P6749 『MdOI R3』Yoshino 题解

返回题目 提交题解

题解仅供学习参考使用

抄袭、复制题解，以达到刷 AC 率/AC 数量或其他目的的行为，在洛谷是严格禁止的。

洛谷非常重视**学术诚信**。此类行为将会导致您成为**作弊者**。 具体细则请查看洛谷社区规则。

提交题解前请务必阅读《洛谷主题库题解规范》。

3 篇题解

Limit 创建时间：2020-08-19 10:30:07

先膜拜一发 BF.

题目大意

给出一个序列,每次可以将一段区间**赋值**为一段公差为 1 的等差数列,或者查询这个序列的逆序对个数.

分析

请先通过 动态逆序对,并且食用 P4062 的这篇题解.

如果您已经认真做了上面两件事,那么多少会对这题有点思路了.

(下面提到的 等差数列 如果没有特殊说明就是指公差为 1 的等差数列)

先考虑一次覆盖会产生的逆序对的个数:

首先可以知道在这样一个等差数列内部是不会有逆序对的.

那么产生的逆序对的个数就等于在等差数列前面的所有数大于等差数列中的数的个数的总和.后面同理.

动态计算这个东西自然就会想到用树套树去维护.

不过这里不是一个数,而是一个等差数列,那么就可以用到这里所提到的做法了,对于内层的权值线段树上不只要维护区间和($\sum_{i=l}^{r} sum_i$)还需要维
内的每个数($\sum_{i=l}^{r}[(i-l+1)sum_i]$),这样就可以通过一些简单的计算得到一个添加一个等差数列所得的贡献了.

接下来考虑如何删掉一次覆盖.

可以发现这题其实只有区间覆盖这一个操作(逆序对个数是时时计算的),所以 ODT 在这题中的复杂度是正确的.一次覆盖最多只会增加 3 个区间
$n+3m$ 显然是可以的.所以每次只需要在 ODT 上把覆盖到的所有区间的贡献在树套树上减掉,再把这次覆盖的贡献加上就可以了.

这样就可以直接用树状数组套权值线段树以及一个 ODT 来维护了.(其实和 BF 的做法没啥区别)

不过,树状数组套线段树见多了也就没啥意思了,所以来分享一下线段树套线段树的做法.

对于外层的线段树上是区间覆盖的标记,考虑标记永久化,对于删掉的覆盖需要将这些标记也删掉,那么如果存在一个节点的上面是有覆盖标记的
和子孙中是不可能有存在标记的节点,因为标记不能下传的原因,所以如果需要查询的节点时某个被覆盖的节点的子孙,那么需要直接在这个节点
算两个等差数列相连之后的逆序对个数,这个东西可以通过简单的画图和计算得出,代码如下:

```
int Calc(int a,int lena,int b,int lenb)//第一个等差数列的首项为 a,长度为 lena,第二个等差数列的首项为 b,长度为 lenb
{
    if(a<=b)//如果 a 的首项小于 b,那么就把 a 中没有贡献的部分去掉,是的保证 a 的首项大于 b
    {
        lena-=(b-a+1);
        a=b+1;
    }
    if(lena<=0)//如果减掉之后第一个等差数列不存在了,那么就删掉
    {
        return 0;
    }
    //一下内容建议自行画图理解
    int f=min(a-b,lenb);
    int l=min(a-b+lena-1,lenb);
    int len=l-f+1;
    return (f+l)*len/2+(lena-len)*lenb;
}
```

写出来常数可能有点大,需要一些卡常技巧:

对于内层的线段树需要用标记永久化,不然每次下传标记的时候需要重新开节点,会导致 MLE,对于外层的线段树修改时对于被完全覆盖的区间
上修改了,可以直接使用上面的这个 `Calc` 函数.

代码细节挺多的.

代码

```
#include<bits/stdc++.h>
#define REP(i,first,last) for(int i=first;i<=last;++i)
#define DOW(i,first,last) for(int i=first;i>=last;--i)
namespace IO
//快读模板
using namespace IO;
using namespace std;
const int TOP=3e4+2;
const int MAXN=3e4+5;
int n,m;
int arr[MAXN];
long long answer=0;
namespace InSGT//内层线段树与 P4062 基本相同
{
    struct LazyTag
    {
        int add;
        inline void Clean()
        {
            add=0;
        }
    }for_make;
    inline LazyTag MakeTag(int add)
    {
        for_make.add=add;
        return for_make;
    }
    struct SegmentTree
    {
        int len,lson,rson,sum;
        long long sum_;
        LazyTag tag;
    }sgt[MAXN*1000];
    int sgt_cnt=0;
    #define LSON sgt[now].lson
    #define RSON sgt[now].rson
    #define MIDDLE ((left+right)>>1)
    #define LEFT LSON,left,MIDDLE
    #define RIGHT RSON,MIDDLE+1,right
    #define NOW now_left,now_right
    inline void PushUp(int now)
    {
        //因为是标记永久化,所以需要在 PushUp 的时候把当前节点的标记的贡献也加上
        sgt[now].sum=sgt[LSON].sum+sgt[RSON].sum+sgt[now].tag.add*sgt[now].len;
        sgt[now].sum_=sgt[RSON].sum_+111*sgt[RSON].sum*(sgt[now].len-sgt[RSON].len)+sgt[LSON].sum_+(sgt[now].len+1)*sgt[now].len/2*sgt[now
    }
    inline void Down(LazyTag tag,int &now,int left,int right)
    {
        sgt[now].tag.add+=tag.add;
        sgt[now].sum+=sgt[now].len*tag.add;
        sgt[now].sum_+=(sgt[now].len+1)*sgt[now].len/2*tag.add;
    }
    void UpdataAdd(int now_left,int now_right,int add,int &now,int left=1,int right=TOP)
    {
        if(now_right<left||right<now_left)
        {
            return;
        }
        if(!now)
        {
            sgt[now=++sgt_cnt].len=right-left+1;
        }
        if(now_left<=left&&right<=now_right)
        {
            Down(MakeTag(add),now,left,right);
            return;
        }
        UpdataAdd(NOW,add,LEFT);
        UpdataAdd(NOW,add,RIGHT);
        PushUp(now);
    }
```

```
    int QuerySum(int now_left,int now_right,int now,int left=1,int right=TOP,int tag=0)//在查询的时候需要记录从根节点到当前节点一路上的标i
    {
        if(now_right<left||right<now_left)
        {
            return 0;
        }
        if(now_left<=left&&right<=now_right)
        {
            return sgt[now].sum+tag*(right-left+1);
        }
        return QuerySum(NOW,LEFT,tag+sgt[now].tag.add)+QuerySum(NOW,RIGHT,tag+sgt[now].tag.add);
    }
    int QuerySum_(int now_left,int now_right,int now,int left=1,int right=TOP,int tag=0)
    {
        if(now_right<left||right<now_left)
        {
            return 0;
        }
        if(now_left<=left&&right<=now_right)
        {
            int len=right-left+1;
            return sgt[now].sum_+(sgt[now].sum+tag*len)*(left-now_left)+(len+1)*len/2*tag;
        }
        return QuerySum_(NOW,LEFT,tag+sgt[now].tag.add)+QuerySum_(NOW,RIGHT,tag+sgt[now].tag.add);
    }
    #undef LSON
    #undef RSON
    #undef MIDDLE
    #undef LEFT
    #undef RIGHT
    #undef NOW
}
namespace OutSGT//外层线段树和 ODT
{
    int Calc(int a,int lena,int b,int lenb)
    {
        if(a<=b)
        {
            lena-=(b-a+1);
            a=b+1;
        }
        if(lena<=0)
        {
            return 0;
        }
        int f=min(a-b,lenb);
        int l=min(a-b+lena-1,lenb);
        int len=l-f+1;
        return (f+1)*len/2+(lena-len)*lenb;
    }
    struct SegmentTree
    {
        int root,tag;
    }sgt[MAXN*4];
    #define LSON (now<<1)
    #define RSON (now<<1|1)
    #define MIDDLE ((left+right)>>1)
    #define LEFT LSON,left,MIDDLE
    #define RIGHT RSON,MIDDLE+1,right
    #define NOW now_left,now_right
    void Build(int now=1,int left=1,int right=n)
    {
        if(left==right)//对于叶节点相当于是覆盖,所以只需要打上标记
        {
            sgt[now].tag=arr[left];
            return;
        }
        REP(i,left,right)
        {
            InSGT::UpdataAdd(arr[i],arr[i],1,sgt[now].root);
        }
        Build(LEFT);
        Build(RIGHT);
    }
    void Cover(int now_left,int now_right,int val,int add,int now=1,int left=1,int right=n)
```

```
{
    if(now_right<left||right<now_left)
    {
        return;
    }
    if(now_left<=left&&right<=now_right)//对于被完全覆盖的区间只需要打上标记
    {
        sgt[now].tag+=add*(val+left-now_left);
        return;
    }
    InSGT::UpdataAdd(val+max(now_left,left)-now_left,val+min(now_right,right)-now_left,add,sgt[now].root);//在内存线段树上修改
    Cover(NOW,val,add,LEFT);
    Cover(NOW,val,add,RIGHT);
}
int QueryL(int now_left,int now_right,int val,int now=1,int left=1,int right=n)//查询前面大于的个数
{
    if(now_left<=left)
    {
        return 0;
    }
    if(sgt[now].tag)
    {
        return Calc(sgt[now].tag,min(right,now_left-1)-left+1,val,now_right-now_left+1);
    }
    if(right<now_left)
    {
        return InSGT::QuerySum_(val+1,val+now_right-now_left,sgt[now].root)+InSGT::QuerySum(val+now_right-now_left+1,TOP,sgt[now].root
    }
    return QueryL(NOW,val,LEFT)+QueryL(NOW,val,RIGHT);
}
int QueryR(int now_left,int now_right,int val,int now=1,int left=1,int right=n)//查询后面小于自己的个数
{
    if(right<=now_right)
    {
        return 0;
    }
    if(sgt[now].tag)
    {
        return Calc(val,now_right-now_left+1,sgt[now].tag+max(left,now_right+1)-left,right-max(left,now_right+1)+1);
    }
    if(now_right<left)
    {
        return InSGT::QuerySum(1,val+now_right-now_left,sgt[now].root)*(now_right-now_left+1)-InSGT::QuerySum_(val,val+now_right-now_l
    }
    return QueryR(NOW,val,LEFT)+QueryR(NOW,val,RIGHT);
}
#undef LSON
#undef RSON
#undef MIDDLE
#undef LEFT
#undef RIGHT
#undef NOW
struct Node//ODT 部分
{
    int l,r;
    mutable int val;
    Node(int left=0,int right=-1,int v=0)
    {
        l=left;
        r=right;
        val=v;
    }
    bool operator <(const Node &b)const
    {
        return l<b.l;
    }
};
set<Node>s;
#define IT set<Node>::iterator
IT Split(int pos)//分裂区间,ODT 基本操作
{
    IT place=s.lower_bound(Node(pos));
    if(place!=s.end()&&place->l==pos)
    {
        return place;
```

```
            return place;
        }
        --place;
        int left=place->l,right=place->r;
        int val=place->val;
        Cover(left,right,val,-1);
        s.erase(place);
        Cover(left,pos-1,val,1);
        Cover(pos,right,val+pos-left,1);
        s.insert(Node(left,pos-1,val));
        return s.insert(Node(pos,right,val+pos-left)).first;
    }
    void Assign(int left,int right,int val)//区间覆盖部分
    {
        IT place_left=Split(left),place_right=Split(right+1);
        while(1)
        {
            IT i=s.lower_bound(Node(left));
            if(i->l==right+1)
            {
                break;
            }
            answer-=QueryL(i->l,i->r,i->val);
            answer-=QueryR(i->l,i->r,i->val);
            Cover(i->l,i->r,i->val,-1);
            s.erase(i);
        }
        Cover(left,right,val,1);
        answer+=QueryL(left,right,val);
        answer+=QueryR(left,right,val);
        s.insert(Node(left,right,val));
    }
    #undef IT
}
namespace ReversePair//最开始时直接树状数组计算逆序对
{
    int tree[MAXN];
    int LowBit(int now)
    {
        return now&-now;
    }
    void Add(int num)
    {
        for(int now=num;now<=TOP;now+=LowBit(now))
        {
            tree[now]++;
        }
    }
    int Query(int num)
    {
        int result=0;
        for(int now=num;now;now-=LowBit(now))
        {
            result+=tree[now];
        }
        return result;
    }
    void Calc()
    {
        REP(i,1,n)
        {
            Add(arr[i]);
            answer+=i-Query(arr[i]);
        }
    }
}
int main()
{
    Read(n,m);
    REP(i,1,n)
    {
        Read(arr[i]);
        OutSGT::s.insert(OutSGT::Node(i,i,arr[i]));
    }
    OutSGT::s.insert(OutSGT::Node(n+1,n+1,0));
```

```
    OutSGT::Build();
    ReversePair::Calc();
    int opt,l,r,val;
    REP(i,1,m)
    {
        Read(opt);
        if(opt==1)
        {
            Read(l,r,val);
            OutSGT::Assign(l,r,val);
        }
        if(opt==2)
        {
            Writeln(answer);
        }
    }
    return 0;
}
```


BFqwq Yoshino 创建时间：2020-08-12 00:10:52

放一下本题的官方题解吧。

其实这个题写正解代码难度相当的高。为了防止空间不够用的现象，我把本题空间改成了 500MB（不过由于我使用了一些优化，所以我的空

对于有一些暴力能够跑过本题的情况，其实出题人根本就没有想到有这样的暴力，不过还是在这里向大家道歉。

以下是本题的官方题解。

Subtask 1

考虑暴力修改，暴力查询逆序对，复杂度 $\mathrm{O}(n^3)$。

Subtask 2

将逆序对改用归并排序或权值树查询，复杂度 $\mathrm{O}(n^2 \log n)$。

Subtask 3

由于修改长度是 1，直接考虑使用树套树维护动态逆序对。在修改时查询 $[1,l-1]$ 区间大于原数的和 $[r+1,n]$ 区间小于原数的数的个数类似的查询并加上。在查询逆序对时，可直接输出。复杂度 $\mathrm{O}(n \log^2 n)$

Subtask 4

显而易见，在值域缩小的同时，修改区间也随之缩小。

于是我们可以考虑对每个值建一个树状数组，如果对应位置有这个值，则树状数组该位置的值就是 1，否则是 0。

在修改时，对 $[1,l-1]$ 区间的每个值分别做一个查询并做一个后缀和，对 $[r+1,n]$ 区间的每个值也做一个查询并做一个前缀和，然后对 $\mathrm{O}(1)$ 的复杂度修改并维护了。

注意不要忘了清楚修改区间内部对逆序对的贡献，这部分可以暴力。

由于值域可以认为是 $\mathrm{O}(\log n)$，故此处的复杂度为 $\mathrm{O}(n \log^2 n)$。

Subtask 5

由于第两次操作一就会重置一次，我们可以考虑直接用数学方法计算出修改的贡献。

对于奇数次操作，直接重置，逆序对清零。

对于偶数次操作，分为 $l<x$ 和 $l>x$ 两种。

如果 $l<x$，则我们修改完后的该区间的值为 $[x,x+r-l]$，易知 $[l,r]$ 与 $[x,x+r-l]$ 的交集为 $[x,l-1]$。

则对于值在 $[x,l-1]$ 区间的数，对逆序对的贡献显然是一个公差为 -1 的等差数列，我们可以直接对此进行求和。

而对于值在 $[l,x-r+l]$ 区间的数，显然对逆序对没有贡献，直接忽略。

如果 $l > r$ 则类似讨论。

于是，我们就可以 $\mathrm{O}(1)$ 完成对逆序对个数的计数（注意此时我们并不需要直接修改这个数列），复杂度 $\mathrm{O}(n)$。

Subtask 6

考虑颜色段均摊。

这是一种类似于珂朵莉树的 trick（事实上这就是珂朵莉树的推平操作），复杂度为均摊 $\mathrm{O}(n \times k)$，其中 k 为单次修改的复杂度。

对于一个区间染色的操作，我们可以直接将一个区间维护为一个点，并且将他装入在 set 或其他容器中。

在修改时，我们在 set 中找到我们需要清除的所有区间（如果边界被包含则将边界所在的区间断开变成两个区间），并**暴力**将它们逐个清除

为什么这样做的复杂度是正确的呢？我们可以来证明一下。

最初的时候，每个数单独构成一个区间，所以一共有 n 个区间。

而每次操作的时候，我们增加的区间包括断开左右边界区间时增加的两个区间，以及插入的新区间。

这样一来，整个操作过程中区间的总个数就是 $n + 3m$ 个。

而显然一个区间最多被插入一次，删除一次，且 nm 同级，所以复杂度为均摊 $\mathrm{O}(n \times k)$。

本题中的操作一并不是颜色段均摊，但显然可以用类似的方法，均摊 $\mathrm{O}(n \times k)$ 完成修改。

但我们在修改的同时，还需要再维护一个动态逆序对，于是考虑令 $k = \log^2 n$，使用树套树修改

考虑插入一个新段对逆序对的贡献（删除就将贡献减去）。

假设这个区间是 $[l, r]$，这个区间的值的范围是 $[x, y](r - l = y - x)$。

那么我们大致可以其他的数分为以下几类：

1. 位于 $[1, l - 1]$ 区间且值小于 x 的数。这类数没有贡献，直接无视。
2. 位于 $[r + 1, n]$ 区间且值大于 y 的数。这类数同样没有贡献，直接无视。
3. 位于 $[1, l - 1]$ 区间且值大于 y 的数。这类数对区间内的所有数都有贡献，只需统计个数并乘以 $r - l + 1$。
4. 位于 $[r + 1, n]$ 区间且值小于 x 的数。这类数同样对区间内的所有数都有贡献，处理方法同上。
5. 位于 $[1, l - 1]$ 区间且值位于 $[x, y]$ 之间的数。注意到，对于值为 $t(t \in [x, y])$ 的数，它对这个区间的贡献为 $t - x$，因为这个区间内有 $t - x$。于是我们可以对**权值**维护区间内的个数以及区间和，也就是一个范围内值在某个区间的数有几个，以及这几个数加起来是多少。x 就是我们所要的结果。
6. 位于 $[r + 1, n]$ 区间且值位于 $[x, y]$ 之间的数。处理方式类似于第五类数。

这样一来，我们就在修改的同时完成了对逆序对的维护，而复杂度没有发生变化。

于是最终的复杂度就是 $\mathrm{O}(n \log^2 n)$。

总结

本题的难度并不高，没有涉及到很复杂的维护方法，也没有涉及到很高级的数据结构，主要还是对一些基本技巧的灵活应用。另外，有一个点家掌握。

另外，本题可能出现空间不够用的情况。在本题的 std 代码中，树套树中的内层线段树使用的是标记永久化的线段树，这也大大节省了本题的过下放的空间，但看到部分其他用户的提交记录，感觉下放的确空间大了好多。

贴一份代码吧，供大家参考。

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
inline int read(){
    register int x=0;
    register bool f=0;
    register char c=getchar();
    while(c<'0'||c>'9'){
        if(c=='-') f=1;
        c=getchar();
    }
    while(c>='0'&&c<='9'){
        x=(x<<3)+(x<<1)+c-48;
        c=getchar();
```

```
    }
    return f?-x:x;
}
char cr[200];int ttt;
inline void print(register int x,register char k='\n') {
    if(!x) putchar('0');
    while(x) cr[++ttt]=x%10+'0',x/=10;
    while(ttt) putchar(cr[ttt--]);
    putchar(k);
}
inline void printl(register ll x,register char k='\n') {
    if(!x) putchar('0');
    while(x) cr[++ttt]=x%10+'0',x/=10;
    while(ttt) putchar(cr[ttt--]);
    putchar(k);
}
const int maxn=52233;
const int len=30000;
struct seg{
    int v,ls,rs,tag;
    ll sum;
}t[maxn*160];
struct ask{
    int v;
    ll sum;
    friend ask operator +(ask a,ask b){
        return (ask){a.v+b.v,a.sum+b.sum};
    }
    friend ask operator -(ask a,ask b){
        return (ask){a.v-b.v,a.sum-b.sum};
    }
};
int rt[maxn],m,n,tot,st[maxn*128],top,tem[maxn][2],cnt[2];
int nnd(){
    if(top) return st[top--];
    return ++tot;
}
int del(int o){
    if(t[o].tag||t[o].sum||t[o].v||t[o].ls||t[o].rs) return o;
    st[++top]=o;
    return 0;
}
inline void change(int &o,register int l,register int r,register int ql,register int qr,register int v){
    if(!o) o=nnd();
    if(ql<=l&&r<=qr){
        t[o].v+=v*(r-l+1);
        t[o].sum+=v*(l+r)*(r-l+1)/2;
        t[o].tag+=v;
        o=del(o);
        return;
    }
    int mid=l+r>>1;
    if(ql<=mid) change(t[o].ls,l,mid,ql,qr,v);
    if(qr>mid) change(t[o].rs,mid+1,r,ql,qr,v);
    t[o].v=t[t[o].ls].v+t[t[o].rs].v+t[o].tag*(r-l+1);
    t[o].sum=t[t[o].ls].sum+t[t[o].rs].sum+(ll)t[o].tag*(l+r)*(r-l+1)/2;
    o=del(o);
}
inline void add(register int x,register int ql,register int qr,register int v){
    for(register int i=x;i<=n;i+=(i&-i))
        change(rt[i],1,len,ql,qr,v);
}
inline ask query(register int o,register int l,register int r,register int ql,register int qr,register ll tag){
    if(ql<=l&&r<=qr){
        ask res=(ask){0,0};
        res.v=t[o].v+tag*(r-l+1);
        res.sum=t[o].sum+tag*(r+l)*(r-l+1)/2;
        return res;
    }
    int mid=l+r>>1;ask res=(ask){0,0};
    if(ql<=mid) res=res+query(t[o].ls,l,mid,ql,qr,tag+t[o].tag);
    if(qr>mid) res=res+query(t[o].rs,mid+1,r,ql,qr,tag+t[o].tag);
    return res;
}
ask find(register int l,register int r,register int ql,register int qr){
```

```
    if(ql>qr||l>r) return (ask){0,0};
    if(l>r) return (ask){0,0};
    ask res=(ask){0,0};
    for(register int i=r;i;i-=(i&-i))
        res=res+query(rt[i],1,len,ql,qr,0);
    for(register int i=l-1;i;i-=(i&-i))
        res=res-query(rt[i],1,len,ql,qr,0);
    return res;
}
struct node{
    int l,r,v;
    friend bool operator <(node a,node b){
        return a.l<b.l;
    }
}tmp1,tmp2,tmp;
set<node> s;
set<node>::iterator it,it1,it2;
ll ans;
inline void split(register int x){
    it=s.upper_bound((node){x,0,0});it--;
    if(it->l==x) return;
    tmp1=(node){it->l,x-1,it->v};
    tmp2=(node){x,it->r,it->v+x-it->l};
    add(it->l,it->v+x-it->l,it->v+it->r-it->l,-1);
    add(x,it->v+x-it->l,it->v+it->r-it->l,1);
    s.erase(*it);
    s.insert(tmp1);s.insert(tmp2);
}
inline void update(register int l,register int r,register int x){
    if(l!=1) split(l);
    if(r!=n) split(r+1);
    while(1){
        tmp=*s.lower_bound((node){l,0,0});
        if(tmp.l==r+1) break;
        int tl=tmp.l,tr=tmp.r,vl=tmp.v,vr=tmp.v+tmp.r-tmp.l;
        add(tl,vl,vr,-1);
        ans-=find(1,tl-1,vr+1,len).v*(tr-tl+1)+find(tr+1,n,1,vl-1).v*(tr-tl+1);
        ask res=find(1,tl-1,vl,vr);
        ans-=res.sum-res.v*vl;
        res=find(tr+1,n,vl,vr);
        ans-=res.v*vr-res.sum;
        s.erase(tmp);
    }
    s.insert((node){l,r,x});
    add(l,x,x+r-l,1);
    register int vr=r-l+x,vl=x;
    ans+=find(1,l-1,vr+1,len).v*(r-l+1)+find(r+1,n,1,vl-1).v*(r-l+1);
    ask res=find(1,l-1,vl,vr);
    ans+=res.sum-res.v*vl;
    res=find(r+1,n,vl,vr);
    ans+=res.v*vr-res.sum;
}
signed main(){
    n=read();m=read();
    for(register int i=1;i<=n;i++){
        int x=read();
        s.insert((node){i,i,x});
        add(i,x,x,1);
        ans+=find(1,i-1,x+1,len).v;
    }
    s.insert((node){n+1,0,0});
    for(register int i=1;i<=m;i++){
        int opt=read();
        if(opt==1){
            int l=read(),r=read(),x=read();
            update(l,r,x);
        }
        if(opt==2) printl(ans);
    }
    return 0;
}
```

Y_B_X 创建时间：2021-11-21 22:17:15

原题链接

> 题意：给定一个序列，支持两种操作：
> $1\ l\ r\ x$：区间覆盖为从 x 开始的公差为 1 的等差数列。
> 2：查询序列的逆序对个数。

Step 1

首先可以发现每个被覆盖的区间，其内部点对之间**没有**对逆序对的贡献。

并且能与这部分区间差生逆序对的点**下标与值**都在一个可控的范围内。

思路类似于镜中的昆虫。

用**珂朵莉树**维护原序列中的被覆盖的段，那一次覆盖操作直接遍历这些段的时间将是 $O(n+m)$。

对于这些段，可以先去除它与 $[1,l-1]$ 与 $[r+1,n]$ 的点之间的逆序对，以及这些段之间的逆序对。

事实上设其中一个段为 $[sl,sr]$，则只需对 $[1,sl-1]$ 与 $[r,n]$ 查询就能包含到这两种情况。

之后还要加上新形成的段与 $[1,l-1]$ 与 $[r+1,n]$ 的逆序对。

Step 2

接下来考虑如何具体实现。

发现上面要求的只是位于 $>r$ 或 $<l$ 的段与一个段的逆序对。

而时间戳又是天然另一维，形成了三维偏序，想到 cdq 分治。

cdq 先干掉时间上的限制，然后位置限制排序解决，剩下是值域上的偏序。

注意这层偏序无论修改还是查询都是基于一个**连续的值域区间**。

先只考虑 $>r$ 的查询，这需要对值域区间内的每一个点，找出值小于的它的个数之和。

但可以反过来对修改考虑对查询的贡献。

设一次修改为区间 $[sv,tv]$，那就能对 $i\in[sv+1,tv+1]$ 的值 i 加上 $i-sv$，而对 $i>tv+1$ 的值 i 加上 $tv-sv+1$。

这就是个**二维前缀和**，对 $sv+1$ 处 $+1$ 而对 $tv+2$ 处 -1。

这样对一个值只需看其被加了几次，就相当于真正有多少个数比它小。

但对于一个查询的区间 $[sv,tv]$ 要累加范围内的每一个点被加几次。

然后发现又多了一个前缀和，成功成为**三维前缀和**。

不妨手推一下得到式子：

$$\sum_{i=1}^{l}\sum_{j=1}^{i}\sum_{k=1}^{j}t_k=\frac{1}{2}\left((l+1)(l+2)\sum_{i=1}^{l}t_i-(2l+3)\sum_{i=1}^{l}it_i+\sum_{i=1}^{l}i^2t_i\right)$$

对后面的三个求和，树状数组能十分方便维护。

而对于 $<l$ 的查询，需要对值域区间每个点，找出比它大的点的个数，可通过反转值域以同样的方法实现。

时间复杂度为 $O(n\log^2 n)$，空间复杂度为 $O(n)$，默认 n,m,V 同阶。

代码：

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int N=1e6+10;
int n,nn,m,x,y,opt,l,r,v,tt,totq,totp;
int a[N];int tmp,res,ans[N];
char ch;int at[N],_a;
inline void read(int &x){
```

```
inline void read(int &x){
    x=0;ch=getchar();while(ch<48)ch=getchar();
    while(ch>47)x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
}
inline void write(ll x){if(x>9)write(x/10);putchar(48|x%10);}
inline int max(int a,int b){return a<b?b:a;}
inline int min(int a,int b){return b<a?b:a;}
#define lowbit(i) i&(-i)
struct bit{
    int t[N],res;
    void clear(){for(int i=1;i<=nn;++i)t[i]=0;}
    inline void clear(int i){for(;i<=nn;i+=lowbit(i))t[i]=0;}
    inline void inquiry(int i){res=0;for(;i;i-=lowbit(i))res+=t[i];}
    inline void update(int i,int v){for(;i<=nn;i+=lowbit(i))t[i]+=v;}
}T,T_,T__;
struct cdq{
    int i,pos,l,r,v;char opt;cdq()=default;
    cdq(int _i,int _pos,int _l,int _r,int _v,char _opt):
        i(_i),pos(_pos),l(_l),r(_r),v(_v),opt(_opt){}
    //opt=1/-1:inq || opt=0:mdf
}q[N],q0[N],p[N],p0[N];// q for < || p for >
inline void clear(int pos){T.clear(pos);T_.clear(pos);T__.clear(pos);}
inline void update(int pos,int v){
    T.update(pos,v);T_.update(pos,v*pos);T__.update(pos,v*pos*pos);
}
inline void inquiry(int pos){
    T.inquiry(pos);T_.inquiry(pos);T__.inquiry(pos);
    res=(T.res*(pos+1)*(pos+2)-T_.res*(2*pos+3)+T__.res)>>1;
}
inline bool cmpq(const cdq a,const cdq b){return a.pos^b.pos?a.pos<b.pos:b.opt;}
inline bool cmpp(const cdq a,const cdq b){return a.pos^b.pos?a.pos>b.pos:b.opt;}
void solveq(int l,int r){if(l^r){
    int mid=(l+r)>>1;register int i;
    solveq(l,mid);solveq(mid+1,r);
    merge(q+l,q+mid+1,q+mid+1,q+r+1,q0+l,cmpq);
    for(i=l;i<=r;++i)q[i]=q0[i];
    for(i=l;i<=r;++i){
        if(q[i].i<=mid&&!q[i].opt){
            update(nn-q[i].r+2,q[i].v);
            update(nn-q[i].l+3,-q[i].v);
        }
        else if(q[i].i>mid&&q[i].opt){
            inquiry(nn-q[i].l+1);ans[q[i].v]+=q[i].opt*res;
            inquiry(nn-q[i].r);ans[q[i].v]-=q[i].opt*res;
        }
    }
    for(i=l;i<=r;++i)if(q[i].i<=mid&&!q[i].opt)
        clear(nn-q[i].r+2),clear(nn-q[i].l+3);
}}
void solvep(int l,int r){if(l^r){
    int mid=(l+r)>>1;register int i;
    solvep(l,mid);solvep(mid+1,r);
    merge(p+l,p+mid+1,p+mid+1,p+r+1,p0+l,cmpp);
    for(i=l;i<=r;++i)p[i]=p0[i];
    for(i=l;i<=r;++i){
        if(p[i].i<=mid&&!p[i].opt){
            update(p[i].l+1,p[i].v);
            update(p[i].r+2,-p[i].v);
        }
        else if(p[i].i>mid&&p[i].opt){
            inquiry(p[i].r);ans[p[i].v]+=p[i].opt*res;
            inquiry(p[i].l-1);ans[p[i].v]-=p[i].opt*res;
        }
    }
    for(i=l;i<=r;++i)if(p[i].i<=mid&&!p[i].opt)
        clear(p[i].l+1),clear(p[i].r+2);
}}
struct node{
    int l,v;mutable int r;node(int _l):l(_l){}
    node(int _l,int _r,int _v):l(_l),r(_r),v(_v){}
    inline bool operator <(const node a)const{return l<a.l;}
};
set<node> s;
#define sit set<node>::iterator
sit itl,itr,its,it;int sl,sr,sv,tv;
```

```
inline void split(int pos){
    its=s.lower_bound(node(pos));
    if(its==s.end()||its->l^pos){
        --its;
        sl=its->l,sr=its->r,sv=its->v;
        ++totq,q[totq]=cdq(totq,sl,sv,sv+sr-sl,-1,0);
        ++totp,p[totp]=cdq(totp,sr,sv,sv+sr-sl,-1,0);
        its->r=pos-1;
        ++totq,q[totq]=cdq(totq,sl,sv,sv+pos-1-sl,1,0);
        ++totp,p[totp]=cdq(totp,pos-1,sv,sv+pos-1-sl,1,0);
        its=s.insert(node(pos,sr,sv+pos-sl)).first;
        ++totq,q[totq]=cdq(totq,pos,sv+pos-sl,sv+sr-sl,1,0);
        ++totp,p[totp]=cdq(totp,sr,sv+pos-sl,sv+sr-sl,1,0);
    }
}
main(){
    nn=3e4;register int i;
    read(n);read(m);
    for(i=1;i<=n;++i){
        read(x);T.update(x,1);
        s.insert(node(i,i,x));
        T.inquiry(x),ans[0]+=i-T.res;
        ++totq,q[totq]=cdq(totq,i,x,x,1,0);
        ++totp,p[totp]=cdq(totp,i,x,x,1,0);
    }
    T.clear();
    for(i=1;i<=m;++i){
        read(opt);
        if(opt==1){
            read(l),read(r),read(v);++tt;
            itl=l^1?(split(l),its):s.begin();
            itr=r^n?(split(r+1),its):s.end();
            for(it=itl;it!=itr;++it){
                sl=it->l,sr=it->r;
                sv=it->v;tv=sv+sr-sl;
                if(sl^1)++totq,q[totq]=cdq(totq,sl-1,sv,tv,tt,-1);
                if(r^n)++totp,p[totp]=cdq(totp,r+1,sv,tv,tt,-1);
            }
            for(it=itl;it!=itr;++it){
                sl=it->l,sr=it->r;
                sv=it->v;tv=sv+sr-sl;
                ++totq,q[totq]=cdq(totq,sl,sv,tv,-1,0);
                ++totp,p[totp]=cdq(totp,sr,sv,tv,-1,0);
            }
            s.erase(itl,itr);s.insert(node(l,r,v));tv=v+r-l;
            if(l^1)++totq,q[totq]=cdq(totq,l-1,v,tv,tt,1);
            if(r^n)++totp,p[totp]=cdq(totp,r+1,v,tv,tt,1);
            ++totq,q[totq]=cdq(totq,l,v,tv,1,0);
            ++totp,p[totp]=cdq(totp,r,v,tv,1,0);
        }
        else at[++_a]=tt;
    }
    solveq(1,totq);solvep(1,totp);
    for(i=1;i<=tt;++i)ans[i]+=ans[i-1];
    for(i=1;i<=_a;++i)write(ans[at[i]]),putchar('\n');
}
```