고급 9회차
Centroid Decomposition

서강대학교 전해성(seastar105)

Divide and Conquer in Tree

일반적으로 직선형의 배열이 주어졌을 때 분할정복의 단계는 보통 이렇다.

1. 배열을 절반으로 나눈다. (Divide)
2. 나눠진 부분에 대해서 먼저 원하는 과정을 수행한다. (Conquer)
3. 나눠진 부분들에 대해서 합쳐주는 과정을 수행한다. (Merge)

이러한 과정을 통해서 매우 빠른 시간복잡도(log)를 가질 수 있게 된다.

직선형 배열이 아닌 트리에서도 분할정복이 가능하지 않을까?

What is Centroid?

Centroid of Tree

정점이 N개인 트리가 있다고 하자.

어떤 정점을 제거했을 때 생기는 모든 서브트리들의 크기가 $\frac{N}{2}$이하라면, 그 정점을 Centroid라고 부른다.

Centroid는 어느 트리에서나 존재한다.

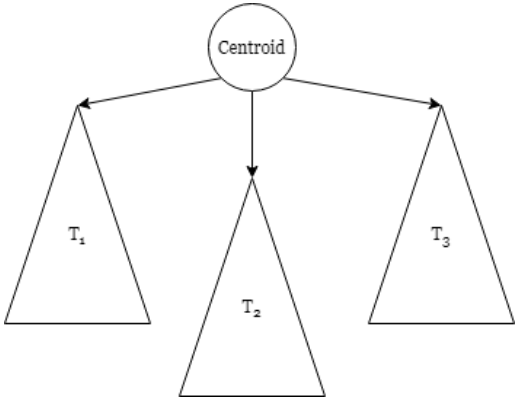
How to find Centroid?

트리가 주어졌을 때, 임의의 정점을 루트로 잡고 각 정점을 루트로 하는 서브트리의 크기는 DFS 한 번에 전처리가 가능하다.

그러면 아래와 같은 pseudocode를 통해 센트로이드를 찾는 것이 가능하다.

N은 트리 크기.

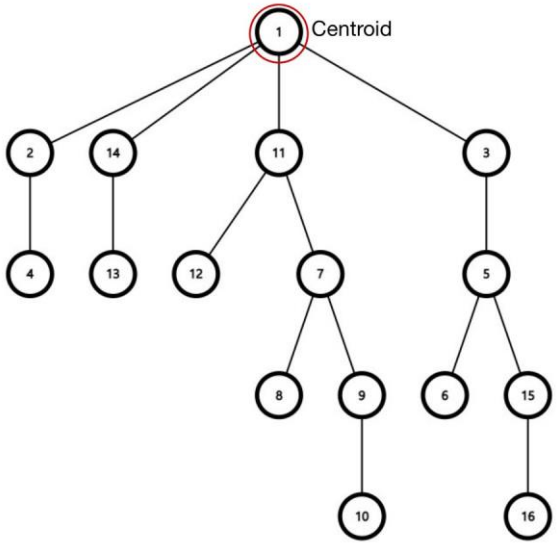
```
def get_centroid(cur_node, N):
    for child_node of cur_node:
        if sz[child_node] > N/2:
            return get_centroid(child_node, N)
    return cur_node
```



Size of T_1 <= N/2
Size of T_2 <= N/2
Size of T_3 <= N/2

Centroid Tree

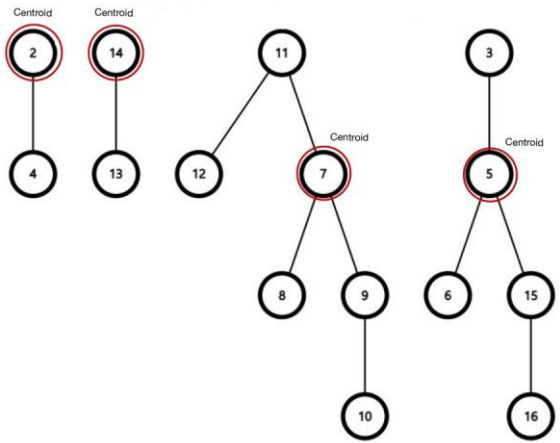
하나의 트리에서 센트로이드를 계속 찾아가면서 센트로이드끼리 연결시켜준 트리를 Centroid Tree라고 부른다.

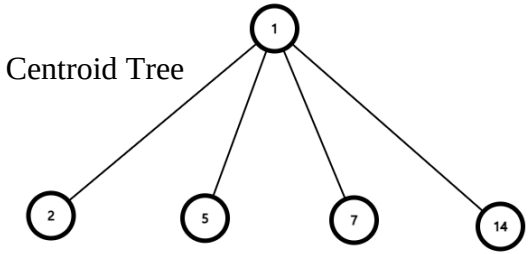


Centroid Tree

Centroid Tree

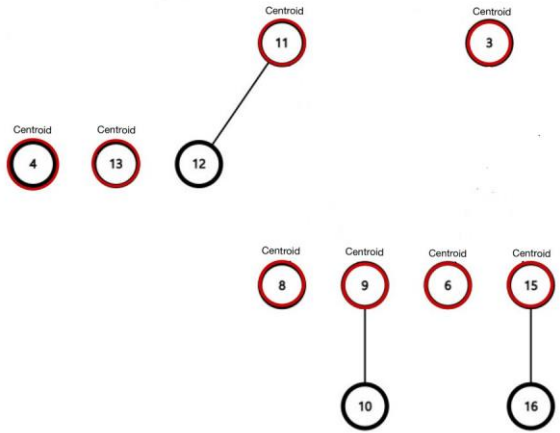
하나의 트리에서 센트로이드를 계속 찾아가면서 센트로이드끼리 연결시켜준 트리를 Centroid Tree라고 부른다.

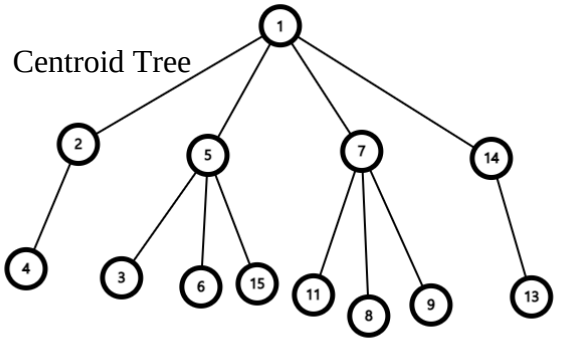




Centroid Tree

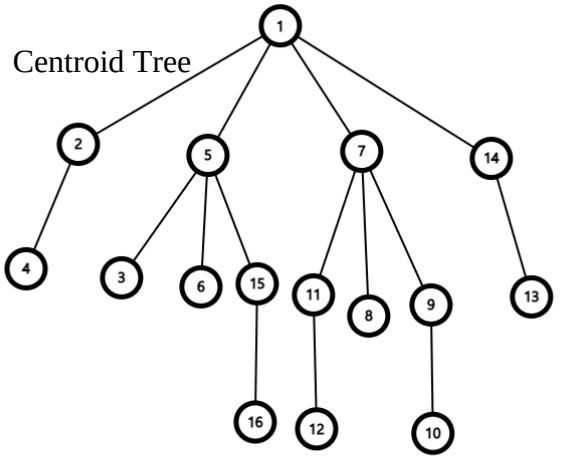
하나의 트리에서 센트로이드를 계속 찾아가면서 센트로이드끼리 연결시켜준 트리를 Centroid Tree라고 부른다.





Centroid Tree

하나의 트리에서 센트로이드를 계속 찾아가면서 센트로이드끼리 연결시켜준 트리를 Centroid Tree라고 부른다.



Property of Centroid Tree

1. Centroid Tree의 높이는 O(logN)이 된다.

센트로이드를 한번 지울 때마다 나눠지는 트리의 크기는 $\frac{N}{2}$이하가 됩니다.

따라서 센트로이드는 O(logN)번 만큼 찾게 됩니다.

그리고 센트로이드 트리의 높이는 센트로이드를 찾은 횟수가 되므로 높이가 O(logN)임을 알 수 있습니다.

Property of Centroid Tree

2. 원래 트리에서의 u, v간의 경로는 센트로이드 트리에서 u->LCA(u,v)->v로 표현이 가능하다. 여기서 LCA(u,v)는 **센트로이드 트리에서의 LCA.**

LCA(u,v)를 w라고 합시다. u, v간의 경로가 u->w->v로 표현이 불가능하다고 가정합니다.
즉, w가 u, v간의 경로 상에 없다고 가정합니다.
그렇다면 w를 제거해도 u, v는 원래의 트리에서 서로 같은 서브트리에 있어야 합니다.
이는 w가 센트로이드 트리에서의 LCA이기 때문에 모순됩니다.

Divide and Conquer in Tree

지금까지 배운 센트로이드를 이용해서 트리에서의 분할정복을 수행하는 과정을 보통 Centroid Decomposition이라고 부릅니다.

예를 들어서 트리에서의 모든 경로를 고려해줘야 하는 문제가 있을 때, $O(N^2)$의 경로를 전부 고려해줘야 합니다.
이럴 때 센트로이드를 찾고(Divide) 각 서브트리들의 문제를 해결한 뒤에(Conquer) 서브트리간의 결과를 합쳐주는 과정(Merge)으로 진행할 수 있습니다.

Time Complexity of Centroid Decomposition

Centroid Decomposition을 통해서 Centroid Tree를 만드는 과정의 시간복잡도를 계산해봅시다.

Centroid Tree의 높이는 O(logN)이었습니다. 그리고 각 서브트리에서 센트로이드를 찾는 것은 DFS 두 번으로 가능하니 O(N)입니다. 따라서, 총 시간복잡도는 O(NlogN)이 됩니다.

여기서, Conquer나 Merge과정의 시간복잡도를 곱하면 총 시간복잡도가 됩니다.

20297번 - Confuzzle

문제 요약

크기 N인 트리가 주어지는데 각 정점에는 번호가 적혀있고 간선의 길이는 모두 1입니다.

이 때, 같은 번호가 적혀 있는 정점간의 길이 중 가장 짧은 길이를 출력해야 합니다.

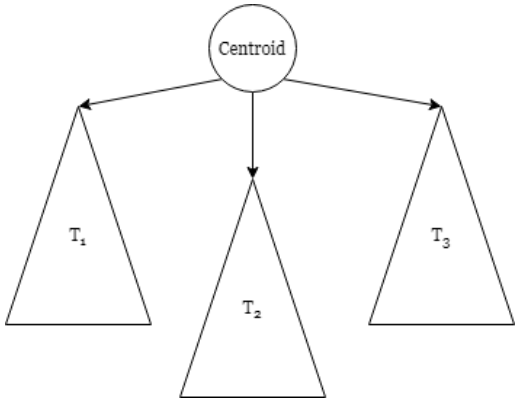
나이브하게 풀면 모든 경로를 봐야 되기 때문에 $O(N^2)$으로 시간초과가 납니다.

이를 Centroid Decomposition으로 풀어보죠.

20297번 - Confuzzle

센트로이드를 찾았다고 생각합시다.

이 트리에서 경로들은 Centroid를 지나는 경로와 Centroid를 지나지 않는 경로로 구분이 가능합니다.

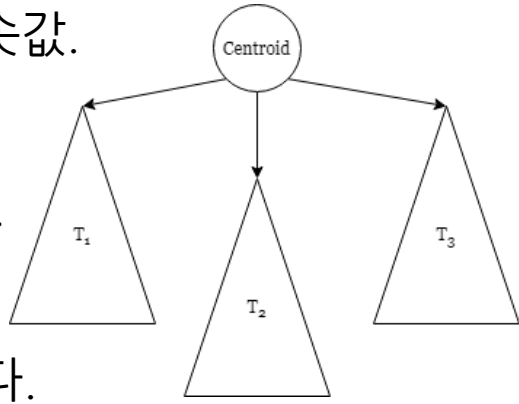


Centroid를 지나는 경로를 처리해주는 과정만 잘 구상하면 똑같은 과정을 T_1, T_2, T_3와 같은 서브트리에 대해서도 수행하면 우리가 원하는 결과를 얻을 수 있습니다.

20297번 - Confuzzle

min_depth[i]라는 값을 정의합시다.

min_depth[i] : 센트로이드로부터 i가 적힌 정점까지의 거리 중 최솟값.



이를 구하는 것은 각 서브트리를 한 번 순회하는 것으로 가능합니다.

그리고 순회함과 동시에 다른 서브트리에서 min_depth[i]가

갱신되었다면 번호가 같은 정점 쌍을 찾은 것이므로 답을 갱신합니다.

이 과정을 통해 Centroid를 지나는 경로들을 모두 볼 수 있습니다.

20297번 - Confuzzle

현재 보고 있는 서브트리의 크기를 N이라고 합시다.

센트로이드를 찾는 것은 O(N), 각 서브트리를 순회하는 것도 O(N)이 걸립니다.

따라서, 총 Centroid Decomposition의 시간복잡도는 O(NlogN)이 됩니다.

이제 실제 코드로 어떻게 구현되는지 살펴봅시다.

20297번 - Confuzzle

```
int get_size(int cur, int par) {
    sub_sz[cur] = 1;
    for(const int &v:G[cur]) {
        if(v == par || vis[v]) continue;
        sub_sz[cur] += get_size(v, cur);
    }
    return sub_sz[cur];
}

int get_cent(int cur, int par, int thr) {
    for(const int &v:G[cur]) {
        if(v == par || vis[v]) continue;
        if(sub_sz[v] > thr) return get_cent(v, cur, thr);
    }
    return cur;
}
```

get_size는 서브트리의 크기들을 계산하는 함수입니다.

get_cent가 cur를 루트로 하는 서브트리에서 centroid를 찾는 함수입니다.

vis배열이 중요한데, 센트로이드를 찾으면 트리에서 지워줘야 하는데 이를 vis배열로 표시합니다.

20297번 - Confuzzle

```
int dnc(int cur) {
    int thr = get_size(cur, 0);
    int ct = get_cent(cur, 0, thr/2);
    for(const int &v:cur_tree) min_depth[v] = INF;
    cur_tree.clear();
    vis[ct] = true;
    min_depth[a[ct]] = 0;
    cur_tree.push_back(a[ct]);
    int ret = INF;
    for(const int &v:G[ct]) {
        if(!vis[v]) {
            ret = min(ret, query(v, ct, 1));
            update(v, ct, 1);
        }
    }
    for(const int &v:G[ct]) {
        if(!vis[v]) ret = min(ret, dnc(v));
    }
    return ret;
}
```

dnc가 cur를 루트로 하는 서브트리에서 Centroid를 지나는 경로들을 처리하는 함수가 됩니다.

query와 update함수가 centroid의 서브트리 정보들을 갱신하고 취합하는 함수가 됩니다.

경로를 다 고려해준 후에는 각 서브트리에 대해서도 dnc를 호출해줍니다.

20297번 - Confuzzle

```
int query(int cur, int par, int depth) {
    int ret = INF;
    if(min_depth[a[cur]] != INF) ret = min(ret, min_depth[a[cur]]+depth);
    for(const int &v:G[cur]) {
        if(v == par || vis[v]) continue;
        ret = min(ret, query(v, cur, depth+1));
    }
    return ret;
}

void update(int cur, int par, int depth) {
    min_depth[a[cur]] = min(min_depth[a[cur]], depth);
    cur_tree.push_back(a[cur]);
    for(const int &v:G[cur]) {
        if(v == par || vis[v]) continue ;
        update(v, cur, depth+1);
    }
}
```

update는 서브트리를 순회하며 min_depth를 갱신하고, query는 같은 번호가 적힌 정점쌍을 찾으면 답을 갱신합니다.

“Any Question?”