Centroid Decomposition

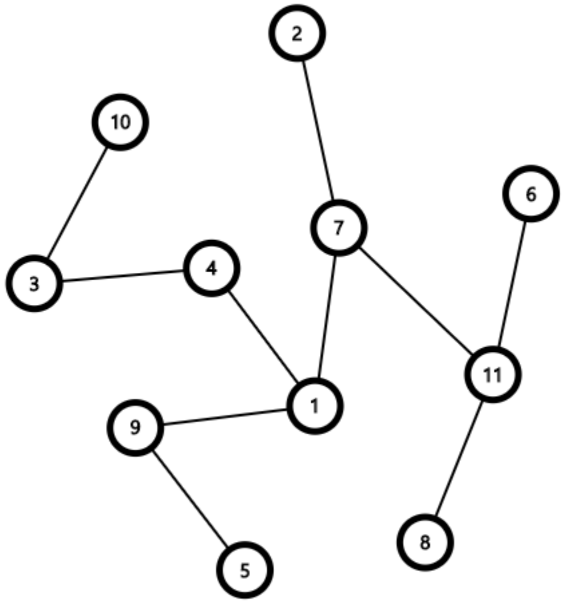
Sogang ICPC Team
2020 Spring 20141574 임지환

Contents

- 트리의 중심(centroid)

$|V| = N$인 트리에서

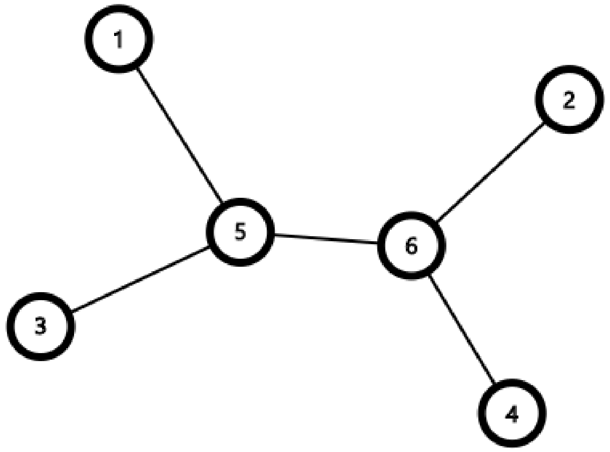
어떤 정점 v를 제거하였을 때 나뉘어진 각 subtree의 크기가 모두 $\frac{N}{2}$이하가 되게 하는 정점



- 트리의 중심(centroid)

$|V| = N$인 트리에서

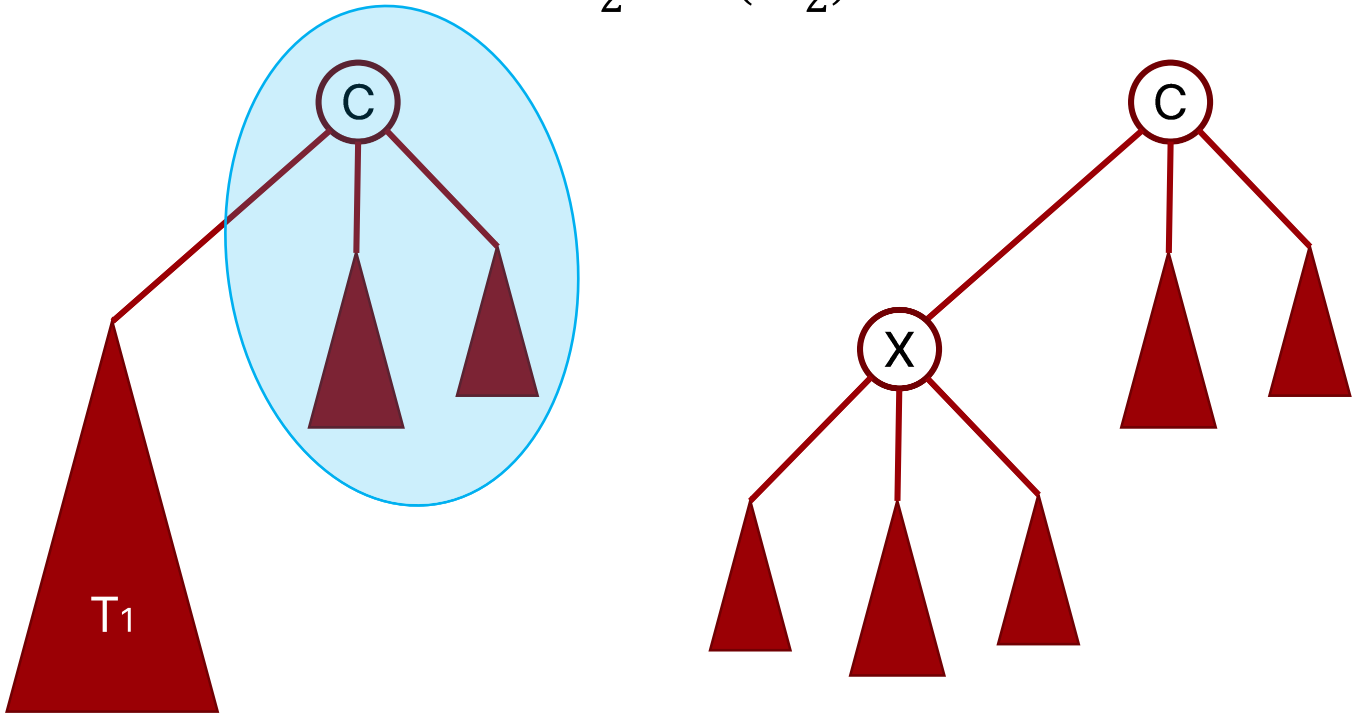
어떤 정점 v를 제거하였을 때 나뉘어진 각 subtree의 크기가 모두 $\frac{N}{2}$이하가 되게 하는 정점



- Proof of Existence of the Centroid on trees

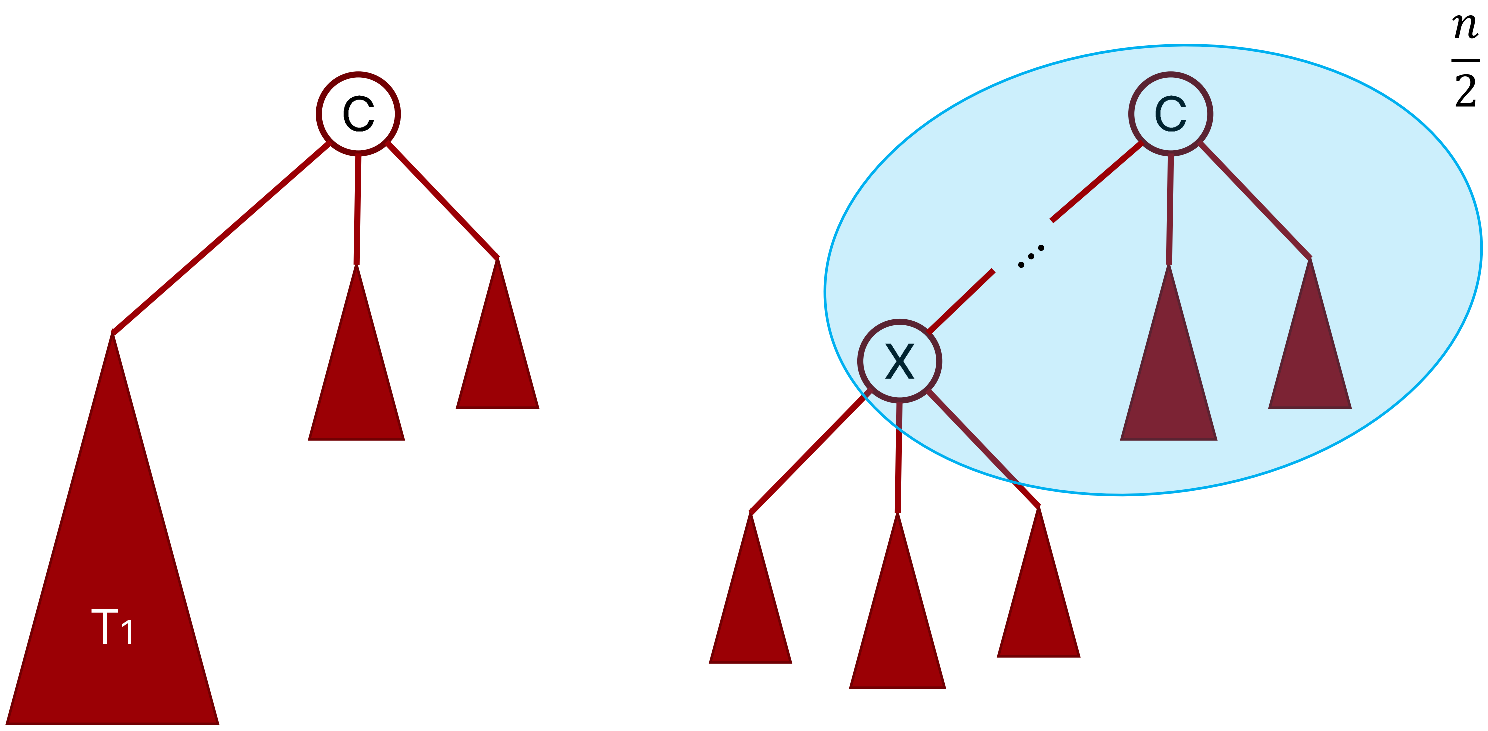
Let $|T_1| > \frac{n}{2} + k \quad (k > 0)$,

$$n - |T_1| = \frac{n}{2} - k \left(< \frac{n}{2}\right)$$

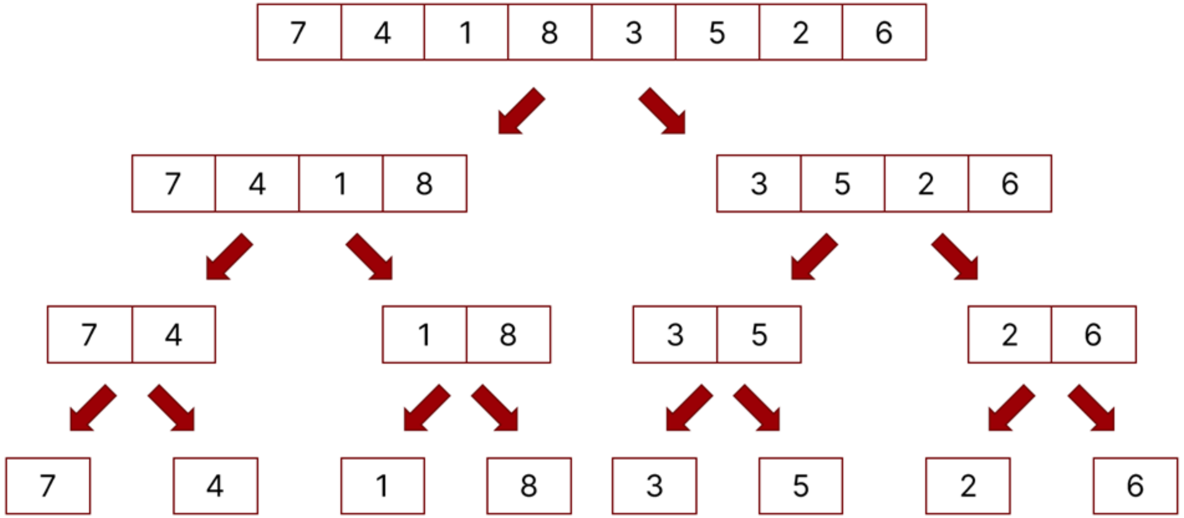


- Proof of Existence of the Centroid on trees

Let $|T_1| > \frac{n}{2} + k \quad (k > 0)$,



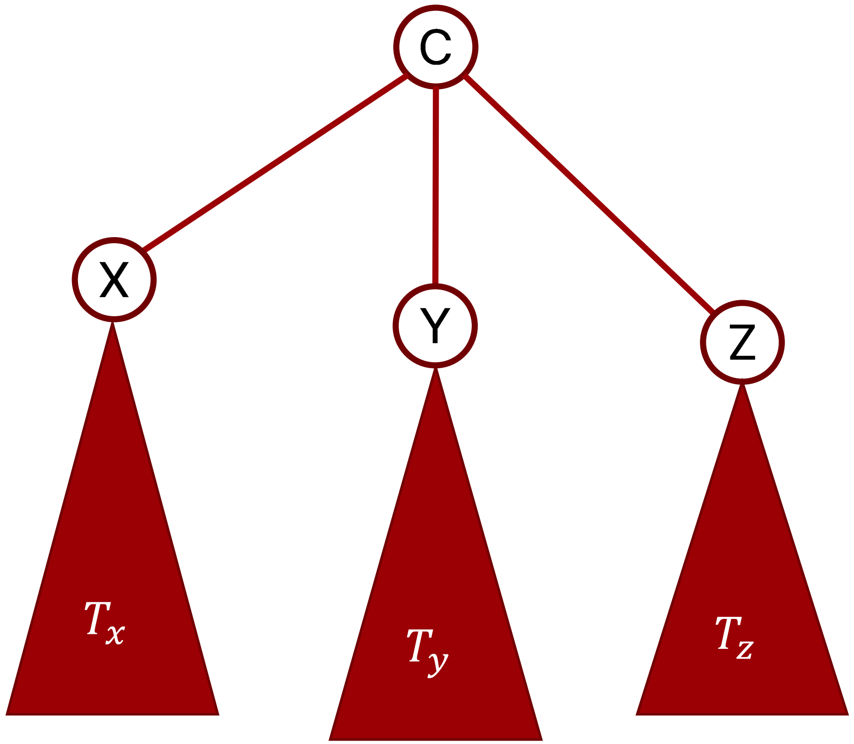
- Revisited : Merge Sort



$$T(N) = 2T\left(\frac{N}{2}\right) + O(N)$$

- Divide & Conquer on trees :

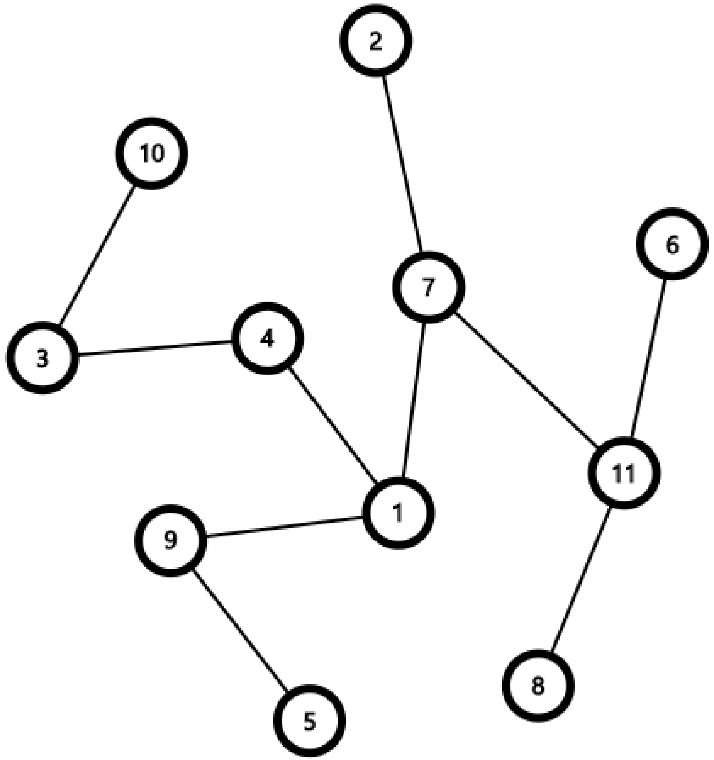
$$t(N) = \sum_{\forall\, u \in G.adj[c]} t(|T_u|) + f(N)$$



1. DFS를 수행하여 각 노드마다의 subtree의 크기 구하기
2. 다시 한번 DFS를 수행하여 현재 노드의 subtree의 크기에 따라 순회

1. DFS를 수행하여 각 노드마다의 subtree의 크기 구하기

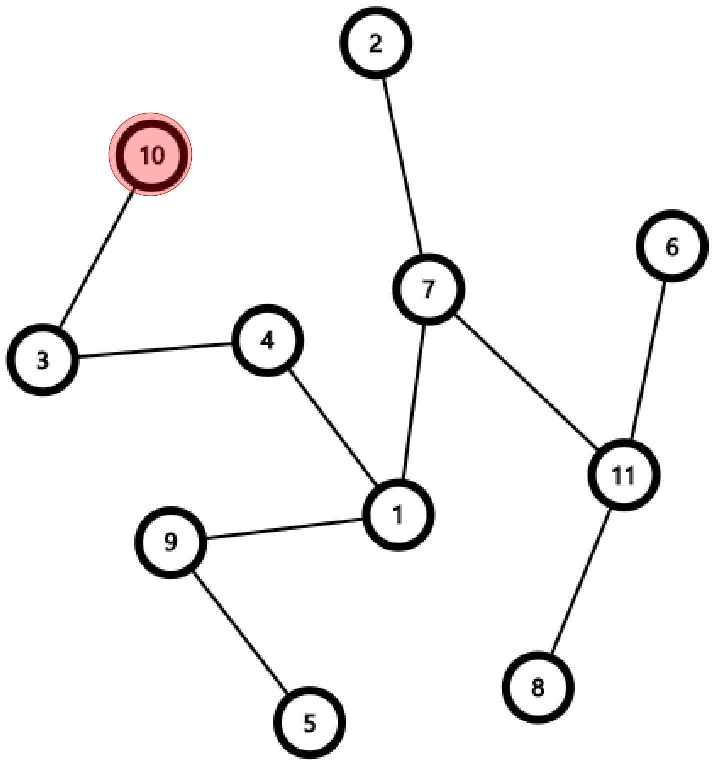
```
bitset<mxn> vis;
int sz[mxn];
int get_sz(int cur, int prv = -1) {
    sz[cur] = 1;
    for (int&nxt : adj[cur]) {
        if (nxt == prv || vis[nxt]) continue;
        sz[cur] += get_sz(nxt, cur);
    }
    mxlen = max(mxlen, sz[cur]);
    return sz[cur];
}
```



index	10	3	4	1	9	5	7	2	11	6	8
size	11	10	9	8	2	1	5	1	3	1	1

2. 다시 한번 DFS를 수행하여 현재 노드의 subtree의 크기에 따라 순회

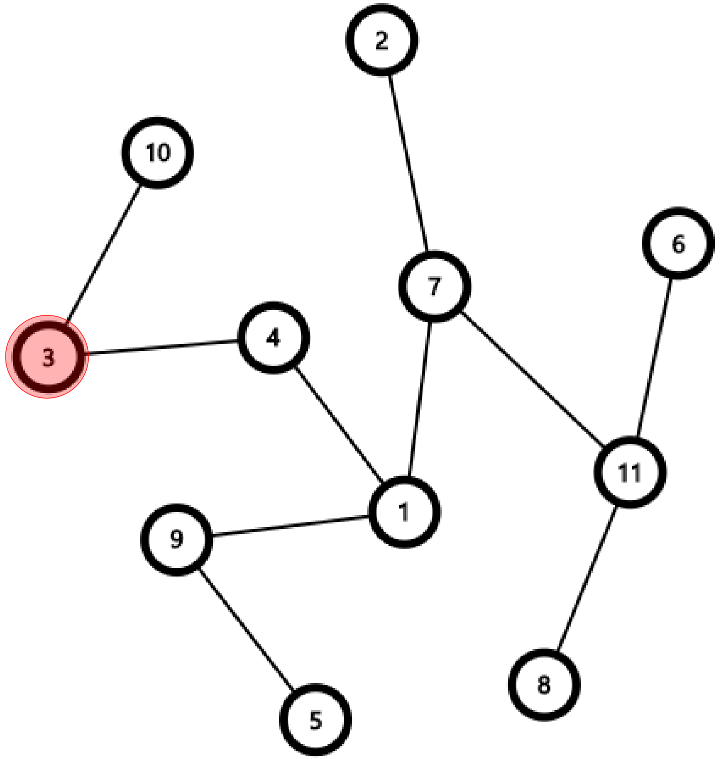
```
int get_centroid(int cur, int tree_sz, int prv = -1) {
    for (int&nxt : adj[cur]) {
        if (nxt == prv || vis[nxt]) continue;
        if (tree_sz < sz[nxt])
            return get_centroid(nxt, tree_sz, cur);
    }
    return cur;
}
```



index	10	3	4	1	9	5	7	2	11	6	8
size	11	10	9	8	2	1	5	1	3	1	1

2. 다시 한번 DFS를 수행하여 현재 노드의 subtree의 크기에 따라 순회

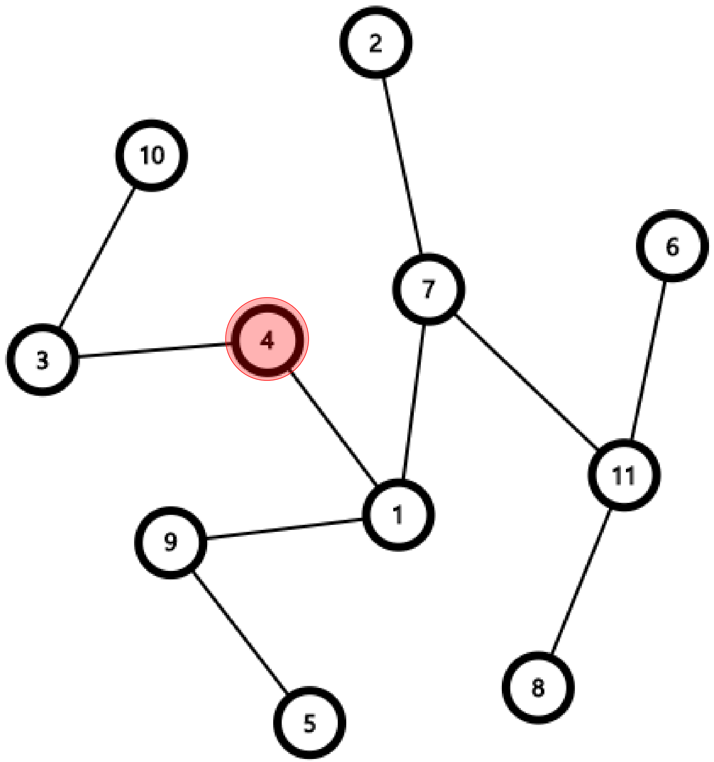
```
int get_centroid(int cur, int tree_sz, int prv = -1) {
    for (int&nxt : adj[cur]) {
        if (nxt == prv || vis[nxt]) continue;
        if (tree_sz < sz[nxt])
            return get_centroid(nxt, tree_sz, cur);
    }
    return cur;
}
```



index	10	3	4	1	9	5	7	2	11	6	8
size	11	10	9	8	2	1	5	1	3	1	1

2. 다시 한번 DFS를 수행하여 현재 노드의 subtree의 크기에 따라 순회

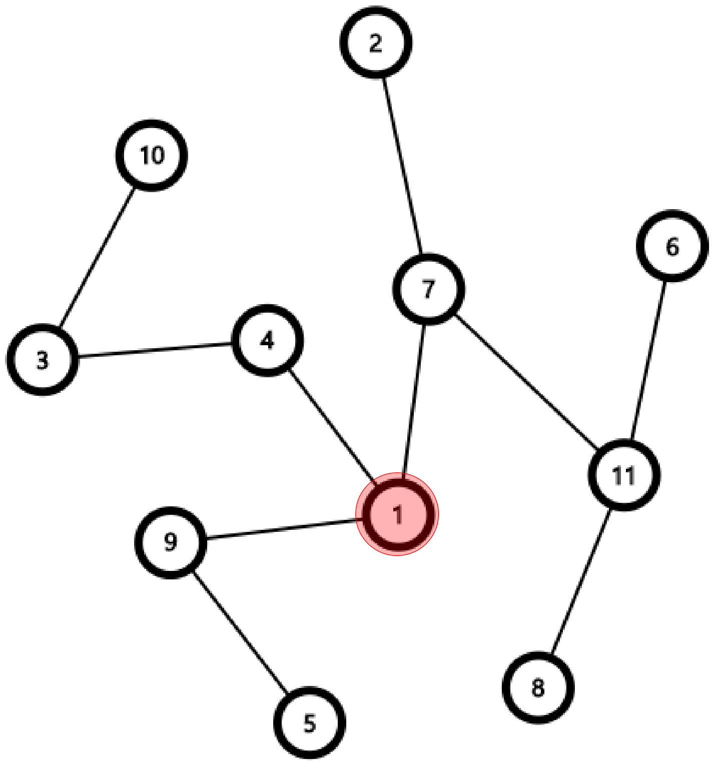
```cpp
int get_centroid(int cur, int tree_sz, int prv = -1) {
    for (int&nxt : adj[cur]) {
        if (nxt == prv || vis[nxt]) continue;
        if (tree_sz < sz[nxt])
            return get_centroid(nxt, tree_sz, cur);
    }
    return cur;
}
```



index	10	3	4	1	9	5	7	2	11	6	8
size	11	10	9	8	2	1	5	1	3	1	1

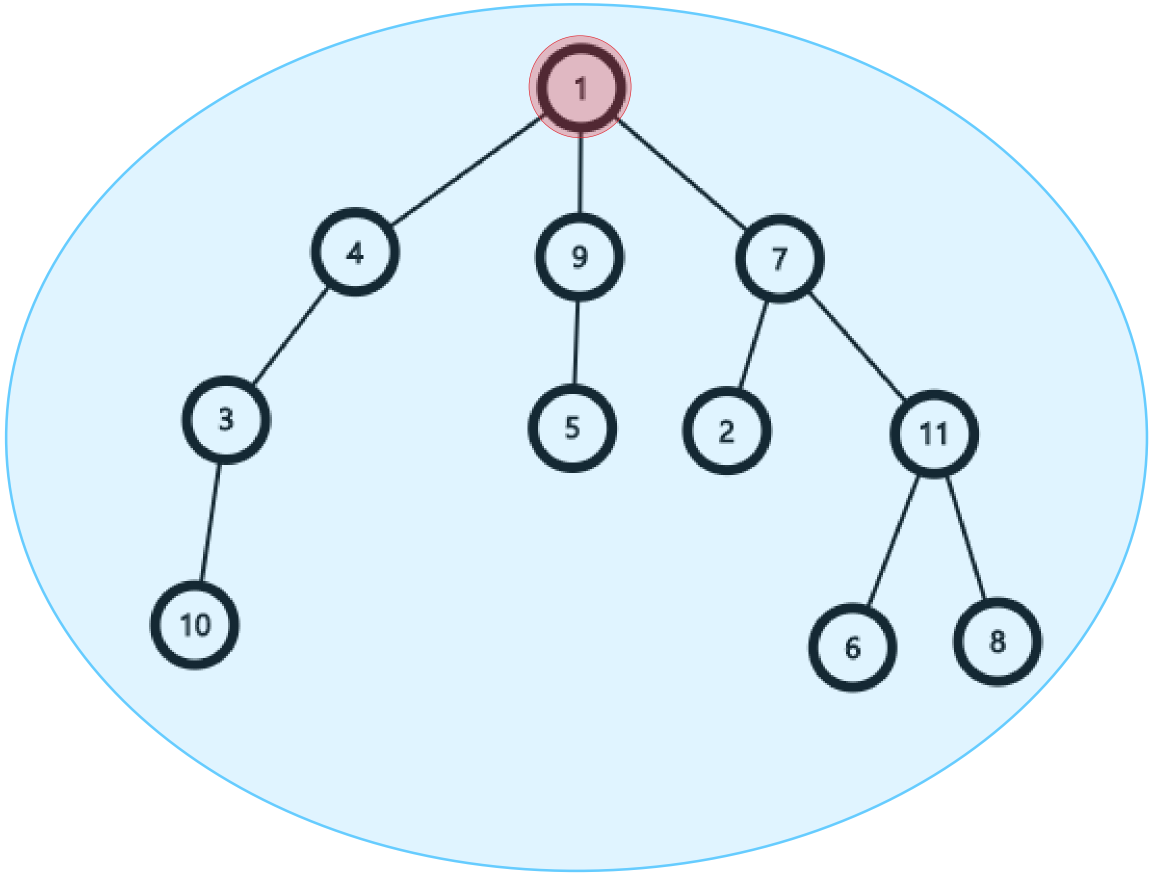
2. 다시 한번 DFS를 수행하여 현재 노드의 subtree의 크기에 따라 순회

```cpp
int get_centroid(int cur, int tree_sz, int prv = -1) {
    for (int&nxt : adj[cur]) {
        if (nxt == prv || vis[nxt]) continue;
        if (tree_sz < sz[nxt])
            return get_centroid(nxt, tree_sz, cur);
    }
    return cur;
}
```

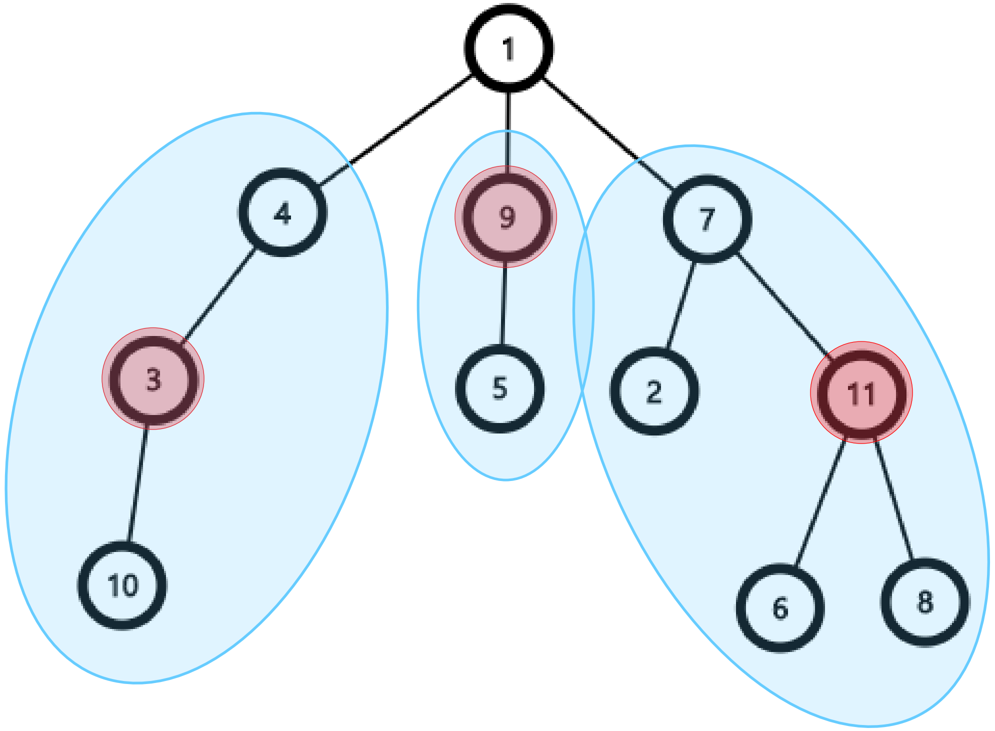


index	10	3	4	1	9	5	7	2	11	6	8
size	11	10	9	8	2	1	5	1	3	1	1

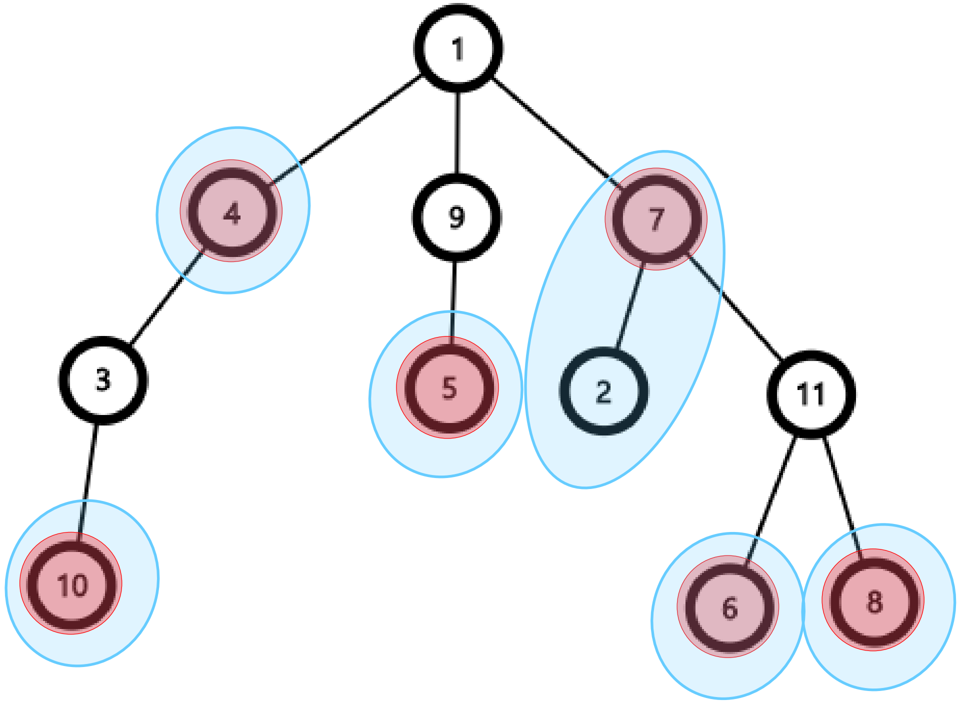
Finding Centroid of tree



Finding Centroid of tree



Finding Centroid of tree



1. Centroid 구하기
   - $O(sizeof\ subtree)$

2. Centroid를 지나는 경로에 대해 처리
   - $if\ O(N)\ then\ T(N) = O(N \log N)$

3. Centroid 제거, Centroid와 인접한 노드들에 대한 subtree(sub-problems)에 대해 재귀적으로 처리
   - Erase?

```
bitset<mxn> vis;
int sz[mxn];
int get_sz(int cur, int prv = -1) {
    sz[cur] = 1;
    for (int&nxt : adj[cur]) {
        if (nxt == prv || vis[nxt]) continue;
        sz[cur] += get_sz(nxt, cur);
    }
    mxlen = max(mxlen, sz[cur]);
    return sz[cur];
}
```

Pattern of subproblems

```cpp
78
79  int f(int cur) {
80      int ret = 0;
81      mxlen = 0;
82      int tree_sz = get_sz(cur);
83      cur = get_centroid(cur, tree_sz / 2);
84      vis[cur] = 1;
85
86      /*
87      do sth
88      */
89
90      for (int&nxt : adj[cur]) {
91          if (vis[nxt]) continue;
92          ret += f(nxt);
93      }
94
95      return ret;
96  }
97
```

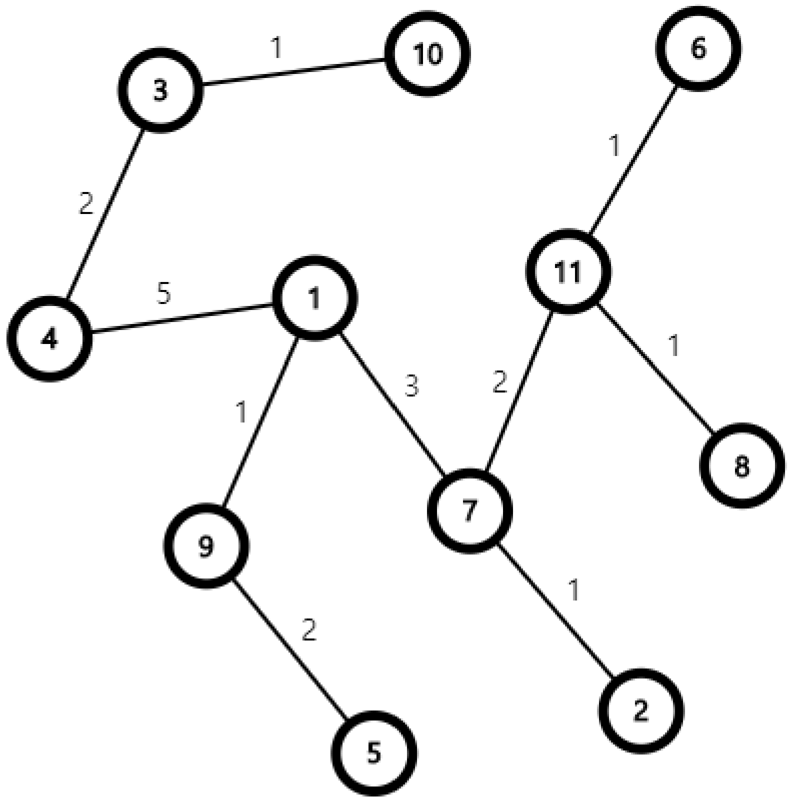
82~83. Centroid 구하기

84. Centroid 제거

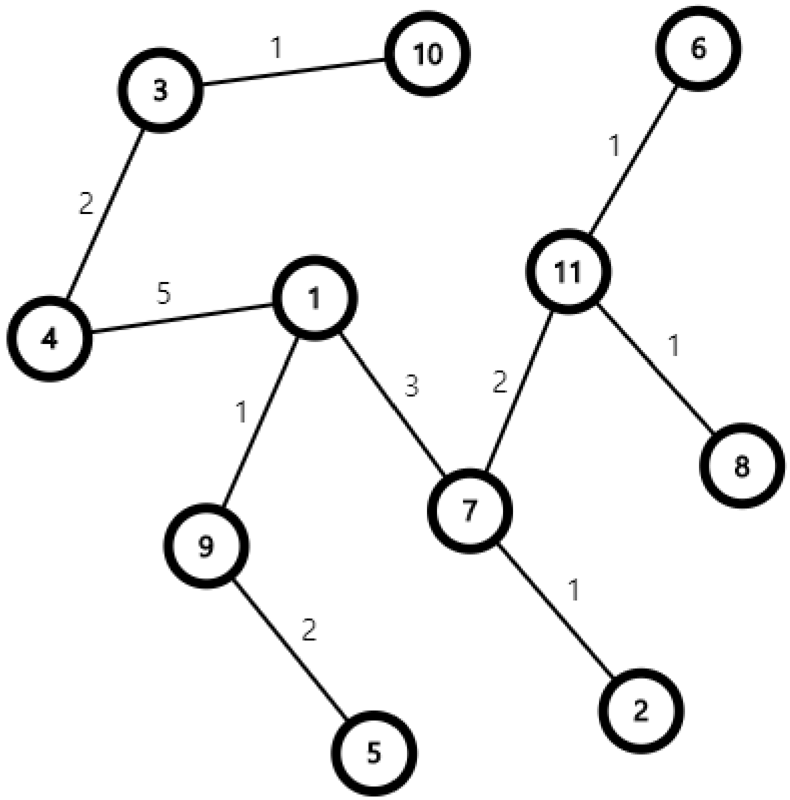
86~88. Centroid를 지나는 경로에 대해 처리

90~93 : Centroid와 인접한 노드들에 대한 subtree (sub-problems)에 대해 재귀적으로 처리

- N($1 \le N \le 200{,}000$)개의 정점으로 구성된 트리
- 길이가 K($1 \le K \le 1{,}000{,}000$)인 경로 중 간선 개수가 가장 작은 경로?



ex) K=8



- Naïve approach

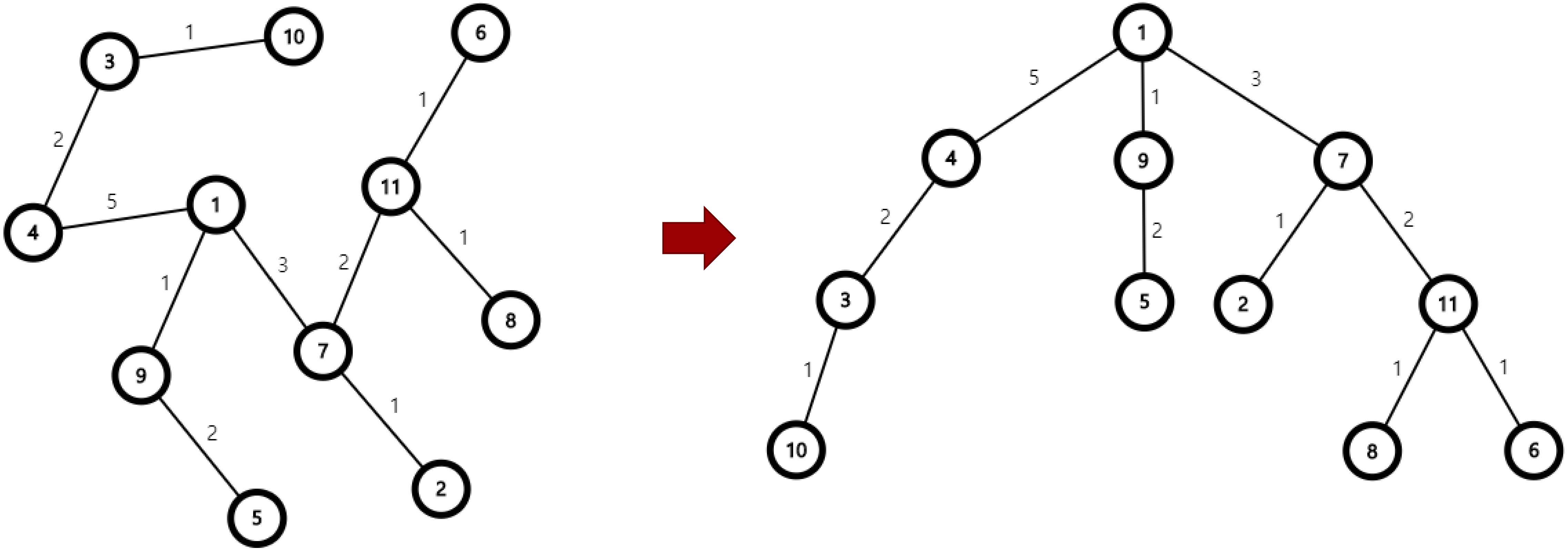
임의의 두 정점 u, v에 대하여 => $O(N^2)$

get dist(u,v) => $O(\log N)$ or possibly $O(1)$

$$\geq O(N^2) = TLE$$

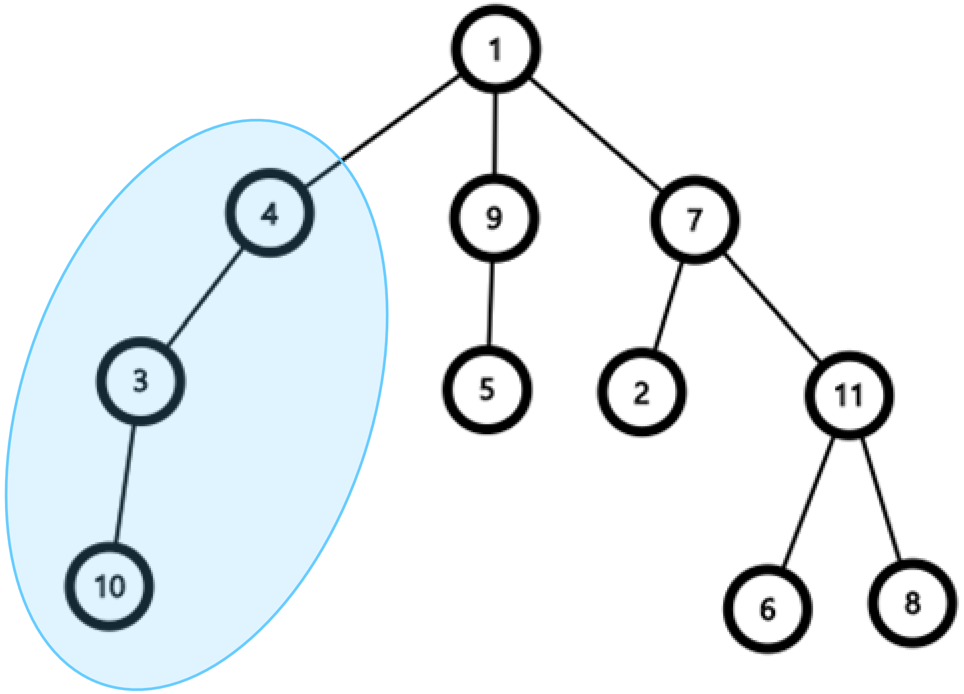
ex) K=8

1. Find Centroid

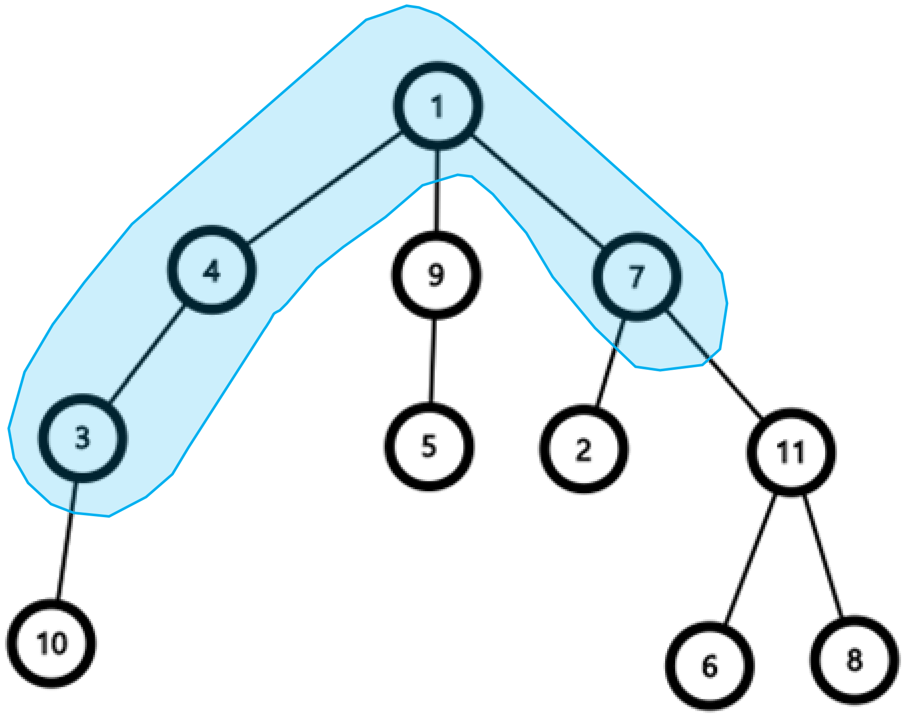


ex) K=8

2. Consider two cases :



(1) subtree 중 하나에 속하는 경로



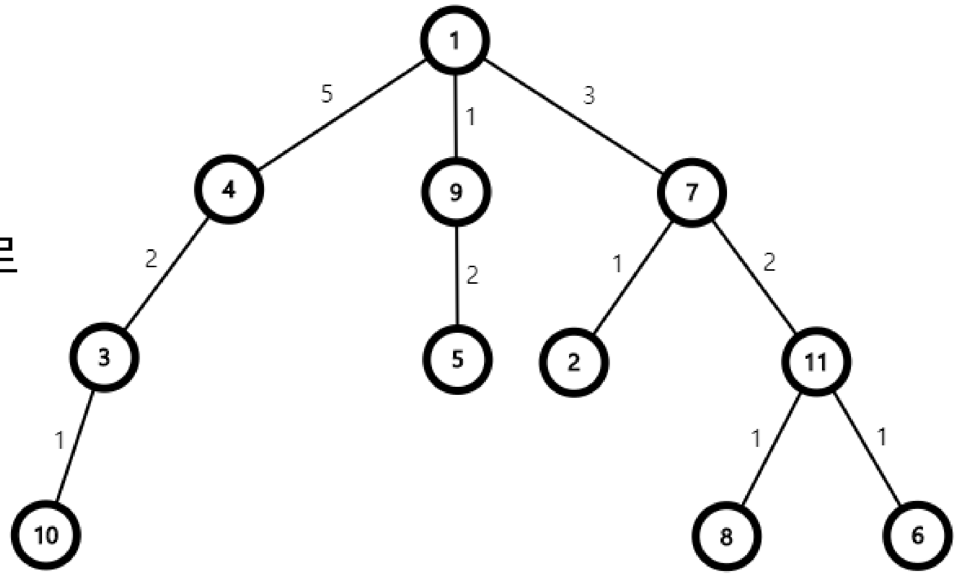
(2) centroid를 지나는 경로

ex) K=8

2. Consider two cases :

(2) centroid를 지나는 경로

- Target : 길이가 K인 경로 중 간선 개수가 가장 적은 것
- 경로 구성
    - ✓ subtree T_i의 정점 중 centroid로부터 길이가 x인 경로
    - ✓ subtree $T_j (i \neq j)$의 정점 중 centroid로부터 길이가 K-x인 경로

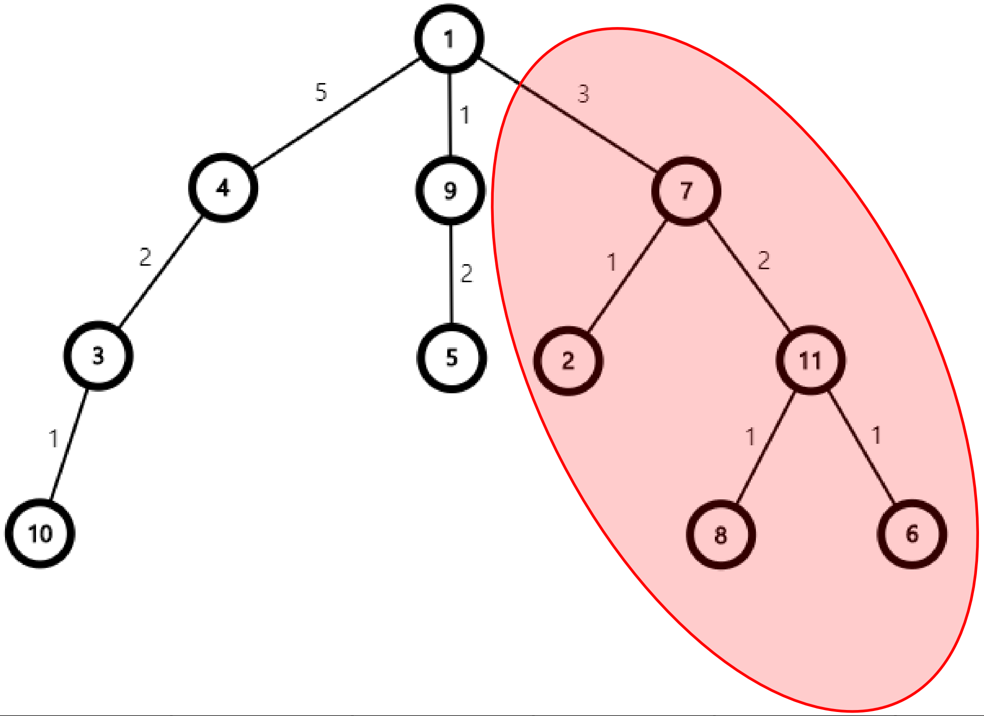


- Requirements

  ✓ subtree T_i의 정점 중 centroid로부터 길이가 x인 경로
  ✓ subtree $T_j (i \neq j)$의 정점 중 centroid로부터 길이가 K-x인 경로

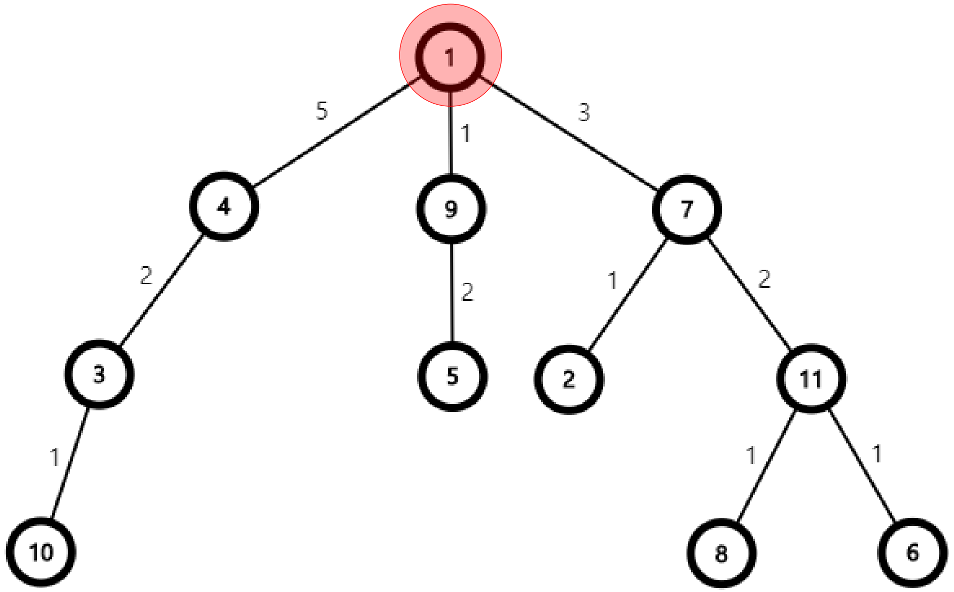
  들 중 centroid로부터 깊이가 가장 낮은 것들의 집합만 필요

ex) K=8



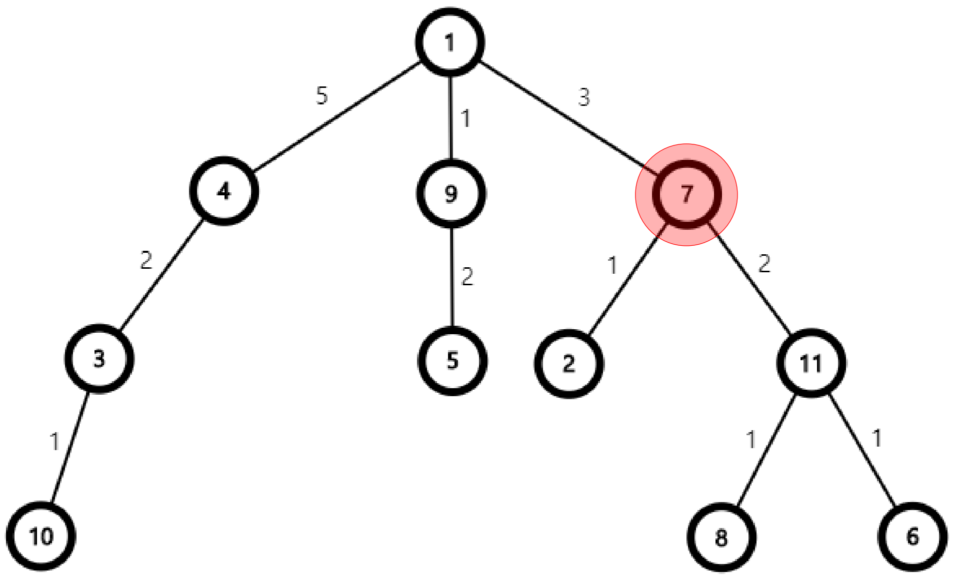
distance	0	1	2	3	4	5	6	7	8
min_dep	INF	INF	INF	INF	INF	INF	INF	INF	INF

ex) K=8



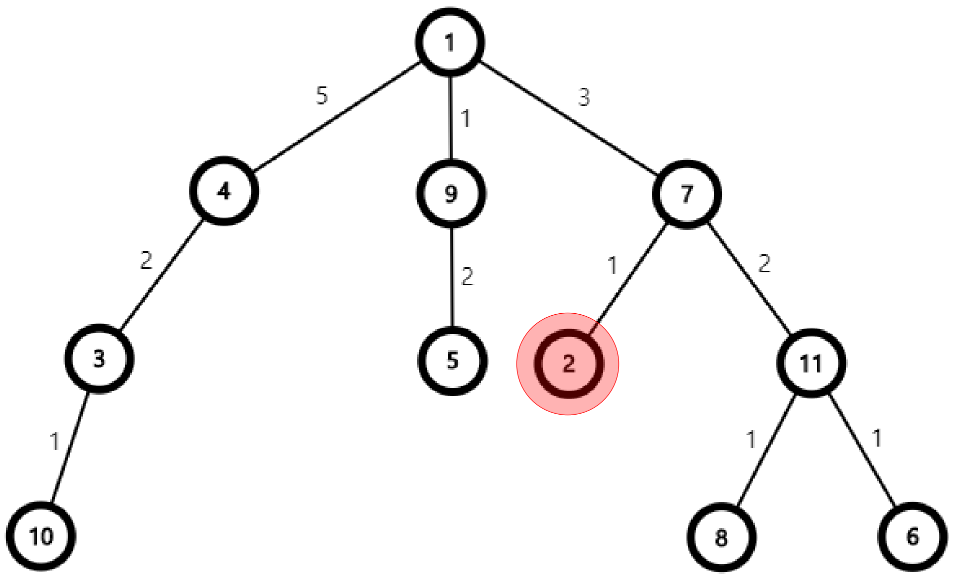
distance	0	1	2	3	4	5	6	7	8
min_dep	0	INF	INF	INF	INF	INF	INF	INF	INF

ex) K=8



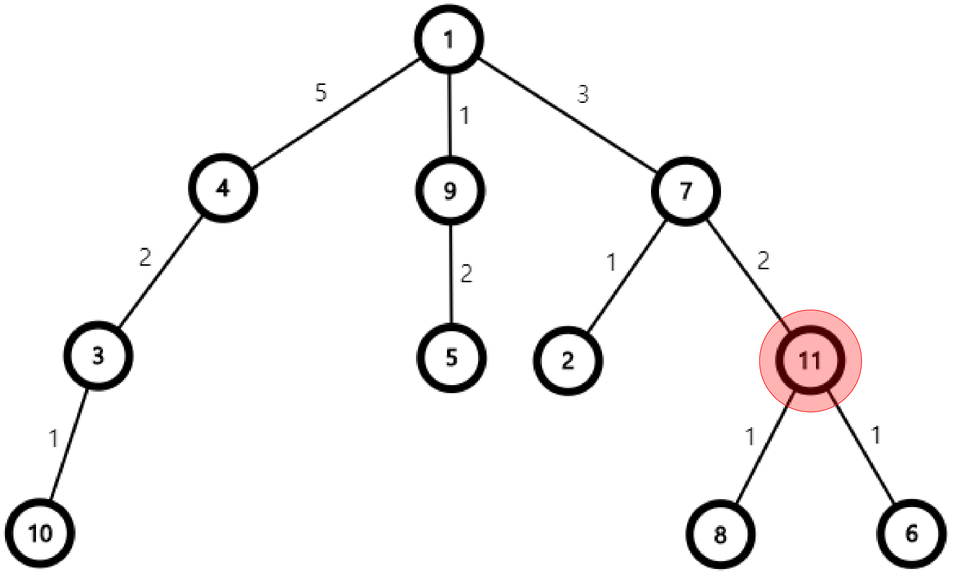
distance	0	1	2	3	4	5	6	7	8
min_dep	0	INF	INF	1	INF	INF	INF	INF	INF

ex) K=8



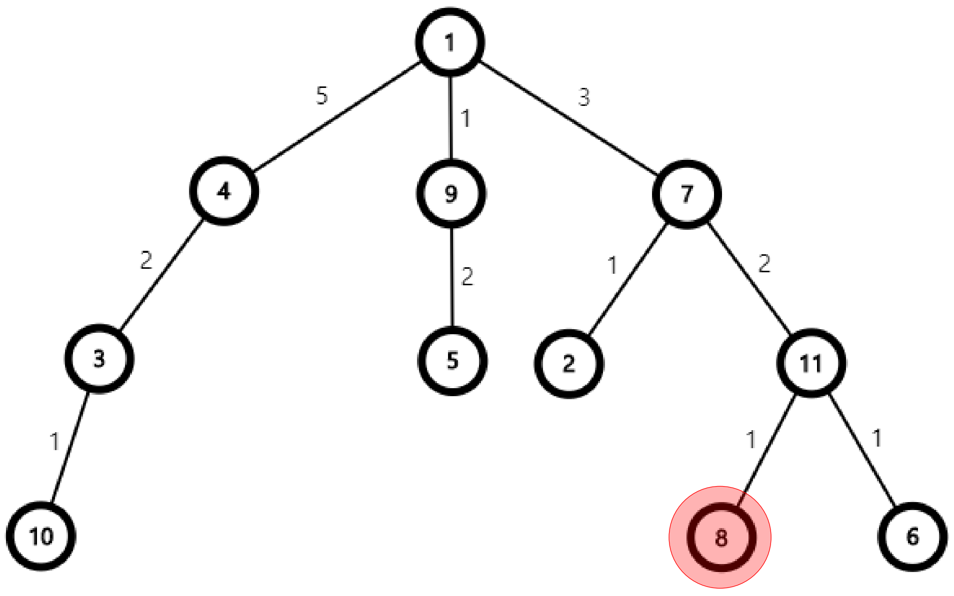
distance	0	1	2	3	4	5	6	7	8
min_dep	0	INF	INF	1	2	INF	INF	INF	INF

ex) K=8



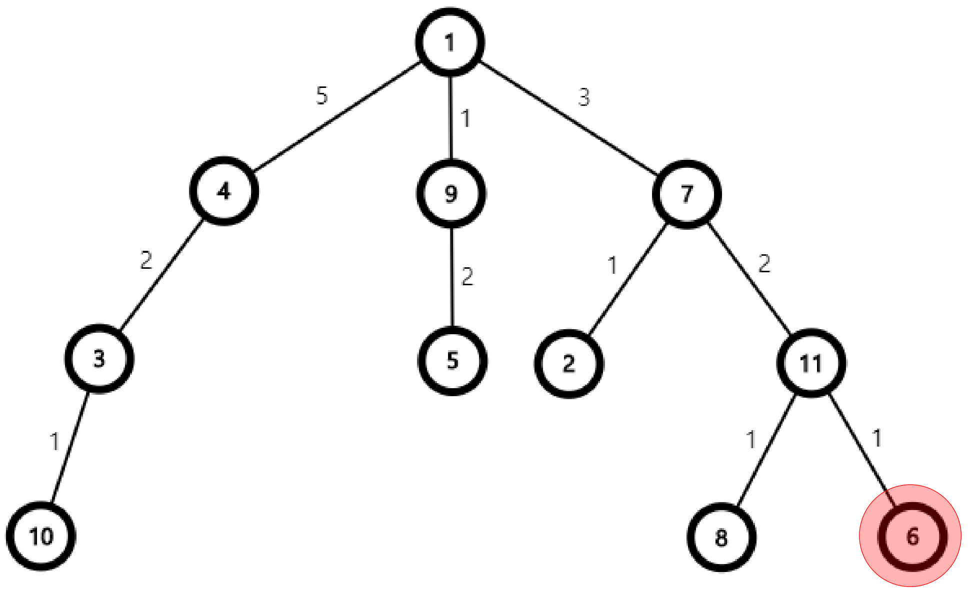
distance	0	1	2	3	4	5	6	7	8
min_dep	0	INF	INF	1	2	2	INF	INF	INF

ex) K=8



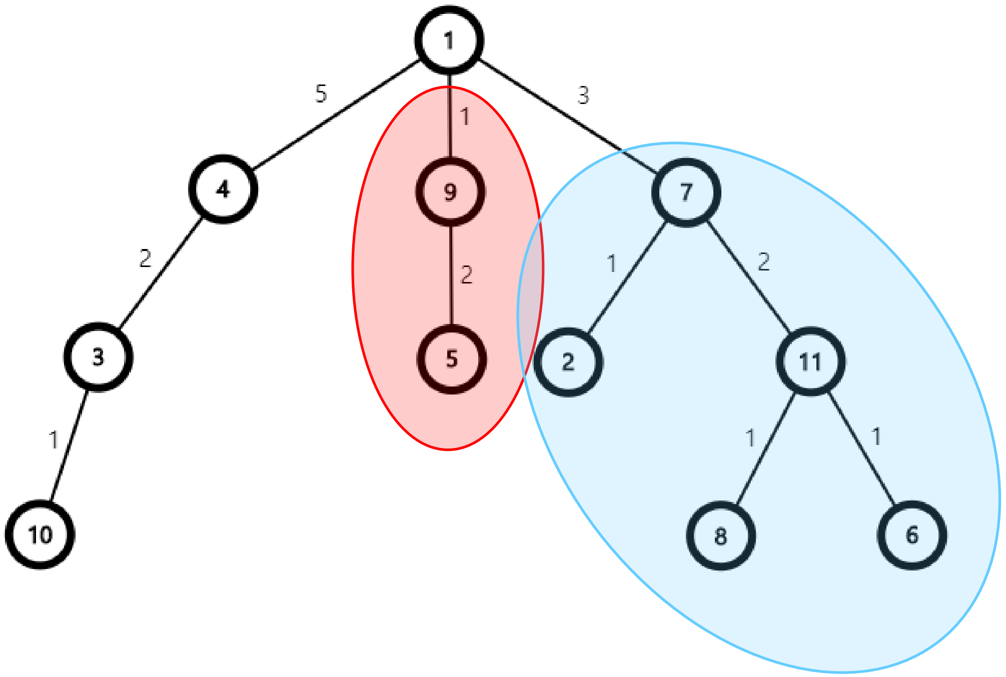
distance	0	1	2	3	4	5	6	7	8
min_dep	0	INF	INF	1	2	2	3	INF	INF

ex) K=8



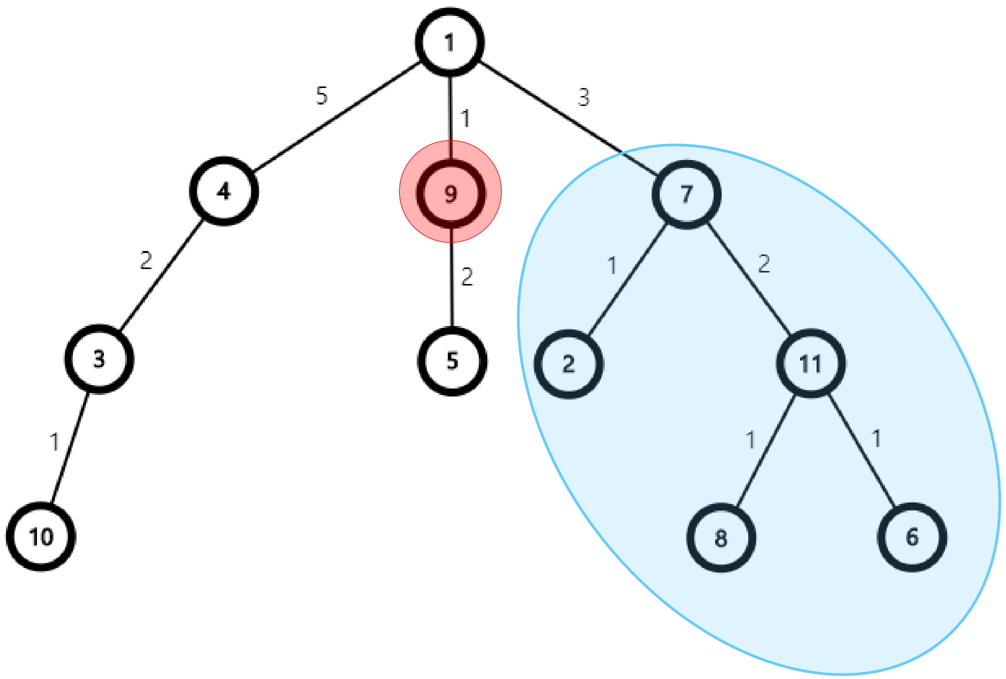
distance	0	1	2	3	4	5	6	7	8
min_dep	0	INF	INF	1	2	2	3	INF	INF

ex) K=8



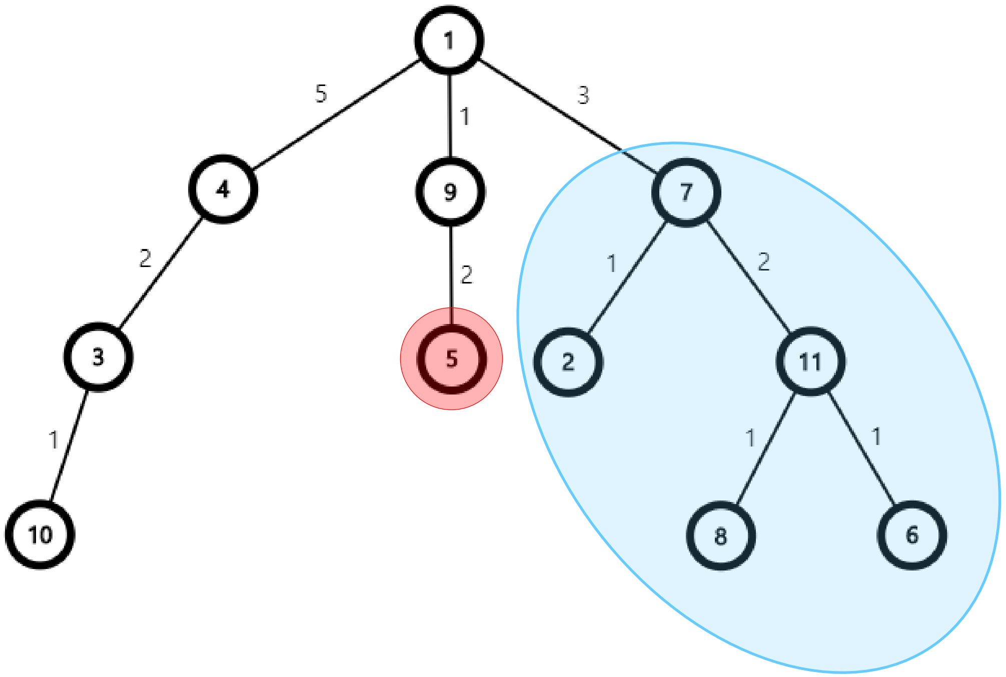
distance	0	1	2	3	4	5	6	7	8
min_dep	0	INF	INF	1	2	2	3	INF	INF

ex) K=8



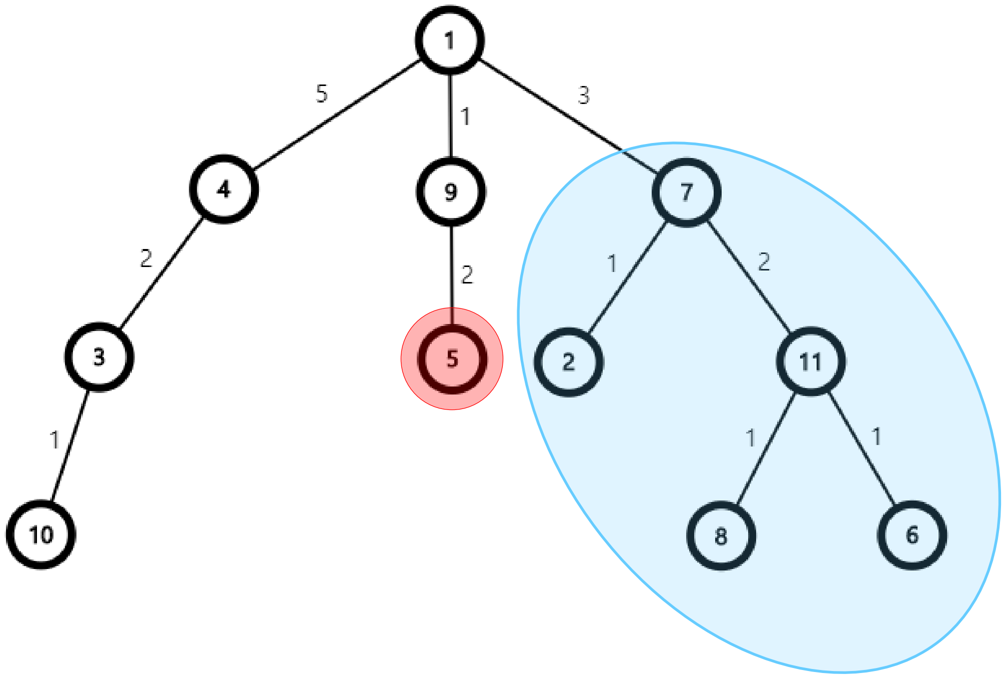
distance	0	1	2	3	4	5	6	7	8
min_dep	0	1	INF	1	2	2	3	INF	INF

ex) K=8



distance	0	1	2	3	4	5	6	7	8
min_dep	0	1	INF	1	2	2	3	INF	INF

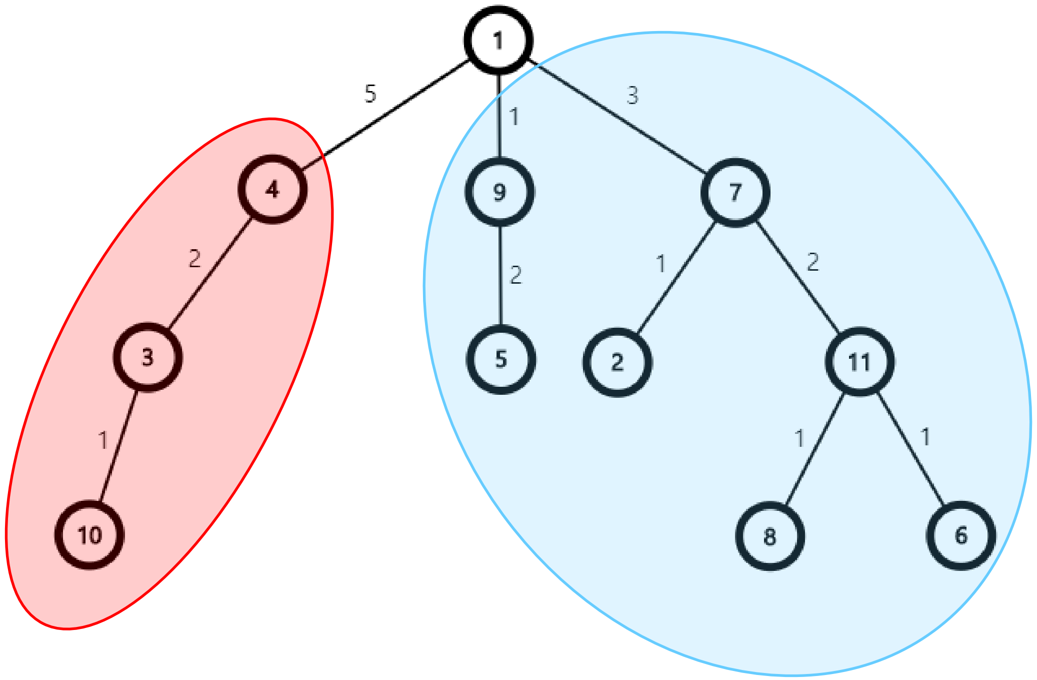
ex) K=8



cand = 2+2=4

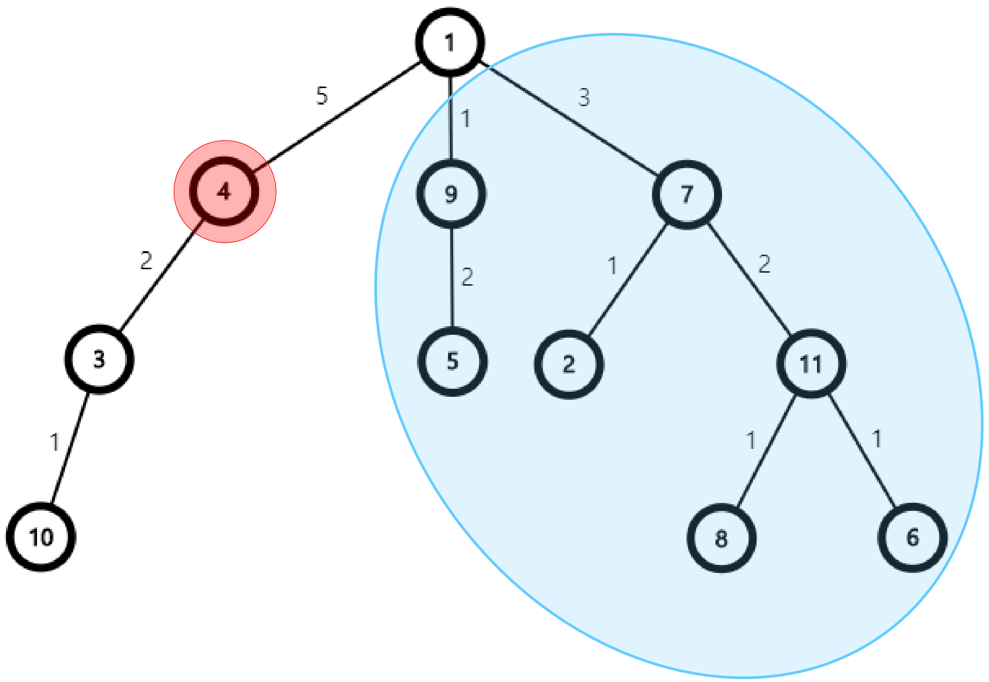
distance	0	1	2	3	4	5	6	7	8
min_dep	0	1	INF	1	2	2	3	INF	INF

ex) K=8



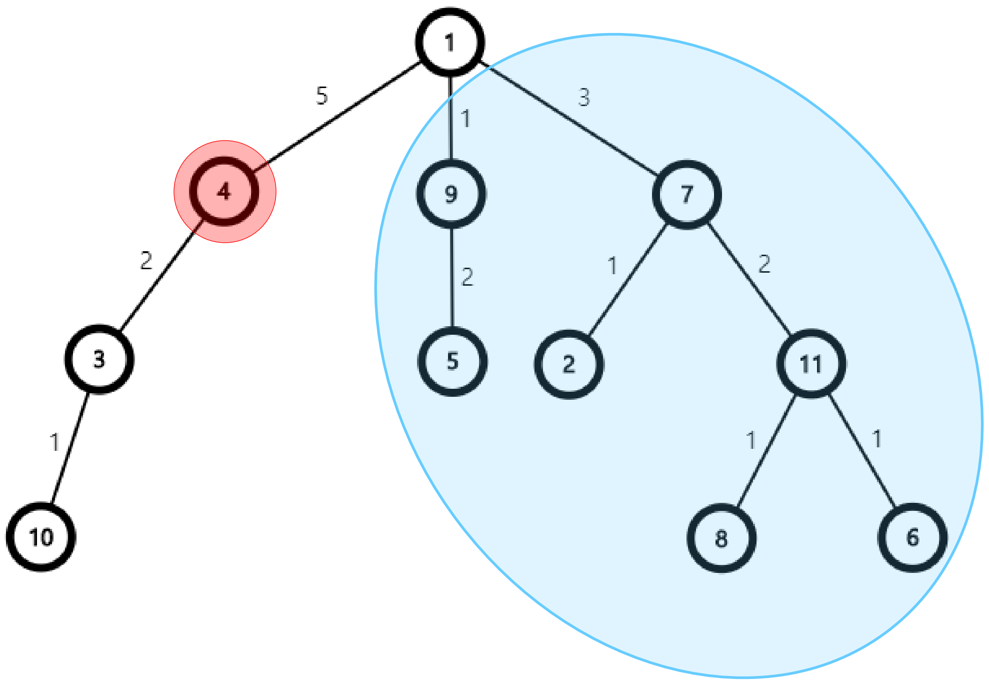
distance	0	1	2	3	4	5	6	7	8
min_dep	0	1	INF	1	2	2	3	INF	INF

ex) K=8



distance	0	1	2	3	4	5	6	7	8
min_dep	0	1	INF	1	2	1	3	INF	INF

ex) K=8



cand = min(prv, 1+1) = 2

distance	0	1	2	3	4	5	6	7	8
min_dep	0	1	INF	1	2	1	3	INF	INF

#5820 경주

```cpp
int f(int cur) {
    if (vis[cur]) return INF;
    int ret = INF;

    int tree_sz = get_sz(cur);
    int centroid = get_centroid(tree_sz / 2, cur);
    if (vis[centroid]) return INF;
    vis[centroid] = 1;

    while (!updated.empty()) mn_dep[updated.front()] = INF, updated.pop();
    mn_dep[0] = 0;

    for (auto&it : adj[centroid]) {
        int nxt = it.first, w = it.second;
        if (vis[nxt]) continue;
        ret = min(ret, solve(nxt, centroid, w));
        update(nxt, centroid, w);
    }

    for (auto&it : adj[centroid]) {
        int nxt = it.first;
        if (vis[nxt]) continue;
        ret = min(ret, f(nxt));
    }

    return ret;
}
```

#5820 경주

```cpp

int solve(int cur, int prv, int dist, int dep = 1) {
    if (dist > K || vis[cur]) return INF;
    int ret = INF;

    ret = min(ret, mn_dep[K - dist] + dep);

    for (auto&it : adj[cur]) {
        int nxt = it.first, w = it.second;
        if (nxt == prv || vis[nxt]) continue;
        ret = min(ret, solve(nxt, cur, dist + w, dep + 1));
    }
    return ret;
}

void update(int cur, int prv, int dist, int dep = 1) {
    if (dist > K || vis[cur]) return;

    updated.push(dist);
    mn_dep[dist] = min(dep, mn_dep[dist]);

    for (auto&it : adj[cur]) {
        int nxt = it.first, w = it.second;
        if (nxt == prv || vis[nxt]) continue;
        update(nxt, cur, dist + w, dep + 1);
    }
}

```

- 트리상에서 $O(N^2)$ solution을 떠올릴 수 있지만 이를 $O(NlogN)$으로 줄이는 과정이 필요할 때 사용
- 최대한 절반에 가깝게 쪼개서 분할하기 위한 기준점 필요 => Centroid
- 임의의 트리는 centroid를 1개 또는 2개 갖는다.
- 모든 노드는 특정 서브트리의 centroid가 될 수 있다.
- https://solved.ac/problems/algorithms/18

Sogang

ICPC Tea