leetcode.com/problems/word-ladder/
Problem List
Run
Submit
Premium
Description | Editorial | Solutions | Submissions
127. Word Ladder
Hard
Topics
Companies
A transformation sequence from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s1 -> s2 -> ... -> sk such that:
Every adjacent pair of words differs by a single letter.
Every si for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
sk == endWord
Given two words, beginWord and endWord, and a dictionary wordList, return the number of words in the shortest transformation sequence from beginWord to endWord, or 0 if no such sequence exists.
Example 1:
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is 5 words long.
Example 2:
12.6K
186
236 Online
Code
C++
Auto
class Solution {
public:
int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
unordered_set<string> wordSet(wordList.begin(), wordList.end());
if (wordSet.find(endWord) == wordSet.end()) return 0;
queue<pair<string, int>> q;
q.push({beginWord, 1});
while (!q.empty()) {
string word = q.front().first;
int length = q.front().second;
q.pop();
for (int i = 0; i < word.length(); ++i) {
string temp = word;
for (char c = 'a'; c <= 'z'; ++c) {
temp[i] = c;
if (temp == endWord) return length + 1;
if (wordSet.find(temp) != wordSet.end()) {
q.push({temp, length + 1});
Saved
Ln 26, Col 10
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2


leetcode.com/problems/number-of-islands/submissions/1599292768/
Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 49 / 49 testcases passed
MohitBehal submitted at Apr 07, 2025 12:45
Editorial
Solution
Runtime
18 ms | Beats 97.51%
Analyze Complexity
Memory
16.14 MB | Beats 88.90%
15%
10%
5%
0%
15ms
29ms
43ms
58ms
72ms
86ms
101ms
Code | C++
class Solution {
public:
void dfs(vector<vector<char>>& grid, int i, int j) {
Code
C++
Auto
class Solution {
public:
void dfs(vector<vector<char>>& grid, int i, int j) {
int m = grid.size();
int n = grid[0].size();
if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] == '0') return;
grid[i][j] = '0';
dfs(grid, i + 1, j);
dfs(grid, i - 1, j);
dfs(grid, i, j + 1);
dfs(grid, i, j - 1);
}
int numIslands(vector<vector<char>>& grid) {
if (grid.empty()) return 0;
int m = grid.size();
int n = grid[0].size();
int count = 0;
for (int i = 0; i < m; ++i) {
for (int j = 0; j < n; ++j) {
if (grid[i][j] == '1') {
Saved
Ln 29, Col 3
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2

leetcode.com/problems/longest-increasing-path-in-a-matrix/

Gmail YouTube Maps Disney+ Hotstar -... Tata Play All Bookmarks

Problem List Run Submit 0 Premium

Description | Editorial | Solutions | Submissions Run Ctrl

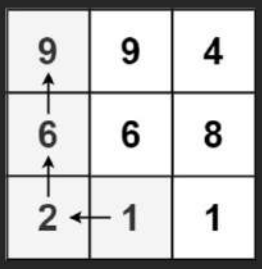
329. Longest Increasing Path in a Matrix

Hard Topics Companies

Given an `m x n` integers `matrix`, return *the length of the longest increasing path in* `matrix`.

From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

Example 1:



Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]

9.2K 68 67 Online

Code

C++ Auto

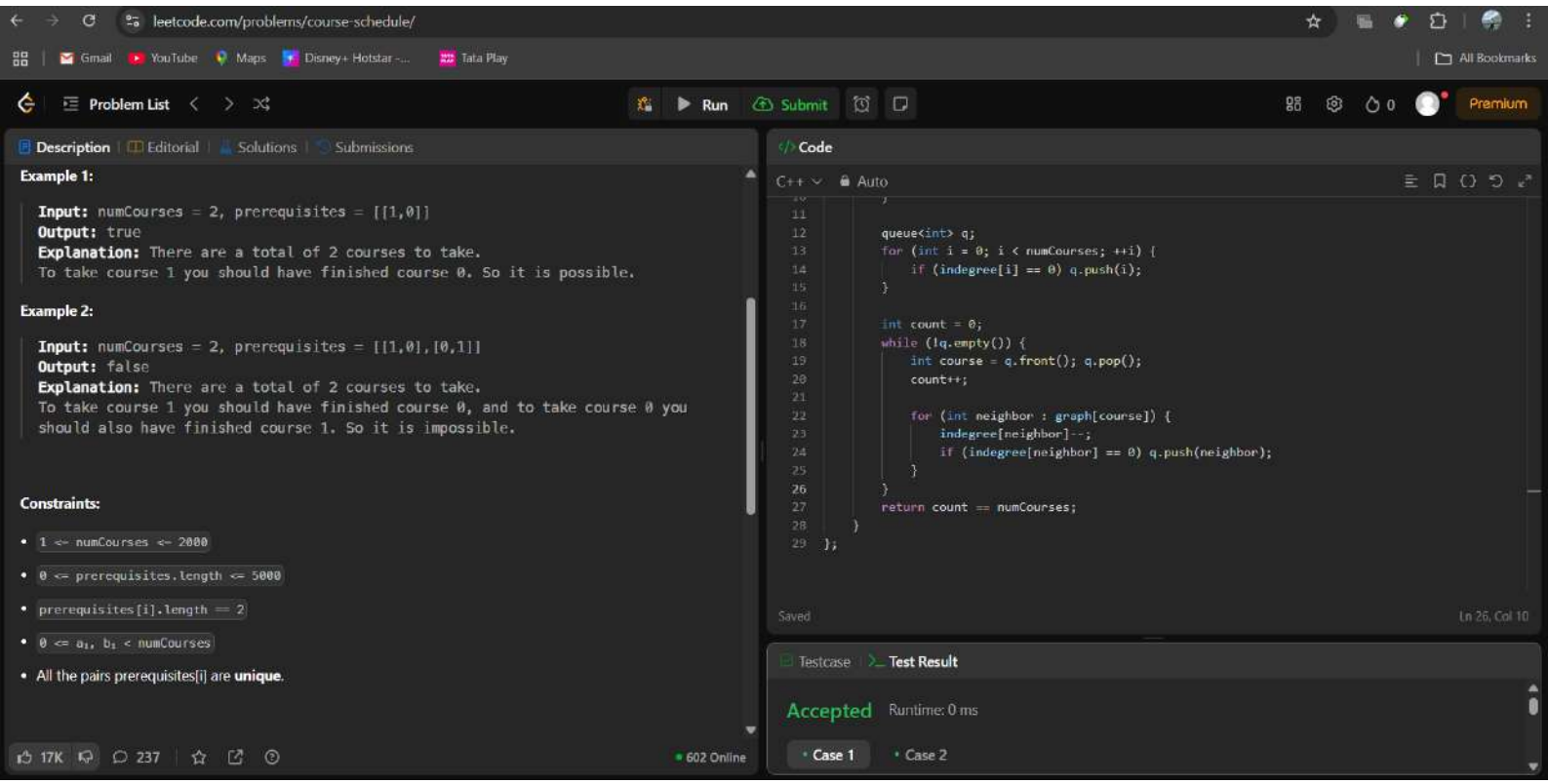
```
        memo[i][j] = maxLen;
        return maxLen;
    }

    int longestIncreasingPath(vector<vector<int>>& matrix) {
        if (matrix.empty() || matrix[0].empty()) return 0;

        m = matrix.size();
        n = matrix[0].size();
        vector<vector<int>> memo(m, vector<int>(n, 0));

        int longest = 0;
        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
                longest = max(longest, dfs(i, j, matrix, memo));

        return longest;
    }
};
```

Saved Ln 35, Col 3

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3


leetcode.com/problems/course-schedule/
Gmail
YouTube
Maps
Disney+ Hotstar -...
Tata Play
All Bookmarks
Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
Example 1:
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
Example 2:
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.
Constraints:
1 <= numCourses <= 2000
0 <= prerequisites.length <= 5000
prerequisites[i].length == 2
0 <= a_i, b_i < numCourses
All the pairs prerequisites[i] are unique.
17K
237
602 Online
Code
C++
Auto
queue<int> q;
for (int i = 0; i < numCourses; ++i) {
if (indegree[i] == 0) q.push(i);
int count = 0;
while (!q.empty()) {
int course = q.front(); q.pop();
count++;
for (int neighbor : graph[course]) {
indegree[neighbor]--;
if (indegree[neighbor] == 0) q.push(neighbor);
return count == numCourses;
};
Saved
Ln 26, Col 10
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2

leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

Gmail YouTube Maps Disney+ Hotstar -... Tata Play All Bookmarks

Problem List Run Submit Premium

Description | Editorial | Solutions | Submissions

236. Lowest Common Ancestor of a Binary Tree

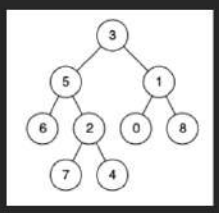
Solved

Medium Topics Companies

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.

17.5K 120 300 Online

Code

C++ Auto

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if (!root || root == p || root == q) return root;

        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        if (left && right) return root;
        return left ? left : right;
    }
};
```

Saved Ln 1, Col 1

Testcase Test Result

Accepted Runtime: 3 ms

Case 1 Case 2 Case 3