【题解】2020 牛客 NOIP 赛前集训营-提高组（第三场）

提高 T1

对于 A，B 可以看做一个整体，其权值为 $A+B$ ，同时注意到每次变化后 $A+B+C$ 的权值其实并未发生改变，那么对 C 分析：

设 $S=A+B+C$

1. 若 $(A+B)>C$，则 $C=2C$，此时 $2C<S$，即 $C=2C\%S$

2. 否则 $C=C-(A+B)=C-(S-C)=2C-S$，此时 $2C>S$，等式 $C=2C\ \%S$ 依然成立。

综上，经过 K 次变换后，C 的通项公式为 $C*2^k\%S$。

使用快速幂可以做到单次询问 log 级别的时间复杂度。

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
inline int getint(){
    int summ=0,f=1;char ch;
    for(ch=getchar();!isdigit(ch)&&ch!='-';ch=getchar());
    if(ch=='-')f=-1,ch=getchar();
    for(;isdigit(ch);ch=getchar()) summ=(summ<<3)+(summ<<1)+ch-48;
    return summ*f;
}
inline int ksm(int x,int mi,int mod){
    int res=1;
    while(mi){
        if(mi&1) res=res*x%mod;
        x=x*x%mod;mi>>=1;
    }
    return res;
}
```

```
signed main(){
    int A,B,C,T;
    cin>>T;
    while(T--){
        A=getint(),B=getint(),C=getint();int k=getint();
        int S=A+B+C;
        cout<<C*ksm(2,k,S)%S<<"\n";
    }
    return 0;
}
```

提高 T2

- $10pts$ 想怎么做怎么做

- $20pts$ 对于每一个询问，暴力枚举l到r，再跑dfs枚举可以到达的点即可，复杂度$O(qn^2)$

- $40pts$ 建立最小生成树，将询问离线，每加入一条边，暴力对每个询问进行修改即可，复杂度$O(qn)$

- 另外$15pts$ 由于$l=r$，建立最小生成树，将询问离线，如果l介于这次加的边和上次加的边之间，则$ans=$上次加边后连通块内颜色个数，加边合并用启发式合并即可

- 另外$15pts$ 将1到10^5的答案预处理出来，$O(1)$询问即可

- $100pts$ 对于每一次加边，如果颜色变化，则记这条边为分割边，将所有分割边记录下来，考虑到颜色数量很少，所以分割边最多500条，每次暴力计算两个分割边间的答案即可

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define ll long long
#define cs const
#define fr first
```

```
#define se second
#define ls (now<<1)
#define rs (now<<1|1)
#define mid ((l+r)>>1)
#define mp make_pair
#define pb push_back
#define ppb pop_back
#define low(i) (i&(-i))
#define par pair<int,int>
#define cn(x) memset(x, 0, sizeof(x))
#define rep(i, x, y) for(int i=x; i<=y; ++i)
#define sep(i, x, y) for(int i=x; i>=y; --i)
#define fore(i, x) for(int i=fir[x]; i; i=nex[i])
cs int G = 3;
cs int ff = 1e6 + 15;
cs int inf = 1e18 + 1;
cs int base = 4;
cs int M = 1e9 + 7;
cs int p = 1e18 + 7;
struct Node { int u, v, w; } e[ff];
struct node { int l, r, x; } Q[ff];
bool cmp(Node x, Node y) { return x.w < y.w; }
int n, m, q, opt, c[ff], X, md, su[ff], dv[ff], top;
int fa[ff];
set<int> a[ff];
int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
void merge(int x, int y)
{
    x = find(x); y = find(y);
    if(x == y) return;
    int szx = a[x].size(), szy = a[y].size();
    if(szx < szy) // merge x to y
    {
        fa[x] = y;
        for(set<int>::iterator it = a[x].begin(); it!=a[x].end(); ++it)
            a[y].insert(*it);
    }
    else
    {
        fa[y] = x;
        for(set<int>::iterator it = a[y].begin(); it!=a[y].end(); ++it)
            a[x].insert(*it);
    }
}
```

```
int qry(int x, int y = 0)
{
    int tmp = 1, tt = 1;
    rep(i, 1, top)
    {
        if(e[dv[i]].w > x) break;
        tmp = e[dv[i]].w; tt = su[dv[i]];
        if(i != 1) y += (e[dv[i]].w - e[dv[i - 1]].w) * su[dv[i - 1]];
    }
    y += tt * (x - tmp + 1);
    return y;
}
void init()
{
    cin >> n >> m >> q >> X >> opt;
    if(opt) cin >> md;
    rep(i, 1, n) scanf("%lld", &c[i]);
    rep(i, 1, n) fa[i] = i, a[i].insert(c[i]);
    rep(i, 1, m)
        scanf("%lld %lld %lld", &e[i].u, &e[i].v, &e[i].w);
    sort(e + 1, e + 1 + m, cmp); su[0] = 1;
    dv[++top] = 0;
    rep(i, 1, m)
    {
        int u = e[i].u, v = e[i].v;
        merge(u, v); su[i] = a[find(X)].size();
        if(su[i] > su[i - 1]) dv[++top] = i;
    }
    int Ans = 0;
    rep(i, 1, q)
    {
        int L, R; scanf("%lld %lld", &L, &R);
        if(opt == 1)
        {
            int lw = (L ^ Ans) % md + 1;
            int rw = (R ^ Ans) % md + 1;
            if(lw > rw) swap(lw, rw);
            L = lw; R = rw;
        }
        Ans = qry(R) - qry(L - 1);
        printf("%lld\n", Ans);
    }
}
signed main()
```

```
{
    int Ts = 1;
    while(Ts--)
        init();
}
```

以上题解是$O(nlogn + qc)$的，这里再给出一种$O(nlogn + qlogc)$的

将所有临界点预处理出来，维护前缀和，二分找到终点对应的边，将剩下的加

上即可

```
//O(nlogn + qlogc)
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define ll long long
#define cs const
#define fr first
#define se second
#define ls (now<<1)
#define rs (now<<1|1)
#define mid ((l+r)>>1)
#define mp make_pair
#define pb push_back
#define ppb pop_back
#define low(i) (i&(-i))
#define par pair<int,int>
#define cn(x) memset(x, 0, sizeof(x))
#define rep(i, x, y) for(int i=x; i<=y; ++i)
#define sep(i, x, y) for(int i=x; i>=y; --i)
#define fore(i, x) for(int i=fir[x]; i; i=nex[i])

cs int G = 3;
cs int ff = 1e6 + 15;
cs int inf = 1e18 + 1;
cs int base = 4;
cs int M = 1e9 + 7;
cs int p = 1e18 + 7;

struct Node { int u, v, w; } e[ff];
struct node { int l, r, x; } Q[ff];
bool cmp(Node x, Node y) { return x.w < y.w; }
```

```
int n, m, q, opt, c[ff], X, md, su[ff], dv[ff], top; // when >= dv  ans
+ 1
int fa[ff];
set<int> a[ff];
int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
void merge(int x, int y)
{
    x = find(x); y = find(y);
    if(x == y) return;
    int szx = a[x].size(), szy = a[y].size();
    if(szx < szy) // merge x to y
    {
        fa[x] = y;
        for(set<int>::iterator it = a[x].begin(); it!=a[x].end(); ++it)
            a[y].insert(*it);
    }
    else
    {
        fa[y] = x;
        for(set<int>::iterator it = a[y].begin(); it!=a[y].end(); ++it)
            a[x].insert(*it);
    }
}

// 1 ~ x
// 1 ~ e[dv[i]].w - 1  e[dv[i]].w ~ x  -- > va = su[dv[i]]
// 1 ~ e[dv[top]].w - 1  e[dv[top]].w ~ x  --> va = su[dv[top]]
int sq[ff];
int qry(int x, int y = 0)
{
    int tmp = 1, tt = 1;
    int l = 1, r = top, ans = top + 1;
    while(l <= r)
        if(e[dv[mid]].w > x) ans = mid, r = mid - 1;
        else l = mid + 1;
    ans --;
    if(ans != 0) tmp = e[dv[ans]].w, tt = su[dv[ans]];
    y += sq[ans] + tt * (x - tmp + 1);
    return y;
}
void init()
{
    cin >> n >> m >> q >> X >> opt;
    if(opt) cin >> md;
```

```
    rep(i, 1, n) scanf("%lld", &c[i]);
    rep(i, 1, n) fa[i] = i, a[i].insert(c[i]);
    rep(i, 1, m)
        scanf("%lld %lld %lld", &e[i].u, &e[i].v, &e[i].w);
    sort(e + 1, e + 1 + m, cmp); su[0] = 1;
    dv[++top] = 0;
    rep(i, 1, m)
    {
        int u = e[i].u, v = e[i].v;
        merge(u, v); su[i] = a[find(X)].size();
        if(su[i] > su[i - 1]) dv[++top] = i;
    }
    rep(i, 2, top) sq[i] = sq[i - 1] + (e[dv[i]].w - e[dv[i - 1]].w) *
su[dv[i - 1]];
    int Ans = 0;
    rep(i, 1, q)
    {
        int L, R; scanf("%lld %lld", &L, &R);
        if(opt == 1)
        {
            int lw = (L ^ Ans) % md + 1;
            int rw = (R ^ Ans) % md + 1;
            if(lw > rw) swap(lw, rw);
            L = lw; R = rw;
        }
        Ans = qry(R) - qry(L - 1);
        printf("%lld\n", Ans);
    }
}
signed main()
{
    int Ts = 1;
    while(Ts--)
        init();
}
```

提高 T3

subtask 0

直接上 n^2 暴力。

一种可行解是对于每一个 1 操作，我们对该点进行 `dfs` 或 `bfs` ，更新其他点被更新到的最小时间，操作 2 就直接 `memset`，操作 3 直接看目前时间与最小时间的大小输出对应答案。

也可以对于 3 操作枚举前面每个修改操作，这里不多讲。

```
#include<bits/stdc++.h>
using namespace std;
int Read() {
    int x = 0, f = 1; char ch = getchar();
    while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
    while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch =
getchar();}
    return x * f;
}
int first[200005], nxt[200005], to[200005], tot;
void Add(int x, int y) {nxt[++tot] = first[x]; first[x] = tot; to[tot]
= y;}
int tim[100005];
void modify(int u, int f, int t) {
    if(!tim[u])  tim[u] = t;
    tim[u] = min(tim[u], t);
    for(int e = first[u]; e; e = nxt[e]) {
        int v = to[e];
        if(v == f)  continue;
        modify(v, u, t + 1);
    }
}
signed main() {
    freopen("Tree6.in", "r", stdin);
    freopen("1.ans", "w", stdout);
    int n = Read(), m = Read();
    for(int i = 1; i < n; i++) {
        int x = Read(), y = Read();
        Add(x, y); Add(y, x);
    }
    dfs(1, 0);
    for(int i = 1; i <= m; i++) {
        int opt = Read(), x = Read();
        if(opt == 1)  modify(x, 0, i);
        if(opt == 2)  memset(tim, 0, sizeof(tim));
        if(opt == 3) {
```

```
            if(tim[x] && tim[x] <= i)  puts("wrxcsd");
            else  puts("orzFsYo");
        }
    }
    return 0;
}
```

subtask 1

菊花图的深度为 2 ， 所以最多在 3 个单位时间后所有点都会被更新到，所以特判即可。

```
#include<bits/stdc++.h>
using namespace std;
int Read() {
    int x = 0, f = 1; char ch = getchar();
    while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
    while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch =
getchar();}
    return x * f;
}
int first[200005], nxt[200005], to[200005], tot = 0, vis[200005],
stk[55], tp;
void Add(int x, int y) {
    nxt[++tot] = first[x];
    first[x] = tot;
    to[tot] = y;
}
signed main() {
    int n = Read(), m = Read();
    for(int i = 1; i < n; i++) {
        int x = Read(), y = Read();
        Add(x, y); Add(y, x);
    }
    for(int i = 1; i <= m; i++) {
        int opt = Read(), x = Read();
        if(opt == 1 && tp < 3)
            stk[++tp] = x;
        if(opt == 2)  tp = 0;
        if(opt == 3 && tp && tp < 3)  stk[++tp] = 0;
        if(opt == 3) {
            if(tp == 3)  puts("wrxcsd");
            else {
                if(tp == 2 && stk[1] == 1)  puts("wrxcsd");
```

```
                else  if(tp == 2 && x == 1)  puts("wrxcsd");
                else  if(tp == 2 && (stk[1] == x || stk[2] == x))
puts("wrxcsd");
                else  if(tp == 1 && stk[1] == x)  puts("wrxcsd");
                else  puts("orzFsYo");
            }
        }

    }
    return 0;
}
```

subtask 2

对于链的情况，由于每个点度数至多为 2， 我们可以写一种基于度数的做法：在 1 操作时将询问的点加入队列，每个单位时间暴力更新所有点扩展到的节点，可以通过该 subtask。

subtask 3

这个子任务其实有 2 种做法，一种是暴力，因为树高为 $\log n$，所以在至多 $2 \cdot \log n$时间内所有点都会被更新到，枚举前面所有的修改，到没有修改或间隔时间大于最大时间时停止，输出答案即可。

另一种解法实际也是根据树高$\log n$来实现的。从第一次修改时一直到其后的 $2 \cdot \log n$个操作按照 subtask 1 的第二种方法暴力处理，之后的操作直接输出 `wrxcsd` 即可。

```
#include<bits/stdc++.h>
#define MAX 50
using namespace std;
int Read() {
    int x = 0, f = 1; char ch = getchar();
    while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
    while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch =
getchar();}
    return x * f;
}
int first[200005], nxt[200005], to[200005], tot = 0;
```

```
void Add(int x, int y) {
    nxt[++tot] = first[x];
    first[x] = tot;
    to[tot] = y;
}
int dep[100005], fa[100005], opt[100005], Ask[100005];
void dfs(int u, int f) {
    fa[u] = f;
    dep[u] = dep[f] + 1;
    for(int e = first[u]; e; e = nxt[e]) {
        int v = to[e];
        if(v == f)  continue;
        dfs(v, u);
    }
}
int getlca(int x, int y) {
    if(dep[x] < dep[y])  swap(x, y);
    while(dep[x] != dep[y])  x = fa[x];
    while(x != y)  x = fa[x], y = fa[y];
    return x;
}
int getdis(int x, int y) {
    return dep[x] + dep[y] - 2 * dep[getlca(x, y)];
}
signed main() {
    int n = Read(), m = Read();
    for(int i = 1; i < n; i++) {
        int x = Read(), y = Read();
        Add(x, y); Add(y, x);
    }
    dfs(1, 0);
    int flag = 0;
    for(int i = 1; i <= m; i++) {
        opt[i] = Read(), Ask[i] = Read();
        if(flag)  ++flag;
        if(opt[i] == 1)
            if(flag == 0)  ++flag;
        if(opt[i] == 2)  flag = 0;
        if(opt[i] == 3) {
            if(flag >= MAX) {
                puts("wrxcsd");
                continue ;
            }
            if(!flag) {
```

```
                puts("orzFsYo");
                continue ;
            }
            int fflag = 0;
            for(int j = i - flag + 1; j < i; j++) {
                if(opt[j] == 1)
                    if(getdis(Ask[i], Ask[j]) < i - j + 1)  fflag = 1,
puts("wrxcsd");
                if(fflag)  break;
            }
            if(!fflag)  puts("orzFsYo");
        }
    }
    return 0;
}
```

由于该代码的正确性是建立在平均树高上的，所以前 3 个 subtask 该代码都能以极为优秀的复杂度跑过。

subtask 3

其实上面的做法已经给了我们提示，我们对询问分块，块内的询问暴力处理，一块询问结束后暴力更新该块所产生的贡献，我用的是 ST 表在 $O(n\log n)$ 复杂度内预处理， $O(1)$ 求出 lca ，常数较为优秀的树剖也可过。

当然，点分树也可过，这里不详讲。

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
inline int getint() {
    int summ = 0, f = 1; char ch;
    for (ch = getchar(); !isdigit(ch) && ch != '-'; ch = getchar()) ;
    if (ch == '-') f = -1, ch = getchar();
    for (; isdigit(ch); ch = getchar())
        summ = (summ << 3) + (summ << 1) + ch - 48;
    return summ * f;
}
const int M = 1e6 + 5;
int n, m, etot, no, t[M], cntnow, dep[M], dfn[M], out[M], lg[M];
int st[500005], minn[1000005][25], ind;
int first[1000005], nxt[1000005], to[1000005], w[1000005], tot;
```

```
inline void Add(int x, int y) {
    nxt[++etot] = first[x];
    first[x] = etot;
    to[etot] = y;
}
void dfs(int u, int fa) {
    dfn[++ind] = u; dep[u] = dep[fa] + 1; st[u] = ind;
    for (int e = first[u]; e; e = nxt[e]) {
        int v = to[e];
        if (v == fa) continue;
        dfs(v, u); dfn[++ind] = u;
    }
}
inline int my_min(int x, int y) { return dep[x] < dep[y] ? x : y; }
void prework() {
    for (int i = 1; i <= ind; i++) minn[i][0] = dfn[i];
    for (int i = 1; i <= lg[n * 2]; i++)
        for (int j = 1; j + (1 << i) - 1 <= n * 2; j++)
            minn[j][i] = my_min(minn[j][i - 1], minn[j + (1 << (i -
1))][i - 1]);
}
int Getlca(int x, int y) {
    if (st[x] > st[y]) swap(x, y);
    int l = st[x], r = st[y], k = lg[r - l + 1];
    return my_min(minn[l][k], minn[r - (1 << k) + 1][k]);
}
inline int Getdis(int x, int y) {
    return dep[x] + dep[y] - 2 * dep[Getlca(x, y)];
}
struct node {
    int t, pos;
} p[M], now[M], pp[M];
int cntp = 0, vis[M / 2], mi[M / 2], bbj;
inline void RE() {
    memset(vis, 0, sizeof(vis));
    memset(mi, 0x7f, sizeof(mi));
    for (int i = 1; i <= cntnow; i++) p[++cntp] = now[i];
    for (int i = 1; i <= cntp; i++) mi[p[i].pos] = min(mi[p[i].pos],
p[i].t);
    queue<node> q; q.push(p[1]);
    int ll = 2;
    for (int i = 1; i <= cntnow; i++) mi[p[i].pos] = min(mi[p[i].pos],
p[i].t);
    while (!q.empty()) {
```

```
            node u = q.front(); q.pop();
            for (int i = first[u.pos]; i; i = nxt[i]) {
                int v = to[i];
                if (!vis[v]) {
                    vis[v] = 1; mi[v] = min(mi[v], mi[u.pos] + 1);
                    q.push((node){mi[v], v});
                    if (mi[v] == p[ll].t) {
                        vis[p[ll].pos] = 1;
                        q.push(p[ll]), ll++;
                    }
                }
            }
        }
    return;
}
void Qu(int u, int nowtime) {
    if ((!bbj) && (nowtime >= mi[u])) {
        puts("wrxcsd");
        return;
    }
    for (register int i = 1; i <= cntnow; i++) {
        if (Getdis(u, now[i].pos) <= nowtime - now[i].t) {
            puts("wrxcsd");
            return;
        }
    }
    puts("orzFsYo");
    return;
}
signed main() {
    int be = clock();
    memset(mi, 0x7f, sizeof(mi));
    lg[0] = -1;
    for (int i = 1; i <= 1000000; i++) lg[i] = lg[i / 2] + 1;
    cin >> n >> m;
    p[0].t = 1e9;
    for (int i = 1, u, v; i < n; i++) {
        u = getint(); v = getint();
        Add(u, v); Add(v, u);
    }
    dfs(1, 0); prework();
    int block = sqrt(m) * 2;
    for (int nn = 1, op, u; nn <= m; nn++) {
        if (nn % block == 0)
```

```
            RE(), cntnow = bbj = 0;
        op = getint(); u = getint();
        if (op == 1) {
            now[++cntnow] = (node){nn, u};
        }
        else if (op == 2) {
            bbj = 1;,cntnow = cntp = 0;
        }
        else
            Qu(u, nn);
    }
    return 0;

}
```

提高 T4 题解

首先勇士只会增加防，于是打每只怪的回合数是不变的。然后又因为在任何时候防都不可能大于怪物的攻，所以每时每刻都一定有伤害，所以 1 防对每只怪的效果是不变的。效果即是降低伤害，以下称作减伤。

可以这么考虑，最小化受到的伤害，相当于**最大化减伤**。

定义怪物 i 的回合数为 h_i ，拿到的蓝宝石数量为 b_i ，定义 $\frac{b_i}{h_i}$ 为一只怪的性价比，设为 t_i。

首先考虑**菊花图**的情况：考虑一个最优的打怪序列 $\{p_1,p_2 \dots,p_n\}$ ，若交换 p_i 和 p_{i+1} ，目前减伤的变化为 $b_{i+1} * h_i - b_i * h_{i+1}$ ，因为交换后的序列一定不更优，

于是有：$b_{i+1} * h_i - b_i * h_{i+1} \leq 0$

移项得：$\frac{b_i}{h_i} \geq \frac{b_{i+1}}{h_{i+1}}$

于是只需要按性价比排序，依次打即可。

然后考虑**菊花图加强版**的情况：用到了以下一个结论：**如果一只怪 a 挡在 b 前面（必须打 a 才能打 b），如果$t_b > t_a$，则打完 a 后立即打 b 一定最优。**

证明：假设存在一个最优的打法为：打完 a 后又打了一连串的怪$\{s1, s2 \dots sm\}$后才打 b，根据前面的证明，所有t_{s_i}一定大于t_b，（否则不会在 b 前面打），又因为$t_b > t_a$，所以所有$t_{s_i} > t_a$ ，那这一连串的怪应该**在 a 之前打会更优**，矛盾，于是不存在任何怪会在打了a之后打，然后打b，即打a之后会立即打 b。

于是可以从叶子开始，如果此节点b比父节点a的性价比高，就将两个节点用并查集缩为一个节点，缩完后整棵树就成了一个**以性价比为关键字的大根堆**。然后将当前能达到的节点的性价比为关键字放入堆中，依次取出最大的，并更新当前能达到的节点。最终得到的序列即是打怪顺序。

然后考虑**树**的情况：此时一只怪后面可能存在多只怪被挡住。仍然是之前的证明，可以证明如果子节点性价比比父节点更高，则打完父节点后一定就打子节点。于是有一个n^2的朴素做法：从叶节点开始，如果 a 比父节点 b 性价比高，就将其缩为一个节点，但此时**树的形态会改变**，于是需要将 a 的所有子节点合并到 b 的子节点下。缩完后也会是一个**大根堆**，每次打怪的时候，进入一个大点之后，每个大点内部处理一下即可。

发现一个大点的内部一定是一次性打完的，于是可以**整体考虑**一个大点，则这个大点以外的每 1 防对这整个大点的减伤为$\sum_i h_i$，同理，打完这一个大点会加$\sum_i b_i$的防御。于是合并时**不需要改变树的形态**，只需要把子节点 a 的参数合并到父节点 b 即可，即$b_b += b_a, h_b += h_a$。于是从叶子节点依次向上传导参数即可。复杂度 O(nlogn)。

```
#include<bits/stdc++.h>
#define int long long
```

```
using namespace std;
int Read() {
    int x = 0, f = 1; char ch = getchar();
    while(!isdigit(ch)) {if(ch == '-')  f = -1; ch = getchar();}
    while(isdigit(ch)) {x = (x << 3) + (x << 1) + ch - '0'; ch =
getchar();}
    return x * f;
}
int first[200005], nxt[200005], to[200005], tot = 0;
void Add(int x, int y) {nxt[++tot] = first[x]; first[x] = tot; to[tot]
= y;}
int fa[100005], b[100005], a[100005], d[100005], hh[100005],
val[100005], HH[100005], Val[100005], tim[100005];
int vis[100005], sc[100005];
int ffa[500005];
int findfa(int x) {return (ffa[x] == x) ? x : ffa[x] = findfa(ffa[x]);}
void fight(int x) {
    //cout << x << endl;
    b[1] -= (a[x] - d[1]) * hh[x];
    d[1] += val[x];
}
void dfs(int u, int F) {
    fa[u] = F;
    for(int e = first[u]; e; e = nxt[e]) {
        int v = to[e];
        if(v == F)  continue;
        dfs(v, u);
    }
}
vector<int> Nxt[100005];
void Do(int u) {
    fight(u); sc[u] = 1;
    for(int i = 0; i < Nxt[u].size(); i++) {
        Do(Nxt[u][i]);
    }
}
signed main() {
    priority_queue<pair<double, int> > q;
    int n; scanf("%lld", &n);
    for(int i = 1; i < n; i++) {
        int x, y;
        scanf("%lld%lld", &x, &y);
        Add(x, y); Add(y, x);
    }
```

```
    dfs(1, 0);
    scanf("%lld%lld%lld", &b[1], &a[1], &d[1]);
    for(int i = 2; i <= n; i++) {
        scanf("%lld%lld%lld%lld", &b[i], &a[i], &d[i], &val[i]);
        hh[i] = b[i] / (a[1] - d[i]);  HH[i] = hh[i]; Val[i] = val[i];
        if(b[i] % (a[1] - d[i]) == 0)  --hh[i], --HH[i];
        q.push(make_pair(1.0 * val[i] / hh[i], i));
    }
    sc[1] = 1;
    for(int i = 1; i <= n; i++)  ffa[i] = i;
    while(!q.empty()) {
        int u = q.top().second; q.pop();
        if(vis[u])  continue;  vis[u] = 1;
        if(sc[fa[u]])  {Do(u); continue;}
        HH[findfa(fa[u])] += HH[u], Val[findfa(fa[u])] += Val[u];
        Nxt[ffa[fa[u]]].push_back(u);
        ffa[u] = ffa[fa[u]];
        q.push(make_pair(1.0 * Val[ffa[fa[u]]] / HH[ffa[fa[u]]],
ffa[fa[u]]));
    }
    cout << b[1] << endl;
    return 0;

}
```