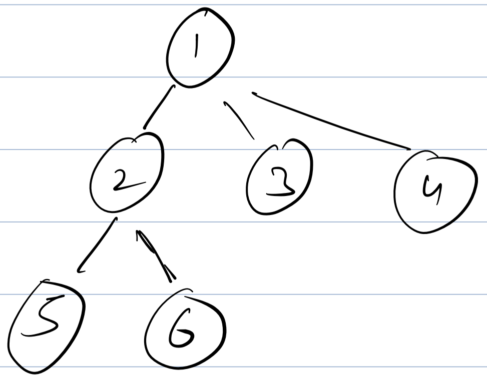
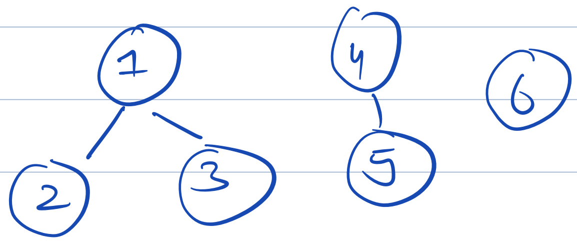


1, 2, 5, 6, 3, 4.

1, 2, 6, 5, 3, 4

1, 2, 6, 5, 4, 3

DFS

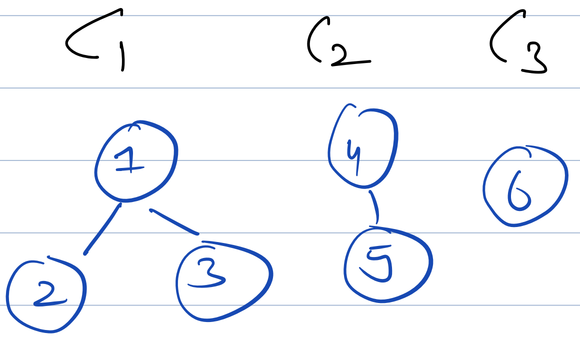


= 1 2 3 4 5 6

```
for (int i=1 ; i ≤ 6 ; i++) {
        if (!visited [i])
                dfs (i);

}
```

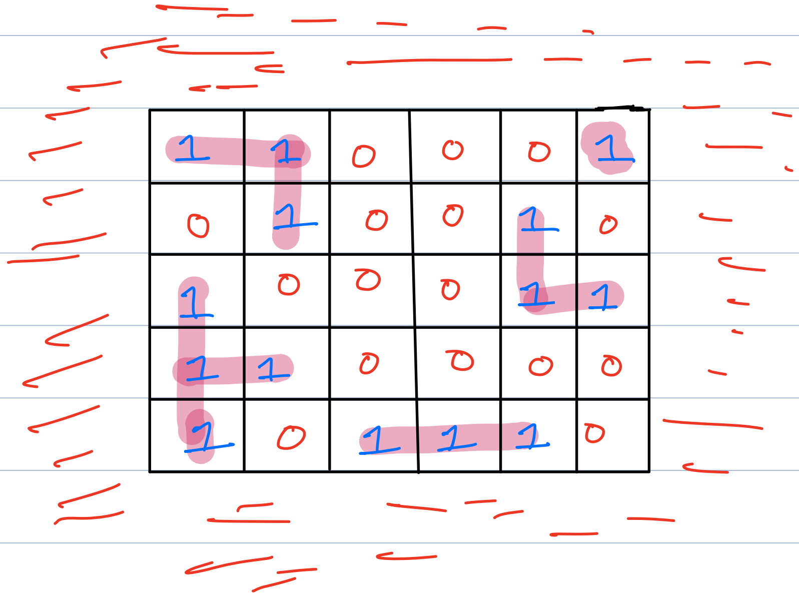
1	2	3	4	5	6
1	1	1	1	1	1

Q Count the number of connected components.

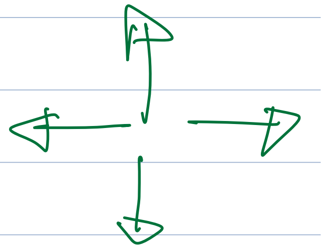


```
cnt = 0;

for (int i=1 ; i ≤ 6 ; i++) {
    if (!visited [i]) {
        dfs (i);
        cnt++;
    }
}
    return cnt
```

Number of Islands

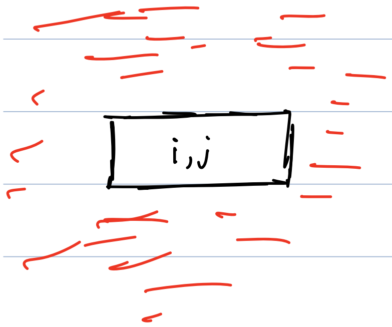


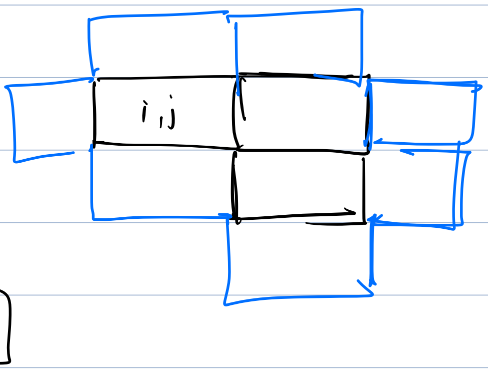
1	1	0	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
1	1	0	0	0	0
1	0	1	1	1	0

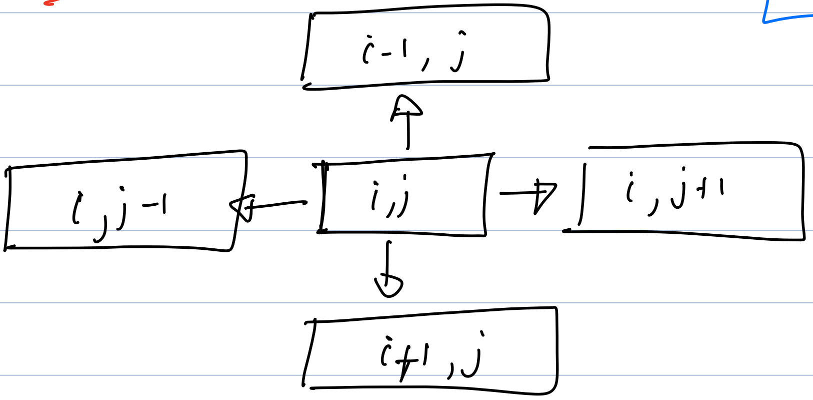


R, D, L, T

no of islands — 5





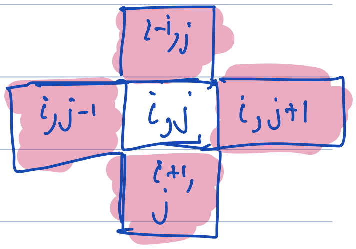


We will use the input 2D matrix to track visited nodes

mat [i] [j] → 1 : land, unvisited
→ 0 : Water
→ 2 : land, visited

```
for (int i = 0; i<n ; i++) {
    for (int j=0 ; j<m ; j++) {
        if (mat [i] [j] == 1) {
            dfs (i,j)
            cnt++;
        }
    }
}
    return cnt;

bool checkIfValid (int i , int j, int n, int m) {
    if (i<0 || i≥n || j<0 || j≥m)
            return false
    return true;
}
```

```
void dfs (int i, int j) {

        mat [i] [j] = 2;
        int x[4] = {1, -1, 0, 0}
        int y [4] = {0, 0, 1, -1}.

        for (int k=0 ; k<4 ;k++ ) {.
              int x_n => i + x[k];
              int y_n => j + y [k]

              if ( checkIfValid (x_n, y_n) &&.
                              mat [x_n] [y_n] = = 1)
                        dfs ( x_n, y_n);

    }

}
```



$$TC : O(V + E)$$

$V = mn$

$E = 4mn.$

$$\underline{\underline{TC : O(mn)}}$$

$$\underline{\underline{SC : O(mn)}}$$

Rotten Oranges.

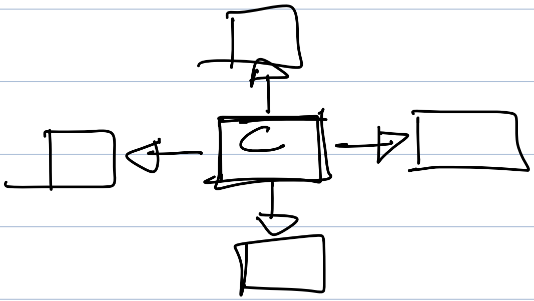
In how many days every living area is covid affected

C	1	1	0	1	0
1	1	0	1	1	0
0	0	1	C	0	1
C	1	1	0	1	1
1	1	0	0	1	C

C ⇒ affected by covid.

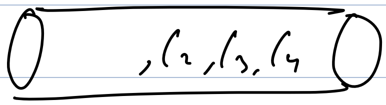
1 ⇒ not affected area

0 ⇒ no life present.



DAY 1

C	1	1	0	1	0
1	1	0	1	1	0
0	0	1	C	0	1
C	1	1	0	1	1
1	1	0	0	1	C



DAY 2

C	1	1	0	1	0
1	1	0	1	1	0
0	0	1	C	0	1
C	1	1	0	1	1
1	1	0	0	1	C

DAY 3

C	1	1	0	1	0
1	1	0	1	1	0
0	0	1	C	0	1
C	1	1	0	1	1
1	1	0	0	1	C

Ans ⇒ 3 days

C	1	1	0	1	0
1	1	0	1	1	0
0	0	1	C	0	1
C	1	1	0	1	1
1	1	0	0	1	C

1) Traverse the matrix and insert covid cells in the Queue.

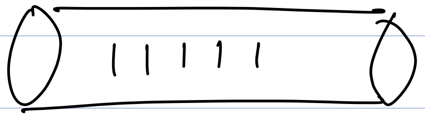
2) Perform BFS and maintain level.

⇓

Whenever from (i,j) you visit a neight (x,y)

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

C	1	1	0	1	0
1	1	0	1	1	0
0	0	1	C	0	1
C	1	1	0	1	1
1	1	0	0	1	C

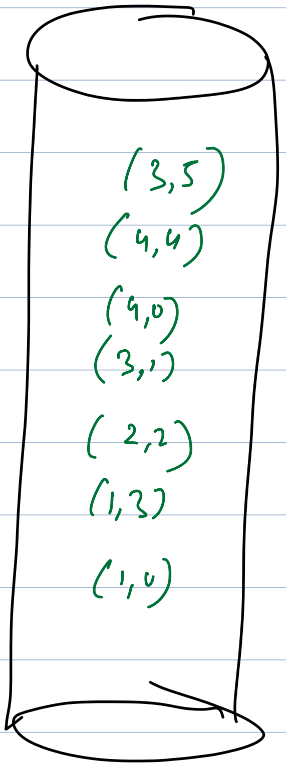


Size = 8

max_level = 2

10:32pm

	0	1	2	3	4	5
0	C	1	1	0	1	0
1	1	1	0	1	1	0
2	0	0	1	C	0	1
3	C	1	1	0	1	1
4	1	1	0	0	1	C

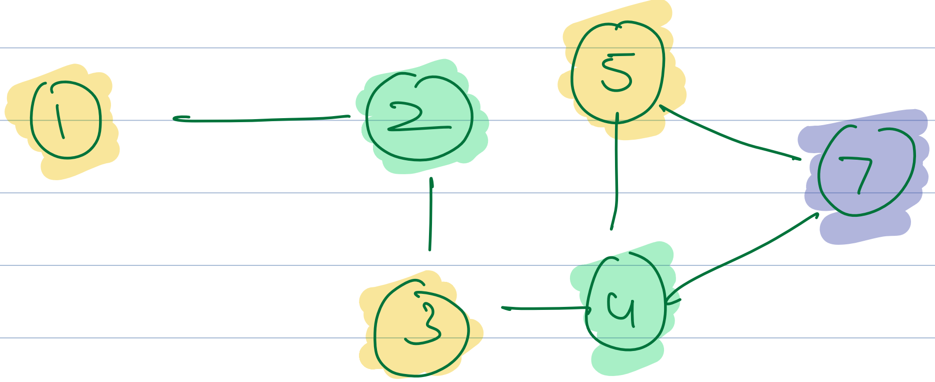


	0	1	2	3	4	5
0	0	1	2		3	
1	1	2		1	2	
2			1	0		2
3	0	1.	2.		2	1
4	1	2			1	0

(0,1)

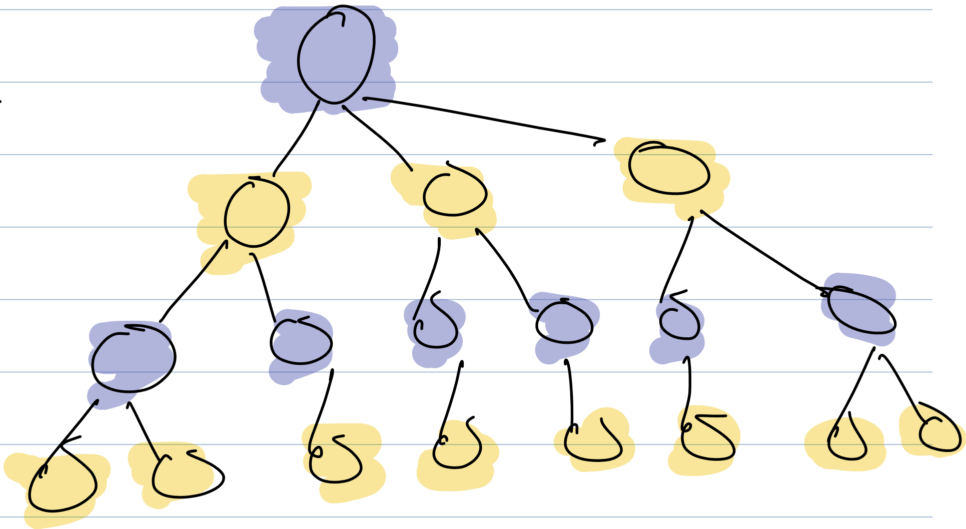
Chromatic numbers ⇒ np hard = [no polynomial time solution]

Min number of colors required to color the graph such that no 2 adjacent nodes have the same color.

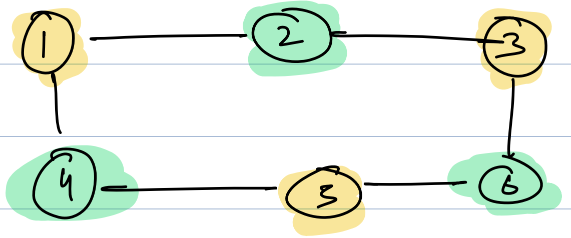


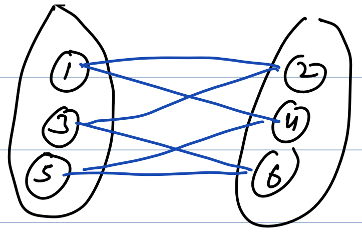
Chromatic number = 3

Tree

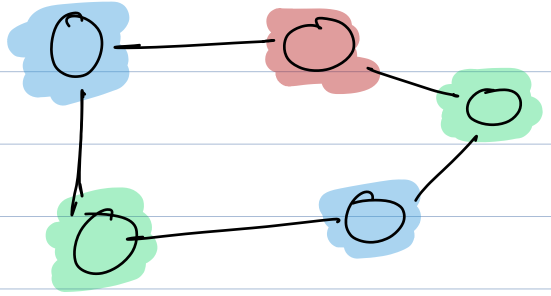


Chromatic number for a tree = 2



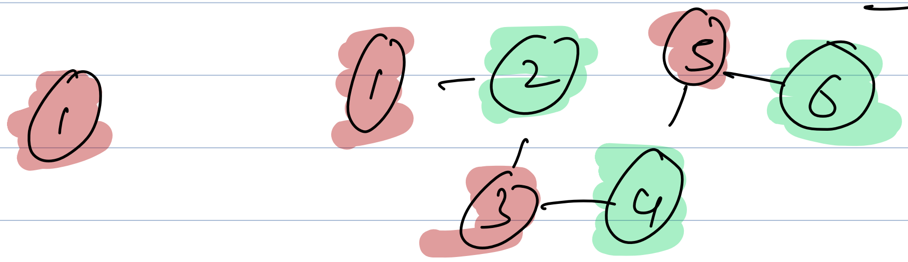


Even Cycles
chromatic number = 2



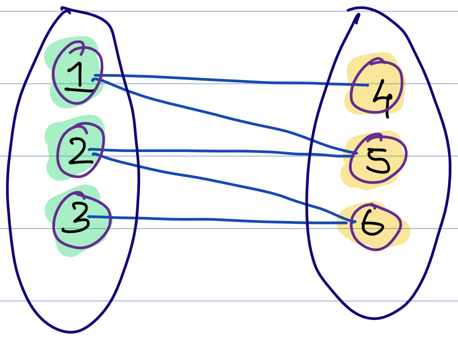
Odd Cycles
chromatic number = 3

For any acyclic graphs the chromatic number ≤ 2

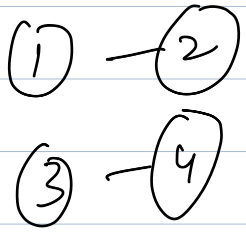


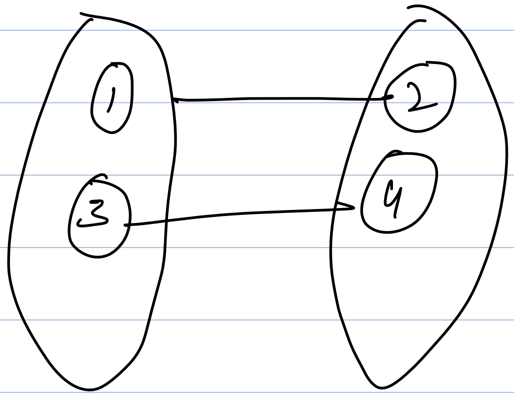
Bipartite graphs

A graph which can be broken down into two distinct sets U & V such that no two nodes in U & V have an edge b/w them.



Chromatic number of a bipartite graph is 2 ,

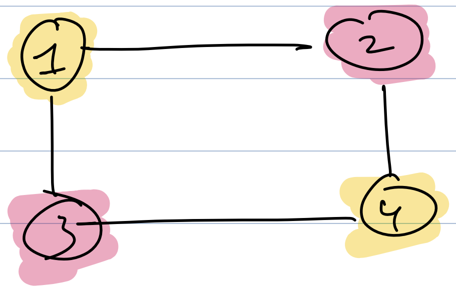




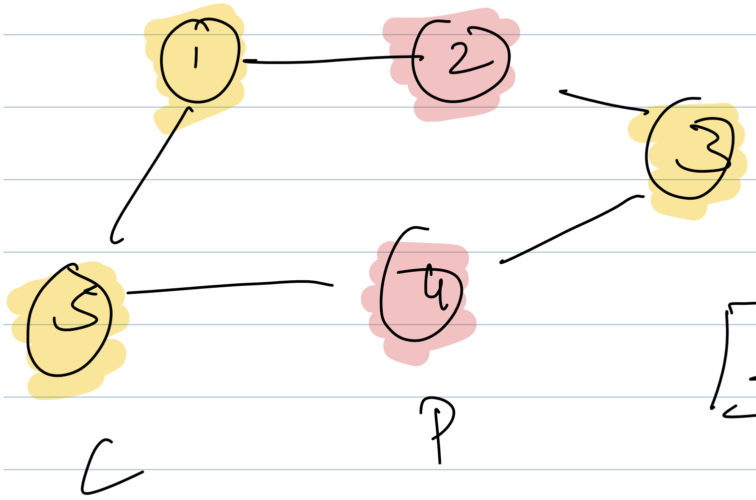
Even length cycles are bipartite.
Trees are bipartite.
Acyclic graphs are bipartite.

🟨 = 1

🟥 = 0



1	0	0	1
1	2	3	4



1	0	1	0	1
1	2	3	4	5

Pseudo Code : (1, 0) Colors → 0
→ 1

```
int visited [n]; int calor [n] , isBipartite = true;

void dfs (int node, int c) {

    visited [node] = 1;
    color [node] = c;

    for (int i=0 ; i< adj [node].size() ; i++) {

        int child = adj [node] [i];

        if (visited [child]) {
            if (color [child] == color [node])
                isBipartite = false;
        } else {
            dfs (child, 1-c)

        }
    }
}
```

0 → 1

1 → 0

1-c

1^c

TC : O(V+E)

SC : O(V+E)