2023 SCCC 봄 #2

BOJ 27648. 증가 배열 만들기

K 이하의 수를 사용해서 증가 배열을 만드는 문제 대신, N과 M이 주어졌을 때 증가 배열을 만들기 위해 필요한 수의 최소 개수 $f(N, M)$을 구하는 문제를 해결해 봅시다. 만약 이러한 $f(N, M)$을 구할 수 있다면 $f(N, M) \le K$일 때는 그 방법대로 수를 채우면 되고, 반대로 $f(N, M) > K$일 때는 `NO`를 출력하면 됩니다.

문제에서 제시한 조건을 다시 한번 정리해 봅시다.

(i, j)에 어떤 수 v를 배치하기 위해서는 $(1, 1)$에서 (i, j)로 이동할 때 거칠 수 있는 모든 칸에 배정된 값이 v 미만이어야 합니다. 즉, (i, j)보다 왼쪽 위에 있는 모든 칸에 v 미만의 수를 배정할 수 있어야 합니다.

아무래도 2차원보다는 1차원에서 문제를 푸는 것이 쉬울 것 같으니 일단 $N = 1$인 1차원 배열의 경우만 생각해 봅시다.

1차원일 때는 위/아래 행이 존재하지 않기 때문에 자신의 왼쪽에 있는 칸들의 값만 신경써도 됩니다. 항상 자신의 왼쪽에 있는 모든 수보다 큰 수를 배치하기 때문에 1차원 배열에 배정된 값을 차례대로 읽으면 증가하는 수열이 됩니다. 따라서 $(1, j)$에 수를 배정할 때는 $(1, j-1)$에 있는 수보다 큰 수인지만 확인해도 충분합니다.

사용해야 하는 수의 최댓값을 최소화해야 하므로 $(1, 1)$에는 1을 배치하는 것이 가장 좋을 것입니다. 바로 오른쪽 칸인 $(1, 2)$에는 딱 1 만큼만 증가시킨 2, 그다음 칸에는 3, 이렇게 $(1, j)$에는 j를 배치하는 것이 최적이라는 것을 쉽게 알 수 있습니다.

이제 이 풀이를 조금 변형해서 2차원에서 문제를 해결해 봅시다. 위/아래 행이 추가되었으므로 어떤 칸에 수를 배치할 때 자신의 위에 있는 수들도 함께 고려해야 합니다.

(i, j)에 배치될 수는 $(1, 1 \cdots j), (2, 1 \cdots j), \cdots, (i-1, 1 \cdots j), (i, 1 \cdots j-1)$보다 큰 수여야 합니다. 이때 $(1, 1 \cdots j)$에 있는 수는 $(1, j)$에 배정된 수보다 작거나 같고, $(2, 1 \cdots j)$에 있는 수는 $(2, j)$에 배정된 수보다 작거나 같습니다. 이런 식으로 각 행에서 고려해야 하는 칸에 배정된 수를 고려하는 대신, 그 칸이 속한 행의 j번째 칸만 고려해도 충분하다는 것을 알 수 있습니다.

따라서 (i, j)에 수를 배치할 때는 $(i, j-1)$과 $(i-1, j)$에 배정된 수만 고려해도 됩니다. 규칙 $A(i, j) = \max\{A(i, j-1), A(i-1, j)\} + 1$에 따라 수를 배치해 보면 $A(i, j) = i + j - 1$이 된다는 것을 알 수 있습니다.

따라서 $f(N, M) = N + M - 1$이고, $N + M - 1$가지의 수만 사용해 $N \times M$ 크기의 2차원 배열을 채우는 방법까지 구할 수 있습니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, M, K; cin >> N >> M >> K;
    if(K < N+M-1){ cout << "NO"; return 0; }
    cout << "YES\n";
    for(int i=1; i<=N; i++){
        for(int j=1; j<=M; j++) cout << i+j-1 << " \n"[j==M];
    }
}
```

BOJ 27498 연애 혁명

각 학생을 정점, 사랑 관계를 간선으로 만든 무향 그래프를 생각해 봅시다. 사랑 관계에 $K(K \geq 3)$각 관계가 존재한다는 것은 그래프에 크기가 K인 사이클이 존재한다는 것과 동치입니다. 따라서 이 문제는 그래프가 주어졌을 때 간선을 제거해서 포레스트(사이클이 없는 무향 그래프)로 만드는 문제라고 생각할 수 있습니다.

제거하는 간선의 가중치 합을 최소화하는 것은 제거하지 않는 간선의 가중치 합을 최대화하는 것과 동일합니다. 따라서 이 문제는 최대 가중치 스패닝 포레스트 문제의 변형이라고 생각할 수 있습니다.

크루스칼 알고리즘을 이용해 최대 가중치 스패닝 포레스트를 구할 것입니다. 이미 성사된 관계($d_i = 1$)는 끊을 수 없기 때문에 미리 두 정점을 병합하고, 그 이후에 크루스칼 알고리즘을 수행하면 제거하지 않는 간선의 가중치 합을 최대화할 수 있습니다. 제거된 간선, 즉 최대 가중치 스패닝 포레스트에 포함되지 않은 간선의 가중치를 더한 것이 정답이 됩니다.

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

int N, M, P[10101], R;
vector<tuple<int,int,int>> E;
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Merge(int u, int v){ return Find(u) != Find(v) && (P[P[u]]=P[v], true); }

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M; iota(P+1, P+N+1, 1);
    for(int i=1; i<=M; i++){
        int a, b, c, d; cin >> a >> b >> c >> d;
        if(d == 1) Merge(a, b);
        else E.emplace_back(c, a, b), R += c;
    }
    sort(E.begin(), E.end(), greater<>());
    for(const auto &[w,u,v] : E) if(Merge(u, v)) R -= w;
    cout << R;
}
```

BOJ 9646 다이어그램과 태블로

이 정도 레벨에서 경우의 수 문제는 대부분 다이나믹 프로그래밍으로 해결할 수 있습니다.

가장 먼저 떠오르는 방법은 $D(i,j,v) := i-1$번째 행까지 전부 채우고, i번째 행은 j번째 열까지 채웠을 때 $A(i,j) = v$가 되는 경우의 수와 같은 방식으로 점화식을 정의하는 것입니다. 하지만 각 행의 값들을 결정할 때 한 줄 위에 있는 행의 값을 전부 고려해야 하므로 상태 전이를 설계하는 것이 쉽지 않습니다.

행 또는 열 단위로 경우의 수를 구할 것입니다. 구체적으로, i번째 행이 $\{a_1, a_2, \cdots, a_n\}$이 되는 경우의 수, 또는 j번째 열이 $\{b_1, b_2, \cdots, b_m\}$이 되는 경우의 수를 구합니다.

행 단위로 경우의 수를 구하기 위해서는 점화식을 $D(i, a) :=$ i번째 행까지 채웠고, i번째 행에 배정된 수들이 배열 a와 같은 경우의 수와 같이 정의해야 합니다. 이때 a로 가능한 가짓수는 ${}_n\mathrm{H}_{|a|} = {}_{n+|a|-1}\mathrm{C}_{n-1}$ 입니다.

열 단위로 경우의 수를 구할 때도 비슷하게 $D(j, b)$를 정의해야 하고, 이때 b로 가능한 배열의 가짓수는 ${}_n\mathrm{C}_{|b|}$ 입니다.

열 단위로 경우의 수를 계산할 때 가능한 b의 가짓수는 최대 35가지($= {}_7\mathrm{C}_3$)로 행 단위로 구할 때보다 작기 때문에 열 단위로 구할 것입니다.

점화식의 상태 전이는 두 배열 a, b이 모든 $0 \le i < \min\{|a|, |b|\}$에서 $a_i \le b_j$를 만족할 때 $D(j, b) \leftarrow D(j-1, a)$와 같은 방식으로 전이하면 됩니다.

점화식의 인자로 배열을 넘기는 것은 귀찮기 때문에 0 이상 ${}_n\mathrm{C}_{|b|}$ 미만의 정수로 인코딩해서 넣으면 더 편하게 구현할 수 있습니다.

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

vector<int> T;
vector<vector<int>> V[8];

void Gen(int n){
    if(!T.empty()) V[T.size()].push_back(T);
    if(T.size() == 7) return;
    for(int i=(T.empty()?0:T.back())+1; i<=n; i++){
        T.push_back(i);
        Gen(n);
        T.pop_back();
    }
}

int N, K, S, A[11], D[111][111];

void Solve(){
    memset(A, 0, sizeof A);
    for(int i=1; i<=K; i++){
        int t; cin >> t; if(i == 1) S = t;
        for(int j=1; j<=t; j++) A[j]++;
    }
    cin >> N;
    for(int i=0; i<8; i++) V[i].clear();
    Gen(N);

    memset(D, 0, sizeof D);
    for(int i=0; i<V[A[1]].size(); i++) D[1][i] = 1;
    for(int i=2; i<=S; i++){
        for(int j=0; j<V[A[i]].size(); j++){
            for(int k=0; k<V[A[i-1]].size(); k++){
                bool flag = true;
                for(int s=0; s<A[i]; s++) flag &= V[A[i-1]][k][s] <= V[A[i]][j]
[s];
                if(flag) D[i][j] += D[i-1][k];
            }
```

```
        }
    }
    cout << accumulate(D[S], D[S]+111, 0LL) << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    while(cin >> K && K) Solve();
}
```

BOJ 27652 AB

문자열들의 집합 A, B가 주어졌을 때 `find(s)` 쿼리를 처리하는 방법을 알아봅시다.

S를 비어있지 않은 두 문자열 a, b의 연결로 표현($S = a + b$)하는 방법은 총 $|S| - 1$가지입니다. A의 원소 중 a를 접두사로 갖는 문자열의 개수를 a', B의 원소 중 b를 접미사로 갖는 문자열의 개수를 b'이라고 하면, 두 문자열 a, b는 정답에 $a' \times b'$ 만큼 기여합니다. 주어진 쿼리를 효율적으로 처리하기 위해서는 S를 분할하는 총 $|S| - 1$가지 방법에서 얻게 되는 a'와 b'들을 효율적으로 구할 수 있어야 합니다.

A의 원소들로 Trie T_A를 만들고, 각 정점에 자신의 후손 중 terminal node인 정점의 개수를 저장합시다. T_A에서 S를 찾는 과정에서 방문하는 정점에 저장된 값을 모두 모으면 a'를 전부 구할 수 있습니다. 마찬가지로 B의 원소를 모두 뒤집은 문자열들로 Trie T_B를 만듭시다. T_B에서 S^R을 찾는 과정에서 방문하는 정점에 저장된 값을 모두 모으면 b'를 전부 구할 수 있습니다.

a'과 b'을 모두 구하는데 $O(|S|)$ 만큼의 시간이 걸리고, 답을 구할 때도 마찬가지로 $O(|S|)$ 만큼의 시간이 걸립니다.

`add`, `delete` 쿼리는 기초적인 Trie 연산이고, 모두 $O(|S|)$ 시간에 처리할 수 있습니다. 따라서 전체 시간 복잡도는 $O(\sum |S|)$입니다.

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

struct Trie{
    int sz; Trie *ch[26];
    Trie(){ sz = 0; fill(ch, ch+26, nullptr); }
    void update(const char *s, int v){
        sz += v; if(!*s) return;
        if(!ch[*s-'a']) ch[*s-'a'] = new Trie;
        ch[*s-'a']->update(s+1, v);
    }
    void get(const char *s, vector<int> &res){
        res.push_back(sz); if(!*s) return;
        if(ch[*s-'a']) ch[*s-'a']->get(s+1, res);
    }
};

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int Q; cin >> Q;
    Trie *A = new Trie, *B = new Trie;
    for(int q=1; q<=Q; q++){
        string op; cin >> op;
        if(op == "find"){
            string s; cin >> s;
            vector<int> a, b;
```

```
            A->get(s.c_str(), a); reverse(s.begin(), s.end());
            B->get(s.c_str(), b); reverse(s.begin(), s.end());
            while(a.size() <= s.size()) a.push_back(0);
            while(b.size() <= s.size()) b.push_back(0);
            long long res = 0;
            for(int i=1; i<s.size(); i++) res += 1LL * a[i] * b[s.size()-i];
            cout << res << "\n";
        }
        else{
            char c; string s; cin >> c >> s;
            if(c == 'B') reverse(s.begin(), s.end());
            (c == 'A' ? A : B)->update(s.c_str(), op[0] == 'a' ? +1 : -1);
        }
    }
}
```

BOJ 26640 Palindrom

주어진 문자열이 팰린드롬인지 확인하는 간단한 문제인데... 문제는 메모리 제한이 너무 작아서 문자열 전체를 저장할 수 없습니다. 문자열의 길이가 입력으로 주어지지 않고 최대 2×10^7 인 것만 알고 있을 때, 메모리를 $4MB$ 이하만 사용해서 주어진 문자열이 팰린드롬인지 확인해야 합니다.

문자열을 저장할 수 없으면 해시값을 저장하면 됩니다. 두 가지 해시값을 저장할 건데, 하나는 해시값을 $f(s_0 s_1 \cdots s_{n-1}) = \sum s_i P^i$ 함수를 이용해 계산하고, 다른 하나는 $g(s_0 s_1 \cdots s_{n-1}) = \sum s_i P^{n-i-1}$을 이용합니다. 어떤 문자열 S가 팰린드롬인 것과 $f(S) = g(S)$인 것은 동치이므로 정수 2개만 이용해 문제를 해결할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr ll P1 = 917, M1 = 998244353;
constexpr ll P2 = 119, M2 = 993244853;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N; char C;
    ll X1 = 1, X2 = 1, H11 = 0, H12 = 0, H21 = 0, H22 = 0;
    while(cin >> C){
        H11 = (X1 * C + H11) % M1;
        H12 = (H12 * P1 + C) % M1;
        H21 = (X2 * C + H21) % M2;
        H22 = (H22 * P2 + C) % M2;
        X1 = X1 * P1 % M1;
        X2 = X2 * P2 % M2;
    }
    if(H11 == H12 && H21 == H22) cout << "TAK";
    else cout << "NIE";
}
```

BOJ 11590 SAVEZ

문자열 s를 마지막 원소로 하는 가장 긴 부분 수열의 길이를 $D(s)$라고 정의합시다. $D(s)$는 s의 접두사이면서 접미사인 부분 문자열 s'에 대해 $D(s') + 1$의 최댓값입니다.

어떤 문자열 s의 접두사이면서 동시에 접미사인 부분 분자열은 KMP 알고리즘의 실패 함수를 따라가면

서 모두 구할 수 있습니다. 따라서 DP의 상태를 문자열이 아닌 해시값을 `std::unordered_map` 등을 이용해 관리하면 문제를 효율적으로 해결할 수 있습니다.

해시맵을 사용하는 경우 전체 시간 복잡도는 $O(\sum |S|)$입니다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr ll P = 917, M = 998244353;

struct Hash{
    vector<int> h, p;
    void build(const string &s){
        int n = s.size();
        h.resize(n+1); for(int i=1; i<=n; i++) h[i] = (h[i-1] * P + s[i-1]) % M;
        p.resize(n+1); p[0] = 1; for(int i=1; i<=n; i++) p[i] = p[i-1] * P % M;
    }
    int get(int l, int r){
        l++; r++;
        int res = (h[r] - 1LL * h[l-1] * p[r-l+1]) % M;
        return res >= 0 ? res : res + M;
    }
};

int N, R;
map<int,int> D;

vector<int> GetFail(const string &s){
    int n = s.size();
    vector<int> fail(n);
    for(int i=1, j=0; i<n; i++){
        while(j > 0 && s[i] != s[j]) j = fail[j-1];
        if(s[i] == s[j]) fail[i] = ++j;
    }
    return fail;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++){
        string s; cin >> s;
        Hash hash; hash.build(s);
        auto fail = GetFail(s);

        int res = 1;
        for(int j=s.size(); j>=1; j=fail[j-1]){
            int now = hash.get(0, j-1);
            auto it = D.find(now);
            if(it != D.end()) res = max(res, it->second+1);
        }
        R = max(R, res);
        D[hash.get(0, (int)s.size()-1)] = res;
    }
    cout << R;
}
```

BOJ 26415 Ghost

halin graph의 treewidth가 3 이하인 tree decomposition을 구하는 문제입니다.

트리의 가장 왼쪽 리프 정점 l과 오른쪽 리프 정점 r이 연결되어야 하므로, l, r을 모두 포함하는 bag이 존재해야 합니다. 각 정점 v를 루트로 하는 서브 트리의 tree decomposition에서, 루트 정점의 bag에 (v, l, r)을 저장하는 방식으로 만들어 봅시다.

v가 리프 정점이면 $l = r = v$입니다. 따라서 (v, v, v)를 만들면 됩니다.

이제 v가 한 개 이상의 자식 정점을 갖고 있는 경우를 생각해 봅시다. v는 v의 자식 정점 c와 연결되어 있으므로 v, c를 포함하는 bag이 존재해야 합니다. (c, l, r)의 부모 정점으로 (v, l, r, c)를 달면 됩니다. c와 연결된 모든 정점을 처리했으므로 더 이상 c를 신경쓸 필요가 없다는 것에 주목합시다.

남은 것은 자식 정점의 정보 $(v, l_1, r_1, c_1), (v, l_2, r_2, c_2), \cdots, (v, l_k, r_k, c_k)$가 주어졌을 때, 이 정보들을 하나의 정보 (v, l, r)로 합치는 것 뿐입니다. 인접한 두 서브 트리를 합치는 방식으로 진행합니다.

(v, s, e, c_1)과 (v, l, r, c_2)를 합치는 방법을 생각해 봅시다. c_1, c_2를 무시할 수 있으므로 dummy를 의미하는 d_1, d_2로 치환해서 $(v, s, e, d_1), (v, l, r, d_2)$로 표기하겠습니다.

왼쪽 서브 트리의 오른쪽 리프 정점 e는 오른쪽 서브 트리의 왼쪽 리프 정점 l과 연결되어야 합니다. $(v, s, e, d_1) \to (v, s, e, l)$을 만들어 봅시다. e와 연결된 모든 정점을 처리했기 때문에 이제부터 e를 무시할 수 있습니다.

지금 만든 bag인 (v, s, e, l)과 두 번째 서브 트리를 담당하는 bag인 (v, l, r, d_2) 모두 l을 포함하고 있기 때문에, 두 bag은 l을 포함하는 bag들로 연결되어야 합니다. (v, s, r, l)을 만들어서 두 bag과 연결하면 됩니다. 이제 l도 무시할 수 있습니다.

합쳐진 두 서브 트리의 왼쪽 리프 정점인 s와 오른쪽 리프 정점인 r이 같은 bag에 포함되었으므로 두 서브 트리를 성공적으로 합쳐졌습니다. 또한 4번째 원소는 무시해도 되는 정점이기 때문에 이 절차를 반복해서 모든 서브 트리를 합칠 수 있습니다.

서브 트리를 모두 합치면 (v, l, r, d) 꼴의 bag 하나를 얻을 수 있습니다. 최종적으로 원하는 결과는 (v, l, r)이므로 (v, l, r, d)의 부모 bag (v, l, r)을 만들어서 연결합시다.

이 과정을 구현하면 $O(N)$ 시간에 treewidth가 3인 tree decomposition을 구할 수 있습니다.

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

struct Container{
    vector<int> par;
    vector<vector<int>> bags;
    Container(const vector<int> &par, const vector<vector<int>> &bags) :
par(par), bags(bags) {}
};

// vertices are labelled by pre-order
Container HalinDecompose(int n, int root, const vector<vector<int>> &gph){
    vector<int> par;
    vector<vector<int>> bags;
    auto make_node = [&](int parent, vector<int> children, vector<int> bag) ->
int {
        int node_id = bags.size();
        for(auto child : children) par[child] = node_id;
        par.push_back(parent); bags.push_back(bag);
```

```
        return node_id;
    };

    // return id of bag {vertex, left_leaf, right_leaf}
    function<int(int)> dfs = [&](int v) -> int {
        // leaf: {v, v, v}
        if(gph[v].empty()) return make_node(-1, {}, {v, v, v});

        vector<int> children_bag(gph[v].size());

        // 1. get subtree info: {c, l, r}
        // 2. connect c with v: {c, l, r} -> {v, l, r, c}
        // from now, we can ignore c
        for(int i=0; i<gph[v].size(); i++){
            int c = gph[v][i], sub = dfs(c);
            int l = bags[sub][1], r = bags[sub][2];
            children_bag[i] = make_node(-1, {sub}, {v, l, r, c});
        }

        // merge two adjacent subtree {v, s, e, dummy1} and {v, l, r, dummy2}
        // 1. connect e with l: {v, s, e, d} -> {v, s, e, l}
        // 2. drop e and keep l condition: {v, s, e, l} -> {v, s, r, l}
        //                                 {v, l, r, d} /
        // from now, we can ignore l
        int res = children_bag[0];
        for(int i=1; i<children_bag.size(); i++){
            int nxt = children_bag[i];
            int s = bags[res][1], e = bags[res][2];
            int l = bags[nxt][1], r = bags[nxt][2];
            int conn = make_node(-1, {res}, {v, s, e, l});
            res = make_node(-1, {conn, nxt}, {v, s, r, l});
        }

        // {v, l, r, dummy} -> {v, l, r}
        return make_node(-1, {res}, {v, bags[res][1], bags[res][2]});
    };

    int rt = dfs(root);
    for(auto &bag : bags){
        sort(bag.begin(), bag.end());
        bag.erase(unique(bag.begin(), bag.end()), bag.end());
    }
    return {par, bags};
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<vector<int>> G(N);
    for(int i=1,p; i<N; i++) cin >> p, G[--p].push_back(i);
    auto [par,bags] = HalinDecompose(N, 0, G);

    cout << bags.size() << "\n";
    for(const auto &bag : bags){
        cout << bag.size() << " ";
        for(int i=0; i<bag.size(); i++) cout << bag[i] + 1 << " \n"
[i+1==bag.size()];
    }
```

```
    for(int i=0; i<par.size(); i++) if(par[i] != -1) cout << par[i] + 1 << " "
<< i + 1 << "\n";
}
```

BOJ 17674 특별관광도시

선택되지 않은 간선의 가중치 합을 최소화하는 문제입니다. 반대로 선택된 간선의 가중치 합을 최대화하는 문제로 생각해서 해결해 봅시다.

Subtask 2. $Q = 1, E_1 = 1$ **(7점)**

선택한 정점이 트리의 루트라고 생각합시다. 루트로 올라가는 간선의 가중치를 최소화하는 문제입니다.

Tree DP를 이용하면 $O(N)$ 전처리를 통해 각 정점이 루트인 경우에 대한 답을 상수 시간에 구할 수 있습니다. 이 문제에 도전할 정도의 실력이라면 다들 알고 있을 것이라 믿습니다.

[코드]

Subtask 4. $N \le 2\,000$ **(30점)**

루트를 고정하는 Subtask 2의 아이디어는 그대로 가져갑니다. 추가로, 루트 정점과 리프 정점만 선택해도 정답을 찾을 수 있다는 점을 관찰해야 합니다. 리프 정점이 아닌 두 정점을 선택하는 것이 최적이라면, 자식 정점을 대신 선택해도 같거나 더 좋은 답을 낼 수 있다는 점을 생각해 보면 좋습니다.

루트가 고정된 상태에서 정점을 선택하는 것은 세 가지 경우로 나눌 수 있습니다.

1. 루트만 선택 (정점 1개 선택)
2. 루트 정점을 선택하고 리프 정점을 1개 이상 선택
3. 루트 정점을 선택하지 않고 리프 정점을 2개 이상 선택

세 가지 경우 모두 루트로 향하는 간선이 전부 선택된다는 점을 관찰할 수 있습니다. 단, 세 번째 경우에서는 두 개 이상의 서로 다른 서브 트리에서 리프를 선택해야 합니다. 2, 3번 경우에서 어떤 간선들이 추가로 선택되는지 알아봅시다.

두 번째 경우부터 살펴보겠습니다. 두 번째 경우에서는 루트에서 선택한 정점으로 가는 간선이 추가로 선택됩니다. 따라서 이 경우에는 루트가 있는 트리에서 정점 K개를 선택했을 때 루트에서 선택한 정점으로 가는 간선들의 가중치 합을 최대화하면 됩니다. 이런 문제는 DP 또는 그리디를 이용해 해결할 수 있습니다.

$D(v, k)$를 v를 루트로 하는 서브 트리에서 리프를 k개 선택했을 때의 최댓값이라고 정의합시다. DP를 MCMF로 모델링할 수 있기 때문에 $D(v, *)$는 볼록합니다. 따라서 $D(u, *)$와 $D(v, *)$를 $D(r, k_1 + k_2) \leftarrow D(u, k_1) + D(v, k_2)$와 같이 합칠 때 기울기가 큰 것부터 하나씩 끼워넣으면 됩니다. 우선순위 큐와 Small to Large를 이용해 $O(N \log^2 N)$에 $D(root, *)$를 모두 구할 수 있습니다.

이 DP 풀이를 응용하면 그리디 기법을 이용해 $O(N \log N)$ 시간에 해결할 수도 있습니다. $D(v, k) - D(v, k-1)$은 k번째로 정점을 선택했을 때 추가되는 경로를 의미합니다. $D(v, *)$에 저장되어 있는 경로 중 v보다 위로 확장될 수 있는 경로는 $D(v, 1) - D(v, 0)$ 뿐이고, 다른 나머지 경로들은 v 밑에서 끊어져서 더 이상 연장되지 않습니다.

끊어진 경로들은 굳이 Small to Large에서 주고 받을 필요가 없고, 전역에 선언되어 있는 우선순위 큐 하나에서 모두 관리해도 무방합니다. 즉, $D(v, *)$에서 v 밑에 있는 모든 경로를 관리하는 대신 가장 긴 경로 하나만 관리하고, 다른 나머지 경로는 전역에 있는 우선순위 큐에서 관리할 수 있습니다. 이때의 시간 복잡도는 $O(N \log N)$입니다.

한국에서는 [KOI 2013 고등부 4번. 수족관 3]의 풀이로도 잘 알려져 있습니다.

따라서 두 번째 경우는 $O(N \log N)$ 시간에 처리할 수 있습니다.

이제 세 번째 경우를 살펴보겠습니다. 세 번째 경우에서는 두 개 이상의 서로 다른 서브 트리에서 리프 정점을 선택해야 합니다. 한쪽에서만 정점을 뽑으면, 그 서브 트리에서 루트로 올라가는 간선이 선택되지 않을 수 있기 때문입니다.

사실 세 번째 경우도 두 번째 경우와 비슷하게 처리할 수 있습니다. 각 경로가 어떤 서브 트리에서 유래했는지 함께 저장한 다음, 가장 큰 경로와 다른 서브 트리에서 유래한 가장 큰 경로를 강제로 포함시키면 됩니다. 따라서 세 번째 경우도 $O(N \log N)$ 시간에 처리할 수 있습니다.

루트가 고정되어 있을 때 $O(N \log N)$ 만큼 걸리므로 전체 시간 복잡도는 $O(N^2 \log N)$입니다.

코드

Subtask 6. (100점)

Subtask 4의 풀이에 Centroid Decomposition을 적용하면 $O(N^2 \log N)$을 $O(N \log^2 N)$으로 줄일 수 있습니다.

Subtask 4 코드에 센트로이드 관련 처리 부분만 추가하면 됩니다.

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

ll N, Q, Sum, C1, C[202020], R[202020];
vector<pair<ll,ll>> G[202020];
int S[202020], U[202020];

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

void TreeDP(int v, int b=-1, ll up=0, ll dw=0){
    for(auto [i,w] : G[v]) if(i == b) C1 += w, up += w;
    C[v] = dw - up;
    for(auto [i,w] : G[v]) if(i != b) TreeDP(i, v, up, dw+w);
}

ll CostToRoot(int root){
    return C1 + C[root];
}

vector<pair<ll,ll>> PathsFromRoot(int root){
    vector<pair<ll,ll>> paths;
    function<ll(int,int,int)> dfs = [&](int st, int v, int b) -> ll {
        ll mx = 0;
        for(auto [i,w] : G[v]){
            if(i == b || U[i]) continue;
            ll nxt = dfs(st, i, v) + w;
            if(nxt > mx) swap(mx, nxt);
```

```
            if(mx != 0) paths.emplace_back(nxt, st);
        }
        return mx;
    };
    for(auto [i,w] : G[root]) if(!U[i]) paths.emplace_back(dfs(i, i, root) + w,
i);
    return paths;
}

void Go(int root){
    root = GetCent(root, GetSize(root)); U[root] = 1;

    ll to_root = CostToRoot(root);
    vector<pair<ll,ll>> paths = PathsFromRoot(root);
    sort(paths.begin(), paths.end(), greater<>());

    // case 1. only root
    R[1] = max(R[1], to_root);
    if(paths.empty()) return;

    // case 2. root and some leaves
    ll cost2 = to_root + paths[0].first;
    R[2] = max(R[2], cost2);
    for(int i=1; i<paths.size(); i++) R[i+2] = max(R[i+2], cost2 +=
paths[i].first);

    // case 3. only leaves
    int idx = find_if(paths.begin(), paths.end(), [&](auto v){ return
paths[0].second != v.second; }) - paths.begin();
    if(idx != paths.size()){
        ll cost3 = to_root + paths[0].first + paths[idx].first;
        paths.erase(paths.begin() + idx);
        R[2] = max(R[2], cost3);
        for(int i=1; i<paths.size(); i++) R[i+2] = max(R[i+2], cost3 +=
paths[i].first);
    }

    for(auto [i,w] : G[root]) if(!U[i]) Go(i);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<N; i++){
        int a, b, c, d; cin >> a >> b >> c >> d; Sum += c + d;
        G[a].emplace_back(b, c); G[b].emplace_back(a, d);
    }
    TreeDP(1);
    Go(1);
    for(int i=2; i<=N; i++) R[i] = max(R[i], R[i-1]);
    cin >> Q;
    for(int i=1,t; i<=Q; i++) cin >> t, cout << Sum - R[t] << "\n";
}
```