HỘI THẢO KHOA HỌC
CÁC TRƯỜNG THPT CHUYÊN
KHU VỰC DUYÊN HẢI VÀ ĐỒNG BẰNG BẮC BỘ
NĂM 2019

Môn: Tin học

Chủ đề: CÂY VÀ QUY HOẠCH ĐỘNG TRÊN CÂY

Tháng 8/2019

MỤC LỤC

QUY HOẠCH ĐỘNG TRÊN CÂY

0. Mở đầu

Ta thấy rằng khi duyệt đồ thị dạng cây sẽ xuất hiện bản chất của bài toán được thiết kế bởi thuật toán Quy hoạch động. Do đó, đối với đồ thị dạng cây, ta có một lớp bài toán mà ta thường gọi là Quy hoạch động trên cây (hay Dynamic programming on the tree).

Chuyên đề này, tôi xin phép chia sẻ một số bài toán như vậy trong quá trình dạy đội tuyển HSG môn Tin học lớp 10 trong hè chuẩn bị lên lớp 11.

Học sinh cần có một số kiến thức để đảm bảo học được chuyên đề này là: Kiến thức về phương pháp Quy hoạch động; Đồ thị, Duyệt đồ thị theo chiều sâu (DFS); kĩ thuật bảng thưa để giải bài toán LCA.

Khi học sinh nắm vững dạng toán này, chúng ta có thể mở rộng ra thành các dạng bài toán trên đồ thị vô hướng tổng quát, ở đó, mỗi thành phần song liên thông được coi như một đỉnh của cây, hoặc thu gọn chỉ xét các cạnh cầu tương ứng là các cạnh của cây trong rừng,...

Sau khi dạy cho học sinh quen các dạng bài về cây như thế này, chúng ta có thể dễ dang hướng dẫn học sinh tiếp cận các kĩ thuật cao hơn để giải các bài toán liên quan đến cây như Heavy light decomposition, Centroid decomposition,... Thực thế tôi cũng đã áp dụng chuyên đề này đối với học sinh của mình, kết quả thu được là rất khả quan, học sinh tiếp cận tốt một số kĩ thuật khó trên cây.

1. Một số khái niệm, kiến thức cơ bản

1.1. Phương pháp quy hoạch động

Như chúng ta đã biết, một bài toán có thể giải được bằng phương pháp quy hoạch động cần đảm bảo 2 đặc điểm nổi bật sau:

- Có tính chất của các bài toán con gối nhau (*overlapping subproblem*): Có thể chia nhỏ một bài toán lớn thành các bài toán con.

- Có cấu trúc con tối ưu (*optimal substructure*): Kết hợp lời giải của các bài toán con ta được lời giải của bài toán lớn hơn.

1.2. Đồ thị dạng cây

Theo định lý Daisy Chain, gọi $T = (V, E)$ là đồ thị vô hướng có n đỉnh, khi đó các mệnh đề sau là tương đương:

- T là cây

- T không chứa chu trình đơn và có $N - 1$ cạnh.

- T liên thông và mỗi cạnh của nó đều là cầu.

- Giữa 2 đỉnh bất kì của T đều tồn tại một đường đi đơn.

- T không chứa chu trình, nhưng hễ cứ thêm vào một cạnh ta lại thu được một chu trình.

- T liên thông vào có $N - 1$ cạnh.

Từ đây ta sẽ gọi cây gốc u là $T[u]$.

1.3. Duyệt đồ thị

Cách cài đặt đơn giản nhất sử dụng kĩ thuật đệ quy để duyệt đồ thị theo chiều sâu, mô hình cài đặt duyệt cây sẽ như sau:

```
void dfs(int u, int parent)
{
  <Khởi tạo thông tin cho đỉnh u>
  for(int v:adj[u])
    if (v!=parent)
     {
       dfs(v,u);
       <Cập nhật lại thông tin cho đỉnh u>
     }
  <Cập nhật lại thông tin cho đỉnh u>
}
```

Cách cài đặt trên dùng để quy hoạch động ngược từ dưới lên, để tổng hợp thông tin từ đỉnh con lên đỉnh cha. Thao tác này khá đơn giản nhưng thể hiện bản chất của Quy hoạch động.

Thông qua duyệt DFS, ta có thể tính toán, tổng hợp các thông tin cần thiết từ đỉnh con đến đỉnh cha hoặc ngược lại. Hay có thể chia nhỏ bài toán, rồi kết hợp các bài toán nhỏ lại để tạo ra bài toán lớn hơn. Mỗi đỉnh được duyệt không quá 1 lần, mỗi cạnh cũng duyệt không quá 1 lần, đo đó độ phức tạp của duyệt cây $T[1]$ luôn chỉ là $O(N)$.

Có một số bài tập trong chuyên đề này cần tổng hợp thông tin xuôi từ đỉnh cha xuống đỉnh con hay gọi là quy hoạch xuôi, phức tạp hơn, cơ bản sẽ làm theo cách là khi cập nhật ngược con lên cha những thông tin gì thì bây giờ sẽ loại bỏ những thông tin đó đi, sau đó cập nhật thông tin từ cha xuống con, ta căn cứ vào tình huống cụ thể để phân tích.

1.4. Tổ tiên chung gần nhất

Tổ tiên chung gần nhất (LCA) là khái niệm trong lý thuyết đồ thị và khoa học máy tính. Cho cây T có gốc và N đỉnh. Tổ tiên chung gần nhất của đỉnh u và v là đỉnh thấp nhất trong cây T mà nhận u, v làm hậu duệ (con, cháu). Có thể coi một đỉnh là con của chính nó – Wikipedia

2. Một số bài tập áp dụng

2.1. Bài tập 1: Nhánh có tổng lớn nhất 1

2.1.1. Đề bài

Cho đồ thị dạng cây $T[1]$, gồm N đỉnh, các đỉnh được đánh số từ 1 đến N, mỗi đỉnh i được gán một số nguyên dương a_i, hỏi đường đi có tổng các số ghi trên các đỉnh từ gốc cây xuống một đỉnh bất kì là bao nhiêu?

Dữ liệu vào:

Dòng đầu tiên là số N là số đỉnh của đồ thị.

Dòng tiếp theo ghi N số nguyên dương $a_1, a_2, \dots a_N$ là các số được gán với N đỉnh theo thứ tự.

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i với $0 \le p_i \le N$; $p_1 = 0$ vì đỉnh 1 là gốc cây ban đầu.

Kết quả ra: Một số duy nhất là đường đi có tổng lớn nhất.

Ràng buộc: $N \le 2.10^5, a_i \le 10^9$.

Ví dụ:

Summax1.inp	**Summax1.out**
14 3 2 1 10 1 3 9 1 5 3 4 5 9 8 0 1 1 1 2 2 3 4 4 4 5 5 7 7	22

2.1.2. Phân tích bài toán

Thuật toán 1: Duyệt trâu, độ phức tạp $O(N^2)$

Duyệt DFS lần 1 để tìm thứ tự cha con của các đỉnh

Với mỗi đỉnh không phải là gốc (khác đỉnh 1), ta duyệt ngược lên cha của nó đến khi gặp đỉnh gốc.

Lấy max của tất cả các đường đi như vậy.

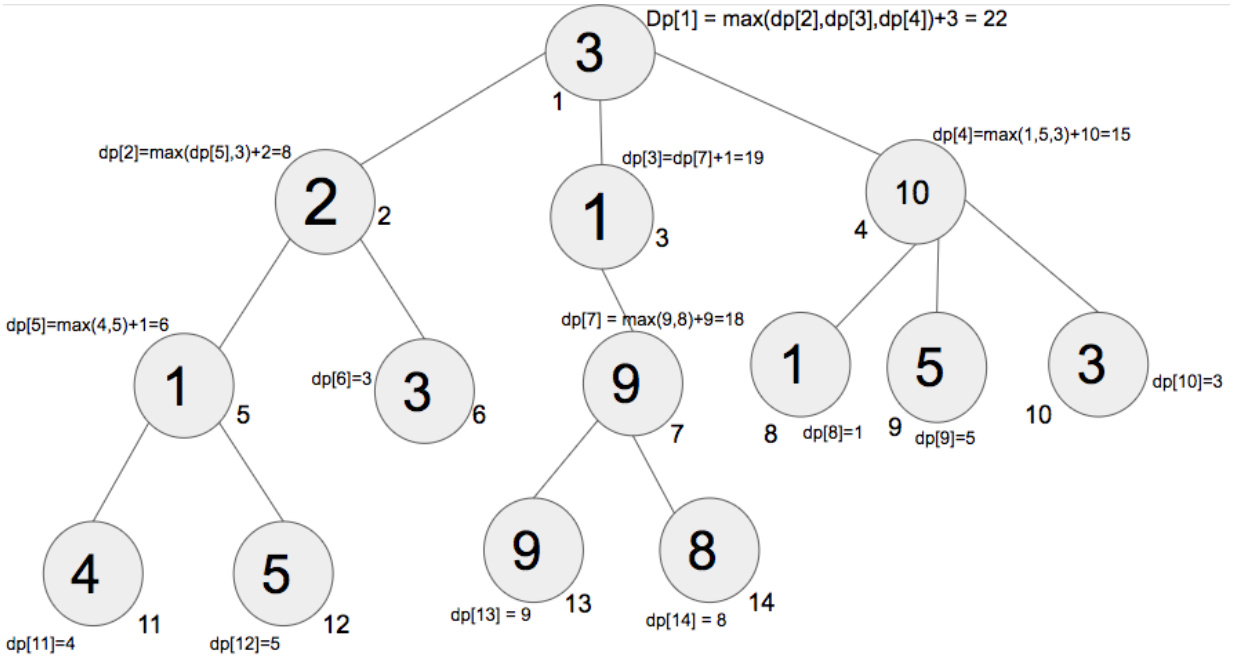
Thuật toán 2: Quy hoạch động, độ phức tạp $O(N)$.

Gọi $dp[u]$ là tổng lớn nhất thu được từ đỉnh i xuống các đỉnh con của nó. Khi đó, ta có:

$$\begin{cases} dp[u] = a[u] \text{ với } u \text{ là lá} \\ dp[u] = \max(a[u] + dp[v]) \text{ với mọi } v \text{ là con của } u \end{cases}$$

Dễ dàng tính được các giá trị $dp[.]$ trong quá trình duyệt DFS cây $T[1]$

Hãy quan sát hình vẽ minh họa bên dưới:



Độ phức tạp thuật toán đúng bằng độ phức tạp duyệt DFS của đồ thị. Ở đây là: $O(n)$.

Bạn hoàn toàn có thể cập nhật ngược lại theo cách ở trên, tức là cập nhật tổng từ gốc đến các đỉnh bên dưới, sau đó tìm max. Độ phức tạp vẫn như trên.

2.1.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define N 200005
#define task "summax1"
#define int long long
using namespace std;
int n,a[N],dp[N];
vector<int> adj[N];
void dfs(int u, int parent) {
  dp[u]=a[u]; //khoi tao thong tin
  for(int v:adj[u]) {
    if (v!=parent) {
      dfs(v,u);
      dp[u]=max(dp[v]+a[u],dp[u]); //cap nhat thong tin
    }
  }
}
signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  freopen(task".inp", "r", stdin);
  freopen(task".out","w",stdout);
  cin>>n;
  for(int i=1; i<=n; i++)
    cin>>a[i];
  for(int i=1; i<=n; i++) {
    int u;
    cin>>u;
    if(u==0)
      continue;
    adj[u].push_back(i);
    adj[i].push_back(u);
  }
  dfs(1,0);
  cout<<dp[1];
}
```

2.1.4. Test:

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.2. Bài tập 2: Tổng lớn nhất trên cây 2

2.2.1. Đề bài

Cho đồ thị dạng cây $T[1]$ gồm N đỉnh, các đỉnh được đánh số từ 1 đến N, mỗi đỉnh i được gán một số nguyên dương a_i. Ta có thể chọn nhiều đỉnh trên cây, tuy nhiên không được chọn 2 đỉnh kề nhau. Hỏi với cách chọn như vậy thì tổng các số lớn nhất có thể nhận được là bao nhiêu?

Dữ liệu vào:

Dòng đầu tiên là số N là số đỉnh của đồ thị.

Dòng tiếp theo ghi N số nguyên dương $a_1, a_2, \dots a_N$ là các số được gán với N đỉnh theo thứ tự.

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i, $p_i = 0$ thì đỉnh i là đỉnh gốc, còn lại $1 \le p_i \le N$.

Kết quả ra: Một số duy nhất là đường đi có tổng lớn nhất.

Ràng buộc: $N \le 2.10^5, a_i \le 10^9$.

Ví dụ:

Summax2.inp	Summax2.out
14 3 2 1 10 1 3 9 1 5 3 4 5 9 8 0 1 1 1 2 2 3 4 4 4 5 5 7 7	41

2.2.2. Phân tích bài toán

Thuật toán 1: Duyệt trâu. $O(2^N)$, xét tất cả các đỉnh với hai trạng thái là chọn hoặc không chọn.

Thuật toán 2: Quy hoạch động. $O(N)$

Rõ ràng bài toán này khá giống bài toán cho một dãy số $a_1, a_2, \dots a_N$, chọn các số không liên tiếp sao cho tổng lớn nhất có thể. Khi đó, gọi $dp[i]$ là tổng lớn nhất khi chọn i số đầu tiên. Ta có

$$dp[i] = \max(dp[i-1], dp[i-2] + a[i])$$

Với trường hợp bài toán cho là đồ thị dạng cây, thì ta gọi $dp[u]$ là giá trị lớn nhất khi xét cây con $T[u]$ của cây ban đầu. Có hai khả năng là chọn hoặc không chọn đỉnh u do đó ta cần xây dựng 2 mảng $dp0[.]$ và $dp1[.]$ như sau:

$$\begin{cases} dp1[u] = a_u + \sum_{v \text{ là con } u} dp0[v] & \text{(chọn } u\text{)} \\ dp0[u] = \sum_{v \text{ là con } u} \max\{dp1[v], dp0[v]\} & \text{(không chọn } u\text{)} \end{cases}$$

Kết quả của bài toán là: $ans = max(dp0[root], dp1[root])$

Độ phức tạp thuật toán đúng bằng độ phức tạp duyệt DFS của đồ thị. Ở đây là: $O(N)$.

2.2.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define N 200005
#define task "summax2"
#define int long long
using namespace std;
int n,root,a[N],dp0[N],dp1[N];
vector<int> adj[N];
void dfs(int u, int parent) {
  dp0[u]=0; //khoi tao thong tin
  dp1[u]=a[u];
  for(int v:adj[u])
    if(v!=parent) {
      dfs(v,u);
      dp0[u]+=max(dp0[v],dp1[v]);
      dp1[u]+=dp0[v]; //cap nhat lai thong tin
    }
}
signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  freopen(task".inp", "r", stdin);
  freopen(task".out","w",stdout);
  cin>>n;
  for(int i=1; i<=n; i++)
    cin>>a[i];
  for(int i=1; i<=n; i++) {
    int p;
    cin>>p;
    adj[p].push_back(i);
    adj[i].push_back(p);
  }
  root=adj[0][0];
  dfs(root,0);
  cout<<max(dp0[root],dp1[root]);
}
```

2.2.4. Test:

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.3. Bài 3: Dán tranh

2.3.1. Đề bài

Để trang trí căn phòng của mình, T3N quyết định mua về một bức tranh thật đẹp, do là người yêu thiên nhiên nên bức trang T3N chọn có dạng đồ thị cây có N đỉnh, các đỉnh được đánh số từ 1 đến N. Để dán bức tranh lên tường, T3N cần cho keo dán vào các đỉnh của cây, mỗi đỉnh i mất chi phí a_i. Để đảm bảo bức tranh hình cây được dán chắc chắn thì mỗi cạnh của cây đều phải được dán ít nhất tại một đầu. Hãy giúp T3N tính chi phí tối thiểu để dán được bức tranh của mình lên tường nhé!

Dữ liệu vào:

Dòng đầu tiên là số N là số đỉnh của đồ thị.

Dòng tiếp theo ghi N số nguyên dương $a_1, a_2, \dots a_N$ là các số được gán với N đỉnh theo thứ tự.

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i, $p_i = 0$ thì đỉnh i là đỉnh gốc, còn lại $1 \le p_i \le N$.

Kết quả ra: Một số duy nhất là chi phí dán tranh nhỏ nhất có thể.

Ràng buộc: $N \le 2.10^5, a_i \le 10^9$.

Ví dụ:

Summax3.inp	**Summax3.out**
14 3 2 1 10 1 3 9 1 5 3 4 5 9 8 0 1 1 1 2 2 3 4 4 4 5 5 7 7	23

2.3.2. Phân tích bài toán

Thuật toán: Quy hoạch động $O(N)$.

Bài toán này tương tự bài toán 2.2. Ta gọi $dp1[u], dp0[u]$ là chi phí nhỏ nhất khi chọn hoặc không chọn đỉnh u. Khi đó:

$$\begin{cases} dp1[u] = a_u + \sum_{v \text{ là con } u} dp0[v] & (\text{chọn } u) \\ dp0[u] = \sum_{v \text{ là con } u} \min\{dp1[v], dp0[v]\} & (\text{không chọn } u) \end{cases}$$

Kết quả là: $\min(dp0[root], dp1[root])$

2.3.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define N 100005
#define task "summax3"
#define int long long
using namespace std;
int n,root,a[N],dp0[N],dp1[N];
vector<int> adj[N];
void dfs(int u, int parent) {
```

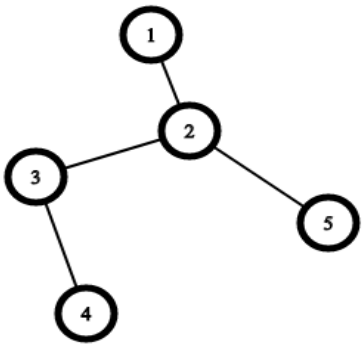
```
    dp0[u]=0; //khoi tao thong tin
    dp1[u]=a[u];
    for(int v:adj[u])
      if(v!=parent) {
        dfs(v,u);
        dp1[u]+=min(dp0[v],dp1[v]);
        dp0[u]+=dp1[v]; //cap nhat lai thong tin
      }
  }
  signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    freopen(task".inp", "r", stdin);
    freopen(task".out","w",stdout);
    cin>>n;
    for(int i=1; i<=n; i++)
      cin>>a[i];
    for(int i=1; i<=n; i++) {
      int p;
      cin>>p;
      adj[p].push_back(i);
      adj[i].push_back(p);
    }
    root=adj[0][0];
    dfs(root,0);
    cout<<min(dp0[root],dp1[root]);
  }
```

2.3.4. Test:

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.4. Bài 4: Khoảng cách K trên cây

2.4.1. Đề bài: Cho một cây T[1] có N đỉnh, các đỉnh được đánh số từ 1 đến N, và một số K, hãy đếm xem có bao nhiêu cặp đỉnh (u,v) mà có khoảng cách đúng bằng K. Khoảng cách được tính bằng số cạnh. Chú ý: cặp (u,v) và cặp (v, u) là như nhau.



Dữ liệu vào: Dòng đầu tiên có 2 số N và K ($1 \leq N \leq 50000, 1 \leq K \leq 200$)

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i, $p_1 = 0$ vì đỉnh 1 là đỉnh gốc, còn lại $1 \leq p_i \leq N$.

Kết quả ra: Một số duy nhất số cặp đỉnh có khoảng cách K.

Ví dụ:

DISTK.inp	DISTK.out
5 2 0 1 2 3 2	4

2.4.2. Phân tích bài toán

Thuật toán 1: Duyệt trâu DFS cho từng đỉnh không đặt cận $O(N^2)$

Thuật toán 2: Duyệt trâu DFS cho từng đỉnh, có đặt cận $O(N.K)$

Thuật toán 3: Quy hoạch động $O(N.K)$

Gọi $dp[u][i]$ là số đỉnh có khoảng cách i đến u trong cây T[u],

Mỗi khi duyệt xong đỉnh v là con của u, thì kết quả được cập nhật:

$$ans+=\sum_{i=0}^{k-1} dp[u][i]*dp[v][k-i-1]$$

Sau đó ta cập nhật lại thông tin đỉnh u:

$$dp[u][i]=\sum_{v\in adj[u]} dp[v][i-1]$$

2.4.3. Chương trình minh họa

```
#include<bits/stdc++.h>
using namespace std;
int n,k;
vector<int> adj[50005];
long long ans,dp[50005][205];
void dfs(int u,int p) {
  dp[u][0]=1;
  for(int v:adj[u])
    if(v!=p) {
      dfs(v,u);
      for(int i=0; i<k; i++)
        ans+=dp[u][i]*dp[v][k-i-1];
      for(int i=0; i<k; i++)
        dp[u][i+1]+=dp[v][i];
    }
}
int main() {
  freopen("DISTK.inp","r",stdin);
  freopen("DISTK.out","w",stdout);
  scanf("%d%d",&n,&k);
  int p;
  scanf("%d",&p);
  for(int i=2; i<=n; i++) {
```

```
        scanf("%d",&p);
        adj[p].push_back(i);
        adj[i].push_back(p);
      }
      dfs(1,0);
      printf("%I64d\n",ans);
      return 0;
    }
```

2.4.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.5. Bài 5: Đường kính của cây

2.5.1. Đề bài:

Cho một cây T có N đỉnh, hãy tìm khoảng cách xa nhất giữa 2 cặp đỉnh bất kì trên cây. Khoảng cách (u, v) được xác định bằng số cạnh trên đường đi (u, v).

Dữ liệu vào:

Dòng đầu tiên là số N là số đỉnh của đồ thị.

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i, $p_i = 0$ thì đỉnh i là đỉnh gốc ban đầu, còn lại $1 \leq p_i \leq N$.

Kết quả ra: Một số duy nhất là độ dài đường đi dài nhất.

Ràng buộc: $N \leq 2.10^5$.

Ví dụ:

Diameter.inp	**Diameter.out**
14 0 1 1 1 2 2 3 4 4 4 5 5 7 7	6
5 4 1 1 0 4	3

2.5.2. Phân tích bài toán

Thuật toán 1: Duyệt trâu DFS cho từng đỉnh, tính khoảng cách từ gốc đến đỉnh xa nhất. Độ phức tạp $O(N^2)$

Thuật toán 2: Ta chọn một đỉnh bắt đầu bất kì u, tìm đỉnh v xa u nhất sử dụng DFS hoặc BFS, rồi lại từ v tìm đỉnh w xa nó nhất. Khoảng cách giữa v và w là khoảng cách xa nhất giữa 2 cặp đỉnh của cây.

Để làm việc đó, ta xây dựng mảng độ sâu, đặt là $dp[.]$.

Độ phức tạp vẫn là $O(N)$.

2.5.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define N 200005
```

```
#define task "diameter"
using namespace std;
int n, root,umax, dmax;
vector<int> adj[N];
void dfs(int u, int parent, int depth) {
  if(dmax<depth) {
    umax=u;
    dmax=depth;
  }
  for(int v:adj[u])
    if(v!=parent)
      dfs(v,u,depth+1);
}
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  freopen(task".inp","r",stdin);
  freopen(task".out","w",stdout);
  cin>>n;
  for(int i=1; i<=n; i++) {
    int p;
    cin>>p;
    if(p==0) {
      root=i;
      continue;
    }
    adj[p].push_back(i);
    adj[i].push_back(p);
  }
  dfs(root,0,0);
  dmax=0;
  dfs(umax,0,0);
  cout<<dmax;
}
```

2.5.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

Nhận xét: Trong các bài trên, chúng ta đều sử dụng thao tác tổng hợp thông tin từ đỉnh con lên đỉnh cha của cây T.

Trong **02 bài tiếp theo** ta sẽ thay đổi mô hình duyệt, ngoài việc tổng hợp thông tin từ dưới lên, chúng ta bổ sung thêm thao tác cập nhật ngược từ cha về con trên cây ban đầu.

2.6. Bài 6: Tô màu cho cây

2.6.1. Đề bài

Cho đồ thị dạng cây T gồm N đỉnh, các đỉnh được đánh số từ 1 đến N. Đầu tiên, tất cả các đỉnh đều có màu trắng. Lần đầu tiên, bạn được chọn bất kì một đỉnh sau đó tô nó thành màu đen, các lượt chơi tiếp theo bạn phải chọn một đỉnh kề với đỉnh màu đen và tô nó thành màu đen. Trò chơi kết thúc khi tất cả các đỉnh đều được tô màu đen.

Mỗi lượt chơi, bạn nhận được số điểm bằng số đỉnh trắng liên thông với đỉnh đã chọn (tính cả đỉnh được chọn). Sau khi cộng điểm thì các cạnh liên thuộc với đỉnh màu đen bị xóa khỏi cây. Hỏi tổng số điểm lớn nhất bạn có thể nhận được là bao nhiêu?

Dữ liệu vào:

Dòng đầu tiên là số N là số đỉnh của cây.

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i, $p_i = 0$ thì đỉnh i là đỉnh gốc ban đầu, còn lại $1 \leq p_i \leq N$.

Kết quả ra: Một số duy nhất là tổng số điểm lớn nhất.

Ràng buộc: $N \leq 2.10^5$.

Ví dụ:

Coloring.inp	**Coloring.out**
14 0 1 1 1 2 2 3 4 4 4 5 5 7 7	67
5 0 1 1 2 2	14

2.6.2. Phân tích bài toán

Thuật toán 1: Duyệt trâu DFS cho từng đỉnh, dùng mô hình DFS như trong phần lí thuyết đã trình bày để tính tổng số điểm. Độ phức tạp $O(N^2)$.

Thuật toán 2: Cải tiến thuật toán 1, chỉ DFS một lần để tính cây với gốc ban đầu, sau đó DFS thêm một lần duy nhất để cập nhật thông tin xuôi từ cha xuống con. Độ phức tạp $O(N)$.

Nhận xét: Rõ ràng kết quả của bài toán chỉ phụ thuộc vào việc chọn đỉnh tô màu đầu tiên (tạm gọi là đỉnh gốc). Mỗi khi chọn gốc khác nhau ta sẽ được một kết quả khác nhau. Do đó ta có N cách chọn gốc, nếu xử lý theo cách như ở trên thì ta mất $O(N^2)$.

Tuy nhiên ở đây, ta sẽ cập nhật ngược lại từ đỉnh cha về đỉnh con bằng cách xử lý lại thông tin mà đỉnh con cập nhật lên cha trong O(1). Khi này thì độ phức tạp thuật toán chỉ là $O(N)$.

Bước 1: Tính thông tin với gốc bất kì như mô hình DFS đã nói ở trên.

Bước 2: Cập nhật lại thông tin đỉnh con đi từ gốc đã chọn. Để cập nhật thì ta phải làm cách nào đó bỏ đi thông tin của con cập nhật lên cha lúc cập nhật ngược và cập nhật phần còn lại của cha cho con khi cập nhật xuôi.

2.6.3. Chương trình minh họa

```
#include <bits/stdc++.h>
#define N 200005
#define int long long
#define task "Coloring"
using namespace std;
int sz[N],n,dp[N];
vector<int> adj[N];
void dfs(int u, int p) {
  sz[u]=1;
  for(int v:adj[u]) {
    if(v==p)
      continue;
    dfs(v,u);
    dp[u]+=dp[v];
    sz[u]+=sz[v];
  }
  dp[u]+=sz[u];
}

void fix(int u, int p) {
  if(p) {
    //cap nhat lai thong tin cua con
    dp[u]=dp[u]+(dp[p]-dp[u]-sz[u])+(n-sz[u]);
  }
  for(int v : adj[u])
    if(v != p)
      fix(v, u);
}
signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  freopen(task".inp","r",stdin);
  freopen(task".out","w",stdout);
  cin>>n;
  for(int i=1; i<=n; i++) {
    int p;
    cin>>p;
    adj[p].push_back(i);
    adj[i].push_back(p);
  }
  dfs(1,0);  //tinh thong tin voi goc la 1
```

```
    fix(1,0); //tinh lai thong tin cac nut con
    cout << *max_element(dp+1,dp+1+n) << endl;
    return 0;
}
```

2.6.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.7. Bài 7: Tổng lớn nhất trên cây 4

2.7.1. Đề bài

Cho đồ thị dạng cây T gồm N đỉnh, các đỉnh được đánh số từ 1 đến N, mỗi đỉnh v được gán một số nguyên dương a_v. Gọi $dist(x,y)$ là khoảng cách giữa 2 đỉnh x, y, khoảng cách được tính bằng số cạnh trên đường đi đơn từ x đến y. Mỗi khi chọn một đỉnh v bất kì làm gốc thì ta thu được tổng giá trị của cây là $\sum_{i=1}^{N} dist(i,v)*a_i$. Hãy tìm giá trị lớn nhất có thể của cây ban đầu.

Dữ liệu vào:

Dòng đầu tiên là số N là số đỉnh của đồ thị.

Dòng tiếp theo ghi N số nguyên dương $a_1, a_2, \dots a_N$ là các số được gán với N đỉnh theo thứ tự.

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i, $p_i = 0$ thì đỉnh i là đỉnh gốc ban đầu, còn lại $1 \le p_i \le N$.

Kết quả ra: Một số duy nhất là giá trị lớn nhất có thể của cây.

Ràng buộc: $N \le 2.10^5, 1 \le a_i \le 2.10^5$.

Ví dụ:

Summax4.inp	**Summax4.out**
14 3 2 1 10 1 3 9 1 5 3 4 5 9 8 0 1 1 1 2 2 3 4 4 4 5 5 7 7	269
8 9 4 1 7 10 1 6 5 0 1 2 1 1 5 5 5	121

2.7.2. Phân tích bài toán

Thuật toán 1: Duyệt trâu $O(N^2)$. Duyệt từng đỉnh để tính giá trị của cây với cách chọn đỉnh đó làm gốc.

Thuật toán 2: Cải tiến thuật toán 1 tương tự bài Coloring.

Bài toán này chỉ khác chút ít bài toán trên nên tôi không đi phân tích nữa. Thiết nghĩ đưa thêm vào đây với mong muốn giúp học sinh có bài để làm quen dạng bài này.

2.7.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define N 200005
#define task "summax4"
#define int long long
using namespace std;
int a[N],n,root, sum, dp[N],sz[N];
vector<int> adj[N];
void dfs(int u, int p) {
  sz[u]=a[u];
  dp[u]=0;
  for(int v:adj[u]) {
    if(v!=p) {
      dfs(v,u);
      sz[u]=sz[u]+sz[v];
      dp[u]=dp[u]+dp[v]+sz[v];
    }
  }
}
void fix(int u, int p) {
  if(p)
    dp[u]=dp[u]+(dp[p]-dp[u]-sz[u])+(sum-sz[u]);
  for(int v:adj[u])
    if(v!=p)
      fix(v,u);
}
signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  freopen(task".inp", "r", stdin);
  freopen(task".out","w",stdout);
  cin>>n;
  if(n<=1) {
    cout<<0;
    return 0;
  }
  for(int i=1; i<=n; i++) {
    cin>>a[i];
    sum+=a[i];
  }
  for(int i=1; i<=n; i++) {
    int p;
```

```
      cin>>p;
      if(p==0) {
        root=i;
        continue;
      }
      adj[p].push_back(i);
      adj[i].push_back(p);
    }
    dfs(root,0);
    cal(root,0);
    cout<<*max_element(dp,dp+1+n);
  }
```

2.7.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.8. Bài 8: Khỉ con học nhảy

2.8.1. Đề bài

Có một chú khỉ con đang học nhảy từ cành nọ sang cành kia tại cái cây chú ưa thích. Cây có N đỉnh được đánh số từ 1 đến N, có N-1 cạnh và chú có thể nhảy giữa các đỉnh có khoảng cách không quá K (khoảng cách được tính là số cạnh trên đường đi ngắn nhất giữa 2 đỉnh).

Gọi $f(s,t)$ là số bước nhảy ít nhất để chú nhảy từ đỉnh s đến đỉnh t.

Hãy tính giá trị biểu thức: $\sum_{1\le s<t\le N} f(s,t)$

Dữ liệu vào:

Dòng đầu tiên là số N là số đỉnh của cây và số K.

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i với $p_i \neq 0, p_i = 0$ thì đỉnh i là đỉnh gốc ban đầu.

Kết quả ra: Một số duy nhất là độ dài đường đi dài nhất.

Ràng buộc: $N \le 2.10^5, 1 \le K \le 10$.

Ví dụ:

Jumping.inp	Jumping.out
6 2 0 1 1 2 2 4	20
14 5 0 1 1 1 2 2 3 4 4 4 5 5 7 7	95

2.8.2. Phân tích bài toán

Thuật toán 1: Duyệt trâu $O(N^2)$. Duyệt trâu DFS cho từng đỉnh làm gốc, tính số bước nhảy từ nó đến tất cả các đỉnh.

Thuật toán 2: Quy hoạch động $O(N.k^2)$

Trước hết, ta xét cần phải tính **S** là tổng các khoảng cách giữa 2 đỉnh (s,t) bất kì trên cây.

Để tính được **S**, ta cần biết mỗi một cạnh tham gia vào đường đi của bao nhiêu cặp đỉnh. Mỗi cạnh được nối từ cha của đỉnh u đến đỉnh u sẽ tham gia vào **S** một lượng $sz[u] * (N - sz[u])$, do đó dễ dàng tính được **S** theo công thức:

$$\mathbf{S} = \sum_{u=1}^{N} sz[u] * (\mathrm{N} - \mathrm{sz[u]})$$

Ở đây $sz[u]$ là số đỉnh trong cây con $T[u]$.

Tiếp theo, với mỗi đường đi có độ dài L ta cần phải thêm vào số bước nhảy là $\left\lceil \frac{L}{k} \right\rceil$ để đảm bảo $L \vdots k$, thay vì tính $\left\lceil \frac{L}{k} \right\rceil$ ta tính $\frac{L+f(L,k)}{k}$ với $f(K,L)$ là số x cạnh nhỏ nhất để $(L+x) \vdots k$ rõ ràng $x \in [0..k-1]$. Ví dụ $f(10,3) = 2, f(11,3) = 1, f(12,3) = 0$. Nhiệm vụ của chúng ta phải đếm được số cạnh cần bổ sung thêm x. Kết quả của bài toán sẽ là:

$$result = \frac{\sum_{u=1}^{N} \mathrm{sz[u]} * (\mathrm{N} - \mathrm{sz[u]}) + \sum_{x=0}^{k-1} \mathrm{x} * \mathrm{cnt[x]}}{k}$$

Xét thời điểm cập nhật lại thông tin đỉnh u, sau khi đã duyệt xong con của nó là đỉnh v. Ta cần làm các việc sau:

- Tính số lượng cạnh cần bổ sung khi cập nhật v vào với u đang có.
- Cập nhật lại các loại độ dài của u khi chia dư cho k từ v lên.

Độ phức tạp thuật toán là: $O(N.k^2)$.

Có thuật toán tốt hơn với độ phức tạp O(N.k) được đính kèm trong thư mục Code.

2.8.3. Chương trình minh họa

```
#include <bits/stdc++.h>
#define N 200005
#define task "Jumping"
using namespace std;
vector<int> adj[N];
int cnt[N][10];
int sz[N];
long long ans;
int n,k,root;
int subtract(int a, int b) {
  return ((a-b)%k+k)%k;
}
void dfs(int u,int p,int depth) {
```

```
    cnt[u][depth%k]=sz[u]=1;
    for(int v:adj[u])
      if(v!= p) {
        dfs(v,u,depth+1);
        for(int i=0; i<k; ++i)
          for(int j=0; j<k; ++j) {
            int remainder=subtract(i+j,2*depth);
            int x=subtract(k,remainder);
            ans+= 1ll*x*cnt[u][i]*cnt[v][j];
          }
        for(int i=0; i<k; ++i)
          cnt[u][i]+=cnt[v][i];
        sz[u]+=sz[v];
      }
    ans+=1ll*sz[u]*(n-sz[u]);
  }
  int main() {
    freopen(task".inp","r",stdin);
    freopen(task".out","w",stdout);
    cin>>n>>k;
    int root;
    for(int i=1; i<=n; i++) {
      int p;
      cin>>p;
      if (p==0)  root=i;
      adj[p].push_back(i);
      adj[i].push_back(p);
    }
    dfs(root,0,0);
    cout<<(ans/k);
  }
```

2.8.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.9. Bài 9: Trọng tâm của cây

2.9.1. Đề bài

Cho cây T có N đỉnh, các đỉnh được đánh số từ 1 đến N. Có Q truy vấn là một số u, hãy xác định trọng tâm cây con T[u]. Trọng tâm của cây được định nghĩa là đỉnh mà khi xóa nó khỏi cây ta có một rừng, trong đó không có cây nào trong rừng có quá ½ số đỉnh của cây ban đầu.

Dữ liệu vào:

Dòng đầu tiên là số N là số đỉnh của cây và số Q là số truy vấn

Dòng tiếp theo có N số $p_1, p_2, .. p_N$ thể hiện có đường đi từ đỉnh i đến p_i với $p_i \neq 0, p_i = 0$ thì đỉnh i là đỉnh gốc ban đầu.

Dòng tiếp theo có Q số tương ứng với Q truy vấn.

Kết quả ra: Q số là câu trả lời của Q truy vấn. Nếu truy vấn có nhiều đáp án, hãy in đỉnh là trọng tâm của cây con gần đỉnh gốc cây ban đầu nhất.

Ràng buộc: $1 \leq N, Q \leq 3.10^5$

Ví dụ:

Centroid.inp	**Centroid.out**
14 9 0 1 1 1 2 2 3 4 4 4 5 5 7 7 1 2 3 4 5 6 7 8 9	1 5 7 4 5 6 7 8 9

2.9.2. Phân tích bài toán

Thuật toán tìm trọng tâm của cây T[u] với độ phức tạp $O(size(T[u])$ như sau:

- Sử dụng DFS để tính kích thước của cây con trong cây T[u].

- DFS một lần nữa tìm trọng tâm của cây như sau: đi vào đỉnh có kích thước lớn hơn ½ kích thước của cây, đến khi nào không tìm được đỉnh như vậy, thì đỉnh đang đứng chính là trọng tâm của cây.

Thuật toán 1: Nếu sử dụng cách làm như trên thì tổng độ phức tạp thuật toán sẽ là $O(Q.N)$.

Thuật toán 2:

- Dùng một lần DFS để tính kích thước cây con của từng đỉnh.

- Xây dựng Euler tour cho cây $T[1]$.

- Với mỗi cây con T[u] ta sẽ xử lý truy vấn trong $O(\log^2 N)$ bằng cách kết hợp Segment tree để tìm đỉnh con có kích thước nhỏ nhất mà còn lớn hơn ½ kích thước cây T[u]. Đấy chính là trọng tâm cây.

- Độ phức tạp chung là $O(NlogN + Q.\log^2 N)$

Thuật toán 3: Sử dụng kĩ thuật phân rã trọng tâm (Centroid Decomposition) để tiền xử lý mất $O(N.logN)$, trả lời truy vấn trong $O(Q.\log^2 N)$.

Thuật toán 4: Dùng kĩ thuật Heavy light Decomposition, với mỗi đỉnh được gán trọng số là kích thước cây con với đỉnh đó là gốc. Dựa trên cây phân rã, tìm đến đỉnh có kích thước nhỏ nhất mà còn lớn hơn ½ kích thước của cây. Đó chính là trọng tâm của cây. Độ phức tạp $O(NlogN + Q.\log^2 N)$.

Thuật toán 5: Quy hoạch động trên cây.

- Dùng một lần DFS để tính kích thước của các cây $T[u]$.

- Với các đỉnh u là lá thì $dp[u] = u$. Để tìm hết $dp[u]$ với u là lá trong không quá $O(N)$.

- Với u không là lá, thì ta đi tìm v mà có kích thước lớn hơn ½ kích thước của u. Chắc chắn $dp[u]$ sẽ nằm trên $dp[v]$, nằm trong đoạn từ $dp[v]$ tới u. Do đó quá trình tìm tất cả các $dp[u]$ không phải là lá cũng chỉ mất $O(N)$.

➔ Độ phức tạp là: $O(N + Q)$.

2.9.3. Chương trình minh họa

```
#include <bits/stdc++.h>
#define N 300005
#define task "centroid"
using namespace std;
vector<int> adj[N];
int n,q,parent[N],sz[N],dp[N];
void dfs(int x) { //tinh kich thuoc cua cay, cay con
  sz[x] = 1;
  for(int y:adj[x]) {
    dfs(y);
    sz[x] += sz[y]; //cap nhat kich thuoc
  }
}
void fix(int x) {
  for(int y:adj[x])
    fix(y); //di xuong duyet tu nut la'
  for(int y:adj[x]) {
    if(sz[y]*2 >= sz[x]) {
      y=dp[y];
      while(sz[y]*2 <= sz[x])
        y=parent[y];
      dp[x]=y; //tim duoc tam
      return; //thi return
    }
  }
  dp[x] = x;
}
int main() {
  freopen(task".inp","r",stdin);
  freopen(task".out","w",stdout);
  scanf("%d %d",&n,&q);
  for(int i=1; i<=n; i++) {
    scanf("%d",&parent[i]);
    adj[parent[i]].push_back(i);
  }
  dfs(1);
  fix(1);
```

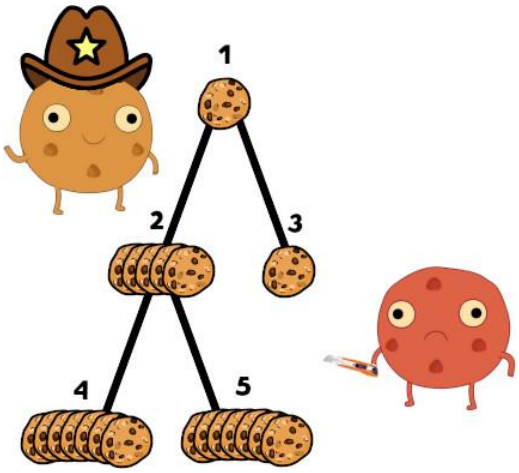
```
    for(int i=0; i<q; i++) {
      int x;
      scanf("%d",&x);
      printf("%d ",dp[x]);
    }
    return 0;
  }
```

2.9.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.10. Bài 10: Bánh Quy

2.10.1. Đề bài



Một trò chơi ăn bánh quy như sau: Có 2 người chơi, chơi luôn phiên nhau. Có một cây $T[1]$ gồm N đỉnh, tại mỗi đỉnh v có đặt số bánh $x[v]$, thời gian để ăn hết một cái bánh tại đây là $t[v]$, thời gian để di chuyển từ đỉnh cha đến nó hoặc ngược lại là $L[v]$, người chơi A đi trước và cố gắng ăn được càng nhiều bánh càng tốt và quay trở lại gốc 1 bằng đường cũ nhưng không được sau thời điểm $\boldsymbol{Time}$ định sẵn. Người chơi B đi sau và cố gắng ngăn người A ăn được nhiều bánh bằng cách cắt đường đi từ vị trí đỉnh v người A đang đứng đến đỉnh con của v. Trò chơi sẽ kết thúc khi người A đi xuống đỉnh lá của cây. Lúc đó người A có thể ăn thêm bánh khi quay trở lại gốc của cây theo đường cũ nếu còn thời gian.

Trong trường hợp cả A và B đều chơi tối ưu, thì A có thể ăn được nhiều nhất bao nhiêu cái bánh?

Dữ liệu vào:

Dòng đầu tiên là số N và $\boldsymbol{Time}$. Dòng thứ 2 chứa N số $x_1, x_2, \dots x_N$.

Dòng thứ 3 chứa N số $t_1, t_2, \dots t_N$

Dòng thứ 4 chứa $N-1$ số $p_2, p_3, \dots p_N$ có nghĩa là có cạnh nối trực tiếp i đến p_i $(1 \le p_i \le N)$.

Dòng thứ 5 chứa $N-1$ số $L_2, L_3, .. L_N$ là thời gian di duyển từ đỉnh 2,3.. N đến đỉnh cha của nó và ngược lại.

Kết quả ra:

Một số duy nhất là số nhiều nhất mà A có thể ăn được.

Ràng buộc: $2 \le N \le 10^5; 1 \le Time \le 10^{18};\ 1 \le t_i, x_i \le 10^6$.

Ví dụ:

Cookies.inp	Cookies.out	Giải thíchh

5 26 1 5 1 7 7 1 3 2 2 2 1 1 2 2 1 1 0 0	11	Người A sẽ đi theo đường 1→2→4 nếu bị người B chặt cạnh 2-5, hoặc 1→2→5 nếu bị người B chặt cạnh 2-4.

2.10.2. Phân tích bài toán

- Gọi dp[u] là giá trị tối ưu khi xét đến đỉnh u.

- Gọi 2 giá trị tối ưu nhất tại một đỉnh là m_1, m_2 với $m_1 > m_2$. Khi đó người chơi A chỉ có thể chọn được một lần duy nhất chọn được m_1 là lần chọn đầu tiên. Còn các lần sau đều chỉ chọn được m_2.

- Thời gian dành cho việc ăn bánh khi xét đến đỉnh u là:

$$Time - 2 * Tổng\ thời\ gian\ đi.$$

- Để ăn được nhiều bánh nhất thì cần ăn những bánh ở đỉnh mà có thời gian ăn ít nhất trước. Nên cần ăn bánh theo thứ tự thời gian tăng dần, đến khi hết thời gian ăn. Do đó nếu không sử dụng cấu trúc dữ liệu thì ta sẽ phải duyệt các thời gian trong O(N) thay vì O(logN) nếu dùng Segment tree. Segment tree sẽ lưu thông tin về thời gian ăn hết bánh và số bánh có thể ăn tối đa tại mỗi đỉnh khi duyệt DFS. Khi lấy thông tin $dp[u]$ thì dựa vào thời gian dành cho việc ăn bánh khi duyệt đến u. Sau khi duyệt xong u thì phải cập nhật xóa đi những thông tin khi của u khi đưa vào ban đầu.

- Kết quả là dp[1].

- Độ phức tạp của thuật toán là $O(N.\log(t_{max}))$

2.10.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define int long long
#define N 100005
#define task "Cookies"
using namespace std;
int n, dp[N],T,L[N],x[N], t[N];
int sumTime[40*N],sz[40*N]; //segment tree
vector<int> adj[N];
void update(int id, int l, int r, int t, int x) {
  if(t<l || r<t)
    return;
  if(l==r) {
    sumTime[id]+= 1ll*t*x; //thoi gian an het so banh
    sz[id]+=x; //so banh co the an duoc
    return;
  }
  update(2*id, l,(l+r)/2,t,x);
```

```
  update(2*id+1,(l+r)/2+1,r,t,x);
  sumTime[id]=sumTime[2*id]+sumTime[2*id+1];
  sz[id]=sz[2*id]+sz[2*id+1];
}
long long get(int id, int l, int r, int t) {
  if(l==r)
    return min(t/l,sz[id]); //an loai banh mat l time
  int w=sumTime[2*id]; //thoi gian an het nua trai
  if(w<t)
    return sz[2*id]+get(2*id+1,(l+r)/2+1,r,t-w);
  return get(2*id,l,(l+r)/2,t);
}
void dfs(int u, int time) {
  if(time<2*L[u])
    return;
  update(1,1,1e6+5,t[u],x[u]); //cap nhat cay ST
  dp[u]=get(1,1,1e6+5,time-2*L[u]);
  int m1=0,m2=0;
  for(int i=0; i<adj[u].size(); i++) {
    int v=adj[u][i];
    dfs(v,time-2*L[u]);
    if(dp[v]>m1)
      m2=m1,m1=dp[v];
    else if(dp[v]>m2)
      m2=dp[v];
  }
  update(1,1,1e6+5,t[u],-x[u]); //cap nhat lai ST
  if(u==1) //A chi chon duoc m1 khi đi buoc dau tien
    dp[u]=max(dp[u],m1);
  else
    dp[u]=max(dp[u],m2);//sau thi chi chon duoc m2
}
signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  freopen(task".inp", "r", stdin);
  freopen(task".out","w",stdout);
  cin>>n>>T;
  for(int i=1; i<=n; i++) {
    cin>>x[i];
  }
  for(int i=1; i<=n; i++) {
```

```
    cin>>t[i];
  }
  for(int i=2; i<=n; i++) {
    int p;
    cin>>p;
    adj[p].push_back(i);
  }
  for(int i=2; i<=n; i++) {
    cin>>L[i];
  }
  dfs(1,T);
  cout<<dp[1];
}
```

2.10.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.11. Bài 11: Hội nghị Mỹ - Triều lần 4

Có thể nói LCA cũng là dạng bài toán quy hoạch động trên cây điển hình. Dưới đây, tôi xin giới thiệu một bài có áp dụng LCA và DFS theo mô hình duyệt cây đã đưa ra ban đầu. *Tôi không trình bày lại thuật toán tìm LCA(u,v) bằng bảng thưa (Sparse Table) vì nó cũng khá đơn giản và có nhiều nguồn tài liệu đã nói.*

2.11.1. Đề bài

Nhằm chuẩn bị cho hội nghị thượng đỉnh Mỹ - Triều lần thứ 4 thành công tốt đẹp, mang lại hiệu quả thực chất, mang lại hòa bình, thịnh vượng trên bán đảo Triều Tiên và khu vực. Việt Nam đã nhận đăng cai sự kiện này. Việt Nam có N khách sạn hạng sang, có N-1 con đường hai chiều nối giữa N khách sạn này, nhằm tạo cho hai bên tham gia đàm phán vui vẻ nhất thì nước chủ nhà phải bố trí sao cho khoảng cách từ chỗ ăn nghỉ đến nơi đàm phán của 2 bên có độ dài như nhau (độ dài tính bằng số cạnh). Do Mỹ và Triều Tiên còn chưa tin tưởng lẫn nhau nên đã bí mật gửi các phương án chỗ ăn nghỉ mà mình chọn cho Việt Nam. Nước chủ nhà cần phải tính toán sẵn phương án bố trí nơi đàm phán thỏa mãn yêu từng cầu trên của Mỹ và Triều Tiên. Mỗi khách sạn hạng sang đều đáp ứng được tất cả các nhu cầu về ăn nghỉ và cũng có thể là nơi đàm phán.

Coi bản đồ thể hiện vị trí của N khách sạn là cây $T[1]$.

Dữ liệu vào:

Dòng đầu tiên là số N, Q $(1 \leq N, Q \leq 10^5)$.

Dòng tiếp theo chứa $N-1$ số $p_2, p_3, \ldots p_N$ có nghĩa có cạnh nối trực tiếp i đến p_i $(1 \leq p_i \leq N)$.

M dòng tiếp theo thể hiện thông tin về khách sạn mà Mỹ, Triều Tiên lựa chọn làm nơi ăn nghỉ, dòng thứ i có 2 số u_i, v_i $(1 \leq u_i, v_i \leq N)$.

Kết quả ra:

Gồm M dòng, mỗi dòng là số khách sạn có thể chọn để đảm bảo yêu cầu.

Ví dụ:

Summit.inp	**Summit.out**
14 3 1 1 1 2 2 3 4 4 4 5 5 7 7 11 12 11 3 2 4	12 2 5
4 1 1 1 2 2 3	1
5 2 1 1 2 2 1 3 1 4	0 2

2.11.2. Phân tích bài toán

Thuật toán 1: Duyệt trâu DFS để xử lý lần lượt từng truy vấn, tìm các điểm mà cách đều (u,v). Độ phức tạp $O(N.Q)$.

Thuật toán 2:

Gọi $dist(u, v)$ là khoảng cách giữa u,v.

LCA(u,v) là tổ tiên chung gần nhất của u,v

Sz[u] là kích thước của cây T[u].

Depth[u] là độ sâu của đỉnh u.

Hướng thứ 1: Duyệt trâu có dùng LCA, mỗi truy vấn ta duyệt tất cả đỉnh trên cây, nếu có cùng khoảng cách tới u, v thì tăng ans lên 1.

Độ phức tạp truy vấn sẽ là: $O(Q.N.logN)$.

Hướng thứ 2: Xử lí khéo léo, chia các trường hợp để đếm nhanh số lượng đỉnh trong $O(logN)$. Độ phức tạp truy vấn còn $O(Q.logN)$.

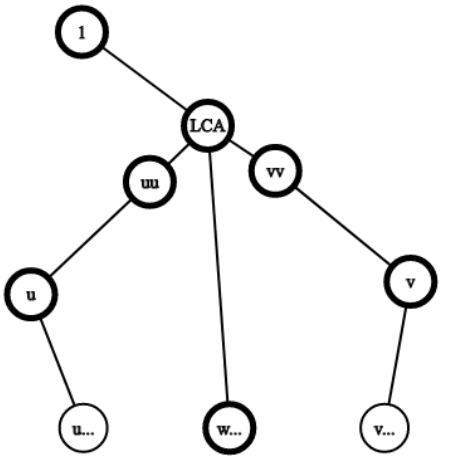
- Nếu $(u = v)$ thì kết quả trả về N.

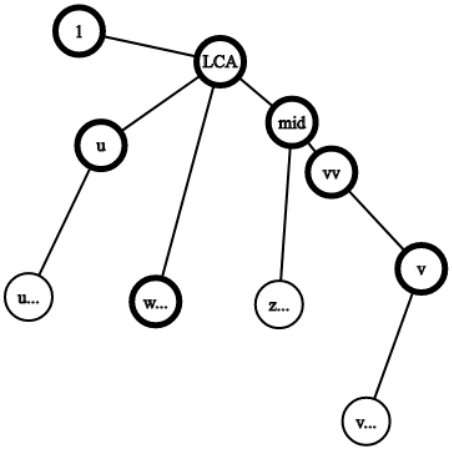
- Nếu $dist(u, v)$ lẻ thì kết quả trả về 0.

- Nếu L$CA(u,v)$ cách đều u,v khi đó tất cả các đỉnh cách đều u, v sẽ nằm ngoài $T[uu]$, $T[vv]$, với uu, vv là con trực tiếp của $LCA(u,v)$. Kết quả trả về: $N - sz[uu] - sz[vv]$

- Còn lại kết quả là: $sz[mid] - size[vv]$

Xem hình bên cạnh:





2.11.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define N 100005
#define task "Summit"
using namespace std;
int  n,m,parent[17][N],  depth[N],
sz[N];
vector <int> adj[N];
void dfs(int u, int p) {
  parent[0][u]=p;
  sz[u]=1;
  depth[u]=depth[p]+1;
  for(int i=1; i<=16; i++)
    parent[i][u]=parent[i-
1][parent[i-1][u]];
  for(int v:adj[u]) {
    if(v!=p) {
      dfs(v,u);
      sz[u]=sz[u]+sz[v];
    }
  }
}
int LCA(int u, int v) {
  if(depth[u]<depth[v])
    swap(u,v);
  for(int i=16; i>=0; i--) {
    if(depth[u]-depth[v]>=(1<<i))
      u=parent[i][u];
  }
  if(u==v)
    return u;
  for(int i=16; i>=0; i--)
    if(parent[i][u]!=parent[i][v]) {
      u=parent[i][u];
```

```
      v=parent[i][v];
    }
  return parent[0][u];
}
int p_th(int u, int d) {
  for(int i=16; i>=0; i--)
    if(depth[u]-d>=(1<<i))
      u=parent[i][u];
  return u;
}
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0), cout.tie(0);
  freopen(task".inp","r",stdin);
  freopen(task".out","w",stdout);
  cin>>n;
  for(int i=1; i<n; i++){
    int u,v;
    cin>>u>>v;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  dfs(1,0);
  cin>>m;
  while(m--) {
    int u, v, ans;
    cin>>u>>v;
    if(depth[u]<depth[v])
      swap(u,v);
    if(u==v)
      cout<<n<<"\n";
    else {
      int lca=LCA(u,v);
      int dist=depth[u]+depth[v]-2*depth[lca];
      if(dist%2)
        cout<<"0\n";
      else {
        int dep_mid=max(depth[u],depth[v])-dist/2;
        if(dep_mid==depth[lca]) {
          ans=N-sz[p_th(u,dep_mid+1)]-sz[p_th(v,dep_mid+1)];
          cout<<ans<<"\n";
        } else {
```

```
                ans=sz[p_th(u,dep_mid)]-sz[p_th(u,dep_mid+1)];
                cout<<ans<<"\n";
            }
        }
    }
  }
}
```

2.11.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.12. Bài 12: Tổng trên cây 5.

2.12.1. Đề bài

Cho một cây có trọng số $T[1]$ gồm N đỉnh được đánh số từ 1 đến N. Gọi $d[u,v]$ là tổng trọng số trên đường đi ngắn nhất từ u đến v. Gọi S[v] là tập hợp các đỉnh u thỏa mãn: $d[1,u] = d[1,v] + d[v,u]$. Hàm số $f[u,v]$ được định nghĩa như sau: $f[u,v] = \sum_{x \in S[v]} \left(d[u,x]\right)^2 - \sum_{x \notin S[v]} \left(d[u,x]\right)^2$. Hãy trả lời Q truy vấn về tính $f(u,v)$ được đưa ra.

Dữ liệu vào:

Dòng đầu tiên là số N.

N-1 dòng tiếp theo thể hiện cạnh nối của cây bởi bộ 3 số u_i, v_i, w_i có nghĩa là cạnh nối $u_i \rightarrow v_i$ có trọng số là w_i.

Dòng tiếp theo là số Q, số truy vấn.

Q dòng tiếp theo thể hiện các truy vấn bởi bộ 2 số u, v có nghĩa là in ra kết quả của $f(u,v)$. Kết quả cần tính chia lấy dư cho $10^9 + 7$.

Kết quả ra: Một dòng gồm Q số là kết quả của Q truy vấn trên.

Ràng buộc: $2 \le N, Q \le 10^5; 1 \le w_i \le 10^9$.

Ví dụ:

Summax5.inp	Summax5.out
5 1 2 1 4 3 1 3 5 1 1 3 1 5 1 1 1 5 2 4 2 1 3 5	10 1000000005 1000000002 23 1000000002

2.12.2. Phân tích bài toán

Nhận xét: $S[v]$ chính là tập các đỉnh con của v.

Thuật toán 1: Để tính $f(u,v)$ trước hết ta đánh dấu các điểm thuộc S[v]. Duyệt DFS bắt đầu từ u để tính các khoảng cách từ u đến tất cả các đỉnh trên cây T[1]. Tỉnh tổng các bình phương khoảng cách sau đó in ra kết quả.

Theo cách tính trên thì mỗi truy vấn ta sẽ giải quyết trong $O(N)$. Do đó độ phức tạp của toàn bài là $O(Q.N.logN)$ chỉ qua được 50% số test.

Thuật toán 2: Cải tiến thuật toán 1, do khi tính khoảng cách từ một đỉnh đến các đỉnh còn lại trên cây ta không kế thừa được, nên độ phức tạp khá lớn. Vậy có thể tính trước các thông tin cần thiết, sau đó truy vấn chỉ trong $O(1)$ được không?

Ta có thể làm được trong $O(1)$ như sau:

- Lưu thông tin tổng khoảng cách từ u đến các con của $T[u]$ trong $F1[u]$, tổng khoảng cách từ đỉnh u đến đỉnh bên ngoài u là $F2[u]$.

- Để có thể tính kế thừa thông tin ta cần sử dụng cấu trúc để lưu 3 thông tin về: kích thước (x0), tổng khoảng cách (x1), tổng bình phương khoảng cách (x2).

Trước hết ta tính $F1[.]$: Xét cạnh từ $u \rightarrow v$ thì:

$F_1[v] = \sum_{v' \in S[v]} \left(d(v',v)\right)^2$, gọi $d(u,v) = w$. Thì thông tin của $T[v]$ cập nhật lên u được tính như sau:

$$F1[u] = F1[u] + \sum_{v' \in S[v]} \left(d(v',v) + w\right)^2$$

$$= F1[u] + \sum_{v' \in S[v]} \left(d(v',v)\right)^2 + 2 * w * \sum_{v' \in S[v]} d(v',v) + S[v] * w^2$$

Sử dụng một lần DFS nữa để tính F2[v] từ F2[u] tương tự bài **Coloring và Summax4.**

Chú ý w có thể là khoảng cách giữa 2 đỉnh (u,v) là $dist(u,v)$ trên cây, để có khoảng cách ta cần dùng LCA. Khi tính $f(u,v)$ cần xét trường hợp $v = LCA(u,v)$.

Độ phức tạp thuật toán $O(NlogN)$ do tìm LCA và tìm khoảng cách giữa 2 đỉnh, còn lại tính $F1[.]$ và $F2[.]$ chỉ mất $O(N)$.

Xem chương trình minh họa bên dưới, có kèm theo comment để giải thích rõ thêm thuật toán 2.

2.12.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define N 100005
#define MOD 1000000007
#define task "summax5"
using namespace std;
struct Node {
  int x2; //tong binh phuong khoang cach trong cay
  int x1; //tong cac khoang cach trong cay
```

```
  int x0; //kich thuoc cua cay
} F1[N],F2[N];
//F1[u] k/c den cac nut trong cay T[u]
//F2[u] tinh  k/c nut u den cac nut ngoai T[u]
Node operator +(Node a, Node b) {
  return Node{(a.x2+b.x2)%MOD,(a.x1+b.x1)%MOD,(a.x0+b.x0)%MOD};
}
Node operator -(Node a,Node b) {
  return Node{(a.x2-b.x2)%MOD,(a.x1-b.x1)%MOD,(a.x0-b.x0)%MOD};
}
Node operator *(Node a,int d) {
  Node c;
  c.x2=(a.x2+2ll*d*a.x1+1ll*d*d%MOD*a.x0)%MOD;
  c.x1=(a.x1+1ll*d*a.x0)%MOD;
  c.x0=a.x0;
  return c;
}
int d[N],parent[17][N],depth[N],Q,n;
//d[.] khoang cach den nut goc.
//parent[.], depth[.] de tinh LCA
vector<pair<int,int>> adj[N];//danh sach ke
int LCA(int x,int y) {
  if(depth[x]<depth[y])
    swap(x,y);
  for(int i=16; i>=0; i--)
    if(depth[x]-depth[y]>=(1<<i))
      x=parent[i][x];
  if(x==y)
    return x;
  for(int i=16; i>=0; i--)
    if(parent[i][x]!=parent[i][y])
      x=parent[i][x],y=parent[i][y];
  return parent[0][x];
}
void dfs(int u,int p) {
  parent[0][u]=p;
  for(int i=1; i<=16; i++)
    parent[i][u]=parent[i-1][parent[i-1][u]];
  F1[u]= {0,0,1};
  for(auto e:adj[u])
    if(e.first!=p) {
      int v=e.first;
```

```
      depth[v]=depth[u]+1;
      d[v]=(d[u]+e.second)%MOD;
      dfs(v,u);
      F1[u]=F1[u]+F1[v]*e.second;
    }
}
void fix(int u,int p) {//tinh F2[u]
  for(auto e:adj[u])
    if(e.first!=p) {
      int v=e.first;
      F2[v]=(F2[u]+F1[u]-F1[v]*e.second)*e.second;
      fix(v,u);
    }
}
int main() {
  freopen(task".inp","r",stdin);
  freopen(task".out","w",stdout);
  scanf("%d",&n);
  for(int i=1; i<n; i++) {
    int x,y,k;
    scanf("%d%d%d",&x,&y,&k);
    adj[x].push_back({y,k});
    adj[y].push_back({x,k});
  }
  dfs(1,0);
  fix(1,0);
  scanf("%d",&Q);
  while(Q--) {
    int u,v;
    scanf("%d%d",&u,&v);
    int lca=LCA(su,v),dist=(1ll*d[u]+d[v]-2*d[lca])%MOD,ans;
    if(lca==v)
      ans=(1ll*F1[u].x2+(F2[u]-F2[v]*dist).x2-(F2[v]*dist).x2)%MOD;
    else
      ans=(2ll*(F1[v]*dist).x2-1ll*(F2[u]+F1[u]).x2)%MOD;
    printf("%d\n",(ans+MOD)%MOD);
  }
}
```

2.12.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

2.13. Bài 13: Khoảng cách

2.13.1. Đề bài

Cho một cây ban đầu có N đỉnh, N-1 cạnh, người ta xóa hết các cạnh của cây, sau đó thực hiện Q truy vấn, mỗi truy vấn thuộc 1 trong 2 loại:

- Loại 1: Cho bởi bộ 4 số (1, u, v, w) nghĩa là truy vấn thực hiện thêm cạnh có trọng số w giữa 2 đỉnh u, v. Ví dụ: (1, 1, 3, 4) thêm cạnh có trọng số 4 từ 1 đến 3. Cạnh được thêm vào đảm bảo không xuất hiện chu trình.

- Loại 2: cho bởi bộ 3 số (2,u,v) Hỏi độ dài đường đi ngắn nhất từ đỉnh u đến đỉnh v hiện tại bằng bao nhiêu. Nếu không có đường đi thì in ra -1.

Các truy vấn là xen kẽ nhau. Hãy trả lời cho các truy vấn loại 2.

Dữ liệu vào: Dòng đầu tiên là số N và Q.

Q dòng sau thể hiện các truy vấn có thể loại 1 hoặc loại 2.

Kết quả ra: In ra câu trả lời cho các truy vấn loại 2 theo thứ tự của nó.

Ràng buộc: $1 \leq N, Q \leq 2.10^5;\ 1 \leq w \leq 10^4$.

Ví dụ:

DISTANCE.inp	DISTANCE.out
5 8 1 1 2 5 1 3 5 7 2 2 4 1 5 4 1 2 3 4 1 1 5 6 2 1 4 2 2 4	-1 8 7 12

2.13.2. Phân tích bài toán

- Thực chất đây là bài toán tính khoảng cách giữa 2 đỉnh trên cây. Tuy nhiên cây là cây động. Các cạnh được thêm vào dần dần.

Thuật toán 1: Nếu ta xử lý như trường hợp cây tĩnh, thì mỗi truy vấn loại 2 ta lại dùng một lần duyệt DFS từ u hoặc v để tính khoảng cách giữa u, v thì độ phức tạp thuật toán sẽ là $O(N.Q)$.

Thuật toán 2: Nhận thấy là thao tác thêm cạnh là hợp nhất 2 cây trong rừng. Do đó, để tối ưu ta không duyệt lại toàn bộ cây sau hợp nhất mà ta cây mà chỉ DFS cập nhật thông tin của cây nhỏ hơn từ cây lớn hơn. Kết hợp cả LCA và hợp cây ta mất thời gian $O((N+Q).\log^2 N)$. Bài này tôi sử dụng thêm cấu trúc DSU.

2.13.3. Chương trình minh họa

```
#include<bits/stdc++.h>
using namespace std ;
typedef pair<int,int> pii;
```

```
#define fi first
#define se second
const int N = 2e5+9;
vector<pii> adj[N];
int parent[N][20],par[N],rnk[N],depth[N],dist[N];
void dfs0(int u,int v) {
  parent[u][0]=v;
  depth[u]=depth[v]+1;
  for(int i=1; i<20; i++)
    parent[u][i]=parent[parent[u][i-1]][i-1];
  for(auto it:adj[u])
    if(it.fi!=v)
      dist[it.fi]=dist[u]+it.se,dfs0(it.fi,u);
}
int Find(int u) {
  return u==par[u]?u:par[u]=Find(par[u]);
}
bool Union(int u,int v,int cst) {
  int pu=Find(u),pv=Find(v);
  if(pu==pv)
    return 0;
  if(rnk[pu]>=rnk[pv])
    rnk[pu]+=rnk[pv],par[pv]=pu,swap(u,v);
  else
    rnk[pv]+=rnk[pu],par[pu]=pv;
  dist[u]=dist[v]+cst;
  dfs0(u,v);
  adj[u].emplace_back(v,cst);
  adj[v].emplace_back(u,cst);
  return 1;
}
int blift(int u,int cst) {
  for(int i=0; i<20; i++)
    if((1<<i)&cst)
      u=parent[u][i];
  return u ;
}
int Findlca(int u,int v) {
  if(depth[u]>depth[v])
    swap(u,v);
  if(blift(v,depth[v]-depth[u])==u)
    return u ;
```

```
  int l = -1, r = depth[u] ;
  while(l+1<r) {
    int mid = l+r>>1 ;
    if(blift(u,depth[u]-mid)!=blift(v,depth[v]-mid))
      r=mid;
    else
      l=mid ;
  }
  return blift(u,depth[u]+1-r);
}
int query(int u,int v) {
  if(Find(u)!=Find(v))
    return -1 ;
  int lca = Findlca(u,v);
  return dist[u]+dist[v]-2*dist[lca];
}
int main() {
  freopen(".inp","r",stdin);
  int n,q ;
  scanf("%d%d",&n,&q);
  for(int i=1; i<=n; i++)
    par[i]=i,rnk[i]=1;
  int ans = 0 ;
  while(q--) {
    int t,u,v,w ;
    scanf("%d%d%d",&t,&u,&v);
    if(t==1) {
      scanf("%d",&w);
      Union(u,v,w);
    } else {
      ans = query(u,v);
      printf("%d\n",ans);
    }
  }
}
```

2.13.4. Test

https://drive.google.com/drive/folders/19QzO16qnw02jb31tItlBiKkvQO_BdLR1?usp=sharing

3. Một số bài tự luyện

Tôi đưa ra đây một số bài tập về cây để học sinh tự luyện thêm.

3.1. Bài 1: Tổng trên cây

Cho một cây có N đỉnh, các đỉnh được đánh số từ 1 đến N, các cạnh đều có trọng số, hãy tỉnh tổng khoảng cách từ một đỉnh u đến các đỉnh còn lại trên cây.

Dữ liệu vào:

Dòng đầu tiên chứa số N là số đỉnh của cây.

$N-1$ dòng tiếp theo, mỗi dòng chứa 3 số nguyên (u, v, w) biểu diễn cạnh nối giữa u và v có trọng số là w $(1 \leq u, v \leq N \leq 10^6, u \neq v, 1 \leq w \leq 10^6)$.

Kết quả ra:

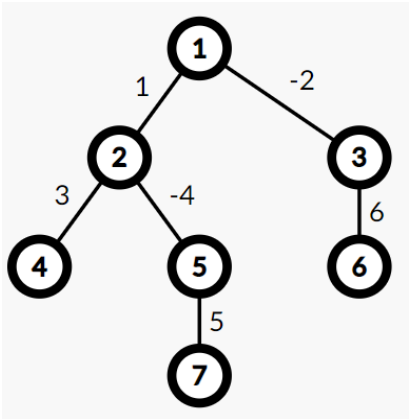
Một dòng gồm N số, số thứ u là tổng khoảng cách từ đỉnh u đến các đỉnh còn lại trên đồ thị.

Ví dụ:

SUMTREE.inpp	SUMTREE.out
4 1 4 7 2 3 5 4 2 6	38 24 34 24
4 1 2 2 3 1 4 4 3 5	15 19 15 25

3.2. Bài 2: Thêm cạnh

Cho một cây T[1] gồm N đỉnh, mỗi cạnh có trọng số w. Có Q truy vấn được cho bởi bộ 3 số (u, v, x) đảm bảo rằng cây con T[u] và T[v] là không có đỉnh chung, bạn được phép thêm một cạnh để nối một đỉnh trong T[u] với một đỉnh trong T[v], hãy tìm độ dài đường đi dài nhất có thể từ 1 đỉnh trong T[u] đến một đỉnh trong T[v] khi đó.



Dữ liệu vào:

Dòng đầu tiên gồm 2 số N, Q $(1 \leq N, Q \leq 10^5)$.

N-1 dòng tiếp theo là cạnh của cây (u, v, w) với $1 \leq u, v \leq n, 1 \leq w \leq 10^9$.

Q dòng tiếp theo mô tả truy vấn (u, v, x) với $1 \leq u, v \leq n, 1 \leq x \leq 10^9$.

Kết quả ra: Q số là đáp số lần lượt của Q truy vấn.

Ví dụ:

TREEDGE.inp	TREEDGE.out
7 3 1 2 1 1 3 -2 2 4 3	10 7 5

2 5 -4 5 7 5 3 6 6 2 3 1 5 4 2 5 6 0	

Với truy vấn 1, ta sẽ nối thêm cạnh 1→6 có trọng số 1. Đường đi có độ dài lớn nhất là đường đi: 2→4→6→3 với độ dài 10.

3.3. Bài 3: LCA.

Cho một cây gốc 1, có N đỉnh, các đỉnh được đánh số từ 1 đến N. Hãy trả lời Q truy vấn tìm LCA của 2 đỉnh u và v.

Dữ liệu vào: Dòng đầu tiên là số N.

N-1 dòng tiếp theo thể hiện cạnh của cây.

Dòng tiếp theo là số Q.

Q dòng tiếp theo là cặp (u,v) cần tìm LCA của chúng.

Kết quả ra: Với mỗi truy vấn, in ra LCA của 2 đỉnh đã cho.

Ràng buộc: $N, Q \leq 100000$.

Ví dụ:

LCA.inp	LCA.out
7 6 1 6 4 4 7 3 4 1 2 2 5 6 3 3 7 7 1 3 5 7 7 6 2 4	3 7 1 1 6 1

3.4. Bài 4: LCA2.

Cho một cây gốc 1, có N đỉnh, các đỉnh được đánh số từ 1 đến N. Hãy trả lời Q truy vấn tìm LCA của 2 đỉnh u và v với trường hợp chọn đỉnh r làm gốc.

Dữ liệu vào:

Dòng đầu tiên là số N.

N-1 dòng tiếp theo thể hiện các cạnh của cây.

Dòng tiếp theo số Q.

Q dòng tiếp theo gồm bộ 3 số (u,v,r) cần tìm LCA của 2 đỉnh u, v trong trường hợp chọn r làm gốc.

Kết quả ra: Q dòng tương tứng đáp án cho Q câu hỏi.

Ràng buộc: $1 \leq N, Q \leq 100000$.

Ví dụ:

LCA2.inp	LCA2.out
5 5 3 5 4 2 5 3 1 5 3 1 1 5 2 3 2 2 1 1 5 5 3 3 5	1 5 2 5 3

3.5. Bài 5: SUM3.

Cho một cây N đỉnh. Có Q truy vấn tăng giá trị của tất cả các đỉnh nằm trên đường đi từ đỉnh u đến đỉnh v một lượng là x. In ra giá trị của các đỉnh của cây sau khi xử lý hết truy vấn.

Dữ liệu vào:

Dòng đầu tiên là số N

Dòng tiếp theo có N số là giá trị ban đầu của các đỉnh.

N-1 dòng tiếp theo thể hiện các cạnh của cây.

Dòng tiếp theo số Q.

Q dòng tiếp theo là bộ (u,v,x) tăng tất cả các đỉnh từ u đến v lượng x.

Kết quả ra: Một dòng là giá trị của N đỉnh sau Q truy vấn.

Ràng buộc: $1 \leq N, Q \leq 100000$.

Ví dụ:

SUM3.inp	SUM3.out
5 0 0 0 0 0 1 2 2 3 3 4 4 5 2 1 3 5 3 5 7	5 5 12 7 7

3.6. Một số bài khác

https://vn.spoj.com/problems/V8ORG - Tổ chức đối lập

https://vn.spoj.com/problems/CTREE -Tô màu nhỏ nhất

https://vn.spoj.com/problems/STONE1 - Rải sỏi

https://vn.spoj.com/problems/NTTREE - Tổng trọng số trên cây

https://vn.spoj.com/problems/MTREE - Another Tree Problem

http://vn.spoj.com/problems/NTPFECT - Đại diện hoàn hảo

https://vn.spoj.com/problems/HBTLCA - Dynamic LCA

https://vn.spoj.com/problems/LUBENICA

http://vn.spoj.com/problems/ITREE - Nhãn của cây

http://vn.spoj.com/problems/VOTREE - Cây

http://vn.spoj.com/problems/UPGRANET - VOI11 Nâng cấp mạng

4. Kết luận

Đồ thị dạng cây cũng là đồ thị đặc biệt, ta có thể dễ dàng biến cây thành mảng bằng Euler tour. Do đó, ta cũng có lớp bài toán trên cây áp dụng với các cấu trúc dữ liệu, thuật toán như đối với mảng.

Do cây tiền tố Trie cũng có thể coi như đồ thị dạng cây, nên ta cũng có dạng toán quy hoạch động trên cây Trie.

Sau khi dạy chuyên đề này, tôi chuyển sang dạy học sinh các dạng bài khác liên quan đến đồ thị dạng cây khá thuận lợi như: Heavy light decomposition, Centroid decomposition, Euler tour, Mo's algorithm trên cây,...

Do kinh nghiệm chưa nhiều, nên chắc chắn có thiếu sót. Tôi rất mong đồng nghiệp và bạn bè chia sẻ, góp ý thêm cho tôi.

Tôi xin chân thành cảm ơn!

5. Nguồn tài liệu tham khảo

http://vnoi.info

https://vn.spoj.com

https://www.geeksforgeeks.org

https://codeforces.com

https://www.codechef.com