



Discover. Learn. Empower.

ENGINEERING

WORKSHEET-9

Student Name: Kumar Arun

UID: 22BCS50204

Branch: CSE

Section/Group: NTPP-603-B

Semester: 6th

Date of Performance: 03/04/25

Subject Name: AP-2

Subject Code: 22CSP-351

Aim(i): *Given an $m \times n$ 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.*

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Source Code:

```
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {
        int islands = 0;
        int rows = grid.size();
        int cols = grid[0].size();
        unordered_set<string> visited;

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

        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                if (grid[r][c] == '1' && visited.find(to_string(r) + "," + to_string(c))
                    == visited.end()) {
                    islands++;
                    bfs(grid, r, c, visited, directions, rows, cols);
                }
            }
        }
    }
};
```

```

    }

    return islands;
}

private:
    void bfs(vector<vector<char>>& grid, int r, int c, unordered_set<string>& visited,
vector<pair<int, int>>& directions, int rows, int cols) {
        queue<pair<int, int>> q;
        visited.insert(to_string(r) + "," + to_string(c));
        q.push({r, c});

        while (!q.empty()) {
            auto [row, col] = q.front();
            q.pop();

            for (auto [dr, dc] : directions) {
                int nr = row + dr;
                int nc = col + dc;
                if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] == '1'
&& visited.find(to_string(nr) + "," + to_string(nc)) == visited.end()) {
                    q.push({nr, nc});
                    visited.insert(to_string(nr) + "," + to_string(nc));
                }
            }
        }
    }
};

```

OUTPUT:

☒ Testcase
 ☒ Test Result
 []

Accepted
 Runtime: 0 ms

• Case 1
• Case 2

Input

```

grid =
[["1","1","1","1","0"],["1","1","0","1","0"],["1","1","0","0","0"],["0","0","0",
"0","0"]]
    
```

Output

```

1
    
```

Expected

```

1
    
```

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

```
grid =  
[["1","1","0","0","0"],["1","1","0","0","0"],["0","0","1","0","0"],["0","0","0",  
"1","1"]]
```

Output

3

Expected

3

Accepted 49 / 49 testcases passed

Meenansh_16380 submitted at Apr 03, 2025 15:50

Editorial Solution

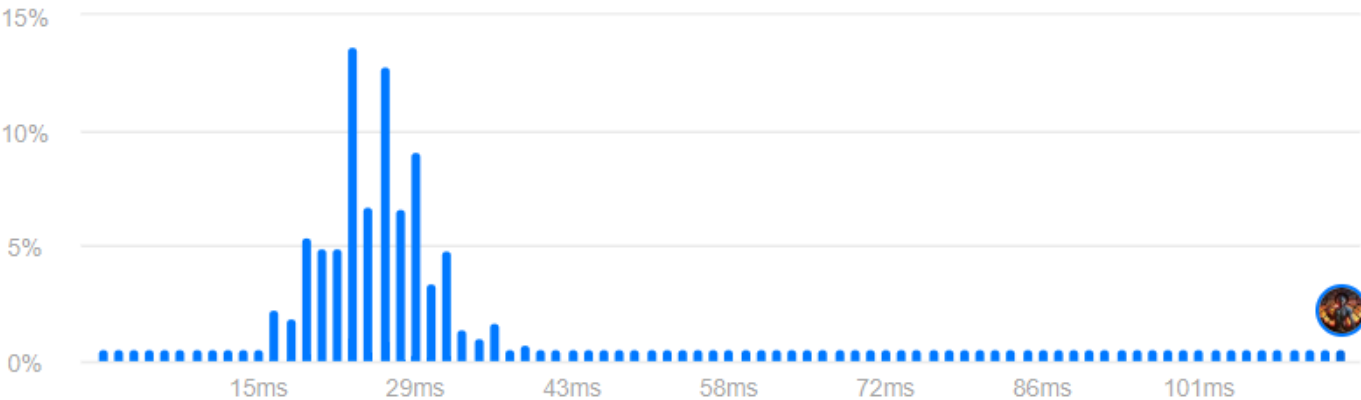
Runtime

119 ms | Beats 5.06%

Analyze Complexity

Memory

34.06 MB | Beats 6.13%



Aim(ii): *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 $1 \leq i \leq k$ 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.

Source Code:

```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
        // make undirected cyclic graph
        map<string, vector<string>> graph;
        int n = wordList.size();
        bool endWord_exists = false;
        for (int i=0; i<n; i++){
            string word = wordList[i];
            for (int j=i+1; j<n; j++){
                string pword = wordList[j];
                if (poss(word, pword)){
                    graph[word].push_back(pword);
                    graph[pword].push_back(word);
                }
            }
            if (word == endWord) endWord_exists = true;
        }
        // word need to exists in the wordList
        if (!endWord_exists) return 0;
        // find the shortest path for each word, starting from endWord
        int time = 1;
        map<string, int> word_path;
        word_path[endWord] = 0;
        queue<string> q;
        q.push(endWord);
        while(q.size()){
            int size = q.size();
            for (int i=0; i<size; i++){
```

```

        string node = q.front(); q.pop();
        for (string neigh : graph[node]){
            if (word_path.find(neigh) == word_path.end()){
                word_path[neigh] = time;
                q.push(neigh);
            }
        }
    }
    time++;
}
// for each word, find the min distance
int distance = 1e9;
for (string& word : wordList){
    if (poss(word, beginWord)){
        // edge case when it doesnt have a path || the matching word is endWord
        // if it has values on it || is endword
        if ((word != endWord && word_path[word] != 0) || word == endWord)
            distance = min(distance, word_path[word] + 2);
    }
}
return (distance == 1e9 ? 0 : distance);
}

private:
    bool poss(string word1, string word2){
        // only one char differs
        int diff = 0;
        for (int i=0; i<word1.size(); i++){
            if (word1[i] != word2[i]) diff++;
        }
        return diff <= 1;
    }
};

```

OUTPUT:

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

```
beginWord =  
"hit"  
  
endWord =  
"cog"  
  
wordList =  
["hot", "dot", "dog", "lot", "log", "cog"]
```

Output

```
5
```

Expected

```
5
```

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

```
beginWord =  
"hit"  
  
endWord =  
"cog"  
  
wordList =  
["hot", "dot", "dog", "lot", "log"]
```

Output

```
0
```

Expected

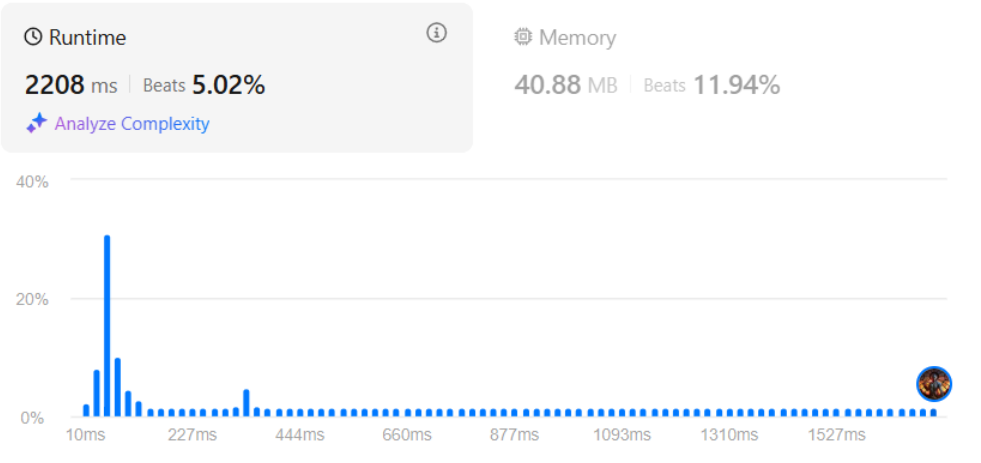
```
0
```

All Submissions

Accepted 51 / 51 testcases passed

Meenansh_16380 submitted at Apr 03, 2025 15:51

Editorial Solution



Aim(iii):

- You are given an $m \times n$ 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.

Source Code:

```
#include <vector>

using namespace std;

class Solution {
public:
    void dfs(int r, int c, vector<vector<char>>& board, vector<vector<bool>>& visited) {
        int m = board.size(), n = board[0].size();
        if (r < 0 || c < 0 || r >= m || c >= n || board[r][c] == 'X' || visited[r][c])
            return;

        visited[r][c] = true;

        dfs(r - 1, c, board, visited);
        dfs(r + 1, c, board, visited);
        dfs(r, c - 1, board, visited);
        dfs(r, c + 1, board, visited);
    }

    void solve(vector<vector<char>>& board) {
        int m = board.size(), n = board[0].size();
        vector<vector<bool>> visited(m, vector<bool>(n, false));
```

```

    for (int i = 0; i < n; i++)
        if (board[0][i] == 'O') dfs(0, i, board, visited);

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

    for (int i = 0; i < n; i++)
        if (board[m - 1][i] == 'O') dfs(m - 1, i, board, visited);

    for (int i = 0; i < m; i++)
        if (board[i][n - 1] == 'O') dfs(i, n - 1, board, visited);

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (board[i][j] == 'O' && !visited[i][j])
                board[i][j] = 'X';
        }
    }
}

};

```


OUTPUT:

Accepted Runtime: 0 ms

- Case 1
- Case 2

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"]]
```

Expected

```
[["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]
```

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

```
board =  
[["X"]]
```

Output

```
[["X"]]
```

Expected

```
[["X"]]
```

Accepted 58 / 58 testcases passed

Meenansh_16380 submitted at Apr 03, 2025 15:57

Editorial

Solution

Runtime

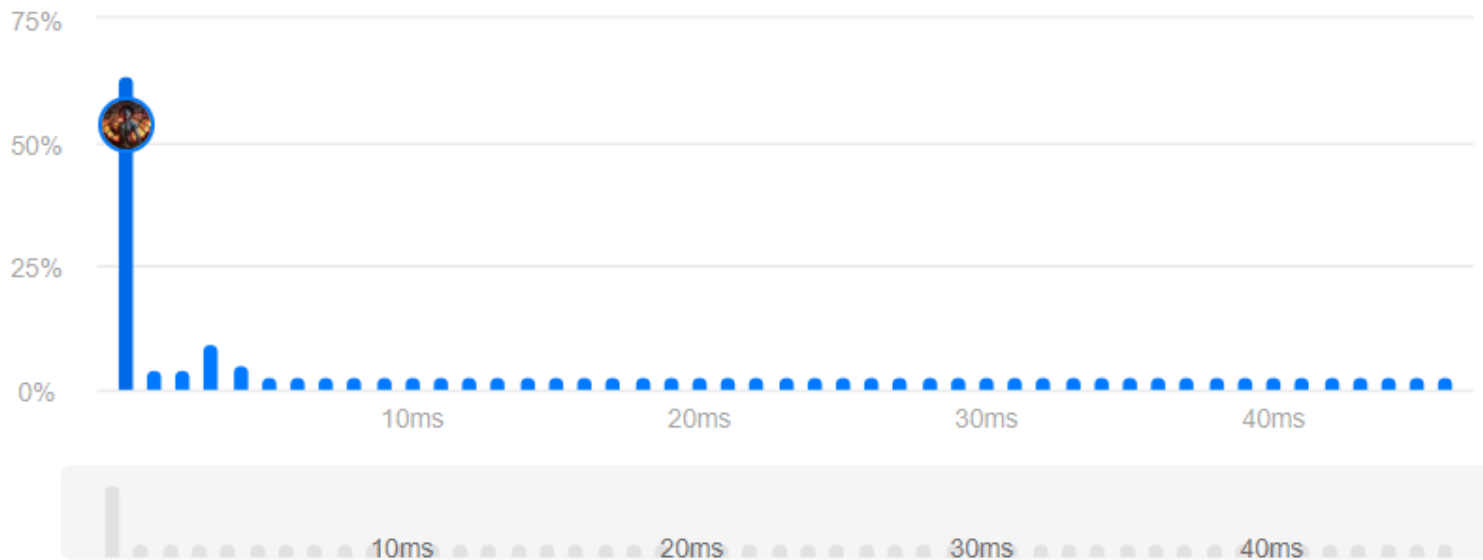


0 ms | Beats 100.00% 🏆

[Analyze Complexity](#)

Memory

14.38 MB | Beats 71.17% 🏆



Learning Outcome

1. We learnt about Graphs.
2. We learnt about Breadth First Search.
3. We learnt about vectors