GRAPHS

Day 5

(SCC, Diameter of a tree, Binary Lifting)

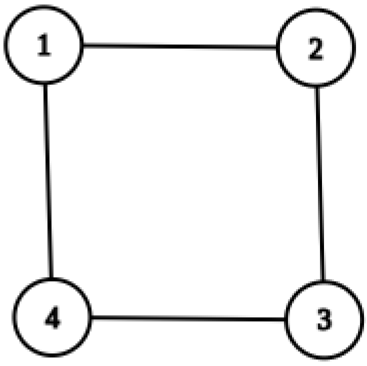
Youtube link :

https://youtu.be/aH_olfWOFLk

Contents:

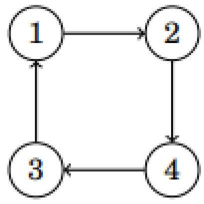
SCC (Strongly Connected Components)

Connected Graph (For undirected graph): An undirected graph such that ∃ a path between every pair of vertices. **Eg.** Graph A given below.



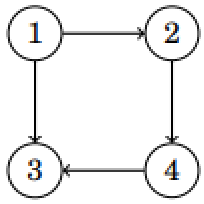
Graph A

SCC (Strongly Connected Component): A subset of vertices in a directed graph, such that ∃ a path between every pair of vertices. **Eg.** Graph B



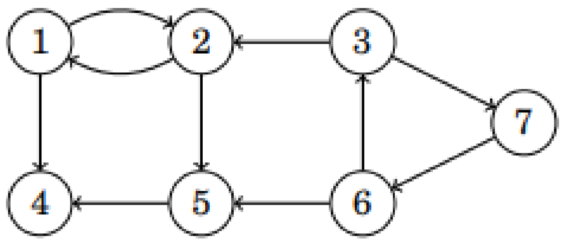
Graph B

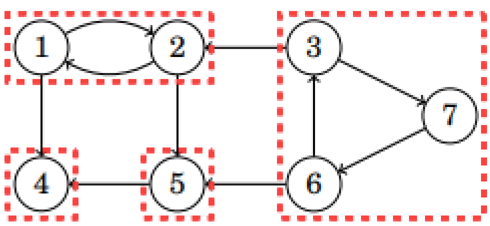
Q. Is Graph C given below an SCC ?



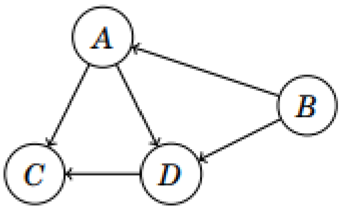
Graph C

There is no path from 4 to 1, so Graph C is not a SCC





If we replace each of the SCCs of any directed graph by a single node, we get a SCC-condensed graph like the graph below:



This SCC-condensed graph will always be a DAG (Directed Acyclic Graph).

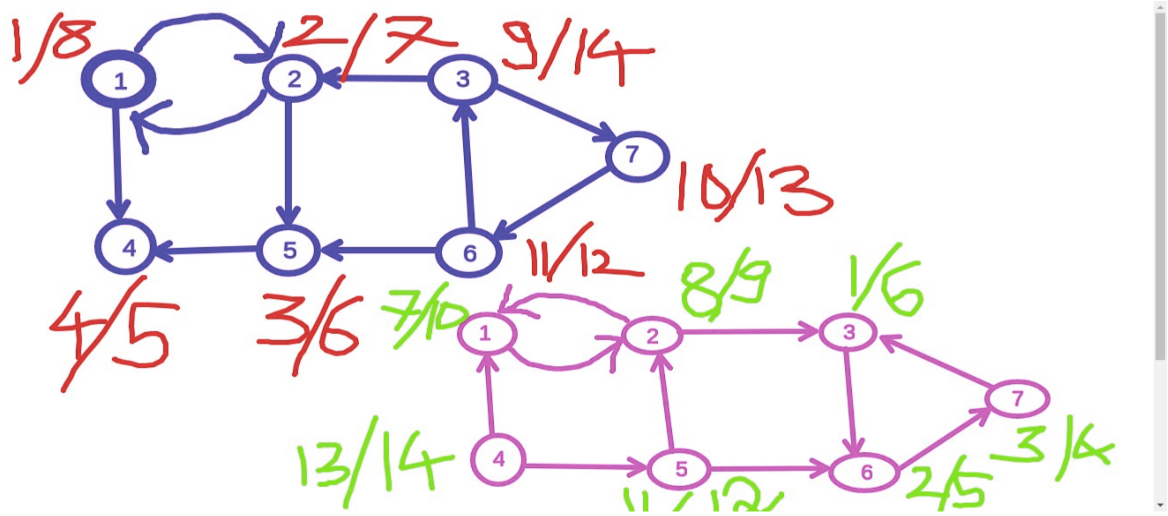
Some properties of SCCs:

- For any directed graph, after reversing all the edges, the SCCs remain the same.
- Source node of original graph becomes sink node in the reversed graph and vice-verca.
- If we start DFS from the source of original graph (or sink of the new reversed graph), every node of the sink SCC would be traversed.

Kosaraju's Algorithm for finding SCCs

1. Run dfs on given graph G and find the nodes in decreasing order of finish time (Topological Sorting).
2. Create a new graph G^T, which has all the edges reversed from the original graph.
3. Start DFS in this new graph G^T, in the topological order of G. Each node visited by dfs() call of a node, belong to the same SCC component.

Example:



SCCs are:

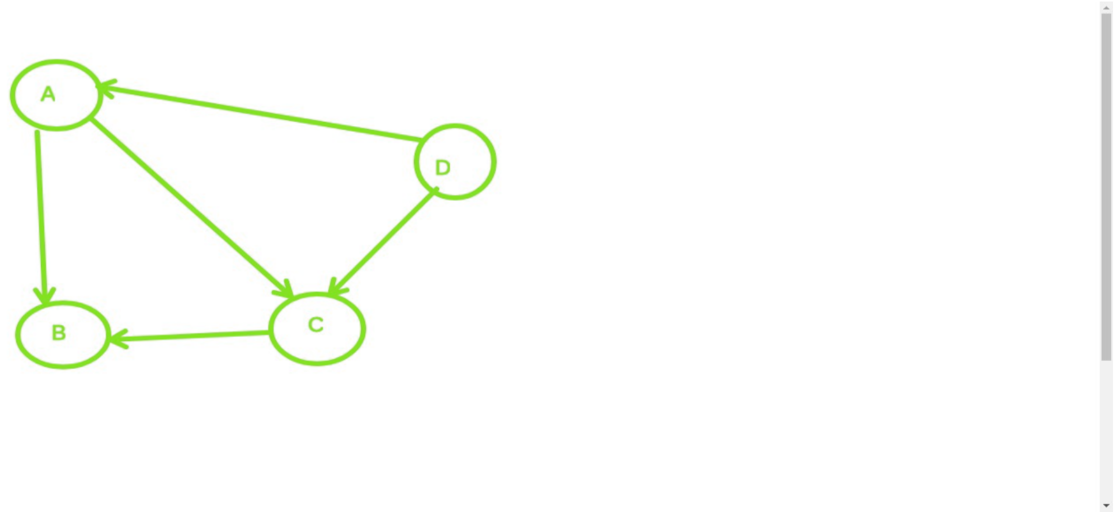
{ 3, 6, 7} - D

{ 1, 2} - A

{4} - B

{5} - C

Now, the SCC condensed graph (also a DAG) is:



Note: Using this SCC-technique, we are able to convert any directed graph into a DAG.

Code for finding all SCCs of a directed graph:

```
vector<bool> vis;
vector<vector<int> > g, gr;
stack<int> st;
vector<int> component;
vector<vector<int> > sccs;

void dfs1(int i)
{
   vis[i]=true;
    for(auto it: g[i])
    {
       if(!vis[it])
       {
          dfs1(it);
       }
    }
   st.push(i);
}

void dfs2(int i)
{
vis[i]=true;
for(auto it: gr[i])
{
    if(!vis[it])
   {
       dfs2(it);
   }
```

```
}
component.push_back(i);
}

int main()
{
int n, m;
cin>>n >> m;
g.resize(n);
gr.resize(n);
for(int i=0; i<m; i++)
{
   int u,v;
   cin>>u>>v;
   u--; // to make u and v on 0-based indexing
   v--;
   g[u].push_back(v);
   gr[v].push_back(u);
}
vis.assign(n,false);
for(int i=0; i<n; i++)
{
    if(!vis[i])
    {
       dfs1(i);
    }
}
vis.assign(n,false);
```

```
while(!st.empty())
{
int t=st.top();
st.pop();
if(vis[t])
     continue;
component.clear();
dfs2(t); // Run DFS in reverse graph
         // in topological order of original graph
sccs.push_back(component);
}
// You can also further convert graph
// to SCC-condensed graph (DAG)
// See practice problem 2 for my sample code
return 0;
}
```

Practice Problems:

1. https://cses.fi/problemset/task/1683

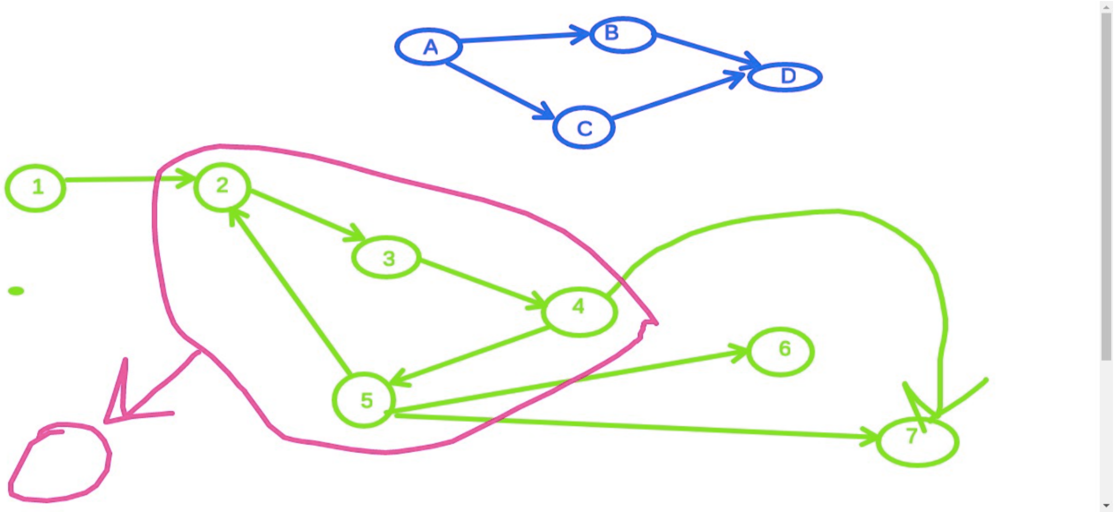
2. https://www.codechef.com/problems/MCO16405

Solution:

Consider this problem for a DAG:

dp[i] = Maximum number of people you can visit if you start from node i

dp[A] = max(dp[B], dp[C]) + people in city A



// First find all the SCCs
// Create a new SCC-condensed graph in which C[i] value of a node is sum of C[i] of all nodes in that SCC, which will always be a DAG
// Now, Apply DP on this DAG

```
long long dp[MAX] ;
for(int i=0; i<n; i++)
{
   dp[i]=0;
}
for(auto u: rev(topo))
{
   for(auto v: new_adj[u])
   {
         dp[u] = max(dp[u], dp[v]);
   }
   dp[u] = dp[u] + C[u];
}
```

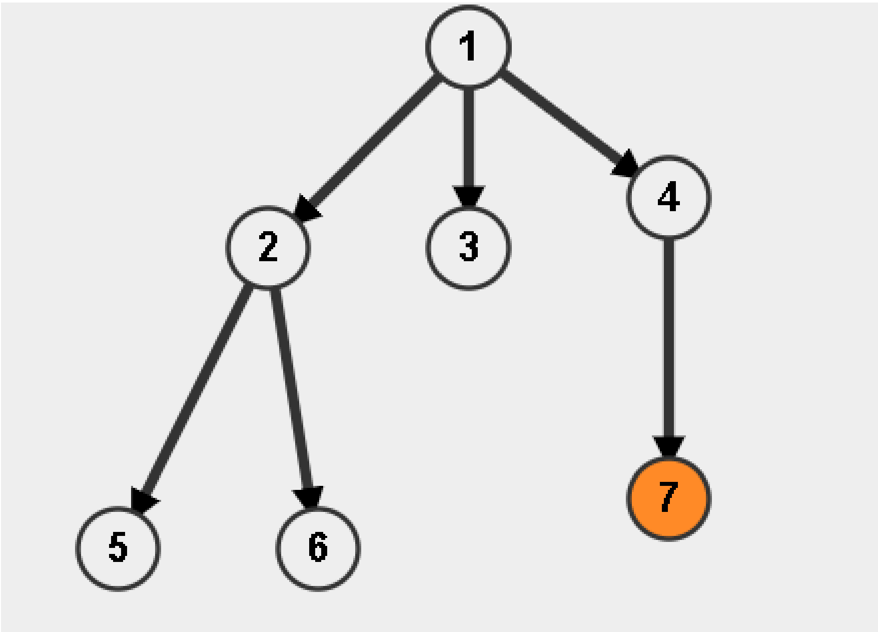
// Try to implement the code for this problem yourself and if you don't get it, you can look at my submission: https://www.codechef.com/viewsolution/40482695

3. https://www.spoj.com/problems/CAPCITY/
[**Hint**: Think of what would be your answer if given graph was a DAG?]
Solution:
  - First find SCCs, and print the SCC of the sink node (the node which was last in the topological order)

A general tip

Whenever you find a problem that involves a directed graph and you can't solve it by simple BFS/DFS or shortest path algorithms, rethink that problem assuming given graph as a DAG and if you can find a solution to it in that way. Then, you can use SCCs to convert the given directed graph to DAG and then apply your solution to it.

Tree Diameter

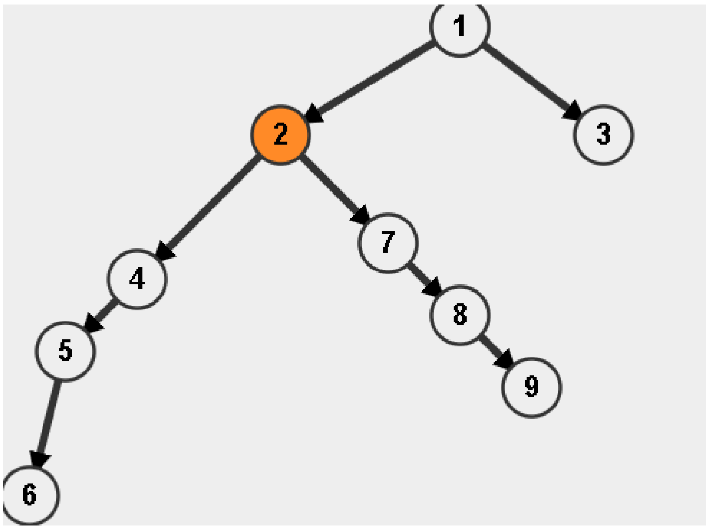


Diameter: Max no of edges present b/w two nodes, in a tree.
More than one Diameter can exist in a tree.
Diameters are not unique always.

In the case of trees, diameter is always formed by leaf nodes.

Diameter Calculation

Method-1: Depth calculation using DFS



v->adj list..

n-->no of nodes..

```
int depth[n],ans[n];

void dfs(int s,int par)
{
  int m1=-1,m2=-1;
 // store top 2 max depths among x childs..
  for(int i=0;i<v[s].size();i++)
  {
   int ch=v[s][i];
   if(ch!=par)
   {
     dfs(ch,s);
```

```
     if(depth[ch]>=m1)
     {
      m2=m1;
      m1=depth[ch];
     }
  else if(depth[ch]>m2)
   m2=depth[ch];
  }

}
//m1,m2-->max values store..
//m1>=m2
depth[s]=m1+1;
//m2=-1
ans[s]=m1+m2+2;
//m1+1
}

// res = max(ans[s])
cout << res; // Diameter
```

Method 2: Run DFS 2 times

1. Assume any **node a** as **root**
2. Start dfs from a and find that **node b**, **having max dist from a**

[Using depth[child]=depth[node]+1]

(This node b will be an endpoint of a diameter)

3. Now, Start dfs from node b and find the **node c, which is at max distance (d) from b.**

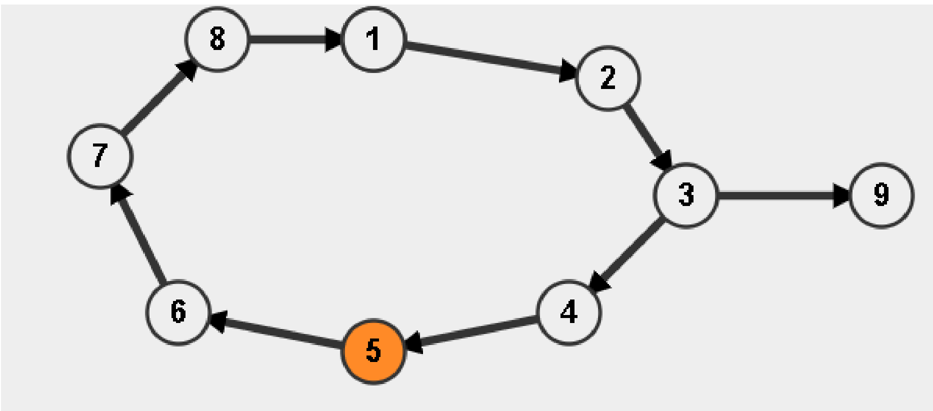
This value of d is the diameter of the tree.

Time complexity of both methods: O(n), where n = no . of nodes

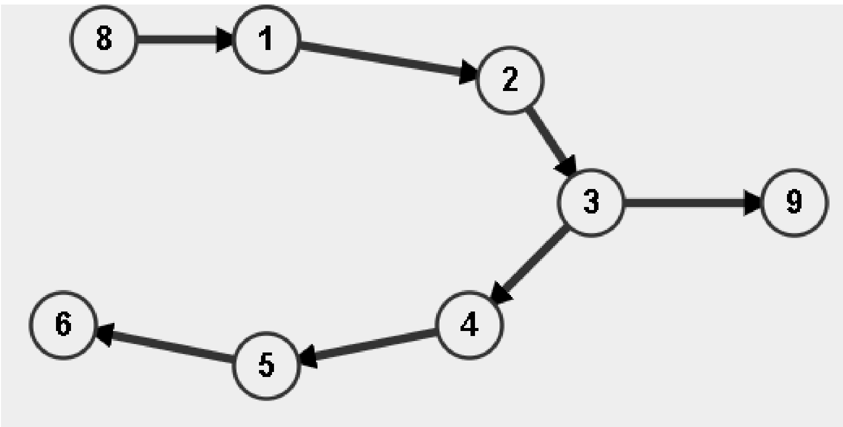
(No of edges in tree=n-1)

Practice Problem for finding diameter:
https://cses.fi/problemset/task/1131

The diameter found using the above methods, is only valid in case of a tree.



Diameter using the above method == 4 → wrong
Actual==5(7-->9)

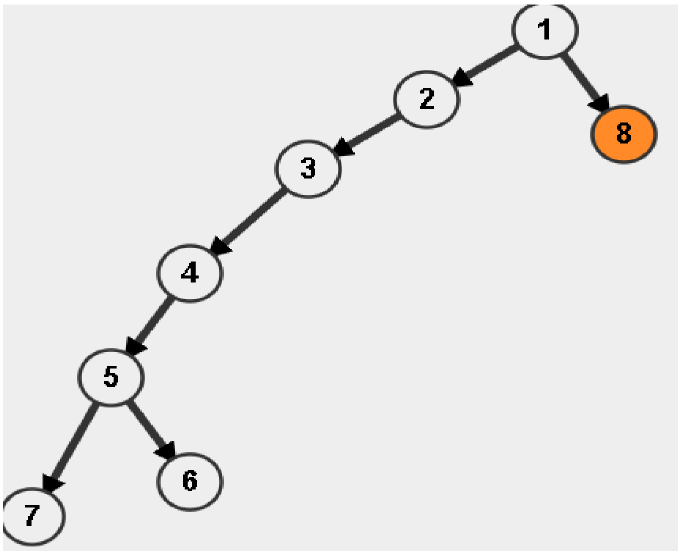


Binary Lifting

Finding k^th parent

Q. Given a tree and a node x, find k^th parent.

Assume root as 1.



par[7][2]==2 ?

2^j-->2^j-1→2^(j-1)

par[7][2]=par[par[7][1]][1](par[4][1])

(2^2)th parent of 7th node==(2^1)th parent of 4th node

(2^j)-->jumps req..half (2^j)/2 ju

mp
k==5..101
2->powers..parent store..
7-->2^(0)==5
7-->2^(1)==4
7-->2^(2)==2
k==5th parent of 7
int ans=7
5--0>101..ans=5
5-->2^2-->1

x=9
2nd parent..
int p=9
if(k>n)
 return root node..
for(int i=0;i<k;i++)
 p=par[p];
p=3..ans…

n->nodes..
Time complexity of above method?
O(n)/query

a^(n) ??
int res=1;
n times loop
 res=res*a;

O(n-->exponent)
n-->binary representation
n=5 (101 in binary)
So, n = 5 = 2^2 + 2^0

- Every distance can be divided into powers of 2 - [with at max log2(n) terms]
Eg. 11 = 2^3 + 2^1 + 2^0
(from binary representation of 11)
10 = 2^3 + 2^1
(from binary representation of 10)

- Using this, we can answer every query in **log(n) time**, if we **precompute the answer of all the powers of 2**

Note: Log(n) values is always <= 30 in general problems, when $n<=10^9$

O(logn)
x..log(x) order-->bits..
8-->1000
log(x)+1

```
int x=log(n)+1; //max possible jump req to reach
// a parent (You can also take x=30)
```

```
vector<vector<int> > v; // adjacency list of tree
int par[n][x];
```

// par[i][j] = 2^(j)th parent of ith node

store 2^1,2^2 parents of node s

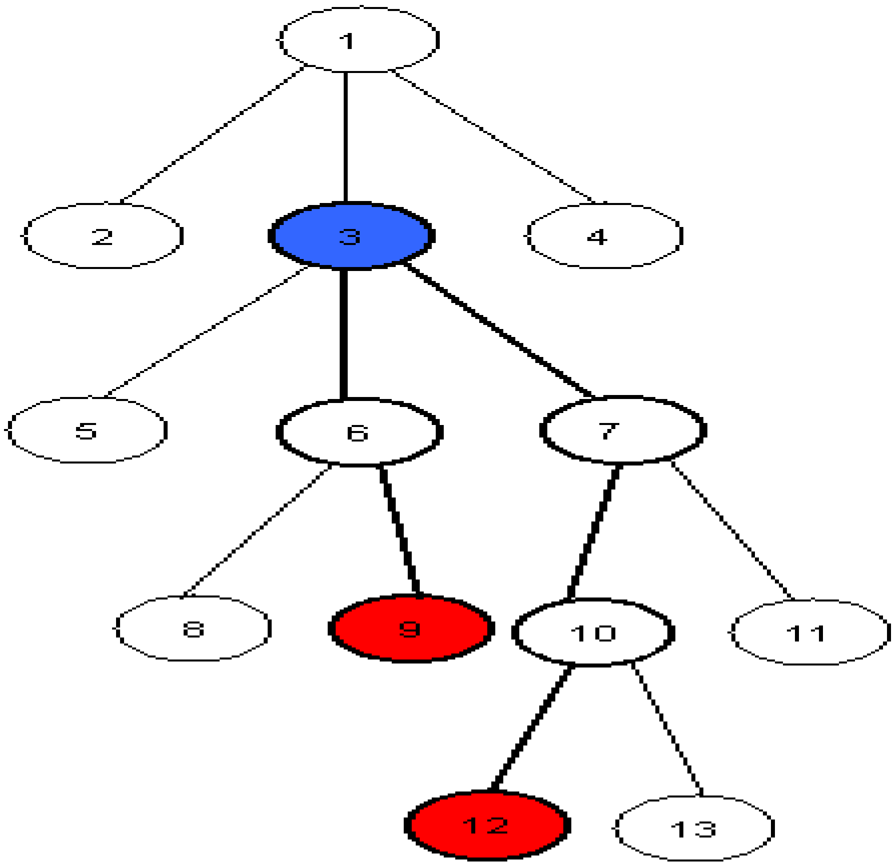
```
void dfs(int s,int p)
{
// s-->source node
// p-->parent of s..
 par[s][0]=p;
 for(int j=1;j<=x;j++)
  par[s][j]=par[par[s][j-1]][j-1];
for(int i=0;i<v[s].size();i++)
{
   int ch=v[s][i];
   if(ch!=p)
    dfs(ch,s)
}
}
```

k-->jump-->binary representation

```
int find_kth(int s,int k)
{
 for(int j=x;j>=0;j--)
 {
   if((1<<j)&k)//jth bit set or not in k..
{
```

```
  s=par[s][j];//jump of 2^j
  k-=(1<<j);
}
return s;
}
```

Time complexity: O(log n) per query



Practice Problem on binary lifting:
https://cses.fi/problemset/task/1687

Try the problem by yourself and if you get stuck, you can check our submission:
https://cses.fi/paste/488fefbbe9ece7ce179075/

Some more practice problems:

1. https://cses.fi/problemset/task/1686
2. https://www.spoj.com/problems/BREAK/

Link to my submission for SPOJ BREAK: (in case, you are stuck)

https://csacademy.com/code/ehtKr5CJ/