Todays Content:

: Cycle detection in Undirected graphs

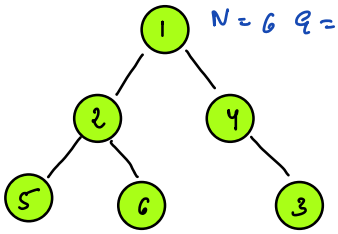
: Bi-partie & Chromatic Number

: Colouring Tree :

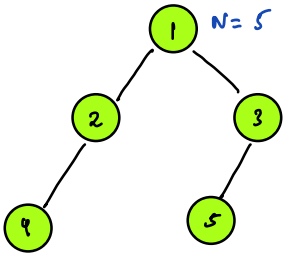
-.

Cycle detection in Undirected

: In Tree with N Nodes, how many Edges = N-1 Edges



N = 6 E = 6



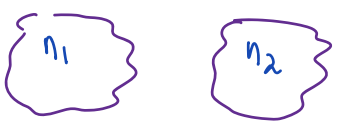
N = 5 E = 4

Note: Even if we add 1 more edge in Tree it becomes Cycle.

Given graph with N Nodes & C connected components?

Nodes	Components	# Edges it should have so that there is no cycle?
N	1	Edges = N-1 so that no cycle

	nodes	edges	
N	2		
C_1	n_1	$n_1 - 1$	Total Edges = $n_1 + n_2 - 2$
C_2	n_2	$n_2 - 1$	= N-2



Obs: If 2 components, Edges = N-2 no cycle

	nodes	edges	
N	3		
C_1	n_1	$n_1 - 1$	Total Edges = $n_1 + n_2 + n_3 - 3$
C_2	n_2	$n_2 - 1$	= N-3
C_3	n_3	$n_3 - 1$	

Obs: If 3 components, Edges = N-3 no cycle

	nodes	edges	
N	n		
C_1	n_1	$n_1 - 1$	Total Edges = N - n
C_2	n_2	$n_2 - 1$	
⋮	⋮	⋮	
C_n	n_n	$n_n - 1$	

Obs: If Graph with N nodes have n components, Edges = N-n no cycle

Q1: Given N Nodes & E Edges Detect Cycle in Undirected Graph

Step1: Calculate no: of Connected Components = x

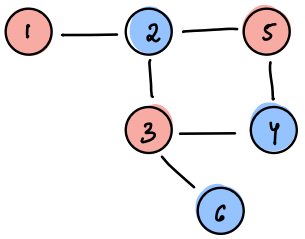
T.C: O(N+E)
S.C: O(N+E)

Step2:
```
if( E == N-x){
    print(no cycle)
}
else{
    print(cycle)
}
```

Chromatic Number / Graph Colouring

Min Colour req to colour every node of graph such that no 2 adjacent nodes have same colours, called Chromatic number

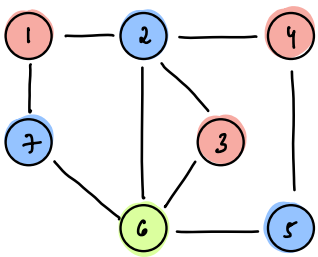
Eg1:



ans = 2

Bi-partite Graph

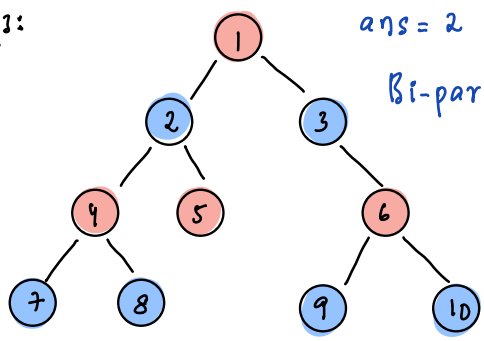
Eg2:



ans = 3

Not Bipartite

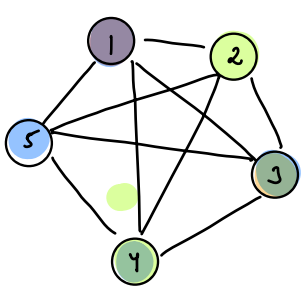
Eg3:



ans = 2

Bi-partite

Eg4:



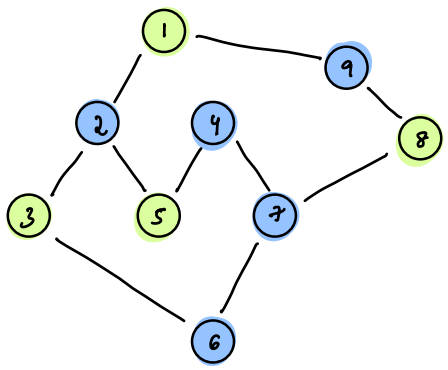
ans = 5

Not Bi-partite

Bipartie Graph:

: A Graph is said to be bi-partie, if it's chromatic number = 2

Q. Given a graph check if it's bi-partie or not?



0 → no colour

1 → green colour

2 → blue colour

B	G	G
6 :	3	7

→ We need to paint with green but already painted with Blue {* not bipartie}

~~1~~	~~2~~	~~9~~	~~5~~	~~5~~	~~8~~	~~6~~	4	7

	0	1	2	3	4	5	6	7	8	9
col :		G	B	G	B	G	B	B	G	B

```
bool bi-partic(int N, int E, int u[], int v[]){
```

TC: O(N+E)
SC: O(N+E)

10:35pm

Step1: Construct Adj list

list<int> g[n+1] TODO

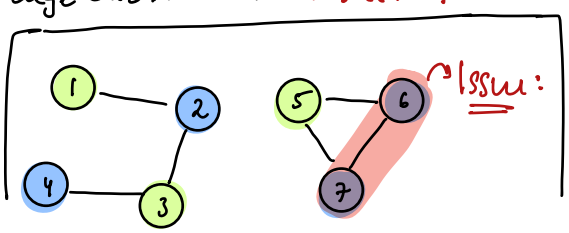
Step2: Queue<int> q;

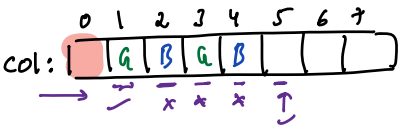
int col[n+1] = 0 // 0 - no colour 1 - Green 2 - Blue

```
for(i=1; i<=n; i++){
    if(c[i]==0){
        col[i] = 1 or 2 // Assign any Colour it will work
        q.push(i)
        while(q.size()>0){
            int u = q.front()
            q.delete() // delete front element
            // iterate on adj nodes of u
            for(i=0; i<g[u].size(); i++){
                int v = g[u][i] //   u (c[u]) ——— v (c[v])
                if(c[v]==0){ // colour to v is not assigned
                    c[v] = 3-c[u]; q.push(v)
                }
                else if(c[u]==c[v]) // Colour to v is assigned
                    // 2 adjacent nodes same colour, not bipartite
                    return false
            }
        }
    }
return true
}
```

if(c[u]==1){c[v]=2}
else{c[v]=1} q.push(v)

Edge Case 1: N=7: Issue: If there are multiple components, check it in Each Component



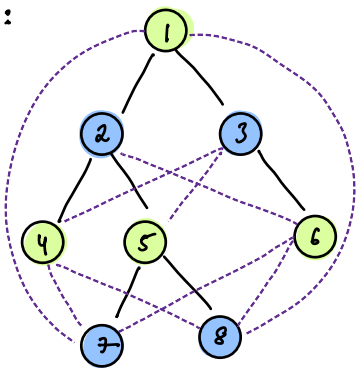


4Q) Given a Tree in Graph format, how many more edges we can add to tree, so that it remains bi-partie.

↳ : Tree : Is always bi-partie graph
Colour alternating colours at levels

↓

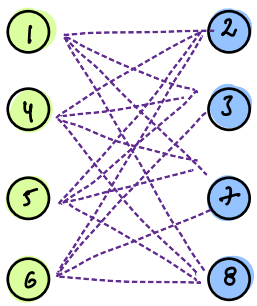
Eg1:



ans = 9

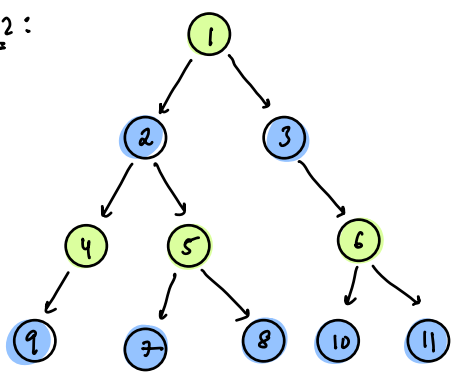
Hint: Edges we added between nodes of different colours ?

// Nodes Green // Nodes Blue



We can have 16 Edges
Given = 7 Edges
Add = 9 Edges

Eg2:



// Nodes Green // Nodes Blue

G = 4 + 7 = 8
Edges = 28
Given = 10
Add = 18

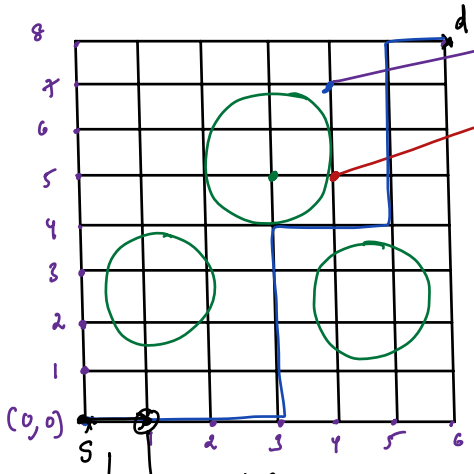
Obs: Calculate no: of nodes in Green = CG & Blue = CB

↳ Final ans = CG * CB - Edges

↓ : ?

```
int addbiparbie( int N, int E, int u[], int v[]) {

    Step1: Construct Adj list                    TC: O(N+E)  SC: O(N+E)
           list<int> g[n+1] TODO

    Step2: Queue<int> q;
           int col[n+1] = 0 // 0-no colour  1- Green  2- Blue
           col[1] = 1 or 2 // Assign any colour it will work
           q.push(1)
           while(q.size()>0){
               int u = q.front()
               q.delete() // delete front element
               // iterate on adj nodes of u
               for(i=0; i< g[u].size(); i++){
                   int v = g[u][i] //   u ---- v
                                       c[u]   c[v]
                   if( c[v]==0) { // colour to v is not assigned
                       c[v] = 3-c[u]; q.push(v)
                   }
               }
           }

    Step3: int cG=0, cB=0
           for(i=1; i<=N; i++){
               if( c[i]== 1) { cG++}
               else { cB++}
           }
           return cG*cB-E

}
```

```
if(c[u]==1){c[v]=2}
else{c[v]=1} q.push(v)
```



point outside circle

point inside circle

$C = (5, 3)$ $r = 2$

$P = 7, 4$

dist (x_1, y_1) (x_2, y_2)

$= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

dist $(5, 3)$ $(7, 4)$

$= \left(\sqrt{\quad}\right)^2 > (2)^2$

$4 + 1$

$= 5, 4$

point is inside any of given circle

If inside any of circles: We cannot enter