

树状数组算法讲义

本讲目标

理解树状数组中每个节点覆盖的区间；掌握单点修改、前缀查询和区间求和；能利用差分处理区间修改，并掌握双树状数组、逆序对与离线区间统计等常见应用。

目录

1 从前缀和到动态前缀和	2
2 lowbit 与节点覆盖范围	2
3 单点修改与区间求和	3
4 建树与手算路径	5
5 区间修改与单点查询	5
6 区间修改与区间求和	6
7 经典应用	7
8 树状数组上二进制倍增	10
9 模型对照与常见错误	11
10 本讲小结	11

1 从前缀和到动态前缀和

1.1 问题模型

给定数列 $a_1, a_2, \dots, a_n$ ，反复执行两种操作：

1. 把某个位置 a_x 增加 k ；
2. 询问区间 $[l, r]$ 的元素和。

静态前缀和能 $O(1)$ 回答区间和，但修改一个元素会影响它后面的所有前缀。树状数组在两种操作之间取得平衡：

方法	单点增加	区间求和
直接使用原数组	$O(1)$	$O(n)$
普通前缀和	$O(n)$	$O(1)$
树状数组	$O(\log n)$	$O(\log n)$

核心想法

不维护每一个完整前缀，而是把前缀拆成少量两两不相交的小区间。每个小区间的和存在一个树状数组节点中。

模板题 洛谷 P3374 【模板】树状数组 1

题目要求支持单点增加和区间求和，是树状数组的基本模型。

2 lowbit 与节点覆盖范围

2.1 lowbit 的定义

$$\text{lowbit}(x) = x \& (-x).$$

它取出 x 的二进制表示中最低位的 1 及其后面的 0。例如：

$$12 = (1100)_2, \text{quad} \text{lowbit}(12) = (0100)_2 = 4.$$

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

2.2 树状数组中的区间

树状数组的第 x 个节点记为 $c[x]$ ，它保存长度为 $\text{lowbit}(x)$ 的一段区间：

$$c[x] = \sum_{i=x-\text{lowbit}(x)+1}^x a_i.$$

x	$\text{lowbit}(x)$	$c[x]$ 覆盖区间	查询时的下一个位置
1	1	[1, 1]	0
2	2	[1, 2]	0
3	1	[3, 3]	2
4	4	[1, 4]	0
6	2	[5, 6]	4
7	1	[7, 7]	6
8	8	[1, 8]	0

例如查询前缀 $[1, 7]$ 时, 依次取出 $c[7], c[6], c[4]$, 对应区间 $[7, 7], [5, 6], [1, 4]$, 它们恰好拼成 $[1, 7]$ 。

两条移动规则

查询前缀时不断令 $x \leftarrow x - \text{lowbit}(x)$; 修改位置 x 时不断令 $x \leftarrow x + \text{lowbit}(x)$ 。前者向左拆分前缀, 后者找到所有包含 a_x 的节点。

3 单点修改与区间求和

3.1 修改与查询

将 a_x 增加 y 时, 需要修改所有覆盖位置 x 的节点。前缀和则沿着查询路径累加:

```

int lowbit(int x){
    return x&-x;
}
void add(int x,long long y){
    for(int i=x;i<=n;i+=lowbit(i)){
        c[i]+=y;
    }
}
long long ask(int x){
    long long s=0;
    for(int i=x;i;i-=lowbit(i)){
        s+=c[i];
    }
    return s;
}
long long sum(int l,int r){
    return ask(r)-ask(l-1);
}

```

修改和前缀查询的路径长度都不超过 $O(\log n)$, 因此区间求和也是 $O(\log n)$ 。

3.2 完整基础模板

```
#include<bits/stdc++.h>
#define N 500010
using namespace std;
int n,m;
long long c[N];
int lowbit(int x){
    return x&-x;
}
void add(int x,long long y){
    for(int i=x;i<=n;i+=lowbit(i)) c[i]+=y;
}
long long ask(int x){
    long long s=0;
    for(int i=x;i>=1;i-=lowbit(i)) s+=c[i];
    return s;
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    cin >> n >> m;
    for(int i=1;i<=n;i++){
        long long x;
        cin >> x;
        add(i,x);
    }
    while(m--){
        int op,x,y;
        cin >> op >> x >> y;
        if(op==1) add(x,y);
        else cout << ask(y)-ask(x-1) << "\n";
    }
    return 0;
}
```

下标必须从 1 开始

$\text{lowbit}(0) = 0$ 。如果在 `add` 中传入 $x = 0$ ，循环位置永远不会改变。因此树状数组通常使用 $1 \sim n$ 的下标。

4 建树与手算路径

4.1 两种建树方式

最容易的建树方法是对每个 a_i 执行一次 $\text{add}(i, a[i])$ ，复杂度为 $O(n \log n)$ 。

也可以线性建树。节点 i 的直接父节点是 $i + \text{lowbit}(i)$ ，将 $c[i]$ 累加到父节点即可：

```
for(int i=1; i<=n; i++){
    c[i] += a[i];
    int j = i + lowbit(i);
    if(j <= n) c[j] += c[i];
}
```

4.2 一个完整例子

设数列为

$$[2, 1, 3, 4, 2, 5, 1, 6].$$

建树后，前八个节点依次为

$$[2, 3, 3, 10, 2, 7, 1, 24].$$

查询前缀 $[1, 7]$ 经过 $7 \rightarrow 6 \rightarrow 4 \rightarrow 0$ ，答案为

$$c[7] + c[6] + c[4] = 1 + 7 + 10 = 18.$$

若令 a_3 增加 2，修改路径为 $3 \rightarrow 4 \rightarrow 8$ ，只有 $c[3], c[4], c[8]$ 会发生变化。

5 区间修改与单点查询

5.1 差分数组

定义差分数组

$$d_i = a_i - a_{i-1}, \quad a_0 = 0.$$

则原数组的第 x 项是差分数组的前缀和：

$$a_x = \sum_{i=1}^x d_i.$$

将区间 $[l, r]$ 每个元素增加 k ，只需执行

$$d_l += k, \quad d_{r+1} -= k.$$

因此可用树状数组维护差分数组。

```

for(int i=1;i<=n;i++){
    cin >> a[i];
    add(i,a[i]-a[i-1]);
}

// [l,r] 每个数增加 x
add(l,x);
add(r+1,-x);

// 查询 a[x]
cout << ask(x) << "\n";

```

模板题 洛谷 P3368 【模板】树状数组 2

题目要求支持区间增加和单点查询，直接在差分数组上使用树状数组。

右端点的下一个位置

当 $r = n$ 时，`add(r+1,-x)` 不会进入更新循环，可以直接调用。数组仍应留出少量额外空间。

6 区间修改与区间求和

6.1 双树状数组的推导

设 $d_i = a_i - a_{i-1}$ ，则原数组前 x 项的和为

$$\begin{aligned}
 \sum_{i=1}^x a_i &= \sum_{i=1}^x \sum_{j=1}^i d_j \\
 &= (x+1) \sum_{i=1}^x d_i - \sum_{i=1}^x i \cdot d_i \\
 &= x \sum_{i=1}^x d_i - \sum_{i=1}^x (i-1) d_i.
 \end{aligned}$$

用 $c[0]$ 维护 d_i ，用 $c[1]$ 维护 $(i-1)d_i$ ，就能在 $O(\log n)$ 时间内求出原数组的前缀和。

```

long long c[2][N];
void add(int k,int x,long long y){
    for(int i=x;i<=n;i+=lowbit(i)) c[k][i]+=y;
}
long long ask(int k,int x){
    long long s=0;
    for(int i=x;i;i-=lowbit(i)) s+=c[k][i];
    return s;
}
void change(int l,int r,long long x){
    add(0,l,x);
    add(0,r+1,-x);
    add(1,l,x*(l-1));
    add(1,r+1,-x*r);
}
long long pre(int x){
    return ask(0,x)*x-ask(1,x);
}
long long sum(int l,int r){
    return pre(r)-pre(l-1);
}

```

初始化时可读入原数组，计算 $d_i = a_i - a_{i-1}$ ，然后分别将 d_i 和 $(i-1)d_i$ 加入两棵树状数组。

区间操作的对应关系

单点修改、区间求和：一棵树状数组维护原数组。区间修改、单点查询：一棵树状数组维护差分。区间修改、区间求和：两棵树状数组维护差分的两类前缀。

7 经典应用

7.1 逆序对

逆序对是满足 $i < j$ 且 $a_i > a_j$ 的下标对。数值最大可达 10^9 时，先离散化到 $1 \sim t$ ，再从右向左扫描。扫到 a_i 时，树状数组中已经放入了它右边的元素， $\text{ask}(x-1)$ 就是右边比它小的元素数量。

```
for(int i=1;i<=n;i++) b[i]=a[i];
sort(b+1,b+n+1);
int t=unique(b+1,b+n+1)-b-1;

long long ans=0;
for(int i=n;i>=1;i--){
    int x=lower_bound(b+1,b+t+1,a[i])-b;
    ans+=ask(x-1);
    add(x,1);
}
cout << ans << "\n";
```

模板题 洛谷 P1908 逆序对

7.2 离线统计区间不同数的个数

将询问按右端点从小到大排序，树状数组中只保留每个数值最后一次出现的位置。当扫描到位置 i 时：

1. 若 a_i 之前出现在位置 p ，将 p 处减 1；
2. 在位置 i 处加 1；
3. 所有右端点为 i 的询问都可以回答。


```
struct Q{
    int l,r,id;
}q[N];
bool cmp(Q x,Q y){
    return x.r<y.r;
}

sort(q+1,q+m+1,cmp);
int now=0;
for(int i=1;i<=m;i++){
    while(now<q[i].r){
        now++;
        if(p[a[now]]) add(p[a[now]],-1);
        add(now,1);
        p[a[now]]=now;
    }
    ans[q[i].id]=ask(q[i].r)-ask(q[i].l-1);
}
```

模板题 [洛谷 P1972 \[SDOI2009\] HH 的项链](#)

7.3 三元上升子序列

对于每个中间位置 i ，分别统计左边比 a_i 小的数量 L_i 和右边比 a_i 大的数量 R_i ，答案为

$$\sum_{i=1}^n L_i R_i.$$

离散化后分别从左向右、从右向左扫描即可。

```

memset(c,0,sizeof(c));
for(int i=1;i<=n;i++){
    l[i]=ask(a[i]-1);
    add(a[i],1);
}
memset(c,0,sizeof(c));
for(int i=n;i>=1;i--){
    r[i]=ask(t)-ask(a[i]);
    add(a[i],1);
}
long long ans=0;
for(int i=1;i<=n;i++) ans+=l[i]*r[i];

```

模板题 洛谷 P1637 三元上升子序列

8 树状数组上二进制倍增

当 c 维护非负频次时，前缀和单调不减。可以在树状数组上直接找到第一个前缀和不少于 k 的位置，也就是动态第 k 小所在位置。

```

int kth(int k){
    int x=0,p=1;
    while((p<<1)<=n) p<<=1;
    for(;p;p>>=1){
        int y=x+p;
        if(y<=n&&c[y]<k){
            x=y;
            k-=c[y];
        }
    }
    return x+1;
}

```

过程中 x 始终是“前缀和仍小于 k ”的最右位置，每次尝试向右跨越 p 个位置。复杂度为 $O(\log n)$ ，比“二分答案 + 每次查询前缀和”的 $O(\log^2 n)$ 更快。

前缀和必须单调

只有当每个位置的频次都非负时，才能用这种方法查找第 k 小。如果树状数组中维护的数可以为负，前缀和不一定单调。

9 模型对照与常见错误

操作模型	维护的内容	查询方法
单点修改、区间求和	原数组	$ask(r)-ask(l-1)$
区间修改、单点查询	差分数组	$ask(x)$
区间修改、区间求和	d_i 与 $(i-1)d_i$	两个前缀组合
逆序对	已扫描数值的频次	查询更小值的数量
离线区间种类数	每个值的最后位置	查询区间标记和
动态第 k 小	非负频次	树上倍增定位

做题时的五步检查

先判断修改和查询各是单点还是区间；再确定维护原数组还是差分；检查下标是否从 1 开始；区间答案使用右前缀减左前缀；最后判断累加结果是否需要 `long long`。

- 无法修改位置 0：否则 `add` 会死循环。
- 区间左端点少减 1：区间 $[l, r]$ 应为 $ask(r)-ask(l-1)$ 。
- 离散化后查询严格大小：求比 a_i 小的数量要查 $ask(x-1)$ ，不能把相等元素算入。
- 多测数据要清空：树状数组通常是全局数组，下一组数据不能沿用上一组的频次。

10 本讲小结

树状数组用 `lowbit` 将前缀分成少量区间，使单点修改和前缀查询都能在 $O(\log n)$ 时间内完成。区间操作的关键是差分：一棵树状数组可实现区间修改与单点查询，两棵树状数组可进一步实现区间求和。在逆序对、离线查询和动态第 k 小中，树状数组维护的本质仍然是动态前缀统计。