เส้นทางสุดน่าเบื่อ (Boring Path)

problem by JomnoiZ

โจทย์ : มีต้นไม้ขนาด N แต่ละโหนดมีค่า $A[i]$ $(1 \leq i \leq N, 1 \leq A[i] \leq 10^9)$ แต่ละ edge มีค่า $w[j]$ $(1 \leq j \leq N-1, 1 \leq w[j] \leq 10^9)$ สำหรับ edge แต่ละเส้น โจทย์ต้องการหา ผลคูณของค่า $A[i]$ ของ 2 โหนดใด ๆ ที่มากที่สุด ที่มี edge เส้นที่พิจารณาอยู่ เป็น edge ที่มีค่า $w[j]$ มากที่สุดใน path ระหว่าง 2 nodes นั้น ๆ

ความรู้ที่ใช้ : Disjoint Set Union + Depth First Search

Subtask #1 $(N \leq 2\,000)$

สามารถทำ brute force ตรง ๆ โดยการพิจารณา edge ทีละเส้น สำหรับ edge แต่ละเส้นจะทำการ DFS 2 รอบ จากคู่โหนดของ edge นั้น เพื่อหาโหนดที่มีค่า $A[i]$ มากที่สุด และมีค่า $w[j]$ ตลอดเส้นทางน้อยกว่าหรือเท่ากับค่า $w[j]$ ของ edge ที่กำลังพิจารณาอยู่ จากนั้นนำค่า $A[i]$ ที่ได้จากการทำ DFS ทั้ง 2 รอบมาคูณกันเป็นคำตอบ

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

const int MAX_N = 2005;

int A[MAX_N];
int U[MAX_N], V[MAX_N], W[MAX_N];
vector <pair <int, int>> graph[MAX_N];

int dfs(int u, int p, int maxW) {
    int maxA = A[u];
    for(auto [v, w] : graph[u]) {
        if(w <= maxW and v != p) {
            maxA = max(maxA, dfs(v, u, maxW));
        }
    }
    return maxA;
}

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);

    int N;
    cin >> N;
    for(int i = 1; i <= N; i++) {
        cin >> A[i];
    }
    for(int i = 1; i <= N - 1; i++) {
        cin >> U[i] >> V[i] >> W[i];

        graph[U[i]].emplace_back(V[i], W[i]);
        graph[V[i]].emplace_back(U[i], W[i]);
    }

    for(int i = 1; i <= N - 1; i++) {
        int maxFromU = dfs(U[i], V[i], W[i]);
        int maxFromV = dfs(V[i], U[i], W[i]);

        cout << 1ll*maxFromU * maxFromV << '\n';
    }
    return 0;
}
```

Time Complexity : $O(N^2)$

Subtask #2 ($w[j]$ แตกต่างกันทั้งหมด)

Observation 1 : จาก subtask 1 สังเกตว่า edge ที่มีค่า $w[j]$ มากกว่าค่า $w[j]$ ของ edge ที่กำลังพิจารณา จะไม่ถูกนำมาคำนวณเลย (ไม่สามารถเดินทางไปยังโหนดถัดไปได้ด้วย edge เส้นนั้น)

จากข้อสังเกตที่ 1 เราสามารถ sort ค่า $w[j]$ ของแต่ละ edge จากน้อยไปมาก แล้วทำ Disjoint Set Union (DSU) เพื่อเชื่อมโหนดตามค่า $w[j]$ สำหรับ parent node ของแต่ละ component จะเก็บค่า $A[i]$ ที่มากที่สุดของทุกโหนดใน component นั้น เมื่อจะทำการเชื่อม 2 component เข้าด้วยกัน จะนำค่า $A[i]$ ของทั้ง 2 component มาคูณกันแล้วเก็บเป็นคำตอบ ก่อนที่จะทำการเชื่อม 2 component นั้น

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

const int MAX_N = 1e5 + 5;

int A[MAX_N];
long long ans[MAX_N];
int parent[MAX_N];
int maxA[MAX_N];

struct Edge {
    int u, v, w, i;

    bool operator<(const Edge &o) const {
        return w < o.w;
    }
};
vector <Edge> edges;

int find_parent(int u) {
    if(u == parent[u]) {
        return u;
    }
    return parent[u] = find_parent(parent[u]);
}

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);

    int N;
    cin >> N;
    for(int i = 1; i <= N; i++) {
        cin >> A[i];
    }
```

```
    for(int i = 1; i <= N - 1; i++) {
        int u, v, w;
        cin >> u >> v >> w;

        edges.push_back({u, v, w, i});
    }

    sort(edges.begin(), edges.end());

    for(int i = 1; i <= N; i++) {
        parent[i] = i;
        maxA[i] = A[i];
    }
    for(auto [u, v, w, i] : edges) {
        u = find_parent(u), v = find_parent(v);

        ans[i] = 1ll*maxA[u] * maxA[v];

        parent[v] = u;
        maxA[u] = max(maxA[u], maxA[v]);
    }

    for(int i = 1; i <= N - 1; i++) {
        cout << ans[i] << '\n';
    }
    return 0;
}
```

Time Complexity : $O(N\ log\ N)$

Subtask 3 (edge มีค่า $w[j]$ ใด ๆ ซ้ำกันไม่เกิน 20 เส้น)

กำหนดให้ K แทนจำนวนครั้งที่ซ้ำกันมากที่สุดของค่า $w[j]$ ใด ๆ

Observation 2 : ในกรณีที่แย่ที่สุด (worst case) เราจะมีค่า $w[j]$ ที่ต้องพิจารณาทั้งหมดไม่เกิน N/K ค่า ทำให้ Time Complexity รวมในการคำนวณสำหรับ edge ทุกเส้นจะเป็น $O(N/K \times$ เวลาในการคำนวณสำหรับแต่ละค่า $w[j])$

เราจะใช้วิธีการเดียวกับ subtask 2 โดยเราจะใช้ DSU เชื่อม edge ทุกเส้นที่มีค่า $w[j]$ เท่ากับค่า $w[j]$ ของ edge เส้นที่กำลังพิจารณา จากนั้นหาคำตอบด้วยวิธีการเดียวกับ subtask 2 จากนั้นยกเลิกการเชื่อมต่อของ edge ทุกเส้นที่มีค่า $w[j]$ เท่ากับค่า $w[j]$ ของ edge ที่กำลังพิจารณา ก่อนที่จะพิจารณา edge เส้นถัดไปที่มีค่า $w[j]$ เท่ากัน โดยการยกเลิกการเชื่อมต่อ edge อาจทำได้โดยใช้การ union by rank/union by size แทนการทำ path compression แล้วทำการยกเลิกการเชื่อมต่อของ edge ในลำดับย้อนกลับ (reverse order) โดยอาจใช้ STL stack ในการ implement โดยจะใช้ Time Complexity สำหรับการคำนวณแต่ละค่า $w[j]$ เป็น $O(K^2\ log\ N)$ และ

เนื่องจากมีค่า $w[j]$ ที่ต้องพิจารณาไม่เกิน N/K ค่า ดังนั้น Time Complexity รวมจึงเป็น $O(NK\ log\ N)$

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

const int MAX_N = 1e5 + 5;

int A[MAX_N];
long long ans[MAX_N];
int parent[MAX_N], sz[MAX_N];
int U[MAX_N], V[MAX_N], W[MAX_N];
int maxA[MAX_N];
stack <tuple <int, int, int, int>> stk;

struct Edge {
    int u, v, i;

    bool operator!=(const Edge &o) const {
        return u != o.u or v != o.v;
    }
};
map <int, vector <Edge>> edges;

int find_parent(int u) {
    if(u == parent[u]) {
        return u;
    }
    return find_parent(parent[u]);
}

void merge(int &u, int &v) {
    u = find_parent(u), v = find_parent(v);
    if(u == v) {
        return;
    }

    if(sz[u] < sz[v]) {
        swap(u, v);
    }
    sz[u] += sz[v];
    stk.emplace(u, v, maxA[u], maxA[v]);
    parent[v] = u;
    maxA[u] = max(maxA[u], maxA[v]);
}
```

```
void rollback(int timer) {
    while(stk.size() > timer) {
        auto [u, v, mu, mv] = stk.top();
        stk.pop();

        sz[u] -= sz[v];
        maxA[u] = mu;
        maxA[v] = mv;
        parent[u] = u;
        parent[v] = v;
    }
}

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);

    int N;
    cin >> N;
    for(int i = 1; i <= N; i++) {
        cin >> A[i];
    }
    for(int i = 1; i <= N - 1; i++) {
        int u, v, w;
        cin >> u >> v >> w;

        edges[w].push_back({u, v, i});
    }

    for(int i = 1; i <= N; i++) {
        parent[i] = i;
        sz[i] = 1;
        maxA[i] = A[i];
    }
    for(auto [w, edge] : edges) {
        int sz = stk.size();
        for(auto e1 : edge) {
            for(auto e2 : edge) {
                if(e1 != e2) {
                    merge(e2.u, e2.v);
                }
            }

            int pU = find_parent(e1.u);
```

```
            int pV = find_parent(e1.v);

            ans[e1.i] = 1ll*maxA[pU] * maxA[pV];

            rollback(sz);
        }

        for(auto e : edge) {
            merge(e.u, e.v);
        }
    }

    for(int i = 1; i <= N - 1; i++) {
        cout << ans[i] << '\n';
    }
    return 0;
}
```

Time Complexity : $O(NK\ log\ N)$

Subtask 4 (ไม่มีเงื่อนไขเพิ่มเติม)

solution by neonaht

เราจะใช้วิธีการคล้ายกับ subtask 3 แต่สำหรับ edge แต่ละเส้นที่มีค่า $w[j]$ ซ้ำกัน เราจะทดลองสร้างกราฟ โดยนำ parent node ของแต่ละ component มาเชื่อมกัน จากนั้นทำการ DFS เหมือน subtask 1 เพื่อหาคำตอบ โดยจะทำการ DFS ไม่เกิน K รอบ แต่ละรอบจะมีจำนวนโหนดไม่เกิน K โหนดสำหรับแต่ละค่า $w[j]$ ดังนั้นจะได้ Time Complexity สำหรับการคำนวณแต่ละค่า $w[j]$ เท่ากับ $O(K^2)$ และเนื่องจากมีค่า $w[j]$ ที่ต้องพิจารณาไม่เกิน N/K ค่า ดังนั้น Time Complexity รวมกับการ sort จึงเป็น $O(NK + N\ log\ N)$ ซึ่งเร็วพอที่จะผ่าน subtask นี้

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

const int MAX_N = 1e5 + 5;

int A[MAX_N], maxA[MAX_N];
int parent[MAX_N];
long long ans[MAX_N];
vector <int> graph[MAX_N];

struct Edge {
    int u, v, i;
};
```

```
map <int, vector <Edge>> edges;

int find_parent(int u) {
    if(u == parent[u]) {
        return u;
    }
    return parent[u] = find_parent(parent[u]);
}

void merge(int u, int v) {
    u = find_parent(u), v = find_parent(v);
    if(u == v) {
        return;
    }

    parent[v] = u;
    maxA[u] = max(maxA[u], maxA[v]);
}

int dfs(int u, int p) {
    int max_A = maxA[u];
    for(auto v : graph[u]) {
        if(v != p) {
            max_A = max(max_A, dfs(v, u));
        }
    }
    return max_A;
}

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);

    int N;
    cin >> N;
    for(int i = 1; i <= N; i++) {
        cin >> A[i];
    }
    for(int i = 1; i <= N - 1; i++) {
        int u, v, w;
        cin >> u >> v >> w;

        edges[w].push_back({u, v, i});
    }
```

```
    for(int i = 1; i <= N; i++) {
        parent[i] = i;
        maxA[i] = A[i];
    }
    for(auto [w, edge] : edges) {
        for(auto e : edge) {
            int u = find_parent(e.u), v = find_parent(e.v);
            graph[u].push_back(v);
            graph[v].push_back(u);
        }
        for(auto e : edge) {
            int u = find_parent(e.u), v = find_parent(e.v);

            ans[e.i] = 1ll*dfs(u, v) * dfs(v, u);
        }
        for(auto e : edge) {
            int u = find_parent(e.u), v = find_parent(e.v);
            graph[u].clear();
            graph[v].clear();
        }

        for(auto e : edge) {
            merge(e.u, e.v);
        }
    }

    for(int i = 1; i <= N - 1; i++) {
        cout << ans[i] << '\n';
    }
    return 0;
}
```

Time Complexity : $O(NK + N\,log\,N)$