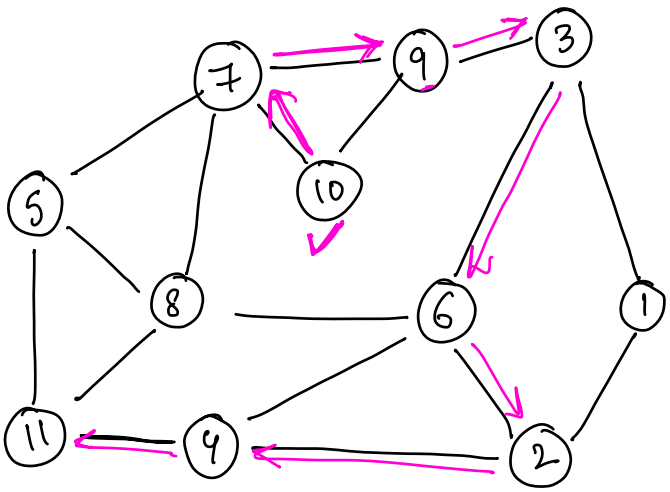
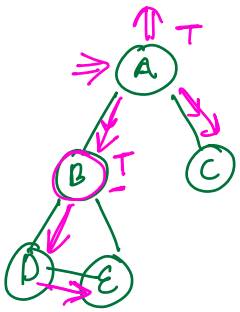
DFS: Depth first search

inorder, pre, post }





A | B C

B | D E A

D | B E

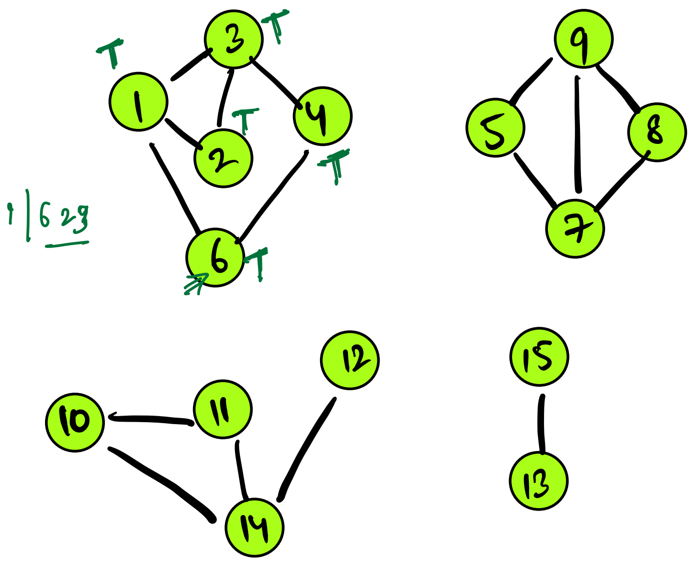
E | B D

```
void dfs(graph[], int u, visited)
{
    visited[u] = true;   print(u);

    for(i=0; i< graph[u].size(); i++)
    {
        int v = graph[u][i];
        if( !visited[v])
        dfs( graph, v, visited);
    }
}
```

T.C: O(n+m)

S.C: O(n)




```
dfs(2);
```

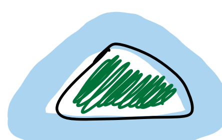
start from a source ⇐ { 1 connected compo, undirected

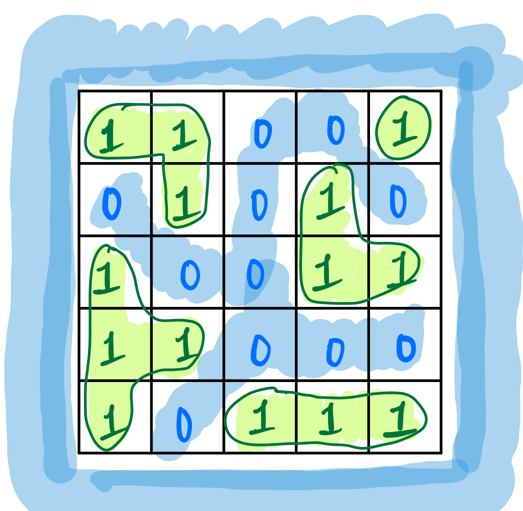
```
for( u=1; u<=n; u++)
{
    if( !visited[u])
    {
        count++;
        dfs(u);
    }
}
```

How many connected components?

DFS is never used in shortest path

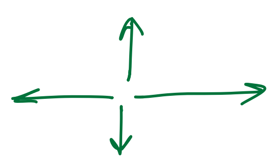
No of Islands



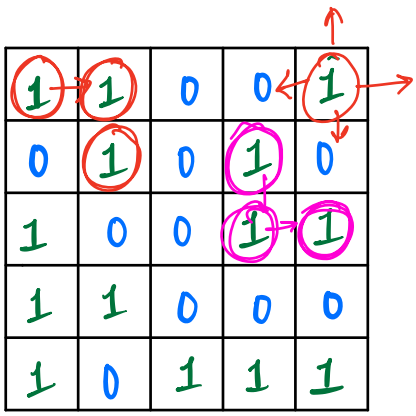


1 → land
0 → water

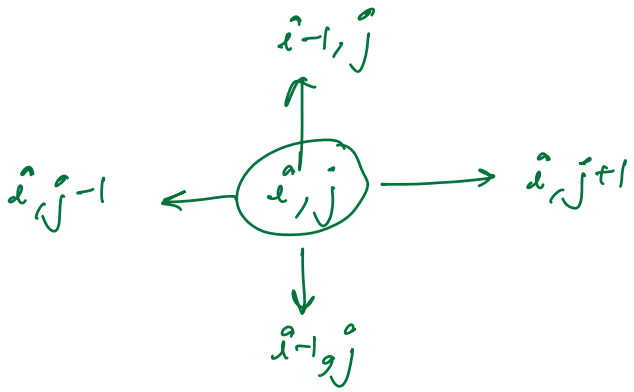
island ⇒ covered from 4 water



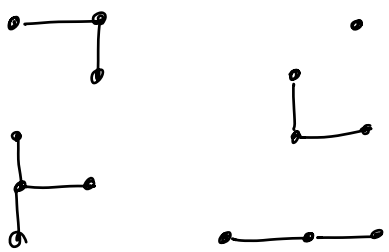
Every cell is kind of a node of a node of a graph



n*m



Every cell is acting as a node in the graph



use same array as visited

```
for( i=0; i<n; i++)
{
    for(j=0; j<m; j++)
    {
        if( mat[i][j] != 0)
        { count++;
          dfs( i, j);
        }
    }
}
```

T.C: n*m + 4*n*m
≈ O(n*m)

S.C: O(n*m)

```
dfs( i-1, j);
dfs( i, j-1);
dfs( i+1, j);
dfs( i-1, j);
```

x, y

$x+dx, y+dy$

$dx[4] = \{-1, 1, 0, 0\}$

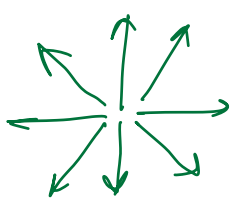
$dy[4] = \{0, 0, -1, 1\}$

$dx[8] = \{-1\ 1\ 0\ 0\ 1\ 1\ -1\ -1\}$

$dy[8] = \{0\ 0\ -1\ 1\ 1\ -1\ 1\ -1\}$

```
void dfs( int i, int j, mat)
{
        mat[i][j] = 0;
    for(d=0; d<4; d++)
    {
        x = i + dx[d];
        y = j + dy[d];
      if( x>=0 && y>=0 && x<n && y<m
          && mat[x][y]!=0)
          dfs(x,y)
    }
}
```

1	1	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	0	0	0
1	0	1	1	1



Rotten Oranges

mat[N][M] → 0 empty
1 fresh
2 Rotten

Every minute any fresh orange adjacent to a rotten orange becomes rotten, find min time when all oranges become rotten. If not possible return -1.

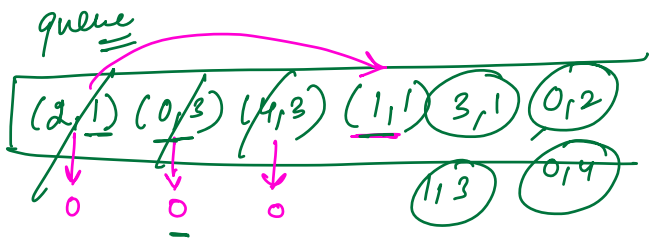
adjacent → 4 dir^n

	0	1	2	3	4
0	~~1~~ 3	0	~~1~~ 3	0	~~1~~ 5
1	~~1~~ 2	~~1~~ 1	~~1~~ 2	~~1~~ 3	~~1~~ 4
2	0	2	0	~~1~~ 4	0
3	0	~~1~~ 1	~~1~~ 2	~~1~~ 3	~~1~~ 4
4	~~1~~ 3	~~1~~ 2	~~1~~ 3	0	0

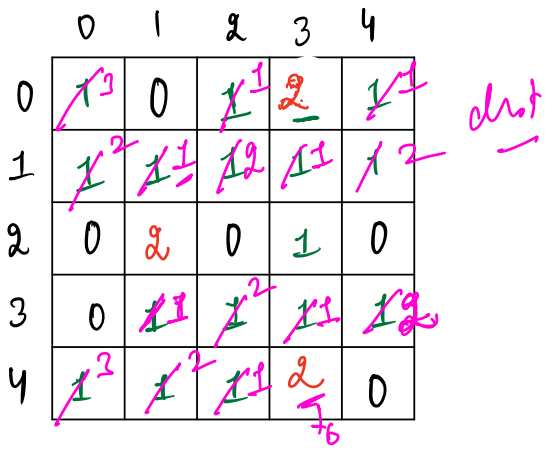
Bfs

to make visited
↓
just set it
mat[i][j] = 2

	0	1	2	3	4
0	1	0	~~1~~ 2	2	~~1~~ 2
1	1	~~1~~ 2	1	~~1~~ 2	1
2	0	2	0	1	0
3	0	~~1~~ 2	1	~~1~~ 2	1
4	1	1	~~1~~ 2	2	0



Multisource BFS



dist[x][y] = dist[i][j] + 1

Traverse the matrix initially & put every source in the queue

just perform BFS

ans = max of distance matrix

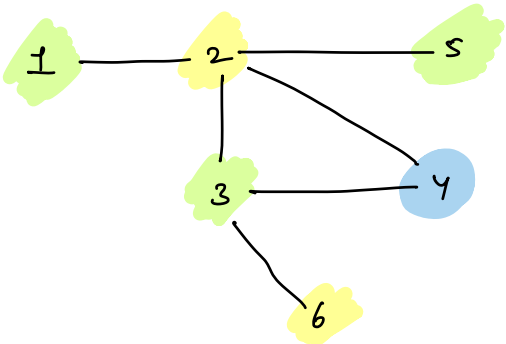
1. Insert every source in que
2. perform BFS
3. find max time

T.C: O(n*m)

S.C: O(n*m)

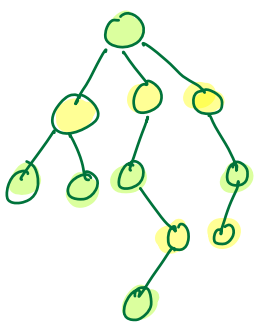
Graph coloring

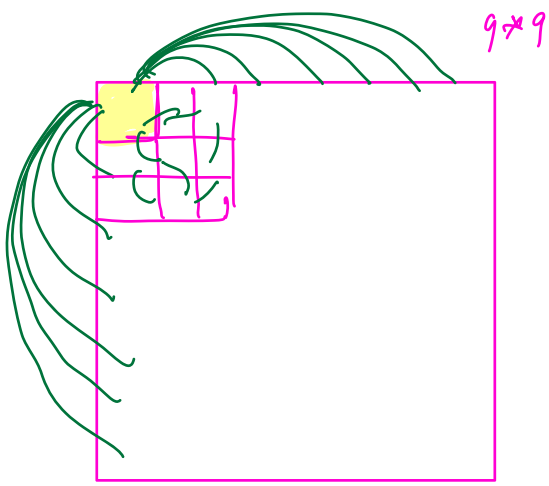
color every node of graph s.t. no 2 adjacent nodes have same color



minimum no of colors = chromatic no

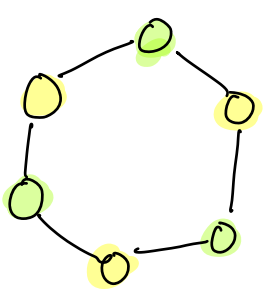
Tree = chromatic no



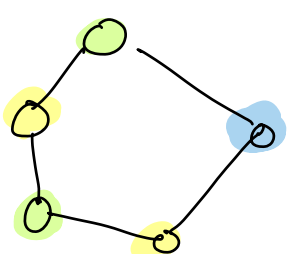


- cycle graph

ch = 2



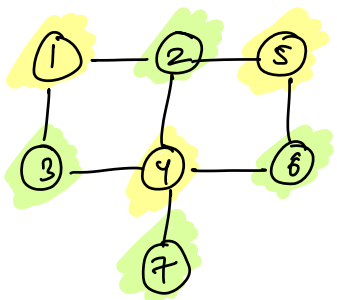
even = 2

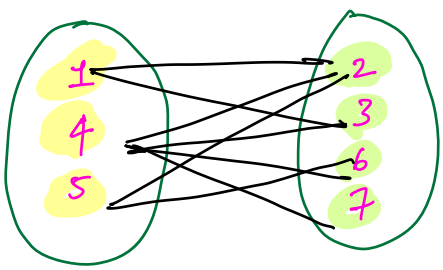


odd = 3

Bipartite graph :

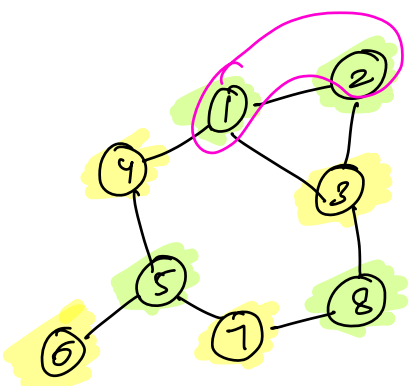
↳ if all nodes can be divided into 2 sets st. no 2 nodes in the same set which are adjacent.





graph, $C \cdot N = 2$ ⇒ Bipartite graph

Trees – Bipartite



color[i] = -1;

color → 0
color → 1

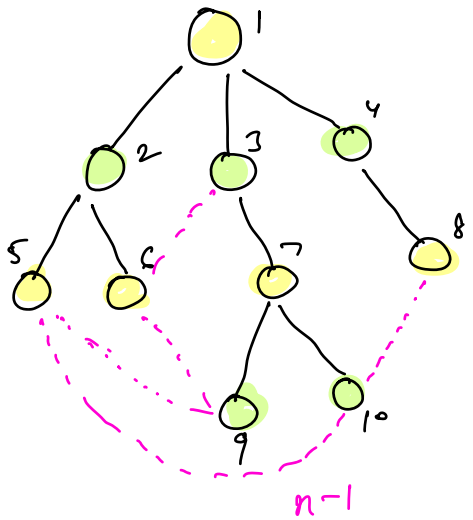
```
bool dfs ( int source)
{
    for( all neighbour of source)
    {
        if ( color[neigh] == -1)
        {
            color[ neigh ] = 1 ^ color[source]
            if(!dfs( neigh)) return false;
        }
        if ( color[ neigh ] == color[source])
        {
            return false.
        }
    } return true;
}
```

#

given a tree with N nodes.

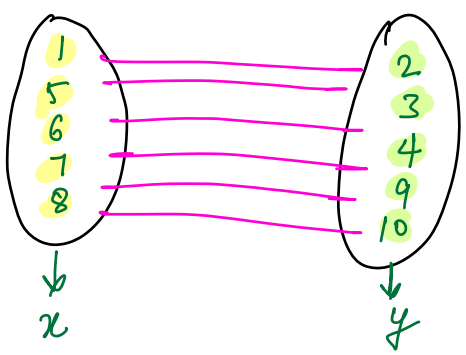
Find the max number of edges which can be added to the tree such that it remains bipartite.

n nodes
↓
n-1 edges



no of nodes at even level

no of nodes at odd level



$x * y - (n-1)$