Student Union Tutorial

Pre-requisites:

This problem will require graph coloring to solve. Make sure your basics of graph theory and graph traversal are clear before attempting this problem. The following links can help you learn the prerequisites:

- http://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/
- http://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/
- https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-graphs-and-their-data-structures-section-1/
- https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-graphs-and-their-data-structures-section-3/
- https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-graphs-and-their-data-structures-section-2/

Problem Description:

This problem requires us to traverse an array from a specified row and column and to count the number of nodes which are similar and adjacent to the one which we started from. If the count exceeds a certain threshold, we are to print “yes” otherwise “no”

Difficulty Level:

Easy

Editorial:

If we view the given input as a 2D implicit graph, we have to count the number of nodes which are adjacent to the one whose row and column number are given and have the same value ('J' or 'C')
If the count exceeds the given threshold, we have to output 'yes' else 'no'

This problem is a good example of a flood fill problem.
'Flood fill is a variant of DFS/BFS and is usually performed on implicit graphs (usually 2d grids)
We will use bfs/dfs to explore and color those neighbors which have the same value ('J' or 'C')

The general algo for coloring such a graph appears like this:

```
int dr[] = {1,1,0,-1,-1,-1, 0, 1}; // trick to explore an implicit 2D grid
```

```
int dc[] = {0,1,1, 1, 0,-1,-1,-1}; // S,SE,E,NE,N,NW,W,SW neighbors
int floodfill(int r, int c, char c1, char c2) { // returns the size of CC
        if (r < 0 || r >= R || c < 0 || c >= C)
                return 0; // outside grid
        if (grid[r][c] != c1)
                return 0; // does not have color c1
        int ans = 1; // adds 1 to ans because vertex (r, c) has c1 as its color
        grid[r][c] = c2; // now recolors vertex (r, c) to c2 to avoid cycling!
        for (int d = 0; d < 8; d++)
                ans += floodfill(r + dr[d], c + dc[d], c1, c2);
        return ans;
}
```

With an understanding of the above algo in mind, one can move to the code.

Complexity of solution:

The complexity of the solution will be the same as that of the complexity of bfs algorithm which is O(V+E) where V is number of vertices in the graph and E is number of edges in the graph.