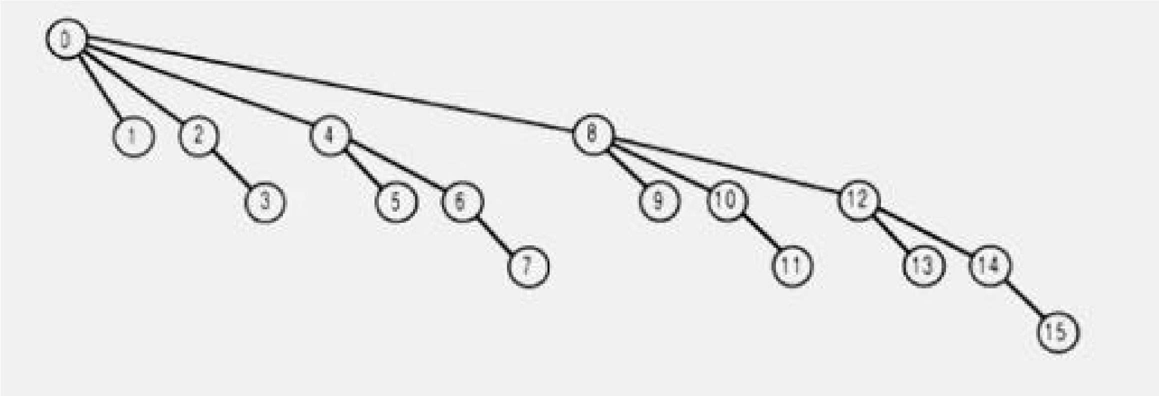
树状数组知识精炼

树状数组常用技巧1

- 单点查询

很显然：query(x)-query(x-1)
但这需要查询两次。



观察到:
a[x] = sum[x] - (query(x-1)- query(lca(x,x-1)))
也就是说 由sum[x] - sum(x-1 ~lca(x,x-1)) 即可。

核心代码

```
int query_pos(int x){
    int ans=sum[x],lca=x-lowbit(x);
    x--;
    while(x!=lca){
        ans-=sum[x];
        x-=lowbit[x];
    }
    return ans;
}
```

树状数组常用技巧2

- 元素非负，查询某个前缀和对应前缀下标 i。

即给定 k 求一个 i 使得 a1+a2+...ai = k。

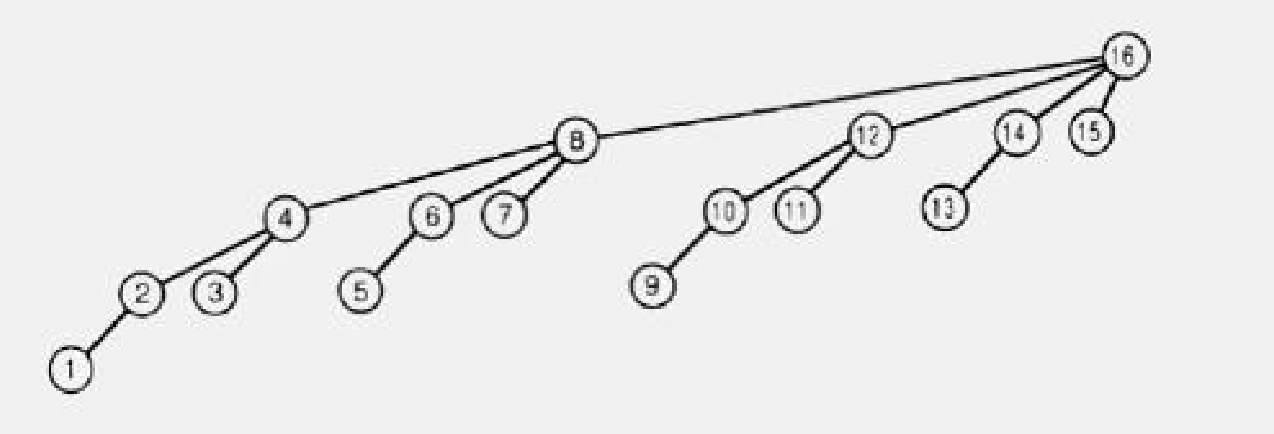
可以二分 i，但是变成了分治套分治（log^2)。

优化:

因为下标为2的幂次方的位置包含了从开始到自己的所有元素。

所以我们可以根据更新树进行二分，初始步长为 n，之后每次减半。

树状数组常用技巧




```
int get_index(int k){
    int i=0,len=n,t;
    while(len){
        t=i+len;
        if(k>=sum[t]){
            i=t;
            k-=sum[t];
        }
        len/=2;
    }
    return i;
}
```

树状数组常用技巧3

- 初始化树状数组

朴素做法：一个一个插入，O(nlogn)。

可以维护一个前缀和数组 pre[x]。
因为sum[x] 维护的是 a[(x-lowbit(x)+1) ~ x],
所以sum[x] = pre[x] - pre[x-lowbit(x)]。
时间复杂度O(n).

例1 poj2155 matrix

给一个 n*n 的二维数组 A，每个元素都是 0 或 1，一开始均为 0.

要求支持 2 种操作：

C x1 x2 y1 y2：对矩形区域所有元素做一次 非操作。

Q x y: 查询 A[x][y]的值

n<=1000, 0<q<50000.

sol

注意到 0 变 1, 1变 0 不好维护。

转化成每次非运算+1，然后判断这个数模2的值。

问题变成矩形+1和单点询问。

然后差分+树状数组维护即可。

例2 poj3468

维护一个长度为 n 的数组。

支持 区间加 和区间求和。

sol

区间加:

差分数组上进行单点加。

区间查询:

二次前缀和。

sum(ai) = (i+1)sum(di)-sum(i*di)

维护 sum(di) 和 sum(i*di)的树状数组。

每次修改: 1、d[l]+=d, d[r+1]-=d.

2、d[l]+=l*d, d[r+1]-=(r+1)*d

例3 hdu2852 kiki’s k-number

维护一个数据结构，支持m次操作:

0 x 插入一个数 x

1 x 删除一个数 x

2 a k 查询比 a 大的第 k 个数。

0<a,x,k<1e5, m<1e5

in:

```
5
0 5
1 2
0 6
2 3 2
2 8 1
7
0 2
0 2
0 4
2 1 1
2 1 2
2 1 3
2 1 4
```

out:

```
No Elment!
6
Not Find!
2
2
4
Not Find!
```

sol

注意到插入的数<1e5.

则可以维护值域数组A。

插入则 A[x]+1, 删除则 A[x]-1.

查找比 a 大的第 k 个数:

二分 i， 找到满足 sum(A[a+1,...i) = k 的 i

如果有重复的数则应满足:sum(A[a+1 ... i-1])<k,sum(A[a+1...i])>=k

进一步优化:

我们可以把sum(A[1...a])先求出来记为p，则可转化为求前缀和了。

例4 poj3067 Japan

Japan的西边和东边分别由 n 个和 m 个岛屿，最北边是1号，从北到南编号依次增加。

现在建了 k 座桥，每座桥连接西边第 a[i] 个岛和东边第 b[i] 个岛。

问这些桥会产生多少个交点。

n,m<=1000

in:
1
3 4 4
1 4
2 3
3 2
3 1

out:
5

sol

先分析交点：

对于桥 i 和 j，产生交点的情况：

a[i]<a[j] && b[i]>b[j]

或

a[i]>a[j]&& b[i]<b[j]

我们只统计第一种情况，这样对于每对 (i, j) 就只会统计一次。

sol

把（a[i], b[i]) 看做二维平面上的点;
问题转化成统计每个点左上方的点。

运用扫描线思想，对 y 从大到小排序，依次把 x 坐标插入树状数组，然后查询 [1, a[i]-1] 上的前缀和，就是每个点左上方点的个数。

时间复杂度 O((n+m)log(n+m))

例5 bzoj4822 老C的任务

二维平面上 n 个点，每个点有点权 w[i].

给出 m 个询问，每次询问一个矩形，问矩形覆盖的点的权值之和。

n,m<=1e5, -2^31<x,y<2^31

sol

题目没有要求在线，因此可以离线。

一个矩形可以拆成 4 个点的询问，每次询问点左下方点权之和。

运用扫描线思想，从小到大枚举 y 坐标，
如果是给定的点，则把点权插入到 x 坐标对应的树状数组的位置。
如果是询问点，就查询 [1...x]的前缀和。
注意坐标需要离散化。

例6 cf605D

给一个长度为 n 的序列。
每次在原序列做一个操作，把 a[i] 修改成 k.
问每次操作后序列的最长上升子序列长度。

n,m<=4e5.

in:	out:
4 4	4
1 2 3 4	3
1 1	3
1 4	4
4 3	
4 5	

sol

首先 LIS 有个 O(nlogn) 的做法。
我们可以求出以第 i 个数结尾的 LIS 长度 d[i]。
同理反着求一遍也可以求出以第 i 个数开头的 LIS 长度。

回到本题，对于每次修改 a[i] = x;
很显然有2种情况：
新的 LIS 包含 a[i].
新的 LIS 不包含a[i].

sol

我们先求一定包含 a[i] 的 LIS:

我们预处理出以第 i 个数结尾的LIS——left[i],
和以第i个数开头的LIS——right[i]。

我们只要求出 左边 max{ left[j] } (j<i, a[k]<x)
和右边 max{ right[j]} (j>i,a[j]>x)
两者长度相加再加1就是包含 a[i] 的新 LIS。

如果这个值大于原序列的LIS，那它一定是答案。

sol

再分析 LIS 不包含 x 的情况:

如果修改的数必须在原序列的LIS中:

　　则不包含修改的数的 LIS 一定会减少1。

如果修改的数可以不在原序列的LIS中:

　　不包含修改的数的 LIS 则不会变化。

问题就变成了对于序列中的每个数，判断其是否必须出现在LIS中。

sol

先用之前的思路算出包含 a[i] 的lis:

len[i] = max(left)+1+max(right)

如果len[i] = 原序列的LIS，那么 i 可能出现在LIS中。

如果 i 可能出现:

又如果存在 j<i 且 a[j]>=a[i] 且 j 可能出现在LIS中，i 有可以不出现在LIS中。

同理如果存在 j>i 且 a[j]<=a[i] 且 j 可能出现在LIS中，那么 i 可以不出现在LIS中。

所以如果 i 不满足可以不出现在 LIS中的条件，则 i 一定出现在LIS中。

sol

则对于每次修改a[i]=x， 先计算包含 x 的LIS长度。
如果比原序列LIS大，则一定是答案。

否则，如果 i 可以不出现，则LIS不变，否则如果 i 一定出现，则LIS会减少1.

sol

回过头来，考虑计算 max{ left[j]} (j<i ,a[j]<x)

这个就是经典二维数点问题。

对 a[j],x 排序。
从小到大枚举 x，把 a[j]<x 的 left[j] 插入树状数组的第 j 个位置。
我们把维护更新和询问树状数组的加法变成求 max.
问题转化成求前缀最大值。

例7 cf961e

给一个长度为 n 的序列 A。

问有多少个有序对 (x, y) 满足:

1、$x<y$

2、$A[x]>=y$

3、$A[y]>=x$

$n<=2e5$.

in:
3
8 12 7

out:
3

in:
3
3 2 1

out:
2

sol

先分析下条件，可以合并前 2 个得到:

数对满足 x<y<=A[x] 且 A[y]>=x 。

像这种有坐标有值的一般都类似于二维数点问题。我们可以用扫描线思想，通过对一维排序按顺序处理来满足这一维的要求，另一维就可以用数据结构来维护。

本题枚举 x 这一维方便，（只需要把>=x的A[y]加进来，否则如果枚举 y，x 有2种情况）

sol

因此我们从大到小枚举 x。

然后把 A[y]>=x 的 y 记录下来。

然后查询在区间 (x, A[x]] 的 y 的个数即可。

显然用树状数组就可以做到,时间复杂度 O(nlogn)。

本题维护 a[i] 的值时需要离散化，我们可以这样处理来避免离散化:

如果 A[i]>n, 那么可以把 A[i] 改为 n。

核心代码

```
#define N 200010
int n,a[N],sum[N],b[N];
int lowbit(int x){
    return x&(-x);
}
void change(int x,int v){
    while(x<=n){
        sum[x]+=v;
        x+=lowbit(x);
    }
}
int query(int x){
    int ret=0;
    while(x){
        ret+=sum[x];
        x-=lowbit(x);
    }
    return ret;
}
```

```
int cmp(int x,int y){
    return a[x]>a[y];
}
int main() {
    n=read();
    for(int i=1;i<=n;i++){
        a[i]=read();
        a[i]=min(a[i],n);
        b[i]=i;
    }
    sort(b+1,b+n+1,cmp);
    int j=0;
    ll ans=0;
    for(int i=n;i>=1;i--){
        while(j+1<=n&&a[b[j+1]]>=i){
            change(b[j+1],1);
            j++;
        }
        if(i<=a[i]){
            ans+=query(a[i])-query(i);
        }
    }
    printf("%lld",ans);
    return 0;
}
```

例8 cf827c

给一个长度只包含 A,T,C,G 的字符串，q次操作，每次有两种:

1、修改某个位置的字母

2、给定一个字符串 s (|s|<=10), 生成一个重复的新串 sssss…。询问

[L, R] 中有几个字母跟新字符串对应。

n,q<=1e5.

in:
ATGCATGC
4
2 1 8 ATGC
2 2 6 TTT
1 4 T
2 2 6 TA

out:
8
2
4

sol

设母串为 T，询问字符串为 s。
注意到每次询问中的字符串 s 长度不超过10,记 Len = |s|。

假设每次询问的区间都是 [1, n]。
我们只对 T 中前 Len 个字符考虑。
对于 s[i], 我们要判断的是 T[i], T[i+Len], T[i+2Len]…T[i+kLen] (i+kLen<=n)是否等于 s[i] 即可。

sol

分析对于每种可能的 Len (最多10种情况)

对于每个 i (1<=i<=Len):

对于每个可能的 s[i] （s[i]∈{A,C,G,T}):

我们都建立一个长度为 n 的树状数组，其中 第 j 位表示 T[j] 是否和 s[i] 相等。

且其中 j 只需要取{i, i+Len, i+2len,… i+kLen}

总共需预处理 4*(1+2+3+…10)=220个长度为 n 的树状数组。

sol

再分析对于一般询问 [L, R]，长度为 Len 的 s:
 只需要对 T[L,…R] 的前 Len 个字符考虑即可。

对于 T[L], 我们先找到 i (i=L%Len)。
然后在 （Len, i, s[i]）对应的树状数组上查询 [L, R] 上有几个 1 即可。

sol

```cpp
#define N 100010
int n,m,sum[4][11][11][N],has[128];
char t[N],s[20];
int lowbit(int x){
    return x&(-x);
}
void change(int c,int p,int v){
    for(int x=p;x<=n;x+=lowbit(x)){
        for(int j=1;j<=10;j++){
            sum[c][j][p%j][x]+=v;
        }
    }
}
int query(int c,int len,int p,int x){
    int ret=0;
    while(x){
        ret+=sum[c][len][p][x];
        x-=lowbit(x);
    }
    return ret;
}
```

```cpp
int main() {
    has['A']=0,has['T']=1,has['C']=2,has['G']=3;
    scanf("%s",t+1);
    n=strlen(t+1);
    for(int i=1;i<=n;i++) change(has[t[i]],i,1);
    int op,l,r,x;
    char c;
    m=read();
    while(m--){
        op=read();
        if(op==1){
            scanf("%d %c",&x,&c);
            change(has[t[x]],x,-1);
            change(has[c],x,1);
            t[x]=c;
        }else{
            scanf("%d %d %s",&l,&r,s);
            int len=strlen(s),ans=0;
            for(int i=0;i<len;i++){
                ans+=query(has[s[i]],len,(i+1)%len,r)
                    -query(has[s[i]],len,(i+1)%len,l-1);
            }
            printf("%d\n",ans);
        }
    }
    return 0;
}
```

例9 cf980e

一个 n 个节点的树，i 号节点权值是 2^i.

现在要求去掉 k 个点，使得保持树连通的情况下剩下的树点权之和最大。

n<=1e5.

sol

可以发现题目等价于要求剩下的点从大到小排序的字典序最大。

因此我们需要贪心的保留编号大的节点。

可以依次判断 n、n-1、n-2… 节点是否可以留下。

sol

很显然 n 号节点肯定可以留下。
所以我们可以以 n 号节点作为根。

考虑如果要保留 n-1 号节点，那就意味着 n-1 号节点到 n 号节点路上经过的所有点都要保留。

如果要保留 i 号节点，那么 i 号节点到 n 号节点路上所有点都要保留。

因此我们需要找到 i 号节点往上最少走几步能走到保留的节点，也即是保留 i 号点需要增加保留的节点数。

sol

又因为节点 k 被保留，那么 k 的父亲也一定被保留了。

我们可以用倍增的思想来存下往上走 2^k 步的点并计算出保留 i 号点会增加的点数。

这样如果计算出可以留下 i，那就从 i 暴力往上走去打上保留的标记。

因为每个点都只会保留一次，所以暴力的时间复杂度均摊为O(n)。

sol

我们也可以维护节点 i 到根节点这条路上几个点已经被保留了，记做 sum。

这样如果保留 i，我们可以用 i 的深度减去这个sum，就是要多保留的节点数。

每次保留 i 的时候，只需要对 i 的所有后代的 sum 加上增加的值就行。

可以用 dfs 序上的区间加和单点询问。

使用树状数组可以维护。时间复杂度O(nlogn)。

例10 bzoj3521

给定一个长度为 n 的 01 串。

你需要找出一个最长的连续子串 s, 使得不管从左往右还是从右往左取，都保证每时每刻已取出的 1 的个数不小于 0 的个数。

n<=1e6.

in: 010110

out: 4

sol

假设我们从 i 开始往右边取，最远可以取到 R[i];
往左边取最远可以取到 L[i].

对于一个子串 s[i…j] 如果合法，则一定满足:

1、j<=R[i]

2、i>=L[j]

在此基础上我们需要最大化 j-i+1.

sol

j<=R[i], i>=L[j]

这又像是二维数点问题。我们仍可以通过扫描线，排序一维方法，消除一维的限制，进而用数据结构去维护另一维。

比如从小到大枚举 i, 每次把所有满足 L[j]<=i 的 j 记录下来。（也可以从大到小枚举 j，每次把满足 R[i]>=j 的 i 记录下来）

然后我们查询小于等于 R[i] 的 j 的最大值。

用树状数组去维护前缀max即可。

sol

问题是怎么求出 L 和 R数组。

我们可以把 1 看做 0， 0 看做 -1。
计算出字符串的前缀和。

如果 [l, r] 可行，那么 min{sum[l···r]} >= sum[l-1].
也即是如果左端点是 l，我们要找到 r 满足对于任意 i<=r，使得 l..i 的区间和都大于0，也就是 sum[i]-sum[l-1]>=0.

sol

也就是给定一个 i，求出大于 i 的最小的 j 满足sum[j]<sum[i-1],
因为 sum 是渐变的，相邻的最多增加个 1 或者减少个1。
所以找到sum[i-1] = sum[j] +1.
找到 j 后，那么 R[i]=j-1.

我们可以从右往左去依次记录上一个出现同样值的位置last.
那么对于 i，我们只需要求 j= last[sum[i-1]-1]即可。

同理，可以求得L数组。
时间的复杂度为O(nlogn).

例11 cf220e

给一个长度为 n 的序列 A。

问有多少个组（l, r）(1<=l<r<=n) 满足

A[1…l, r…n] （l 到 r 之间的数删掉）的逆序对个数不出过 k.

n<=1e5,0<=k<=1e18

in:
3 1
1 3 2

out:
3

in:
5 2
1 3 2 1 7

out:
6

sol

可以发现，如果 (l, r) 可行，那么 (l, r+1) 一定可行。
因此我们可以枚举 l,只要算出了最小的 r，就可以把 n-r+1 加入答案。

进一步，发现 l, r 是单调的，即随着 l 右移，r 也会右移。
l 从1 移到 n，rr也会从 2 移到 n。
并且每次 l 右移一位，r 会右移若干位。

因此我们可以维护 l 和 r 2个指针移动，时间复杂度是线性的。

sol

初始状态 l=1, r=2。

我们可以建立 2 个树状数组 S 和 T。

S 维护 A[1…l], T维护 A[r…n]。

一开始先计算逆序对。

如果逆序对数>k, r 就需要右移。

下面讨论 l,r 右移的影响。

sol

r 右移:

S 中大于 A[r] 的数对逆序对的贡献会-1。

T 中小于 A[r] 的数对逆序对的贡献会-1。

一直右移直到逆序对数<=k。

这个时候求出了 l 对应的最小的 r, 把 n-r+1 加入答案。

然后 l 右移:

S 中大于 A[l] 的数对逆序对贡献会+1,

T 中小于 A[l] 的数对逆序对贡献会+1。

然后继续维护 r 右移。。。直到 l 移动到 n 或 A[1…l] 中逆序对已经>k。

核心代码

```cpp
#define N 100010
int n,cnt;
int a[N],bin[N],S[N],T[N];
ll k;

int lowbit(int x){
    return x&(-x);
}
void change(int sum[],int x,int v){
    while(x<=cnt){
        sum[x]+=v;
        x+=lowbit(x);
    }
}
int query(int sum[],int x){
    int ret=0;
    while(x){
        ret+=sum[x];
        x-=lowbit(x);
    }
    return ret;
}
```

```cpp
ll cur=0;
for(int i=n;i>=1;i--){
    change(T,a[i],1);
    cur+=query(T,a[i]-1);
}
change(T,a[1],-1);
change(S,a[1],1);
int l=1,r=2;
ll ans=0;
while(l<n){
    while((cur>k&&r+1<=n)|| r<=l){
        cur-= l-query(S,a[r]);
        cur-= query(T,a[r]-1);
        change(T,a[r],-1);
        r++;
    }
    if(cur>k) break;
    ans+=n-r+1;
    l++;
    change(S,a[l],1);
    cur+=l-query(S,a[l]);
    cur+=query(T,a[l]-1);
}
cout<<ans<<endl;
```

例12 cf383c

有一棵树，有2种操作：

1 x val： 在 x 号节点 +val, 然后 x 的每个儿子 –val， 然后给 x 的每个儿子的儿子 +val…直到没有儿子。

2 x： 查询 x 号节点的值。

n,m<=2e5

in:
```
5 5
1 2 1 1 2
1 2
1 3
2 4
2 5
1 2 3
1 1 2
2 1
2 2
2 4
```

out:
```
3
3
0
```

sol

我们想要直接进行 x 的后代的区间操作。

发现 dfs 序和 bfs 序都不满足要求。

因此我们还是按照树的深度分层。

我们发现点的深度的奇偶性是固定的。

因此我们可以用 2 个数据结构分别去维护奇数层的点和偶数层的点就行。

这样我们避免了 +val 还是 –val 的判断。

直接对 x 的后代 +val 或 –val 就行。

在维护奇数层的数据结构中偶数层的值是错的，但是我们不关心，同理偶数层也如此。

sol

因此采用 dfs 序把一个点的后代转化为区间即可。

目标是区间+和单点询问。
差分+树状数组来维护即可。
时间复杂度 $O((n+m)\log n)$。

例12进阶 cf396c

有一棵树，有2种操作：

1 x val k：在 x 号节点 +val, 然后 x 的每个儿子 +val-k，然后给 x 的每个儿子的儿子 +val-2k…直到没有儿子。

2 x：查询 x 号节点的值。

n,m<=3e5

in:
3
1 1
3
1 1 2 1
2 1
2 2

out:
2
1

sol

还是把树按照深度分层。

设修改的点 x 深度为 d[x].

对于 x 的后代 y，要加的值就是

val - (d[y] - d[x])*k

= (val+d[x]*k) - k*d[y]

注意第一项对所有的 y 是一个定值。

第二项对于所有的 y 只要维护 d[y] 的系数之和即可。

仍然用 dfs 序修改 x 的后代，需要2个树状数组，分别维护(val+d[x]*k) 和 -k 的 和。查询单点y时第一项直接加，第二项查询结果*d[y]

时间复杂度 O((n+m)logn)。