Content.

Graph

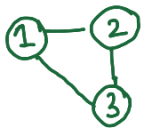
Graph represent → Adj Matrix and Adj list.

```java
public class AdjList {
    public static void main(String[] args) {
        ArrayList<ArrayList> adj = new ArrayList<>();
        int v = 3;
        // Initializing with empty value. No initialization then adj.get(i) will give
IndexOutOfBound Exception.
        for (int i = 0; i <= v; i++) {
            adj.add(new ArrayList<>());
        }
        // Undirected graph.
        // 1--2
        adj.get(1).add(2);
        adj.get(2).add(1);
        // 2---3

        adj.get(2).add(3);
        adj.get(3).add(2);
        // 1---3
        adj.get(1).add(3);
        adj.get(3).add(1);

        for (int i = 1; i <= v; i++) {
            System.out.print(i + " -> ");
            for (int j = 0; j < adj.get(i).size(); j++) {
                System.out.print(adj.get(i).get(j) + " ");
            }
            System.out.println();
        }

    }
}
```

1 → {2,3}

2 → {1, 3}

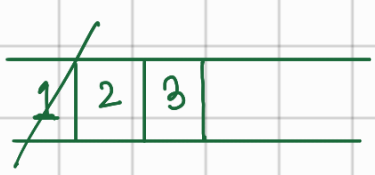
3 → {1,2}.



The adjacency list initialised at 0 to N and the graph value 1 to N.
(The given value will be 1).
[[ ], [2,3], [1,3], [2,1]].

BFS → [[], [2, 3], [1, 3], [2, 1]]

```java
    // [[], [2, 3], [1, 3], [2, 1]]
    // BFS.
    Queue<Integer> q = new LinkedList<>();
    ArrayList<Integer> adj = new ArrayList<>();
    boolean vis[] = new boolean[3+1];
    vis[1] = true;
    q.add(1);
    while (!q.isEmpty()){
        int node = q.poll();
        adj.add(node);
        for(int n:adjList.get(node)){
            if(vis[n]==false){
                vis[n] = true;
                q.add(n);
            }
        }
    }
```



queue added 1.
get all adj node of 1 and add then take top element and continue (q.size()>0).

vis array (N+1) number can start 0 or 1.
when vis marking as 1 as undirected so when we get 3 again there will be 1 & 2.
and they are already visited.

DFS.

```java
public static void dfs(int i, ArrayList<ArrayList<Integer>> adjList,
ArrayList<Integer> adj, boolean vis[]){
        adj.add(i);
        vis[i] = true;
        for(int n:adjList.get(i)){
            if(vis[n]==false){
                dfs(n,adjList,adj,vis);
            }
        }
    }
```

3
2
1

adj.add(1,2,3).

Flood Fill Algorithm.

Given matrix and (i,j) change the value to color and adj block to color.

	0	1	2
0	1	1	1
1	1	(1)	0
2	1	0	1

srcR = 1, srcCol = 1. making mat[sR][sC] = color. and dfs on the i and j.

```
2 2 2
2 2 0
2 0 1
```

	0	1	2
0	1	1	1
1	2	2	0
2	(2)	2	2

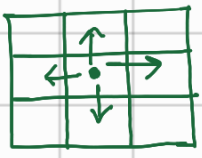
newColor = 3.

srcRow = 2 srcCol = 0. Initial color 2 → color connected to 2 are changed to 3.

dfs on the node and all color connected to it and currColor = sameColor then mark the newcolor and call dfs.

```java
        public int[][] floodFill(int[][] image, int sr, int sc, int color) {
            int vis[][] = new int[image.length][image[0].length]; // mXn matrix
taking care of the m and the n size.
            for (int row[] : vis)
                Arrays.fill(row, -1);
            int c = image[sr][sc];
            dfs(image, vis, sr, sc, c, color);
            return image;
        }

        void dfs(int[][] image, int vis[][], int i, int j, int c, int color) {
            if (i < 0 || j < 0 || i >= image.length || j >= image[0].length ||
image[i][j] != c)
                return;
            if (vis[i][j] == 1) {
                return;
            }
            if (image[i][j] == c) {
                image[i][j] = color;
            }
            vis[i][j] = 1;
            dfs(image, vis, i + 1, j, c, color);
            dfs(image, vis, i, j + 1, c, color);
            dfs(image, vis, i, j - 1, c, color);
            dfs(image, vis, i - 1, j, c, color);
        }
    }
```

Rotten Oranges.



	0	1	2
0	2	1	1
1	1	1	0
2	0	1	1

2 → rotten oranges.
1 → fresh oranges.
0 → empty.

↓ After 1 unit of time the adj node will get rotten.

2	2	1
2	1	0
0	1	1

(See only the adj node are changed & the adj node of the changed node is not changed so one layer → BFS).

↓ After 1 unit of time.

2	2	2
2	2	0
0	1	1

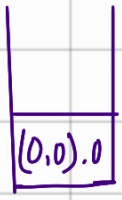
Ans → 4 unit.

↓ 1 unit.

2 2 2
2 2 0
0 2 1 → (1 unit.) → 2 2 2
2 2 0
0 2 2

Brainstorm → Travel one time and get the 2 (position). BSF store ((i,j), time)

2 position (0,0).

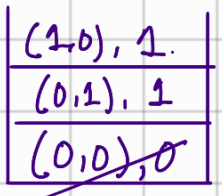


Rotten oranges will impact the nb

level order BFS.

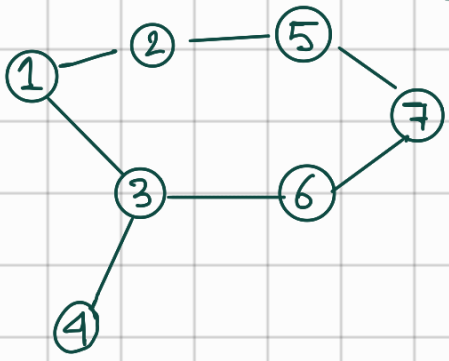
↓ Take the top and add nb.

Get the maps count in the queue



(Time add 1 and add in queue).

Detect cycle in an undirected graph.



BFS.

1	2,3	5,6,4	7

DFS.

When adj of 5 are vis and added in queue then 7 was visited and again as adj 6, 7 will come.

Mark visited then add queue and already vis then cycle.

```
vis[] = new int[n+1];
while(!q.isEmpty()).
{   int top = q.top();
    if(vis[top]!=-1) return true;
    vis[top]=1;
    for (int nb: adj.get(top))
       { if (vis[nb]!=-1)
            return true;
         q.add(nb);
       }
}
return false;
```

→ Here only the node visited then it is showing as cycle but it is not true. It is undirected and will always be visited by 2 nodes.

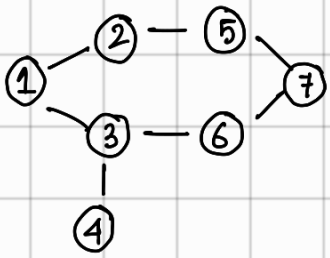
(1)—(2)—(3)

Here 1 will mark 2 and when 2 will be taken then it will see 1 and say there is a cycle. (Wrong).

Solution → To store the parent of the node. In the example 7 parent is 5 and next time 7 parent is 6.

queue → (1,-1).

Detect Cycle in an Undirected Graph DFS.



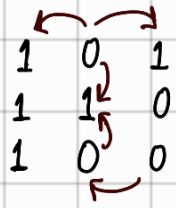
1 → {2, 3}
2 → {1, 5}
3 → {1, 4, 6}
4 → {3}
5 → {2, 7}
6 → {3, 7}
7 → {5, 6}

4	(no nb) return to 3.
~~3~~	(put the nb of 3 → 1,4,6)
~~6~~	6 parent
~~7~~	4 put return to 3.
~~5~~	then put 1 (vis[1]=1).
~~2~~	(DFS going with the value).
~~1~~	(Get the nb of 1 → 2,3)

```
public boolean isCycle (A<A<I>> adj)
{ int node = adj.size();
  int vis[] = new int[node];
  for (int i=0; i<node; i++)
   { if (vis[i]==0)
      if (dfs(i,-1, vis, adj))
          return true;
   }
  return false;
}
   public boolean dfs ( int node, int parent, int vis[], A<A<I>> adj)
    { # if (node!=parent && vis[node]!=0)
              return true;
        vis[node] = 1;
        for (int nb : adj.get(node))
        { if (vis[nb] == 0)
            { if (dfs(nb, node, vis, adj))
                 return true; }
          else
            if (node!=parent && vis[node]!=0)
                 return true;
        }
       return false;
   }
```

// The problem the current node should not be seen as parent node. In the nb node we need to put.

Distance of nearest cell having 1.



1	0	1
1	1	0
1	0	0

O/P →

0	1	0
0	0	1
0	1	2

initially dist [] = MAX.
when 1 put 0.
when 0 see min & put (min (prev, curr)).

The point is with this approach calling BFS on every node.
(Multi Source BFS is the solution).

Getting the smallest distance.
BFS call with a counter.

~~(0,0,0)~~ | (0,1,1) | (1,0,0) → value 1 so dist = 0.

1 0
(already 1) (called value + 1) (x, y, distance).

~~(0,1,1)~~ | (1,0,0) | ~~(0,0,2)~~ | (0,2,2) | ~~(1,1,2)~~

The distance value = 1.
(value filled and min taken)
(put the min value when 1 then 0.)

When calling nb the distance value + 1.

~~(1,0,0)~~ | (0,2,0) | (1,1,0).

~~(1,0,0)~~ | ~~(0,2,0)~~ | ~~(1,1,0)~~ | (2,0,0)

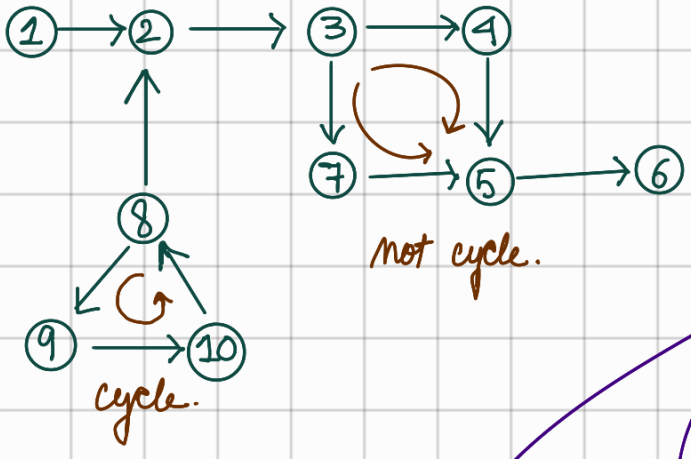
The nb (1,0) are all 1 so dist = 0.

~~(0,2,0)~~ | ~~(1,1,0)~~ | (2,0,1) | (0,1,1)

~~(1,1,0)~~ | (2,0,0) | (0,1,1) |

(1,2,1) | (2,1,1)

Detect cycle in a directed graph.



Brainstorming.

Detect with the vis and parent will not work. Mark ③ as visited and take ④ then ⑤. There can be store the parent and again 3 to 7 to 5. Now 5 is visited with parent ④ & ⑦. It will show as cycle but it is not.

We need to store the vis node in same path.

In ③ to ④ it went and again ③ to ⑦ different path.

We need to maintain vis and pathVis array of nodes.

When going from 1 to 3 to 4 to 6. then mark 1 in vis and pathVis. When coming back from 6 to 4 to 3 to go to 7 then mark 6, 5 and 4 in pathVis as 0.

Then 7 and 5 are marked in pathVis as 1. They will come back to 1 and all pathVis as 0.

There are two common algorithms:

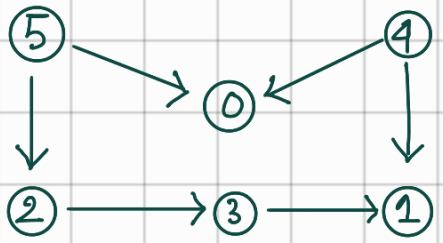
Using **DFS** (Depth-First Search) – Most common.
Keep track of recursion stack (nodes currently being visited)

Using **BFS** (Kahn's Algorithm / Topological Sort)
If topological sorting is not possible (because some nodes remain unprocessed), then a cycle exists.

There are 2 component and 1 to 7 all are marked in vis array.
Next ⑧ then ⑨ and ⑩ and the nb are visited in vis and pathVis. (Cycle).

Topological Sorting BFS.

DAG ⟶ Linear Ordering of vertices such that if there is an edge between u and v then u appears before v.



5 4 2 3 1 0

4 5 2 3 1 0

Get the indegree of all vertex.

0 → 2
1 → 2
2 → 1
3 → 1
4 → 0.
5 → 0.

queue added → 4 & 5.

5	4	

Topological Sort with BFS

↓

Kahn's Algorithm.

adj node.

0 → { }
1 → { }
2 → {3}
3 → {1}.
4 → {0,1}
5 → {0,2}.

Get top of queue (5) and nb ⟶ (0,2) mark the indegree of 0 & 2 and reduce by 1.

0 indegree → 1

2 indegree → 0.

Add (2) to the queue.

4	2	

Get (4) and mark indegree of nb. (0,1).

0 indegree 0. 1 indegree 1.

There is cycle then few node will be not visited.

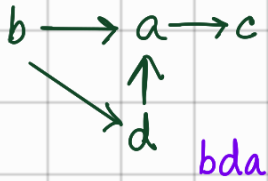
Alien Dictionary.

ba
abcd
abca
cab
cad

b a.
a bcd
b<a

a b c d
a b c a
d<a.

c a b
c a d
b<d



bdac.

a b c a
c a b
a<c

The order is unknown.
The dependency and the word given can be used to identify the order.

There is a new alien language that uses the English alphabet. However, the order of the letters is unknown to you.

You are given a list of strings `words` from the alien language's dictionary. Now it is claimed that the strings in `words` are **sorted lexicographically** by the rules of this new language.

If this claim is incorrect, and the given arrangement of string in `words` cannot correspond to any order of letters, return `""`.

Otherwise, return *a string of the unique letters in the new alien language sorted in* ***lexicographically increasing order*** *by the new language's rules.* If there are multiple solutions, return ***any of them***.

Example 1:

```
Input: words = ["wrt","wrf","er","ett","rftt"]
Output: "wertf"
```

2 words at a time.

aaa
aaaa

When a word is prefix of one then will come first.

Make the letter as a node

ⓐ ⓑ ⓒ ⓓ

0 1 2 3

Str[i] = baa
Str[i+1] = abcd.
b<a (no need to the entire string).

Make the DAG from the char and convert to number node.