

Centroid Decomposition

나정휘

<https://justicehui.github.io/>

목차

- Prerequisites
- Motivation
- Centroid Decomposition
- Centroid Tree
- Convert to Binary Tree

Prerequisites

Prerequisites

- DFS
- Divide and Conquer

Prerequisites

- Divide and Conquer
 - 수열에 대한 분할 정복
 - 수열의 모든 연속한 부분 수열을 고려하는 문제
 - ex. 히스토그램에서 가장 큰 직사각형, 구간 합이 K인 연속 부분 수열의 개수, ...
 - 해결 방법
 - 구간 $[l, r]$ 의 중간 지점 m 을 잡고
 - m 을 포함하는 모든 부분 수열을 처리
 - 고려하지 않은 부분 수열은 $[l, m-1]$ 또는 $[m+1, r]$ 에 속함
 - $T(n) = 2T(n/2) + f(n)$

Motivation

Motivation

- Divide and Conquer
 - 수열에서의 분할 정복이 효율적인 이유
 - 수열의 크기가 절반이 되도록 나눌 수 있음
 - $f(n) \in o(n^2)$ 이면 $T(n) = 2T(n/2) + f(n) \in o(n^2)$ 라서 $O(n^2)$ 보다 효율적
 - Big-O Notation과 Small-O Notation의 차이를 주의해야 함
 - 트리에서 모든 경로를 고려하는 문제
 - 트리에서도 분할 정복을 할 수 있을까?
 - 어떤 점을 기준으로 분할해야 하지?
 - 리프 정점은 어림도 없어 보이는데...
 - 1:n-1 분할
 - 루트 정점은 어떨까?
 - 루트의 차수가 1이라면...

Motivation

- Divide and Conquer
 - 트리에서의 분할 정복
 - 재귀 깊이가 $O(\log n)$ 이 되도록 보장하는 방법?
 - 정점을 제거했을 때 모든 컴포넌트의 크기가 상수 배 이하가 되어야 함
- Centroid
 - 제거했을 때 모든 컴포넌트의 크기가 절반 이하가 되는 정점
 - 항상 존재할까?
 - 항상 존재한다면 어떻게 찾을 수 있을까?
 - ...
 - 를 알아보시다.

질문?

Centroid

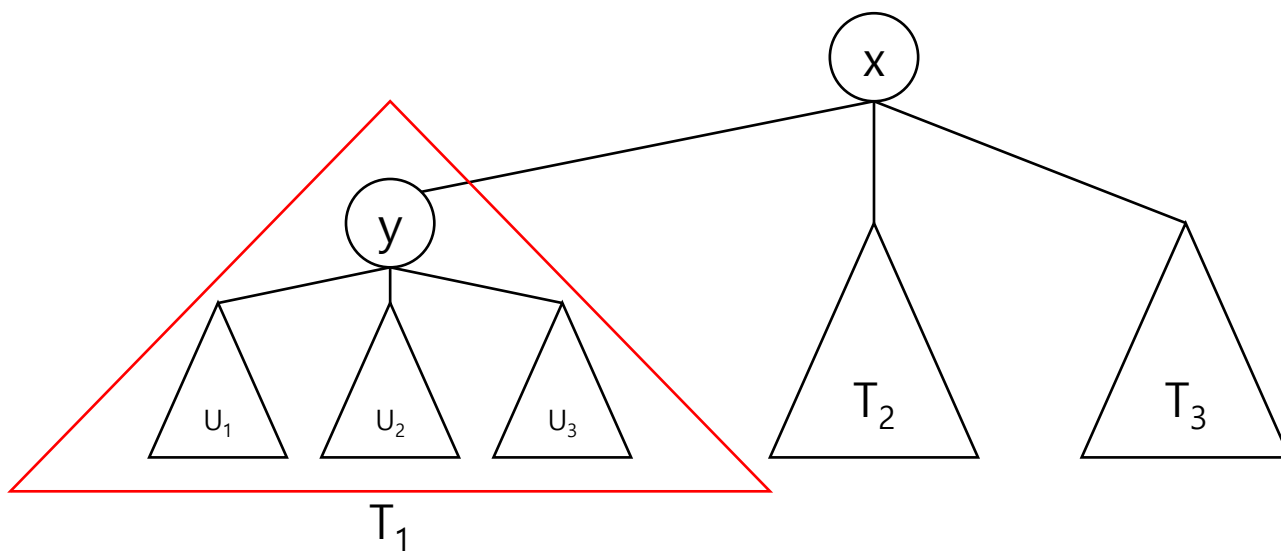
Centroid

- Centroid
 - 제거했을 때 모든 컴포넌트의 크기가 원본 트리의 절반 이하가 되는 정점
 - $\text{max_size}(v) := v$ 를 제거했을 때 나뉘지는 컴포넌트들의 크기 최댓값
 - $\text{max_size}(v) \leq n/2$ 이면 v 는 센트로이드
 - Lemma 1. 트리에서 센트로이드는 항상 존재한다.
 - 센트로이드를 찾는 매우 간단한 알고리즘
 - Lemma 2. 센트로이드가 여러 개 존재한다면, 서로 인접한 위치에 존재한다.
 - 센트로이드는 최대 2개 존재

Centroid

- Centroid

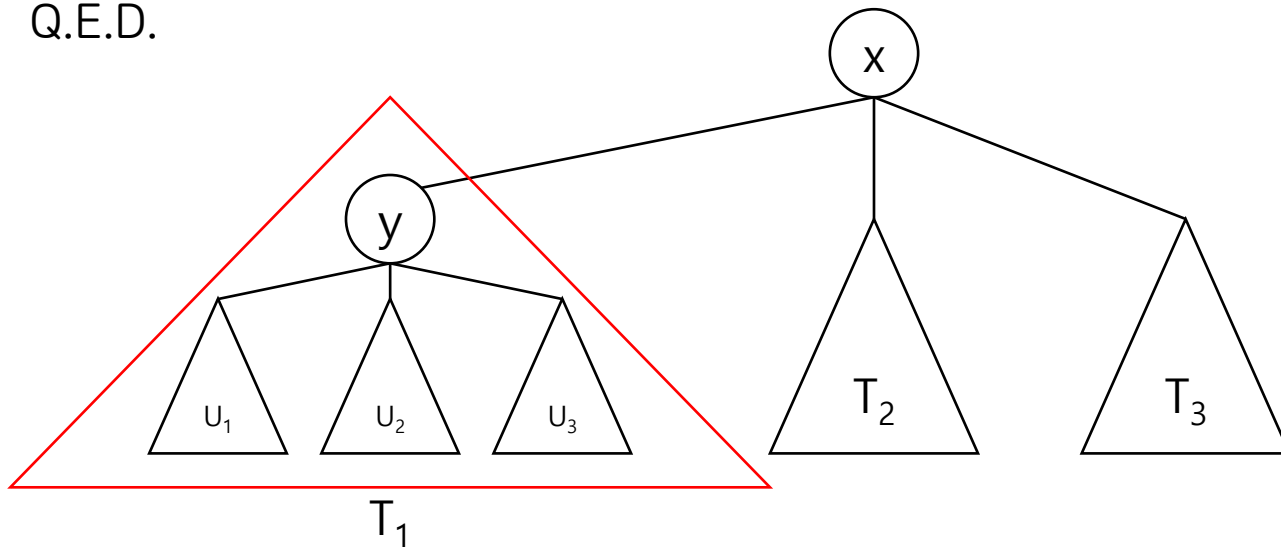
- Lemma 1. 트리에서 센트로이드는 항상 존재한다.
 - (귀류법) 센트로이드가 없는 크기 N 짜리 트리 T 가 있다고 하자.
 - T 에서 $\max_size(v)$ 가 최소인 정점을 x 라고 하자.
 - 가정에 의해 x 를 제거했을 때 쪼개지는 가장 큰 서브트리 T_1 은 $|T_1| > N/2$ 를 만족함
 - T_1 에 속하면서 x 와 인접한 정점 y 를 생각해 보자.



Centroid

- Centroid

- Lemma 1. 트리에서 센트로이드는 항상 존재한다.
 - T_1 에 속하면서 x 와 인접한 정점 y 를 생각해 보자.
 - $|T_1| > N/2$ 이므로 $|T - T_1| < N/2$ 를 만족함
 - y 밑에 있는 서브 트리 U_i 들의 크기는 $|T_1|$ 보다 작음
 - 따라서 $\text{max_size}(x) > \text{max_size}(y)$
 - $\text{max_size}(v)$ 가 최소인 정점이 x 라는 가정이 모순이므로 트리에는 항상 센트로이드가 존재
 - Q.E.D.



Centroid

- Centroid
 - Corollary 1. 아래 알고리즘은 항상 센트로이드를 찾는다.



```
vector<int> g[SZ]; // adjacent list
int size[SZ];

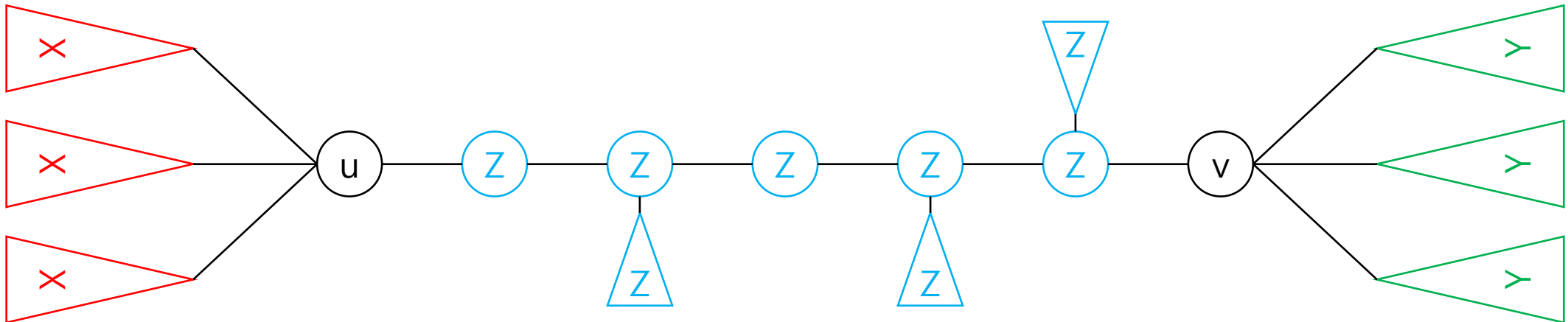
void getSize(int v, int b=-1){
    size[v] = 1;
    for(auto i : g[v]) if(i != b) size[v] += getSize(i, v);
    return size[v];
}

int getCent(int v, int sz=n, int b=-1){
    for(auto i : g[v]){
        if(i != b && size[i] * 2 > sz) return getCent(i, sz, v);
    }
    return v;
}
```

Centroid

- Centroid

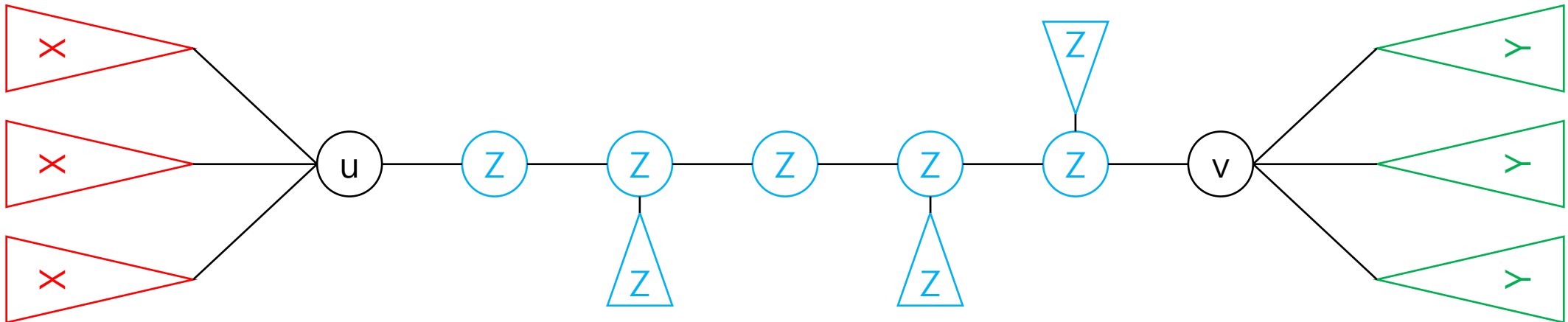
- Lemma 2. 센트로이드가 여러 개 존재한다면, 서로 인접한 위치에 존재한다.
 - 트리의 두 센트로이드를 u, v 라고 하자.
 - u 를 루트라고 할 때, 다음을 정의하자.
 - $X[i]$ = u 의 자식 정점을 루트로 하는 서브 트리의 크기, 단 v 를 포함하는 서브 트리는 제외
 - $Y[i]$ = v 의 자식 정점을 루트로 하는 서브 트리의 크기
 - Z = X, Y, u, v 에 포함되지 않은 다른 모든 정점들의 개수



Centroid

- Centroid

- Lemma 2. 센트로이드가 여러 개 존재한다면, 서로 인접한 위치에 존재한다.
 - N 개의 정점으로 구성된 트리이므로 $X + Y + Z + 2 = N$
 - u 가 센트로이드이므로 $Y + Z + 1 \leq N/2$
 - v 가 센트로이드이므로 $X + Z + 1 \leq N/2$
 - 두 식을 연립하면 $X + Y + 2Z + 2 = N + Z \leq N$
 - $Z \geq 0$ 이므로 $Z = 0$ 일 때만 부등식이 성립
 - 따라서 u 와 v 는 인접한 위치에 있어야 함
 - Q.E.D.



Centroid

- Centroid
 - Corollary 2. 센트로이드는 1개 또는 2개 존재한다.
 - 센트로이드는 항상 존재하므로 1개 이상 있음
 - 센트로이드가 3개 이상이면 인접하지 않은 두 정점이 존재하므로 2개 이하로 존재

Centroid

- 정리
 - 센트로이드: 제거했을 때 모든 컴포넌트의 크기를 $n/2$ 이하로 만드는 정점
 - 센트로이드는 DFS를 이용해 쉽게 구할 수 있음
 - 센트로이드는 1개 또는 2개 존재
 - 센트로이드가 2개 존재한다면 두 정점은 인접함

질문?

Centroid Decomposition

Centroid Decomposition

- BOJ 5820 경주 (IOI'11 D1P2 Race)
 - 간선의 가중치가 양수인 트리
 - 길이가 정확히 $K (\leq 10^6)$ 인 경로의 최소 간선 개수를 구하는 문제
- 수열에서는 어떻게 풀까?
 - $S[i] = A[1] + \dots + A[i]$
 - $0 \leq i < j \leq N$ 이면서 $S[j] - S[i] = K$ 를 만족하는 $j - i$ 의 최솟값을 구해야 함
- 중간 지점 m 고정, m 을 포함하는 모든 구간을 고려
- $j = m, m+1, \dots, r$ 을 보면서
- $S[l..m-1]$ 에서 $S[i] = S[j] - K$ 인 i 의 최댓값을 구하면 됨
- map 사용하면 $T(n) = 2T(n/2) + O(n \log n) = O(n \log^2 n)$ 에 해결 가능
- $K \leq 10^6$ 임을 이용하면 $T(n) = 2T(n/2) + O(n) = O(n \log n)$ 에 해결 가능

Centroid Decomposition

- BOJ 5820 경주 (IOI'11 D1P2 Race)
 - 센트로이드 c 를 잡아서, c 를 지나는 모든 경로를 처리하자.
 - 다른 경로는 c 를 제거해서 생기는 컴포넌트들에 대해 재귀적으로 처리하면 됨
 - $D[x]$: c 에서 x 까지의 거리, DFS를 이용해 $O(n)$ 에 구할 수 있음
 - $D[u] + D[v] = K$ 를 만족하면서 서로 다른 서브 트리에 속하는 u, v 를 모두 확인하면 됨
 - $V[x]$: c 와의 거리가 x 인 최소 깊이 ($x \leq K$, 각 서브 트리마다 DFS로 전처리)
 - 다른 서브 트리에서 c 와 $K-x$ 만큼 떨어진 정점이 있다면
 - (현재 정점의 깊이) + (c 와 $K-x$ 만큼 떨어진 정점의 최소 깊이)로 정답 갱신
 - $\text{Answer} \leftarrow \text{dep} + V[K-x]$
 - 센트로이드가 고정되었을 때 $O(N)$, 전체 $O(N \log N)$ 에 해결할 수 있음

Centroid Decomposition

- BOJ 5820 경주 (IOI'11 D1P2 Race)
 - 구현 팁
 - 센트로이드 자체도 하나의 서브 트리로 취급하면 편함
 - 서브 트리의 정보를 병합하는 방식으로 구현
 - T_1 : 이미 처리한 서브 트리들의 정보를 모두 합친 것
 - T_2 : 이번에 처리하는 서브 트리
 - T_1 과 T_2 에서 각각 정점을 하나씩 뽑은 경로를 고려
 - 이후 T_2 의 정보를 T_1 에 더함
 - 아직 처리하지 않은 서브 트리 T_2 를 잡고 반복

```

#include <bits/stdc++.h>
using namespace std;
constexpr int INF = 0x3f3f3f3f;

int N, K, R = INF, S[202020], U[202020], V[1010101];
vector<pair<int,int>> G[202020];
vector<int> Used;

int GetSize(int v, int b=-1){
    S[v] = 1;
    for(auto [i,w] : G[v]) if(i != b && !U[i]) S[v] += GetSize(i, v);
    return S[v];
}

int GetCent(int v, int sz, int b=-1){
    for(auto [i,w] : G[v]) if(i != b && !U[i] && S[i]*2 > sz) return GetCent(i, sz, v);
    return v;
}

void Update(int v, int b, int dst, int dep){
    if(dst > K) return;
    V[dst] = min(V[dst], dep); Used.push_back(dst);
    for(auto [i,w] : G[v]) if(i != b && !U[i]) Update(i, v, dst+w, dep+1);
}

void Apply(int v, int b, int dst, int dep){
    if(dst > K) return;
    R = min(R, V[K-dst] + dep);
    for(auto [i,w] : G[v]) if(i != b && !U[i]) Apply(i, v, dst+w, dep+1);
}

void Solve(int v){
    v = GetCent(v, GetSize(v));
    U[v] = 1; V[0] = 0; Used.push_back(0);
    for(auto [i,w] : G[v]) if(!U[i]) Apply(i, v, w, 1), Update(i, v, w, 1);
    for(auto i : Used) V[i] = INF; Used.clear();
    for(auto [i,w] : G[v]) if(!U[i]) Solve(i);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1,u,v,w; i<N; i++){
        cin >> u >> v >> w; u++; v++;
        G[u].emplace_back(v, w);
        G[v].emplace_back(u, w);
    }
    memset(V, 0x3f, sizeof V);
    Solve(1);
    cout << (R < INF ? R : -1);
}

```


질문?

Centroid Tree

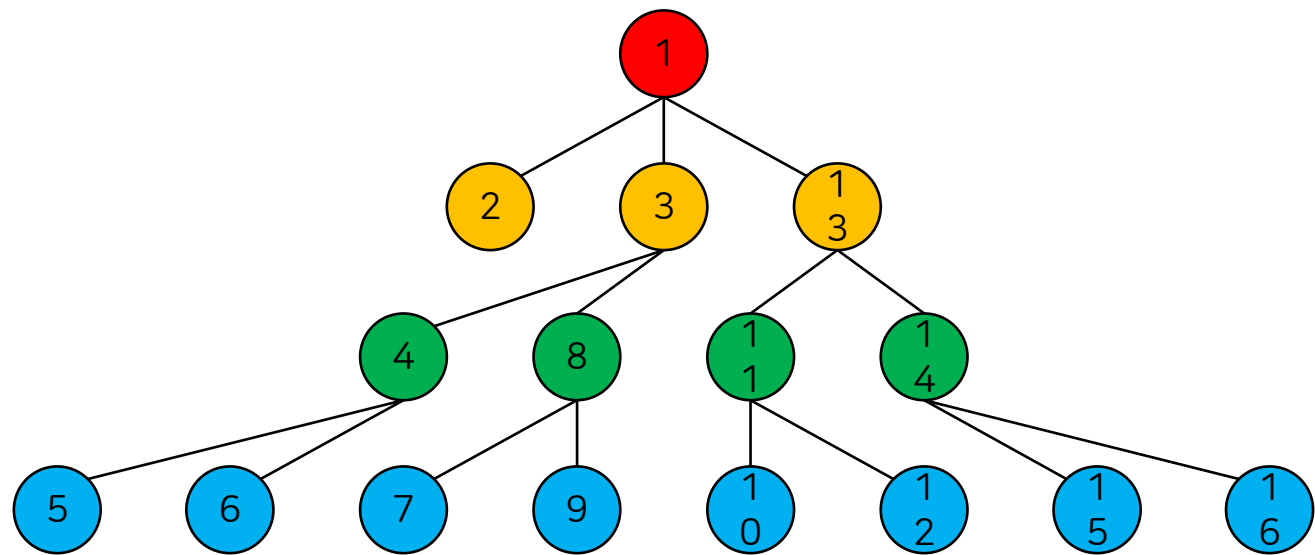
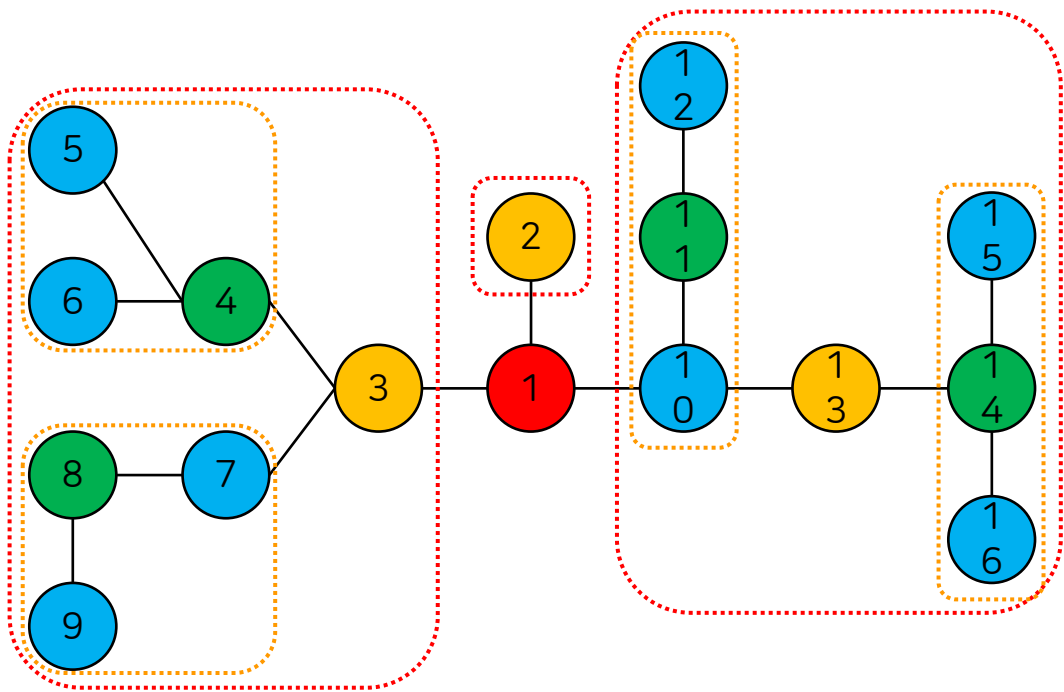
Centroid Tree

- Centroid Tree
 - 이진 탐색 트리: 이진 탐색 과정을 트리로 나타낸 것
 - 머지 sort 트리: 합병 정렬 과정을 트리로 나타낸 것
 - 세그먼트 트리: 분할 정복 과정을 트리로 나타낸 것
 - 센트로이드 트리: ???

Centroid Tree

- Centroid Tree
 - 이진 탐색 트리: 이진 탐색 과정을 트리로 나타낸 것
 - 머지 sort 트리: 합병 정렬 과정을 트리로 나타낸 것
 - 세그먼트 트리: 분할 정복 과정을 트리로 나타낸 것
 - 센트로이드 트리: 센트로이드 분할 과정을 트리로 나타낸 것

Centroid Tree



Centroid Tree

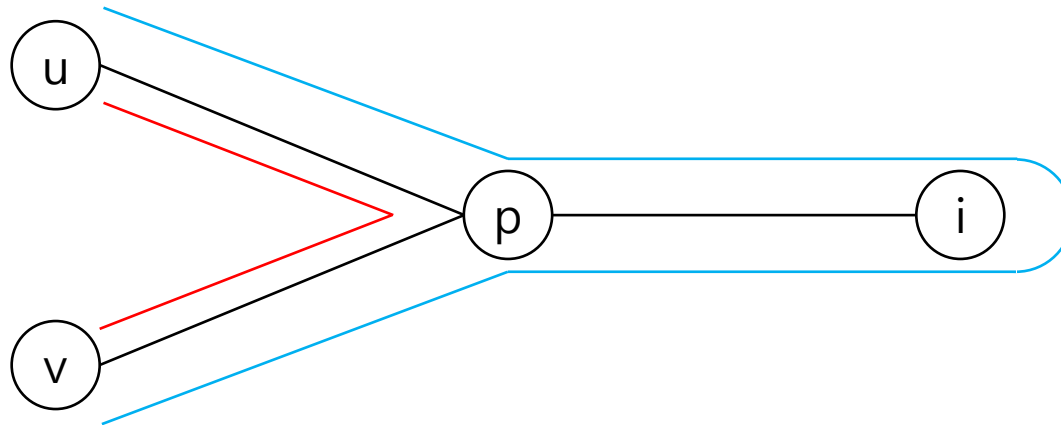
- Centroid Tree
 - 트리의 높이는 $O(\log N)$
 - 매번 크기가 절반 이하로 감소
 - 원본 트리의 $u - v$ 경로는 센트로이드 트리에서의 $LCA(u, v)$ 에 의해 2개로 분할됨
 - $p = LCA_c(u, v)$ 라고 하면, $u - p$ 경로와 $p - v$ 경로에 중복 정점이 없음 (p 제외)
 - 센트로이드 트리에서 v 의 조상을 P_1, \dots, P_k 라고 하면 v 에서 출발하는 경로를 $k+1$ 가지로 분류할 수 있음
 - LCA 가 v 인 경로, LCA 가 P_1 인 경로, LCA 가 P_2 인 경로, ...
 - 센트로이드 트리의 각 정점마다 서브 트리에 있는 정점들의 정보를 저장하면
 - 어떤 정점 v 를 지나는 모든 경로를 고려하는 대신 v 의 조상 정점($k \leq \log N$ 개)의 서브 트리만 확인하면 됨

Centroid Tree

- BOJ 13514 트리와 쿼리 5
 - 1 v : 정점 v 의 색을 바꾸는 쿼리 (흰색/검은색)
 - 2 v : v 와 가장 가까운 흰색 정점의 거리를 구하는 문제
- v 에서 출발하는 모든 경로를 확인하는 대신, 센트로이드 트리에서 v 의 조상을 확인
 - 센트로이드 트리의 각 정점에서 multiset을 관리하자.
 - $S[i]$: i 를 루트로 하는 서브 트리에 있는 흰색 정점까지의 거리
 - 거리는 원본 트리 상에서의 거리, LCA로 구할 수 있음
- 센트로이드 트리에서 v 의 조상을 보면서, $\text{Dist}(v, i) + *S[i].begin()$ 의 최솟값을 구하면 됨

Centroid Tree

- BOJ 13514 트리와 쿼리 5
 - multiset의 원소를 확인할 때 서로 다른 서브 트리에서 왔는지 확인해야 할까?
 - 최솟값을 구하는 문제이므로 확인할 필요 없음
 - 같은 서브 트리에서 왔다면 u 와 v 를 분할하는 다른 센트로이드(p)에서 더 작은 값을 가져간 상태
 - 만약 최댓값을 구하는 문제라면 서로 다른 서브 트리에서 왔는지 확인해야 함



Centroid Tree

```
void Build(int v, int b=-1){
    v = GetCent(v, GetSize(v));
    U[v] = 1; CP[v] = b;
    for(auto i : G[v]) if(!U[i]) Build(i, v);
}

void Update(int v){
    A[v] ^= 1;
    for(int i=v; i!=-1; i=CP[i]){
        if(A[v]) ST[i].emplace(Dist(v, i), v);
        else ST[i].erase(make_pair(Dist(v, i), v));
    }
}

int Query(int v){
    int res = INF;
    for(int i=v; i!=-1; i=CP[i]) if(!ST[i].empty()) res = min(res, Dist(v, i) + ST[i].begin()->first);
    return res < INF ? res : -1;
}
```

Centroid Tree

- BOJ 13513 트리와 쿼리 4
 - 가장 먼 두 흰색 정점의 거리를 구하는 문제
 - v 의 조상 정점을 루트로 하는 서브 트리의 정점들을 볼 때
 - v 와 다른 곳에서 유래한 정점만 확인해야 함
 - $S1[i]$ = i 를 루트로 하는 서브 트리에 속한 정점들의 거리
 - $S2[i]$ = i 의 자식을 루트로 하는 서브 트리에서 가장 멀리 있는 정점들을 모은 것
 - $S1[c_1]$ 의 최댓값, $S1[c_2]$ 의 최댓값, ..., i
 - 어떤 자식 정점 밑에 있는 건지 함께 저장
 - $S2[i]$ 에서 가장 큰 값 / 두 번째로 큰 값만 보면 됨

질문?

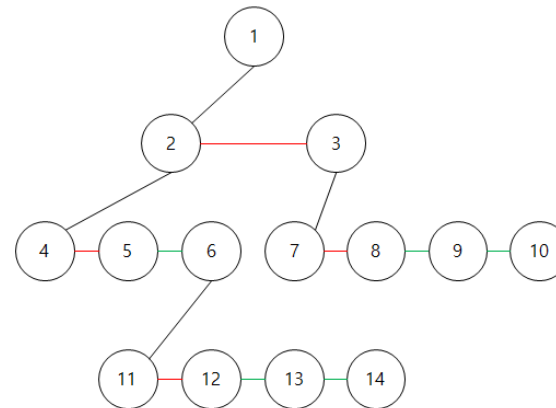
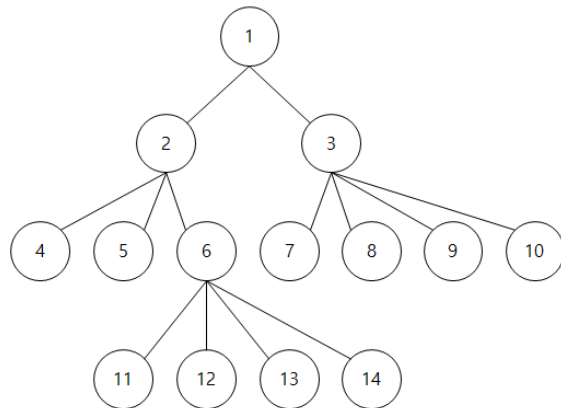
Convert to Binary Tree

Convert to Binary Tree

- Convert to Binary Tree
 - 이진 트리의 장점
 - 모든 정점의 차수가 3 이하
 - 센트로이드 분할 과정에서 $O(\deg^2)$ 만큼의 시간이 걸려도 상수로 취급할 수 있음
 - `for(auto i : G[v]) for(auto j : G[v])`
 `if(!U[i] && !V[i]) f(i, j);`

Convert to Binary Tree

- Convert to Binary Tree
 - 모든 트리는 N-1개의 정점을 추가해서 조상-자손 관계가 동일한 이진 트리로 바꿀 수 있음
- 트리를 표현하는 방법
 - 계층 구조
 - 그래프의 인접 리스트로 표현
 - left child right sibling
 - 한 포인터는 가장 왼쪽 자식, 다른 포인터는 오른쪽 형제



Convert to Binary Tree

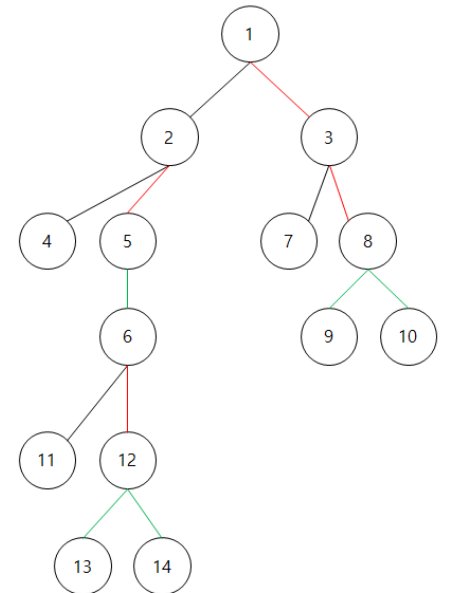
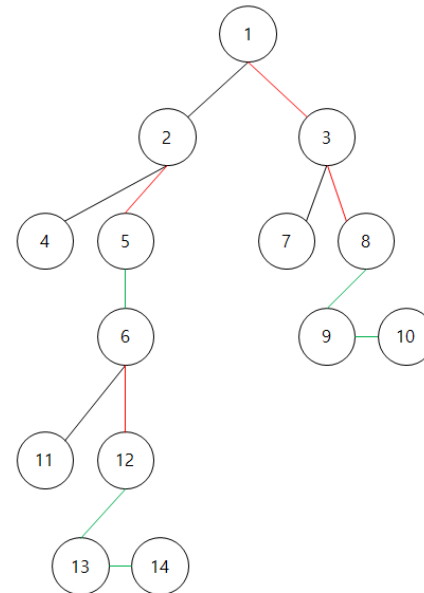
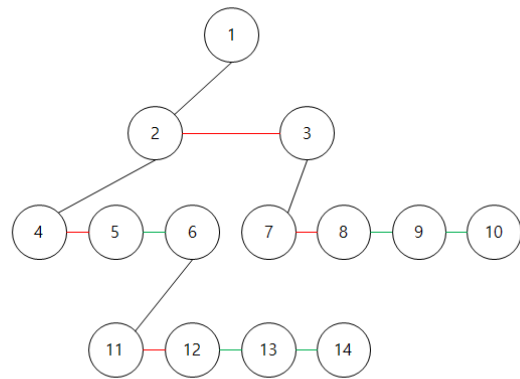
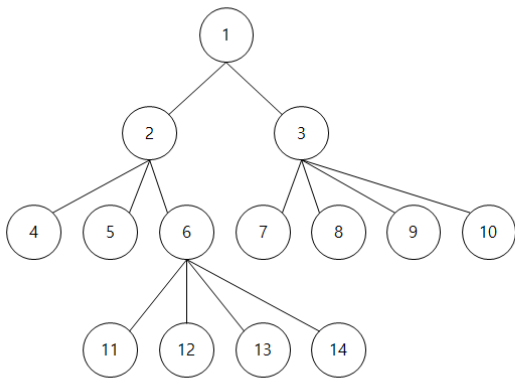
- Convert to Binary Tree

- left child right sibling

- 초록색 간선(두 번째 자식 이후 오른쪽 간선)을 형제 정점의 자식으로 넣어도 조상-자손 관계 유지

- 형제 관계는 어떻게 유지할 수 있을까?

- 거리는 어떻게 유지할 수 있을까?

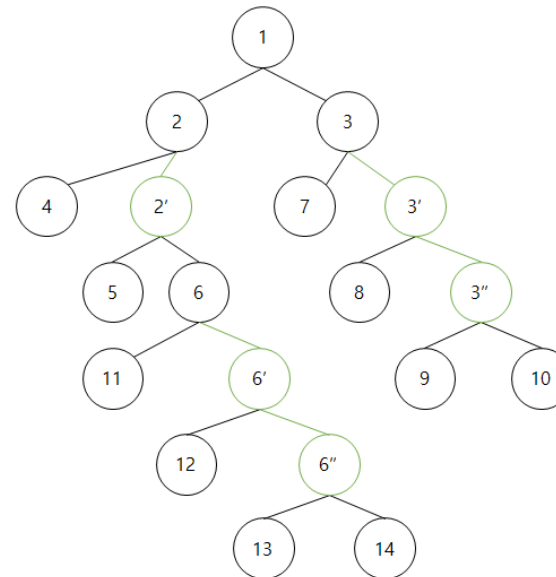
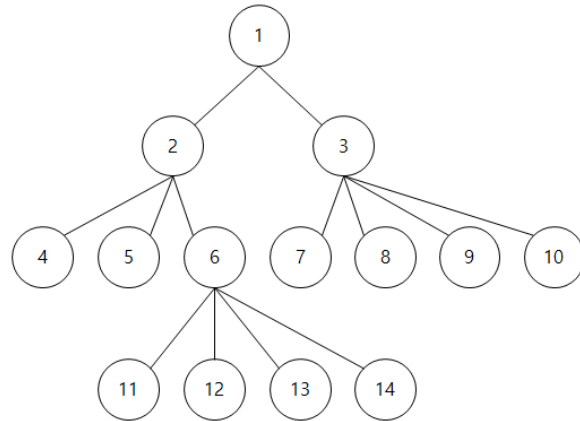


Convert to Binary Tree

- Convert to Binary Tree

- 형제 관계/거리 유지

- right sibling 간선을 밑으로 내릴 때마다 더미 정점 추가
 - 더미 정점은 최대 N-1개 추가됨
 - 연두색 간선 가중치 0으로 하면 거리 관계 유지 가능



Convert to Binary Tree

```

● ● ●

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

int N;
vector<pair<int,int>> Inp[101010], G[202020];

// original vertex, virtual vertex, back, adjacent list idx
void make_binary(int ori=1, int vir=1, int b=-1, int idx=0){
    for(int x=idx; x<Inp[ori].size(); x++){
        auto [i,w] = Inp[ori][x]; if(i == b) continue;
        if(G[vir].empty() || x+1 == Inp[ori].size() || x+2 == Inp[ori].size() && Inp[ori][x+1].first == b){
            G[vir].emplace_back(i, w);
            make_binary(i, i, ori);
            G[i].emplace_back(vir, w);
            continue;
        }
        int nxt = ++N;
        G[vir].emplace_back(nxt, 0);
        make_binary(ori, N, b, x);
        G[nxt].emplace_back(vir, 0);
        break;
    }
}
```

질문?

Convert to Binary Tree

- BOJ 17969 Gene Tree (2019 ICPC Seoul Regional B)
 - 모든 리프 정점 쌍에 대해, 거리 제공의 합을 구하는 문제
 - 이진 트리만 주어지므로 굳이 바꿀 필요는 없음
 - BOJ 16121 사무실 이전 문제는 이진 트리로 바꾸면 똑같은 문제
- 센트로이드 c 를 지나는 경로를 모두 처리하자.
 - 두 서브 트리 A, B에서 리프 정점을 하나씩 골라서 경로를 만들었을 때 거리 제공의 합
 - $a1$ = A의 리프 정점 깊이 합
 - $a2$ = A의 리프 정점 깊이 제공 합
 - $a3$ = A의 리프 정점 개수
 - $b1, b2, b3$ 도 똑같이 정의하면
 - 정답은 $(a3 * b2) + (b3 * a2) + 2 * a1 * b1$ 만큼 증가
 - c 와 인접한 모든 정점 쌍(≤ 6 가지) 모두 고려해도 됨



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

struct Info{
    ll s1, s2, cnt;
    Info() : Info(0, 0, 0) {}
    Info(ll v) : Info(v, v*v, 1) {}
    Info(ll s1, ll s2, ll cnt) : s1(s1), s2(s2), cnt(cnt) {}
    ll calc(const Info &t) const { return cnt * t.s2 + t.cnt * s2 + 2 * s1 * t.s1; }
    Info& operator += (const Info &t) { s1 += t.s1; s2 += t.s2; cnt += t.cnt; return *this; }
    Info operator + (const Info &t) const { return Info(*this) += t; }
};

int N, S[101010], U[101010]; ll R;
vector<pair<ll,ll>> G[101010];

int GetSize(int v, int b=-1){
    S[v] = 1;
    for(auto [i,w] : G[v]) if(i != b && !U[i]) S[v] += GetSize(i, v);
    return S[v];
}

int GetCent(int v, int sz, int b=-1){
    for(auto [i,w] : G[v]) if(i != b && !U[i] && S[i]*2 > sz) return GetCent(i, sz, v);
    return v;
}

Info DFS(int v, int b, int d){
    Info res;
    if(G[v].size() == 1) res += d;
    for(auto [i,w] : G[v]) if(i != b && !U[i]) res += DFS(i, v, d + w);
    return res;
}

void Solve(int v){
    v = GetCent(v, GetSize(v)); U[v] = 1;
    vector<Info> ch;
    if(G[v].size() == 1) ch.emplace_back(0);
    for(auto [i,w] : G[v]) if(!U[i]) ch.push_back(DFS(i, -1, w));
    for(int i=0; i<ch.size(); i++) for(int j=0; j<i; j++) R += ch[i].calc(ch[j]);
    for(auto [i,w] : G[v]) if(!U[i]) Solve(i);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1,s,e,w; i<N; i++) cin >> s >> e >> w, G[s].emplace_back(e, w), G[e].emplace_back(s, w);
    Solve(1);
    cout << R;
}
```

질문?