树状数组

李淳风

长郡中学

2024 年 6 月 15 日

引入

树状数组是一种支持**单点修改**和**区间查询**的，代码量小的数据结构。

例如，已知一个数列 a，单点修改就是每次给定 x, y，把 a_x 增加 y。
区间查询就是给定 l, r，查询 $a_l + a_{l+1} + \cdots + a_r$ 的值。
类似的，区间增加就是给定 l, r, y，把 $a_l, a_{l+1}, \cdots, a_r$ 都增加 y。
单点查询就是给定 x，查询 a_x 的值。
如果我们想使用树状数组进行区间增加和区间查询，也可以通过差分数组和辅助数组的帮助来完成。

引入

树状数组是一种支持**单点修改**和**区间查询**的，代码量小的数据结构。

例如，已知一个数列 a，单点修改就是每次给定 x, y，把 a_x 增加 y。
区间查询就是给定 l, r，查询 $a_l + a_{l+1} + \cdots + a_r$ 的值。
类似的，区间增加就是给定 l, r, y，把 $a_l, a_{l+1}, \cdots, a_r$ 都增加 y。
单点查询就是给定 x，查询 a_x 的值。
如果我们想使用树状数组进行区间增加和区间查询，也可以通过差分数组和辅助数组的帮助来完成。

普通树状数组的维护的信息以及运算需要满足**结合律**以及**可差分**，如加法，乘法，异或等。

- 结合律：$(a \circ b) \circ c = a \circ (b \circ c)$，其中 $\circ$ 是一个二元运算符。
- 可差分：具有逆运算，即若已知 $a \circ b$ 和 a，可以求出 b。

需要注意的是，模意义下的乘法若要可差分，需要保证每个数都存在逆元。$\gcd, \max$ 这类信息不可差分，所以不能用普通的树状数组处理。

事实上，所有树状数组能解决的问题，都可以使用线段树来解决。但是树状数组代码远远短于线段树，而且时间效率常数也更小。

原理

任意一个正整数 x，都可以被二进制分解为下列形式：

$$x = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_m}$$

同时满足 $i_1 > i_2 > \cdots > i_m$。
因此我们可以把区间 $[1, x]$ 分成 $O(\log x)$ 个小区间：

- 长度为 2^{i_1} 的小区间 $[1, 2^{i_1}]$
- 长度为 2^{i_2} 的小区间 $[2^{i_1} + 1, 2^{i_1} + 2^{i_2}]$

  $\cdots$
- 长度为 2^{i_m} 的小区间 $[2^{i_1} + 2^{i_2} + \cdots + 2^{i_{m-1}} + 1, 2^{i_1} + 2^{i_2} + \cdots + 2^{i_m}]$

这些小区间都有一个共同的特点，若区间右端点为 R，则区间长度为 R 的在二进制分解下最小的 2 的次幂，即 $lowbit(R)$。

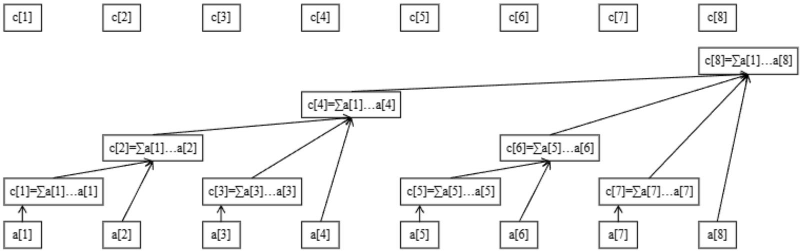
原理

例如，区间 $[1,7]$ 可以被分为 $[1,4],[5,6],[7,7]$ 三个区间，长度分别为 $lowbit(4)=4, lowbit(6)=2, lowbit(7)=1$。
对于 $lowbit$ 这个运算，相信大家都不陌生了。$lowbit(x)=x\&(-x)$，表示 x 最低的非零的二进制位表示的数。
因此，给定一个正整数 x，下列代码可以计算出区间 $[1,x]$ 分成的 $O(\log x)$ 个小区间：

```
while(x>0){
  printf("[%d,%d]\n",x-(x&-x)+1,x);
  x-=x&-x;
}
```

原理

对于需要维护的序列 a，我们建立一个数组 c，其中 c_x 保存序列 a 的区间 $[x - lowbit(x) + 1, x]$ 中所有数的和，即 $c_x = \sum_{i=x-lowbit(x)+1}^{x} a_i$。



树状数组的结构如图所示。它的一些性质如下：

- 每个节点 c_x 保存以它为根节点的子树中所有叶子节点的和；
- c_x 的父亲节点是 $c_{x+lowbit(x)}$；
- c_x 的子节点个数等于 $lowbit(x)$ 的位数，为 $c_{x-2^i}(2^i < lowbit(x))$；
- 树的深度为 $O(\log N)$。

基本操作

树状数组支持的基本操作有两个。第一个操作是查询前缀和，即序列 a 中区间 $[1, x]$ 的和。按照我们之前提到的方法，应该求出 x 的二进制表示中每个等于 1 的位，把 $[1, x]$ 分为 $O(\log N)$ 个小区间，而每个小区间的区间和都已经保存在了数组 c 中。

基本操作

树状数组支持的基本操作有两个。第一个操作是查询前缀和，即序列 a 中区间 $[1,x]$ 的和。按照我们之前提到的方法，应该求出 x 的二进制表示中每个等于 1 的位，把 $[1,x]$ 分为 $O(\log N)$ 个小区间，而每个小区间的区间和都已经保存在了数组 c 中。

所以，把之前用于划分区间的代码稍加改动，即可在 $O(\log N)$ 的时间内查询前缀和：

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

基本操作

树状数组支持的基本操作有两个。第一个操作是查询前缀和，即序列 a 中区间 $[1, x]$ 的和。按照我们之前提到的方法，应该求出 x 的二进制表示中每个等于 1 的位，把 $[1, x]$ 分为 $O(\log N)$ 个小区间，而每个小区间的区间和都已经保存在了数组 c 中。

所以，把之前用于划分区间的代码稍加改动，即可在 $O(\log N)$ 的时间内查询前缀和：

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

当然，若要查询序列 a 中 $[l, r]$ 的区间和，只需转换成两个前缀和的差，计算 $ask(r) - ask(l - 1)$ 即可。

基本操作

树状数组支持的第二个基本操作是单点增加，即给定 x, y，给 a_x 加上 y。由于我们还要维护 c 数组，我们接着来考虑 c 中的哪些位置会随着 a_x 改变而改变。

基本操作

树状数组支持的第二个基本操作是单点增加，即给定 x, y，给 a_x 加上 y。由于我们还要维护 c 数组，我们接着来考虑 c 中的哪些位置会随着 a_x 改变而改变。

我们在之前的性质中提到过，每个节点 c_x 保存以它为根节点的子树中所有叶子节点的和。因此只有节点 c_x 以及它的所有祖先节点保存的“区间和”包括了 a_x。而这样的祖先至多只有 $\log N$ 个，我们逐一对它们的 c 值进行更新即可。

基本操作

树状数组支持的第二个基本操作是单点增加，即给定 x, y，给 a_x 加上 y。由于我们还要维护 c 数组，我们接着来考虑 c 中的哪些位置会随着 a_x 改变而改变。
我们在之前的性质中提到过，每个节点 c_x 保存以它为根节点的子树中所有叶子节点的和。因此只有节点 c_x 以及它的所有祖先节点保存的“区间和”包括了 a_x。而这样的祖先至多只有 $\log N$ 个，我们逐一对它们的 c 值进行更新即可。

```
void add(int x,int y){
  for(;x<=N;x+=x&-x) c[x]+=y;
}
```

基本操作

树状数组支持的第二个基本操作是单点增加，即给定 x, y，给 a_x 加上 y。由于我们还要维护 c 数组，我们接着来考虑 c 中的哪些位置会随着 a_x 改变而改变。
我们在之前的性质中提到过，每个节点 c_x 保存以它为根节点的子树中所有叶子节点的和。因此只有节点 c_x 以及它的所有祖先节点保存的“区间和”包括了 a_x。而这样的祖先至多只有 $\log N$ 个，我们逐一对它们的 c 值进行更新即可。

```
void add(int x,int y){
  for(;x<=N;x+=x&-x) c[x]+=y;
}
```

在执行所有操作之前，a 数组通常有着初值，所以我们需要对树状数组进行初始化。
通常而言，直接建立一个全为 0 的数组 c，然后对于每个位置 x 执行一次 $add(x, a[x])$，就能用 $O(N\log N)$ 的复杂度完成初始化。
而更加高效的初始化方法是，对于每个节点，借助 $lowbit$ 运算扫描它的子节点并求和。这样树状数组中的每条边只会被遍历一次，时间复杂度是 $O(N)$。

逆序对

对于一个序列 a，若 $i < j$ 且 $a_i > a_j$，则称 a_i 与 a_j 构成逆序对。我们之前已经了解了归并排序求逆序对的解法，现在来尝试使用树状数组求解。

我们考虑枚举 a_i，判断有多少个 a_j 能够与它构成逆序对。不难发现，如果我们已经保证了 $i < j$ 的话，问题就变成了求有多少个 $a_j < a_i$。如果我们用 t_x 表示 x 这个值出现的次数，我们要求的就是 t 数组中 $[1, a_i - 1]$ 的区间和。

逆序对

对于一个序列 a，若 $i<j$ 且 $a_i>a_j$，则称 a_i 与 a_j 构成逆序对。我们之前已经了解了归并排序求逆序对的解法，现在来尝试使用树状数组求解。

我们考虑枚举 a_i，判断有多少个 a_j 能够与它构成逆序对。不难发现，如果我们已经保证了 $i<j$ 的话，问题就变成了求有多少个 $a_j<a_i$。如果我们用 t_x 表示 x 这个值出现的次数，我们要求的就是 t 数组中 $[1,a_i-1]$ 的区间和。

因此我们可以使用树状数组来维护 t 数组，具体做法如下：

- 在序列 a 的数值范围上建立树状数组；
- 倒序扫描给定的 a，对于每个数 a_i，在树状数组中查询前缀和 $[1,a_i-1]$，累加到答案之中；再执行“单点修改”操作，把位置 a_i 上的数加一（即 $t[a[i]]++$），表示 a_i 又出现了一次；
- ans 即为所求。

对于 $i<j$ 与 $a_i>a_j$ 这两个限制条件，我们先通过倒序扫描保证了 $i<j$，再使用树状数组来求这种情况下 $a_i>a_j$ 的数的个数。时间复杂度位 $O(N\log N)$，值域较大时需要先进行离散化。

例题

楼兰图腾

平面上有 N 个点，每个点的横、纵坐标范围都是 1 到 N，任意两个点的横、纵坐标都不相同。
若三个点 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ 满足 $x_1 < x_2 < x_3$，$y_1 > y_2$ 并且 $y_3 > y_2$，则称这三个点构成"v" 字图腾。
若三个点 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ 满足 $x_1 < x_2 < x_3$，$y_1 < y_2$ 并且 $y_3 < y_2$，则称这三个点构成"⌢" 字图腾。
求平面上两种图腾的个数。$1 \le N \le 2 \times 10^5$。

例题

楼兰图腾

平面上有 N 个点，每个点的横、纵坐标范围都是 1 到 N，任意两个点的横、纵坐标都不相同。
若三个点 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ 满足 $x_1 < x_2 < x_3$， $y_1 > y_2$ 并且 $y_3 > y_2$， 则称这三个点构成"v" 字图腾。
若三个点 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ 满足 $x_1 < x_2 < x_3$， $y_1 < y_2$ 并且 $y_3 < y_2$， 则称这三个点构成"^" 字图腾。
求平面上两种图腾的个数。$1 \leq N \leq 2 \times 10^5$。

题目的第一句话告诉我们，如果我们把这 N 个点按照横坐标排序，那么它们的纵坐标是 1 到 N 的一个排列。我们把这个排列记为 a。
现在我们来考虑求"v" 字图腾的数量。因为要求 $y_1 > y_2$ 且 $y_3 > y_2$，所以我们可以对于每一个 a_i，求出在它左边且比它大的 a 的个数 $left_i$，在它右边且比它大的 a 的个数 $right_i$，那么答案就是 $\sum_{i=1}^{N} left_i * right_i$。
$left_i$ 与 $right_i$ 的求法与逆序对的求法一样。第二种图腾的求法同理。

树状数组的拓展

A Tiny Problem with Integers

给定长度为 $N(N \leq 10^5)$ 的数列 A，然后输入 $Q(Q \leq 10^5)$ 行操作指令。
第一类指令形如"C l r d"，表示把数列中第 l ~ r 个数都加 d。
第二类指令形如"Q x"，表示询问数列中第 x 个数的值。

本题的指令有“区间增加”与“单点查询”，但是树状数组只支持“单点修改”，需要做出一些转化来解决该问题。
在思考这个问题之前，我们可以先来思考一个简化版。也就是所有第二类指令之后都没有第一类指令。

A Tiny Problem with Integers

这个简化版的问题由于是先修改，最后再询问，我们可以使用差分数组来解决。
我们考虑新建一个 a 数组的差分数组 b，满足 $b_1 = a_1, b_x = a_x - a_{x-1}(x > 1)$，也就是 a 是 b 的前缀和，$a_x = \sum_{i=1}^{x} b_i$。
对于每次区间 $[l, r]$ 的加法， a 数组在 $[l, r]$ 内任意相邻的数的差没有变化，那么对于 b 数组来说，就只有 $b_l+ = d$， $b_{r+1}- = d$ 这两项变化。
因此我们可以在 b 数组中模拟区间加法的操作，最后再通过 b 数组还原出 a 数组即可。

A Tiny Problem with Integers

这个简化版的问题由于是先修改，最后再询问，我们可以使用差分数组来解决。
我们考虑新建一个 a 数组的差分数组 b，满足 $b_1 = a_1, b_x = a_x - a_{x-1}(x > 1)$，也就是 a 是 b 的前缀和，$a_x = \sum_{i=1}^{x} b_i$。
对于每次区间 $[l, r]$ 的加法，a 数组在 $[l, r]$ 内任意相邻的数的差没有变化，那么对于 b 数组来说，就只有 $b_l += d$，$b_{r+1} -= d$ 这两项变化。
因此我们可以在 b 数组中模拟区间加法的操作，最后再通过 b 数组还原出 a 数组即可。
于是，如果我们想要随时查询的话，对 b 数组建立树状数组即可。原本的“区间增加”就变成了两次“单点修改”，原本的“单点查询”就变成了“区间查询”。

树状数组的拓展

A Simple Problem with Integers

给定长度为 $N(N \leq 10^5)$ 的数列 A，然后输入 $Q(Q \leq 10^5)$ 行操作指令。
第一类指令形如"C l r d"，表示把数列中第 l ~ r 个数都加 d。
第二类指令形如"Q l r"，表示询问数列中第 l ~ r 个数的和。

在上一道题中，我们用树状数组维护了一个差分数组 b，对于每条指令"C l r d"，我们把它变成了两次单点修改操作。
而我们知道 $a_x = \sum_{i=1}^{x} b_i$，因此有：

$$\sum_{i=1}^{x} a_i = \sum_{i=1}^{x} \sum_{j=1}^{i} b_j = \sum_{i=1}^{x} (x - i + 1) b_i$$

也就是：

$$\sum_{i=1}^{x} a_i = (x+1) \sum_{i=1}^{x} b_i - \sum_{i=1}^{x} (b_i \times i)$$

A Simple Problem with Integers

因此，在本题中，除了一个维护 b_i 的树状数组 c_0 之外，我们还需要一个维护 $b_i \times i$ 的树状数组 c_1。

因此，在进行操作"C l r d" 时，我们需要用树状数组 c_0 维护，给 l 位置加上 d，给 $r+1$ 位置减去 d；用树状数组 c_1 维护，给 l 位置加上 $l*d$，给 $r+1$ 位置减去 $(r+1)*d$。可以用函数表示为 $add(0,l,d), add(0,r+1,-d), add(1,l,l*d), add(1,r+1,-(r+1)*d)$。

这样一来，我们对于"Q l r"，同样拆成两个前缀和的差，表示为：

$$((r+1)*ask(0,r) - ask(1,r)) - (l*ask(0,l-1) - ask(1,l-1))$$

A Simple Problem with Integers

因此，在本题中，除了一个维护 b_i 的树状数组 c_0 之外，我们还需要一个维护 $b_i \times i$ 的树状数组 c_1。
因此，在进行操作"C l r d" 时，我们需要用树状数组 c_0 维护，给 l 位置加上 d，给 $r+1$ 位置减去 d；用树状数组 c_1 维护，给 l 位置加上 $l*d$，给 $r+1$ 位置减去 $(r+1)*d$。可以用函数表示为 $add(0,l,d), add(0,r+1,-d), add(1,l,l*d), add(1,r+1,-(r+1)*d)$。
这样一来，我们对于"Q l r"，同样拆成两个前缀和的差，表示为：

$$((r+1)*ask(0,r)-ask(1,r))-(l*ask(0,l-1)-ask(1,l-1))$$

```
long long ask(int k,int x){
  long long ans=0;
  for(;x;x-=x&-x) ans+=c[k][x];
  return ans;
}
void add(int k,int x,long long y){
  for(;x<=n;x+=x&-x) c[k][x]+=y;
}
```

例题

Lost Cows

有 n 头奶牛 $n \le 10^5$，已知它们的身高为 $1 \sim n$ 且各不相同，但不知道每头奶牛的具体身高。
现在这 n 头奶牛排成一列，已知第 i 头奶牛前面有 a_i 头奶牛比它矮，求每头奶牛的身高。

例题

Lost Cows

有 n 头奶牛 $n \le 10^5$，已知它们的身高为 $1 \sim n$ 且各不相同，但不知道每头奶牛的具体身高。
现在这 n 头奶牛排成一列，已知第 i 头奶牛前面有 a_i 头奶牛比它矮，求每头奶牛的身高。

我们倒着来考虑一下。
最后一头奶牛前面有 a_n 头奶牛比它矮，那么显然它的身高 $H_n = a_n + 1$。
而对于倒数第二头奶牛：

- 若 $a_{n-1} < a_n$，那么它的身高 $H_{n-1} = a_{n-1} + 1$；
- 若 $a_{n-1} \ge a_n$，那么它的身高 $H_{n-1} = a_{n-1} + 2$。

Lost Cows

我们现在来考虑第 k 头奶牛。由于我们已知 a_k 和 $H_{k+1}, \cdots, H_n$，那么它的身高 H_k 就是 $1 \sim n$ 这 n 个数，除去 $H_{k+1}, \cdots, H_n$ 这 $n-k$ 个数，剩下的 k 个数中，第 a_k+1 小的数。
这样我们就可以把问题进行转化。我们建立一个长度为 n 的 01 序列 b，初始全为 1。然后，从 n 到 1 倒序扫描每个 a_i，对于每一个 a_i:

- 查询 b 中第 a_i+1 个 1 在什么位置，这个位置就是第 i 头奶牛的身高 H_i。
- 把 b_{H_i} 变为 0。

也就是说，我们需要实施维护一个 01 序列，支持查询第 k 个 1 的位置，以及修改序列中一个数。

Lost Cows

一个想法时我们可以用树状数组维护 b 的前缀和。修改的话直接在树状数组中进行修改；而对于查询，我们可以每次二分一个答案 mid，并通过树状数组求出前缀和来与 k 进行比较。
时间复杂度 $O(n\log^2 n)$。

Lost Cows

一个想法时我们可以用树状数组维护 b 的前缀和。修改的话直接在树状数组中进行修改；而对于查询，我们可以每次二分一个答案 mid，并通过树状数组求出前缀和来与 k 进行比较。
时间复杂度 $O(n \log^2 n)$。

实际上，由于我们树状数组的一些特殊性质，我们可以去掉二分的这个 log。若一个数 x 最低的非零二进制位为第 p 位，也就是 $lowbit(x) = 2^p$，那么 c_x 维护的区间和是 $[x - 2^p + 1, x]$。
利用这么一个性质，我们可以倍增求解。首先初始化两个变量 $ans = 0, sum = 0$，然后从 $\lfloor \log n \rfloor$ 到 0 枚举 p。
对于每个 p，若 $ans + 2^p < n$ 且 $sum + c[ans + 2^p] < k$，则令 $ans+ = 2^p, sum+ = c[ans + 2^p]$。
这样一来，$H_i = ans + 1$ 即为所求。复杂度为 $O(n \log n)$。
这种思想也是一种常用的思想，即“以 2 的整数次幂为步长，能累加则累加”。由于树状数组恰好维护了区间长度为 2 的次幂的一些信息，所以可以直接调用，无需求前缀和，少了一个 log 的复杂度。