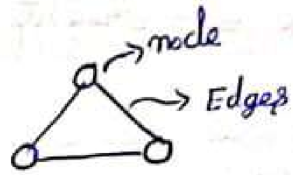
Graph DFS Usage

Graphs

4 parts
- → DFS / BFS
- → Shortest Path
- → Modelling / Union Find....
- → Trees

A graph is made up nodes & edges.



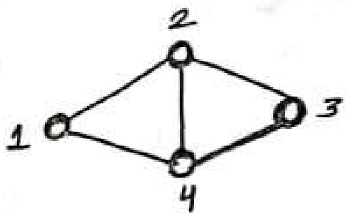
DFS :-

1) Visit a node → Mark it visited.
2) Recursively visit all ~~nodes~~ its unvisited neighbours.
3) DFS give information about all nodes reachable from a given node.

Pseudocode :-

```
DFS (x) {
   visited [x] = true
   for (v : neighbours [x]){
       if (! visited [v])  DFS (v)
   }
}
```

Exa :-

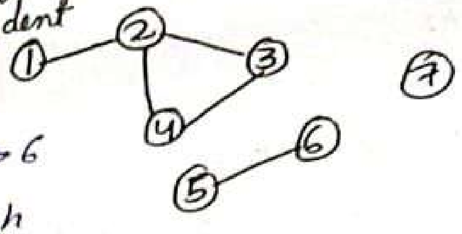


- DFS (1) :
  - → Mark 1 visited.
  - → Neighbours ⇒ {2, 3}
- DFS (2) :
  - → Mark 2 visited
  - → Neighbours → {1, 3, 4}
  - → Already visited : {1, 3}
  - → Continue with 4.
- DFS (3)
  - → Mark 3 visited
  - → Neighbours ⇒ {1, 2, 4} (all visited)
- DFS (4)
  - → Mark 4 visited
  - → Neighbours : {2, 3} (all visited)

All nodes visited.

* DFS solves reachability → from one node, find all nodes you can reach.

Q. In a classroom, there are N students and you are given M relations. For each student x, if you give gossip to that student how many people will it reach?



Ex: Friendships : $1 \leftrightarrow 2, 2 \leftrightarrow 3, 2 \leftrightarrow 4, 5 \leftrightarrow 6$

Gossip is given to student 1 → travels through 2, 3, 4

· Total reached = 4 students

Idea :- For each node
1) DFS (x)
2) Count no of nodes visited

We convert the given M relation to adjacency list.

$$\begin{array}{c} \#V \quad\quad \\ \begin{bmatrix} 7 & 5 \\ 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 2 & 4 \\ 5 & 6 \end{bmatrix} \end{array} \leftarrow \text{No of edges } (\#E) \quad \text{Edge list} \quad \Rightarrow$$

vector<vector<int>> g ;

Adjacency list

```
[
  0 : [ ]
  1 : [2]
  2 : [1 3 4]
  3 : [2 4]
  4 : [3 2]
  5 : [6]
  6 : [5]
  7 : [ ]
]
```

Time & Memory Complexity of Adjacency list:-

T.C → $O(N+2M)$ M.C → $O(N+2M)$

Implementation of Adjacency list :-

```
vector<vector<int>> g;
void solve ( ){
    int n, m;
    cin >> n >> m;
    g.resize(n+1);
    for(int i=0; i<m; i++){
        int x, y;
        cin >> x >> y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
}
```

```
using namespace std;

vector<vector<int>> g;
vector<int> visited;

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

```cpp
void solve(){
    int n,m;
    cin>>n>>m;
    g.resize(n+1);
    for(int i=0;i<m;i++){
        int x,y;
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }

    for(int i=1;i<=n;i++){
        visited.assign(n+1,0);
        dfs(i);
        int cnt=0;
        for(int node=1;node<=n;node++){
            if(visited[node])cnt++;
        }
        cout<<cnt<<endl;
    }
}
```

T.C per node :-

For every node → $O(1 + \#\text{ of neighbours})$

In Worst case, a node can reach every node

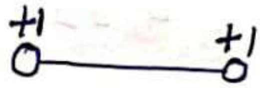
so, $\sum_{i=1}^{N} O(1 + \#\text{ of neighbours}) \Rightarrow \underbrace{\sum^{N} 1}_{N} + \underbrace{\sum^{N} \#\text{ of neighbours}}_{2M}$

Why 2M ?

For each pair in M, it is stored twice :

→ once in adjacency list of u.

→ once in adjacency list of v.



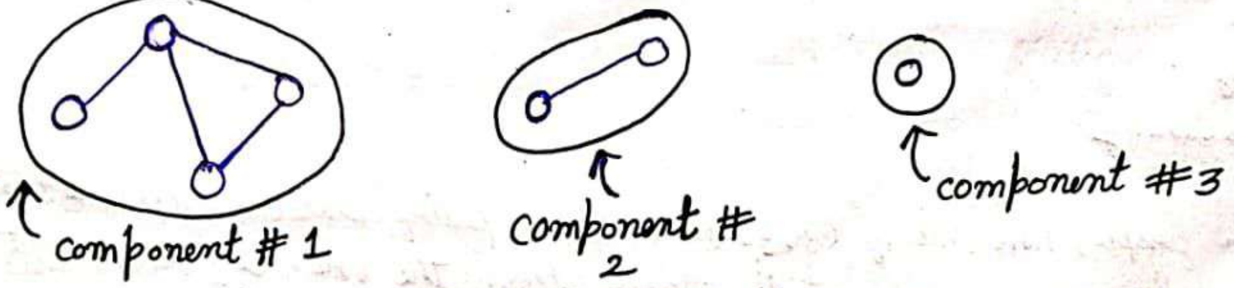
So every edge inc. the total no. of neighbours by 2. Hence, $2 * M$.

$\Rightarrow O(N+M)$

* If constraints are $N, M \leq 10^5$ → our solution will give us TLE

∴ It will need optimization

Connected Components :-



→ DFS can be used to find connected components.

→ Each connected components gets a unique no.

```
vector<vector<int>> g;

vector<int> visited;
vector<int> comp_no;

void dfs(int node,int cc){
    visited[node]=1;
    comp_no[node]=cc;
    for(auto v:g[node]){
        if(!visited[v]){
            dfs(v,cc);
        }
    }
}

void solve(){
    int n,m;
    cin>>n>>m;

    g.resize(n+1);
    visited.resize(n+1);
    comp_no.resize(n+1);

    for(int i=0;i<m;i++){
        int x,y;
        cin>>x>>y;
```

```
}

void solve(){
    int n,m;
    cin>>n>>m;

    g.resize(n+1);
    visited.resize(n+1);
    comp_no.resize(n+1);

    for(int i=0;i<m;i++){
        int x,y;
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }

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

```
vector<int> comp_size(cur_comp_no+1,0);
for(int i=1;i<=n;i++){
    comp_size[comp_no[i]]++;
}

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

DFS (in main func^n)

→ For each component, DFS is called once.

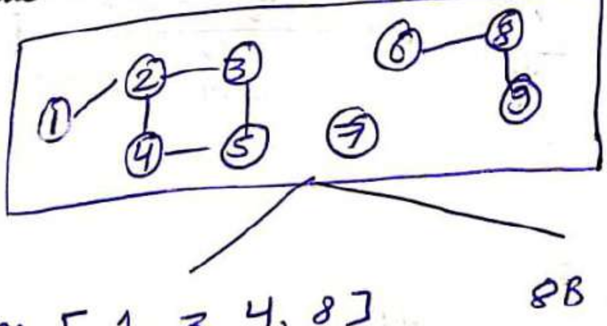
→ Example calls :

- DFS (1,1)
- DFS (3,2)
- DFS (7,3)

→ T·C per DFS : $O(V_{comp} + E_{comp})$

→ T·C ~~cox~~ across all components : $O(N+M)$

Q. Given a classroom with N students and M relations. There is too many gossip hence it is decided that this class will now be divided into 2 sections. Divide students in such a way there is no gossip ever.



Class 8

8A [1, 3, 4, 8]   8B [2, 5, 7, 8, 9]

No friendship among these groups

Classical Problem : Bi-partite coloring / Graph checking

```cpp
#include<bits/stdc++.h>
using namespace std;

vector<vector<int>> g;

vector<int> visited;
vector<int> section;

bool is_bipartite=1;

void dfs(int node,int color){
    section[node]=color;

    // DFS Part
    visited[node]=1;
    for(auto v:g[node]){
        if(!visited[v]){
            dfs(v,(1+2)-color);
        }
        else if(color[v]==color[node]){
            is_bipartite = 0;
        }
    }
}
```

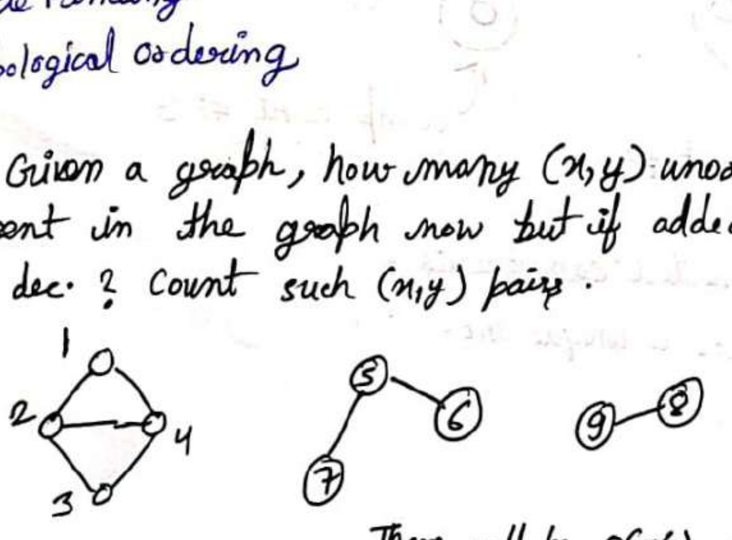
```cpp
for(int i=1;i<=n;i++){
    if(!visited[i]){
        dfs(i,1);
    }
}

if(is_bipartite){
    cout<<"YES\n";
    for(int i=1;i<=n;i++){
        cout<<section[i]<<" ";
    }
    cout<<endl;
}else{
    cout<<"NO\n";
}
```

Applications :-

1. Reachability
2. Component Numbering
3. Bi-partite
4. Cycle Finding
5. Topological ordering

H.W Q. Given a graph, how many (x, y) unordered pairs exist which are not present in the graph now but if added then the no. of components will dec. ? Count such (x,y) pairs.



In this exa, if you ~~add a~~ connect 1 to 5, the no. of components will dec.

There can be $O(n^2)$ pairs but given constraints are $N, M \leq 10^5$.