**HEAVY LIGHT DECOMPOSITION**

Heavy light Decomposition là một cấu trúc dữ liệu được xây dựng trên cây cho phép giải quyết một lớp các bài toán truy vấn khi có sự thay đổi các giá trị cho trên các đỉnh hoặc các cạnh của cây. Chính xác hơn Heavy light Decomposition là một cách phân rã cây.

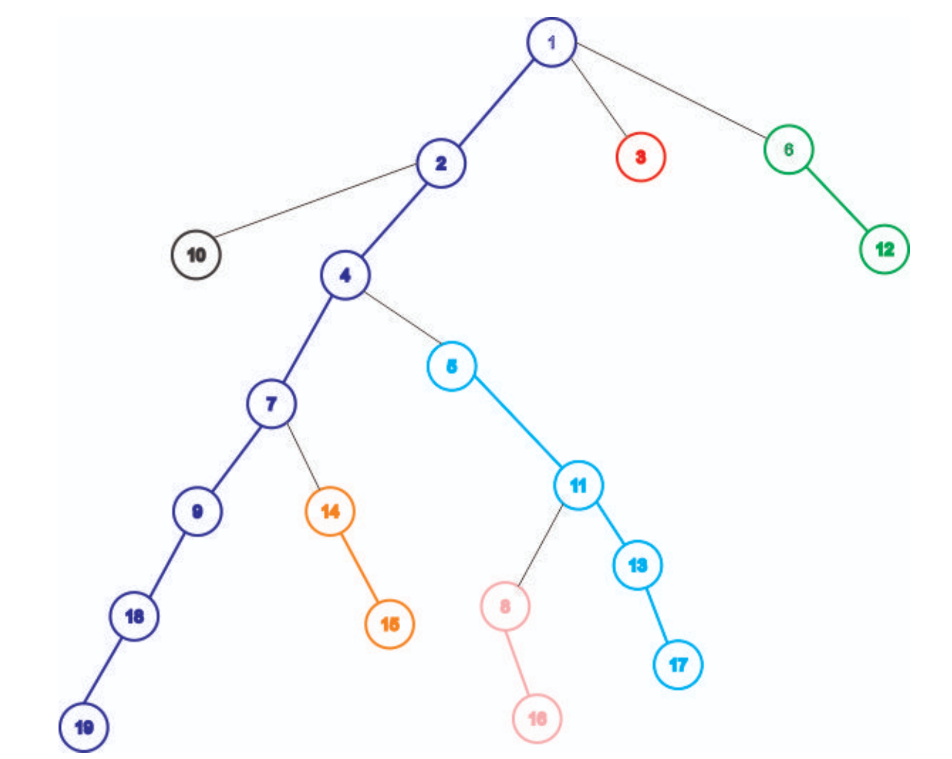
Giả sử cho một cây có trọng số G=(V,E). Bằng cách thực hiện BFS trên cây từ gốc ta định hướng lại cây và thu được các thông tin sau:

* - đỉnh cha của đỉnh
* - số lượng đỉnh có trên cây con gốc

Khi đó ta định nghĩa:

* Một cạnh với được gọi là cạnh nặng (heavy) nếu như . Ngoài ra với mỗi đỉnh thì chỉ có nhiều nhất một cạnh nặng nối từ nó đến con của nó
* Cạnh không phải là cạnh nặng thì được gọi là cạnh nhẹ (light)
* Đường nặng là đường chứa các cạnh nặng

Hình dưới đây cho một mô tả trực quan về cạnh nặng và cạnh nhẹ (các cạnh nhẹ có màu đen):



Nếu không xét các cạnh nhẹ thì tập hợp các cạnh nặng tạo thành một rừng các cây cạnh nặng có dạng đường thẳng (ta gọi tắt là các đoạn cạnh nặng).

Bây giờ chúng ta tạo một cây mới bằng cách coi mỗi đoạn cạnh nặng là một đỉnh, các cạnh của cây mới là cạnh nhẹ. Cây này được gọi là Heavy light Decomposition (HLD). Nếu như số đỉnh của cây ban đầu là thì độ sâu của HLD không vượt quá vì mỗi khi đi xuống theo một cạnh nhẹ thì số đỉnh bị giảm đi tối thiểu là một nửa. Do vậy mọi truy vấn tiến hành trên cây mới sẽ có thời gian không vượt quá

Mọi hành trình từ đỉnh u về gốc luôn gồm:

- Đi về đầu đường nặng

- Nhảy lên Pd

- Đi về đầu đường nặng

-…

Nếu ta để con nhiều đỉnh nhất ở vị trí đầu tiên trong danh sách kề thì các đỉnh của đường nặng nằm liên tiếp nhau

Về nguyên tắc mọi truy vấn về đường đi trong cây ban đầu sẽ được chuyển thành truy vấn đường đi trên HLD và truy vấn trên các đỉnh của HLD. Vì các đỉnh của HLD có cấu trúc đường thẳng nên các truy vấn trên đỉnh có thể sử dụng Interval Tree (IT), Binary Indexed Tree (BIT) hay cây nhị phân đầy đủ (BST). Và chi phí thời gian cho mỗi truy vấn sẽ là .

## Input;

int n;

vector<int> g[maxn];

## Output:

int Pd[maxn]; // Đỉnh cha

int pos[maxn]; // Vị trí các đỉnh khi DFS trên HLD

int head[maxn]; // head[u] - Đầu "đường nặng" chứa u

int d\_HLD[maxn]; // Mảng độ sâu trên HLD

*Các biến nháp*

int cl[maxn], s[maxn], id, smax[maxn];

*Code cơ bản:*

**//DFS lần 1 để tính s[...], smax[...] và đưa cạnh nặng về đầu**

void DFS(int u) {

cl[u]=1;

s[u]=1;

smax[u]=0;

int sz=g[u].size(), imax=0;

for(int i=0;i<sz;++i) {

int v=g[u][i];

if (cl[v]==0) {

Pd[v]=u;

DFS(v);

s[u] += s[v];

if (s[v]>smax[u]) {

smax[u]=s[v];

imax=i;

}

}

}

swap(g[u][0],g[u][imax]);

}

**DFS trên HLD**

void HLD(int u) {

pos[u]=++id;

for(auto &v : g[u]) if (Pd[v]==u) {

if (2\*s[v]>=s[u]) head[v]=head[u], d\_HLD[v]=d\_HLD[u];

else head[v]=v, d\_HLD[v]=d\_HLD[u]+1;

HLD(v);

}

}

**Tính LCA Bài toán tìm tổ tiên chung gần nhất**

Cho u,v là 2 đỉnh trên cây khi đó w=LCA(u,v) là tổ tiên chung gần nhất của u,v nếu:

w là tổ tiên của cả u và v;

Độ sâu của v là lớn nhất.

Code:

int LCA(int u,int v) {

while (d\_HLD[u]>d\_HLD[v]) u=Pd[head[u]];

while (d\_HLD[v]>d\_HLD[u]) v=Pd[head[v]];

while (head[u]!=head[v]) {

u=Pd[head[u]];

v=Pd[head[v]];

}

if (pos[u]<pos[v]) return u;

else return v;

}

**Cập nhật tăng các cạnh trên đường đi từ u đến v**

int E\_pd[maxn]; // E\_pd[u] - độ dài cạnh (u,Pd[u])

void IncEdges(int u,int v,int Delta) {

while (d\_HLD[u]>d\_HLD[v]) {

update(1,1,n,pos[head[u]]+1,pos[u],Delta);

u=head[u];

E\_pd[u] += Delta;

u=Pd[u];

}

while (d\_HLD[v]>d\_HLD[u]) {

update(1,1,n,pos[head[v]]+1,pos[v],Delta);

v=head[v];

E\_pd[v] += Delta;

v=Pd[v];

}

while (head[u]!=head[v]) {

update(1,1,n,pos[head[u]]+1,pos[u],Delta);

u=head[u];

E\_pd[u] += Delta;

u=Pd[u];

update(1,1,n,pos[head[v]]+1,pos[v],Delta);

v=head[v];

E\_pd[v] += Delta;

v=Pd[v];

}

if (pos[u]<pos[v]) {

update(1,1,n,pos[u]+1,pos[v],Delta);

} else {

update(1,1,n,pos[v]+1,pos[u],Delta);

}

}

**Lấy cạnh lớn nhất trên đường đi từ u đến v**

int GetEdges(int u,int v) {

int kq=INT\_MIN;

while (d\_HLD[u]>d\_HLD[v]) {

kq=maxx(kq,get(1,1,n,pos[head[u]]+1,pos[u]);

u=head[u];

kq=max(kq,E\_pd[u]);

u=Pd[u];

}

while (d\_HLD[v]>d\_HLD[u]) {

kq=max(kq,get(1,1,n,pos[head[v]]+1,pos[v]);

v=head[v];

kq=max(kq,E\_Pd[v]);

v=Pd[v];

}

while (head[u]!=head[v]) {

kq=max(kq,get(1,1,n,pos[head[u]]+1,pos[u]);

u=head[u];

kq=max(kq,E\_pd[u]);

u=Pd[u];

kq=max(kq,get(1,1,n,pos[head[v]]+1,pos[v]);

v=head[v];

kq=max(kq,E\_Pd[v]);

v=Pd[v];

}

if (pos[u]<pos[v]) {

kq=max(kq,get(1,1,n,pos[u]+1,pos[v]);

} else {

kq=max(kq,get(1,1,n,pos[v]+1,pos[u]);

}

return kq;

}