September 20, 2025

GRAPH DRILL SESSION

Ques 1. Satisfiability of Equality Equations

https://leetcode.com/problems/satisfiability-of-equality-equations/description/

> You are given an array of strings `equations` that represent relationships between variables where each string `equations[i]` is of length `4` and takes one of two different forms: `"x_i==y_i"` or `"x_i!=y_i"`.Here, `x_i` and `y_i` are lowercase letters (not necessarily different) that represent one-letter variable names.
>
> Return `true` *if it is possible to assign integers to variable names so as to satisfy all the given equations, or* `false` *otherwise.*

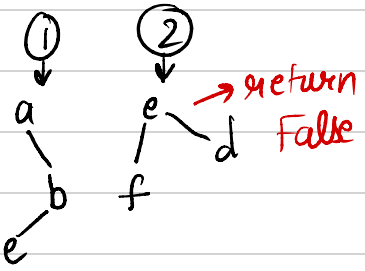
→ a==b, if a equals to b then that means that b is also equals a. So, we can say that a is connected to b with an undirected edge.

→ b==c

implies a==c, TRUE

If we get a statement like a≠c, then it is contradictory and hence here we will ans = FALSE.

≈ a==b
b==e
e==d
e==f
d!=f



Deriving from ②, we can see that d==f but it is given that d!=f, Hence ans = FALSE

Relate this problem to connected components.

Approach : //O(V+E)

1. Make a graph with the help of adjacency list.
2. In the adjacency list, if a==b then a will have b and vice versa.

Using prev example, the adjacency list of that relation will look like:

a : b
b : a c
c : b
d : e
e : df
f : e

Made adjacency list Based on equality relation ONLY (not using inequalities)

3. Perform DFS traversal to find connected components

For prev example:

Nodes :	a	b	c	d	e	f
Component :	1	1	1	2	2	2

No

4. Now use inequality equation to verify the inadequacy. We encounter d!=f, then that means they should not share same component number but our adjacency list assigns them both 2. Hence ANS = FALSE

Return false in such a case otherwise return True.

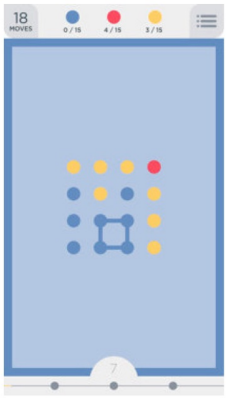
```cpp
bool equationsPossible(vector<string>& equations) {

    map<char, int> component;
    map<char, vector<char>> edges;
    set<char> st;
    map<char, bool> visited;

    for(int i=0; i<equations.size();i++){
        char x, y;

        x = equations[i][0];
        y = equations[i][3];

        st.insert(x);
        st.insert(y);

        if(equations[i][1] == '='){
            edges[x].push_back(y);
            edges[y].push_back(x);
        }

    }
int comps = 0;

for(auto it=st.begin(); it!=st.end(); it++){

    char ch = *it;

    if(visited.count(ch) == 0){

        comps++;

        queue<char> q;

        q.push(ch);

        while(!q.empty()){

            char f = q.front();

            q.pop();

     while(!q.empty()){

         char f = q.front();

         q.pop();

         visited[f] = 1;

         component[f] = comps;

         for(int i=0; i<edges[f].size(); i++){

             char v = edges[f][i];

             if(visited.count(v) == 0){
                 q.push(v);
             }
         }

    }


}
```

```cpp
    for(int i=0; i<equations.size(); i++){

        char x = equations[i][0];
        char y = equations[i][3];

        if(equations[i][1] == '!'){
            if(component[x] == component[y])
                return false;
        }

        if(equations[i][1] == '='){
            if(component[x] != component[y])
                return false;
        }
    }

    return true;

}
```

Ques 2. Fox and Two Dots

https://codeforces.com/problemset/problem/510/B

Fox Ciel is playing a mobile puzzle game called "Two Dots". The basic levels are played on a board of size $n \times m$ cells, like this:



Graph is in the form of a grid.

Each cell contains a dot that has some color. We will use different uppercase Latin characters to express different colors.

The key of this game is to find a cycle that contain dots of same color. Consider 4 blue dots on the picture forming a circle as an example. Formally, we call a sequence of dots $d_1, d_2, ..., d_k$ a cycle if and only if it meets the following condition:

1. These k dots are different: if $i \neq j$ then d_i is different from d_j.
2. k is at least 4.
3. All dots belong to the same color.
4. For all $1 \leq i \leq k - 1$: d_i and d_{i+1} are adjacent. Also, d_k and d_1 should also be adjacent. Cells x and y are called adjacent if they share an edge.

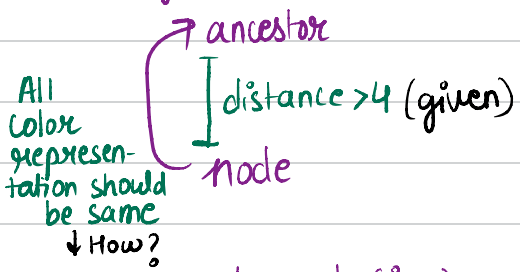
Determine if there exists a cycle on the field.

Cycle detection in an undirected graph

Single Cycle detected → stop & Return False else True.

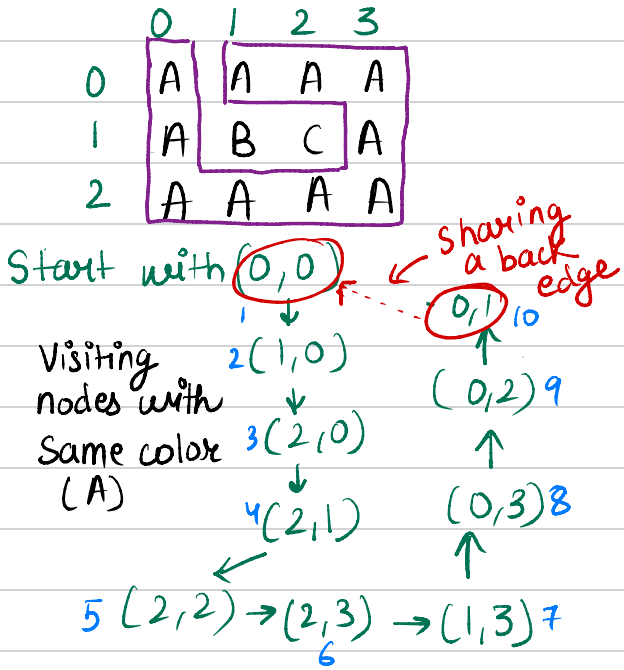
AlgoZenith

If you are using DFS, then how is a cycle detected? Current node, part of the path or the last visited node of the path, has a backedge with an ancestoring node



If you are at a node (i, j) then you will only explore the neighbouring nodes with the same color (meaning same component number)

Example : 1.



Cycle comprises of 10 nodes and after 10, we reach the ancestor node

→ distance > 4 (distance = 10)

RETURN TRUE

2.

	0	1	2	3
0	A	A	A	A
1	A	B	C	A
2	A	A	D	A

1 (0,0) → 2 (1,0) → 3 (2,0) → 4 (2,1)

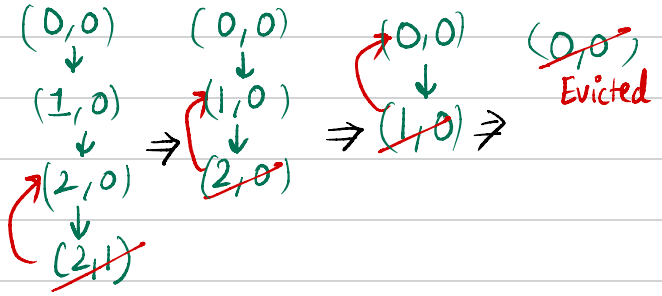
No backage

A	A	A	A
A	B	C	A
A	A	D	A

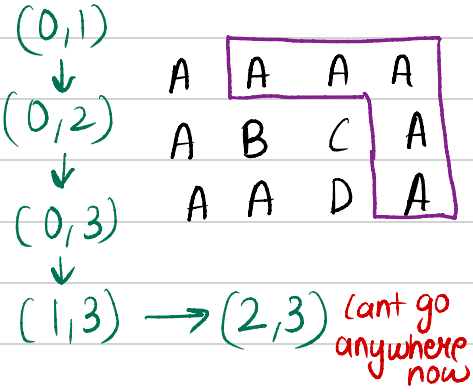
cant go anywhere because the right cell is diff color

Path 1 has no cycle.

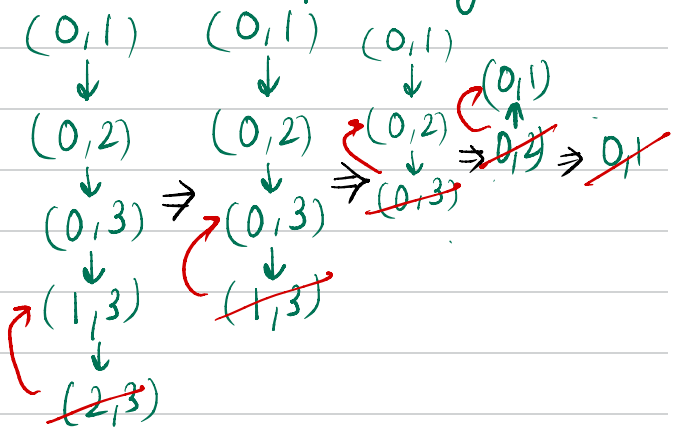
Since from (2,1), we can't go anywhere then we will backtrack to (2,1) and look for possible path but (2,0) will also be evicted because there is no neighbouring cell with same color.



Explore more paths. Start from (0,1)



In path 2, we will again back track and keep evicting.



Similarly, we will start exploring other paths but we will figure out in this example there is no similar color nodes that can lead us to a cycle.
Hence, RETURN FALSE
Ans.

Approach:

1. Use DFS traversal as it is 4-directional
2. Track ancestors using map
3. Maintain ordering of nodes using vector.
4. If the ancestor of the current node is in the map, that means we have backage (given it shouldn't be immediate parent)
5. Now, if the distance b/w the found out ancestor node & current node is greater than 4. Then Return True else False.

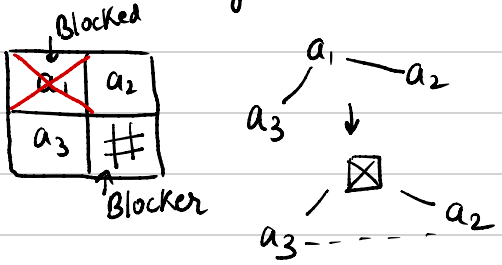
Ques 2. Maze

https://codeforces.com/problemset/problem/377/A

Pavel loves grid mazes. A grid maze is an $n \times m$ rectangle maze where each cell is either empty, or is a wall. You can go from one cell to another only if both cells are empty and have a common side.

Pavel drew a grid maze with all empty cells forming a connected area. That is, you can go from any empty cell to any other one. Pavel doesn't like it when his maze has too little walls. He wants to turn exactly k empty cells into walls so that all the remaining cells still formed a connected area. Help him.

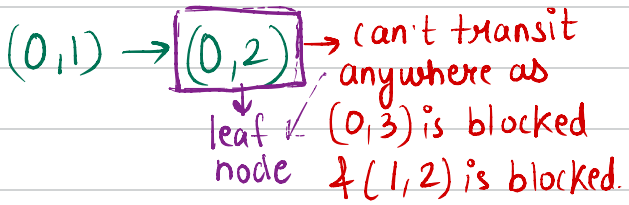
- Let say if a cell is blocked (i.e, its edges are removed) then it will lose its neighbours, but we have to create a new way to connect the 2 neighbours.



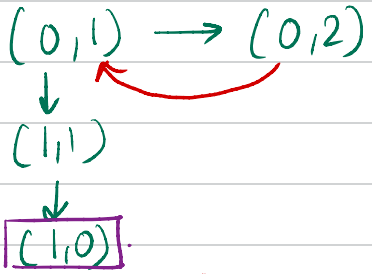
Example 1: 3 4 2 ← Mark 2 nodes

	0	1	2	3
0	#	.	.	#
1	.	.	#	.
2	#	.	.	.

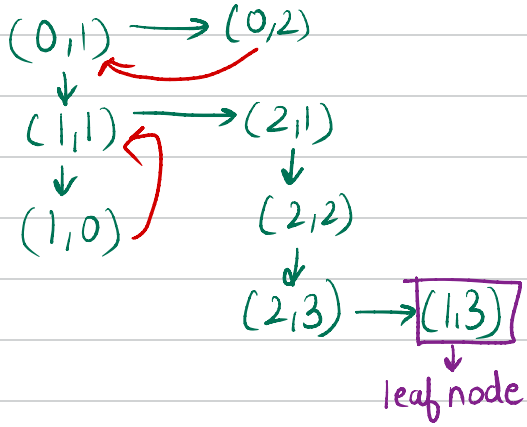
Let's start with (0,1) as the source.



Since (0,2) is leaf, then we will backtrack to its parent and explore



leaf node since can't go anywhere from here

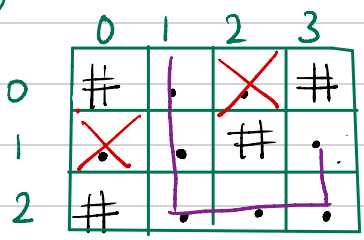


leaf node

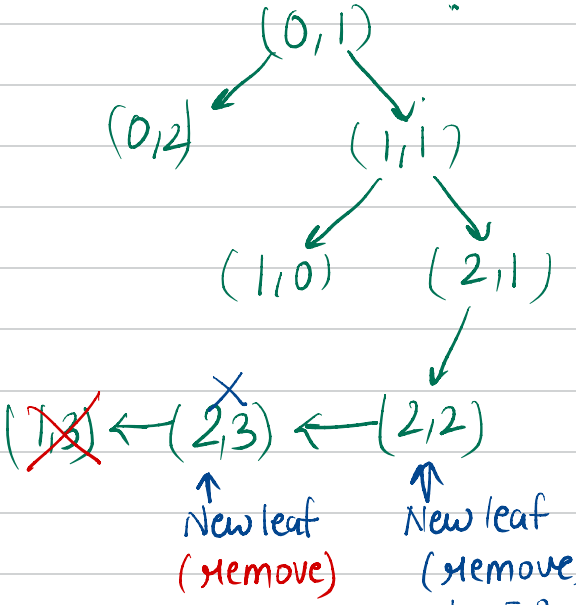
List of leaf nodes:

(0,2)
(1,0)
(1,3)

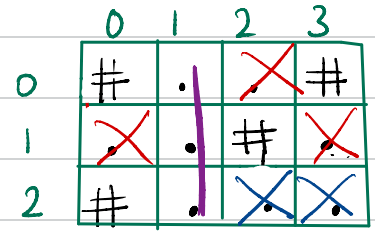
We can mark any of these 2 nodes



Here K=2 (less than leaf nodes) but if K=5 (greater than count of leaf nodes) then in this example, we will remove the 3 leaf nodes and then will remove 2 from the new leaf nodes (made after removing 3 nodes). They were the parent of leaf nodes.



K=5 is done.



Have to track the degree associated with node. lowest ones will get priority followed by the second lowest one and so on

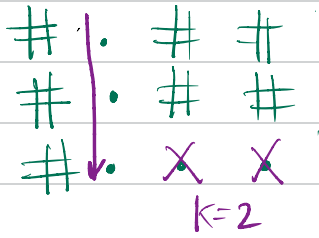
Approach:

Maintain open cells → Blocked

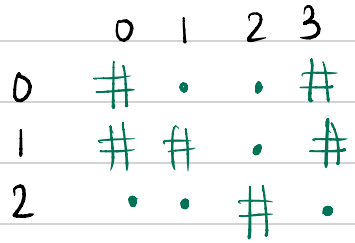
OC ⇒ S - K (Max amount of Connected cells).

no. of open cells

Now, we will start our traversal with the first open cell and continue unit we reach OC cells.



→ Multiple Connected Components



6 open cells (S=6)

K = 3

When input coming as '.' then simply S++.