树状数组

【问题】

给定一个序列 A，有两个操作：

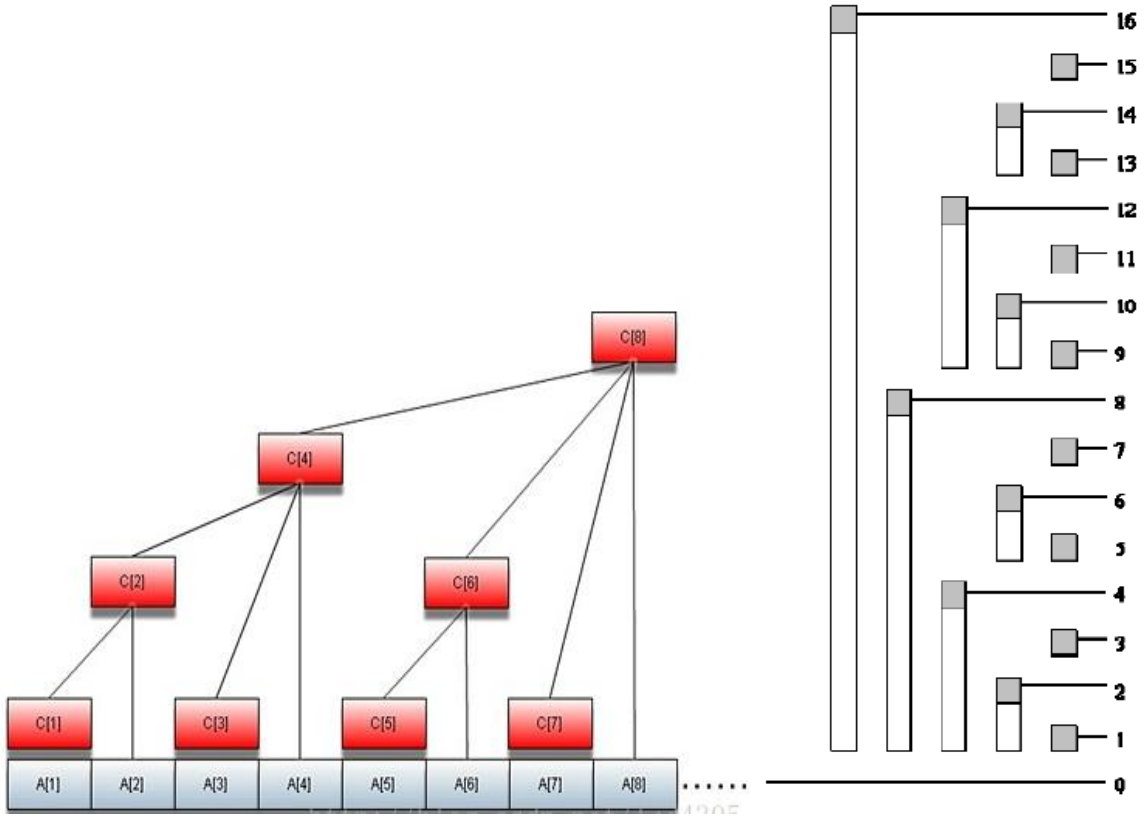
操作 1：将 A[i]+X；

操作 2：询问区间[l,r]的总和。

一、数状数组原理与实现

对于区间总和的询问，如果序列静态没有修改，那么可以用前缀和解决，但是如果带了修改后，前缀和就无法完成了，而树状数组便是解决这类问题的强有力工具。

树状数组是前缀和数组的升级版，他也是一个一维数组，然后每个位置维护了一段区间的总和。从含义上讲，它维护的区间长的像一颗树。如图：



数组 C 每个维护了序列 A 的一个区间的和，区间大小的覆盖关系，就构成一个树的关系。

那么，到底对于 C[i]来说，它应该维护哪个区间的和呢？

这里就要介绍一个很有意思的整数特点：**低位技术**。

对于每一个正整数 x，其对应的二进制数的最右边的 1 就称为最低位。如 $x=12=(1100)_2$，最低位的 1 权值为 4，则 4 就是 12 的低位。显然对于每个数，不停去掉低位就可以变为 0。

树状数组便是利用低位的特征，**C[i]维护 i 的低位长度的总和**。然后**查询前缀和**时，通过不停去掉低位从而达到拼接区间效果。如要查询 S[15]，查询过程为 C[15]+C[14]+C[12]+C[8],因为 15 的低位是 1，去掉低位后变为 14，而 14 低位是 2,去掉低位变为 12,12 低位为 4，去掉后为 8，最后查询 C[8]后结束。因为 C[i]维护 i 的低位长度和，查询时每次跳过低位长度刚好跳到 0 结束刚好完成前缀和查询。

当一个位置的数发生修改时，通过上图可知，他会影响所有覆盖改点的位置。如对于 A[5]修改时，可以看出对 C[5],C[6],C[8],C[16]..产生了影响。那么如何知道到底哪些点被影响

呢，我们发现一个规律，从 5-6-8-16 变化过程中，每次增加的数的大小是 2 的幂，也就是当前数的低位大小。如 5 的低位为 1，+1 等于 6,6 的低位是 2，+2 等于 8，如此类推。于是，对于修改一个位置的数，**不停的加上低位数**就可以找到所有覆盖该的区间对于位置（这就是为什么称为树的原因）。

具体实现：

1、求低位

```
int lowbit(int x){
    return x&-x;
}
```

2、查询

```
long long query(int k){
    long long ret=0;
    while(k){
        ret+=c[k];
        k-=lowbit(k);
    }
    return ret;
}
```

3、修改

```
    void update(int k,int  val){//位置 K 的数加 val
        while( k< = len){//len 表示序列总长度
            c[k]+=val;
            k+=lowbit(k);
        }
    }
```

4、总结

- 树状数组**本质是维护前缀和**。
- 统计每个点向前长度为 lowbit 的总和，通过二进制拼接得到前缀和。

二、实例应用

例 1：A2120 单点修改区间查询

【解析】模板题，直接套用模板即可。

【参考答案】

```
#include<bits/stdc++.h>
using namespace std;
#define N 1000010
#define lowbit(x) (x&(-x))
#define ll long long
ll c[N];int  n, m;
void update(int k,int v){
      while(k<=n) {c[k]+=v;k+=lowbit(k);}
}
ll query(int k){
```

```
    ll res=0;
    while(k) {res+=c[k];k-=lowbit(k);}
    return res;
}
inline int read(){
   int f=1,k=0;
    char c=getchar();
    while(c!='-'&&(c<'0'||c>'9')) c=getchar();
    if(c=='-') f=-1,c=getchar();
    while(c>='0'&&c<='9') k=(k<<3)+(k<<1)+c-48,c=getchar();
    return f*k;
}
int main(){
    n=read();m=read();
    for(int i=1;i<=n;i++){
        int x=read();
        update(____);
    }
    for(int i=1;i<=m;i++){
         int p,x,y;
         p=read(),x=read();y=read();
         if(p==1)
            update(x,y);
         else
            printf("%lld\n",query(y)-_______);
    }
    return 0;
}
```

例 2：A2005 逆序对

题意：给出一个序列，计算其中逆序对数量。

【解析】有的时候，我们采用另一种记录方法：a[i]表示值为 i 的数有几个，即计数数组。

这种表示的应用也很多，LIS 就是采用这种记录方法。

这样记录的问题是，复杂度跟值的大小有关系，当值很大时，需要提前**离散化**。

比如，在本题求逆序对，我们可以用前面学过的归并排序来计算，也可以用一个新的办法，我们记录一个每个数出现的次数，这样在遍历一个新数 i 时，只需要去看比他大的数出现的次数即可。即设 a[i]表示 i 出现的次数，每一次新增一个数 x，先查询大于 x 的数量，再将 x 统计到数组 a 中。

离散化。

离散化即将一个数哈希映射为另一个数，具体做法是将所有要离散化的数字排序，排序序列的位置来代表这个数本身。具体做法参考本题代码。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define N 100100
```

```
#define ll long long
int n,len,ret,ans,a[N],b[N],c[N];
inline int lowbit(ll x){
    return x&(-x);
}

inline void update(int x,int  y){
    for(;x<=len;x+=lowbit(x))
        c[x]+=y;
}

inline int query(int x){
    int ret=0;
    for(;x;x-=lowbit(x))
        ret+=c[x];
    return ret;
}

int main(){
    scanf("%lld",&n);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        b[i]=a[i];
    }
    sort(a+1,a+n+1);
    len = unique(a + 1, a + n + 1) - a - 1;//去重
    for(int i=n;i>0;i--){//倒序遍历每个数，查询数组中小于 x 的数量
        int x=lower_bound(a+1,a+len+1,b[i])-a;
        //获取每个数离散后的值
        ans+=query(x-1);
        update(x,1);
    }
    printf("%lld\n",ans);
    return 0;
}
```

例 3：A2121 区间修改，单点查询

【解析】

树状数组的查询本质是查询前缀和，所以树状数组就是对前缀和的维护，而这里需要查询一个点的值，修改区间的值就需要对问题进行转换，转换的办法是利用差分数组。

差分数组和前缀和数组是互逆的，设差分数组 d[i]=a[i]-a[i-1]，则数组 d[i]的前缀和数组就是原数组。这样单点的查询就相当于在差分数组上查询前缀和。而对于原数组区间[L,R]的修改，只需要在差分数组上 d[L]+v,d[R+1]-v 即可。

总结起来，就是利用差分数组将问题转换为了单点修改，前缀和查询。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
#define ll long long
struct BIT{
    ll c[N];int len;
    void init(int n){
        memset(c,0,sizeof(c));len=n;
    }
    inline int lowbit(int x){return x&-x;}
    void update(int k,int v){
        for(;k<=len;k+=lowbit(k))
            c[k]+=v;
    }
    ll query(int k){
        ll ans=0;
        for(;k;k-=lowbit(k))
            ans+=c[k];
        return ans;
    }
}A;
int n,m;
int main(){
    scanf("%d%d",&n,&m);
    A.init(n);
    for(int i=1;i<=n;i++){
        int x; scanf("%d",&x);
        A.update(i,x);A.update(i+1,-x);
    }
    while(m--){
        int op,x,y,v;
        scanf("%d",&op);
        if(op==1){
            scanf("%d%d%d",&x,&y,&v);
            ____________;
            ____________;
        }
        else{
            scanf("%d",&x);
            printf("%lld\n",A.query(x));
        }
    }
    return 0;
}
```

例 4：A2122 区间修改，区间查询

【解析】

这个问题如果用树状数组来解决，也需要用差分思想对问题转换。

设原数组第 i 位的值为 a[i]，di=a_i-a_{i-1}，则有(这里认为 a_0=0)

则有：$a_x=\sum_{i=1}^{x} d_i$

问题询问的是：$\sum_{i=1}^{x} a_i$

可以变形为：$\sum_{i=1}^{x} a_i=\sum_{i=1}^{x}\sum_{j=1}^{i} d_j=\sum_{i=1}^{x}(x-i+1)d_i$

$$\sum_{i=1}^{x} a_i=(x+1)\bullet\sum_{i=1}^{x} d_i-\sum_{i=1}^{x} i\bullet d_i$$

观察上述两个式子，我们发现询问的前 x 个数的总和可以用两个数组树状来维护，第一个维护 di，另一个维护 i·di。

通过前面的分析，我们能知道，树状数组本质只能解决单点修改，查询前缀和的操作。对于不是这样的问题如果要用树状数组来解决则需要做一定问题转换。实际中，对于区间询问区间修改的操作，多数采取线段树的办法，但树状数组因为空间需求小，算法效率高的特点也是算法竞赛中非常重要的算法。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=1e6+10;
struct BIT{
      ll c[maxn];int n;
      void inti(int _n){
       n=_n;memset(c,0,sizeof(c));
      }
      inline int lowbit(int x){
       return x&-x;
      }
      inline void update(int k,ll v){
       for(;k<=n;k+=lowbit(k))c[k]+=v;
      }
      inline ll query(int k){
       ll ret=0;
       for(;k;k-=lowbit(k))ret+=c[k];
       return ret;
      }
}A,B;
int n,q;
signed main(){
```

```
    scanf("%d%d",&n,&q);
    A.inti(n);B.inti(n);
    for(int i=1;i<=n;i++){
     int x;scanf("%d",&x);
     A.update(i,x);A.update(i+1,-x);
     B.update(i,(ll)i*x);B.update(i+1,(ll)-(i+1)*x);
    }
    while(q--){
     int op,x,y,v;
     scanf("%d",&op);
     if(op==1){
         scanf("%d%d%d",&x,&y,&v);
         A.update(x,v);A.update(y+1,-v);
         B.update(x,(ll)x*v);B.update(y+1,(ll)-(y+1)*v);
     }else{
         scanf("%d%d",&x,&y);
         ll R=(ll)(y+1)*A.query(y)-B.query(y);
         ll L=(ll)x*A.query(x-1)-B.query(x-1);
         printf("%lld\n",R-L);
     }
    }
    return 0;
}
```

例 5：P2562 数星星

题意：给定 n 个平面上的点，定义每个点的等级是该点左下方（含正左正下）的点数量，统计每个等级的点数量

【解析】

由于点的信息是按 y 坐标优先的增序给出，所以不用考虑 y 的因数，只用考虑 x 的因数。由于之前的 y 坐标都小于等于当前的，后面的肯定也不会等级比当前点低，所以我们只需要统计之前的点中 x 坐标小于等于当前点的有多少个。树状数组维护

【启发】

本题完成后，最需要我们注意的是解决该问题的一个思想。当一些问题与二元组相关时，可以先对一元排序，然后考虑另外一元的因素即可，这种思维被称为降维思维，在一些复杂问题中非常有用。

例如：在树状数组求逆序中，也可以理解为进行了降维处理，根据逆序对的定义 ai>aj　i<j。一组逆序对就包含了两个维度：下标、值。计算中数组下标本身是有序的，所以利用树状数组不考虑下标因素统计值的关系。

这类问题还可以称为二维偏序问题