Heavy-light decomposition (HLD) là kĩ thuật phân tách một cây thành nhiều chuỗi đỉnh (chain) rời nhau. Sau đó, chúng ta có thể áp dụng các cấu trúc dữ liệu như Interval Tree hay Binary-Indexed Tree lên những chuỗi này để có thể cập nhật dữ liệu hoặc trả lời các truy vấn trên một đường đi giữa 2 đỉnh trong cây.

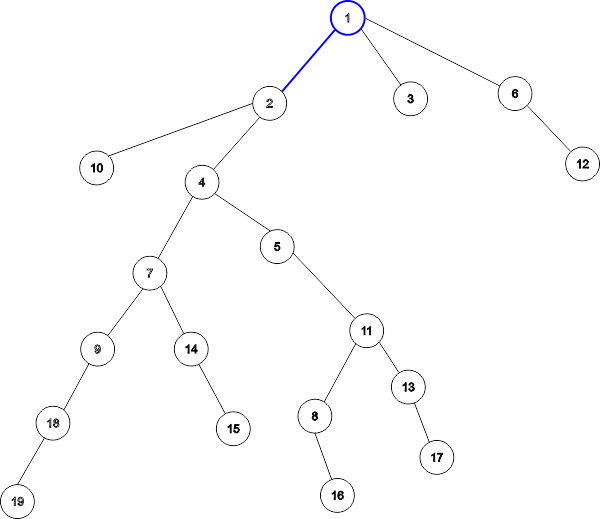
**Thuật toán phân tách cây**

Trước hết, chúng ta có các định nghĩa như sau:

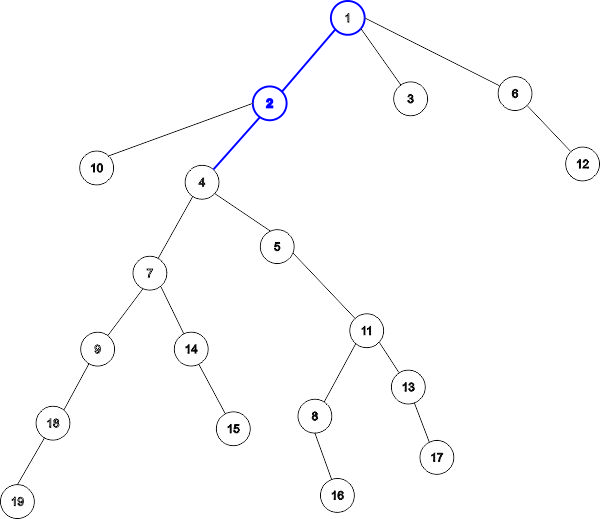
* Đỉnh con đặc biệt (heavy vertex): Trong số những đỉnh con của một đỉnh u không phải là lá, đỉnh đặc biệt v là gốc của cây con có kích thước lớn nhất.
* Cạnh đặc biệt (heavy edge): Là cạnh nối giữa u và v.
* Những đỉnh con còn lại của u gọi là đỉnh thường (light vertex) và những cạnh nối giữa u đến các đỉnh đó gọi là cạnh thường (light edge).

Dễ thấy là với mỗi đỉnh không phải là lá đều có thể chọn được đúng một cạnh và một đỉnh con đặc biệt của nó. Để tạo các chuỗi đỉnh, chúng ta làm như sau: bắt đầu từ đỉnh gốc, di chuyển xuống đỉnh con đặc biệt của nó và tiếp tục di chuyển xuống các đỉnh con tiếp theo đến khi gặp đỉnh lá thì kết thúc. Đường đi từ đỉnh gốc đến đỉnh lá này tạo thành một chuỗi đỉnh. Chúng ta lại lặp lại thao tác này với các đỉnh còn lại đến khi tất cả các đỉnh đều thuộc đúng một chuỗi nào đó.

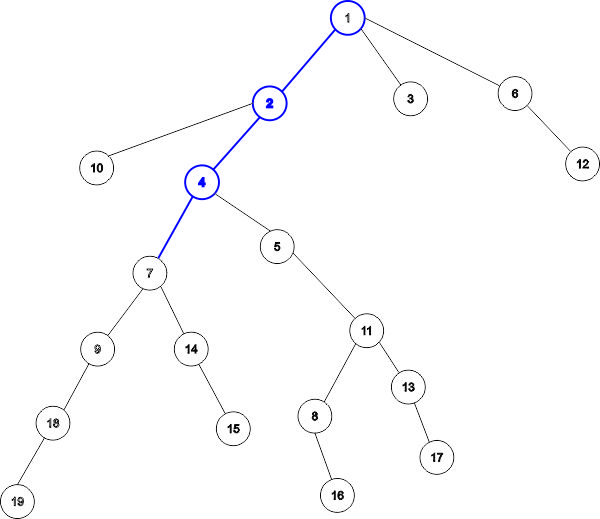
Để cho dễ hiểu, chúng ta có ví dụ sau:



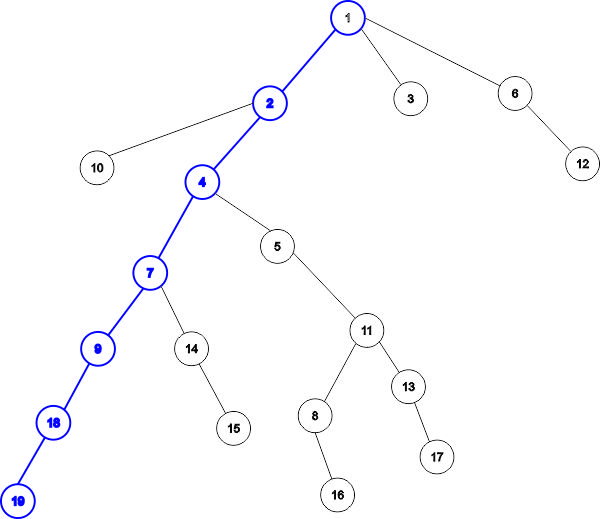
Từ đỉnh 1 di chuyển xuống đỉnh 2. Đỉnh đặc biệt của đỉnh 1 là đỉnh 2 vì cây con có đỉnh 2 làm gốc có kích thước lớn nhất.



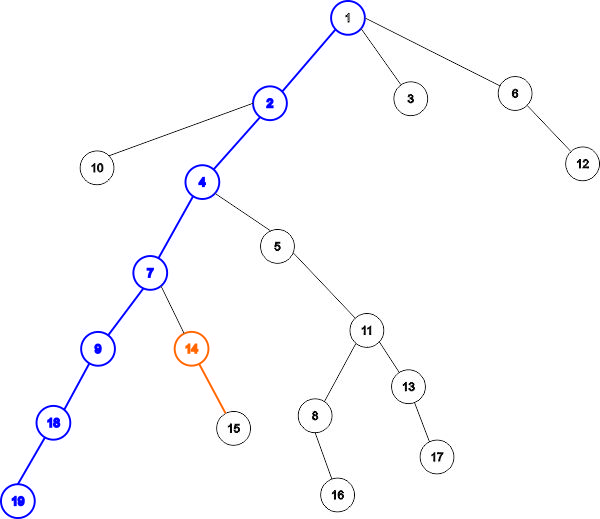
Từ đỉnh 2 di chuyển xuống đỉnh 4 vì cây con có gốc là đỉnh 4 có kích thước lớn nhất.



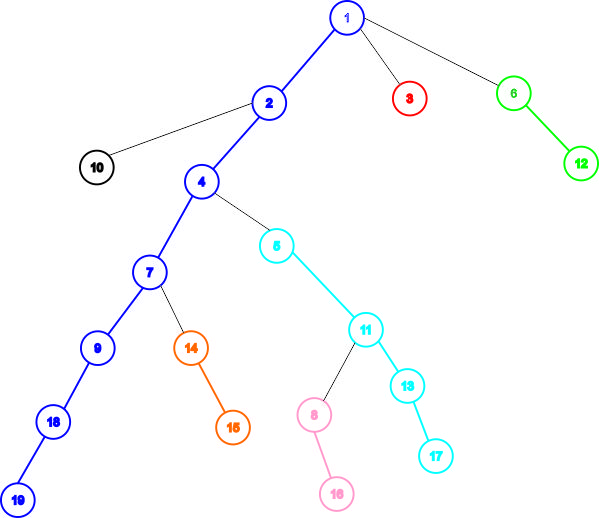
Từ đỉnh 4 di chuyển xuống đỉnh số 7. Tại đây 2 cây con có gốc là đỉnh 7 và đỉnh 5 đều có kích thước như nhau nên ta có thể chọn bất kì đỉnh nào.



Tiếp tục thực hiện cho đến khi gặp đỉnh lá. Như vậy là chúng ta đã có được một chuỗi đỉnh.



Chúng ta bắt đầu chuỗi mới ở một đỉnh gần nhất và lặp lại quá trình trên.



Cuối cùng chúng ta sẽ có một tập các chuỗi đỉnh rời nhau. Những cạnh được tô màu là cạnh đặc biệt và cạnh không được tô màu là cạnh thường.

// nChain chuỗi hiện tại. Sau khi kết thúc việc phân tách thì đây sẽ là tổng số chuỗi.

// chainHead[c] đỉnh đầu của chuỗi c

// chainInd[u] chuỗi mà đỉnh u nằm trong.

void hld(int u) {

// Nếu chuỗi hiện tại chưa có đỉnh đầu (đỉnh gần gốc nhất) thì đặt u làm đỉnh đầu của nó.

if (chainHead[nChain] == 0) chainHead[nChain] = u;

// Gán chuỗi hiện tại cho u

chainInd[u] = nChain;

// Giải thích bên dưới

posInBase[u] = ++nBase;

// Biến lưu đỉnh con đặc biệt của u

int mxVtx = -1;

// Tìm đỉnh con đặc biệt trong số những đỉnh con của u

for (int i = 0; i < adj[u].size(); i++) {

int v = adj[u][i];

if (v != parent[u]) {

if (mxVtx == -1 || nChild[v] > nChild[mxVtx]) {

mxVtx = v;

}

}

}

// Nếu tìm ra đỉnh con đặc biệt (u không phải là đỉnh lá) thì di chuyển đến đỉnh đó

if (mxVtx > -1)

hld(mxVtx);

// Sau khi đi hết một chuỗi thì tăng nChain lên và bắt đầu một chuỗi mới

for (int i = 0; i < adj[u].size(); i++) {

int v = adj[u][i];

if (v != parent[u] && v != mxVtx) {

nChain++;

hld(v);

}

}

}

Để có thể tiếp tục, chúng ta cần biết ít nhất các thông tin sau:

* Với một chuỗi, đỉnh đầu (đỉnh gần đỉnh gốc nhất) của nó là đỉnh nào.
* Với một đỉnh, chuỗi mà nó nằm trong là chuỗi nào.
* Ngoài ra chúng ta còn có mảng posInBase[]. Đây là mảng lưu lại vị trí của các đỉnh sau khi chúng ta "trải" các chuỗi trên lên một đường thẳng. Điều này sẽ giúp cho việc cài đặt các cấu trúc dữ liệu như Interval Tree hoặc Binary Indexed Tree một cách gọn gàng hơn.

http://i.imgur.com/fbhjbh8.jpg

Giả sử với hình trên thì posInBase[7] = 4; posInBase[14] = 8 ...

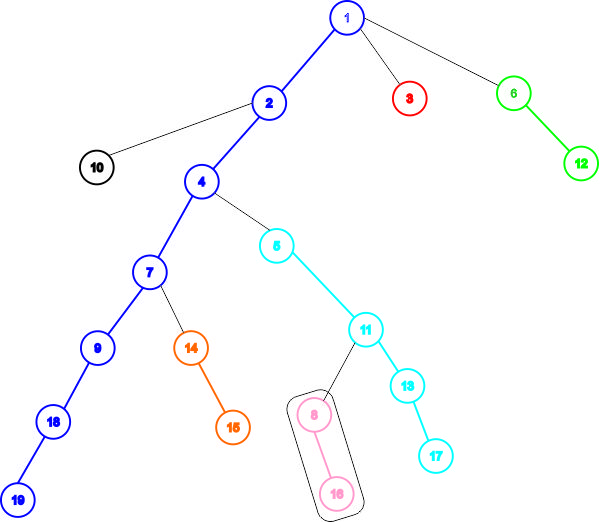
**Cập nhật và truy vấn một đường đi trên cây**

Thay vì cập nhật hoặc truy vấn một đường đi từ đỉnh u đến đỉnh v trên cây, chúng ta có thể thực hiện các thao tác này trên 2 đường đi từ u đến lca(u, v) và từ v đến lca(u, v) (lca là hàm tìm cha chung gần nhất của 2 đỉnh).

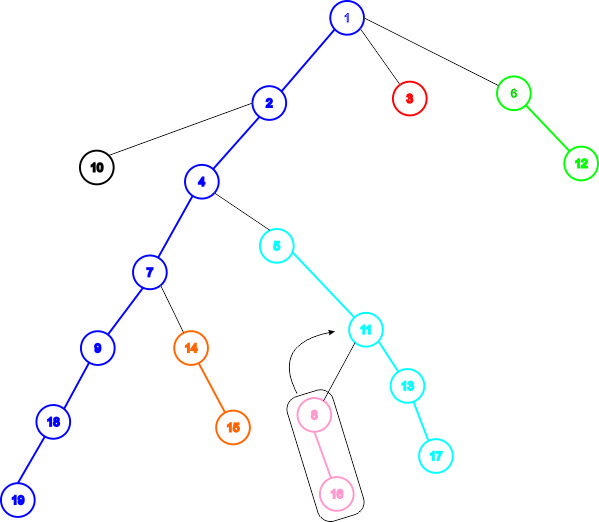
Giả sử chúng ta cần cập nhật đường đi từ u đến lca(u, v) (gọi tắt là a). Nếu a và u không cùng một chuỗi, chúng ta thực hiện thao tác cập nhật lên đường đi từ đỉnh u đến đỉnh đầu của chuỗi hiện tại. Sau đó cho u nhảy lên đỉnh cha của đỉnh đầu này rồi lặp lại thao tác cập nhật. Đến khi u và a nằm trên cùng một chuỗi, chúng ta chỉ cần cập nhật đoạn từ u đến a và kết thúc. Thao tác truy vấn được thực hiện tương tự.

Ví dụ:

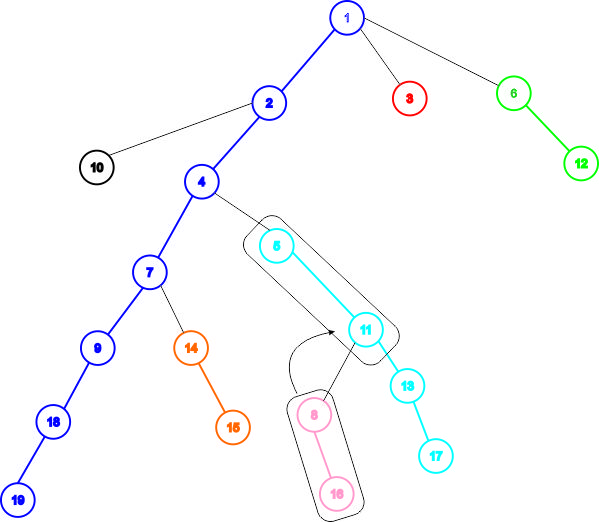
Chúng ta cần thực hiện cập nhật trên đường đi từ u = 16 đến a = 1.



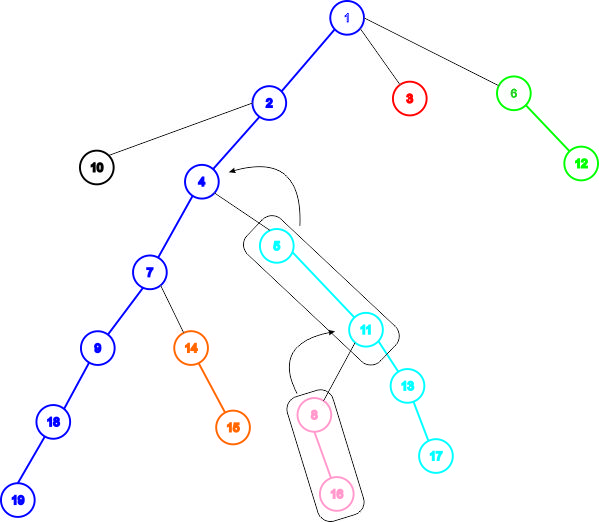
Gọi hàm update interval tree cho đoạn từ đỉnh 8 đến đỉnh 16.



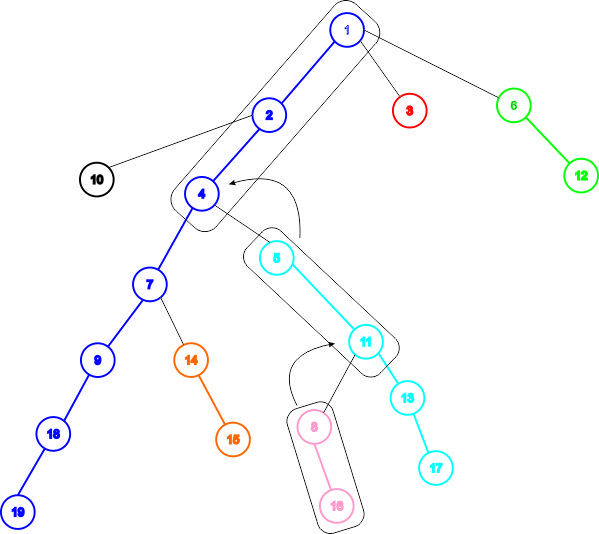
Nhảy lên đỉnh cha của đỉnh đầu của chuỗi hiện tại. Lúc này u = 11.



Gọi hàm update interval tree cho đoạn từ đỉnh 11 đến đỉnh 5.



Nhảy lên đỉnh cha của đỉnh đầu của chuỗi hiện tại. Lúc này u = 4.



Gọi hàm update interval tree cho đoạn từ đỉnh 4 đến đỉnh 1 và kết thúc.

void update(int u, int a) {

// uchain chuỗi hiện tại của u

// achain chuỗi của a

int uchain = chainInd[u], achain = chainInd[a];

while (1) {

// Nếu u và a cùng nằm trên một chuỗi thì update đoạn từ u đến a và kết thúc.

if (uchain == achain) {

updateIntervalTree(..., posInBase[a], posInBase[u], ...);

break;

}

// Nếu u và a không nằm trên cùng một chuỗi thì update đoạn từ u đến đỉnh đầu của chuỗi hiện tại.

updateIntervalTree(..., posInBase[chainHead[uchain]], posInBase[u], ...);

// Nhảy lên đỉnh cha của đỉnh đầu hiện tại.

u = parent[chainHead[uchain]];

uchain = chainInd[u];

}

}

**Độ phức tạp**

Với một cây có n đỉnh, khi đi từ đỉnh gốc đến một đỉnh lá bằng một đường đi bất kì thì số lần chúng ta phải nhảy chuỗi sẽ không vượt quá log(n). Để chứng minh điều này, chúng ta có thể thấy rằng, khi nhảy từ một đỉnh bất kì đến đỉnh con thường của nó thông qua cạnh thường thì số lượng đỉnh con có thể đi được sẽ giảm đi xuống còn tối đa một nửa so với ban đầu (nếu số đỉnh con còn lại nhiều hơn một nửa số đỉnh ban đầu thì đỉnh chúng ta nhảy tới đã là đỉnh con đặc biệt). Và nếu chúng ta tiếp tục nhảy qua nhiều chuỗi mới thì số lượng đỉnh sẽ giảm theo bội của 2. Bên cạnh đó, chúng ta sử dụng cấu trúc dữ liệu đặc biệt cho việc cập nhật hoặc truy vấn thông tin các đỉnh trong cùng một chuỗi nên độ phức tạp của thao tác này cũng là O(log(n)).

Như vậy độ phức tạp của một thao tác cập nhật hoặc truy vấn một đường đi trên cây sẽ là O(log(n)).

**Bài tập áp dụng**

[QTREE](http://vn.spoj.com/problems/QTREE)

[QTREE3](http://vn.spoj.com/problems/QTREE3/)

[QTREEX](http://vn.spoj.com/problems/QTREEX)

[EpicTree](https://www.hackerrank.com/contests/epiccode/challenges/epic-tree)

**Tham khảo**

<http://wcipeg.com/wiki/Heavy-light_decomposition>

<http://blog.anudeep2011.com/heavy-light-decomposition/>