Image Matching

Images are stored in the form of a grid. Image recognition is possible by comparing grids of two images and checking if they have any *matching regions*.

You are given two grids where each cell of the grids contains either a *0* or a *1*. If two cells share a side then they are adjacent. Cells containing *1* form a *connected region* if we can reach any cell of that region by moving through the adjacent cells that contain *1*. We overlay the first grid onto the second and if a region of the first grid completely matches a region of the second grid, the regions are matched. You have to count total number of such matched regions in the second grid.

For example, given two 3x3 grids *1* and *2*:

111 111 → 111 111
100 100 → 100 100
100 101 → 100 101

We find *2* regions in the second grid: { (0,0),(0,1),(0,2),(1,0),(2,0) } and { (2,2) }.
Regions in grid 1 cover the first region of grid 2, but not the second region. There is *1* matching region.

Making a slight alteration to the above example:

111 111
101 100
100 101

There are no matching regions. From the first graph, the *1* at position *(1,2)* is not matched in the second grid's larger region. The second grid position *(2,2)* is not matched in grid *1*.

Function Description
Complete the function *countMatches* in the editor below. The function must return the number of matching regions.

countMatches has the following parameter(s):
grid1[grid1[0],...grid1[n-1]]: an array of bit strings representing the rows of image 1
grid2[grid2[0],...grid2[n-1]]: an array of bit strings representing the rows of image 2

Constraints
- $1 \le n \le 100$
- $1 \le |grid1[i]|, |grid2[i]| \le 100$
- *grid cells contain only 0 or 1*

▶ **Input Format For Custom Testing**

▼ **Sample Case 0**

Sample Input 0

```
3
001
011
100
3
001
011
101
```

Sample Output 0

```
1
```

Explanation 0
First grid forms 2 regions. They are { (0,2), (1,1), (1,2) } and { (2,0) }
Second grid forms 2 regions. They are { (0,2), (1,1) , (1,2), (2,2)} and { (2,0) }
So, only one region matches.

▶ **Sample Case 1**

▶ **Sample Case 2**

YOUR ANSWER

Draft saved 09:55 pm

Original code | C

```c
#include ↔

char* readline();
char* ltrim(char*);
char* rtrim(char*);


// Complete the countMatches function below.
int countMatches(int grid1_count, char** grid1, int grid2_count, char** grid2) {


}


int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char* grid1_count_endptr;
    char* grid1_count_str = ltrim(rtrim(readline()));
    int grid1_count = strtol(grid1_count_str, &grid1_count_endptr, 10);

    if (grid1_count_endptr == grid1_count_str || *grid1_count_endptr != '\0') { exit(EXIT_FAILURE); }

    char** grid1 = malloc(grid1_count * sizeof(char*));

    for (int i = 0; i < grid1_count; i++) {
        char* grid1_item = readline();

        *(grid1 + i) = grid1_item;
    }

    char* grid2_count_endptr;
    char* grid2_count_str = ltrim(rtrim(readline()));
    int grid2_count = strtol(grid2_count_str, &grid2_count_endptr, 10);

    if (grid2_count_endptr == grid2_count_str || *grid2_count_endptr != '\0') { exit(EXIT_FAILURE); }

    char** grid2 = malloc(grid2_count * sizeof(char*));

    for (int i = 0; i < grid2_count; i++) {
        char* grid2_item = readline();

        *(grid2 + i) = grid2_item;
    }

    int res = countMatches(grid1_count, grid1, grid2_count, grid2);

    fprintf(fptr, "%d\n", res);

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);
```

```
        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

```

Line: 9 Col: 1

Test against custom input

Run Code | Submit code & Continue

(You can submit any number of times)

Download sample test cases *The input/output files have Unix line endings. Do not use Notepad to edit them on windows.*