Graph Coloring

Chromatic Number: Min number of colours you need to paint all nodes such that no 2 nodes adjacent have same colour

{2 nodes connected directly by edge}

Eg:

Given N Nodes → N = 4

Max Chromatic Number: N

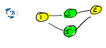
N nodes are connected

N = 4

2 Colours

{Bi-partite Graph}

Eg:



Maps:

$S_1 - S_3$

$S_1 - S_2$

{Given a map} → convert to Graph: 4 colour theorem

Max Chromatic

{104 y Cons}

Note: All T are Bi-partite Graphs

{For any tree we can have levels order traversal}

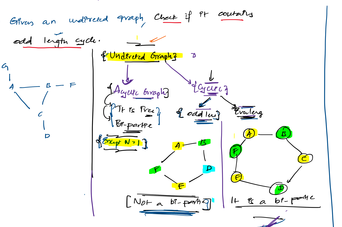
level : 0

level : 1

level : 2

level : 3

Q) Given an undirected graph Check if it contains odd length cycle.

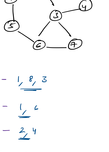


① If given undirected graph is not a bi-partite, then it contains odd length cycle.

② If given undirected graph is a bi-partite, then it contain even length cycle. → False

{It doesn't help to say it can be true as well}

① Given a connected how to check it is bi-partite or not



VS[] :

1	2	3	4	5	6	7	8

Queue

1	X	X	X	X	4

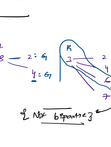
1 4 7 R 1 R 7

2 - 1, 8, 3

5 - 1, 6

8 - 2, 4

3 - 2, 4, 5, 7



→ using BFS

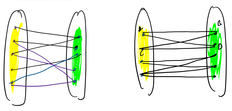
→ For a node if colour is assigning it is visited.

Col[v] → -1 : Not visited
0 : Visited, 4 colour Red
1 : Visited, 4 colour Green

T.C : O(N+E) S.C : O(N)

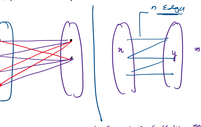
```
bool IsBool (vector<int> g[], int col[], int n){
    Queue<int> q;
    q.push(0); col[0] = 0;
    while( q.size()>0)
        int u = q.front();
        q.pop();
        for( int v : g[u]){
            if( col[v] == -1) {
                col[v] = 1 - col[u] //Assig colour
                q.push(v)
            }
            else {
                if( col[u] == col[v]) return false
            }
        }
    return True;
}

main() {
    int ans = true;
    // col[N+1] = -1;
    // Construct g[]
    for(int i = 1; i < N; i++) {
        if( col[i] == -1){
            ans = ans && IsBoll( g, col, i)
        }
    }
    return ans;
}
```

A graph is bi-partite



Q) Given a disjoint Complete bi-partite graph how many max edges we can add such that graph is still bi-partite

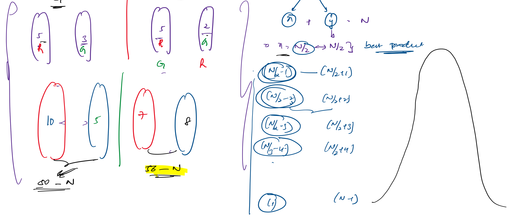
Eg:



→ {x*y - N Edges}

→ You need to use n & y {to count of x & y we get using colors[].}

→ In above graph say we have more than 1 component?



Friendship

1) Given a {friendship graph N people} are divided to parity.

But a person (x) can move parity if {(N-1) at least friends}

{Neighbour we can also attend party}

connected by edge.

Eg:

K = 2

Ans = 3

Idea = 2 → Not able go

K = 1 ✓

K = 2 except 6 all above

K = 3, 3, 4, 5 R

Idea:

1) Construct a m[] contains no. of edge are given from every node. / → { length of adj list for every elem}

2) Node with neighbour < K they cannot attend party.

Pick those all in queue.

m[]:

0	1	2	3	4	5	6	7	8
	2	X	X	3	3	1	X	3	0
	7	3	X				8	2	
		4							

q: { X | X | X | X }

{ (7) nodes }

N - Element pushed in queue

{All elements pushed in queue got party}

1 - 2, 3, 7

2 - 3, 1, 7

3 - 7, 4, 5, 8, 2

4 - 5, 8 - 3

5 - 8, 4, 3

6 - 7

7 - 1, 2, 3, 4

8 - 3 4 5

Pseudo Code

1) Construct m[]

2) Insert all elem who have len than k

for i = 0;

while(q.size() > 0) {

for u = q.front();

q.pop() {q.} 3 element popped count}

for(int v : g[u]){

if (m[v] < k) { Have already to queue continue }

else { m[v]--; if (m[v] < k) { q.push(v) } }

}

}

return N - c

{Total nodes, Not Element deleted}

u - v

u - v

Edge: { u.push(v) v.push(u) }

Adj Matrix

Adj list: O(N+E)

Sessions g[N] : { g[u].add(v) g[v].add(u) }