**Find the string in grid :-**

Medium Accuracy: 22.88% Submissions: 39K+ Points: 4

Given a 2D grid of **n**\***m** of characters and a **word**, find all occurrences of given word in grid. A word can be matched in **all 8 directions** at any point. Word is said to be found in a direction if all characters match in this direction (not in zig-zag form). The 8 directions are, **horizontally left**, **horizontally right**, **vertically up**, **vertically down**, and **4 diagonal directions**.  
  
**Note:** The returning list should be **lexicographically smallest**. If the word can be found in multiple directions starting from the same coordinates, the list should contain the coordinates only once.

**Example 1:**

**Input:**grid = {{a,b,c},{d,r,f},{g,h,i}},

word = "abc"

**Output:**{{0,0}}

**Explanation:**From (0,0) we can find "abc" in horizontally right direction.

**Example 2:**

**Input:**grid = {{a,b,a,b},{a,b,e,b},{e,b,e,b}}

word = "abe"

**Output:**{{0,0},{0,2},{1,0}}

**Explanation:**From (0,0) we can find "abe" in right-down diagonal.   
From (0,2) we can find "abe" in left-down diagonal.   
From (1,0) we can find "abe" in horizontally right direction.

**Your Task:**  
You don't need to read or print anything, Your task is to complete the function **searchWord()**which takes grid and word as input parameters and returns a list containing the positions from where the word originates in any direction. If there is no such position then returns an empty list.

**Expected Time Complexity:**O(n\*m\*k) where k is constant  
**Expected Space Complexity:**O(1)

**Constraints:**  
1 <= n <= m <= 50  
1 <= |word| <= 15

**Code :-**

//{ Driver Code Starts

#include<bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// up, down, left, right, up-left, up-right, down-left, down-right

int x[] = {-1, 1, 0, 0, -1, -1, 1, 1};

int y[] = { 0, 0, -1, 1, -1, 1, -1, 1};

bool cmp(vector<int> &v1, vector<int> &v2){

if(v1[0]==v2[0])

return v1[1] < v2[1];

return v1[0] < v2[0];

}

class Solution {

public:

bool valid(vector<vector<char>> &grid, string &word, int i, int j, int dir){

int row=grid.size(), col=grid[0].size(), ind=1;

while(i<row and i>=0 and j<col and j>=0 and ind<word.size()){

if(grid[i][j] != word[ind])

break;

i += x[dir];

j += y[dir];

++ind;

}

if(ind==word.size())

return true;

return false;

}

vector<vector<int>>searchWord(vector<vector<char>>grid, string word){

int row=grid.size(), col=grid[0].size();

vector<vector<int>> ans;

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

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

// 1st letter matches, check for entire

if(grid[i][j] == word[0]){

for(int dir=0; dir<8; ++dir){

if(valid(grid, word, i+x[dir], j+y[dir], dir)){

vector<int> temp = {i,j};

ans.push\_back(temp);

break;

}

}

}

}

}

sort(ans.begin(), ans.end(), cmp);

return ans;

}

};

//{ Driver Code Starts.

int main(){

int tc;

cin >> tc;

while(tc--){

int n, m;

cin >> n >> m;

vector<vector<char>>grid(n, vector<char>(m,'x'));

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

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

cin >> grid[i][j];

}

string word;

cin >> word;

Solution obj;

vector<vector<int>>ans = obj.searchWord(grid, word);

if(ans.size() == 0)

{

cout<<"-1\n";

}

else

{

for(auto i: ans){

for(auto j: i)

cout << j << " ";

cout << "\n";

}

}

}

return 0;

}

// } Driver Code Ends

**T.C :- O(N \* M \* K), K = length of word**

**S.C :- O(1) without returning answer vector**