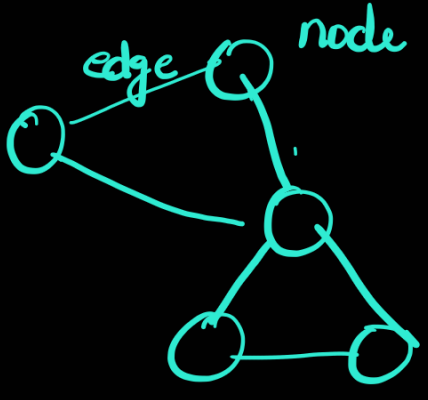
# DFS

Graph

↳ DFS + BFS enough for most problems




Many problems can be modelled like this

N students in a class

M relations (a friend to b)

1 2

2 3

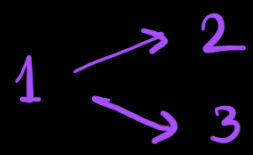
1 3

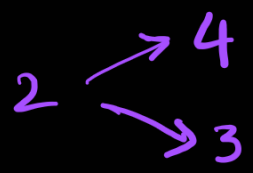
2 4

5 6

Rumor to $i^{th}$ person to what many number of people will it reach

AlgoZenith

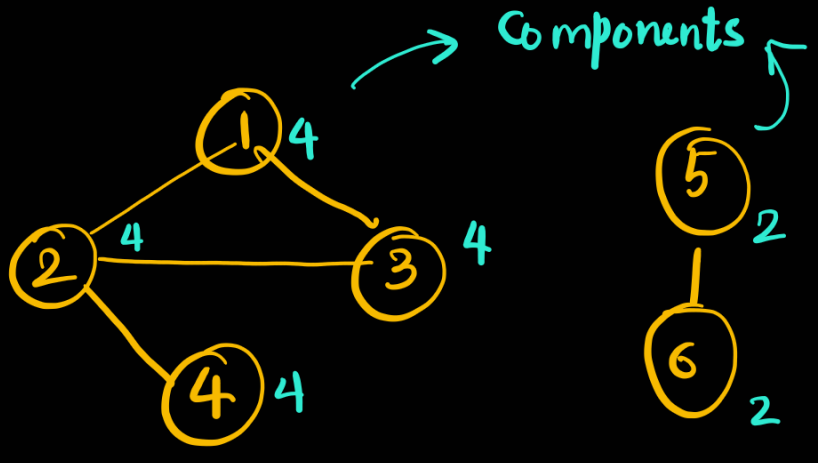



X Already knows

5 6 will not know the rumor

A friend with B, B is friend with A.




Node - person

edge - friend

if start from 1, all people reachable from 1 know the rumor

find size of component in which i lies in

Component has all nodes reachable from one another

We use DFS for this.

Input

N M

Edges...

for each nbr

if v is unvisited

visit

dfs on it

dfs(x) - Explore everything reachble from x

To track what is explored, not explored maintain visited array

vis = [0, 0, 0, 0, 0, 0, 0]

```
dfs (x)
    vis[x] = 1
    for (all nbr v of x)
        if (!vis[x])
            DFS(v)
```

→ for every node, dfs explores nbrs

Build a list of neighbours

| | |
|---|---|
| 1 → 2 3 | 6 5 |
| 2 → 1 3 4 | 1 2 |
| 3 → 2 1 | 2 3 |
| 4 → 2 | 1 3 |
| 5 → 6 | 2 4 |
| 6 → 5 | 5 6 |

↓

Adjacency list - easily get neighbour
Store neighbours in a vector

N nodes M edges
Time, Memory required for adjacency list

first build

① Create empty adj list  O(N) T  O(N) M

1 → []
2 → []
3 → []
4 → []
5 → []
6 → []

vector < vector <int >> g(N+1)

for (edge e)
  e = (u, v)
  g[u].add(v)
  g[v].add(u)

② Process edges

(u, v)  O(2) T  O(2) M

2 elements → 2 * E

N nodes

Bothe time, memory - O(V + 2E)
- O(V + E)

```
int n, m;
vector <vector <int >> g;

void solve ( ) {
    cin >> n >> m;
    g. resize (n+1);
    for ( int i=0; i<m; i++){
        int a, b;
        cin >> a >> b;
        g[a]. pb(b);
        g[b]. pb(a);
    }
}
```

g[node] → Immediate neighbours of node

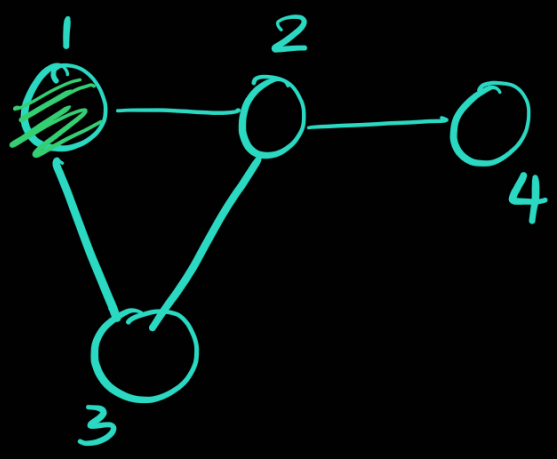
```cpp
int n,m;
vector<vector<int>> g;

vector<int> vis;
void dfs(int node){
    vis[node]=1;
    for(auto v:g[node]){
        if(!vis[v]){
            dfs(v);
        }
    }
}
```

```cpp
void solve(){
    cin>>n>>m;
    g.resize(n+1);
    vis.resize(n+1); -> remember to reset
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    for(auto v:vis)cout<<v<<" ";cout<<endl;
    dfs(1);
    for(auto v:vis)cout<<v<<" ";cout<<endl;
}
```

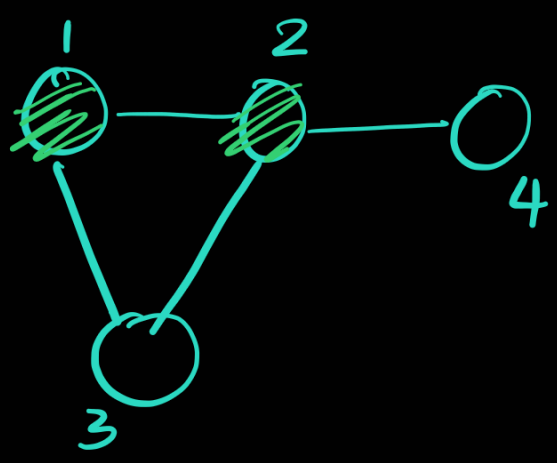
DFS(1)

[2 3 4]
↑



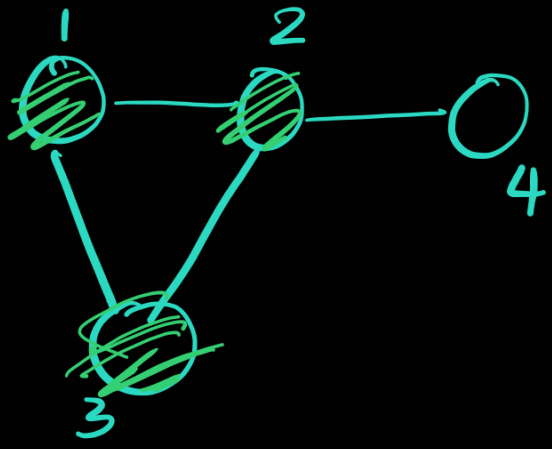

DFS(2)

[1 3 4]
↑ ↑
x




DFS(3)

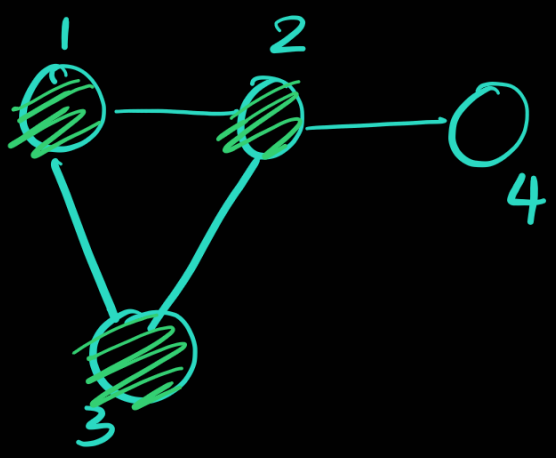
[1 2]
↑ ↑
X X




AlgoZenith

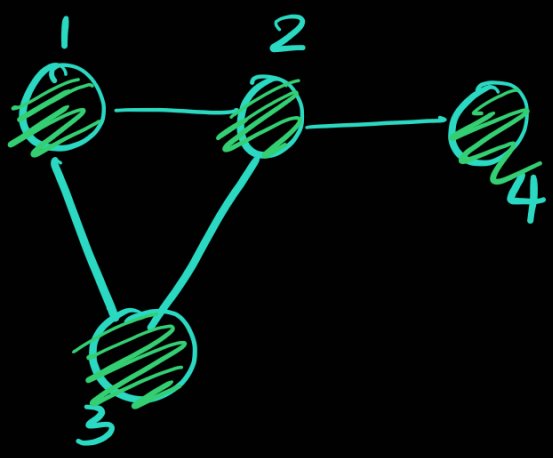
DFS (2)

[1 3 4]




DFS (4)

[2 3]




DFS (2)

DFS (1)

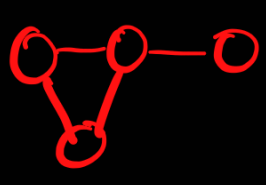
return

```cpp
for(int i=1;i<=n;i++){
    vis.assign(n+1,0);
    dfs(i);
    int cnt = 0;
    for(auto v:vis)cnt+=v;
    cout<<cnt<<endl;
}
```

Time Complexity

Can dfs affect unreachable nodes? NO

V nodes
E edges
} of a single component



DFS(1) ?

dfs of every node is called exactly once because of visited

DFS(1)
  DFS(2)
    DFS(3)
    DFS(4)

Mark visited,
iterate over nbs

for every node

O(1)

O(Number of nbrs)

} computation done

$$\sum_{i=1}^{Nodes} (1 + \#\ nbrs)$$

$= V + E$

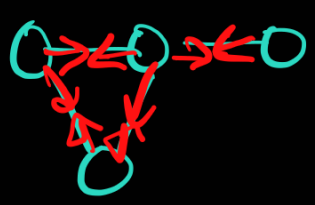
of the component

where V is size

of reached graph

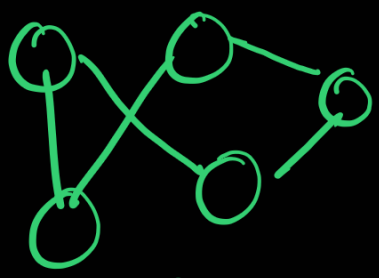
adjacency list [i] represents # of nbrs of i

Sum of nbrs $= 2 * E$



every edge adds 2 nbrs

Total neighbours $= 2E$



$O(V(V+E))$

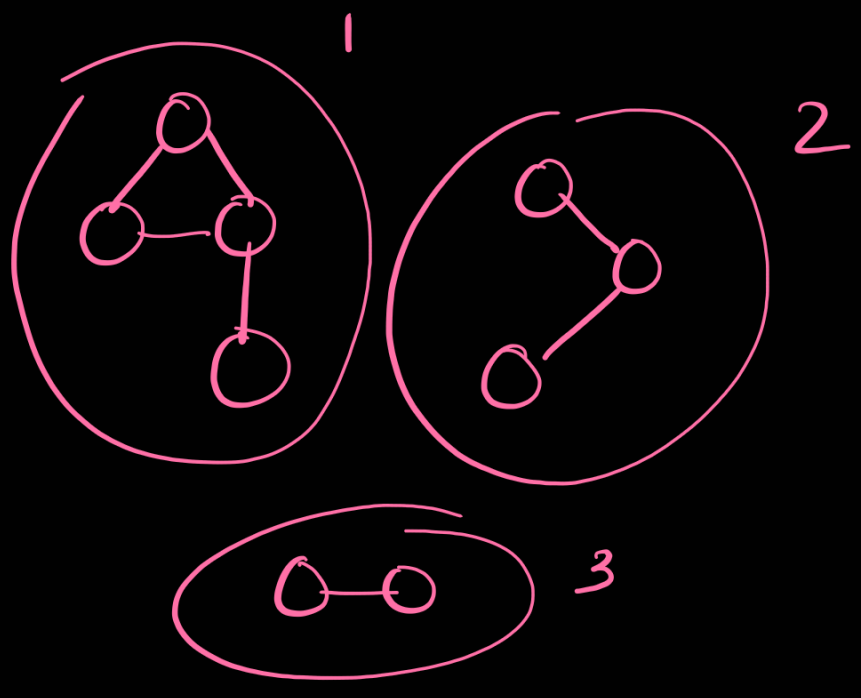
for every node DFS

$N, M \leq 10^5$

$10^5 (10^5 + 10^5) \approx 2 \times 10^{10}$ ✗ $10^8$ limit

This logic won't work

Strategy :- Component Numbering




Maintain numbering

Component no for every node

Maintain

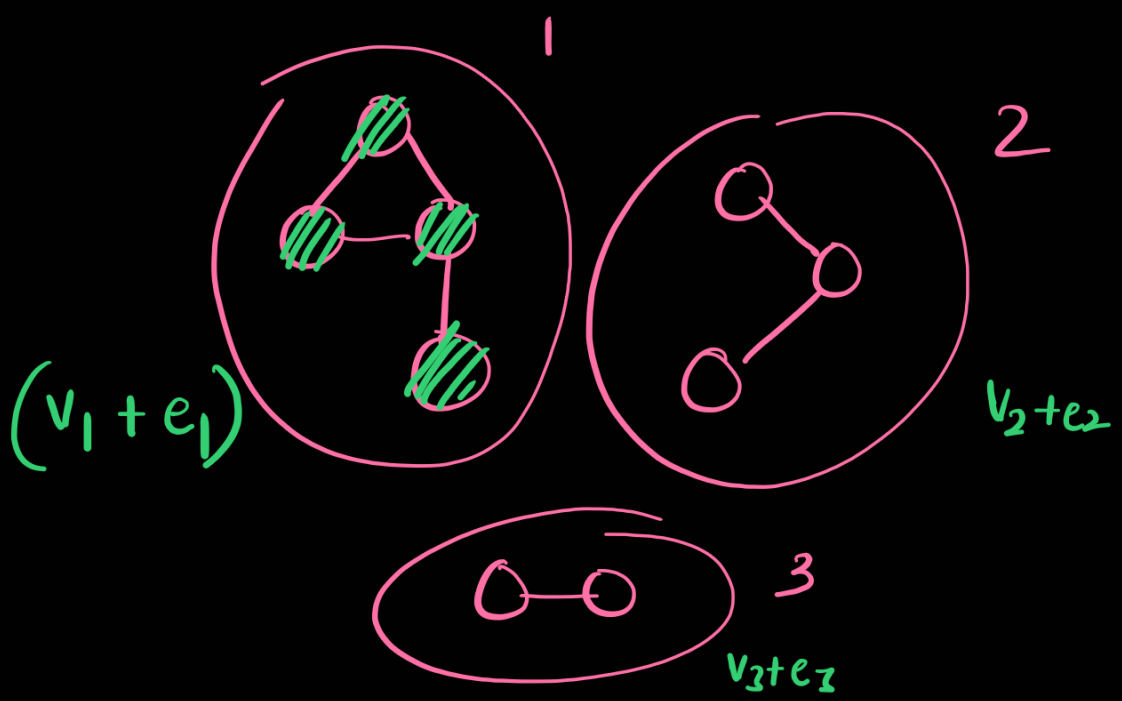
vector <int> comp_no;

```
int comp_no = 0;
for(int i=1;i<=n;i++){
    if(!vis[i]){
        comp_no++;
        dfs(i,comp_num);
    }
}
```

if a node is not visited, it belongs to a new component

```
vector<int> vis;
vector<int> comp_num;
void dfs(int node,int comp_no){
    vis[node]=1;
    comp_num[node] = comp_no;
    for(auto v:g[node]){
        if(!vis[v]){
            dfs(v, comp_no);
        }
    }
}
```

Mark all nodes in a component by same no.




DFS(1) marks everything

DFS(2) does nothing

First unvisited node is hit by for loop

DFS hits all nodes exactly once

$O(V+E)$

$$V_1 + e_1 + V_2 + e_2 + V_3 + e_3 = O(V+E)$$

for every component no., we can find size

```cpp
vector<int> vis;
vector<int> comp_num;
vector<int> comp_size;
void dfs(int node,int comp_no){
    vis[node]=1;
    comp_num[node] = comp_no;
    comp_size[comp_no]++;
    for(auto v:g[node]){
        if(!vis[v]){
            dfs(v, comp_no);
        }
    }
}

void solve(){
    cin>>n>>m;
    g.resize(n+1);
    vis.resize(n+1);
    comp_num.resize(n+1);
    comp_size.resize(n+1,0);
```

```cpp
for(auto v:comp_num)cout<<v<<" ";cout<<endl;
for(auto v:comp_size)cout<<v<<" ";cout<<endl;
```

```cpp
for(int i=1;i<=n;i++){
    cout<<comp_size[comp_num[i]]<<endl;
}
```

size(comp no) gives size

O(N+M) will pass

Queries

if x gets rumor, will y also get it?

O(1)

comp_no[x] == comp_no[Y]

N persons in a classroom

M relations

1 2

2 3

1 3 X

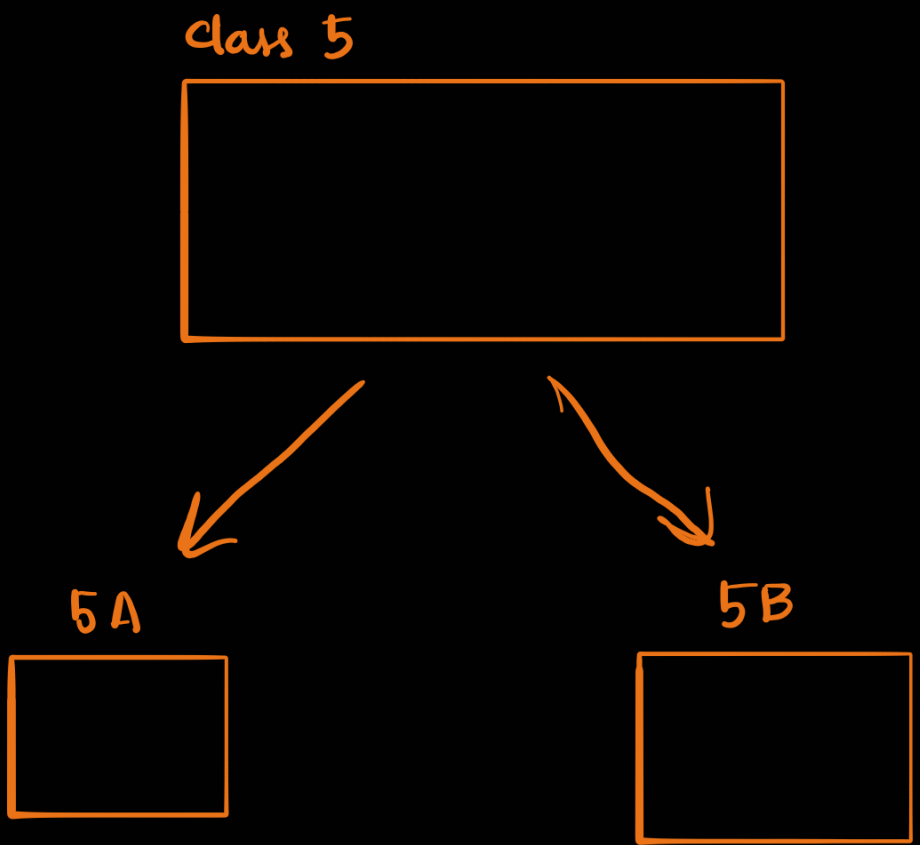
2 4

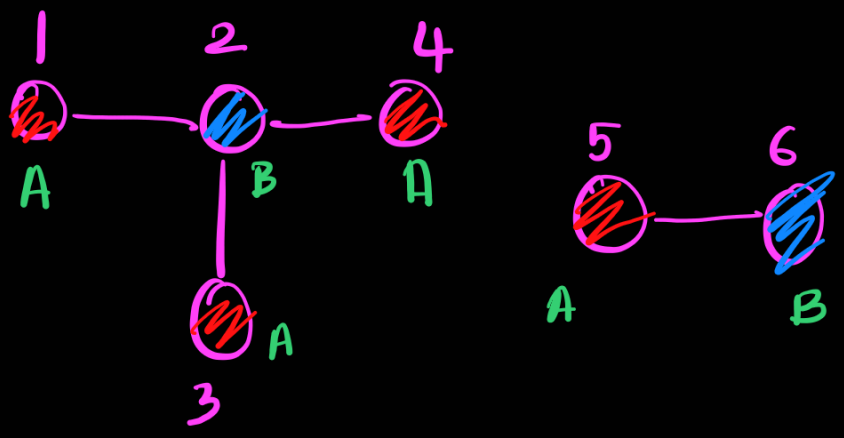
5 6

if 2 students are friends, they are noisy

No 2 friends should be in same class

Divide into 2 groups






2 different colors

Bipartite - Assign 2 colors s.t no two nbrs has same color

Simply color it

Maintain Vector<int> color for every node

```
void solve(){
    cin>>n>>m;
    g.resize(n+1);
    vis.resize(n+1);
    color.resize(n+1);

    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    for(int i=1;i<=n;i++){
        if(!vis[i]){
            dfs(i,1);
        }
    }
```

AlgoZenith

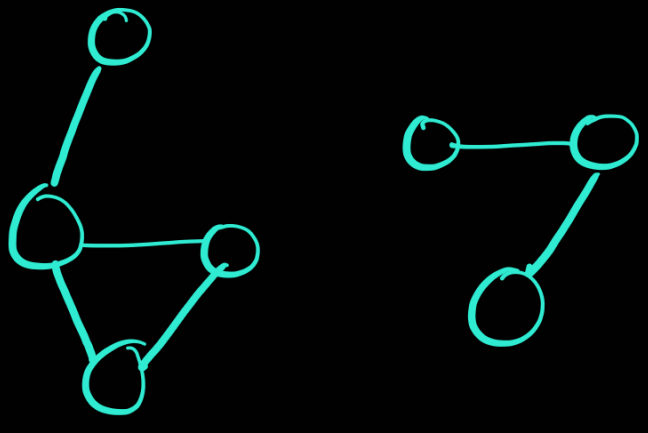
```cpp
bool is_bipartite = 1;

void dfs(int node,int col){
    vis[node]=1;
    color[node]=col;
    for(auto v:g[node]){
        if(!vis[v]){
            dfs(v,(1+2)-col);
        }else if(color[node]==color[v]){
            is_bipartite = 0;
            return;
        }
    }
}
```

Alternate

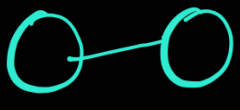
$1 \rightarrow 3-1=2$

$2 \rightarrow 3-2=1$



N nodes

How many maximum edges?



$^{N}C_{2}$

M has already been added

How many more edges to add to reduce Number of components