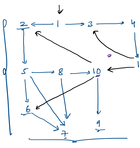
Graphs 3
Friday, 29 July 2021 8:09 PM



DFS: (Recursive Call)

Trees: Cycle

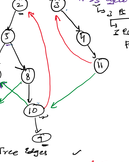
Edges: Nodes even if we go to already visited, we cannot directly say cycle. eg: 5 → 7, 11 → 10, 8 → 7

Edges: Nodes, if we have a edge, to a node v which is currently in stack. Now we can say there is a cycle

v has an edge to its parent node or encounter already there, cycle exists

```
bool DFS(int u, vector<int> g, int vis[], int st[]){
    if (vis[u]) return false;
    vis[u] = true;
    for(int v : g[u]){ // from u → v there is an edge
        // If v is st[] then cycle & v is another
        if(st[v]) { return true }
        else {
            st[v] = true;
            bool ch = DFS(v, g, vis, st);
            st[v] = false;
            if (ch) { return true }
        }
    }
    return false;
}
```

```
main() {
    // Define g, vis[], st[]
    for(int sre = 1; sre <= N; sre++){
        if(vis[sre]) continue;
        st[sre] = true;
        bool ch = DFS(sre, g, vis, st)
        if(ch) { return true }
        st[sre] = false
    }
}
```



vis: | ✓ | T | T | | T | | | |
1 2 3 4 5 6 7 8

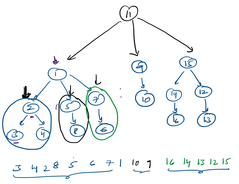
eg)

A → B // Before doing A we need to do B
{B depending on A} Do A & then B

// Before doing something its dependencies must be created.

C B D A
C D B A
D C B A

DFS + postorder



3 4 2 8 5 6 7 1 10 9 16 14 13 12 15 11

```
void DFS(int s, bool vis[], vector<int> g[], vector<int> topo){
    if(vis[s]) { return }
    vis[s] = true;
    // Before pushing s, Apply DFS on all children
    for(int v: g[s]){
        DFS(v, vis, g, topo)
    }
    topo.push_back(s);
}
```

```
main() {
    vis[N+1] = false;
    vector<int> TS;
    for(int i = 1; i <= N; i++){
        if(vis[i]){//continue}
        DFS(i, vis, g, topo)
    }
    // We have Topo Sort
    Note: Reverse entire topo
    reverse(topo.begin(), topo.end())
}
```

eg:

Topo: 3 2 1

Note:
In Directed graph if cycle is present then cannot have topo.
But we can not detect cycle

1) Before Applying Topo Sort
   1) Check if there is cycle or not

Final Note:
1) What exactly we are getting with
   postorder reverse it actually

eg:

Postorder: 3 4 2 8 5 6 7 1
Rev po: 1 7 6 5 8 2 4 3
Levels: 1 2 5 7 2 4 8 6
Preorder: 1 2 3 4 5 8 7 6

2 → 5 : 5 should be overtake before 2

Pg 458/17

Eg) Given a directed graph, print/count Number of strongly connected components

Eg1:

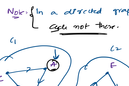
1 components | 1 components

(of which that component, all nodes can travel to all other nodes)

Eg2:

3 Components

Eg3:



4 components

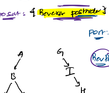
Note: In a directed graph of n nodes {Strongly Connected Component > n} cycle not there.

4 Strongly connected components

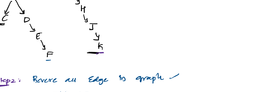
Topo sort: {Reverse postorder} Algo: (Kosaraju's Algorithm)

Post: C F E D B A K J H I G

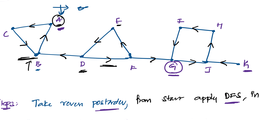
Rev po: G I H J K A B D E F C



Step 2: Reverse all edge to graph



Step 3: Take reverse postorder, from start apply DFS, in reversed graph



Rev po: G I H J K A B D E F C
Visited: ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓
Queue: J K A C B F

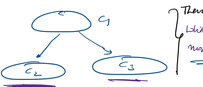
How many times: DFS: 4
4 Strongly Connected

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

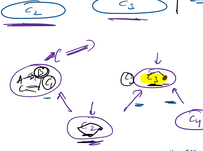
Why Kosaraju works

After reverse Edges

There will a node from {G1} which will can before all node to G2 in



There will a node from C1 which will can before all node to C2 & C3



→ There will be a node from C1 will will can before all node to C2
→ There will be a node from C3 will will can before all node to C2 & C4

C1 C3 C2 C4