



**HỘI THẢO KHOA HỌC LẦN THỨ XII CÁC TRƯỜNG CHUYÊN KHU VỰC
DUYÊN HẢI VÀ ĐỒNG BẰNG BẮC BỘ – NĂM 2019**

CHUYÊN ĐỀ MÔN TIN HỌC
QUY HOẠCH ĐỘNG TRÊN CÂY

MỤC LỤC

PHẦN I: MỞ ĐẦU	2
PHẦN II: NỘI DUNG.....	3
I. LÝ THUYẾT	3
1. Cây.....	3
2. Phương pháp quy hoạch động	4
3. Quy hoạch động trên cây	4
II. BÀI TẬP ỨNG DỤNG.....	5
1. Bài 1: Bài toán Tìm đường đi dài nhất trong cây	5
2. Bài 2: Bài toán Bức ảnh vui nhộn	10
3. Bài 3: Bài toán Cuốn sách phép thuật	12
4. Bài 4: Bài toán Cây con lớn nhất.....	17
5. Bài 5: Bài toán Tập đỉnh bao phủ	20
6. Bài 6: Bài toán: Bữa tiệc vui vẻ	23
7. Bài 7: Bài toán Tách cây.....	26
8. Bài 8: Bài toán Ánh sáng tối ưu	29
9. Bài 9: Bài toán Di chuyển nhanh nhất.....	34
10. Bài 10: Bài toán Liên bang Berland	38
11. Hướng dẫn thuật toán một số bài toán khác.....	43
11.1. Bài toán Tâm của cây.....	43
11.2. Bài toán Cắt cây	43
11.3. Bài toán Cây táo	45
11.4. Bài toán Đuổi bò.....	46
11.5. Bài toán Khách sạn đặc biệt.....	47
11.6. Một số bài toán khác.....	49
PHẦN III: KẾT LUẬN.....	50
TÀI LIỆU THAM KHẢO	51

PHẦN I: MỞ ĐẦU

Trong các kỳ thi học sinh giỏi tin học như: Học sinh giỏi khu vực duyên hải; Học sinh giỏi quốc gia; Olympic tin học quốc tế...Thì các lớp bài toán về tối ưu hóa luôn được ưu tiên lựa chọn trong các đề thi, vì tính ứng dụng vào thực tiễn cao.

Có rất nhiều phương pháp để giải quyết lớp các bài toán tối ưu như: Phương pháp nhánh cận, phương pháp tham lam, phương pháp quy hoạch động ... Tùy từng bài toán cụ thể mà ta chọn một phương pháp để áp dụng nhằm đạt được hiệu quả (hiệu quả về phép tính toán (tốc độ), hiệu quả về bộ nhớ) tốt nhất. Trong đó phương pháp Quy hoạch động luôn được ưu tiên lựa chọn vì tính hiệu quả của chúng cao hơn các phương pháp khác trong đại đa số các bài toán về tối ưu hóa.

Có nhiều bài toán khác nhau có thể sử dụng quy hoạch động như bài toán cái túi, dãy con đơn điệu tăng dài nhất, bài toán ghép cặp, bài toán di chuyển... Trong khuôn khổ chuyên đề “Quy hoạch động trên cây” này, tôi chỉ xin trao đổi với các bạn đồng nghiệp một số bài tập về cây có thể áp dụng phương pháp quy hoạch động, cũng như những kinh nghiệm của tôi khi dạy nội dung này.

Nhân đây, tôi cũng xin gửi lời cảm ơn tới các thầy cô, các bạn đồng nghiệp đã giúp đỡ, cung cấp tài liệu và góp ý để tôi bổ sung, hoàn thiện chuyên đề. Hi vọng rằng, chuyên đề sẽ cung cấp cho các bạn đồng nghiệp và các em học sinh một phần kiến thức bổ ích.

PHẦN II: NỘI DUNG

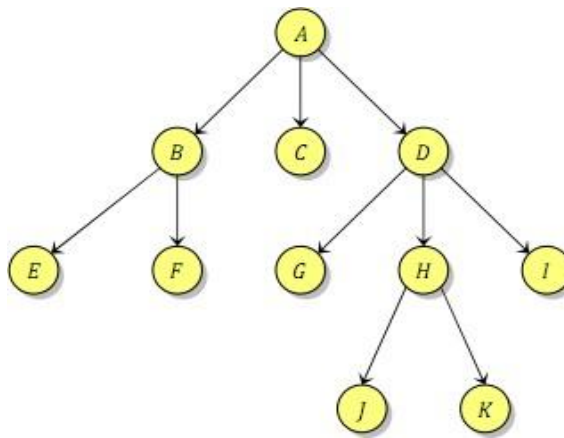
I. LÝ THUYẾT

Lý thuyết về cây và về phương pháp quy hoạch động có rất nhiều nội dung. Nhưng trong khuôn khổ tài liệu này, tôi chỉ xin trình bày những phần nội dung trực tiếp liên quan đến chủ đề Quy hoạch động trên cây.

1. Cây

Cây là đồ thị vô hướng đặc biệt có những tính chất sau:

- Liên thông và không có chu trình
- Giữa 2 đỉnh bất kì có 1 đường đi đơn
- Nếu bỏ đi 1 cạnh bất kì thì cây phân thành 2 cây con



Hình 1. Hình ảnh minh họa về cây

Các bài toán về quy hoạch động trên cây thường sử dụng đến phép duyệt DFS :

DFS sẽ lần lượt ghé thăm các đỉnh của đồ thị một cách đệ quy. Các bước trong một lần ghé thăm đỉnh u được mô tả như sau:

- Đánh dấu u đã được thăm
- Duyệt trong các đỉnh v kề với u : nếu v chưa được thăm thì gọi đệ quy để ghé thăm v .

Giải thuật DFS có thể viết theo mô hình dưới đây:

```
void dfs(int u)
{
    free[u]=false; // đánh dấu đỉnh u đã được thăm
    for (int v=1; v<=n; v++)
        if ((tồn tại cạnh u, v) và (free[u][v]==true)) // tồn tại đỉnh kề
            với u, chưa được thăm
            dfs(v); //duyet đỉnh v
}
```

2. Phương pháp quy hoạch động

Phương pháp Quy hoạch động được sử dụng để giải quyết nhiều bài toán tối ưu. Phương pháp này được thực hiện dựa trên những thao tác cơ bản sau:

Bước 1: Tìm nghiệm của các bài toán con nhỏ nhất

Bước 2: Tìm ra công thức (hoặc quy tắc) xây dựng nghiệm của bài toán con thông qua nghiệm của các bài toán con cỡ nhỏ hơn

Bước 3: Tạo ra một bảng lưu giữ các nghiệm của các bài toán con. Sau đó tính nghiệm của các bài toán con theo công thức đã tìm ra và lưu vào bảng

Bước 4: Từ các bài toán con đã giải để tìm ra nghiệm của bài toán.

3. Quy hoạch động trên cây

Mỗi bài toán tối ưu có một đặc thù riêng, cách giải hoàn toàn khác nhau, cho nên việc áp dụng phương pháp Quy hoạch động cho các bài toán cũng hoàn toàn khác nhau, không có khuôn mẫu chung cho tất cả các bài.

Với các bài tập về cây cũng vậy, tuy nhiên, khi làm bài ta có thể lưu ý một số kinh nghiệm sau:

- Thuật toán Quy hoạch động trên cây chủ yếu dựa vào các tính chất đặc biệt trên của cây. Vậy nên trong các bài tập, ta cần xác định quan hệ cha – con. Các bài tập thường cần đến thao tác cập nhật dữ liệu từ các nút con lên u , và cập nhật từ nút cha xuống dưới u .
- Việc xác định quan hệ cha – con trong cây có thể sử dụng phép duyệt DFS, khi đó những đỉnh v được thăm tiếp theo trong quá trình duyệt đỉnh u sẽ nhận đỉnh u làm cha. Thuật toán DFS vừa được nhắc ở trên sẽ được cài đặt linh hoạt tùy vào bài toán cụ thể.
- Ta có một số bài toán quy hoạch động trên cây cơ bản sau:

Bài toán 1: Tập độc lập lớn nhất trên cây

Bài toán 2: Tìm đường đi dài nhất trong cây

Bài toán 3: Tính tổng tối đa của các giá trị từ nút gốc đến bất kỳ lá nào mà không truy cập lại bất kỳ nút nào.

Bài toán 4: Tìm ra chiều cao tối đa của cây khi bất kỳ nút nào trong cây được coi là gốc của cây.

Bài toán 5: Tìm số cây phụ khác nhau với kích thước nhỏ hơn hoặc bằng một số nguyên k .

....

Những bài toán này, và những bài toán là mở rộng, nâng cao của chúng sẽ được hướng dẫn tường minh trong mục **II. Bài tập ứng dụng**.

II. BÀI TẬP ỨNG DỤNG.

1. Bài 1: Bài toán Tìm đường đi dài nhất trong cây

Nguồn bài: <https://www.spoj.com/problems/PT07Z/>

1.1. Đề bài

Cho một cây không trọng số. Độ dài của một đường đi trong cây là số cạnh đi qua từ nút nguồn tới nút đích.

Yêu cầu: Hãy viết chương trình tìm đường đi dài nhất trong cây.

Dữ liệu: Vào từ file văn bản **PT07Z.INP**

- Dòng đầu tiên chứa một số nguyên N là số nút trong cây ($0 < N \leq 10000$).
- $N - 1$ dòng sau, mỗi dòng chứa hai số (u, v) mô tả một cạnh của cây ($1 \leq u, v \leq N$).

Kết quả: Ghi ra file văn bản **PT07Z.OUT**

Một số nguyên duy nhất là độ dài đường đi dài nhất trong cây đó

Ví dụ:

PT07Z.INP	PT07Z.OUT
3 1 2 2 3	2

Ràng buộc:

- Có 50% số test tương ứng 20% số điểm có $N \leq 1000$
- 50% số test còn lại tương ứng 50% số điểm có $1000 < N \leq 10000$

❖ **Xác định bài toán:**

Input: Số nguyên N ($0 < N \leq 10000$).

$N - 1$ cặp (u, v) ($1 \leq u, v \leq N$).

Output: Độ dài đường đi dài nhất trong cây.

1.2. Hướng dẫn giải thuật.

Cách 1:

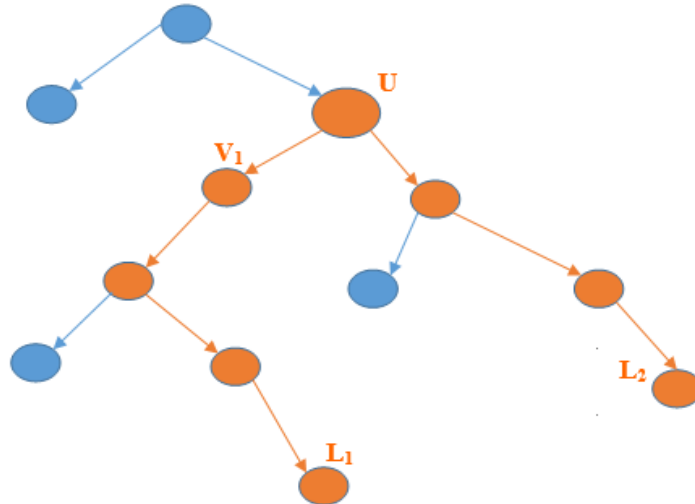
Đầu tiên, ta thực hiện DFS từ một đỉnh bất kì nào đó (chẳng hạn đỉnh 1), giả sử đỉnh có khoảng cách lớn nhất đến đỉnh 1 là đỉnh k . Thực hiện DFS lần thứ 2 từ đỉnh k , thì kết quả là khoảng cách từ đỉnh có khoảng cách xa k nhất đến k .

Cách 2:

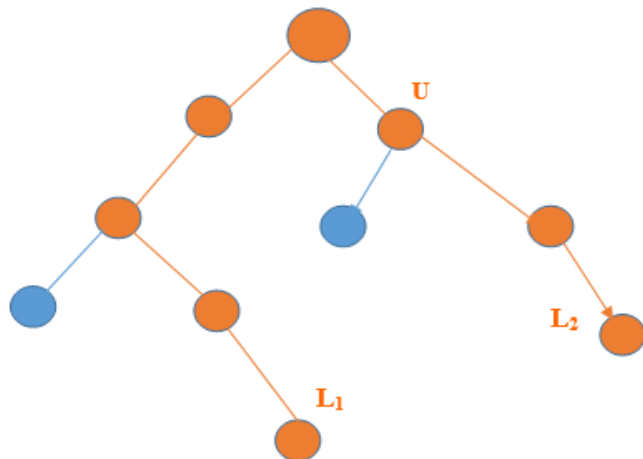
Khi ta đã quy ước thứ tự của cây (Nút nào là cha, nút nào là con) thì khi xét đến một nút U có các đỉnh con là V_i ta có một số nhận xét sau:

- Nếu cây con dài nhất đi qua U thì nó sẽ có hai trường hợp:

- Trường hợp 1: Cây con sẽ đi qua 2 lá mà U quản lý và đi qua cả U với 2 lá này có khoảng cách xa nhất so với U (Hình 2).

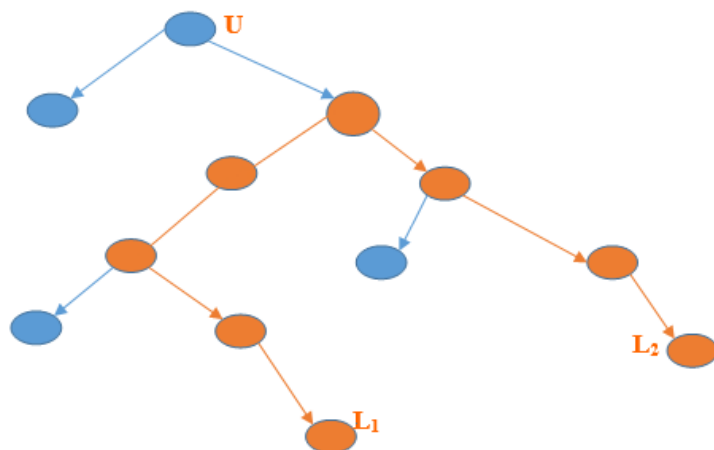


Hình 2. Minh họa trường hợp 1 của cây con dài nhất đi qua đỉnh U đang xét



Hình 3. Minh họa trường hợp 2 của cây con dài nhất đi qua đỉnh U đang xét

Nếu cây con dài nhất không đi qua U thì cây con dài nhất sẽ hoặc nằm trong các nút thuộc sự quản lý của U hoặc không (Hình 4).



Hình 4. Minh họa trường hợp cây con dài nhất không đi qua đỉnh U đang xét
Từ đặc điểm trên ta thực hiện quy hoạch động với cây như sau:

- Xét đỉnh U có các đỉnh con trực tiếp là các đỉnh V_i .

Gọi $F[U]$ là cây con lớn nhất mà U quản lý (có thể đi qua U hoặc không).

Gọi $LF[U]$ là khoảng cách từ U đến lá xa nhất mà U quản lý.

- *Bài toán con nhỏ nhất:*

Xét đối với các lá thì đó chính là bài toán con nhỏ nhất. Ta dễ dàng thấy $F[lá] = 1$, $LF[lá] = 1$;

- *Công thức truy hồi:*

Từ trên ta rút ra công thức để tính $LF[U]$, $F[U]$ như sau:

$$LF[U] = \text{Max}(LF[V_i]) + 1 \text{ với } V_i \text{ là các con của } U$$

- Với Trường hợp 1 thì: $F[U] = LF[V1] + LF[V2] + 1$

Với $LF[V1]$, $LF[V2]$ là hai giá trị lớn nhất của các đỉnh V_i là con của U .

- Với Trường hợp 2 khi mà cây con dài nhất nằm trong các đỉnh U quản lý thì rõ ràng: $F[U] = \max(F[V_i])$.

Vậy kết quả cuối cùng của bài toán chính là $F[1] - 1$ với $F[1]$ chính là đỉnh gốc của cây.

1.3. Chương trình.

Cách 1:

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define Mx 10005
vector<ll> gr[Mx];
ll ress,k,n;
bool visited[Mx];
void Dfs(ll u,ll kc)
{
    if (ress<kc)
    {
        ress=kc;
        k=u;
    }
    visited[u]=false;
    forr(i,0,gr[u].size())
        if (visited[gr[u][i]])
            Dfs(gr[u][i],kc+1);
}
int main()
{
    freopen("PT07Z.inp","r",stdin);
```



```

freopen("PT07Z.out", "w", stdout);
ll u, v;
cin >> n;
forr(i, 1, n)
{
    cin >> u >> v;
    gr[u].push_back(v);
    gr[v].push_back(u);
}
ress = 0;
k = 1;
forr(i, 1, n + 1) visited[i] = true;
Dfs(1, 0);
forr(i, 1, n + 1) visited[i] = true;
Dfs(k, 0);
cout << ress;
}

```

Cách 2:

```

#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define forr(i, a, b) for(ll i = a; i < b; i++)
#define Mx 10005
ll n;
vector<ll> gr[Mx];
ll F[Mx], LF[Mx];
bool visited[Mx];
void Dfs(ll u)
{
    ll M1 = 0, M2 = 0, v;
    visited[u] = false;
    forr(i, 0, gr[u].size())
    {
        v = gr[u][i];
        if (visited[v])
        {
            Dfs(v);
            F[u] = max(F[u], F[v]);
            LF[u] = max(LF[u], LF[v] + 1);
            if (LF[v] > M2)
                if (LF[v] > M1) { M2 = M1; M1 = LF[v]; }
            else M2 = LF[v];
        }
    }
}

```

```

    }
    }
    F[u]=max(F[u],M1+M2+1);
}
int main()
{
    freopen("PT07Z.inp","r",stdin);
    freopen("PT07Z.out","w",stdout);
    ll u,v;
    cin>>n;
    forr(i,1,n)
    {
        cin>>u>>v;
        gr[u].push_back(v);
        gr[v].push_back(u);
    }
    forr(i,1,n+1)
    {
        F[i]=1;
        LF[i]=1;
        visited[i]=true;
    }
    Dfs(1);
    cout<<F[1]-1;
}

```

1.4. Độ phức tạp:

Trong bài trên ta đã cài đặt theo ý tưởng trên bằng thuật toán duyệt theo chiều sâu Dfs vừa để định hướng thứ tự ưu tiên vừa tính toán đệ quy các $F[u]$, $LF[u]$.

- Do các đỉnh chỉ được duyệt duy nhất một lần nên độ phức tạp của thuật toán là $O(n) * F$ với F là độ phức tạp khi duyệt Dfs các đỉnh con theo cạnh của đỉnh cần xét.
- Nếu ta lưu trữ các cạnh của đồ thị bằng mảng 2 chiều thì độ phức tạp của thuật toán sẽ là $O(n^2)$. Còn ở 2 cách trên ra lưu trữ các cạnh bằng các kiểu cấu trúc khác (cụ thể là kiểu Vector) nên ta đã giảm được độ phức tạp của thuật toán xuống còn $O(n)$.

1.5. Test

<https://drive.google.com/file/d/17kxiG996b3IJML8zCcgydecTmP580C5j/view?usp=sharing>

1.6. Cảm nhận

- Để cài đặt tốt bài này ta cần phải cẩn thận trong phân chia các trường hợp khi xét đến một nút để tìm ra tính chất và công thức hợp lý cho mỗi trường hợp, phải nắm thật chắc các tính chất của duyệt Dfs để cài đặt đem lại hiệu quả.

- Đây là một bài nền móng cho các bài quy hoạch động sau nên bạn đọc có thể cải tiến Code cài đặt để đem lại những thông tin cần thiết hơn để có thể áp dụng các bài sau này như: Thay mảng đánh dấu Visited bằng Cha (Đánh dấu thứ tự cha con), hoặc có thể thay đổi tính chất mảng $F[U]$ thành cây con dài nhất đi qua nút U ...

2. Bài 2: Bài toán Bức ảnh vui nhộn

Nguồn bài: <http://codeforces.com/problemset/problem/522/A>

2.1. Đề bài

Một ngày nọ, Polycarp đăng một bức ảnh trên mạng xã hội Facebook. Nhiều người bạn của anh ấy chia sẻ lại bức ảnh, một số khác thì trực tiếp đăng lại bức ảnh đó.

Việc đăng và chia sẻ ảnh là các chuỗi liên nhau có dạng " **tên1** reposted **tên2**", trong đó **tên1** là tên của người đã chia sẻ bức ảnh và **tên2** là tên của người đã đăng bức ảnh. Biết rằng, vào thời điểm chia sẻ ảnh, với mỗi chuỗi "tên1 reposted tên2" thì "tên1" chưa có bức ảnh trong bảng tin của họ và "tên2" đã có ảnh trong bảng tin của họ. Polycarp đã được đăng ký là "**Polycarp**" và ban đầu bức ảnh chỉ có trong bảng tin của anh ấy.

Yêu cầu: Tìm mức độ phổ biến của bức ảnh mà Polycarp đã đăng, biết mức độ phổ biến của bức ảnh là độ dài của chuỗi chia sẻ ảnh lớn nhất.

Dữ liệu: Vào từ file văn bản **REPOST.INP**

- Dòng đầu tiên chứa số nguyên n ($1 \leq n \leq 200$) là số lần chia sẻ.
- n dòng sau là các chuỗi chia sẻ ảnh theo thứ tự. Mỗi chuỗi trong số chúng được viết trên một dòng duy nhất có dạng "tên1 reposted tên2". Tất cả các tên trong đầu vào có thể bao gồm chữ cái viết thường hoặc viết hoa, chữ số có độ dài từ 2 đến 24 ký tự. Tên người dùng không phân biệt chữ hoa chữ thường.

Kết quả: Ghi ra file văn bản **REPOST.OUT**

- Tìm mức độ phổ biến bức ảnh (độ dài chuỗi chia sẻ ảnh)

Ví dụ:

REPOST.INP	REPOST.OUT
5 tourist reposted Polycarp Petr reposted Tourist WJMZBMR reposted Petr sdya reposted wjmzbnr vepifanov reposted sdya	6

❖ Xác định bài toán:

Input: Số n nguyên dương ($1 \leq n \leq 200$).

n chuỗi có dạng " **tên1** reposted **tên2**".

Output: Một số nguyên duy nhất là mức độ phổ biến của bức ảnh.

2.2. Hướng dẫn giải thuật.

- Dựa vào những dữ kiện đã cho của đề bài ta dễ dàng thấy đây là một cây có nút gốc là “polycarp”, và hướng đi của các nút đã được định sẵn (theo thứ tự chia sẻ bức ảnh). Nhiệm vụ của chúng ta là tìm khoảng cách xa nhất từ nút gốc tới nút lá.

Từ nhận xét trên ta thực hiện quy hoạch động như sau:

- Gọi $F[U]$ là khoảng cách từ nút gốc cho đến nút U .
- Bài toán con nhỏ nhất: $F[1] = 1$ (tại nút gốc)
- Công thức truy hồi: $F[V] = F[U] + 1$ (với V là nút con của U)

2.3. Chương trình.

```
#include <bits/stdc++.h>
#include <map>
using namespace std;
#define ll long long
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define Maxn 100015
map<string,ll> F;
ll n,ress;
int main()
{
    freopen("repost.inp","r",stdin);
    freopen("repost.out","w",stdout);
    string a,b,c;
    cin>>n;
    ress=1;
    F["polycarp"]=1;
    forr(i,0,n)
    {
        cin>>a>>b>>c;
        forr(i,0,a.size()) a[i]=tolower(a[i]);
        forr(i,0,c.size()) c[i]=tolower(c[i]);
        F[a]=F[c]+1;
        if(F[a]>ress) ress=F[a];
    }
    cout<<ress;
}
```

2.4. Độ phức tạp:

- Thông thường ta sẽ cài đặt bằng Dfs nhưng do dữ kiện của đề bài đã cho “tên2” luôn tồn tại ở phía trước rồi nên thứ tự duyệt theo các cạnh vẫn đáp ứng được kết quả chính xác. Vậy nên ở bài trên ta đã duyệt tuần tự. Độ phức tạp: $O(n)$.

2.5. Test

<https://drive.google.com/file/d/1i9R8Tfa78PgXjSaUYUIrDgGdUL690N-M/view?usp=sharing>

2.6. Cảm nhận

Một số lưu ý khi làm bài:

- Do chuỗi không phân biệt viết hoa – thường nên ta phải chuyển tất cả về chữ thường để có thể cài đặt.
- Giá trị tại gốc là $F[1]$ phải được tạo sẵn.
- Do đề cho tên các nút bằng chữ nên ta sẽ dùng map để tiện cho lưu trữ. Nếu không dùng map ta có thể dùng mảng để lưu nhưng sẽ tốn thêm $O(n)$. Mặc dù vẫn đáp ứng được yêu cầu bài toán nhưng sẽ không tối ưu bằng map.

3. Bài 3: Bài toán Cuốn sách phép thuật

Nguồn bài: <http://codeforces.com/problemset/problem/337/D>

3.1. Đề bài

Harry đã biết về một cuốn sách dạy phép thuật được cha mẹ mình giấu trong một khu đầm lầy. Khu đầm lầy này có N khu dân cư được đánh số thứ tự từ 1 đến N và được nối với nhau bởi $N - 1$ con đường, trong đó mọi cặp khu dân cư đều có đường đi đến nhau.

Khoảng cách giữa 2 khu định cư (x, y) được hiểu là số lượng con đường đi qua để đi từ x đến y . Harry biết rằng cuốn sách được giấu tại một khu dân cư nào đó, và nó có một sức mạnh quyền năng sẽ làm ảnh hưởng xấu tới những khu dân cư khác (làm cho khu các dân cư này xuất hiện bóng ma hoặc người sói) cách khu dân cư chứa cuốn sách trong vòng bán kính có khoảng cách là d .

Harry có trong tay một danh sách M khu dân cư bị ảnh hưởng bởi cuốn sách, đó là các khu dân cư p_1, p_2, \dots, p_M .

Yêu cầu: Hãy giúp Harry xác định xem có bao nhiêu khu dân cư có thể chứa cuốn sách mà anh cần tìm.

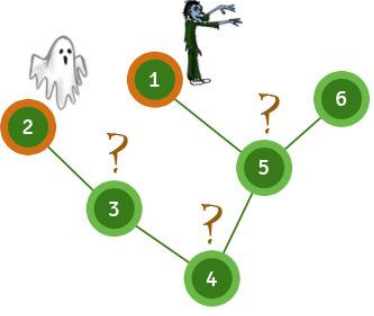
Dữ liệu: vào từ file văn bản **BOOK.INP**

- Dòng đầu tiên chứa 3 số nguyên N, M và d ($1 \leq M \leq N \leq 10^5$; $0 \leq d \leq N - 1$)
- Dòng thứ hai chứa M số nguyên p_1, p_2, \dots, p_M ($1 \leq p_i \leq N$)
- $N - 1$ dòng tiếp theo, mỗi dòng chứa cặp x, y thể hiện một con đường nối trực tiếp từ x đến y .

Kết quả: Ghi ra file văn bản **BOOK.OUT**

- Đưa ra số lượng khu đầm lầy có thể chứa cuốn sách phép thuật, nếu như không có thì đưa ra số 0.

Ví dụ:

BOOK.INP	BOOK.OUT	Hình vẽ giải thích
6 2 3 1 2 1 5 2 3 3 4 4 5 5 6	3	

Ràng buộc:

- Có 25% số test tương ứng 25% số điểm có $N \leq 50$
- Có 25% số test khác tương ứng 25% số điểm có $50 < N \leq 100$
- Có 25% số test khác tương ứng 25% số điểm có $100 < N \leq 5 \cdot 10^4$
- 25% số test còn lại tương ứng 25% số điểm có $5 \cdot 10^4 < N \leq 10^5$

❖ **Xác định bài toán:**

Input:

- 3 số nguyên N, M và d ($1 \leq M \leq N \leq 10^5$; $0 \leq d \leq N - 1$)
- M số nguyên p_1, p_2, \dots, p_M ($1 \leq p_i \leq N$)
- $N - 1$ cặp (x, y)

Output: số lượng khu đầm lầy có thể chứa cuốn sách phép thuật hoặc số 0

3.2. Hướng dẫn giải thuật.

- Đề bài cho là một cây đã được định hướng thứ tự ưu tiên giữa các nút, trong đó có các nút đen, vấn đề của bài toán thực chất là đếm số nút thỏa mãn khoảng cách từ nó tới các nút đen không vượt quá d . Nếu ta làm theo thông thường là cứ với mỗi nút tìm xem nút đó thỏa mãn hay không bằng cách tìm khoảng cách tất cả các nút đen so với nó thì độ phức tạp cho mỗi đỉnh sẽ là $O(n)$. Nếu thế thì sẽ không đáp ứng được bài toán. Vậy mấu chốt của bài toán ta phải xác định được khoảng cách xa nhất từ một nút đen bất kỳ tới nút ta đang xét đó chỉ trong $O(1)$. Yêu cầu của bài toán có nét tương đồng với bài toán xác định khoảng cách xa nhất của từ một nút đến nút lá (*Bài toán 2. Bức ảnh vui nhộn*)

Từ các nhận xét trên, ta sử dụng Quy hoạch động như sau:

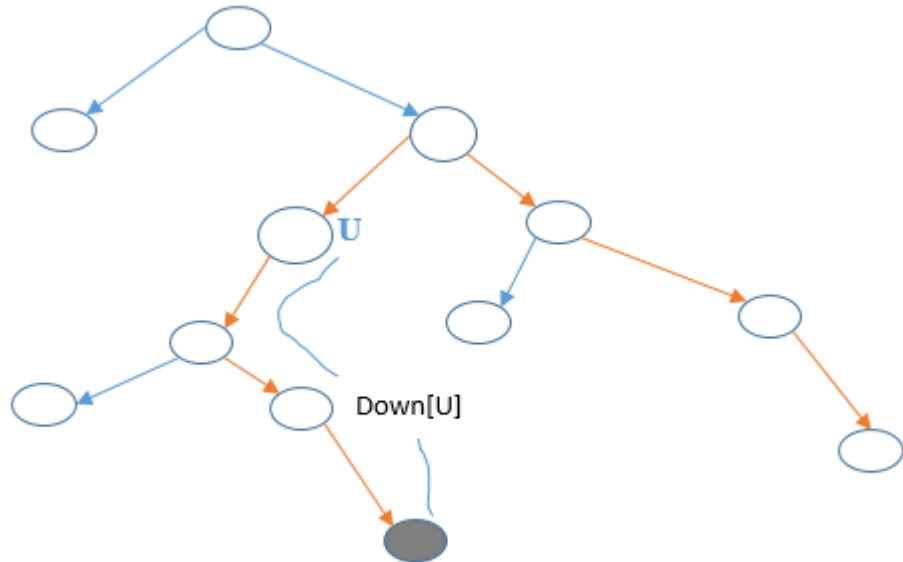
- *Bài toán con nhỏ nhất:* Là trường hợp ở tại các đỉnh có ma hoặc sói, vì ở mỗi đỉnh ta xác định khoảng cách xa nhất từ nó tới nút có ma hoặc sói, nên trong trường hợp này khoảng cách là 0.

- Công thức truy hồi:

Ta xét 2 trường hợp sau:

Trường hợp 1: Nút đen xa nhất là một nút V_i là nút con mà U quản lý. Để tìm được khoảng cách giữa U và V_i ta sử dụng một mảng $Down[u]$ với ý nghĩa là khoảng cách xa nhất từ nút U đến nút đen mà U quản lý. (Hình 5)

Vậy, công thức truy hồi là: $Down[U] = \max(Down[V_i] + 1)$.

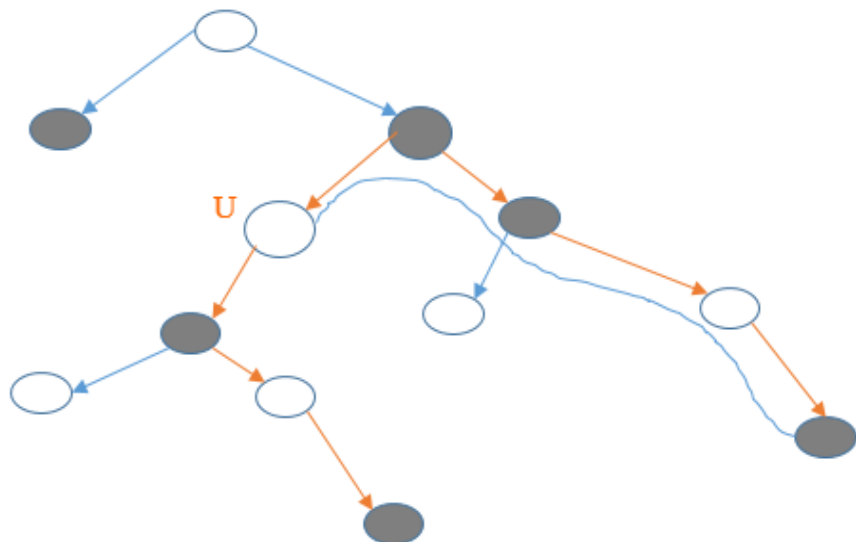


Hình 5. Nút đen xa nhất so với U là một nút mà U quản lý

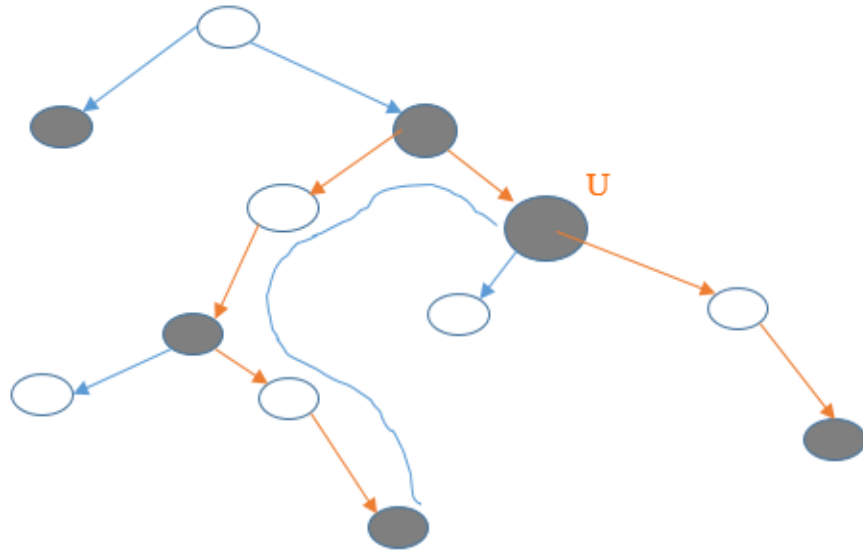
Trường hợp 2: Nút đen xa nhất sẽ nằm ở phía trên U . Như vậy lại có thêm hai khả năng hoặc V_i nằm ở phần bù của U so với nút cha của U (Hình 6). Hoặc nó có thể đi lên phía trên của cha của U (Hình 7). Để tìm kiếm được như vậy ta sẽ dùng $Up[U]$ với ý nghĩa là khoảng cách xa nhất từ nút U tới nút đen nằm trên nó.

Vậy, công thức truy hồi là: $Up[U] = Up[Cha[U]] + 1$ (nếu nút đen xa nhất nằm trên nút cha của U)

Hoặc $Up[U] = Down[V_i] + 2$ (nếu nút đen xa nhất nằm ở phần bù của U so với nút cha của U , và V_i là nút khác nút U và cũng là con của nút cha U)



Hình 6. Nút đen xa nhất nằm ở phía trên U và ở phần bù của U so với nút cha của U



Hình 7. Nút đen xa nhất nằm ở phía trên U và đi lên phía trên nút cha của U

- Với ý tưởng trên thì ta sẽ sử dụng Dfs để tiến hành cài đặt. Vấn đề của chúng ta là mảng Up chỉ tính được khi đã có mảng Down nên rõ ràng ta phải Dfs 2 lần. Lần một là để tính Down, một lần nữa để tính Up.
- Ngoài ra để tìm được V_i là phân bù của U so với cha U thì ta sẽ sử dụng thêm mảng LM để lưu trữ.

3.3. Chương trình.

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define Maxn 100005
const ll ma=-1000000;
vector<ll> gr[Maxn];
ll LM[3][Maxn],n,m,Affac[Maxn],Down[Maxn],Up[Maxn],Cha[Maxn],d;
void DfsD(ll u)
{
    ll v;
    if (Affac[u]) Down[u]=0;
    else Down[u]=ma;
    LM[1][u]=ma;
    LM[2][u]=ma;
    forr(i,0,gr[u].size())
    {
        v=gr[u][i];
        if (v!=Cha[u])
        {
            Cha[v]=u;
            DfsD(v);
            if (Down[v]>LM[2][u])
```



```

        {
            if (Down[v]>LM[1][u])
            {
                LM[2][u]=LM[1][u];
                LM[1][u]=Down[v];
            }
            else LM[2][u]=Down[v];
            Down[u]=LM[1][u]+1;
        }
    }
}

void DfsU(ll v)
{
    ll u;
    if (v==1)
    {
        if (Affac[1]) Up[1]=0;
        else Up[1]=ma;
    }
    else
    {
        u=Cha[v];
        Up[v]=Up[u]+1;
        if ((Affac[v]) and (Up[v]<0)) Up[v]=0;
        if (LM[1][u]==Down[v])
            Up[v]=max(Up[v],LM[2][u]+2);
        else Up[v]=max(Up[v],LM[1][u]+2);
    }
    forr(i,0,gr[v].size())
        if (gr[v][i]!=Cha[v])
            DfsU(gr[v][i]);
}

int main()
{
    freopen("book.inp","r",stdin);
    freopen("book.out","w",stdout);
    ll x,y;
    cin>>n>>m>>d;
    forr(i,1,n+1) Affac[i]=0;
    forr(i,0,m)
    {
        cin>>x;

```

```

        Affac[x]=1;
    }
    forr(i,0,n-1)
    {
        cin>>x>>y;
        gr[x].push_back(y);
        gr[y].push_back(x);
    }
    DfsD(1);
    DfsU(1);
    ll ress=0;
    forr(i,1,n+1)
    if ((Down[i]<=d) and (Up[i]<=d)) ress++;
    cout<<ress;
}

```

3.4. Độ phức tạp.

- Độ phức tạp của thuật toán: $O(n)$.

3.5 Test

<https://drive.google.com/file/d/1IaoYcrCHM4ZjEzyeBwYlb3w3EcwKE7l4/view?usp=sharing>

3.6 Cảm nhận

- Đây là bài toán có khá nhiều trường hợp khi xét 1 đỉnh nên nếu ta không cẩn thận thì sẽ bỏ sót các trường hợp của bài toán.
- Khi ta sử dụng mảng LM nếu nóng vội không kiểm tra xem V_i ta đang lưu trữ có trùng với nút U ta đang xét hay không thì dẫn đến kết quả sai.
- Khi xét một đỉnh ta cũng nên cân nhắc giá trị ban đầu của $Up[U]$ và $Down[U]$ phù hợp với nút đó là đen hay trắng.

4. Bài 4: Bài toán Cây con lớn nhất

Nguồn bài: www.spoj.com/CTN/problems/MAXTREE

4.1. Đề bài

Cho một cây với các cạnh có trọng số. Trọng số của cây được định nghĩa bằng tổng trọng số các cạnh. Một cây con là cây thu được bằng cách xóa đi một số đỉnh (và các cạnh có ít nhất một đỉnh bị xóa).

Yêu cầu: Cho một cây, hãy tìm cây con có trọng số lớn nhất.

Dữ liệu: Vào từ file văn bản **MAXTREE.INP**:

- Dòng đầu ghi số N là số đỉnh của cây. ($1 \leq N \leq 5 \cdot 10^4$)
- $N - 1$ dòng sau, mỗi dòng ghi 3 số u, v, c thể hiện một cạnh của cây nối 2 đỉnh u, v , có trọng số là c . ($1 \leq u, v \leq N, -10^4 \leq c \leq 10^4$)

Kết quả: Đưa ra file văn bản **MAXTREE.OUT**:

- Ghi ra một số duy nhất là trọng số lớn nhất có thể của một cây con.

Ví dụ:

MAXTREE.INP	MAXTREE.OUT
5	3
5 1 2	
1 2 -5	
2 3 1	
2 4 2	

Ràng buộc:

- Có 30% số test tương ứng 30% số điểm có $N < 100$
- Có 40% số test khác tương ứng 40% số điểm có $100 \leq N < 10000$
- 30% số test còn lại tương ứng 30% số điểm có $10^4 \leq N \leq 5 \cdot 10^4$

❖ **Xác định bài toán:**

Input: Số nguyên dương N ($1 \leq N \leq 5 \cdot 10^4$)

$N - 1$ bộ số u, v, c ($1 \leq u, v \leq N, -10^4 \leq c \leq 10^4$)

Output: một số nguyên là trọng số lớn nhất của một cây con.

4.2. Hướng dẫn giải thuật.

- Yêu cầu của bài toán là tìm cây con có tổng các cạnh lớn nhất (trọng số lớn nhất). Vậy để làm được bài này ta phải tìm cách xây dựng các cây con từ bé tới lớn rồi tìm cách ghép các cây sao cho trọng số lớn nhất.
- Giả sử cây đã quy định thứ tự (Nút cha, nút con) thì với nút U , nếu ta xét trong phạm vi các nút U quản lý để xây dựng cây con thì có hai trường hợp. Hoặc là cây con ta xây dựng có U , hoặc cây ta xây dựng không có U :
 - TH1: Cây con ta xây dựng có U thì nó sẽ nằm độc lập hoặc nó sẽ gắn với cây con có gốc là V_i với V_i là con của U .
 - TH2: Cây con không đi qua U thì rõ ràng cây con đó phải nằm ở một trong số các nhánh V_i là con của U .

Từ các nhận xét trên ta sử dụng Quy hoạch động như sau:

- **Công thức truy hồi:**
 - Gọi $F[U]$ là trọng số lớn nhất của cây đi qua U và các nút do U quản lý. $G[U]$ là trọng số lớn nhất của cây do U quản lý nhưng không đi qua U .
 - Suy ra, $F[U]$ bằng tổng trọng số các cây dương (cây có trọng số dương) có gốc là V_i là con của U (Vì nếu cây V_i có trọng số âm thì việc thêm cây V_i vào U chỉ làm giảm trọng số cây U).
 - $G[U] = \max(G[V_i], F[V_i])$.
 - Kết quả bài toán chính là $\max(G[1], F[1])$.
- **Bài toán con nhỏ nhất:** Là bài toán tại các nút lá (Nút không có cấp dưới), khi đó $F[\text{lá}] = 0, G[\text{lá}] = 0$.

4.3. Chương trình.

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define Maxn 100005
const ll ma=-1e18;
struct data{
    ll con,gt;};
vector<data> gr[Maxn];
ll F[Maxn],Ress,n,G[Maxn];
bool visit[Maxn];
void Dfs(ll u)
{
    ll v,c,l1=0,l2=0;
    F[u]=0; G[u]=0;
    visit[u]=false;
    forr(i,0,gr[u].size())
    {
        v=gr[u][i].con;
        if (visit[v])
        {
            Dfs(v);
            c=gr[u][i].gt;
            F[u]=max(F[u],F[u]+F[v]+c);
            G[u]=max(G[u],max(G[v],F[v]));
        }
    }
}
int main()
{
    freopen("maxtree.inp","r",stdin);
    freopen("maxtree.out","w",stdout);
    ll u,v,c;
    data tg;
    cin>>n;
    forr(i,1,n+1) visit[i]=true;
    forr(i,1,n)
    {
        cin>>u>>v>>c;
        tg.con=v; tg.gt=c;
        gr[u].push_back(tg);
        tg.con=u;
        gr[v].push_back(tg);
    }
    Dfs(1);
    cout<<max(F[1],G[1]);
}
```

4.4. Độ phức tạp

- Trong bài trên, ta đã sử dụng thuật toán duyệt Dfs để tạo thứ tự cho cây đồng thời tính luôn các giá trị của mảng F và G .
- Do mỗi đỉnh ta chỉ duyệt 1 lần nên độ phức tạp sẽ là $O(n) * P$ với P là độ phức tạp khi tham chiếu tới các đỉnh con. Do ta cài bằng vector nên P xấp xỉ bằng 1. Vì vậy, độ phức tạp của thuật toán là $O(n)$.

4.5. Test.

<https://drive.google.com/file/d/1UrR559CT46NuXXZGGnfVmf9aS2cFRwdt/view?usp=sharing>

4.6. Cảm nhận

Cây ta đang xét có thể chỉ là một nút nên khi cài đặt phải cân trọng trong việc đặt các giá trị cho F và G .

5. Bài 5: Bài toán Tập đỉnh bao phủ

Nguồn bài: <https://www.spoj.com/problems/PT07X/>

5.1. Đề bài

Cho một cây không trọng số.

Yêu cầu: Viết chương trình tìm tập đỉnh nhỏ nhất trong cây sao cho mỗi cạnh có ít nhất một điểm cuối của nó trong tập đó.

Dữ liệu: Vào từ file văn bản **PT07X.INP**:

- Dòng đầu chứa một số nguyên N là số nút trong cây ($0 < N \leq 100000$).
- $N - 1$ dòng sau, mỗi dòng chứa 2 số nguyên (u, v) có nghĩa là có một cạnh giữa nút u và nút v ($1 \leq u, v \leq N$).

Kết quả: Ghi ra file văn bản **PT07X.OUT**:

- Một dòng duy nhất chứa số lượng nút trong cây thỏa mãn yêu cầu trên.

Ví dụ:

PT07X.INP	PT07X.OUT
3 1 2 1 3	1

Ràng buộc:

- Có 30% số test tương ứng 30% số điểm có $N < 1000$
- Có 40% số test khác tương ứng 40% số điểm có $1000 \leq N \leq 10^4$
- 30% số test còn lại tương ứng 30% số điểm có $10^4 < N \leq 10^5$

❖ Xác định bài toán:

Input: Số nguyên N ($0 < N \leq 100000$).

$N - 1$ cặp (u, v) ($1 < u, v \leq N$)

Output: Số lượng nút trong cây thỏa mãn.

5.2. Hướng dẫn giải thuật.

- Ta có một số nhận xét về bài toán như sau:
 - Do tập mà ta cần chọn phải thỏa mãn yêu cầu các cạnh trong cây phải có ít nhất 1 đỉnh thuộc tập hợp, nên suy ra, với hai đỉnh bất kỳ có cạnh nối với nhau thì ít nhất một trong hai đỉnh phải được chọn, hay nói cách khác không thể có hai đỉnh đồng thời không được chọn.
 - Đối với một đỉnh ta chỉ xoay quanh vấn đề đỉnh đó có được chọn hay không và nếu chọn hay không chọn thì nó ảnh hưởng thế nào đến các đỉnh khác. Ví dụ ta đang xét đỉnh U , nếu đỉnh U không được chọn thì tất cả các đỉnh nối với U đều phải được chọn, còn nếu U được chọn thì các đỉnh kia có thể chọn hay không chọn tùy ý.
- Từ nhận xét trên dễ dàng nghĩ ngay đến sử dụng quy hoạch động như sau:
Khi đã xây dựng thứ tự cho các đỉnh (Nút con - nút cha) thì ta sẽ sử dụng mảng $F[2][N]$ với ý nghĩa: $F[0][i]$ là số lượng đỉnh nhỏ nhất khi ta không chọn nút i và có thể chọn một số đỉnh con do i quản lý, tương tự: $F[1][i]$ là số lượng đỉnh min khi ta chọn đỉnh i và một số đỉnh con i quản lý.
- *Bài toán con nhỏ nhất:*
Bài toán con nhỏ nhất ở đây chính là với các nút lá: $F[0][\text{lá}] = 0$, $F[1][\text{lá}] = 1$.
- *Công thức truy hồi:*
 - $F[0][u] = \sum F[1][v]$ với v là các đỉnh con của u .
 - $F[1][u] = \sum \min(F[1][v], F[0][v])$ với v là các đỉnh con của u .Do ta sẽ bắt đầu từ đỉnh 1 nên đỉnh 1 sẽ là gốc của cây và đỉnh 1 sẽ quản lý tất cả các đỉnh, vậy nên kết quả chính là $\min(F[1][1], F[0][1])$.

5.3. Chương trình.

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define N 500005
const ll ma=1000000000;
vector<ll> gr[N];
ll res,n,Cha[N],F[2][N];
void Dfs(ll u)
{
```

```

F[0][u]=0;
F[1][u]=1;
forr(i,0,gr[u].size())
    if (gr[u][i]!=Cha[u])
        {
            ll v=gr[u][i];
            Cha[v]=u;
            Dfs(v);
            F[0][u]+=F[1][v];
            F[1][u]+=min(F[1][v],F[0][v]);
        }
}
int main()
{
    ll u,v;
    freopen("PT07X.inp","r",stdin);
    freopen("PT07X.out","w",stdout);
    cin>>n;
    forr(i,0,n-1)
        {
            cin>>v>>u;
            gr[u].push_back(v);
            gr[v].push_back(u);
        }
    Cha[1]=0;
    Dfs(1);
    cout<<min(F[1][1],F[0][1]);
}

```

5.4. Độ phức tạp.

- Để thực hiện được ý tưởng trên ta sẽ sử dụng Dfs để vừa phân thứ tự cho cây vừa tính luôn mảng F .
- Mỗi đỉnh ta chỉ thăm 1 lần nên độ phức tạp sẽ là $O(n) * k$ với k là độ phức tạp khi ta thăm các đỉnh con của đỉnh đang xét. Ta sẽ sử dụng vector để lưu trữ các cạnh nên độ phức tạp trung bình $k = O(1) \Rightarrow$ độ phức tạp của thuật toán là $O(n)$.

5.5. Test.

<https://drive.google.com/file/d/1Bg169tleDPnoVpT3wp42LbEsAbnJnpG3/view?usp=sharing>

5.6. Cảm nhận

- Khi cài đặt ta cần chú ý giá trị đầu tiên khi tính các giá trị F bởi nó sẽ quyết định trực tiếp đến kết quả của bài toán.
- Việc lựa chọn cách lưu trữ các cạnh ảnh hưởng trực tiếp đến độ phức tạp của thuật toán nên ta sẽ phải chọn lựa phù hợp với yêu cầu của bài toán, ngoài vector ta có thể sử dụng danh sách kề hoặc các cấu trúc khác.

6. Bài 6: Bài toán: Bữa tiệc vui vẻ

Nguồn bài: <http://acm.timus.ru/problem.aspx?space=1&num=1039>

6.1. Đề bài.

Một công ty có N nhân viên (kể cả giám đốc, $1 \leq N \leq 2000$), công ty được tổ chức theo sơ đồ hình cây với: K là cha của L có nghĩa là K là cấp trên trực tiếp của L , ông giám đốc là gốc cây. Nhân viên thứ i (có tên là i) có độ vui vẻ h_i với $-128 \leq h_i \leq 127$.

Ông giám đốc muốn tổ chức một bữa tiệc có tổng độ vui vẻ của những người đến dự là lớn nhất và nếu có một nhân viên nào đó đi dự thì sẽ không có cấp trên trực tiếp của anh ta. Đương nhiên là ông giám đốc phải có mặt.

Yêu cầu: Bạn hãy giúp ông giám đốc tổ chức bữa tiệc sao cho độ vui vẻ là lớn nhất có thể.

Dữ liệu: Vào từ file văn bản **BUATIEC.INP**:

- Dòng đầu ghi số nguyên dương N ($1 \leq N \leq 2000$)
- Dòng thứ hai ghi N số h_1, h_2, \dots, h_n ($-128 \leq h_i \leq 127$)
- $N - 1$ dòng cuối cùng ghi 2 số L và K

Kết quả: Ghi ra file văn bản **BUATIEC.OUT**:

Một số S là tổng độ vui vẻ lớn nhất có thể nhận được.

Ví dụ:

BUATIEC.INP	BUATIEC.OUT
7 1 1 1 1 1 1 1 2 1 3 1 4 2 6 2 5 3 7 3	5

Ràng buộc:

- Có 50% số test tương ứng 50% số điểm có $N < 1000$
- 50% số test còn lại tương ứng 50% số điểm có $1000 \leq N < 2000$

❖ Xác định bài toán:

Input:

- Số nguyên N ($1 \leq N \leq 2000$)
- N số h_1, h_2, \dots, h_n ($-128 \leq h_i \leq 127$)
- $N - 1$ bộ 2 số L và K

Output: Một số S là tổng độ vui vẻ lớn nhất

6.2. Hướng dẫn giải thuật.

• Ở bài tập này, khi ta xét đến một đỉnh U thì ta sẽ quan tâm có chọn đỉnh U vào bữa tiệc hay không và khi chọn hay không chọn U thì sẽ ảnh hưởng thế nào tới việc chọn con và cha của nút U vào bữa tiệc.

• Từ nhận xét trên, ta sử dụng quy hoạch động như sau:

Sử dụng mảng F với ý nghĩa:

- $F[1][U]$ là độ vui vẻ lớn nhất để khi ta xét đỉnh U ta sẽ chọn U và một số đỉnh con V_i do U quản lý.
- $F[0][U]$ là độ vui vẻ lớn nhất để khi ta xét đỉnh U ta sẽ không chọn U và chọn một số đỉnh con V_i do U quản lý.
- *Bài toán con nhỏ nhất:* Bài toán con nhỏ nhất là tại các nút lá (Nút không có cấp dưới), khi đó $F[1][\text{lá}] = h[\text{lá}]$, $F[0][\text{lá}] = 0$.
- *Công thức truy hồi:*

Từ ý nghĩa mảng F trên ta thấy, nếu đã chọn U thì rõ ràng con của U sẽ không được chọn, còn nếu không chọn U thì con của U ta có thể tùy ý chọn. Vậy công thức truy hồi cho bài toán là :

- $F[1][U]$ bằng tổng tất cả $F[0][V_i] > 0$. (Vì nếu $F[0][V_i] < 0$ thì thêm vào U chỉ làm giảm đi độ vui vẻ).
- Tương tự $F[0][U]$ bằng tổng các $\max(F[1][V_i], F[0][V_i]) > 0$.

Do bắt buộc có giám đốc nên kết quả sẽ là $F[1][\text{Giám đốc}]$.

Ta sẽ sử dụng thuật toán duyệt Dfs để duyệt cây đồng thời tính luôn các giá trị của mảng F .

6.3. Chương trình.

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define Maxn 100005
const ll ma=-100000000;
vector<ll> gr[Maxn];
ll F[2][Maxn],Ress,n,h[Maxn];
void Dfs(ll u)
{
```

```

    ll v;
    F[0][u]=0;
    F[1][u]=h[u];
    forr(i,0,gr[u].size())
    {
        v=gr[u][i];
        Dfs(v);
        F[1][u]=max(F[1][u],F[0][v]+F[1][u]);
        F[0][u]=max(F[0][u],F[0][u]+max(F[0][v],F[1][v]));
    }
}
int main()
{
    ll u,v,boss,sl[Maxn];
    freopen("buatiec.inp","r",stdin);
    freopen("buatiec.out","w",stdout);
    cin>>n;
    forr(i,1,n+1) cin>>h[i];
    forr(i,1,n+1) sl[i]=0;
    forr(i,1,n)
    {
        cin>>v>>u;
        sl[v]++;
        gr[u].push_back(v);
    }
    forr(i,1,n+1) if (sl[i]==0) boss=i;
    Dfs(boss);
    cout<<F[1][boss];
}

```

6.4. Độ phức tạp.

- Do mỗi đỉnh ta chỉ duyệt 1 lần nên độ phức tạp sẽ là $O(n) * P$ với P là độ phức tạp khi tham chiếu tới các đỉnh con. Do ta cài bằng vector nên P xấp xỉ bằng 1.

Vậy, độ phức tạp của thuật toán là $O(n)$.

6.5. Test

<https://drive.google.com/file/d/1MywhnrNSnc7R4N8NPFyHCMmwEDxkFFUV/view?usp=sharing>

6.6. Cảm nhận

Về ý tưởng và cài đặt của bài toán này thì đã quá rõ ràng nhưng có một vấn đề nếu bạn bỏ qua thì sẽ dẫn đến kết quả sai. Đó chính là vấn đề ai là giám đốc, nếu vô tư như những bài trước coi 1 là giám đốc thì bạn đã phạm phải sai lầm nghiêm trọng. Để xác định ai là giám đốc bạn nên xem xét nó có cấp trên hay không.

7. Bài 7: Bài toán Tách cây

Nguồn bài: <https://codeforces.com/contest/461/problem/B>

7.1. Đề bài.

Cho một cây có n đỉnh. Có ít nhất một đỉnh màu đen và các đỉnh khác có màu trắng.

Xét một tập hợp bao gồm k cạnh ($0 \leq k < n$) của cây. Nếu bạn xóa các cạnh này khỏi cây, thì nó sẽ chia cây thành $(k + 1)$ phần. Lưu ý rằng mỗi phần sẽ là một cây có các đỉnh màu đen.

Yêu cầu: Hãy tìm số cách tách cây sao cho kết quả thu được là mỗi phần sẽ có chính xác một đỉnh màu đen? Kết quả tìm được chia dư cho $1000000007 (10^9 + 7)$.

Dữ liệu: Vào từ file văn bản **DIVIDE.INP**

- Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 10^5$) là số lượng đỉnh của cây.
- Dòng thứ hai chứa $n - 1$ số nguyên p_0, p_1, \dots, p_{n-2} ($0 \leq p_i \leq i$). Trong đó p_i có nghĩa là có một cạnh nối đỉnh $(i + 1)$ của cây và đỉnh p_i . Biết các đỉnh cây được đánh số từ 0 đến $n - 1$.
- Dòng thứ ba chứa n số nguyên x_0, x_1, \dots, x_{n-1} (x_i là 0 hoặc 1) mô tả màu của các đỉnh. Nếu x_i bằng 1, đỉnh i có màu đen. Nếu x_i bằng 0, đỉnh i có màu trắng.

Kết quả: Ghi ra file văn bản **DIVIDE.OUT**

- Một số nguyên duy nhất là số cách để tách cây, mà số này chia dư cho $1000000007 (10^9 + 7)$.

Ví dụ:

DIVIDE.INP	DIVIDE.OUT
3 0 0 0 1 1	2
10 0 1 2 1 4 4 4 0 8 0 0 0 1 0 1 1 0 0 1	27

❖ Xác định bài toán:

Input:

- Số nguyên n ($2 \leq n \leq 10^5$)
- $n - 1$ số nguyên p_0, p_1, \dots, p_{n-2} ($0 \leq p_i \leq i$).
- n số nguyên x_0, x_1, \dots, x_{n-1} (x_i là 0 hoặc 1)

Output: Số cách để tách cây

7.2. Hướng dẫn giải thuật

Ta có một số nhận xét từ bài toán như sau:

- Chỉ có một đường đi duy nhất giữa 2 đỉnh bất kỳ. Giả sử trên cây có x nút màu đen thì ta cần chia cây thành x vùng để mỗi vùng có đúng 1 nút đen, điều này tương đương với ta phải xóa đi $x - 1$ cạnh để hai nút đen bất kỳ không thể đi đến nhau.
 - Vậy khi ta đã quy ước chiều cho cây, khi xét đến 1 đỉnh ta sẽ quan tâm đỉnh đang xét đó có nằm ở một cây con đã có nút đen nào hay chưa.
- Từ các nhận xét trên, ta sử dụng quy hoạch động như sau:

Sử dụng mảng F với ý nghĩa:

- $F[1][U]$ là số cách để đỉnh U nằm trên 1 cây đã có nút đen với cây đó là cây con của cây có gốc là U (hay nói cách khác là cây chứa đỉnh U và các nút con do U quản lý tạo thành).
- $F[0][U]$ là số cách để đỉnh U nằm trên 1 cây đã không có nút đen với cây đó là cây con của cây có gốc là U (hay nói cách khác là cây chứa đỉnh U và các nút con do U quản lý tạo thành).

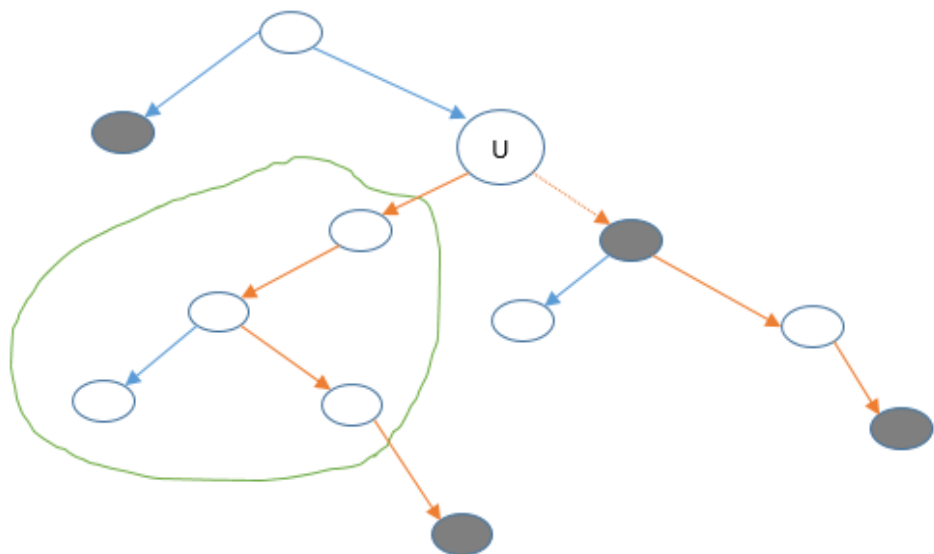
- Bài toán con nhỏ nhất:**

Bài toán con nhỏ nhất là ở các nút lá. Nếu là nút lá đen thì $F[1][\text{lá}] = 0$, $F[0][\text{lá}] = 1$. Còn đối với nút lá trắng thì ngược lại.

- Công thức truy hồi:**

- Đối với $F[0][U]$ ta rõ ràng thấy đối với nút đen thì $F[0][U] = 0$ vì rõ ràng không thể tạo cây như yêu cầu. Còn với nút U là trắng thì đương nhiên để tạo cây không có nút đen thì đỉnh U đang xét phải nối với các gốc cây con cũng không có nút đen hoặc nút U sẽ không nối với một số đỉnh con. Suy ra, tổng số cách ($F[0][U]$) chính là tổng số cách tạo các cây con không có nút đen có gốc là V_i ($F[0][V_i]$ với V_i là các con của U).

$$\Rightarrow F[0][U] = F'[0][U] * F[0][V_i] \quad (\text{Nối với cây trắng}) \\ + F'[0][U] * F[1][V_i] \quad (\text{Loại bỏ với cây có nút đen})$$



Hình 8: Nối nút U với cây con trắng hoặc loại bỏ cây con có nút đen.

- Còn đối với $F[1][U]$ thì sẽ phức tạp hơn. Nếu U là nút đen thì công việc của chúng ta là sẽ nối nó với các cây con V_i trắng ($F[0][V_i]$) hoặc chúng ta sẽ tách nút U ra khỏi cây con đen V_i ($F[1][V_i]$). Còn nếu nút U là trắng thì để tạo ra được cây đen bắt buộc ta phải nối nó với cây con V_i đã có một nút đen ($F[1][V_i]$)

⇒ Vậy công thức sẽ là:

$$F[1][u] = (F'[1][u] * F[1][v] + F'[1][u] * F[0][v] + F'[0][u] * F[1][v]).$$

Với các F' là số cách đã sẵn có trước khi ta tính V_i

7.3. Chương trình

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define Maxn 100015
const ll ma=-1000000000;
const ll Mo=1000000007;
vector<ll> gr[Maxn];
ll F[2][Maxn],Re[2][Maxn],n,Black[Maxn];
bool visited[Maxn];
void Dfs(ll u)
{
    ll v;
    visited[u]=false;
    F[0][u]=1-Black[u];
    F[1][u]=Black[u];
    forr(i,0,gr[u].size())
    {
        v=gr[u][i];
        if (visited[v])
        {
            Dfs(v);
            F[1][u]=(F[1][u]*F[1][v]+F[1][u]*F[0][v]+F[0][u]*F[1][v])%Mo;
            F[0][u]=(F[0][u]*(F[1][v]+F[0][v]))%Mo;
        }
    }
}
int main()
{
    freopen("divide.inp","r",stdin);
    freopen("divide.out","w",stdout);
    ll x;
    cin>>n;
    forr(i,0,n-1)
```

```

{
    cin >> x;
    gr[x].push_back(i+1);
    gr[i+1].push_back(x);
}
forr(i,0,n) cin >> Black[i];
forr(i,0,n) visited[i]=true;
Dfs(1);
cout << F[1][1] << "\n";
}

```

7.4. Độ phức tạp:

- Thuật toán được cài đặt giống với các bài trước. Ta sẽ sử dụng thuật toán duyệt Dfs để duyệt cây và tạo thứ tự cho cây, đồng thời tính luôn các giá trị của mảng F .
- Do mỗi đỉnh ta chỉ duyệt 1 lần nên độ phức tạp sẽ là $O(n) * P$ với P là độ phức tạp khi tham chiếu tới các đỉnh con. Do ta cài bằng vector nên P xấp xỉ bằng 1.

⇒ Độ phức tạp của thuật toán là $O(n)$.

7.5. Test.

<https://drive.google.com/file/d/1EYpmnwAYpDotjENCId9DSHz48YaJI2XT/view?usp=sharing>

7.6. Cảm nhận

- Khi ta xét đến một đỉnh U thì ngay từ đầu tiên khi ta chưa xét bất cứ nút con nào của nó thì $F[1][U]$ và $F[0][U]$ cũng đã có giá trị và giá trị phụ thuộc vào nó là nút đen hay nút trắng. (Ta phải tính F ban đầu cho nút).
- Đây là một bài toán nhiều trường hợp nếu phân tích bài toán không kỹ ta có thể bỏ sót các trường hợp đặc biệt của bài toán. Vậy để ra kết quả bài toán chính xác bạn cần chia nhỏ ra các bước phân tích: chia nút đen trắng, chia trường hợp nhỏ lẻ...

8. Bài 8: Bài toán Ánh sáng tối ưu

Nguồn bài: <https://www.spoj.com/problems/VOCV/>

8.1. Đề bài.

Thành phố Hà Nội có một mạng lưới các đường hai chiều với các thuộc tính sau:

- Không có nhiều hơn một đường giữa mỗi cặp nút giao.
- Mọi ngã ba được kết nối với mọi ngã ba khác trực tiếp qua đường hoặc qua các nút giao khác bằng một đường dẫn duy nhất.
- Khi đèn được đặt ở ngã ba, tất cả các đường phố gặp nhau tại ngã ba này cũng được thấp sáng.

Đèn chiếu sáng hợp lệ là một tập hợp các nút giao thông sao cho nếu đèn được đặt ở những vị trí này, tất cả các đường phố sẽ được thắp sáng. Một ánh sáng tối ưu là một đèn chiếu sáng hợp lệ sao cho nó chứa số lượng mỗi nút ít nhất.

Yêu cầu:

- Tìm số lượng đèn trong một ánh sáng tối ưu?
- Tìm tổng số ánh sáng tối ưu như vậy trong thành phố?

Dữ liệu: Vào từ file văn bản **VOCV.INP**

- Dòng đầu tiên của đầu vào chứa số nguyên dương $t \leq 20$, là số test.
- Mỗi test sẽ có các dữ liệu sau:
 - Dòng đầu chứa số nguyên dương $n \leq 100010$ biểu thị số lượng điểm nút trong mạng. Mỗi đường giao nhau được đánh số với một số nguyên duy nhất trong khoảng từ 1 đến n .
 - $n - 1$ dòng sau chứa 2 số nguyên (u, v) ($1 \leq u, v \leq n$) biểu thị rằng có một đường giữa ngã ba u và ngã ba v .

Kết quả: Ghi ra file văn bản **VOCV.OUT**

- Gồm t dòng, dòng thứ k tương ứng với mạng thứ k ($1 \leq k \leq t$).
- Dòng thứ k chứa hai số nguyên cách nhau bởi một khoảng trắng
 - Số nguyên đầu tiên trên dòng thứ k phải là số lượng đèn chiếu sáng tối ưu của mạng k .
 - Số nguyên thứ hai phải là $N \% 10007$, phần còn lại bằng số lượng đèn chiếu sáng tối ưu khi chia cho 10007.

Ví dụ:

VOCV.INP	VOCV.OUT
2	2 3
4	1 1
1 2	
2 3	
3 4	
3	
1 2	
1 3	

Ràng buộc:

- Có 20% số test tương ứng 20% số điểm có $N \leq 100$
- Có 40% số test tương ứng 40% số điểm có $100 < N < 5000$
- 40% số test còn lại tương ứng 40% số điểm có $5000 \leq N \leq 10^5$

❖ **Xác định bài toán:**

Input:

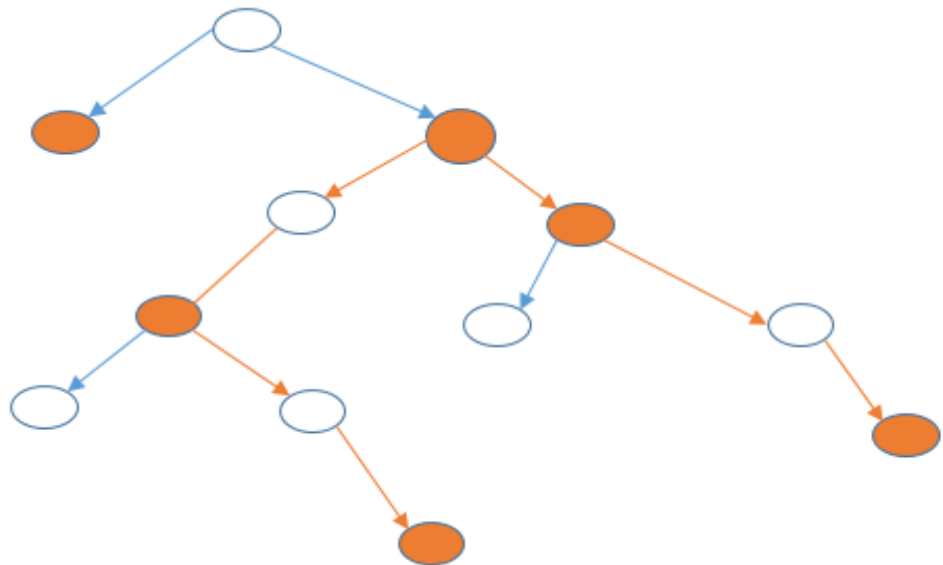
- Số nguyên n ($2 \leq n \leq 10^5$)
- $n - 1$ số nguyên p_0, p_1, \dots, p_{n-2} ($0 \leq p_i \leq i$).
- n số nguyên x_0, x_1, \dots, x_{n-1} (x_i là 0 hoặc 1)

Output: Gồm t dòng, dòng thứ k tương ứng với mạng thứ k ($1 \leq k \leq t$).

8.2. Hướng dẫn giải thuật.

- Dựa vào đề bài và các thuộc tính đầu ta rút ra các nhận xét sau:
 - Dựa vào thuộc tính một và hai mà đề bài đã cho ta thấy đồ thị có hình cây, trong đó chưa có sự định hướng ưu tiên thứ tự giữa các nút.
 - Để một đường phố được thấp sáng (Cạnh) thì có ít nhất một trong hai nút gia thông của cạnh đó phải được thấp sáng.

Vậy, để tạo ra một hệ thống đèn sáng hợp lệ thì hai nút nối trực tiếp nhau phải có ít nhất một nút được thấp sáng. Tức là khi quy ước nút cha và nút con thì nút con và nút cha không thể đồng thời đèn tắt. Hệ thống đèn sáng hợp lệ như hình 9 dưới đây:



Hình 9. Minh họa hệ thống đèn sáng hợp lệ

Từ các nhận xét trên ta sử dụng quy hoạch động để giải quyết bài toán như sau:

- Do cây là đồ thị vô hướng nên ta sẽ chọn một đỉnh làm đỉnh gốc rồi duyệt Dfs quy ước thứ tự cho các nút. Vậy khi nút U tắt đèn thì rõ ràng các nút con của U phải bật đèn. Còn nếu U bật đèn thì các nút con của U có thể bật tắt tùy ý.
- Các nút có thể tắt hoặc bật nên ta sẽ sử dụng mảng $F[2][n]$ với ý nghĩa: $F[0, U]$ là số lượng nút được bật đèn nhỏ nhất để hệ thống đèn do nút U quản lý là hợp lệ khi nút U không bật đèn. Còn $F[1, U]$ là số lượng nút được bật đèn nhỏ nhất để hệ thống đèn do nút U quản lý là hợp lệ khi nút U bật đèn.

- Bài toán con nhỏ nhất:

Bài toán con nhỏ nhất đó chính là ở các nút lá: $F[0][\text{lá}] = 0$, $F[1][\text{lá}] = 1$, $Re[0][\text{lá}] = 1$, $F[0][\text{lá}] = 1$.

- Công thức truy hồi:

- $F[0][U] = \sum F[1][v]$ với v là các nút con của U .
- $F[1][U] = \sum \min(F[1][v], F[0][v])$ với v là các nút con của U .

Để giải quyết câu hỏi có bao nhiêu cách thỏa mãn thì ta sẽ sử dụng thêm mảng $Re[2][n]$ để lưu trữ số cách với ý nghĩa $Re[0][U]$ là số cách hợp lệ để đèn U tắt và $F[0][U] \min$, $Re[1][u]$ là số các để đèn U bật và $F[1][U] \min$. Vậy công thức là:

- $Re[0][U] = \prod F[1][v]$ với v là các nút con của U .
- $Re[1][U] = \prod (F[1][v] \text{ hoặc } F[0][v])$ tương ứng với khi chọn tắt hoặc bật đèn v .

Vậy nghiệm của bài toán là $\min(F[0][1], F[1][1])$ và Re tương ứng.

8.3. Chương trình.

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define Maxn 100015
const ll ma=-1000000000;
const ll Mo=10007;
vector<ll> gr[Maxn];
ll F[2][Maxn],Re[2][Maxn],n;
bool visited[Maxn];
void Dfs(ll u)
{
    ll v;
    visited[u]=false;
    F[0][u]=0; F[1][u]=1;
    Re[1][u]=1; Re[0][u]=1;
    forr(i,0,gr[u].size())
    {
        v=gr[u][i];
        if (visited[v])
        {
            Dfs(v);
            F[0][u]=F[0][u]+F[1][v];
            Re[0][u]=(Re[0][u]*Re[1][v])%Mo;
            if (F[1][v]<F[0][v])
```

```

        {
            F[1][u]=F[1][u]+F[1][v];
            Re[1][u]=(Re[1][u]*Re[1][v])%Mo;
        }
        if (F[1][v]>F[0][v])
        {
            F[1][u]=F[1][u]+F[0][v];
            Re[1][u]=(Re[1][u]*Re[0][v])%Mo;
        }
        if (F[1][v]==F[0][v])
        {
            F[1][u]=F[1][u]+F[1][v];
            Re[1][u]=(Re[1][u]*(Re[1][v]+Re[0][v]))%Mo;
        }
    }
}
}
int main()
{
    freopen("VOCV.inp","r",stdin);
    freopen("VOCV.out","w",stdout);
    ll t,x,y;
    cin>>t;
    while (t>0)
    {
        t--;
        cin>>n;
        forr(i,0,n+1) gr[i].clear();
        forr(i,0,n-1)
        {
            cin>>x>>y;
            gr[x].push_back(y);
            gr[y].push_back(x);
        }
        forr(i,1,n+1) visited[i]=true;
        Dfs(1);
        if (F[1][1]>F[0][1]) cout<<F[0][1]<<" "<<Re[0][1]<<"\n";
        if (F[1][1]<F[0][1]) cout<<F[1][1]<<" "<<Re[1][1]<<"\n";
        if (F[1][1]==F[0][1])
            cout<<F[0][1]<<" "<<(Re[0][1]+Re[1][1])%Mo<<"\n";
    }
}

```

8.4. Độ phức tạp:

- Ta đã sử dụng duyệt Dfs để tạo thứ tự cho các nút đồng thời tính F , Re .
- Do mỗi đỉnh chỉ duyệt một lần nên độ phức tạp sẽ là $O(n)$ khi sử dụng vector để lưu trữ các cạnh.

8.5. Test.

<https://drive.google.com/file/d/1QrSK7vIBDOEZqvXIdsDHRxSXGj3K8qSf/view?usp=sharing>

8.6. Cảm nhận

- Ở việc tính mảng F thì rất đơn giản và dễ dàng nhưng khi tính với Re cần rất cẩn thận ở trường hợp đặc biệt khi mà ở đỉnh U bật đèn tức là tính $Re[1][U]$, cụ thể khi trường hợp tính $Re[1][U]$ mà có $F[1][v] = F[0][v]$ thì rõ ràng đỉnh v tắt hay bật đều đạt tối ưu nên số cách ở đây là $Re[0][v] + Re[1][v]$ chứ không phải chỉ là mỗi $Re[0][v]$ hay $Re[1][v]$ như các trường hợp khác.
- Tương tự khi lấy kết quả bài toán ở nút 1 thì ta cũng phải xem xét số cách bài toán trong các trường hợp đặt biệt.

9. Bài 9: Bài toán Di chuyển nhanh nhất

9.1. Đề bài.

Trong một vùng sinh thái có n hòn đảo đánh số từ 1 đến n . Ở k hòn đảo trong số này có các khu nhà dành cho khách du lịch. Các đảo được nối với nhau bởi đúng $n - 1$ cây cầu sao cho mọi đảo đều đi đến được với nhau. Với mỗi cây cầu, đều xác định được thời gian để một chiếc ô tô tải đi hết của nó (từ đảo này sang đảo kia). Hàng ngày Nam phải chuyển thực phẩm đến k khu nhà nghỉ bằng cách từ đất liền cập bến một đảo nào đó, sau đó dùng xe tải xuất phát từ đảo này đi hết tất cả k hòn đảo có khu nhà nghỉ. Anh ta dừng lại tại hòn đảo cuối cùng trong số k hòn đảo có nhà nghỉ và từ đây đi tàu thủy về đất liền. Nam muốn xác định với mỗi hòn đảo dùng làm nơi xuất phát thì tổng quãng đường lái xe tải tối thiểu là bao nhiêu?

Dữ liệu: Đọc dữ liệu từ file **DVTRUCK.INP**

- Dòng đầu tiên ghi hai số nguyên n, k ($1 < n \leq 5 \cdot 10^5, 1 < k < n$)
- $n - 1$ dòng tiếp theo, mỗi dòng chứa ba số nguyên A, B, C ($1 \leq A, B \leq N, 1 \leq C \leq 106$) thể hiện có một cầu nối đảo A với đảo B có độ dài C .
- k dòng cuối cùng, mỗi dòng ghi chỉ số của một hòn đảo có nhà nghỉ

Kết quả: Ghi ra file **DVTRUCK.OUT**

Gồm n dòng, dòng thứ i ghi quãng đường ngắn nhất mà Nam phải lái xe nếu như xuất phát từ đảo thứ i .

Ví dụ:

DVTRUCK.INP	DVTRUCK.OUT
5 2	5
2 5 1	3
2 4 1	7
1 2 2	2
1 3 2	2
4	
5	

Ràng buộc:

- Có 40% số test tương ứng 40% số điểm có $N \leq 1000$
- Có 20% số test tương ứng 20% số điểm có $1000 < N < 10^5$
- 40% số test còn lại tương ứng 40% số điểm có $10^5 \leq N < 5 \cdot 10^5$

❖ **Xác định bài toán:**

Input:

- Hai số nguyên n, k ($1 < n \leq 5 \cdot 10^5, 1 < k < n$)
- Các bộ số nguyên A, B, C ($1 \leq A, B \leq N, 1 \leq C \leq 10^6$)

Output:

- n số i ghi quãng đường ngắn nhất phải lái xe nếu xuất phát từ đảo thứ i

9.2. Hướng dẫn giải thuật.

Bài toán có thể diễn tả là với mỗi đỉnh, tìm hành trình đi đến k đỉnh được đánh dấu từ trước sao cho độ dài của hành trình là ngắn nhất. Dễ thấy độ dài hành trình ngắn nhất với xe xuất phát từ đỉnh u bằng $2 \times (\text{tổng khoảng cách từ } u \text{ đến } k \text{ điểm đánh dấu}) - (\text{độ dài đường đi dài nhất từ } u \text{ đến một đỉnh được đánh dấu})$.

- Tính tổng khoảng cách từ 1 đỉnh đến các đỉnh được đánh dấu
 - Gọi $T1[u]$ = tổng khoảng cách từ u đến các đỉnh được đánh dấu trong cây con gốc u
 - Gọi $T2[u]$ = tổng khoảng cách từ u đến các đỉnh được đánh dấu ngoài cây con gốc u .

Ta có các công thức ($w = prev[u]$):

$$T1[u] = \sum \{T1[v] + S[v] * L(u,v) : v \in con(u)\}$$

$$T2[u] = T2[w] + (T1[w] - T1[u] - S[u] * L(w,u)) + (k - S[u] * L(w,u))$$

Trong đó $S[u]$ = số lượng đỉnh đánh dấu trong cây con gốc u .

Tổng khoảng cách từ các điểm đánh dấu đến u được tính bởi $T1[u] + T2[u]$

- Tính đường đi dài nhất từ u đến một đỉnh được đánh dấu

- Đặt $f[u]$, $f2[u]$ là độ dài đường đi dài nhất, đường đi dài nhì từ một đỉnh đánh dấu thuộc cây con gốc u đến u ta có:

$$f[u] = f2[u] = -\infty \text{ nếu } S[u] = 0$$

$f[u] = \max(0, f[v] + L(u,v) : v \in \text{con}(u))$ trong trường hợp ngược lại, $f2[u]$ được tính kèm khi tính $f[u]$.

- Đặt $g[u]$ độ dài đường đi dài nhất từ u đến đỉnh đánh dấu không thuộc cây con gốc u .

Ta có $g[u] = -\infty$ nếu $S[u] = k$ ngược lại:

$$g[u] = \max(0, g[w] + L(w,u), f[w] \text{ (hoặc } f2[w]) + L(w,u))$$

Độ dài đường đi dài nhất từ u đến một đỉnh được đánh dấu bằng $\max(f[u], g[u])$

Vậy với mỗi đỉnh u đáp số là: $2 \times (T1[u] + T2[u]) - \max(g[u], f[u])$

Dễ thấy tất cả các công thức trên có thể tính sau khi có được duyệt DFS.

9.3. Chương trình.

```
#include <bits/stdc++.h>
#define pii pair<int, int>
#define PB push_back
#define MP make_pair
#define ll long long
#define F first
#define S second
#define maxc 1000000007
#define maxn 500005
using namespace std;
int n, k, par[maxn], tp[maxn], fa[maxn], fb[maxn], id = 0, s[maxn], fal[maxn];
ll ta[maxn], tb[maxn];
bool luu[maxn];
vector<pii> ke[maxn];
void DFS(int u)
{
    if (luu[u]) s[u]++;
    tp[++id] = u;
    for (auto p : ke[u])
    {
        int v = p.F; int w = p.S;
        if (par[v]) continue;
        par[v] = u;
        DFS(v);
        s[u] += s[v];
    }
}

int main()
{
```

```

    ios_base::sync_with_stdio(0); cin.tie(NULL); cout.tie(NULL);
    freopen("dvtruck.inp", "r", stdin);
    freopen("dvtruck.out", "w", stdout);
    cin >> n >> k;
    for (int u, v, c, i = 1; i < n; i++)
    {
        cin >> u >> v >> c;
        ke[u].PB(MP(v, c));
        ke[v].PB(MP(u, c));
    }
    for (int d, i = 1; i <= k; i++)
    {
        cin >> d;
        luu[d] = 1;
    }
    DFS(1);
    for (int i = n; i >= 1; i--)
    {
        int u = tp[i];
        for (auto p : ke[u])
        {
            int v = p.F;
            int c = p.S;
            if (par[v] != u) continue;
            if (s[v])
            {
                ta[u] += ta[v] + c;
                if (fa[u] < fa[v] + c)
                {
                    fa1[u] = fa[u];
                    fa[u] = fa[v] + c;
                }
                else if (fa1[u] < fa[v] + c) fa1[u] = fa[u] + c;
            }
        }
    }
    for (int i = 1; i <= n; i++)
    {
        int u = tp[i];
        for (auto p : ke[u])
        {
            int v = p.F;
            int c = p.S;
            if (par[v] != u) continue;

```

```

    if (s[v] == k) continue;
    fb[v] = fb[u] + c;
    if (fa[u] == fb[u] + c) fb[v] = max(fb[v], fa[u] + c);
    else fb[v] = max(fb[v], fa[u] + c);
    if (s[v]) tb[v] = tb[u] + ta[u] - ta[v];
    else tb[v] = tb[u] + ta[u] - ta[v] + c;
}
}
for (int i = 1; i <= n; i++)
    cout << 2ll*(ta[i]+tb[i]) - max(fa[i], fb[i]) << '\n';
return 0;
}

```

9.4. Độ phức tạp.

- Thuật toán có độ phức tạp là $O(n)$

9.5. Test.

<https://drive.google.com/file/d/1zO6-KvvZzbUkqHlN6DKdAxRwGrpYgro3/view?usp=sharing>

9.6. Cảm nhận

Khác với một số bài toán đã trình bày, code mẫu này được cài đặt lưu địa chỉ, giúp dễ dàng hơn trong việc truy vết và dễ dàng sửa lỗi nếu gặp sai sót.

10. Bài 10: Bài toán Liên bang Berland

Nguồn bài: <https://codeforces.com/contest/440/problem/D>

10.1. Đề bài.

Gần đây, để dễ dàng quản lí, liên bang Berland muốn được chia thành các tiểu bang riêng biệt. Hơn nữa, họ yêu cầu rằng có một tiểu bang bao gồm chính xác k thị trấn.

Hiện tại, Berland có n thị trấn, một số thị trấn được nối với nhau bằng đường hai chiều. Berland chỉ có $n - 1$ đường. Bạn có thể đến bất kỳ thành phố nào từ thủ đô, nghĩa là mạng lưới đường bộ tạo thành hình cây.

Yêu cầu: Đưa ra kế hoạch chia liên bang thành các tiểu bang như sau:

- Trong mỗi tiểu bang, đều có các con đường kết nối các thị trấn. Nghĩa là ta có thể đi từ một thị trấn đến một thị trấn bất kì khác.
- Có một tiểu bang bao gồm chính xác k thành phố.
- Số lượng đường kết nối các tiểu bang khác nhau là tối thiểu.

Dữ liệu: Vào từ file văn bản **BERLAND.INP**

- Dòng đầu chứa số nguyên n, k ($1 \leq k \leq n \leq 400$).

- $n - 1$ dòng tiếp theo, mỗi dòng chứa các cặp số nguyên (u, v) ($1 \leq u, v \leq N, u \neq v$) mô tả một con đường ở Berland. Giả sử rằng các thị trấn được đánh số từ 1 đến n .

Kết quả: Ghi ra file văn bản **BERLAND.OUT**

- Dòng đầu tiên in số đường tối thiểu t . Sau đó in ra t con đường được giữ lại. Mỗi con đường thì ghi theo thứ tự mà chúng được nhập. Nếu có nhiều cách chia, có thể in bất kì cách chia nào.
- Nếu không có cách chia nào thỏa mãn thì in ra 0.

Ví dụ:

BERLAND.INP	BERLAND.OUT
5 3 1 2 1 3 1 4 1 5	2 3 4
1 1	0

Ràng buộc:

- Có 50% số test tương ứng 50% số điểm có $N \leq 100$
- 50% số test còn lại tương ứng 50% số điểm có $100 < N \leq 400$

❖ **Xác định bài toán:**

Input:

- Hai số nguyên n, k ($1 \leq k \leq n \leq 400$).
- $n - 1$ cặp số nguyên (u, v) ($1 \leq u, v \leq N, u \neq v$)

Output:

- Số đường tối thiểu t .
- t con đường thỏa mãn

10.2. Hướng dẫn thuật toán

- Qua đề bài toán ta thấy:
 - Do sau khi chia thành các vùng thì có ít nhất một cây con có đúng k đỉnh. Nên nếu ta chọn các đỉnh lần lượt làm gốc cho cây con khi phân chia cây ban đầu ra thì kết quả bài toán sẽ nằm ở một trong số các đỉnh.
 - Giả sử ta đang xét đỉnh U và nếu U là kết quả bài toán tức là đỉnh U chính là gốc của cây con có đúng k đỉnh thì rõ ràng ta cần loại bỏ cạnh nối của đỉnh U và cha của nó, đồng thời loại bỏ một vài đỉnh do U quản lý để cây con gốc U sẽ có đúng k đỉnh.

- Từ các nhận xét trên ta sử dụng quy hoạch động như sau:

Xây dựng các cây con có độ lớn từ bé đến lớn với mỗi đỉnh gốc. Cụ thể hơn sẽ là xây dựng $F[U][i]$ có ý nghĩa là số cạnh bé nhất phải xóa để đỉnh U đang xét có i nút. Hay là cây con gốc U có độ lớn là i (Số nút của cây). Với ý nghĩa như thế ta sẽ xây dựng i từ lớn đến bé ($N \rightarrow 0$), lý do phải làm thế thì bạn có thể xem bài toán quy hoạch động cái túi để hiểu thêm.

- *Bài toán con nhỏ nhất:*

Bài toán con nhỏ nhất là bài toán tại mỗi đỉnh khi ngắt hết các cạnh, đó là $F[U][1]=0$.

- *Công thức truy hồi:*

Giả sử đỉnh U khi ta đã xét đến đỉnh con thứ V_x của U và ta đang có F tương ứng, giờ ta xét đỉnh con kế tiếp V_x là V_y thì V_y sẽ có thể đóng góp cho cây con gốc U từ 0 đến size (V_y) với size (V_y) là số nút mà V_y quản lý. Từ đó ta sẽ có công thức là $F[U][i] = \text{Min}(F[U][i], F[U][j] + F[V_y][i - j])$ với $j(0, i)$

- Vấn đề đầu tiên đã được giải quyết rất dễ dàng với mảng F . Nhưng như hầu hết các bài toán quy hoạch động khác, việc tìm ra kết quả bài toán đôi khi không phức tạp bằng truy vết. Để truy vết được bài này ta phải sử dụng thêm dữ liệu để lưu quá trình chọn lựa khi thực hiện tính mảng F .

10.3. Chương trình

```
#include <bits/stdc++.h>
#define ll int
#define pb push_back
#define forr(i,a,b) for(ll i=a;i<b;i++)
#define ford(i,a,b) for(ll i=a;i>=b;i--)
#define MaxN 405
const ll ma=1e9;
using namespace std;
vector<ll> gr[MaxN];
struct data{
    ll con,Dg;
};
ll n,K,x,y,i,F[MaxN*2][MaxN],vt[2*MaxN],cs,size[MaxN];
ll ans,ID[MaxN][MaxN],Cha[MaxN],ress,kq,cd[2*MaxN];
data tv[2*MaxN][MaxN];
bool lay[MaxN];
void dfs(ll x){
    ll z,y;
    forr(i,0,gr[x].size())
        if(gr[x][i]!=Cha[x])
        {
            Cha[gr[x][i]]=x;
            dfs(gr[x][i]);
        }
}
```

```

cs++;
F[cs][1]=0;
size[x]=1;
forr(i,0,gr[x].size())
    if(gr[x][i]!=Cha[x])
    {
        y=gr[x][i];
        cs++;
        cd[cs]=x;
        forr(j,1,size[x]+1)
            forr(k,0,size[y]+1)
            {
                if(k==0)z=1;
                else z=F[vt[y]][k];
                if (F[cs][j+k]>F[cs-1][j]+z)
                {
                    F[cs][j+k]=F[cs-1][j]+z;
                    tv[cs][j+k].con=y;
                    tv[cs][j+k].Dg=k;
                }
            }
            size[x]+=size[y];
    }
vt[x]=cs;
cd[cs]=x;
}
void truyvet(ll dd,ll k)
{
    ll u,t;
    if (k==0) return;
    lay[cd[dd]]=true;
    if (k==1) return;
    while (k>1)
    {
        truyvet(vt[tv[dd][k].con],tv[dd][k].Dg);
        k=k-tv[dd][k].Dg;
        dd--;
    }
}
void in(ll u)
{
    forr(i,0,gr[u].size())
        if (gr[u][i]!=Cha[u])
        {
            if (lay[gr[u][i]]) in(gr[u][i]);
            else cout<<ID[u][gr[u][i]]<<" ";
        }
}
int main()
{
    freopen("berland.inp","r",stdin);

```

```

    freopen("berland.out", "w", stdout);
    ll z;
    cin >> n >> K;
    forr(i, 1, n)
    {
        cin >> x >> y;
        gr[x].pb(y);
        gr[y].pb(x);
        ID[x][y] = ID[y][x] = i;
    }
    memset(F, 3, sizeof(F));
    dfs(1);
    ans = ma;
    forr(i, 1, n + 1)
    {
        if (i == 1) z = 0; else z = 1;
        if (F[vt[i]][K] + z < ans)
        {
            ans = F[vt[i]][K] + z;
            kq = i;
        }
    }
    forr(i, 1, n + 1) lay[i] = false;
    cout << ans << "\n";
    lay[kq] = true;
    if (kq != 1) cout << ID[kq][Cha[kq]] << " ";
    truyvet(vt[kq], K);
    in(kq);
}

```

10.4. Độ phức tạp

Ta đã sử dụng thuật toán duyệt Dfs để tạo thứ tự cho cây đồng thời tính luôn các giá trị của mảng F . Do khi tính toán ngay chính mảng $F[U][i]$ bị thay đổi giá trị nên thay vì chạy ngược i phức tạp ta sẽ phân lớp F ra cho mỗi lần tính toán. (Tức là một đỉnh U sẽ có nhiều lớp F). Độ phức tạp: $O(n^3)$.

10.5. Test.

<https://drive.google.com/file/d/1r5hNO8yj3WzbxY0s1t3-WpQq9qPYbmHs/view?usp=sharing>

10.6. Cảm nhận

Đây là một bài toán rất thú vị về quy hoạch động trên cây. Để giải quyết được bài toán bạn phải nắm thật chắc thuật toán quy hoạch động trên cây, bài toán cái túi và kết hợp nó trên cây một cách hiệu quả để giải quyết được bài toán.

Vấn đề khó khăn nhất của bài toán là truy vết, để truy vết được thì quá trình tính toán F , mỗi lần thay đổi F ta dùng mảng vt để lưu lại các thay đổi và sử dụng thêm các mảng phụ để lưu trữ các thông tin cần thiết. Ta sẽ đệ quy để truy vết, cụ thể là phần code ở trên đã tách truy vết thành 2 phần là phần tìm các đỉnh thuộc cây con k (Truyvet()) và phần tìm các đỉnh bị xóa (In()).

11. Hướng dẫn thuật toán một số bài toán khác

11.1. Bài toán Tâm của cây

Đề bài:

Cho một cây, một đỉnh được gọi là tâm của cây nếu khoảng cách xa nhất từ đỉnh đó đến một đỉnh trong cây là nhỏ nhất.

Yêu cầu: Cho một cây. Hãy xác định các tâm của nó

Dữ liệu: Đọc dữ liệu từ file **CTREE.INP**

- Dòng 1: Ghi số nguyên dương $n \leq 10^5$ là số đỉnh của cây
- $n - 1$ dòng tiếp theo, mỗi dòng ghi hai số u, v mô tả một cạnh của cây

Kết quả: Ghi ra file **CTREE.OUT**

- Dòng 1: Ghi c là số tâm của cây
- Dòng 2: Ghi c số liệt kê tăng dần là số hiệu các đỉnh là tâm của cây

Ví dụ:

CTREE.INP	CTREE.OUT
3	1
1 2	2
2 3	

Hướng dẫn thuật toán:

- Để giải quyết bài toán này thì nhiệm vụ đầu tiên của chúng ta là phải tìm được: đối với mỗi đỉnh thì khoảng xa nhất tới các đỉnh khác là bao nhiêu. Điều này được giải quyết rất dễ từ ý tưởng của bài PT07Z (Bạn đọc tự xem lại).
- Sau khi giải quyết xong nhiệm vụ này thì ta chỉ cần tìm min trong số các đỉnh.

11.2. Bài toán Cắt cây

Nguồn bài: <https://www.hackerrank.com/challenges/tree-pruning/problem>

Đề bài:

Cho một đồ thị dạng cây gồm n đỉnh đánh số từ 1 đến n , đỉnh i có trọng số a_i . Cho k lần cắt cây (mỗi lần xóa một cạnh hiện đang có) ta sẽ có một rừng gồm $k + 1$ cây. Trọng số của một cây được định nghĩa bằng tổng trọng số các đỉnh có trong cây đó. Bài toán yêu cầu hãy tìm cách cắt cây sao cho trọng số lớn nhất của các cây sinh ra là nhỏ nhất.

Dữ liệu: Đọc dữ liệu từ file **TRCUT.INP**

- Dòng đầu tiên gồm hai số n, k ($1 \leq k < n \leq 5 \cdot 10^5$)

- Dòng thứ hai ghi n số a_1, a_2, \dots, a_n là trọng số các đỉnh ($0 \leq a_i \leq 10^3$)
- $n - 1$ dòng tiếp theo, mỗi dòng gồm hai số u, v mô tả một cạnh của cây

Kết quả: Ghi ra file **TRCUT.OUT**

Một số nguyên duy nhất là trọng số của cây có trọng số lớn nhất trong phương án tối ưu

Ví dụ:

TRCUT.INP	TRCUT.OUT
8 2	20
7 4 3 8 5 7 5 4	
2 1	
3 1	
4 3	
5 2	
6 1	
7 6	
8 1	

Ràng buộc:

- Subtask 1 (20%): $n \leq 20$
- Subtask 2 (20%): $n \leq 200$
- Subtask 3 (20%): $n \leq 1000, a_i = 1 \forall i = 1, 2, \dots, n$
- Subtask 4 (40%): $n \leq 5 \cdot 10^5$

Hướng dẫn thuật toán:

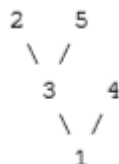
- Đối với Subtask 1, 2 thì ta có thể giải quyết dễ dàng bằng các cách khác nhau.
- Đối với Subtask 3 ta có thể áp dụng ý tưởng thuật toán của *Bài 10. Liên bang Berland* đã hướng dẫn ở trên để giải quyết.
- Đối với Subtask 4: Thay vì tìm cách tách thành k nhóm rồi kiểm tra trọng số lớn nhất của các cây con thì ta sẽ làm ngược lại: Chọn trọng số rồi kiểm tra xem có chia được k nhóm thỏa mãn các cây con có trọng số không lớn hơn trọng số đã chọn không. Trọng số lớn nhất thỏa mãn chính là kết quả của bài toán. Để làm như vậy ta sẽ dùng chặt nhị phân trọng số lớn nhất và điều kiện kiểm tra là có chia thành được k nhóm phù hợp với trọng số không.
- Để kiểm tra với một trọng số có phù hợp không thì ta sẽ dùng Dfs để thăm các đỉnh và kiểm tra cây con khi Dfs(U) chia được tối thiểu bao nhiêu nhóm. Bắt đầu từ bài toán nhỏ các lá sau đó cứ thêm các nút cha vào cho đến khi quá trọng số thì tách nhóm. Để làm được điều này ta sẽ dùng hàng đợi queue.

11.3. Bài toán Cây táo

Nguồn bài: <http://acm.timus.ru/problem.aspx?space=1&num=1018>

Đề bài:

Một cây táo có hình dạng giống như một cây nhị phân tức là từ một cành bất kỳ có đúng hai cành con xuất phát từ điểm cuối của nó (hoặc là không có cành con nào). Chúng ta đánh số các điểm cuối (đầu nút) của các cành lần lượt là $1, 2, \dots, N$. Giả sử 1 luôn được dùng để đánh số cho gốc của cây táo. Ví dụ dưới đây là một cây táo có 5 điểm nút (4 cành):



Theo kinh nghiệm, muốn chất lượng quả táo ngon thì không nên để quá nhiều cành trên cây táo. Tốt nhất là mỗi cây chỉ để lại Q cành. Bạn hãy viết chương trình bỏ đi một số cành của cây táo sao cho số cành còn lại là Q và số quả táo trên các cành giữ lại là lớn nhất.

Dữ liệu: Đọc dữ liệu từ file **APPLE.INP**

- Dòng đầu tiên chứa hai số nguyên dương N và Q ($1 \leq Q \leq N$; $1 < N \leq 100$). N xác định số điểm nút trên cây táo. Q xác định số cành cần giữ lại.
- $N - 1$ dòng tiếp theo mô tả các cành táo mỗi cành được mô tả bởi ba số nguyên: hai số nguyên đầu tiên là số hiệu các nút ở hai đầu nút của cành và số nguyên thứ ba mô tả số lượng táo có trên cành đó. Không có cành táo nào chứa quá 30000 quả táo.

Kết quả: Ghi ra file **APPLE.OUT**

Ghi một số nguyên duy nhất là số táo lớn nhất được giữ lại trên Q cành (lưu ý rằng 1 luôn là gốc của cây)

Ví dụ:

APPLE.INP	APPLE.OUT
5 2	21
1 3 1	
1 4 10	
2 3 20	
5 3 20	

Hướng dẫn thuật toán:

- Bài toán này tương tự *Bài 10. Liên bang Berland*, tuy nhiên ở bài toán này, gốc của cây đã được quy ước.
- Có thể phát biểu lại bài toán là: Tìm số quả bị loại bỏ min khi loại bỏ $N - Q - 1$ cành. Ta sẽ dùng $F[u, t]$ là lượng quả min khi cây con ở đỉnh u bị loại t cành. Vậy kết quả bài toán chính là $Sum - F[1, N - Q - 1]$.
- Vậy với mỗi cành nối đỉnh U với đỉnh V là con U thì ta quan tâm có bỏ cành đó hay không và khi loại bỏ thì sẽ ra sao. Bạn đọc suy nghĩ và tìm ra công thức truy hồi cho $F[u, t]$ (Xem lại *Bài 10. Liên bang Berland* để vận dụng).

11.4. Bài toán Đuổi bò

Nguồn bài: <http://usaco.org/index.php?page=viewproblem2&cpid=213>

Đề bài:

Đã tới giờ cho uống nước ở nông trại của nông dân John (FJ), nhưng các con bò lại đang bỏ chạy! FJ muốn tập trung chúng lại, và ông ta cần sự giúp đỡ của bạn.

Nông trại của FJ là một dãy gồm có N ($1 \leq N \leq 200000$) bãi cỏ được đánh số từ 1.. N và được nối bằng $N - 1$ con đường hai chiều. Chuồng bò nằm ở bãi cỏ thứ nhất, và từ bãi thứ nhất, ta có thể đi đến tất cả các bãi cỏ còn lại. Những con bò của FJ đang ở bãi cỏ của chúng vào sáng nay, nhưng không ai biết chúng đã đi đâu cho tới bây giờ. FJ biết rằng những con bò chỉ muốn chạy xa khỏi nhà chứa, nhưng cũng rất lười nên không thể chạy một đoạn đường có độ dài lớn hơn L (theo hướng xa nhà chuồng). Với mỗi bãi cỏ, FJ muốn biết có bao nhiêu bãi cỏ mà những con bò bắt đầu tại bãi cỏ đó có thể dừng chân.

Lưu ý: Số dạng 64 bit (trong Pascal là int64, trong C/C++ là long long, và trong Java là long) cần dùng để lưu các khoảng cách.

Dữ liệu: Vào từ file văn bản **RUNAWAY.INP**

- Dòng đầu tiên ghi hai số nguyên N, L ($1 \leq N \leq 200000, 1 \leq L \leq 10^{18}$)
- $N-1$ dòng tiếp theo, dòng thứ i ghi hai số p_i, l_i với p_i là bãi cỏ đầu tiên trên đường đi ngắn nhất từ $i + 1$ đến nhà chứa, l_i là khoảng cách con đường đó ($1 \leq p_i < i + 1, 1 \leq l_i \leq 10^{12}$)

Kết quả: Ghi ra file văn bản **RUNAWAY.OUT**

Gồm N dòng, dòng thứ i là số lượng bãi cỏ có thể đi tới được từ i bằng cách rời xa nhà chứa (bãi cỏ 1) với độ dài không quá L

Ví dụ:

RUNAWAY.INP	RUNAWAY.OUT
4 5	3
1 4	2
2 3	1
1 5	1

Giải thích:

- Con bò ở bãi cỏ 1 (chuồng) có thể ở bãi 1, 2, 4
- Con bò ở bãi cỏ 2 có thể ở bãi 2, 3
- Các con bò ở bãi 3 và 4 không thể đi xa hơn nữa khỏi bãi 1 (chuồng) nên nó chỉ có thể ở đó

Hướng dẫn thuật toán

Yêu cầu của bài này là: Xét một cây gốc T , với mỗi nút v , tìm và đưa ra số lượng nút con của v trong một khoảng cách nhất định L . Nếu chúng ta gán cho mỗi nút v độ sâu là D , ta cần đếm số lượng nút con có độ sâu nhiều nhất là $D + L$.

Cách đơn giản và ngắn gọn nhất để thực hiện vấn đề này là 'lật lại' những gì chúng ta cần tính toán. Đối với mỗi đỉnh, chúng ta tìm xem cây nằm cách đỉnh đó bao xa. Điều này có thể được thực hiện bằng cách duyệt DFS và tìm kiếm nhị phân.

Sau đó, ta cần chuyển đổi lại thành thông tin về nút con. Với mỗi một nút, ta tìm nút cha thứ L rồi dùng cộng dồn từ nút đó lên nút cha thứ L . Mỗi nút có tổng là bao nhiêu thì đó chính là đáp án.

11.5. Bài toán Khách sạn đặc biệt

Nguồn bài: <http://codeforces.com/contest/580/problem/C>

Đề bài:

Vào ngày sinh nhật, Minh quyết định đến một nhà hàng ở FLC để tổ chức tiệc kỉ niệm. Minh cũng sống tại một phòng của khách sạn FLC.

Có thể coi khu FLC là một cây có n đỉnh với gốc ở đỉnh 1. Đỉnh 1 chứa phòng của Minh. Các nút lá chứa các nhà hàng. Và một số đỉnh của cây có chứa các con mèo. Minh thì rất sợ mèo. Vì vậy Minh sẽ không thể đến nhà hàng nếu trên đường từ nhà của Minh đến nhà hàng chứa nhiều hơn m đỉnh liên tiếp gặp mèo.

Nhiệm vụ của bạn là giúp Minh đếm số lượng nhà hàng nơi Minh có thể đến.

Dữ liệu: Vào từ file **HOTEL.INP**

- Dòng đầu tiên chứa hai số nguyên n và m ($2 \leq n \leq 10^5$, $1 \leq m \leq n$) là số đỉnh của cây và số đỉnh tối đa liên tiếp gặp mèo mà Minh vẫn chịu được.
- Dòng thứ hai chứa n số nguyên a_1, a_2, \dots, a_n , trong đó mỗi a_i bằng 0 (nếu đỉnh i không có mèo) hoặc bằng 1 (nếu đỉnh i có một con mèo).
- $n - 1$ dòng tiếp theo chứa các cạnh của cây theo định dạng " $x_i y_i$ " (không có dấu ngoặc kép) ($1 \leq x_i, y_i \leq n, x_i \neq y_i$), trong đó x_i và y_i là các đỉnh của cây, được nối với nhau bởi một cạnh. Tập hợp các cạnh đã cho tạo thành một cây.

Kết quả: Ghi ra file **HOTEL.OUT**

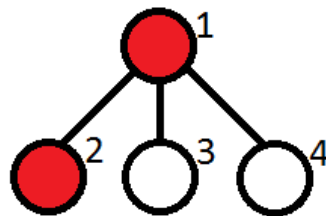
- Một số nguyên duy nhất là số lượng lá riêng biệt của cây trên đường đi từ nhà của Minh chứa nhiều nhất m đỉnh liên tiếp gặp mèo.

Ví dụ:

HOTEL.INP	HOTEL.OUT
4 1 1 1 0 0 1 2 1 3 1 4	2
7 1 1 0 1 1 0 0 0 1 2 1 3 2 4 2 5 3 6 3 7	2

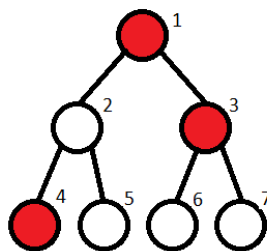
Giải thích:

- Ví dụ thứ nhất:



Các đỉnh chứa mèo được đánh dấu màu đỏ. Các nhà hàng ở các đỉnh 2, 3, 4. Minh chỉ không thể đến nhà hàng nằm ở đỉnh 2.

- Ví dụ thứ 2:



Các nhà hàng nằm ở các đỉnh 4, 5, 6, 7. Minh không thể đến nhà hàng 6, 7.

Hướng dẫn thuật toán

- Ta sẽ duyệt Dfs(u, k) với u là đỉnh đang xét và k là số đỉnh liên tiếp gặp mèo cho đến u , vậy khi thăm đỉnh v là các đỉnh con của u : nếu v có mèo thì ta sẽ thăm Dfs($v, k + 1$) còn không thì sẽ là Dfs($v, 0$). Nếu $k > m$ thì ta không cần thăm con của u . Đáp án là số lượng đỉnh thăm được. Bạn cũng có thể loại bỏ k trong Dfs bằng lưu một mảng Children[] ở ngoài.
- Độ phức tạp: $O(n)$.

11.6. Một số bài toán khác

<https://www.iarcs.org.in/inoi/online-study-material/problems/rivers.php>
<https://www.iarcs.org.in/inoi/online-study-material/problems/coffee-shop.php>
<https://www.iarcs.org.in/inoi/online-study-material/problems/mobiles-apio.php>
<https://www.iarcs.org.in/inoi/online-study-material/problems/catering-contracts.php>
<https://www.iarcs.org.in/inoi/online-study-material/problems/catering-contracts-2pi.php>
<https://www.hackerearth.com/challenges/competitive/march-clash-15/algorithm/counting-on-tree-1/description/>
<http://ntucoder.net/Problem/Details/5655>
<http://vn.spoj.com/problems/CTREE/>
<https://vn.spoj.com/problems/PTREE/>
<http://vn.spoj.com/problems/BRIDGES>
<http://vn.spoj.com/problems/STONE1>
<http://vn.spoj.com/problems/NTTREE>
<http://vn.spoj.com/problems/MTREE>
<http://vn.spoj.com/problems/NTPFECT>
<https://vn.spoj.com/problems/ITREE/>
<https://vn.spoj.com/problems/HBTLCA/>
<https://vn.spoj.com/problems/VOTREE/>
<https://vn.spoj.com/problems/UPGRANET/>

.....

PHẦN III: KẾT LUẬN

Trong chuyên đề, tôi đã trình bày về các phương pháp quy hoạch động, một số lý thuyết về cây, và ứng dụng các kiến thức đó để giải một số bài toán về quy hoạch động trên cây.

Sau khi áp dụng phương pháp quy hoạch động trên cây, tôi thấy nó mang lại hiệu quả rất rõ rệt. Nó cho tôi góc nhìn tổng quan về một số bài toán tối ưu hóa trên cây, để khi gặp dạng toán này, tôi có hướng làm, phương pháp để thực hiện. Hầu hết học sinh đều đánh giá đây là một nội dung hay, cần được củng cố, luyện tập để nâng cao kiến thức, kỹ năng của học sinh trong các kỳ thi học sinh giỏi hiện nay.

Do thời gian còn hạn chế và kiến thức còn chưa được sâu, rộng nên chắc chắn chuyên đề còn nhiều thiếu sót. Tôi rất mong quý thầy cô đồng nghiệp đóng góp ý kiến để chuyên đề được hoàn thiện hơn.

TÀI LIỆU THAM KHẢO

1. Tài liệu giáo khoa chuyên Tin tập 1, 2, 3.
2. Một số vấn đề đáng chú ý trong môn Tin học
3. Website: <http://vn.spoj.com/>
4. Website: <http://codeforces.com/>
5. Website: <https://www.hackerrank.com>
6. Website: <http://usaco.org>