MICROSOFT PREPARATION

DAY : 09

Word Search :

Problem Link : https://leetcode.com/problems/word-search/

Test Cases Passed : 86 / 86

Time Used : 15.30

Difficulty Level : MEDIUM

Approach Used :

DFS():

- Calculating the dimensions of the grid
- Checking if the current word is the target word :
    - Return true
- Mark the element as visited
- Check for the adjacent elements :
    - Check for the validity of adjacent elements :
        - Check if the adjacent word get us to the desired word ie. adjacent word is target [word.size()] and is unvisited :
            - Add the adjacent element to the word and check if the dfs call returns true :
                - Return true ie. we can complete the word
            - Else
                - Pop the last element of the word
- Un-visit the words occurred on the way as they can be reused
- Return false as this combination didn't reached the complete word

Exists():

- Calculate the dimensions of the grid
- Get the first word of the target word
- Formulate a string that contains only the first word of target
- Iterate for all elements in the matrix :
    - Check if the element is same as the first word :

        - Check if the dfs call for the word as dfs(i,j,visited,grid,word,targetword) returns true :
            - Return true
    - Return false because we cannot get word out of the matrix

Solution :

```
bool dfs(int row, int col, vector<vector<bool>>& visited, vector<vector<char>>&
board, string& word, string& target) {
       int n = board.size();
       int m = board[0].size();

       if (word == target) {
           return true;
       }

       visited[row][col] = true;



       int delRow[] = {-1, 0, 1, 0};
       int delCol[] = {0, 1, 0, -1};

       for (int i = 0; i < 4; i++) {
           int nrow = row + delRow[i];
           int ncol = col + delCol[i];

           if (nrow >= 0 && nrow < n && ncol >= 0 && ncol < m) {
               if (!visited[nrow][ncol] && board[nrow][ncol] ==
target[word.size()])
{
                   word += target[word.size()];
                   if (dfs(nrow, ncol, visited, board, word, target)) {
                       return true;
                   } else {
                       word.pop_back();
                   }
               }
           }
       }

       visited[row][col] = false;
       return false;
   }

   bool exist(vector<vector<char>>& board, string word) {
       if(word.size()==0)
```

```
        {
            return true;
        }
        int n = board.size();
        int m = board[0].size();
        vector<vector<bool>> visited(n, vector<bool>(m, false));

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (board[i][j] == word[0]) {
                    string firstWord = word.substr(0,1);
                    if (dfs(i, j, visited, board, firstWord, word)) {
                        return true;
                    }
                }
            }
        }

        return false;
    }
```