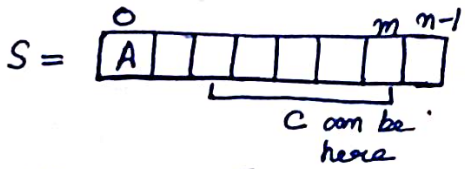
CPP Drill Session

Whenever we are solving problem :-

1. Read the problem.
2. Visualize the solution.
3. Build Strategy / Logic
4. Formulate - How will my code look before even typing code
5. Write code
6. Debug

1. B - Accepted



S[0] → A

[S[2] ..... S[m]] → Range of C

[[0 ..... N-1] of C → small letters] : small letters

Code :-

Conditions : 1) Initial char of S is an uppercase A.
2) One occurrence of C b/w third element from beginning & second last element
3) All letter except A & C are lowercase.

Contest Duration: 2018-08-05(Sun) 17:30 - 2018-08-05(Sun) 19:10 (local time) (100 minutes)

B - AcCepted

Editorial

Time Limit: 2 sec / Memory Limit: 1024 MiB

Score : 200 points

Problem Statement

You are given a string S. Each character of S is uppercase or lowercase English letter. Determine if S satisfies all of the following conditions:

- The initial character of S is an uppercase A.
- There is exactly one occurrence of C between the third character from the beginning and the second to last character (inclusive).
- All letters except the A and C mentioned above are lowercase.

Constraints

- $4 \leq |S| \leq 10$ ($|S|$ is the length of the string S.)
- Each character of S is uppercase or lowercase English letter.

```cpp
void Solve_Karo_Jaldi_Sa_Dusra_Bhi_Karna_Hai() {
    cin >> s;

    if (s.empty() || s[0] != 'A') {
        cout << "WA" nt
        return;
    }

    int cCnt{}, cIdx{-1};

    for(int i = 2; i < sz(s) - 1; i++) {
        if (s[i] == 'C') {
            cCnt++;
            cIdx = i;
        }
    }
```

```
if (cCnt != 1) {
    cout << "WA" nt
    return;
}

for0(i, sz(s)) {
    if (i == 0 or i == cIdx) continue;
    if (!islower(s[i])) {
        cout << "WA" nt
        return;
    }
}
```

```
    for0(i, sz(s)) {
        if (i == 0 or i == cIdx) continue;
        if (!islower(s[i])) {
            cout << "WA" nt
            return;
        }
    }

    cout << "AC" nt
}
```

Time Limit: 2 sec / Memory Limit: 256 MiB

Score : 200 points

Problem Statement

You are given an $H \times W$ grid.

The squares in the grid are described by H strings, $S_1, ..., S_H$.

The j-th character in the string S_i corresponds to the square at the i-th row from the top and j-th column from the left $(1 \leq i \leq H, 1 \leq j \leq W)$.

`.` stands for an empty square, and `#` stands for a square containing a bomb.

Dolphin is interested in how many bomb squares are horizontally, vertically or diagonally adjacent to each empty square.

(Below, we will simply say "adjacent" for this meaning. For each square, there are at most eight adjacent squares.)

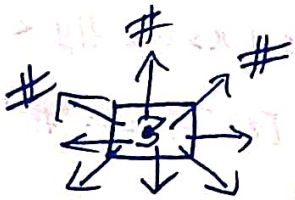
He decides to replace each `.` in our H strings with a digit that represents the number of bomb squares adjacent to the corresponding empty square.

Print the strings after the process.

Constraints

- $1 \leq H, W \leq 50$
- S_i is a string of length W consisting of `#` and `.`.

2. Minesweeper



What we need to do ?

Visualize : For a particular cell which is dot, it is asking no. of bomb cells in 8 directions & replace ~~it with~~ dot with the count.

Strategy :

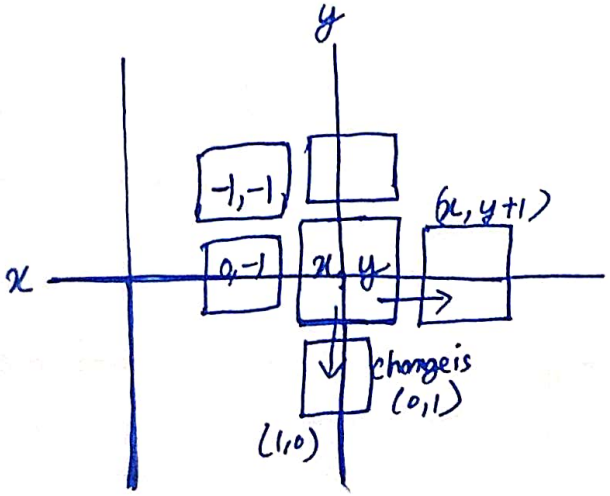
1) ~~For e~~ Traverse each cell, for the cell which is '.' count its neighbour.
2) To count neighbours, go through all 8 directions
3) If '#', then count ++.

* If to go in 8 dir, we write like this

if (x-1, y-1)
⋮
8 if else

→ This would be too long.

So, let's say we are at (x, y)



* The 8 changes we need to do from (x,y) to get its neighbour is same for all 8 cells.

```cpp
#include<bits/stdc++.h>
using namespace std;

int n,m;
string arr[51];

int dx[] = {0,1,1,1,0,-1,-1,-1};
int dy[] = {1,1,0,-1,-1,-1,0,1};

bool bomb(int x,int y){
    if(x<0||x>=n||y<0||y>=m)return 0;
    if(arr[x][y]=='#')return 1;
    else return 0;
}
```

```
int count_nei(int x,int y){
  int cnt = 0;
    for(int dir=0;dir<8;dir++){
        if(bomb(x+dx[dir],y+dy[dir])){
            cnt++;
        }
    }
    return cnt;
}

int main(){
    cin>>n>>m;
    for(int i=0;i<n;i++){
      cin>>arr[i];
    }
    for(int i=0;i<n;i++){
      for(int j=0;j<m;j++){
        if(arr[i][j]=='.'){
          int x = count_nei(i,j);
          arr[i][j] = char(x+'0');
        }
        cout<<arr[i][j];
      }
      cout<<endl;
    }
```

040. Valid Sudoku

time limit per test: 1 second
memory limit per test: 256 megabytes

Sudoku is a popular number puzzle involving placing digits (1-9) in a 9 by 9 grid. A sudoku is considered `valid` if all of the following are true:

-All digits are integers from 1 to 9

-No number occurs in the same row twice

-No number occurs in the same column twice

-No number occurs in the same 3 by 3 box twice.

Given a sudoku board, print whether or not it is valid.

Input

The input consists of nine lines, each containing nine space-separated integers ranging from 1 to 9.

Output

If the sudoku is valid, output VALID, otherwise output INVALID.

Examples

input Copy

```
1 2 3 4 5 6 7 8 9
4 5 6 7 8 9 1 2 3
7 8 9 1 2 3 4 5 6
2 3 1 5 6 4 8 9 7
5 6 4 8 9 7 2 3 1
8 9 7 2 3 1 5 6 4
3 1 2 6 4 5 9 7 8
6 4 5 9 7 8 3 1 2
9 7 8 3 1 2 6 4 5
```

output Copy

```
VALID
```

3. Valid Sudoko

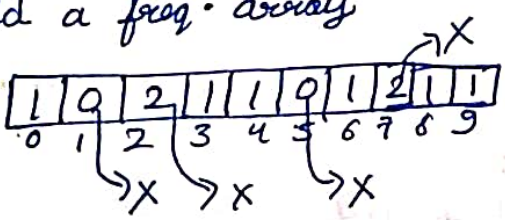
Check (each row / each col / 3×3 box)

↓

[1 .... 9] → all no. appeared exactly once.

Logic:

1. Build a freq. array



* check in the freq. array if one of them being 0 or more than 1. If yes, so output will be invalid.

2. Check each row & column

3. Check 3×3 box

→ Go to the centre of 3×3 grid

→ Check all its 8 neighbours from centre ~~including itself~~ that, all of them appeared once.

→ Include the centre no also as 9th direction.

```cpp
#include<bits/stdc++.h>
using namespace std;


bool valid(int x, int y,vector<vector<int>> &grid){
    vector<int> freq(10);
    for(int i=x; i<x+3; i++){
        for(int j=y; j<y+3; j++){
            freq[grid[i][j]]++;
            if(freq[grid[i][j]] > 1) return false;
        }
    }
    return true;
}


void solve(){
    int grid[9][9];
    for(int i=0; i<9; i++){
        for(int j=0; j<9; j++){
            cin>>grid[i][j];
            if(grid[i][j] < 0 || grid[i][j] > 9){
                cout << "INVALID";
                return;
            }


        }
    }

    for(int i=0; i<9; i++){
        int freqRow[10]={};
        for(int j=0; j<9; j++){
            freqRow[grid[i][j]]++;
            if(freqRow[grid[i][j]] > 1){
                cout << "INVALID";
                return;
            }
        }
    }
```

```
    for(int i=0; i<9; i++){
        int freqCol[10]={};
        for(int j=0; j<9; j++){
            freqCol[grid[j][i]]++;
            if(freqCol[grid[j][i]] > 1){
                cout << "INVALID";
                return;
            }
        }
    }


    for(int i=0; i<9; i+=3){
        for(int j=0; j<9; j+=3){
            if(!valid(i,j,grid)){
                cout <<"INVALID";
                return;
            }
        }
    }

    cout << "VALID";


}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    int t=1;
    //cin >> t;
    for(int i=0; i<t; i++){
        solve();
    }
}
```

Problem List | Submit | Register

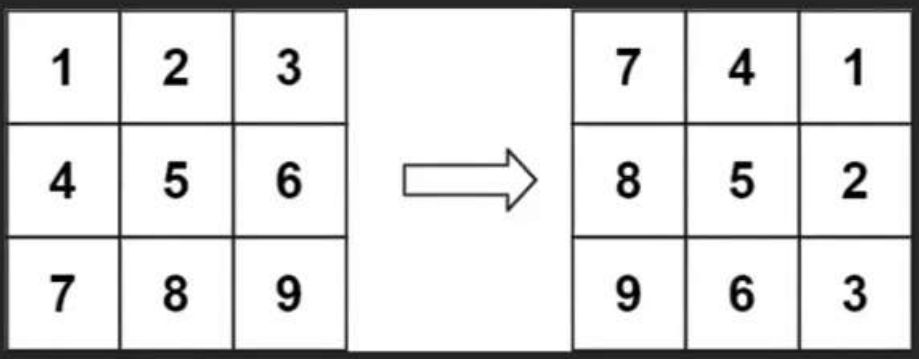
Description | Solutions | Editorial | Submissions

Medium | Topics | Companies

You are given an `n x n` 2D `matrix` representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

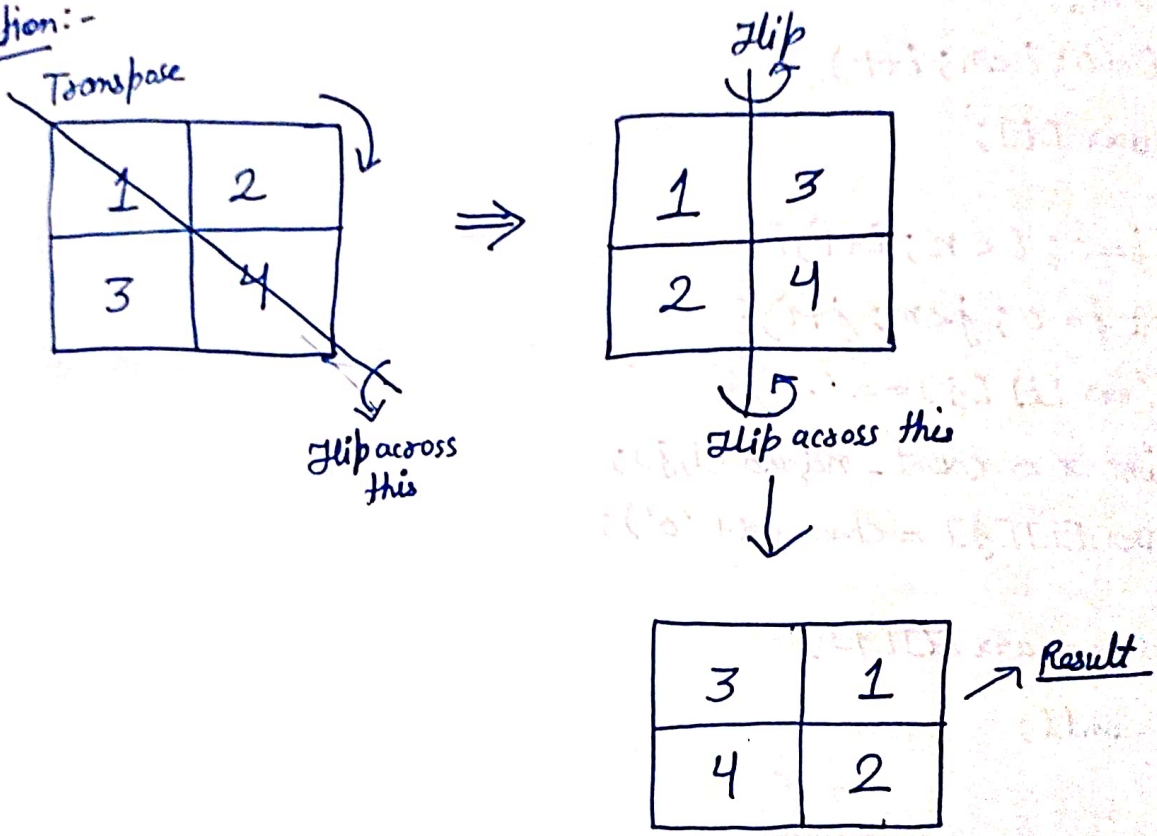
Example 1:




```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

4. Rotate Image by 90°

1st solution:-



2nd solution :-

1) Find corresp. in 4 posⁿ & swap them one by one.
2) Repeat it for remaining cells.

```java
class Solution {
    public void rotate(int[][] matrix) {
        // mat transpose
        for(int i=0; i<matrix.length; i++){
            for(int j=i; j<matrix[i].length; j++){
                int temp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = temp;
            }
        }

        // rev row
        for(int i=0; i<matrix.length; i++){
            for(int j=0; j<matrix.length/2; j++){
                int temp = 0;
                temp = matrix[i][j];
                matrix[i][j] = matrix[i][matrix.length-1-j];
                matrix[i][matrix.length-1-j] = temp;
            }
        }
```

Contest Duration: 2017-02-11(Sat) 17:30 - 2017-02-11(Sat) 19:10 (local time) (100 minutes)

Time Limit: 2 sec / Memory Limit: 256 MiB

Score : 200 points

Problem Statement

You are given an image A composed of N rows and N columns of pixels, and a template image B composed of M rows and M columns of pixels.
A pixel is the smallest element of an image, and in this problem it is a square of size 1×1.
Also, the given images are binary images, and the color of each pixel is either white or black.

In the input, every pixel is represented by a character: . corresponds to a white pixel, and # corresponds to a black pixel.
The image A is given as N strings $A_1, ..., A_N$.
The j-th character in the string A_i corresponds to the pixel at the i-th row and j-th column of the image A $(1 \leqq i, j \leqq N)$.
Similarly, the template image B is given as M strings $B_1, ..., B_M$.
The j-th character in the string B_i corresponds to the pixel at the i-th row and j-th column of the template image B $(1 \leqq i, j \leqq M)$.

Determine whether the template image B is contained in the image A when only parallel shifts can be applied to the images.

Constraints

- $1 \leqq M \leqq N \leqq 50$
- A_i is a string of length N consisting of # and ..
- B_i is a string of length M consisting of # and ..

5. Approach :-

1. Loop over every psbl MXM subgrid of A where i & j go from 0 to N-M
2. For each subgrid of A, compare it with B:
   → If all characters matches with B, we found template
3. If found : print "Yes" otherwise "No"

```cpp
#include <iostream>
#include <vector>
using namespace std;

bool matchesTemplate(const vector<string> &A, const vector<string> &B, int row, int col, int M) {
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < M; j++) {
            if (A[row + i][col + j] != B[i][j])
                return false;
        }
    }
    return true;
}

int main() {
    int N, M;
    cin >> N >> M;

    vector<string> A(N), B(M);
```

```cpp
    // Input grid A
    for (int i = 0; i < N; ++i) {
        cin >> A[i];
    }

    // Input template B
    for (int i = 0; i < M; ++i) {
        cin >> B[i];
    }

    // Try all possible top-left corners for MxM subgrid in A
    for (int i = 0; i <= N - M; ++i) {
        for (int j = 0; j <= N - M; ++j) {
            if (matchesTemplate(A, B, i, j, M)) {
                cout << "Yes" << endl;
                return 0;
            }
        }
    }
```

```
    // Try all possible top-left corners for MxM subgrid in A
    for (int i = 0; i <= N - M; ++i) {
        for (int j = 0; j <= N - M; ++j) {
            if (matchesTemplate(A, B, i, j, M)) {
                cout << "Yes" << endl;
                return 0;
            }
        }
    }

    cout << "No" << endl;
    return 0;
}
```