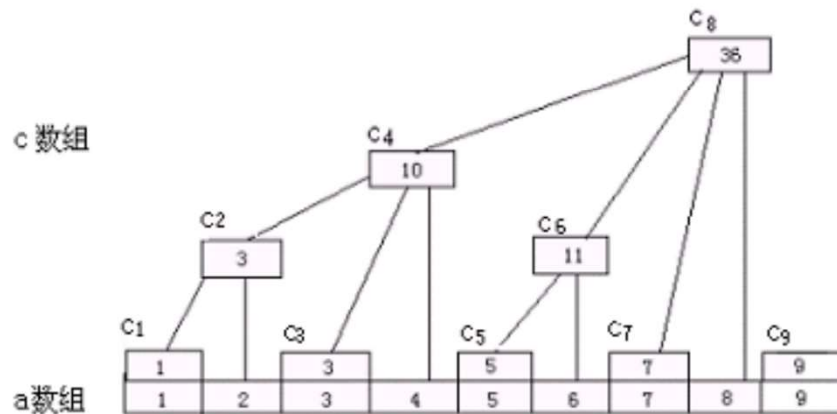


数状数组学习笔记-----林大 CHENYU

树状数组或者二叉索引树也称作 Binary Indexed Tree, 又叫做 BIT; 它的查询和修改的时间复杂度都是 $\log(n)$, 空间复杂度则为 $O(n)$, 这是因为树状数组通过将线性结构转化成树状结构, 从而进行跳跃式扫描。通常使用在高效的计算数列的前缀和, 区间和。

BIT 是一个好东西, 代码段, 复杂度低, 但是要理解透了才行!

树状数组图示



先看懂原理, 我就喜欢死扣原理!

通过上面的图发现: 奇数下标的 $a[i]$ 就是自己本身的值! 偶数下标的 $a[i]$ 存放的不是自己本身, 而是几个数的和! 到底是几个数呢? 二分思想, 下标等于 2^n 的时候, 存储的是左边的所有和。而对于下标=6来说, $a[6]$ 只能存 $(a[5]+a[6])$ 的和; 貌似 $a[10]$ 也只能存储 $a[9]+a[10]$

我们就可以自己在纸上画出任意下标的树状数组的图了! 画 $a[9]-----a[16]$

那么 $a[10]$ 的下一个元素的下标是啥呢? ? ? ? ? ?

10 后面的最近的 2^n 是 16,

10 对应的 2 进制为 1010

各位取反 0101

末尾+1 0110 (-10)

10 和 (-10) 做一下 & 运算: $0010 = 2$

所以 10 的下一个点为 12;

1&1=1 0&0=0 1&0=0 0&1=0-----够细致了吧!

12 的 2 进制是: 1100,

各位取反： 0011

末尾+1： 0100 (-12)

12 和 -12 做 & 运算： 0100 = 4；

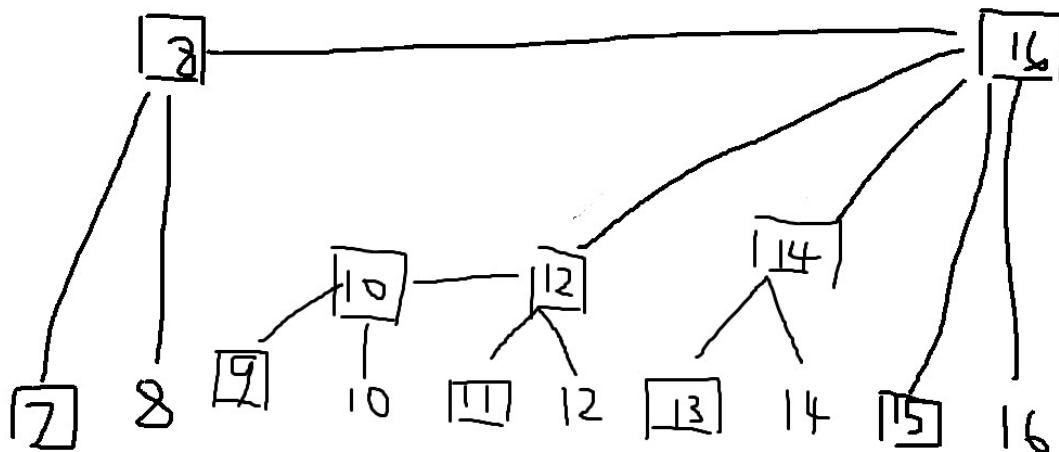
所以 12 的下一位是 16！

12 是 8 和 16 的中间点，所以说这里是二分思想（白书里有说）

同理 14 的下一位也是 16！

结论：(i&-i) 就是每个元素往前和往后的下标增量；

下面是我手工画的图（用画笔），是不是画得很漂亮啊！



以上是关于 lowbit 的产生，就此打住！

一个理解问题：a[7]的值，就存在 a[7]里，但 a[8]里存的可不是第 8 个输入的值啊！

偶数下标的值，全部丢失，后面通过单点查询才可以求出来！

这里有 2 个序列：1 个是原始输入的序列，1 个是 BIT（树状数组）维护的序列，就是我画图的序列，而刚开始输入的原始序列基本上输入完就消失了，没存储。一切的一切要根据维护的序列来工作。

下面看一下基本函数的写法：

```
int update(int i,int v)
{
    while(i<=n)
    {
        a[i]+= v;
        i+=(i&-i);
    }

    return 0;
}
```

```

int sum(int i)
{
    int res=0;
    while(i>0)
    {
        res+=a[i];
        i=i-(i&-i);
    }
    return res;
}

```

----- 单点更新和区间查询：

这里没啥好讲的，直接做模版题就好！单点更新后一般是没操作的，一般都跟着区间查询，那就查询一下就可以了！

洛谷的模板：3374 的代码：

```

#include <bits/stdc++.h>
using namespace std;
const int N=5e5+5;
int a[N];
int n,m;
int update(int i,int v)
{
    while(i<=n)
    {
        a[i]+= v;
        i+=(i&-i);
    }
}

return 0;
}

```

```

int sum(int i)
{int res=0;
while(i>0)
{
    res+=a[i];
    i=i-(i&-i);
}
}

```

```

    return res;
}

int main()
{
    ios::sync_with_stdio(0);
    int op,x,y,t;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        {cin>>t;
         update(i,t);

        }
    for(int i=1;i<=m;i++)
    {
        cin>>op>>x>>y;
        if (op==1)
        {

            update(x,y);
        }
        if (op==2)
        {
            int ans=0;
            ans=sum(y)-sum(x-1);
            cout<<ans<<endl;

        }

    }

    //cout << "Hello world!" << endl;
    return 0;
}

```

区间更新和单点查询：

这个好像复杂一点点，也就是我们说的差分，注意概念，就是差分！

说一下差分：

现在我们有一个从小到大的数列 $a[]$

a 1 3 6 8 9

然后还有一个差分数组 $b[]$

b 1 2 3 2 1 对应： 1, 3-1,6-3,8-6,9-8,

相信某些同学绝已经看出端倪了..这里 $b[i]=a[i]-a[i-1]$ ，我令 $a[0]=0$ ，故 $b[1]=a[1]$ 。

A 数组中的 8 等于 B 数组 前 4 项的和；同理：A 数组中的 6 等于 B 数组中前 3 项的和，

于是单点查询变成了 B 数组的 BIT 求和问题，用模版即可！

然后把差分后的序列扔到树状数组里：

```
For(int i=1;i<=5;i++)
{
    Update(i,b[i]);
}
```

再说区间修改..

我们知道，树状数组对于单点值的修改十分方便（不懂的去看树状数组 1），对于区间的修改就比较尴尬..而我们又不想敲死长的线段树..怎么办呢，这时候差分就显出优势

还是上面的 a[] 和 b[]，现在我们使区间[2,4]的所有数均+2，则 a[]/b[]变为

a 1 5 8 10 9

b 1 4 3 2 -1

事实上，这里只有 b[2]和 b[5]发生了变化，因为区间内元素均增加了同一个值，所以 b[3], b[4]是不会变化的。

这里我们就有了第二个式子：对于区间[x,y]的修改（增加值为 d）在 b 数组内引起变化的只有 b[x]+=d, b[y+1]-=d。（这个也很好推的..）

这样，我们就把树状数组的软肋用差分解决了。

下面是洛谷 3368 的代码：

```
#include <iostream>
using namespace std;
const int maxn=500005;
int n,m;
int a[maxn],b[maxn],c[maxn];
int update(int i,int x)
{
    while(i<=n)
    {a[i]+=x;
    i+=(i&-i);
    }
}
int sum(int i)
{
    int ans=0;
    while(i)
    {
        ans+=a[i];
        i-=(i&-i);
    }
    return ans;
}
int main()
{
    ios::sync_with_stdio(false);
```

```

int t,x,y,k;
cin>>n>>m;
for(int i=1;i<=n;i++)
{
    cin>>b[i];
    if (i==1) c[i]=b[i];
    else
        c[i]=b[i]-b[i-1];
    update(i,c[i]);
}
for(int i=1;i<=m;i++)
{
    cin>>t;
    if (t==1)
    {
        cin>>x>>y>>k;
        update(x,k);
        update(y+1,-k);
    }
    if (t==2)
    {
        cin>>x;
        cout<<sum(x)<<endl;
    }
}
return 0;
}

```

树状数组的应用

把问题转化成用 BIT 能解决的思路是有难度的，这里的练习，对于我们过几天学线段树有大帮助，会加快我们学习线段树的周期。

第一种：类似洛谷【小鱼比可爱】和 POJ 和林大【数星星】的题目，计算我左边比我小的数有多少？

是按照输入的顺序计算左边比我小的个数有多少？这里的误区在于区分整个序列里有多少比我小（用前缀和计算），而输入的顺序比我小的数用 BIT；题目的变化还有【比我小或者小于等于我】，这是要弄清楚的！

【比我小】：sum(x-1);-----然后在 update(x,1);

【小于等于我】：sum(x)-----然后在 update(x,1)

因为谁告诉你数据没有重复呢？做数星星的那道题，在纸上好好画画~~

第二种：就是以前做过的“逆序对”的题目，以前是归并排序做的，现在看来归并大法还是好啊，因为动态逆序对一归并还可以做，而 BIT 则就不行了，但这题是 BIT 的菜，板子题；

先把题目转化一下思想：

3 2 5 1 这组数，把这 4 个值看做下标，记住是下标

在 BIT 的图里，下标是顺序的，但输入的顺序是不一样的，update(x,1)的时间是按照输入的顺序更新的！

3

2 这时候，3 已经更新了，也就是在 2 的后面的和里，已经有一个 1 了！

大脑考虑这个 1 (3 产生的) 其实一直叠加到 $a[n]$ 里， n 是 x 下标的最大值！

当 2 进来时，时下面的图： $a[3]$ 里已经有值了，3 在 2 的后面 (BIT 里)



所以计算逆序对，就看 $a[x+1]$ ----- $a[n]$ 的和就行！($\text{sum}(n) - \text{sum}(x)$)

这里的 n 不是代表数组的个数，绝对不是，是下标的最大值，否则 RE

下面说一下昨天我在群里说的离散化问题：

【离散化】：

因为是把输入的序列值当做 BIT 的下标，所以 x 的值能达到 INT 的最大值！

定义 BIT 数组的时候是没办法开这么大的数组的，只能离散化！

下面说一下我的 lower_bound() 离散化的方法，大家可以自己尝试一下：

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 10005;
int a[maxn], t[maxn];
int n;
int main() {
    scanf("%d", &n);

    for(int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        t[i] = a[i]; //t 数组相当于 a 数组的拷贝
    }
```

```

sort(t + 1, t + 1 + n); //将 t 数组排序

int m = unique(t + 1, t + 1 + n) - t - 1; //去重, m 是 a 数组不重复元素个数

cout<<m<<endl;

for(int i = 1; i <= n; i++) {
    a[i] = lower_bound(t + 1, t + 1 + m, a[i]) - t; //下标从 1 开始
}

for(int i=1;i<=n;i++)
{
    printf("%d ", a[i]); //离散化后的值, 不改变相对顺序
}

//下面是从 0 开始输入的

for(int i=0;i<n;i++)
    cin>>a[i], t[i]=a[i];

sort(t, t+n);

int m=unique(t, t+n)-t;

cout<<"m="<m<<endl;

for(int i=0;i<n;i++)
{
    a[i]=lower_bound(t, t+m, a[i])-t+1;

    cout<<a[i]<<" ";
}

cout<<endl;

return 0;
}

```

-----以下为手工离散化的方法-----

树状数组的下标不能为 0, 所以输入的 X 要+1 !

输入数据可能为 0, 所以输入值要加 1 ;

10	12	9	15
1	2	3	4

结构体 2 个变量

然后根据第一行从小到大排序：结果：

9	10	12	15
---	----	----	----

3	1	2	4
---	---	---	---

然后把第一行换成 1 2 3 4 (因为第一行的数很大, 能到 100000000)

1	2	3	4
---	---	---	---

3	1	2	4
---	---	---	---

如果第一行有重复的数字, 要注意：

9	9	12	15
---	---	----	----

3	1	2	4
---	---	---	---

则要换成：

1	1	2	3
---	---	---	---

3 1 2 4

然后按照第 2 行从小到大的顺序再排序：

2 3 1 4

1 2 3 4

这是第一行的数字，就是离散后的结果，把这个结果扔进树状数组，就可以计算逆序数了！

逆序对代码：

```
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <algorithm>
using namespace std;
int a[40004],b[40004],c[40004],d[40004];
int n,pp;
struct sa
{
    int big;//大的数
    int sm;//离散化后的数
}vis[40004];
int cmp(const sa &a1,const sa &b1)
{
    return a1.big < b1.big;
}
int cmp1(const sa &a,const sa &b)
{
    return a.sm < b.sm;
}
int update(int i,int x)
{
    while(i<=n)
    {
        a[i]+=x;
        i+=(i&-i);
    }

    return 0;
}
int sum(int i)
{
    int ans=0;
    while(i>0)
    {
```

```

        ans+=a[i];
        i-=(i&-i);

    }
    return ans;

}

int main()
{
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    memset(c,0,sizeof(c));
    memset(d,0,sizeof(d));
    int ans=0,tmp;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&b[i]);
        c[i]=i;
        vis[i]={b[i],c[i]};
    }
    sort(vis+1,vis+1+n,cmp);
    int num=0;
    for(int i=1;i<=n;i++)
    {
        if (vis[i].big!=vis[i-1].big)
            d[i]=(++num);
        else
            d[i]=d[i-1];
    }

    for(int i=1;i<=n;i++) vis[i].big=d[i];
    sort(vis+1,vis+1+n,cmp1);

    for(int i=1;i<=n;i++)
    {
        update(vis[i].big,1);
        ans+=(sum(n)-sum(vis[i].big));
    }

    cout <<ans << endl;
    return 0;
}

```

【数星星】 代码：

```
#include <iostream>
#include <algorithm>
#include <string.h>
const int maxn=32000+10;
int a[maxn],d[maxn];
int n;
int update(int i,int x)
{
    while(i<=maxn)
    {
        a[i]+=x;
        i+=(i&-i);
    }
}

int sum(int i)
{
    int ans=0;
    while(i)
    {
        ans+=a[i];
        i-=(i&-i);
    }
    return ans;
}

using namespace std;

int main()

{
    ios::sync_with_stdio(false);
    while(cin>>n)
    {memset(a,0,sizeof(a));
        memset(d,0,sizeof(d));
        int x,y;

        for(int i=1;i<=n;i++)
        {
            cin>>x>>y;
            x++;
```

```

        int tmp=sum(x);
        update(x,1);
        d[tmp]++;

    }

    for(int i=0;i<n;i++)
        cout<<d[i]<<endl;
    }

    return 0;
}

```

【小鱼比可爱】代码：

```

#include <iostream>
#include <string.h>
#include <algorithm>
using namespace std;
int n;
int a[10000],b[10000],c[10000],d[10000];
struct sa
{
    int big;
    int sm;
}vis[105];
int cmp(const sa &a,const sa &b )
{
    return a.big <b.big;
}
int cmp1(const sa &a,const sa &b)
{
    return a.sm <b.sm;
}
int update(int i,int x)
{
    while(i<=105)
    {
        a[i]+=x;
        i+=(i&-i);
    }
}
int sum(int i)

```

```

{
    int ans=0;
    while(i)
    {
        ans+=a[i];
        i-=(i&-i);
    }

    return ans;
}

int main()
{
    memset(a,0,sizeof(a));

    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>b[i];
        //c[i]=i;
        vis[i]={b[i]+1,i};
    }

    sort(vis+1,vis+1+n,cmp); //第一次排序
    int num=0;
    for(int i=1;i<=n;i++) //计算把第一行换成 1 2 3 4 。。。的

```

数组

```

{
    if (vis[i].big!=vis[i-1].big)
        c[i]=++num;
    else
        c[i]=c[i-1]; //重复的处理
}

for(int i=1;i<=n;i++) //把第一行换成 1 2 3 4
    vis[i].big=c[i];
sort(vis+1,vis+1+n,cmp1); //第 2 次排序

```

```

for(int i=1;i<=n;i++)
{
    d[i]=sum(vis[i].big-1);//本题是说比自己满意度小的，所以
    是-1;如果是小于等于自己的，则不能-1;

    update(vis[i].big,1);
}
for(int i=1;i<=n;i++)
    cout<<d[i]<<" ";

return 0;
}

```

林大 OJ 1465 【门前的树】这题

这题直接做是不行的，这题是在纸上反复画就可以了。

误区一：简单的认为可以区间更新+1，然后再找最大值就行，这是不对的！

比如：【2,4】区间更新+1， 【7,10】区间更新+1；

然后询问 【3,8】 的树的种类，区间最大值还是 1，并不是 2！

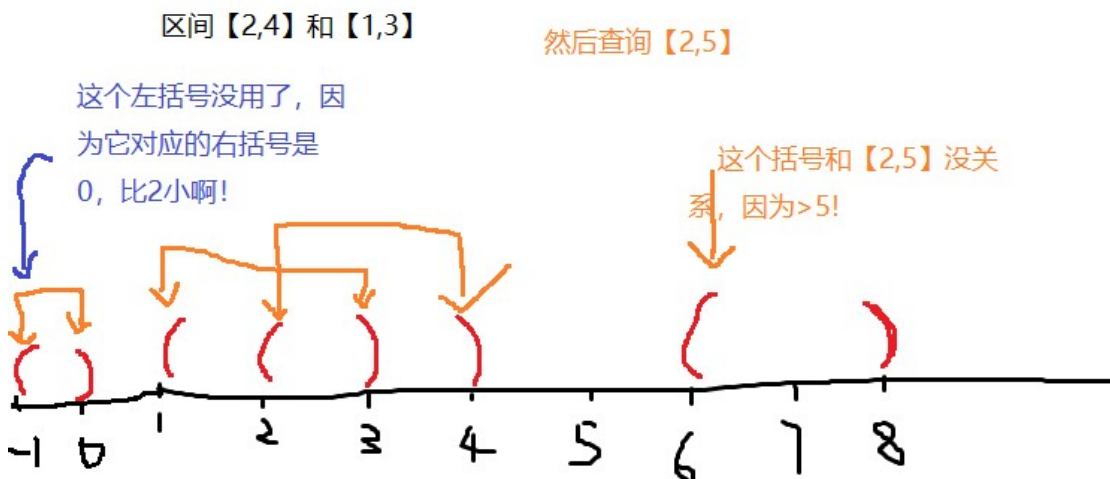
所以区间更新-----求最大值得思路就死心吧-----即使可以的话，计算最大值也会 TLE 的！

正确思路：

在纸上 起点---就画 (括号；

终点---就画) 括号；

然后自己-----看-----看-----看-----发现………噢：



根据这个图，发现对于区间【2,5】，5 前面所有的（括号所产生的区间有可能都在【2,5】范围内！

而 5 后面的（括号，都没用！

但 5 前面（包括 5）有很多（括号对应的区间在 2 之前就结束了，例如-1 点的（括号，它对

应的) 点是 0, 比 2 小啊 !

结论 : 5 点 (包括 5) 左边的所有左括号-----减去 2 左边失效的左括号-----就是答案 !

2 左边失效的左括号等于 -----2 左边的右括号 !

使用 2 个树状数组, 分别存储 左括号和右括号, 然后 $\text{sum}(\text{终点} : \text{左括号数组}) - \text{sum}(\text{起点} - 1 : \text{右括号数组})$ 就可以了 !

Nefu-1465 代码 :

```
#include <iostream>
#include <string.h>
#include <cstdio>
using namespace std;
const int maxn=1e5;
int a[maxn],b[maxn];
int n,m;
int update1(int i,int x)
{
    while(i<=n)
    {
        a[i]+=x;
        i+=(i&-i);
    }
    return 0;
}
int update2(int i,int x)
{
    while(i<=n)
    {
        b[i]+=x;
        i+=(i&-i);
    }
    return 0;
}
int sum1(int i)
{
    int ans=0;
    while(i)
    {
        ans+=a[i];
        i-=(i&-i);
    }
    return ans;
}
int sum2(int i)
{
    int ans=0;
    while(i)
    {
        ans+=b[i];
        i-=(i&-i);
    }
}
```

```

        return ans;
    }
    int main()
    {
        int k,x,y;
        scanf("%d%d",&n,&m);
        for(int i=1;i<=m;i++)
        {
            scanf("%d%d%d",&k,&x,&y);
            if (k==1)
            {
                update1(x,1);
                update2(y,1);
            }
            if (k==2)
            {
                int ans=sum1(y)-sum2(x-1);
                printf("%d\n",ans);
            }
        }
    }

    return 0;
}

```

洛谷 2345 【奶牛集会】

本题最开始暴力 AC 掉了， $4e8$ 的数据量啊！按道理应该卡住的，必须 $n \cdot \log n$ 的算法。

在纸上推导了很久，发现如果 v 要是不重复就好了，只建一个 BIT 数组就可以，可以参考差分的做法，但后来发现 v_i 和 v_j 是可以重复的，只好放弃，果断使用 2 个 BIT 数组，先按照 V 从小到大排序，保证处理时 V 的值都是比当前 V 小。

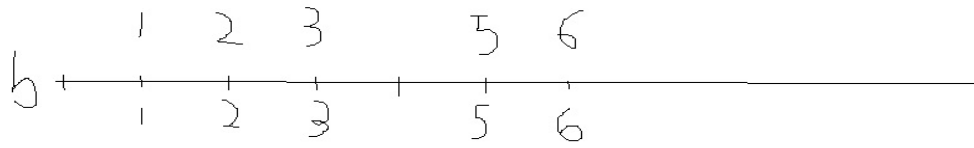
然后就是 X 的问题了，建立 $a[]$ ，存放 $x[i]$ 的个数 $update(a,1)$ ，统计比 $x[i]$ 小的有几个，比 $x[i]$ 大的有几个！

建立 $b[]$ ，统计小于 $x[i]$ 的和；还统计大于 $x[i]$ 的和，这 2 个 BIT 数组的下标都是 $x[i]$ ；

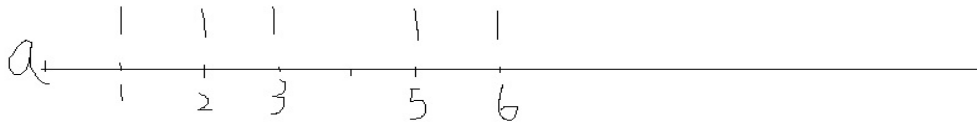
例如 x 值：

6 5 1 2 3

b数组记录x的值，下标也为x，可以统计小于x的和；大于x的和！



a数组的记录个数，下标为x,更新方式update(x,1)



比 3 小的有 2 个；比 3 小的和等于 3； $2*3-3=3$ ；
比 3 大的有 2 个；比 3 大的和等于 11； $11-2*3=5$ ；
3 对应的 V；答案= $v*(2*3-3+11-2*3)$ ；
树状数组先查询，然后计算后再更新；

```
```cpp
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
const int N=20005;
typedef long long LL;
LL a[N],b[N];
struct sa
{
 int v;
 int x;
}vis[N];
int n,tp;
int cmp(const sa &a,const sa &b)
{
 return a.v<b.v;
}
int update(LL *p,int i,int x)
{
 while(i<=tp)//这里是 tp， 不能是 n;
 {
 p[i]+=x;
 i+=(i&-i);
 }
}
```

```

 }
}
LL sum(LL *p,int i)
{ LL ans=0;
 while(i)
 {
 ans+=p[i];
 i=(i&-i);
 }
 return ans;
}
int main()
{ LL num1,num2,ans=0;
 LL add1,add2;
 tp=0;
 scanf("%d",&n);
 for(int i=1;i<=n;i++)
 {
 scanf("%d%d",&vis[i].v,&vis[i].x);
 tp=max(tp,vis[i].x);
 }
 sort(vis+1,vis+1+n,cmp);
 for(int i=1;i<=n;i++)
 { int r=vis[i].x;
 num1=sum(a,r);//小于 x 的个数 ;
 num2=sum(a,tp)-sum(a,r);//大于 x 的个数 ;
 add1=sum(b,r);//小于 x 的和 ;
 add2=sum(b,tp)-sum(b,r);//大于 x 的和 ;
 ans+=vis[i].v*(num1*r-add1+add2-num2*r);
 update(a,r,1);
 update(b,r,r);

 }
 printf("%lld\n",ans);
 return 0;
}

```

### 洛谷 【2344】 奶牛抗议

从题目描述，典型的 DP 题，写出入门级的 DP 代码：

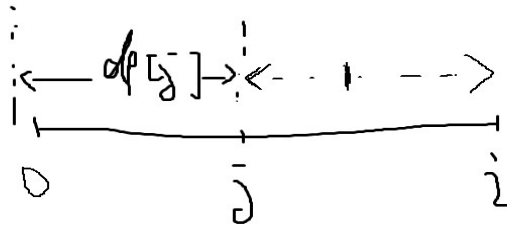
```

For (i=1;i<=n;i++)
 For(int j=i-1;j>=0;j--)
 If (sum[i]>=sum[j]) dp[i]=dp[i]+dp[j];

```

这里  $\text{sum}[i]$  代表奶牛的前缀和；

DP 的初始值  $\text{dp}[0]=1$ ;



$\text{dp}[j]$  表示长度为  $j$  的合法分类数，范围  $0-j$ ; 而  $j+1-----i$  区间，表示的是已经增序排列的一个区间了！就是区间和  $>0$   
根据乘法原理：方法数  $= \text{dp}[j] * 1$ ; 连续区间的方法数  $= 1$

用图说一下  $D$  的含义：

所以本题是用树状数组来维护 DP，这种题这两年也很常见到，唯一的区别在于更新的时候要更新查询的和；

```
Ans=sum(i);
```

```
Res+=ans;
```

```
Update(l,ans);
```

本题另外一个问题在于： $\text{dp}[0]=1$ ;

而  $\text{num}[i]$  只有  $n$  个， $\text{dp}[0]$  放到哪里呢？

把  $0$  和  $\text{num}[i]$  一起，共  $n+1$  个值，然后开始离散化，最后把第一数（也就是  $0$  离散化后的值），把他更新成  $1$ ，因为  $\text{dp}[0]=1$ ;

本题的困惑在  $\text{sum}[i]$  有正数和负数，所以  $0$  和这些数离散化的时候， $0$  就不是最小的了！

代码：

```
#include <iostream>
#include <stdio.h>
#include <algorithm>
#include <string.h>
using namespace std;
const int N=1e5+50;
const int mod=1000000009;
int a[N],d[N],h[N];
struct sa
{
 int big;
 int sm;
}vis[N];
int cmp(const sa &a,const sa &b)
{
 return a.big<b.big;
}
int cmp1(const sa &a,const sa &b)
{

```

```

 return a.sm<b.sm;
 }
 int n,tp;
 int update(int i,int x)
 {
 while(i<=tp+10)
 {
 a[i]=(a[i]+x)%mod;

 i+=(i&-i);
 }
 return 0;
 }
 int sum(int i)
 {

 int ans=0;
 while(i)
 {
 ans=(ans+a[i])%mod;

 i-=(i&-i);
 }
 return ans;
 }
 int main()
 {
 int ans,num=1;
 scanf("%d",&n);
 vis[0].big=0;
 vis[0].sm=0;;
 for(int i=1;i<=n;i++)
 {
 scanf("%d",&h[i]);
 vis[i].big=vis[i-1].big+h[i];
 vis[i].sm=i;
 }
 sort(vis,vis+1+n,cmp);
 d[0]=0;
 for(int i=0;i<=n;i++)
 {
 if (vis[i].big!=vis[i-1].big)
 d[i]=(++num);
 else
 d[i]=num;
 }
 }

```

```
}
tp=num;

for(int i=0;i<=n;i++)
 vis[i].big=d[i];
 //vis[0].big=1;
sort(vis,vis+1+n,cmp1);

ans=0;
update(vis[0].big,1);
for(int i=1;i<=n;i++)
{
 ans=sum(vis[i].big);
 update(vis[i].big,ans);
}
cout <<ans<<endl;
return 0;
}
```