Lab9 Solutions

Is tree?

Tree คือ Graph ที่ไม่มี cycle และเชื่อมกันทุก vertex ดังนั้นวิธีการที่จะใช้ในการตรวจสอบคือ

- Graph นั้นจะต้องมี edge จำนวน n-1 เส้นเท่านั้น
- ถ้าหาก Depth First Search(DFS) ที่ vertex ใด ๆ ในกราฟ จะต้องเข้าถึงครบทุก vertex

ปล. โค้ดตัวอย่างมีการใช้ accumulate() เป็นการหาผลรวมของข้อมูลใน vector เพื่อหาว่า visit ครบทุก vertex ไหม (ถ้า visit แล้ว visited จะมีค่าเป็น true หรือ 1 ถ้าไปครบทุก vertex ผลรวมจะต้องเป็น n)

Example Code

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<vector<int>> graph;
vector<bool> visited;

void dfs(int u=0)
{
    if(visited[u]) return;
    visited[u] = true;
    for(int i=0;i<graph[u].size();i++) dfs(graph[u][i]);
}

int main()
{
    int n,m,u,v;
    cin >> n >> m;
    graph.resize(n);
    visited.resize(n,false);
    for(int i=0;i<m;i++)
{
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    dfs();
    if(m==n-1 && accumulate(visited.begin(),visited.end(),0)==n) cout << "YES";
    else cout << "NO";
}
```

$time\ complexity : O(m)$

Family

ลูกหลานของลูกก็ถือเป็นลูกหลานของเราเหมือนกัน หลักการของการหาลูกหลานในข้อนี้คือ

$$\textbf{จำนวนลูกหลานของตัวเอง} = \textbf{จำนวนลูกของตัวเอง} + \textbf{จำนวนลูกหลานของลูกทั้งหมด}$$

$$\textbf{จำนวนลูกหลานของตัวเอง} = \sum_{i} \mathbf{1} + \textbf{จำนวนลูกหลานของลูกคนที่ } \boldsymbol{i}$$

สังเกตว่าความสัมพันธ์นี้มีลักษณะแบบ *recursion* ดังนั้นจะสามารถเขียน *recursive function* เพื่อหาจำนวนลูกหลานของทุกคนในตระกูลได้

ข้อควรระวัง: เพื่อให้ไม่มีการหาลูกหลานซ้ำซ้อนหลายรอบ ควรเก็บจำนวนของลูกหลานที่เคยคิดไว้แล้วใน *array* หรือ *vector* จะได้นำข้อมูลมาใช้ได้เลย ไม่ต้อง *call function* เพื่อคิดใหม่

$Example\ Code$

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

vector<int> ans;
int count(vector<int> graph[],int n = 0)
{
    int children = 0;
    for(int i=0;i<graph[n].size();i++)
        children += 1 + count(graph,graph[n][i]);
    return ans[n] = children;
}

int main()
{
    int n,par;
    cin >> n;
    vector<int> graph[n];
    ans.resize(n);

    for(int i=1;i<n;i++) cin >> par, graph[par-1].push_back(i);
    count(graph);
    for(int i=0;i<n;i++) cout << ans[i] << ' ';
}
```

$time\ complexity : O(n)$

King Scroll

ใช้โครงสร้าง tree ที่มี 3 child node แต่ละ node เก็บข้อมูลลำดับ เพศ และสถานะการมีชีวิต ถ้าหากกษัตริย์องค์ปัจจุบันสวรรคต ให้ทำ tree traversal แบบ pre-order เริ่มจาก จอร์จที่ 1(root) และเลือกจอร์จที่เป็นเพศชายและยังมีชีวิตอยู่องค์แรกที่พบเป็นกษัตริย์องค์ต่อไป

ข้อควรระวัง: ไม่ควรสร้างแผนผังเครือญาติทั้งหมดก่อน แล้วพิจารณาการสวรรคตทีหลัง เพราะการเลือกกษัตริย์องค์ต่อไปจะต้องพิจารณาจากแผนผังเครือญาติในขณะที่กษัตริย์องค์ปัจจุบันสวรรคต อาจจะไปเลือกจอร์จที่ยังไม่ประสูติมาเป็นกษัตริย์องค์ต่อไปได้

Example Code

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

vector<int> kings = {1};
int found=0;

typedef struct _node{
    int george;
    char gender, alive;
    struct _node *left, *mid, *right;
} Node;

Node* newNode(int george,char gender)
{
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->george = george;
    temp->gender = gender;
    temp->alive = 'L' ;
    temp->left = NULL, temp->right = NULL, temp->mid = NULL;
    return temp;
}
```

```
Node* root = newNode(1,'M');

Node* searchGeorge(int george,Node* node = root)
{
    if(node == NULL) return NULL;
    if(node->george==george) return node;

    Node* temp = NULL;
    if((temp = searchGeorge(george,node->left)) != NULL) return temp;
    if((temp = searchGeorge(george,node->mid)) != NULL) return temp;
    if((temp = searchGeorge(george,node->right)) != NULL) return temp;
    return NULL;
}

void insertGeorge(int parent,int george,char gender)
{
    Node* insertFrom = searchGeorge(parent);

    if(insertFrom->left==NULL) insertFrom->left = newNode(george,gender);
    else if(insertFrom->mid==NULL) insertFrom->mid = newNode(george,gender);
    else if(insertFrom->right==NULL) insertFrom->right = newNode(george,gender);
}

void nextgeorge(Node* node = root)
{
    if(node==NULL) return;
    if(node->gender == 'M' && node->alive=='L' && found==0)
    {
        kings.push_back(node->george);
        found = 1;
    }
    nextgeorge(node->left);
    nextgeorge(node->mid);
    nextgeorge(node->right);
}

void sawannakot(int george)
{
    Node* temp = searchGeorge(george);
    temp->alive = 'D';
    if(temp->george == kings.back())
    {
        nextgeorge();
        found=0;
    }
}
```

```
int main()
{
    int n,a,george=1;
    char b;
    cin >> n;
    for(int i=0;i<n;i++)
    {
        cin >> a >> b;
        if(b=='D') sawannakot(a);
        else if(b=='M'||b=='F') insertGeorge(a,++george,b);
    }
    for(int i=0;i<kings.size();i++) cout << kings[i] << endl;
}
```

$time\ complexity : O(\mathrm{n}^2)$