「题单」IOI2020 国家集训队作业 Part 1

Jiayi Su (ShuYuMo)

2020-10-19 17:05:04

CF505E Mr. Kitayuta vs. Bamboos

- 给定 n 个数（竹子的高度）$h_{1\dots n}$。
- 你需要进行 m 轮操作，每轮操作为 k 次修改，每次修改可以选择一个数 h_i 修改为 $\max(h_i - p, 0)$（砸到地下）。
- 每轮操作后每个 h_i 将会被修改为 $h_i + a_i$（每天竹子会生长）。
- 你需要最小化最终 $h_{1\dots n}$ 中的最大值。
- $n \le 10^5$，$m \le 5 \times 10^3$，$k \le 10$ 要求最小化最大值，考虑二分一个最大的高度，将问题转化成可行性问题，即，给出一个高度，问能否使这些竹子最终不超过这个高度。

考虑限制这个问题不平凡的因素：因为每次降低竹子高度的时候不能保证竹子一定降低 p ，也就是说，“砸竹子”这一动作存在浪费。我们考虑尽可能的让“砸竹子”这一动作更少的浪费，即：尽可能在后面的时间点砸竹子，但是无法保证到底是在哪一次砸竹子，有可能已经结束了，没有形式化的计算流程。

可以考虑反向思考，即：假设已经长到 二分的高度。每次生长操作相当于每次会下落 a_i，而每次砸的操作相当于提升了 p 的高度。

开始时，对每一个竹子计算出其需要多少次会下落到负高度，用堆维护，每次取出最快的可能下落到负高度的 k 个竹子，进行“拔高”操作。然后判断最后的高度是否大于等于 h_i。

其实二分的本质是为了确定“尽可能的靠后砸竹子”的标准。反向思考是为了便于贪心处理策略。## code

```cpp
int n, m, k, p;
int H[_], A[_];
#define mp make_pair
#define fi first
#define se second
int height[_];
bool check(int MaxH){
    fill(height + 1, height + 1 + n, MaxH);
    priority_queue<pair<int, int>, vector<pair<int, int> > , greater<pair<int, int> > > Q;
    while(!Q.empty()) Q.pop();
    for(int i = 1; i <= n; i++) if(MaxH - m * A[i] < H[i]) Q.push(mp( MaxH / A[i], i ));

    for(int i = 1; i <= m; i++) {
        for(int j = 1; j <= k; j++){
            if(Q.empty()) return true;
            pair<int, int> now = Q.top(); Q.pop();
            if(now.fi < i) return false;
            height[now.se] += p;
            now.fi = height[now.se] / A[now.se];
            if(height[now.se] - m * A[now.se] < H[now.se]) Q.push(now);
```

```
        }
    }
    return Q.empty();
}
signed main(){
    n = read(), m = read(), k = read(), p = read(); // 竹子数 天数 修改次数 最大降低高度
    for(int i = 1; i <= n; i++) H[i] = read(), A[i] = read();

    int L = 0, ans = 0, R = 1LL << 60; for(int i = 1; i <= n; i++) L = min(L, A[i]);
    while(L < R){
        int mid = (L + R) >> 1;
        if(check(mid)) ans = mid, R = mid;
        else L = mid + 1;
    }
    printf("%lld\n", ans);
    return 0;
}
```

CF521D Shop

- 有 k 个正整数 $a_{1\dots k}$。
- 有 n 个操作，每个操作给定正整数 b，有三种可能：将 a_i 赋值为 b，将 a_i 加上 b，将 a_i 乘以 b。
- 你可以从 n 个操作中选择最多 m 个操作，并按照一定顺序执行。
- 你的目标是最大化 $\prod_{i=1}^{k} a_i$ 的值。
- $k, n \leq 10^5$。

化归思想，考虑如果只有乘法，那一定是把操作数字从大到小排序，然后以此操作。

考虑操作顺序，最优的操作一定是先赋值，再加法，最后乘法。其中赋值只可能选相应位置最大的赋值，加法也是从大到小考虑。

显然赋值可以直接转化成加法，而加法如果从大到小考虑也可以直接转化成乘法（乘一个实数）。

这样就把所有操作转化成了乘法。贪心考虑即可。

最后输出有顺序，应按照上面的策略（先赋值，再加法，最后乘法）以此输出（操作顺序）。

code

```
#include <climits>
#include <cstring>
#include <cstdio>
#include <cmath>
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
const int _ = 1e5 + 100;
#define LL long long
int read() { int x; scanf("%d", &x); return x; }
```

```cpp
int n, m, k;
int v[_];
pair<int, int> Give[_];
vector<pair<int, int > > add[_];
vector<pair<double, int> > mul[_];
vector<pair<double, int> > All;
int T[_];
int id[_];
#define double long double
bool ICMP  (const pair<int, int> & x, const pair<int, int> & y) { return (x > y); }
bool ICMP_D(const pair<double, int> & x, const pair<double, int> & y) { return (x > y); }
bool MAGIC(const int &x, const int &y) { return T[x] < T[y]; }
int main(){
    k = read(), n = read(), m = read();
    for(int i = 1; i <= k; i++) v[i] = read();

    for(int i = 1; i <= n; i++){
        int type = read(); T[i] = type;
        int a = read(), b = read();
        if(type == 1){
            Give[a] = max(Give[a], make_pair(b, i));
        } else if(type == 2){
            add[a].push_back(make_pair(b, i));
        } else {
            mul[a].push_back(make_pair((double)(b), i));
        }
    }
    for(int i = 1; i <= k; i++)
        if(Give[i].first)
            add[i].push_back(make_pair(Give[i].first - v[i], Give[i].second));
    for(int i = 1; i <= k; i++) sort(add[i].begin(), add[i].end(), ICMP);
    for(int i = 1; i <= k; i++){
        LL now = v[i];
        for(int j = 0; j < add[i].size(); j++){
            mul[i].push_back(
                make_pair(
                    (double)(now +0ll+ add[i][j].first) / (double)(now),
                    add[i][j].second
                )
            ); now += add[i][j].first;
        }
    }
    for(int i = 1; i <= k; i++) for(int j = 0; j < mul[i].size(); j++) All.push_back(mul[i][j]);
    sort(All.begin(), All.end(), ICMP_D);
    int tot = 0;
    for(int i = 0; i < All.size(); i++) if(All[i].first > 1) { tot++; }
    tot = min(tot, m);
```

```
    for(int i = 1; i <= tot; i++) id[i] = All[i - 1].second;
    sort(id + 1, id + 1 + tot, MAGIC);  printf("%d\n", tot);
    for(int i = 1; i <= tot; i++) printf("%d%c", id[i], " \n"[i == n]);
    return 0;
}
```

CF526F Pudding Monsters

- 给定一个 $n \times n$ 的棋盘，其中有 n 个棋子，每行每列恰好有一个棋子。
- 求有多少个 $k \times k$ 的子棋盘中恰好有 k 个棋子。
- $n \leq 3 \times 10^5$。

每行每列恰好有一个棋子的棋盘，可以抽象成一个排列。而操作也是对于一个排列的一段进行操作的。

题意可以转化为：- 给出一个排列 - 求连续段数量，即连续的一段，且段中的元素排好序之后必须是公差为 1 的等差数列.

连续段计数

考虑如何刻画“连续的一段，且段中的元素排好序之后必须是公差为 1 的等差数列”这个条件，可以形式化的定义：

$$\text{MaxVal} - \text{MinVal} = (R - L)$$

即

$$f_R(L) = \text{MaxVal} - \text{MinVal} - (R - L) = 0$$

这个条件成立的基础在于，所求数列不能有重复元素，而且也保证了函数 $f_x(L) \geq 0$, 查询时只需要维护最小值以及最小值的数量即可。

可以考虑枚举一个端点，比如右端点 R，然后查找有多少符合条件的左端点 L ，用线段树维护 $f_R(x)$ 在每个点处的函数值。

考虑当 $f_x(L)$ 移动到 $f_{x+1}(L)$ 时，值的变化。变化的值有 R , 可能变化的有 $MaxVal, MinVal$。用单调栈实时维护更新后会波及哪些元素的最小值。事实上，单调栈处理的过程就是在动态维护一个后缀最小/大值数组。

这里有几个与其有相似之处的题目，但是可能做法并不一样，遇到不要想成一样的题目：- 算术天才 与等差数列 (不保证序列中一定元素不重复) - 考虑判断一个序列排好序后是否为等差数列 - 准确做法 - $\max - \min = (r - l)k$ - 相邻两数差的绝对值的 gcd 是 k - 区间 $[l, r]$ 内的数不重复 - 概率性做法 - hash 思想，维护 n 次方和 - 树上排列 (ZROI) (同样不保证元素一定不重复) - 给定一颗 n 个点的树。每个点都一个正整数点权 A_i ，你需要支持以下两种操作：- 1、询问点 x 和点 y 之间的路径上的所有点（包括点 x 和点 y ）的点权是否构成一个从 1 开始的排列（即若这条链长度为 len ，那么问点权集合为 $1, 2, \cdots, len$ ）。- 2、将 A_x 修改为 y 。- 由于在树上难以判断某区间内的数字不重复，Hash 维护 n 次方和。- 给出一个没有重复元素的序列，询问某区间中的元素是否能加入小于 k 个元素使其拍好序后构成一个等差数列。- 条件变成了

$$f_R(L) = \text{MaxVal} - \text{MinVal} - (R - L) \geq k$$

- count(ZROI)已知一个集合 S 中的最大元素为 N，且这个集合中的元素可以构成一个等差数列，给出一些形如 $x \in S$、$x \notin S$ 的限制。求最终的集合有多少种情况。- 考虑刻画一个等差数列需要什么参数：公差，和每个元素 mod 公差的值。枚举这两个参数，依次计数即可。

（引用正睿的两道题目，不知道是不是有版权问题，如果有，会立即删除（~~顺便给正睿 OI 打个广告~~））。

code

```
int n;
int A[_];
#define fir first
#define sec second
namespace SegmentTree{
    const int _ = 3e6 + 100;
    struct Node{
        int MIN;
        int sMIN;
        int tar;
        Node operator + (const Node & rhs) const {
            Node res;
            res.MIN  = min(MIN, rhs.MIN);
            res.sMIN = MIN == rhs.MIN ? ( sMIN + rhs.sMIN) : (MIN < rhs.MIN ? sMIN : rhs.sMIN) ;
            return res;
        }
    }v[_];
    int tot = 0;
    int ch[_][2];
    #define ls(o) (ch[o][0])
    #define rs(o) (ch[o][1])
    #define make (tot++, ch[tot][0] = ch[tot][1] = v[tot].MIN = v[tot].sMIN = 0, tot)
    int Groot() { return make; }
    void maintain(int o) {
        v[o].MIN  = min(v[ls(o)].MIN, v[rs(o)].MIN);
        v[o].sMIN = v[ls(o)].MIN == v[rs(o)].MIN ? v[ls(o)].sMIN + v[rs(o)].sMIN : ( v[ls(o)].MIN < v[r
    }
    void tar(int o, int x) { v[o].MIN += x; v[o].tar += x; }
    void pushdown(int o){
        if(v[o].tar){
            tar(ls(o), v[o].tar); tar(rs(o), v[o].tar);
            v[o].tar = 0;
        }
    }
    void build(int o, int L, int R){
        if(L == R) { v[o].MIN = 0; v[o].sMIN = 1; return; }
        int mid = (L + R) >> 1;
        ls(o) = make; rs(o) = make;
        build(ls(o), L, mid); build(rs(o), mid + 1, R);
        maintain(o);
    }
    void update(int o, int nowl, int nowr, int L, int R, int x){
        if(L > R) return ;
        if(L <= nowl && nowr <= R) return tar(o, x);
        int mid = (nowl + nowr) >> 1;  pushdown(o);
```

```
        if(L <= mid) update(ls(o), nowl, mid, L, R, x);
        if(R  > mid) update(rs(o), mid + 1, nowr, L, R, x);
        maintain(o);
    }
    Node query(int o, int nowl, int nowr, int L, int R) {
        if(L <= nowl && nowr <= R) return v[o];
        int mid = (nowl + nowr) >> 1; pushdown(o);
        Node Ans; Ans.MIN = INT_MAX;
        if(L <= mid) Ans = Ans + query(ls(o), nowl, mid, L, R);
        if(R  > mid) Ans = Ans + query(rs(o), mid + 1, nowr, L, R);
        return Ans;
    }
    int query(int o, int L, int R){
        Node res = query(o, 1, n, L, R);
        return res.MIN == 0 ? res.sMIN : 0;
    }
}  using SegmentTree::Groot; using SegmentTree::build; using SegmentTree::query; using SegmentTree::upd
namespace Mon_Stack{
    int t0t = 0, t1t = 0;
    pair<int, int> S0[_], S1[_];
    int work(int *A, int n){
        int Ans = 0;
        int root = Groot();
        build(root, 1, n);
        S0[0].fir = S1[0].fir = 0;
        for(int i = 1; i <= n; i++) {
            update(root, 1, n, 1, i - 1, -1);
            while(t0t != 0 && S0[t0t].sec >= A[i]) { update(root, 1, n, S0[t0t - 1].fir + 1, S0[t0t].fi
            while(t1t != 0 && S1[t1t].sec <= A[i]) { update(root, 1, n, S1[t1t - 1].fir + 1, S1[t1t].fi
            update(root, 1, n, S0[t0t].fir + 1, i, -A[i]);
            update(root, 1, n, S1[t1t].fir + 1, i, A[i]);
            int t = query(root, 1, i);
            Ans += t;
            S0[++t0t] = make_pair(i, A[i]); S1[++t1t] = make_pair(i, A[i]);
        }
        return Ans;
    }
} using Mon_Stack::work;
signed main(){
    n = read(); for(int i = 1; i <= n; i++) { int x = read(), y = read(); A[x] = y; }
    printf("%lld\n", work(A, n));
    return 0;
}
```

```
#include <climits>
#include <cmath>
#include <cstring>
```

```cpp
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
#define int long long
const int MOD = 1e9 + 7;
const int _M = 2e5 + 100;
inline int read() { char c = getchar(); int sign = 1; int x = 0; while(c > '9' || c < '0') { if(c=='-')
int MAX, m;
int have[_M], tht = 0;
int none[_M], tnt = 0;
int d[_M], tdt = 0;
int c[_M];//tct = tdt
bool CanBe[_M];
int ToL[_M];
int ToR[_M];
void divide(int s, int x) {
    for(int i = 1; i * i <= x; i++){
        if(x % i == 0) d[++tdt] = i, c[tdt] = s % i;
        if(i * i != x) d[++tdt] = x / i, c[tdt] = s % d[tdt];
    }
    for(int i = 1; i <= tdt; i++) CanBe[i] = true;
}
signed main(){
    int t;
    MAX = read(), m = read();
    for(int i = 1; i <= m; i++){
        int type = read();
        if(type == 1) have[++tht] = read();
        else          none[++tnt] = read();
    }
    sort(have + 1, have + 1 + tht); sort(none + 1, none + 1 + tnt);
    int maxd = INT_MAX, Sta;
    for(int i = 2; i <= tht; i++) {
        if(maxd > have[i] - have[i - 1]){
            maxd = have[i] - have[i - 1];
            Sta = have[i - 1];
        }
    }
    divide(Sta, maxd);

    /* 公差 及 特征 检查   */
    for(int i = 1; i <= tht; i++){
        for(int j = 1; j <= tdt; j++){
            if(!CanBe[j]) continue;
            if(have[i] % d[j] != c[j]) CanBe[j] = false;
        }
```

```
}
for(int i = 2; i <= tht; i++){
    int D = have[i] - have[i - 1];
    for(int j = 1; j <= tdt; j++){
        if(!CanBe[j]) continue;
        if(D % d[j] != 0) CanBe[j] = false;
    }
}
t = 0;
for(int i = 1; i <= tdt; i++){
    if(!CanBe[i]) continue;
    d[++t] = d[i];
    c[  t] = c[i];
    CanBe[t] = true;
} tdt = t;
/* 公差 及 特征 检查   完成 */
for(int i = 1; i <= tdt; i++) {
    if(CanBe[i]) {
        ToL[i] = (c[i] == 0 ? d[i] : c[i]) - 1;
        int t = MAX % d[i];
        if(t == c[i]) ToR[i] = MAX + 1;
        if(t  > c[i]) ToR[i] = MAX - (t - c[i]) + 1;
        if(t  < c[i]) ToR[i] = MAX - d[i] + (c[i] - t) + 1;
    }
}
t = 0;
for(int i = 1; i <= tdt; i++){
    if(!CanBe[i]) continue;
    d[++t] = d[i];
    c[  t] = c[i];
    CanBe[t] = true;
} tdt = t;
int MINS = have[1], MAXE = have[tht];
for(int i = 1; i <= tnt; i++){
    int now = none[i];
    for(int j = 1; j <= tdt; j++){
        if(!CanBe[j]) continue;
        if(now % d[j] == c[j]){
            if(now < MINS)
                ToL[j] = max(ToL[j], now);
            else if(now > MAXE)
                ToR[j] = min(ToR[j], now);
            else CanBe[j] = false;
        }
    }
}
for(int i = 1; i <= tdt; i++) if(ToL[i] > MINS || ToR[i] < MAXE) CanBe[i] = false; else ToL[i]++, T
```

```
    /* 非法数字存在性检查 */
    int ans = 0;
    for(int i = 1; i <= tdt; i++){
        if(!CanBe[i]) continue;
        int k = (  (MINS - ToL[i]) / d[i] + 1) * ( (ToR[i] - MAXE) / d[i] + 1) % MOD; // TODO;
        ans = (ans + k) % MOD;
    }
    printf("%lld", ans);
    return 0;
}

#include <climits>
#include <cstring>
#include <cmath>
#include <iostream>
#include <algorithm>
#include <vector>
inline int read() { char c = getchar(); int sign = 1; int x = 0; while(c > '9' || c < '0') { if(c=='-')
using namespace std;
const int _ = 6e5 + 100;
const int base = 15;
const int MOD = 1020031005;
int SS[_];
int pow(int a, int b = base){ int ans = 1; while(b){ if(b & 1) ans = (ans *1ll* a) % MOD; a = (a *1ll*

int head[_];
int NodeVal[_];
struct edges{
    int node;
    int nxt;
}edge[_];
int tot = 0;
void add(int u, int v){
    tot++;
    edge[tot].node = v;
    edge[tot].nxt  = head[u];
    head[u]        = tot;
}

int n, q;
int dep[_], fa[_], dfn[_], rnk[_], top[_], son[_], si[_], dfc = 0;
void dfs0(int now, int f, int dp){
    int &S = si[now] = 1; int &Mid = son[now] = 0; fa[now] = f; dep[now] = dp;
    for(int i = head[now]; i ; i = edge[i].nxt){
        int ex = edge[i].node; if(ex == f) continue;
        dfs0(ex, now, dp + 1); S += si[ex];
        if(si[ex] > si[Mid]) Mid = ex;
```

```
    } if(!Mid) Mid = 0;
}
void dfs1(int now, int f, int tp){
    dfn[now] = ++dfc; rnk[dfc] = now;
    top[now] = tp;
    if(son[now]) dfs1(son[now], now, tp);
    for(int i = head[now]; i ; i = edge[i].nxt){
        int ex = edge[i].node; if(ex == f || ex == son[now]) continue;
        dfs1(ex, now, ex);
    }
}
void clear(){
    memset(head, 0, sizeof(head));
    dfn[0] = rnk[0] = si[0] = top[0] = fa[0] = son[0] = 0;
    tot = 0;
    dfc = 0;
}
namespace SegmentTree{
    const int _ = 3e6 + 100;
    int tot = 0;
    int ch[_][2];
    int v[_];
    #define make (tot++, ch[tot][0] = ch[tot][1] = v[tot] = 0, tot);
    #define ls(x) (ch[x][0])
    #define rs(x) (ch[x][1])
    #define maintain(o) (v[o] = (v[ls(o)] +0ll+ v[rs(o)]) % MOD )
    void init_s() { tot = 0; }
    int  Groot(){ return make; }
    void build(int o, int L, int R){
        if(L == R) return (void)(v[o] = pow(NodeVal[rnk[L]]));
        int mid = (L + R) >> 1;
        ls(o) = make; rs(o) = make;
        build(ls(o), L, mid); build(rs(o), mid + 1, R);
        maintain(o);
    }
    void update(int o, int nowl, int nowr, int p, int x){
        if(nowl == nowr) return (void)(v[o] = pow(x));
        int mid = (nowl + nowr) >> 1;
        if(p <= mid) update(ls(o), nowl, mid, p, x);
        if(p  > mid) update(rs(o), mid + 1, nowr, p, x);
        maintain(o);
    }
    int query(int o, int nowl, int nowr, int L, int R){
        if(L > R) return 0;
        if(L <= nowl && nowr <= R) return v[o];
        int mid = (nowl + nowr) >> 1;
        int ans = 0;
```

```
        if(L <= mid) ans = (ans +0ll+ query(ls(o), nowl, mid, L, R)) % MOD;
        if(R  > mid) ans = (ans +0ll+ query(rs(o), mid + 1, nowr, L, R)) % MOD;
        return ans;
    }
} using SegmentTree::Groot; using SegmentTree::init_s; using SegmentTree::build; using SegmentTree::upd
int root = 0;
int LCA;
int QueryOnPath(int x, int y){
    int ans = 0;
    while(top[x] != top[y]){
        if(dep[top[x]] > dep[top[y]]) swap(x, y);
        ans = (ans +0ll+ query(root, 1, n, dfn[top[y]], dfn[y])) % MOD;
        y = fa[top[y]];
    }
    if(dep[x] < dep[y]) swap(x, y); LCA = y;
    ans = (ans +0ll+ query(root, 1, n, dfn[y], dfn[x])) % MOD;
    return ans;
}
void doit(){
    clear();
    n = read(), q = read();
    for(int i = 1; i <= n; i++) NodeVal[i] = read();
    for(int i = 1; i <  n; i++){ int u = read(), v = read(); add(u, v); add(v, u); }
    dfs0(1, 1, 1);
    dfs1(1, 1, 1);
    init_s(); root = Groot();
    build(root, 1, n);
    while(q--){
        int opt = read(), x = read(), y = read();
        if(opt == 1){
            int r = QueryOnPath(x, y);
            int Len = dep[x] + dep[y] - (dep[LCA] << 1) + 1;
            puts(SS[Len] == r ? "Yes" : "No");
        } else{
            update(root, 1, n, dfn[x], y);
        }
    }
}
int main(){
    for(int i = 1; i <= 3e5; i++) SS[i] = ( SS[i - 1] +0ll+ pow(i) ) % MOD;
    int T = read();
    while(T--) doit();
    return 0;
}
```

CF521E Cycling City

- 给定一张 n 个点 m 条边的无向简单图。
- 问图中能否找到两个点，满足这两个点之间有至少三条完全不相交的简单路径。
- $n, m \le 2 \times 10^5$，图不保证连通

考虑生成树，对原图做一个生成树，考虑每个在生成树外的边，覆盖在树上，树上边被覆盖两次的，即为一种方案

由于树的拓扑结构比图要简单很多，直接思考图的生成树解决图内问题是一个常见思路。

CF547D Mike and Fish

待填