树状数组

Binary Indexed Tree

河南省实验中学信息技术组

2026 年 04 月 23 日

知识回顾

- 前缀和
- 差分
- 位运算

【引例】数列操作 A

【题目描述】
给定 $n(1 \le n \le 10^5)$ 个整数序列，定义如下两种操作：

- 第一类操作形如 `c x y`，表示序列中第 x 个数加上 y；
- 第二类操作形如 `q l r`，表示查询序列下标区间 $[l, r]$ 的元素和。

现在有 $m(1 \le m \le 10^5)$ 条指令，对于每条查询指令，输出一个整数表示答案。

【输入格式】
第 1 行两个整数 $n, m(n, m \le 10^5)$。
第 2 行 n 个整数表示整数序列。
接下来 m 行，每行一个操作，操作类型见题面。

【输出格式】
对于每个查询操作，输出一个整数表示答案。

【样例输入】

```
4 3
1 4 2 3
q 1 3
c 2 50
q 2 3
```

【样例输出】

```
7
56
```

【引例】数列操作 A

- 方法 1：
  - 使用一维数组 a 存储序列。
  - 对于第一类操作，执行 $a[i]+=y$ 即可；
  - 对于第二类操作，循环累加区间 $[l,r]$ 的元素和。
  - 第一种操作时间复杂度 O(1)，第二种操作时间复杂度 O(N)。
- 方法 2：
  - 使用一维数组 a 存储序列。
  - 定义前缀和数组 s，其中 $s_1=a_1, s_i=s_{i-1}+a_i(2\le i\le n)$。
  - 对于第一类操作，循环更新前缀和；
  - 对于第二类操作，输出区间和 s_r-s_{l-1}。
  - 第一种操作时间复杂度 O(N)，第二种操作时间复杂度 O(1)。

思考

- 普通数组 a 能快速修改，但不能快速查询前缀和；前缀和数组 s 能快速查询前缀和和区间和，但不能快速修改。
- 增加数和查询两种操作的效率不可能都是 $O(1)$，所以两种操作都让步，使用一个部分和数组 f，使得增加数和查询的平均时间复杂度较低。
- 其中一种部分和数组 f 可以使两种操作的时间复杂度均为 $O(\log N)$，这就是树状数组。

树状数组

给定一个数组 a，建立一个数组 f，其中 $f[x]$ 保存数组 a 的区间

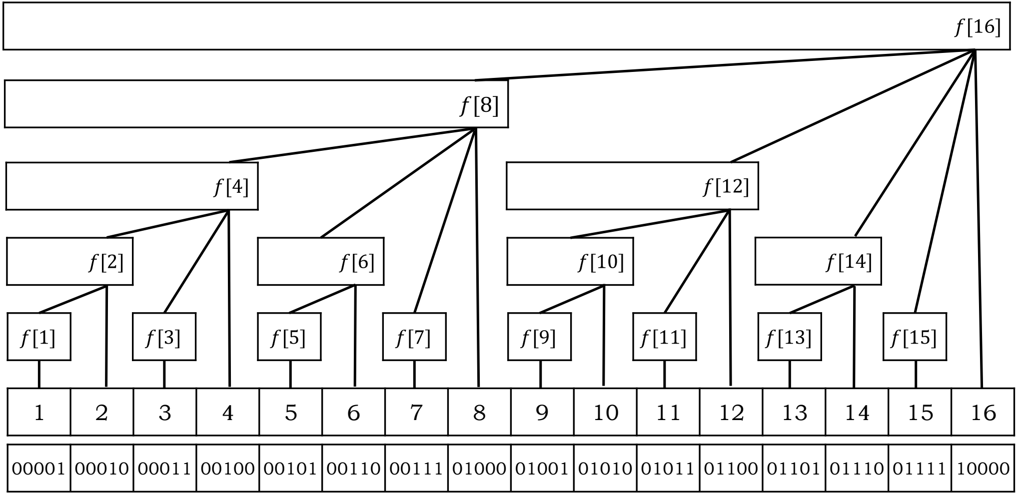
$$[x - lowbit(x) + 1, x]$$

中所有数的和，其中 $lowbit(x) = x\&(-x)$，即为 x 二进制下最低位的 1 所代表的 2 的次幂。

树状数组正如其名，可以看做是一个树形结构，树中最下边一行是 n 个叶子结点，代表数值 $a[1 \sim n]$。该结构满足如下性质：

- 每个内部结点 $f[x]$ 保存以它为根的子树中所有叶子结点的和。
- 每个内部结点 $f[x]$ 的子结点个数等于 $lowbit(x)$ 的位数。
- 除树根外，每个内部结点 $f[x]$ 的父结点是 $f[x + lowbit(x)]$。
- 树的深度为 $\mathrm{O}(\log N)$。

树状数组


$f[16]$
$f[8]$
$f[4]$
$f[12]$
$f[2]$
$f[6]$
$f[10]$
$f[14]$
$f[1]$
$f[3]$
$f[5]$
$f[7]$
$f[9]$
$f[11]$
$f[13]$
$f[15]$
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
00001 00010 00011 00100 00101 00110 00111 01000 01001 01010 01011 01100 01101 01110 01111 10000

查询前缀和

- 根据定义，$f[x]$ 表示 a 中区间 $[x - lowbit(x) + 1, x]$ 中所有数的和。那么，$x - lowbit(x)$ 表示下一段需要累加的部分区间的末尾，依此类推

$$s[x] = f[x] + s[x - lowbit(x)], x - lowbit(x) \geq 1$$

- 根据上述公式，我们可以写出如下代码

```
int ask(int x)
{
    int ans = 0;
    for(; x; x -= x & (-x)) ans += f[x];
    return ans;
}
```

- 时间复杂度：$O(\log N)$。
- 如果需要查询区间 $[l, r]$ 中所有数的和，只需计算 $ask(r) - ask(l - 1)$ 即可。

查询前缀和

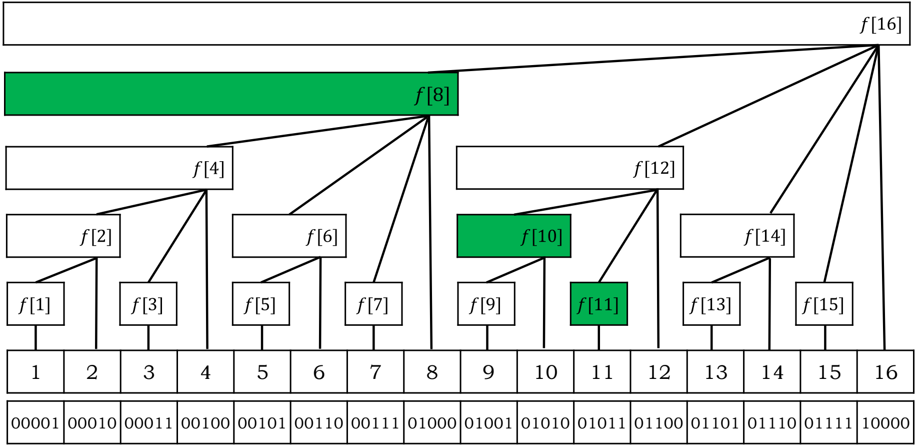


图: 查询前缀和 s_{11}

单点修改

- 在位置 x 增加常数 y 后会对树状数组中那些数值产生影响?
- 根据树状数组的结构和性质，只有结点 $f[x]$ 及其所有的祖先节点保存的部分区间和包含 $a[x]$。
  - 首先，影响 $f[x]+=y$;
  - 下一个影响的是 $f[x+lowbit(x)]+=y$，例如修改位置 1 影响的位置有 $1 \to 1+lowbit(1)=2 \to 2+lowbit(2)=4 \to \cdots\cdots$。
- 对所有受到影响的位置都进行更新，直到位置越界

```
// 位置 x 上的数值增加 y
void add(int x, int y)
{
    for(; x <= n; x += x & (-x)) f[x] += y;
}
```

- 时间复杂度：$O(\log N)$。

单点修改

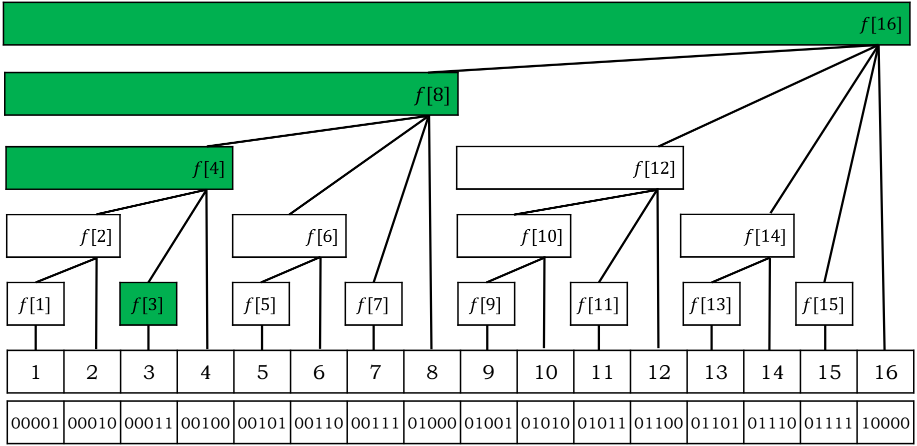


图: 将 $a[3]$ 增加一个值

单点修改

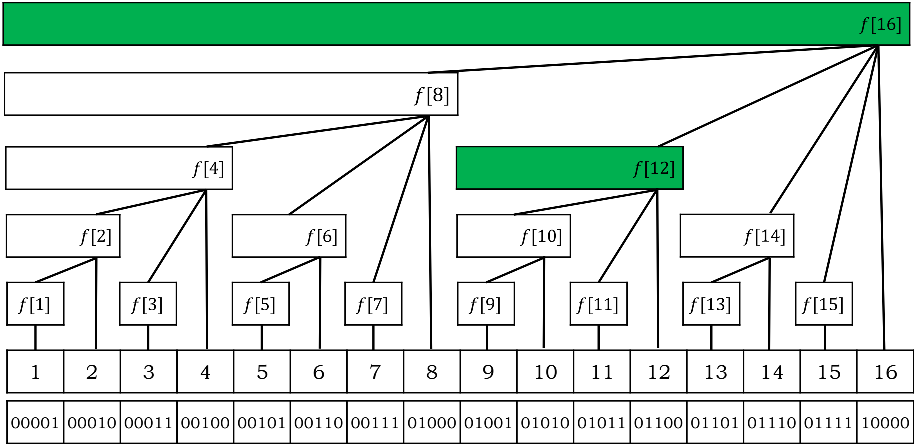


图: 将 $a[12]$ 增加一个值

初始化

- 方案 1
  - 建立一个全 0 的数组 f；
  - 对于每个位置 x，执行 $add(x, a[x])$ 就完成了构造树状数组。
  - 时间复杂度：$O(N \log N)$。
- 方案 2
  - 根据原数组 a 求出前缀和 s；
  - 对于每个位置 x，利用定义求出 $f[x] = s[x] - s[x - lowbit(x)]$。
  - 时间复杂度：$O(N)$。

单点修改, 区间查询

给定 $n(1 \le n \le 10^5)$ 个整数序列，定义如下两种操作：

- 第一类操作形如 c x y，表示序列中第 x 个数加上 y；
- 第二类操作形如 q l r，表示查询区间 $[l, r]$ 的元素和。

现在有 $m(1 \le m \le 10^5)$ 条指令，对于每条查询指令，输出一个整数表示答案。

单点修改, 区间查询

- 定义 f 为序列 a 的树状数组。
- 位置 x 上数值增加 y

```
add(x, y);
a[x] += y;
```

- 查询区间 $[l, r]$ 中数值之和：$s[r] - s[l-1]$

```
int ans = ask(r) - ask(l - 1);
```

区间修改, 单点查询

给定 $n(1 \leq n \leq 10^5)$ 个整数序列，定义如下两种操作：

- 第一类操作形如 c l r y，表示序列中第 $l \sim r$ 个数都加上 y；
- 第二类操作形如 q x，表示查询序列第 x 个数的值。

现在有 $m(1 \leq m \leq 10^5)$ 条指令，对于每条查询指令，输出一个整数表示答案。

区间修改, 单点查询

- 差分序列的前缀和是原序列，前缀和序列的差分是原序列。
- 定义数组 b 为数组 a 的变化量的差分，初始时全为 0。
- 对于区间修改操作，我们将其转化为：$b[l]$ 加上 y，$b[r+1]$ 减去 y。
- 那么对于数组 b 来说，它的前缀和 s 刚好反映了原序列 a 的变化，其中 $s[x]$ 表示到目前为止 $a[x]$ 增加的值的总和。
- 定义 f 为数组 b 的树状数组，则

```
// 单点查询
int ans = a[x] + ask(x);
// 区间修改
add(l, y);
add(r + 1, -y);
```

区间修改, 区间查询[1]

给定 $n(1 \le n \le 10^5)$ 个整数序列，定义如下两种操作：

- 第一类操作形如 c l r y，表示序列中第 $l \sim r$ 个数都加上 y；
- 第二类操作形如 q l r，表示查询区间 $[l, r]$ 的元素和。

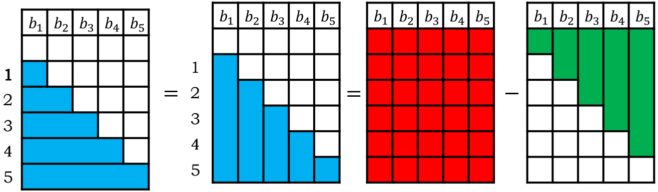
现在有 $m(1 \le m \le 10^5)$ 条指令，对于每条查询指令，输出一个整数表示答案。

[1]树状数组实现较难理解，建议使用线段树。

区间修改, 区间查询

- 在“区间修改, 单点查询”中，我们只能快速查询某一位置的值，无法查询区间和。因为只维护了某一位置的增量，那如果我们求出各个位置增量的的区间和 (也即前缀和)，然后加上 a 的区间和 (不变) 即可。
- 在“区间修改, 单点查询”中，我们知道差分数组 b 的前缀和 $\sum_{j=1}^{i} b[j]$ 表示经过一些指令后 $a[i]$ 增加的值。那么序列 a 的前缀和整体增加的值为[2]：

$$\sum_{i=1}^{x}\sum_{j=1}^{i} b[j] = \sum_{i=1}^{x}(x-i+1)\times b[i] = (x+1)\sum_{i=1}^{x} b[i] - \sum_{i=1}^{x} i\times b[i]$$



[2]Tips：分离包含多个变量的项，使公式中不同变量之间相互独立。

区间修改, 区间查询

- 增加一个树状数组 g，用于维护 $i \times b[i]$。
- 对于修改指令 c l r y:
  - 在树状数组 f 中，将位置 l 上的数加 y，位置 $r+1$ 上的数减 y;
  - 在树状数组 g 中，将位置 l 上的数加 $l*y$，位置 $r+1$ 上的数减 $(r+1)*y$;
- 对于查询指令 q l r:

$$
\begin{aligned}
ans = sum(r) - sum(l-1) = (s[r] + (r+1) \times ask(f,r) - ask(g,r)) \\
- (s[l-1] + (l-1+1) \times ask(f,l-1) - ask(g,l-1))
\end{aligned}
$$

```cpp
// 修改指令
add(f, l, y), add(f, r + 1, -y);
add(g, l, l * y), add(g, r + 1, -(r + 1) * y);
// 查询指令
int ans = s[r] + (r + 1) * ask(f, r) - ask(g, r);
ans -= s[l - 1] + l * ask(f, l - 1) - ask(g, l - 1)
```

逆序对

【逆序对】

设 a 为一个有 n 个数字的有序集 $(n > 1)$，其中所有数字各不相同。如果存在正整数 i, j 使得 $1 \leq i < j \leq n$ 而且 $a[i] > a[j]$，则 $(a[i], a[j])$ 这个对称为 a 的一个逆序对。

例如：数组 $\{3, 1, 4, 5, 2\}$ 的逆序对有 $(3, 1), (3, 2), (4, 2), (5, 2)$，共 4 个。

那么，对于一个数组，如何求出这个数组的逆序对数呢?

【输入格式】

第一行一个整数 $n(n \leq 10^5)$。

接下来一行 n 个整数，表示序列。输入保证 $1 \leq a[i] \leq 10^6$。

【输出格式】

一个整数，表示序列的逆序对个数。

【样例输入】

```
10
9 1 4 2 6 7 5 8 3 10
```

【样例输出】

```
16
```

逆序对

- 对于序列 a，如果用 $t[a[i]]$ 保存数值 $a[i]$ 出现的次数，那么数组 t 在 $[l, r]$ 上的区间和就表示序列 a 在 $[l, r]$ 内的数的个数。
- 按照上述思想，利用树状数组求出一个序列的逆序对数：
  - 在序列 a 的数值范围上建立树状数组 f，初始化为 0；
  - 倒序扫描给定的序列 a，对于每个数 $a[i]$：
    - 在树状数组中查询前缀和 $[1, a[i]-1]$，累加到答案中；
    - 在树状数组中执行单点增加操作，即把位置 $a[i]$ 上的数值加 1，表示数值 $a[i]$ 又出现了 1 次。
- 因为是倒序扫描，所以前缀和 $[1, a[i]-1]$ 中存放的就是在序列 a 中在 $a[i]$ 后边但是比 $a[i]$ 小的数。
- 这种存储数据出现次数或其他统计信息的树状数组被称为权值树状数组。

逆序对

```cpp
long long ans = 0;
for(int i = n; i >= 1; --i)
{
    ans += ask(a[i] - 1);
    add(a[i], 1);
}
cout << ans;
```

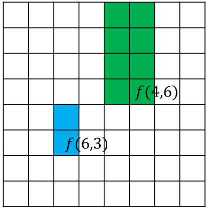
- 时间复杂度：$\mathrm{O}(N \log M)$，M 为数值范围大小。
- 当数值范围较大时，时间复杂度较高，该算法效率降低。
- 当然可以先进行离散化，但是离散化必须要使用排序算法，那就不如直接用归并排序来求解逆序对。
- 树状数组一般主要针对全排列或者数据范围小的数据，一般需要维护逆序对变化。

二维树状数组

- 类比二维前缀和数组，可以定义二维树状数组 f，其中

$$f[x][y] = \sum_{i=x-lowbit(x)+1}^{x} \sum_{j=y-lowbit(y)+1}^{y} a[i][j]$$

表示矩形区域的元素之和。



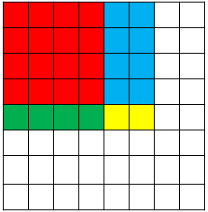
- 高维树状数组定义可以类比二维树状数组。

二维树状数组

- 类比一维树状数组，二维树状数组的前缀和查询操作如下：

```
int ask(int x, int y)
{
    int ans = 0;
    for(int i = x; i; i -= i & -i)
        for(int j = y; j; j -= j & -j)
            ans += f[i][j];
    return ans;
}
```

- 例如：$s[5][6] = f[5][6] + f[5][4] + f[4][6] + f[4][4]$。



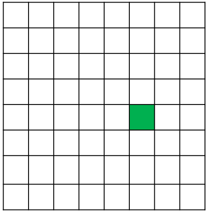
- 时间复杂度：$\mathrm{O}(\log^2 N)$。

二维树状数组

- 类比一维树状数组，二维数组的单点修改操作如下：

```
void add(int x, int y, int z)
{
    for(int i = x; i <= m; i += i & -i)
        for(int j = y; j <= n; j += j & -j)
            f[i][j] += z;
}
```

- 例如：修改 $a[5][6]$ 会影响 $f[5][6], f[5][8], f[6][6], f[6][8], f[8][6], f[8][8]$。



- 时间复杂度：$O(\log^2 N)$。

练习

- 数列操作 A(COGS 264)
- 数列操作 B(COGS 1316)
- 数列操作 C(COGS 1317)
- 全排列逆序对个数 (COGS 1750)
- 会议座位 (洛谷 P5149)
- 公路交叉 (COGS 1974)
- 奶牛排队 (COGS 1975)
- 人工湖泊 (COGS 1697)
- 移动电话 [IOI 2001](COGS 1532)