U458258 平均数之和

长度 $len=1$ 的区间和：

$$res[1]=a[1]+a[2]+...+a[n]$$

长度 $len=2$ 的区间和：

$$res[2]=a[1]+2(a[2]+...+a[n-1])+a[n]=s1+(a[2]+...+a[n-1])$$

长度 $len=3$ 的区间和：

$$\$res[3]=s2+(a[3]+...+a[n-2])\$$$

也就是 $res[i]$ 相对于 $res[i-1]$ 增加或者减少一个区间和，可以利用前缀和递推求出 $res[1\sim n]$。

```
void work(){
    ll ans = 0, res = 0;
    for(int len=1;len<=n;++len){
        res = (res + s[n-len+1] - s[len-1] + mod) % mod;
        ans = (ans + res * qpow(len, mod-2)) % mod;
    }
    cout<<ans<<'\n';
}
```

U232856 矩形加矩形求和（离线）

算法1 $h=1$

对于一维的情况，可以把修改和询问放一起离散化之后求前缀和解决。

复杂度 $O(qlogq)$。

算法2 $h,w\le 500$

使用二维差分+二维前缀和，不用离散化，复杂度 $O(hw+q)$。

算法3 $h\le 500$

对于 $h\le 500$ 的数据，离散化后维护每行的前缀和，询问时 `lower_bound()` 查询（相当于 $h=1$ 的情况做500次）。如果把询问也离散化的化，可能空间不太够。复杂度 $O((h+logq)q)$。

算法4

离散化之后按 x 升序从左->右做扫描线，一个查询 (qx,qy_1,qy_2) 的答案为之前所有 (x,y,w) 的总贡献：

$$\sum_{qy_1\le y\le qy_2}(qx-x)w=qx\sum_{qy_1\le y\le qy_2}w-\sum_{qy_1\le y\le qy_2}x\times w$$

也就是需要在 y 这个维度上支持 w 和 $x\times w$ 的区间修改、区间求和，使用线段树或者树状数组支持。

注意如果修改/查询的是差分或者前缀和数组，边界要相应 $+/-1$。

总复杂度 $O(qlogq)$。

```
#include<bits/stdc++.h>
#define MAXN 800005
#define mod (2021)
using namespace std;
typedef long long ll;

void add(int& a, int b){
    a += b;
    if(a >= mod) a -= mod;
}

int n,m,q1,q2,Nx,Ny;
int cx[MAXN],cy[MAXN];

struct req{
    int r1,r2,c1,c2,x;
} p1[MAXN], p2[MAXN];

struct seg{
    int y,v;
};

int ans[MAXN];
vector<seg> adj[4][MAXN];

int fw[4][MAXN];
int lbt(int x) {return x & (-x);}
void change(int f, int x, int v){
    for(;x<=Ny;x+=lbt(x)){
        add(fw[f][x], v);
    }
}

int getsum(int f, int x){
    int ans = 0;
    for(;x;x-=lbt(x)){
        add(ans, fw[f][x]);
    }
    return ans;
}

void add(int x, int y, int v){
    change(0, y, v);
    change(1, y, (ll)cy[y]*v%mod);
    change(2, y, (ll)cx[x]*v%mod);
    change(3, y, (ll)cx[x]*cy[y]%mod*v%mod);
}

int query(int qx, int qy){
    int ans = 0;
    add(ans, ((ll)cx[qx]*cy[qy]%mod + cx[qx] + cy[qy] + 1) % mod * getsum(0, qy) % mod);
    add(ans, mod - ((ll)cx[qx] + 1) * getsum(1, qy) % mod);
    add(ans, mod - ((ll)cy[qy] + 1) * getsum(2, qy) % mod);
    add(ans, getsum(3, qy));
```

```
    return ans;
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin>>n>>m>>q1>>q2;

    int r1,r2,c1,c2,x;
    for(int k=1;k<=q1;k++){
        cin>>r1>>r2>>c1>>c2>>x;
        p1[k] = {r1,r2,c1,c2,x};
        cx[++Nx] = r1; cx[++Nx] = r2+1;
        cy[++Ny] = c1; cy[++Ny] = c2+1;
    }

    for(int k=1;k<=q2;k++){
        cin>>r1>>r2>>c1>>c2;
        p2[k] = {r1,r2,c1,c2};
        cx[++Nx] = r1-1; cx[++Nx] = r2;
        cy[++Ny] = c1-1; cy[++Ny] = c2;
    }

    sort(cx+1, cx+1+Nx);
    sort(cy+1, cy+1+Ny);
    Nx = unique(cx+1, cx+1+Nx) - cx - 1;
    Ny = unique(cy+1, cy+1+Ny) - cy - 1;

    //
    for(int k=1;k<=q1;k++){
        r1 = lower_bound(cx+1, cx+1+Nx, p1[k].r1) - cx;
        r2 = lower_bound(cx+1, cx+1+Nx, p1[k].r2+1) - cx;
        c1 = lower_bound(cy+1, cy+1+Ny, p1[k].c1) - cy;
        c2 = lower_bound(cy+1, cy+1+Ny, p1[k].c2+1) - cy;

        adj[0][r1].push_back({c1,p1[k].x});
        adj[0][r1].push_back({c2,mod-p1[k].x});
        adj[0][r2].push_back({c1,mod-p1[k].x});
        adj[0][r2].push_back({c2,p1[k].x});
    }

    for(int k=1;k<=q2;k++){
        r1 = lower_bound(cx+1, cx+1+Nx, p2[k].r1-1) - cx;
        r2 = lower_bound(cx+1, cx+1+Nx, p2[k].r2) - cx;
        c1 = lower_bound(cy+1, cy+1+Ny, p2[k].c1-1) - cy;
        c2 = lower_bound(cy+1, cy+1+Ny, p2[k].c2) - cy;

        adj[3][r1].push_back({c1,k});
        adj[3][r2].push_back({c2,k});
        adj[2][r1].push_back({c2,k});
        adj[2][r2].push_back({c1,k});
    }

    for(int x=1;x<=Nx;++x){
        for(seg s:adj[0][x]) add(x,s.y,s.v);
        for(seg s:adj[2][x]) ans[s.v] = (ans[s.v] - query(x,s.y) + mod)
% mod;
        for(seg s:adj[3][x]) ans[s.v] = (ans[s.v] + query(x,s.y)) % mod;
```

```
    }
    for(int i=1;i<=q2;++i) cout<<ans[i]<<'\n';
    return 0;
}
```

U111091 区间2段覆盖

先看 $K=1$ 的情况：

观察1：

- 存在最优区间以某个村庄作为终点

那么做法就是枚举终点，维护一个指针指向起点，计算答案即可，复杂度$O(N)$。

当 $K=2$ 时，需要考虑两种情况：

1. 第一个区间终点和第二个区间起点在同一条道路内
2. 否则，存在一个分界村庄 i ，2条道路分别在 i 之前和 i 之后

观察2：

- 若2个选定区间的端点在同一条道路内，则存在最优方案使得它们首尾相连

调整法可以证明，所以case1，就当作 $K=1$做；

对于case2，分别dp之后维护前缀/后缀max，枚举分界点 i 即可。

总复杂度线性

```
#include<bits/stdc++.h>
#define MAXN 200005
#define P int(1e9+7)
using namespace std;
typedef long long ll;

int T,N,K,q;

int x[MAXN], tt = 0;
int g[MAXN], p[MAXN], f[MAXN];
char s[MAXN];

int solve1(int q){
    int k = 1, ans = 0, ans1;
    for(int i=2;i<=N;i++){
        while(x[i]-x[k]>q) ++k;
        ans1 = g[i] - g[k];
        if(s[k]=='1') ans1 += q - (x[i]-x[k]);
        ans = max(ans, ans1);
    }
    //cerr<<"ans1 = "<<ans1<<endl;
    return ans;
}

int solve2(int q){
    memset(p, 0, sizeof(p));
    memset(f, 0, sizeof(f));
```

```
    int ans1;
    int k = 1;
    for(int i=2;i<=N;i++){
        while(x[i]-x[k]>q) ++k;
        ans1 = g[i] - g[k];
        if(s[k]=='1') ans1 += q - (x[i]-x[k]);
        p[i] = max(p[i-1], ans1);
    }

    k = N;
    for(int i=N-1;i>=1;i--){
        while(x[k]-x[i]>q) --k;
        ans1 = g[k] - g[i];
        if(s[k+1]=='1') ans1 += q - (x[k]-x[i]);
        f[i] = max(f[i+1], ans1);
    }

    int ans = 0;
    for(int i=2;i<=N;i++){
        ans = max(ans, p[i] + f[i]);
    }
    return ans;
}

int main(){
    cin>>T;
    while(T--){
        cin>>N>>K>>q;
        for(int i=1;i<=N;i++) cin>>x[i];
        cin>>s+2;
        s[1] = '0';
        tt = 0;
        for(int i=2;i<=N;i++){
            g[i] = g[i-1];
            if(s[i]=='1'){
                g[i] += x[i] - x[i-1];
                tt += x[i] - x[i-1];
            }
        }
        int ans;
        if(K==1) ans = tt - solve1(q);
        else ans = tt - max(solve1(2*q), solve2(q));
        cout<<ans<<'\n';
    }
    return 0;
}
```

U232520 黑色矩形

https://www.luogu.com.cn/problem/U232520

参考题解：https://www.luogu.com.cn/blog/ChenXingLing/solution-p6551