Codeforces Round #728 (Div. 1)

比赛链接

B. Tree Array

题意

给定 n 个点的树，点编号分别为 $1\sim n$

初始时等概率随机选择一个点，接下来每次操作从未选择的且与已经选择的点相连的点中等概率随机选择一个点。

根据选择的顺序可以得到由点的编号构成的序列，问序列逆序对的期望个数。

题解

首先枚举初始选择点，并将初始选择点作为根节点。

不难发现，每个节点只有在所以祖先结点被选取后才有机会选取。

同时对任意点对 (u,v)当 $p=\text{LCA}(u,v)$ 被选取后，他们的其余祖先节点被选取的概率总是相等的。

于是不妨设 $u\gt v,d_1=d_u-d_p,d_2=d_v-d_p$于是 (u,v) 构成逆序对的概率等价于如下模型：

两个袋子中分别有 d_1,d_2 个球，每次有 t 概率从某个袋子中选一个球，也有 $1-2t$ 概率无事发生，问 d_2 个球的袋子先被取空的概率。

不难发现 $\text{dp}(d_1,d_2)=\cfrac {\text{dp}(d_1-1,d_2)+\text{dp}(d_1,d_2-1)}2$然后暴力统计固定根每个点对贡献即可，时间复杂度 $O(n^3)$

```
const int MAXN=205,mod=1e9+7;
int dp[MAXN][MAXN];
int quick_pow(int a,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*a%mod;
        a=1LL*a*a%mod;
        k>>=1;
    }
    return ans;
}
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
```

```
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
vector<int> ch[MAXN];
int dep[MAXN],ans;
void dfs(int u,int fa){
    dep[u]=dep[fa]+1;
    ch[u].clear();
    ch[u].push_back(u);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        dfs(v,u);
        _for(j,0,ch[u].size())
        _for(k,0,ch[v].size()){
            int x=ch[u][j],y=ch[v][k];
            int d1=dep[max(x,y)]-dep[u],d2=dep[min(x,y)]-dep[u];
            ans=(ans+dp[d1][d2])%mod;
        }
        _for(j,0,ch[v].size())
        ch[u].push_back(ch[v][j]);
    }
}
int main()
{
    int inv2=quick_pow(2,mod-2);
    _for(i,1,MAXN)dp[0][i]=1;
    _for(i,1,MAXN)_for(j,1,MAXN)
    dp[i][j]=1LL*(dp[i-1][j]+dp[i][j-1])*inv2%mod;
    int n=read_int();
    _for(i,1,n){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    _rep(i,1,n)
    dfs(i,0);
    enter(1LL*ans*quick_pow(n,mod-2)%mod);
    return 0;
}
```

C. Converging Array

题意

一开始有 $a[1\sim n],b[1\sim n-1]$□接下来有无限次操作。

每次操作随机选择一个 $1\le i\le n-1$ $a_i=\min\left(a_i,\cfrac {a_i+a_{i+1}-b_i}2\right),a_{i+1}=\max\left(a_i,\cfrac {a_i+a_{i+1}+b_i}2\right)$

可以证明无限次操作后序列 $a[1\sim n]$ 收敛于固定值。

每次询问给定 x 问有多少序列 $a[1\sim n]$ 满足：

1. $0\le a_i\le c_i$
2. 最终收敛后 $a_1\ge x$

题解

设最终得到序列为 $f[1\sim n]$ 观察发现，每次操作不改变 $\sum a_i$ 且最后有 $f_{i+1}-f_i\ge b_i$

设 $sa_i=\sum_{j=1}^i a_j,sb_1=0,sb_{i+1}=\sum_{k=1}^j b_k,ssb_i=\sum_{j=1}^i sb_i,sc_i=\sum_{j=1}^i c_j$

不难发现最后形式为 $f_1+sb_1,f_1+sb_2,f_1+sb_3 \cdots f_1+sb_k,f_{k+1}\cdots$ 其中 $f_t\gt f_1+sb_t(t\gt k)$

同时有 $\sum_{i=1}^k f_i+sb_i=\sum_{i=1}^k a_i$ 且 $\sum_{i=1}^t f_i+sb_i=\sum_{i=1}^t a_i\ge \sum_{i=1}^t a_i(t\neq k)$

于是有 $f_1=\min_{i=1}^n\left(\cfrac {sa_i-ssb_i}i\right)$ 枚举询问的 x 对每个 x 保证 $\cfrac {sa_i-ssb_i}i\ge x$ 即可计算答案个数。

设 $\text{dp}(i,s)$ 表示 $\sum_{j=1}^i a_i=s$ 的情况 $m=\max c_i$ 利用差分可以 $O(n^2m)$ 计算答案。

最后有 $\min\left(\cfrac {-ssb_i}i\right)\le f_1\le \min\left(\cfrac {sc_i-ssb_i}i\right)$ 于是有 $\max(f_1)-\min (f_1)\le m$

即有效的 x 只有 $O(m)$ 个，最终时间复杂度 $O\left(n^2m^2\right)$

```
const int MAXN=105,MAXV=1e4+5,mod=1e9+7,inf=1e9;
int b[MAXN],c[MAXN],dp[MAXN][MAXV],ans[MAXN],n;
int solve(int x){
    mem(dp,0);
    dp[0][0]=1;
    _rep(i,1,n){
        _for(j,1,MAXV)
        dp[i-1][j]=(dp[i-1][j]+dp[i-1][j-1])%mod;
        _for(j,max(x*i+b[i],0),MAXV){
            if(j-c[i]>0)
            dp[i][j]=(dp[i-1][j]-dp[i-1][j-c[i]-1]+mod)%mod;
            else
            dp[i][j]=dp[i-1][j];
        }
    }
    int ans=0;
    _for(i,0,MAXV)
```

```
    ans=(ans+dp[n][i])%mod;
    return ans;
}
int Div(int a,int b){
    if(a<0)
    return (a-b+1)/b;
    else
    return a/b;
}
int main()
{
    n=read_int();
    _rep(i,1,n)c[i]=read_int();
    _rep(i,2,n)b[i]=read_int()+b[i-1];
    _rep(i,2,n)b[i]=b[i]+b[i-1];
    int lef=inf,rig=inf,s=0;
    _rep(i,1,n){
        s+=c[i];
        lef=min(lef,Div(-b[i],i));
        rig=min(rig,Div(s-b[i],i));
    }
    _rep(i,lef,rig)
    ans[i-lef]=solve(i);
    int q=read_int();
    while(q--){
        int x=read_int();
        x=min(max(lef,x),rig+1);
        enter(ans[x-lef]);
    }
    return 0;
}
```

D. Inverse Inversions

题意

现有一个 $1\sim n$ 的排列，设 p_i 表示元素 i 在排列中的位置。已知序列 b其中 $b_i=\sum_{j=1}^i[p_j\gt p_i]$

接下来两种操作：

1. 修改某个 b_x
2. 询问某个 p_x

题解

设 $P(i,r)=\sum_{j=1}^r[p_j\gt p_i]$$c_i=i-1-b_i$于是有 $P(i,i)=c_i,P(i,n)=p_i-1$

不难发现 $P(i,r+1)=P(i,r)+[c_{r+1}\le P(i,r)]$。

对修改操作仅维护 c_i 时间复杂度 $O(1)$ 对询问操作直接暴力转移，时间复杂度 $O(n)$。

考虑分块，设分块大小为 B 对每个块 $[l_i,r_i]$ 考虑线段树维护 $P(x,l_i)(l_i\le x\le r_i)$。

对线段树的每个区间 $[L,R]$ 维护 $P(x,R)(L\le x\le R)$ 于是 push_up 操作可以双指针维护，时间复杂度 $O(R-L)$。

修改某个 b_x 对应线段树的单点修改操作，于是修改操作的总复杂度 $O(B)$。

对于查询操作，设 $x\in [l_i,r_i]$ 可以先 $O(B)$ 暴力计算出 $P(x,r_i)$。

然后对后续每个块 k 利用序列 $P(x,r_k)$ 可以二分计算贡献，时间复杂度 $O(\frac nB\log B)$。

不难发现取 $B=O(\sqrt{n\log n})$ 时复杂的最小，此时时间复杂度为 $O\left(q\sqrt{n\log n}\right)$ 空间复杂度 $O(n\log n)$。

```
const int MAXN=1e5+5,MAXB=600;
int b[MAXN];
struct Tree{
    int lef[MAXB<<2],rig[MAXB<<2];
    vector<int> p[MAXB<<2];
    void push_up(int k){
        int pos1=0,pos2=0,k1=k<<1,k2=k<<1|1;
        p[k].clear();
        _rep(i,lef[k],rig[k]){
            if(pos1<p[k1].size()&&pos2<p[k2].size()){
                if(p[k1][pos1]+pos2<p[k2][pos2])
                p[k].push_back(p[k1][pos1++]+pos2);
                else
                p[k].push_back(p[k2][pos2++]);
            }
            else if(pos1<p[k1].size())
            p[k].push_back(p[k1][pos1++]+pos2);
            else
            p[k].push_back(p[k2][pos2++]);
        }
    }
    void build(int k,int L,int R){
        lef[k]=L,rig[k]=R;
        int M=L+R>>1;
        if(L==R){
            p[k].push_back(b[M]);
            return;
        }
        build(k<<1,L,M);
        build(k<<1|1,M+1,R);
        push_up(k);
    }
    void update(int k,int pos){
```

```
        if(lef[k]==rig[k]){
            p[k][0]=b[pos];
            return;
        }
        int mid=lef[k]+rig[k]>>1;
        if(pos<=mid)
        update(k<<1,pos);
        else
        update(k<<1|1,pos);
        push_up(k);
    }
    int query(int v){
        int lef=1,rig=p[1].size(),ans=0;
        while(lef<=rig){
            int mid=lef+rig>>1;
            if(v+mid>p[1][mid-1]){
                ans=mid;
                lef=mid+1;
            }
            else
            rig=mid-1;
        }
        return ans;
    }
}tree[MAXB];
int blk_id[MAXN],lef[MAXB],rig[MAXB];
int main()
{
    int n=read_int(),blk_sz=sqrt(n*log(n))/2+1,m=n/blk_sz;
    if(n%blk_sz)m++;
    _rep(i,1,n)b[i]=i-1-read_int();
    _for(i,0,m){
        lef[i]=i*blk_sz+1;
        rig[i]=min((i+1)*blk_sz,n);
        _rep(j,lef[i],rig[i])
        blk_id[j]=i;
        tree[i].build(1,lef[i],rig[i]);
    }
    int q=read_int();
    while(q--){
        int opt=read_int(),x=read_int();
        if(opt==1){
            b[x]=x-1-read_int();
            tree[blk_id[x]].update(1,x);
        }
        else{
            int ans=b[x];
            _rep(i,x+1,rig[blk_id[x]]){
                if(ans>=b[i])
                ans++;
```

```
            }
            _for(i,blk_id[x]+1,m)
            ans+=tree[i].query(ans);
            enter(ans+1);
        }
    }
    return 0;
}
```

另外附上一份时间复杂度 $O\left(q\sqrt n\log n\right)$空间复杂度 $O(n)$ 的代码，树状数组上二分写的。

```
const int MAXN=1e5+5;
#define lowbit(x) ((x)&(-x))
int c[MAXN];
void add(int pos,int v){
    while(pos<MAXN){
        c[pos]+=v;
        pos+=lowbit(pos);
    }
}
int query(int b){
    int pos=0,curb=0;
    for(int i=16;i>=0;i--){
        if(pos+(1<<i)<MAXN&&c[pos+(1<<i)]+(1<<i)+curb<b){
            curb+=c[pos+(1<<i)]+(1<<i);
            pos+=(1<<i);
        }
    }
    return pos+1;
}
int b[MAXN],lef[MAXN],rig[MAXN],blk_id[MAXN];
vector<int> a[MAXN];
void build(int k){
    a[k].clear();
    _rep(i,lef[k],rig[k]){
        int t=query(b[i]+1)-1;
        a[k].push_back(t);
        add(t+1,1);
    }
    sort(a[k].begin(),a[k].end());
    _for(i,0,a[k].size())
    add(a[k][i]+1,-1);
}
int main()
{
    int n=read_int(),blk_sz=sqrt(n/2)+1,m=n/blk_sz;
    if(n%blk_sz)m++;
    _rep(i,1,n)b[i]=read_int();
    _for(i,0,m){
        lef[i]=i*blk_sz+1;
```

```
        rig[i]=min((i+1)*blk_sz,n);
        _rep(j,lef[i],rig[i])
        blk_id[j]=i;
        build(i);
    }
    int q=read_int();
    while(q--){
        int opt=read_int(),x=read_int();
        if(opt==1){
            b[x]=read_int();
            build(blk_id[x]);
        }
        else{
            int ans=b[x];
            _rep(i,x+1,rig[blk_id[x]]){
                if(ans>=b[i])
                ans++;
            }
            _for(i,blk_id[x]+1,m)
            ans+=upper_bound(a[i].begin(),a[i].end(),ans)-a[i].begin();
            enter(n-ans);
        }
    }
    return 0;
}
```