题解

软件

二分+DP。

二分答案。设答案为mid，f[i][j]表示前i个人完成了第一个软件的j个模块，此时还可以完成第二个软件的最多模块数。设第i个人要完成k个模块，那么前i-1个人完成了j-k个模块。完成第二个软件的模块数为(mid-k*d1[i])/d2[i]。那么方程出来了：

f[i][j]=max(f[i][j], f[i-1][j-k]+(mid-k*d1[i])/d2[i])

那么判断是否合法，就比较 f[n][m]]与m的大小，如果 f[n][m]≥m则说明可行。

最大后缀值个数

10pts

暴力即可。

50pts

发现这就是一条链的情况，也就相当于本题搬到了序列上。

根据后缀最大值的定义，不难发现序列上的后缀最大值的权值是递减的，而且后缀最大值的位置是递增的，这启发我们用单调栈维护所有后缀最大值。

具体的，单调栈内按权值递减的维护元素，当加进当前位置的元素时，所有权值小于它的栈内元素都不再是后缀最大值，将他们全部从栈内弹出。

最后单调栈内剩下的权值都是后缀最大值，时间复杂度$O(n)$。

100pts

树可以看做一些链并在一起，所以我们只需要对树进行Dfs，即可将树上问题变成序列上的问题。

然而树上Dfs就要求我们每次将一个元素加进单调栈，还要支持在Dfs回退的时候对单调栈上一次的操作进行撤销。

这样每次将一个元素加进单调栈的时候，我们二分出应该将它加进的位置。将该位置上的元素放到栈最后的缓冲区，将栈顶标记为当前位置。

同时再开一个新的栈，维护每次操作之前的栈顶位置，这样撤销的时候，只需要把缓冲区最上面的元素放到这个位置，再把栈顶设置为记录的位置即可。

时间复杂度$O(nlogn)$，二分的常数极小，可以通过本题。

树

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() {
    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 $\mathcal{O}(n\log n)$ 复杂度内预处理， $\mathcal{O}(1)$ 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 = 3e6 + 5;
int n, m, etot, no, t[M], cntnow, dep[M], dfn[M], out[M], lg[M];
int st[3000005], minn[3000005][26], ind;
int first[3000005], nxt[3000005], to[3000005], w[3000005], 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], mi[M], 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 (int i = 1; i <= cntnow; i++) {
        if (Getdis(u, now[i].pos) <= nowtime - now[i].t) {
            puts("wrxcsd");
            return;
        }
    }
    puts("orzFsYo");
    return;
}
signed main() {
    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) * 3;
    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;
}
```

魔塔

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 hi ，拿到的蓝宝石数量为 bi ，定义 bi/hi 为一只怪性价比，设为 ti。
首先考虑菊花图的情况：考虑一个最优的打怪序列 $\{p1, p2 \cdots, pn\}$ ，若交换 pi 和 $pi+1$ ，目前减伤的变化为 $bi+1 * hi - bi * hi+1$ ，因为交换后的序列一定不 更优， 于是有：$bi+1 * hi - bi * hi+1 \leqslant 0$
移项得：$bi / hi \geqslant bi+1 / hi+1$
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$\boldsymbol{tb} > \boldsymbol{ta}$，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$\{s1, s2 \cdots sm\}$后 才打 b，根据前面的证明，所有t_{si}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$t_{si} > t_a$ ，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，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$\sum_i hi$，同理，打完这一个大点会加$\sum_i \boldsymbol{bi}$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;
}
```