GRAPHS

Day 2

Youtube links :

Part 1: https://youtu.be/YwJEJA9M3zg
Part 2: https://youtu.be/8teBctXfwoM

Contents:

1. DFS
2. Applications of DFS
3. Topological Sorting
4. SCCs (Kosaraju's algorithm)

Some discussion about expected time complexity

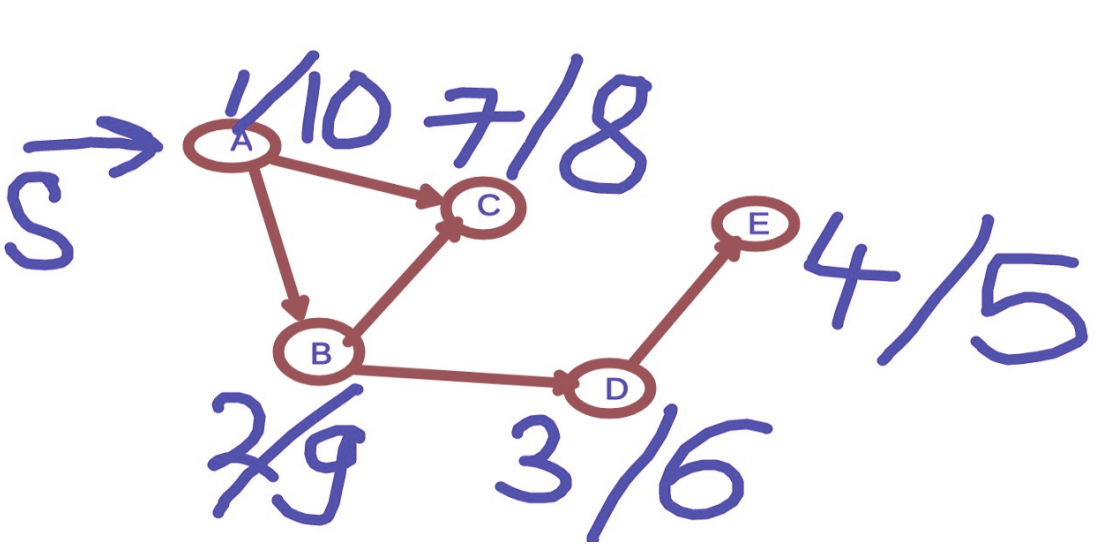
Generally, time limit is 1 sec
10^7-10^8 operations in 1 sec

If, N <= 10^5
O(N^2) code will get TLE
O(N log N) code will get accepted

If N<=20
O(2^N) code will get accepted

There are t <=10^5 testcases
Summation of N <=10^5 for all the testcases
(It is same as 1 testcase with N <= 10^5)
In graph questions, when no. of nodes , n <=10^5, use adjacency list (to avoid MLE)

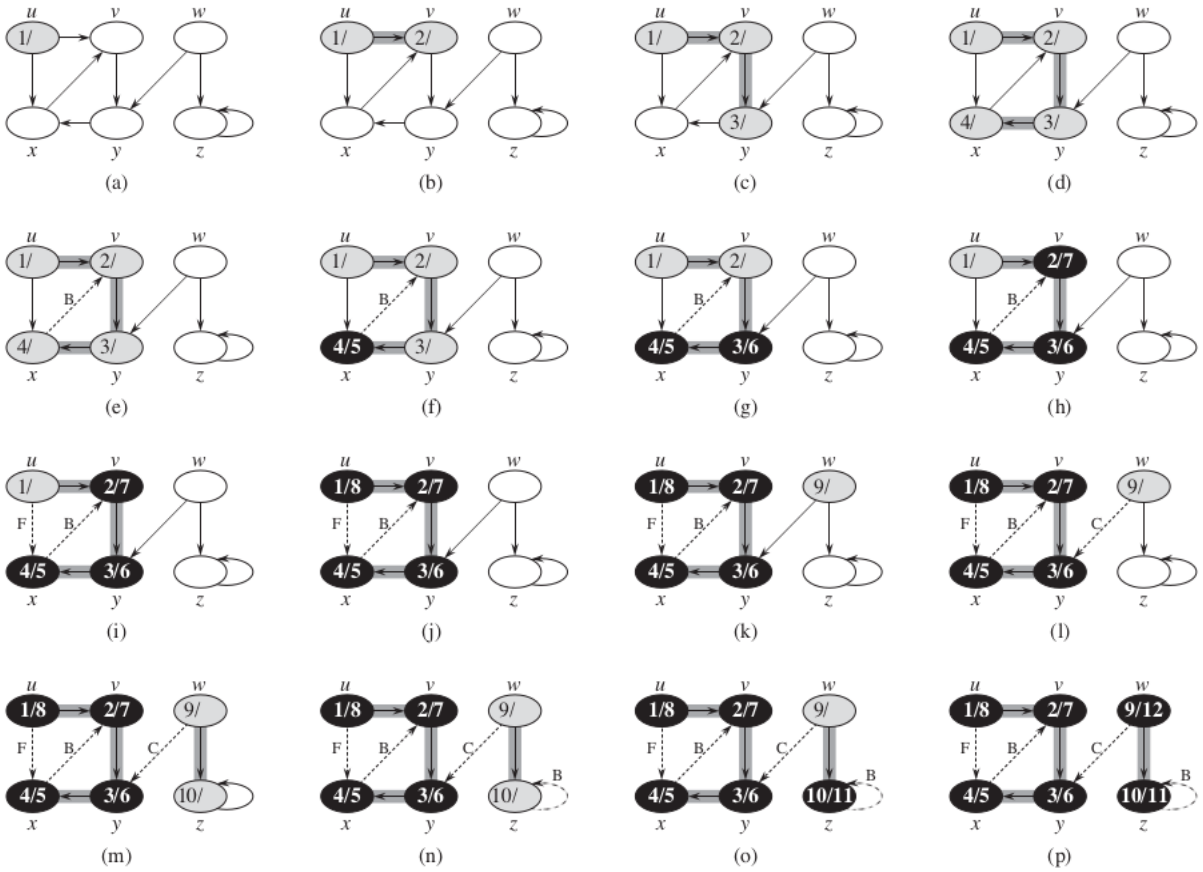
DFS (Depth first search)

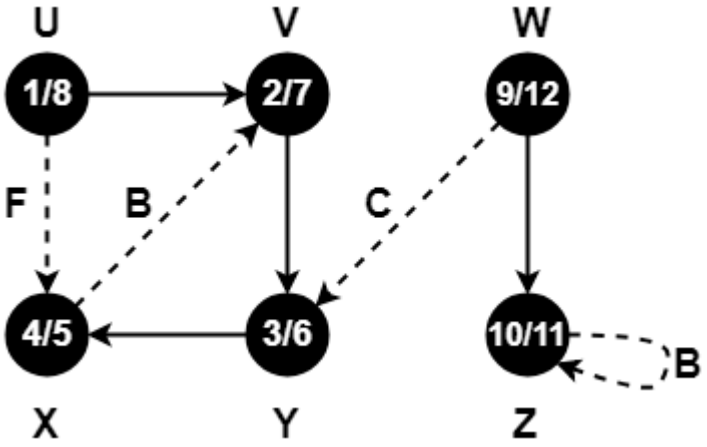


Visiting Time
Finishing Time

Colour codes in DFS

White: Unvisited node
Gray: Visited but not finished node
(Visited but all its neighbours are not visited)
Black: Finished node
(Visited and all its neighbours are also visited)
Try running DFS on below graph and calculating visiting and finishing time of each node:





Taking graph input as adjacency list

```
int n; // no. of nodes
vector<vector<int> > adj; // adjacency list
int main()
```

```
{
cin>>n;
int e; // no. of edges;
cin>>e;
adj.resize(n);
while (e--)
{
int u,v;
cin>>u>>v;
adj[u].push_back(v);
adj[v].push_back(u);  // for undirected graph
}
}
```

Code for normal DFS

```
int n; // no. of nodes
vector<vector<int> > adj; // adjacency list
vector<bool> vis; // initialise all values
with false

void dfs(int node)
{
vis[node]=true;
for(auto child: adj[node])
{
```

```
      if ( ! vis[child] )
     {
          dfs(child);
     }
}
}
```

Code for DFS when you need to calculate colour of each node at every instant

```
vector<int> colour;
// 0 - white (Use constants or #define for colours)
// 1 - gray
// 2 - black

void dfs(int node)
{
colour[node]=gray;
for(auto child: adj[node])
{
      if ( colour[child] ==0 )
     {
          dfs(child);
     }
}
colour[node]=black;
}
```

Code for DFS when you need to calculate visiting time and finishing time of nodes

```
int timerCode=1;
vector<int> visTime;
vector<int> finishTime;
void dfs(int node)
{
vis[node]=true;
visTime[node]=timerCode;
timerCode++;
for(auto child: adj[node])
{
      if ( ! vis[child] )
     {
          dfs(child);
     }
}
finishTime[node]=timerCode
timerCode++;
}
```

Calling DFS in main()

```
for(int i=0; i<n; i++)
{
  if (!vis[i])
   {
```

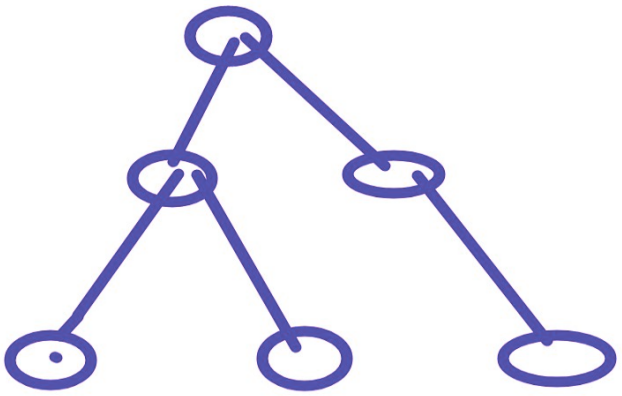
```
        dfs(i);
    }
}
```

Tree

Tree is an undirected connected graph without any cycle

Properties:

- If there are n nodes in a tree, then it will have n-1 edges



- If you are given a connected graph with n nodes and n-1 edges, then the given graph is a tree
- If you are given a connected graph with n nodes and n edges, then the given graph will contain exactly 1 cycle
- Each node has a single parent in tree

You have been given a tree of n nodes with root at 0. Find level of each node using DFS

```
vector<int> level;
// level[0] =0;
void dfs(int node)
{
vis[node]=true;
for(auto child: adj[node])
{
      if ( ! vis[child] )
     {
   level[child]=level[node]+1;
          dfs(child);
     }
}
}
// in main, call dfs(0); // Because 0 is
root of tree
```

DFS on trees can also be implemented as:

```
// call dfs(0, -1) in main()
void dfs(int node, int par)
{
for(auto child: adj[node])
{
      if (child != par)
     {
```

```
    level[child]=level[node]+1;
            dfs(child);
      }
}
}
```

Q. Checking whether given graph is bipartite graph

Link: https://cses.fi/problemset/task/1668

It can be done using DFS

```
vector<int> teamNum;
bool dfs(int node, int currTeam)
{
vis[node]=true;
teamNum[node]=currTeam;
for(auto child: adj[node])
{
    if ( teamNum[child] == teamNum[node])
          return false;
      if ( ! vis[child] )
      {
           bool temp=dfs(child, 3 - currTeam);
           if(temp==false)
                return false;
      }
}
return true;
}
```

```
int main()
{


for(int i=0; i<n; i++)
{
  if(!vis[i])
  {
   bool temp=dfs(i,1);
   if(!temp)
   {
    cout<<"IMPOSSIBLE";
    return 0;
   }
  }
}
// Print the team values if division is possible
}
```

Detecting cycle in a graph

no of nodes=n

```
int colour[n];
```

Return true if cycle present else false.

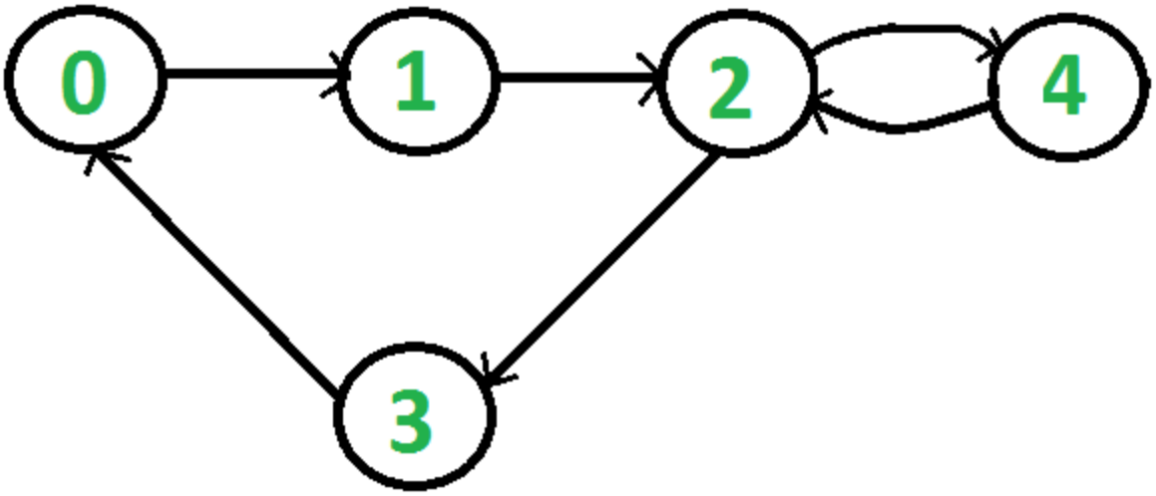
```
bool dfs(int node, int parent)
{
colour[node]=1;
for(auto child: adj[node])
{
      if ( child!=parent && colour[child] == 1 )
     {
    // You can remove the condition for
child!=parent, when you want to detect even the
2 node cycles in directed graph like 1->2, 2->1
           return true;
     }
else if( child!=parent && colour[child] == 0 )
     {
           bool temp = dfs(child, node);
           if( temp == true) return true;
     }
}
colour[node]=2;
return false;
}
```

```
int main()
{
    // Take graph input
```

```
// .....
   for(int i=0;i<n;i++)
      colour[i]=0;
   // Mark colour of all nodes as WHITE
(unvisited)

  for(int i=0; i<n; i++)
{
   if(colour[i] == 0)
   {
      bool cycle = dfs(i,-1);
      if(cycle)
      {
          cout<<"Cycle Present";
          return 0;
      }
   }
}
cout<<"No cycle found";
return 0;
}
```

You can try running this code over below example:

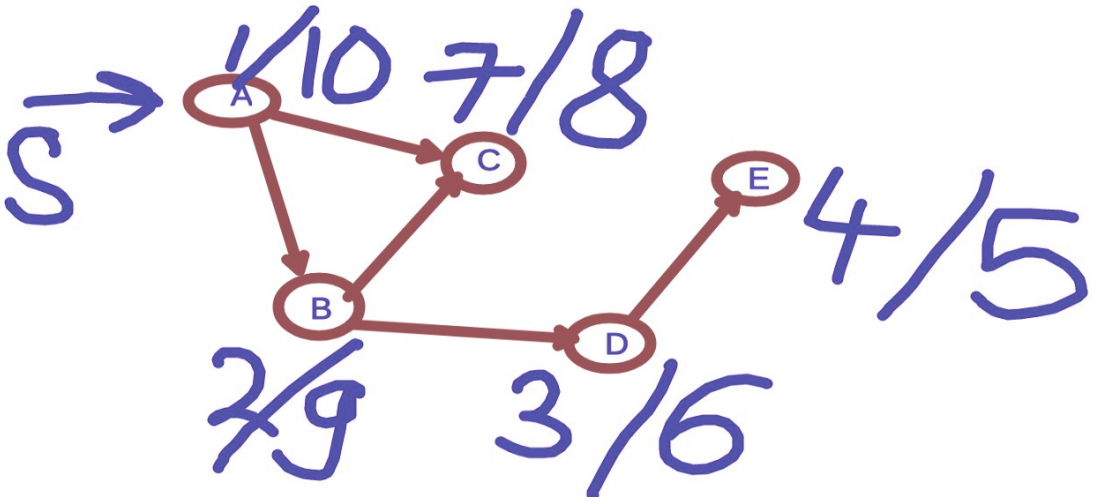


*dfs(0)
col[0]=1
0->child==1
**dfs(1,0(parent))
col[1]=1
***dfs(2)
col[2]=1
****dfs(3)
col[3]==0
col[3]=1
3->
child=0
color of child==1-->cycle
col[0]==1
return true;

****dfs(4)

col[3]=2
col[2]=2

col[1]=2
col[0]=2

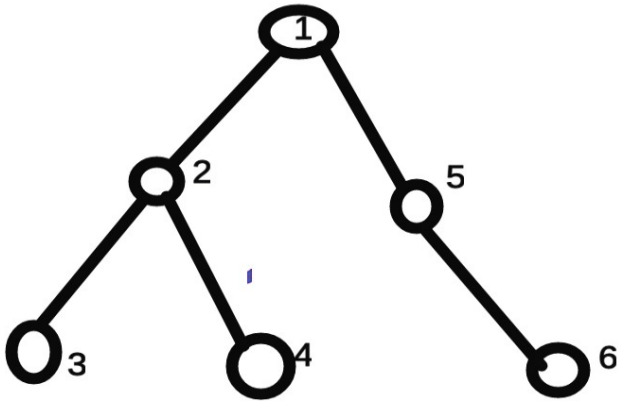


dfs(A)
col[A]=1
A->child:B,C
*dfs(B)
child:C,D
**dfs(C)
col[C]=1;
child C-NULL
col[C]=2
false ret

**dfs(D)
col[D]=1
D->CHILD:E
***dfs(E)
col[E]=1
col [E]=2

ret false;
col[D]=2
ret false

*dfs(C)



Given a tree, you need to find the gcd of all the values of nodes in subtree of every node. 0 is root node

Example, in above graph, ans[5] = gcd(5,6)
ans[2]= gcd(2,3,4)

No. of nodes, n <= 10^5

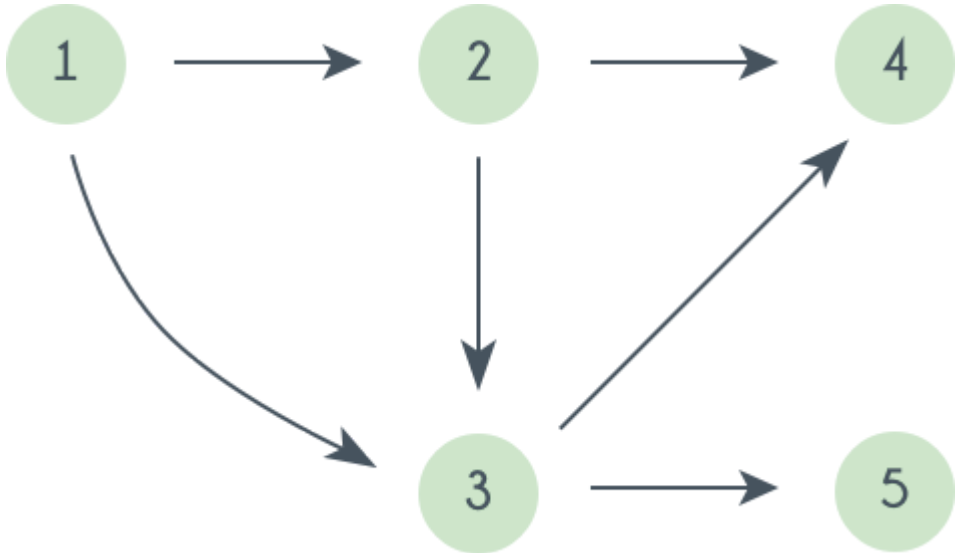
Try to solve this problem using DFS

vector<int> ans(n);

```
// call dfs(0, -1) in main()
void dfs(int node, int par)
```

```
{
ans[node]=val[node];
for(auto child: adj[node])
{
      if (child != par)
     {
          dfs(child);
          ans[node] = __gcd(ans[node], ans[child]);
     }
}
}
```

Topological sort:



DAG -> Directed Acyclic graph

(U,v) u->v - u comes before v
1 before 3

1 before 2
2 before 4
2 before 3
3 before 4
3 before 5

(v1,v2,v3….,vn,v1)
V1 before v2
V2 before v3
Vi before v(i+1) 1<=i<n
Vn before v1
In_deg = {0,1,2,2,1}
T : {1}
In_deg = {-1,0,1,2,1}
T: {1,2}
In_deg={-1,-1,0,1,1}
T: {1,2,3}
In_deg = {-1,-1,-1,0,0}
T: {1,2,3,4,5} , {1,2,3,5,4}

Algo:

```
int n,m;
vector<int> adj[n];
```

```cpp
int in_deg[n]={0};

for(int i=0;i<m;i++){
    int x,y;
    cin>>x>>y;
    adj[x].push_back(y);
    in_deg[y]++;
}
queue<int> q;
for(int i=0;i<n;i++){
    if(in_deg[i]==0){
        q.push(i);
    }
}
vector<int> topo_sort;
while(!q.empty()){
    int x  = q.front();
    q.pop();
    topo_sort.push_back(x);
    for(int i=0;i<adj[x].size();i++){
        int y = adj[x][i];
        in_deg[y]--;
        if(in_deg[y]==0){
```

```
            q.push(y);
        }
    }
}
```

Method 2 of topological sort

Topological sort is order of nodes in <u>decreasing order of finishing times</u>

```
int n; // no. of nodes
vector<vector<int> > adj; // adjacency list
vector<bool> vis;
 // initialise all values with false

stack<int> st;
void dfs(int node)
```

```
{
vis[node]=true;
for(auto child: adj[node])
{
      if ( ! vis[child] )
     {
          dfs(child);
     }
}
st.push(node);
// push to stack when finished
}

int main()
{
   while(!st.empty())
 {
     cout<<st.top()<<'\n';
     st.pop();
 }
}
```

Topological sort is also used for DP on graph problems.

Try these 2 problems:

1. https://www.hackerearth.com/practice/algorithms/graphs/topological-sort/practice-problems/algorithm/lonelyisland-49054110/

2. https://cses.fi/problemset/task/1679/

Warning: As told earlier, Graph Algorithms and DP are one of the most important topics, from the point of internship preparation. And you are only a few months away from your internship tests. Most of you are not confident in DP also. Do you want to leave graphs like this, as well ? You all know that time that goes, never comes back.

Still, It has been observed in both CR-1 and CR-2 that nowadays, only a few students are participating in the classes. So, we have decided that the upcoming class would be the deciding factor for

whether we should continue conducting classes or not.
If this response is continued, the next CodeISM class may be the last class of this session. We'll just share the resources and you can continue on your own, if you wish that way.

Some more practice problems

1. https://www.codechef.com/problems/CHFNFRN
2. https://cses.fi/problemset/task/1666
3. https://codingcompetitions.withgoogle.com/kickstart/round/0000000000019ff43/00000000000003379bb
4. https://cses.fi/problemset/task/1669
5. https://cses.fi/problemset/task/1678/