Nikolas Mavrogeneiadis
gravitorious
August 23, 2024

Maximize the Root

Problem: *Link*
For the solution, check the next page.

We can solve this problem using dp in $O(n)$.

Consider the simple case that our tree has only 4 nodes and the root(node 2) has 3 children $(1, 3, 4)$. The value of node 2 is 5, means $a[2] = 5$. Also, $a[1] = 5, a[3] = 7, a[4] = 2$.

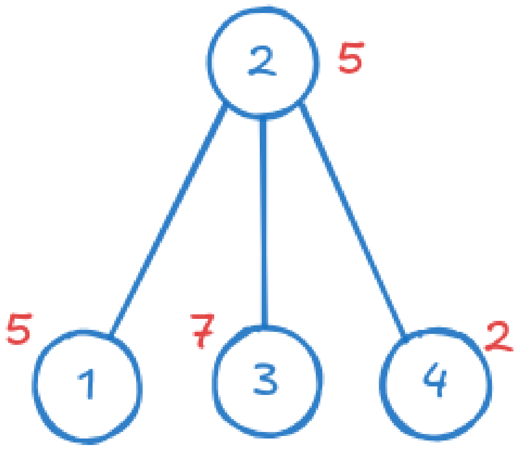

Figure 1: first test case

Obviously, the maximum value the root can get is 7 because if we increase the value of the root more than twice, the value of node 4 will become negative, which is forbidden. If one of the values of children was 0 then we wouldn't increase the value of the root.

To maximize the value that *root* can get, we may need to modify some node values from nodes that are part of some subtrees.

Consider the following tree.

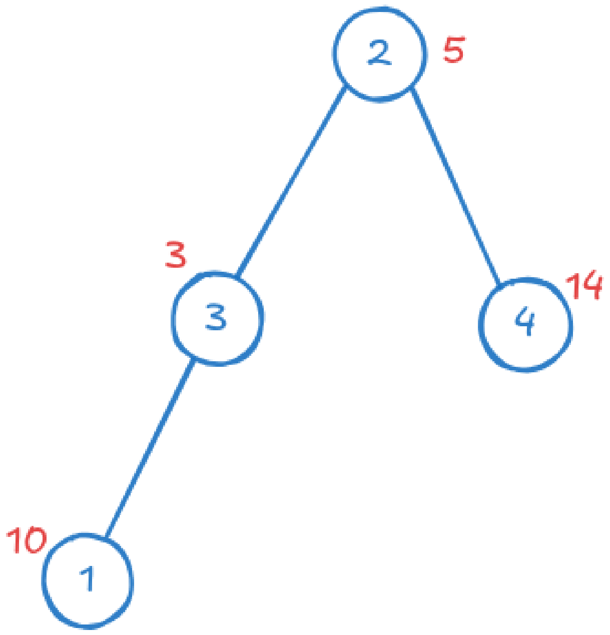

Figure 2: general case

So now the root has two children. The node 4 with value 14 and the node 3 with value 3. Before we say that can only increase the value of root only by 3, we can see the node 3 is a root of the subtree $\{3, 1\}$. This means that we can increase the value of node 3 without affecting the nodes 2 and 4. But what is the maximum value we can?

In fact, we can increase the value of node 3 until it becomes less than the value of node 1 by 1, or until they become equal. So the value of node 3 will become 6 and and the value of node 1 will become 7. That's optimal.

Consider the general case that the parent node has the value a and the child has the value b with $a < b$.

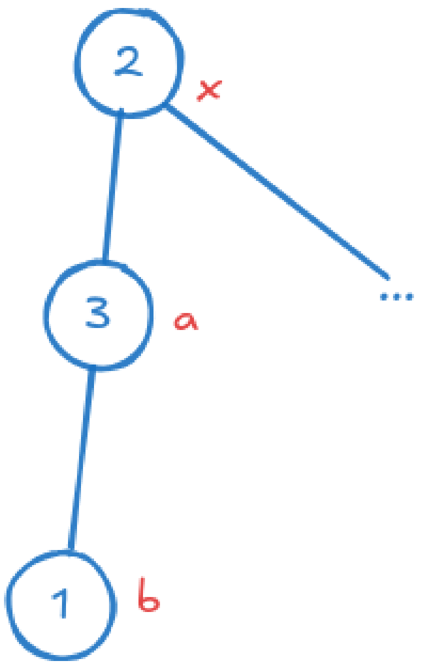

Figure 3: second test case

Then, $b - a = k > 0$ and k natural. If k is even then we can make a and b to be equal ($k = 2y$, $b - a = 2k => b - k - (a + k) = 0$). If k is odd, we can make them such that their difference is 1. To do that, we have to perfom $\lfloor \frac{b-a}{2} \rfloor = \lfloor \frac{k}{2} \rfloor$ operations. But why is this optimal? Until $k = 1$ or $k = 0$, we can always do better by continue increasing the value a and decreasing the value b, meaning that we can increase the parent of node 3 more. If a becomes greater than b, then if 2 tries to increase its value by a, then b will become negative. Hence we have to stop when $k = 0$ or $k = 1$ and return the value a to 2.

By the previous analysis, we can start repeating the above method for all subtrees of the tree starting from those with the smallest distance. The leaves of the tree return just their values. Now what happens when a node v takes all the values from its children? If the value of v is greater to at least one value of its children, then it is obvious that we can increase only by this value, so we return it. If the value of v is less than all of them, we pick the minimum of them and we perform the previous analysis (if don't pick the minimum and pick another, then the node with the minimum value will become negative).

Assuming that the array a has the values of each node, it holds that

$$dp[u] = \begin{cases} a[v] & \text{if v is a leaf} \\ min(dp[v]) & \text{if } a[u] \geqslant min(dp[v]) \forall v \; is \; child \; of \; v \\ a[u] + \lfloor \frac{(min(dp[v]) - a[u])}{2} \rfloor & \text{if } a[u] < min(dp[v]) \forall v \; is \; child \; of \; v \\ a[u] + min(dp[v]) & \text{if u is the initial root } \forall v \text{ is child of v} \end{cases} \tag{1}$$

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

#ifdef XOX
#include "/home/nikos/gravitonlib/mydebug/debug.h"
#else
#define debug(...) 77
#endif

using ll = long long;
using ld = long double;
using uint = unsigned int;
using ull = unsigned long long;

#define getunique(v) {sort(v.begin(), v.end()); v.erase(unique(v.begin(), v
    .end()), v.end());}

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

template <class T>
void chkmax(T &x,T y){
    if(x < y) x = y;
}

template <class T>
void chkmin(T &x,T y){
    if(x > y) x = y;
}

constexpr int popcnt(int x){
    return __builtin_popcount(x);
}

constexpr int ctz(int x){
    return __builtin_ctz(x);
}

constexpr double int_part(double x, double *intpart){
        return modf(x, intpart); //returns the real part
}

ll udiv(ll a, ll b) {
        return a / b + ((a ^ b) > 0 && a % b);
}  // divide a by b rounded up
ll ddiv(ll a, ll b) {
        return a / b - ((a ^ b) < 0 && a % b);
}  // divide a by b rounded down
```

```
long long myRand(long long B) {
        //0 to B-1
        return (unsigned long long)rng() % B;
}

int myUniRand(int a, int b){
        //a to b
        uniform_int_distribution<int> distribution(a,b);
        return distribution(rng);
}

void solve(){
        int n;
        cin >> n;
        vector<int> a(n);
        for(int i = 0; i < n; i++) cin >> a[i];
        vector<vector<int>> v(n, vector<int>());
        for(int i = 1; i <= n - 1; i++){
                int k;
                cin >> k;
                --k;
                v[k].push_back(i);
        }
        vector<bool> visited(n);
        vector<int> dp = a;
        function<int(int)> dfs = [&] (int u){
                int r = (int)1e9 + 10;
                bool child = true;
                for(const auto &x : v[u]){
                        if(!visited[x]){
                                child = false;
                                visited[x] = true;
                                r = min(r, dfs(x));
                        }
                }
                if(child) return a[u];
                if(u == 0) return a[u] + r;
                if(a[u] >= r) return r;
                else return a[u] + (r - a[u]) / 2;
        };
        visited[0] = true;
        int ans = dfs(0);
        cout << ans << '\n';
}

int main(){
        ios::sync_with_stdio(false);
        cin.tie(0);
        //freopen("input.txt", "r", stdin);
        //freopen("output.txt", "w", stdout);
        int tc;
        cin >> tc;
        while(tc--) solve();
        return 0;
}
```