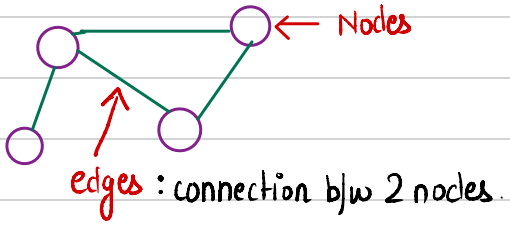
September 7, 2025

GRAPHS - DFS USAGE

Topic : GRAPHS
Week 1 → DFS | BFS
Week 2 → Shortest Path
Week 3 → Modelling | Union Find...
Week 4 → Trees

* For now, let us understand that a graph is made up of nodes & edges.



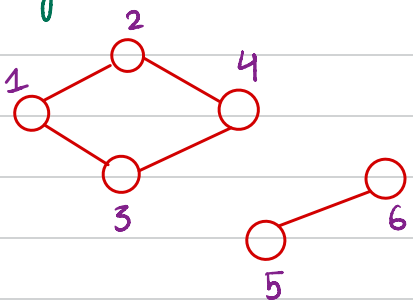
* DFS Algorithm

Visit neighbouring nodes and ask them to visit their nearby nodes recursively. (Recursive Algorithm)
When we call DFS on a particular node, it will give information of all the reachable nodes from the given node.

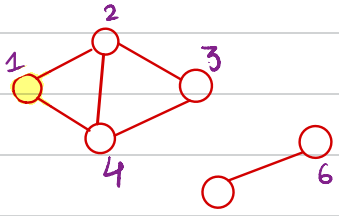
```
DFS(x){
    visited[x] = True
    for (v : neigbours[x]){
        if (!visited[v])
            DFS(v)
    }
}
```

↓ Call to Mark neighbouring nodes as visited.

Runthrough :

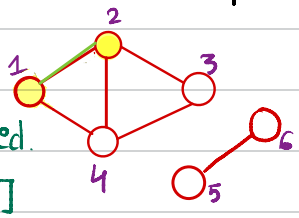


DFS(1):
Mark 1 as visited

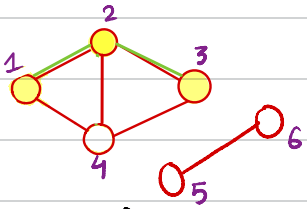


(NL)
Neighbours list of 1 : [2, 3]

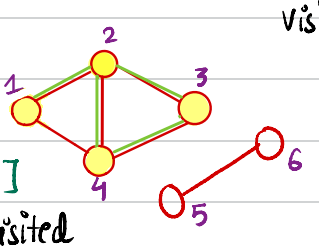
DFS(2):
→ Mark 2 as visited.
NL: → [1, 3, 4]
↑ Already visited
→ [1, 3, 4]
↑ Visit here



DFS(3):
→ Mark 3 as visited
→ NL of 3:
• [2, 4]
↑ Already visited, check next
• [2, 4]
↑ Visit here



DSF(4):
→ Mark 4
NL: [1, 2, 3]
All are visited



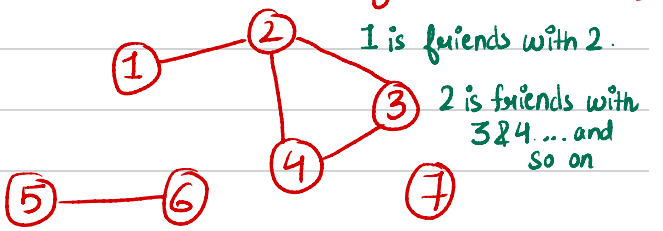
AlgoZenith

And then you go back up tracing other neighbours of parent call. But in this case it can be seen that all nodes are visited already so when going to parent call to eventually DFS(1), there won't be any other recursive call

DFS(1)
↳ [✱, ✱]
↳ DFS(2)
↳ [✱, ✱, ✱]
↳ DFS(3)
↳ [✱, ✱]
↳ DFS(4)
↳ (✱, ✱, ✱)

DFS simply solves reachability problem

Ques 1. In a classroom, there are N students and you are given M relations (who is friends with whom)



1 is friends with 2.
2 is friends with 3 & 4 ... and so on

If a gossip is given to 'x'th student, Find out how many students will it reach?

Ex: If you give gossip to 1 then 2 will know it. 2 will tell its friends 3 & 4. So, for (1) it will reach 4 students.

Idea:
For every node, we will do DFS and count how many nodes got visited.
1. DFS(x)
2. Count # of visited

Input: → 7 (N) 5 ← M

Also called vertices/ nodes

1	2
2	3
3	4
2	4
5	6

Edge list
M relations given

We convert the given M relation to Adjacency List (what nodes are neighbour of what)
→ vector < vector<int>> g;

Adjacency list:
0
1 : [2]
2 : [1, 3, 4]
3 : [2, 4]
4 : [3, 2]
5 : [6]
6 : [5]
7 : [ ] ← empty as no relation exist

Neighbours Array

So if we have
→ 1 2
Then we will just append 2 in 1's list And 1 in 2's list
Similarly, → 5 6
Append 5 in 6's list and append 6 in 5's list.

→ 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);//O(N)
    for(int i=0;i<m;i++){
        int x,y;
O(2M)   cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
    }
}
```

AlgoZenith

```cpp
vector<vector<int>> g;
vector<int> visited;


void dfs(int node){
    visited[node]=1;
    for(auto v:g[node]){
        if(!visited[v]){
            dfs(v);
        }
    }
}
void solve(){ || O(N(N+M))
    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++){ //O(N+M)
        visited.assign(n+1,0);→clear the visited array
O(N+M)→dfs(i); ||Fill visited array for node i.
        int cnt=0;
 O(N) →for(int node=1;node<=n;node++){
            if(visited[node])cnt++; ||count how many nodes reachable from i
        }
        cout<<cnt<<endl;
    }
}
```

→ Time Complexity : $O(N(N+M))$

|| DFS fxn : $O(N+M)$

Mark → O(1)

Go through neighbours array

So,

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

In worst case, a node can reach every node. $\sum_{i=1}^{N} O(1 + \#\text{of neighbours})$

Breaking the summation :

$\sum_{i=1}^{N} 1 + \sum_{i=1}^{N} \#\text{of neighbours}$

$\sum_{i=1}^{N} 1$ → N

$\sum_{i=1}^{N} \#\text{of neighbours}$ → 2M ↳ why?

Why 2M?

For each pair in M, when its added to the adjacency list, they contribute 2 times in each other's list.

So, every edge increases the total number of neighbours by 2. Hence, 2*M.

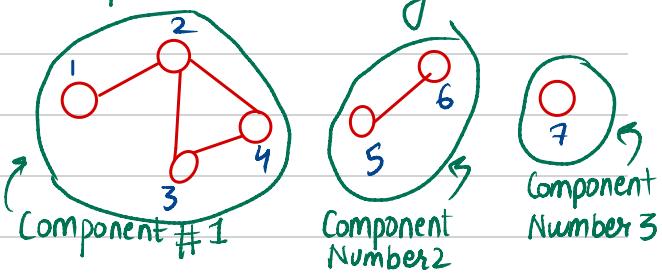
⇒ O(N+M)

* If the given constraints are

$N, M \leq 10^5$

Our current solution will give us TLE. Need Optimisation.

* Component Numbering



Mark the respective nodes according to their component numbers.

vector <int> comp_no;

```cpp
vector<int> visited;
vector<int> comp_no;

void dfs(int node,int cc){ → passing component no.
    visited[node]=1;
    comp_no[node]=cc;
    for(auto v:g[node]){
        if(!visited[v]){
            dfs(v,cc);
        }
    }
}
```

Inside main, Now the DFS call will be:

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

So, For our example, only 3 DFS calls will be made now:

DFS(1,1)
DFS(5,2)
DFS(7,3)

→ In DFS(x) = $O(V_{reachable} + E_{reachable})$

Here in DFS(x, cc), we are doing 1 DFS call per component.

So,

For Component #1(cc1) = $O(V_{cc1} + E_{cc1})$

For cc2 = $O(V_{cc2} + E_{cc2})$

⋮

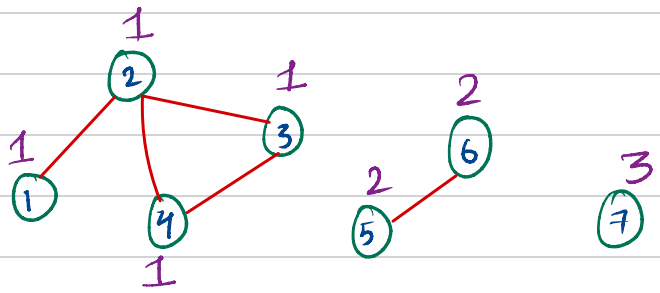
+ For ccX = $O(V_{ccX} + E_{ccX})$

$\Rightarrow O(N+M)$

Because the sizes of vertices of all the components is the total number of nodes (N) and Number of Edges in each component is disjoint, so if you add them, it will simply be the total number of edges (M)

New T.C $\Rightarrow O(N+M)$

* Applications of DFS



→ So, we have this array:

Component No. Array:

Component No. →	1	1	1	1	2	2	3
Node →	1	2	3	4	5	6	7

We can use this to build a frequency array which can help us in finding out how many nodes are in each component

Component size Array:

⇒

4	2	1
1	2	3

← frequency of nodes

← Component Number

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

New addition to prev code

```
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;
}
```

Test Case 1

Input

```
7 5
1 2
2 3
3 4
2 4
5 6
```

Output

```
4
4
4
4
2
2
1
```

AlgoZenith

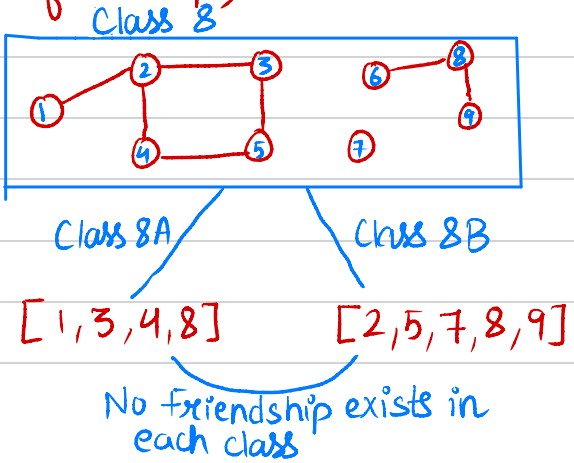
* Component Numbering on top of Graph is a very important Technique. Some other questions one can solve:

→ Given a gossip to 1, will it reach a node x (and you are given Q queries i.e x is given Q times)

Now, with component number array, we can solve each query in O(1)

```
if ( comp_no[1] == comp_no[x])
    then return True
     else return False
```

Ques 2. 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 that there is no gossip ever. (Break all friendships)



If the section can be divided, then for every node, Print the section it will go to.

Classical Problem: Bi-Partite Coloring / Graph checking

Solution:

vector <int> Section;

```
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(section[v] == section[node]){
            is_bipartite = 0;
        }
    }
}

    int n,m;
    cin>>n>>m;

    g.resize(n+1);
    visited.resize(n+1);
    section.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++){
        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";
}
```

Test Case 1

Input

```
7 5
1 2
2 3
3 4
2 4
5 6
```

Output

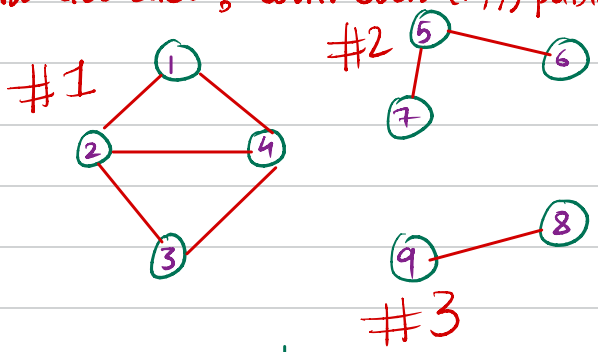
```
NO
```

* Important and all Possible Applications of DFS

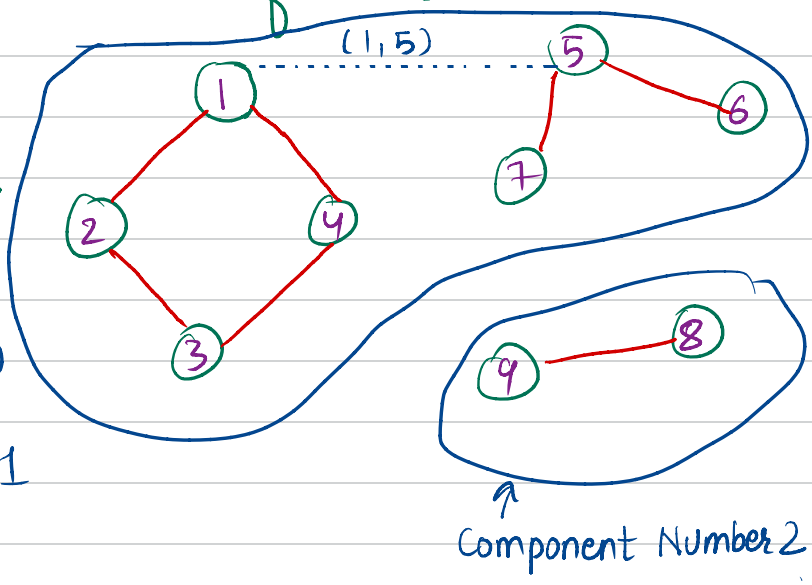
1. Reachability
2. Component Numbering
3. Bi-Partite
4. Cycle Finding
5. Topological Ordering

Homework Ques.

Given a graph, how many (x, y) unordered pairs exist which are not present in the graph now but if added then the number of components will decrease? Count such (x,y) pairs. (Total)



In this example, if you add (1, 5) → Connect these nodes
The number of components will decrease from 3 to 2.



Ideally there can be $O(N^2)$ such pairs but the given constraints are:

$N, M \leq 10^5$