高级数据结构-树状数组

一、AcWing 241. 楼兰图腾

【题目描述】

在完成了分配任务之后，西部314来到了楼兰古城的西部。

相传很久以前这片土地上(比楼兰古城还早)生活着两个部落，一个部落崇拜尖刀(`V`)，一个部落崇拜铁锹(`∧`)，他们分别用`V`和`∧`的形状来代表各自部落的图腾。

西部314在楼兰古城的下面发现了一幅巨大的壁画，壁画上被标记出了n个点，经测量发现这n个点的水平位置和竖直位置是两两不同的。

西部314认为这幅壁画所包含的信息与这n个点的相对位置有关，因此不妨设坐标分别为$(1, y_1), (2, y_2), \ldots, (n, y_n)$，其中$y_1 \sim y_n$是1到$n$的一个排列。

西部314打算研究这幅壁画中包含着多少个图腾。

如果三个点$(i, y_i), (j, y_j), (k, y_k)$满足$1 \le i < j < k \le n$且$y_i > y_j, y_j < y_k$，则称这三个点构成`V`图腾；

如果三个点$(i, y_i), (j, y_j), (k, y_k)$满足$1 \le i < j < k \le n$且$y_i < y_j, y_j > y_k$，则称这三个点构成`∧`图腾；

西部314想知道，这n个点中两个部落图腾的数目。

因此，你需要编写一个程序来求出`V`的个数和`∧`的个数。

【输入格式】

第一行一个数n。

第二行是n个数，分别代表$y_1, y_2, \ldots, y_n$。

【输出格式】

两个数，中间用空格隔开，依次为`V`的个数和`∧`的个数。

【数据范围】

对于所有数据，$n \le 200000$，且输出答案不会超过$int64$。

$y_1 \sim y_n$是$1 \sim n$的一个排列。

【输入样例】

```
5
1 5 3 2 4
```

【输出样例】

```
3 4
```

【分析】

以`v`为例，我们需要遍历每个中间点，即对于每一个三元组(i,j,k)，我们遍历j的位置，以j为最低点的`v`的数量为j左边高于j的点数量 $\times$ j右边高于j的点数量，因此我们可以正序扫描一遍数组，对于每个元素$a[i]$，统计之前比$a[i]$大的元素数量即为i左边高于i的元素数量，统计完后将$a[i]$加入到树状数组中，即$add(a[i],1)$，表示值为$a[i]$的元素出现次数加一；接着逆序扫描一遍数组，对于每个元素$a[i]$，统计之前比$a[i]$大的元素数量即为i右边高于i的元素数量，统计完后也将$a[i]$加入到树状数组中。

因此本题的完整做法如下：

- 从左向右依次遍历每个数$a[i]$，使用树状数组统计在i位置之前所有比$a[i]$大的数的个数、以及比$a[i]$小的数的个数。
  统计完成后，将$a[i]$加入到树状数组；
- 从右向左依次遍历每个数$a[i]$，使用树状数组统计在i位置之后所有比$a[i]$大的数的个数、以及比$a[i]$小的数的个数。
  统计完成后，将$a[i]$加入到树状数组；
- 枚举每一个点i，其之前与之后比$a[i]$大的数的个数的乘积之和即为`v`的总数，其之前与之后比$a[i]$小的数的个数的乘积之和即为`^`的总数。

【代码】

```
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;

typedef long long LL;
const int N = 200010;
int a[N], c[N];
int ge[N], le[N];//ge[i]表示第i个数左边比它大的元素数量,le[i]比它小的元素数量
int n;
LL res1, res2;//分别表示两种图腾的数量

```

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

int ask(int x)
{
    int res = 0;
    for (; x; x -= lowbit(x)) res += c[x];
    return res;
}

void add(int x, int y)
{
    for (; x <= n; x += lowbit(x)) c[x] += y;
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)//顺序求一遍每个元素左边比它大或小的元素数量
    {
        cin >> a[i];
        ge[i] = ask(n) - ask(a[i]);
        le[i] = ask(a[i] - 1);
        add(a[i], 1);//将a[i]加入树状数组，即数字a[i]出现1次
    }
    memset(c, 0, sizeof c);//清空树状数组
    for (int i = n; i; i--)//逆序求一遍每个元素右边比它大或小的元素数量
    {
        res1 += (LL)ge[i] * (ask(n) - ask(a[i]));
        res2 += (LL)le[i] * ask(a[i] - 1);
        add(a[i], 1);//将a[i]加入树状数组，即数字a[i]出现1次
    }
    cout << res1 << ' ' << res2 << endl;
    return 0;
}
```

二、AcWing 242. 一个简单的整数问题（区间修改，单点查询）

【题目描述】

给定长度为N的数列A，然后输入M行操作指令。

第一类指令形如`C l r d`，表示把数列中第$l \sim r$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。

对于每个询问，输出一个整数表示答案。

【输入格式】

第一行包含两个整数N和M。

第二行包含N个整数$A[i]$。

接下来M行表示M条指令，每条指令的格式如题目描述所示。

【输出格式】

对于每个询问，输出一个整数表示答案。

每个答案占一行。

【数据范围】

$1 \le N, M \le 10^5$

$|d| \le 10000$

$|A[i]| \le 10^9$

【输入样例】

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

【输出样例】

```
4
1
2
5
```

【分析】

树状数组能够完成**单点修改、求区间和**，而本题需要完成的是**区间修改、求单点值**。区间修改可以使用差分数组完成，若在$a[l,r]$加上d，则只需要在差分数组的l和$r+1$处进行修改即可，即$c[l]+=d, c[r+1]-=d$，求单点的值即为求差分数组在该点的前缀和。因此可以构造一个差分树状数组，使得修改及求前缀和的操作时间复杂度都为$O(logn)$。

【代码】

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <string>
using namespace std;

typedef long long LL;
const int N = 100010;
int a[N];
int n, m;
LL c[N];

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

void add(int x, int y)
{
    for (; x <= n; x += lowbit(x)) c[x] += y;
}

LL ask(int x)
{
    LL res = 0;
    for (; x; x -= lowbit(x)) res += c[x];
    return res;
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {

        cin >> a[i];
```

```
        add(i, a[i] - a[i - 1]);//构造差分树状数组
    }
    while (m--)
    {
        string op;
        int l, r, d;
        cin >> op >> l;
        if (op == "C")
        {
            cin >> r >> d;
            add(l, d), add(r + 1, -d);//在a[l,r]加上d即为在差分数组c[l]+d,c[r+1]-d
        }
        else cout << ask(l) << endl;//求a[l]即为求差分数组c[l]的前缀和
    }
    return 0;
}
```

三、AcWing 243. 一个简单的整数问题2（区间修改，区间查询）

【题目描述】

给定一个长度为N的数列A，以及M条指令，每条指令可能是以下两种之一：

- `C l r d`，表示把$A[l], A[l+1], \dots, A[r]$都加上$d$。
- `Q l r`，表示询问数列中第$l \sim r$个数的和。

对于每个询问，输出一个整数表示答案。

【输入格式】

第一行包含两个整数N和M。

第二行包含N个整数$A[i]$。

接下来M行表示M条指令，每条指令的格式如题目描述所示。

【输出格式】

对于每个询问，输出一个整数表示答案。

每个答案占一行。

【数据范围】

$1 \le N, M \le 10^5$

$|d| \leq 10000$

$|A[i]| \leq 10^9$

【输入样例】

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

【输出样例】

```
4
55
9
15
```

【分析】

假设原数组为a，差分数组为b，我们已经知道$a[l,r]+d$等价于$b[l]+=d, b[r+1]-=d$，$a[i]$等价于$b[i]$的前缀和，那么如何算出$a[i]$的前缀和呢？

假设我们需要计算$a[1]+a[2]+\cdots+a[x]$，那么就相当于计算$(b[1])+(b[1]+b[2])+\cdots+(b[1]+b[2]+\cdots+b[x])$，如下图所示：

0	b[1]	b[2]	b[3]	…	b[x]
1	b[1]	b[2]	b[3]	…	b[x]
2	b[1]	b[2]	b[3]	…	b[x]
3	b[1]	b[2]	b[3]	…	b[x]
…	…	…	…	…	…
x	b[1]	b[2]	b[3]	…	b[x]

其中蓝色标注的是我们需要计算的结果，将其补全（红色标注）后蓝色部分即为整体减去红色部分，即$(b[1]+b[2]+\cdots+b[x])*(x+1)-(1*b[1]+2*b[2]+\cdots+x*b[x])$，可以看到式子中有两部分前缀和，一部分是$b[x]$的前缀和，另一部分是$x*b[x]$的前缀和，因此同时维护两个差分树状数组即可。

【代码】

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <string>
using namespace std;

typedef long long LL;
const int N = 100010;
int a[N];
int n, m;
LL c1[N], c2[N];//c1[i]维护差分数组b[i]的前缀和,c2[i]维护i*b[i]的前缀和

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

void add(LL c[], int x, LL y)
{
    for (; x <= n; x += lowbit(x)) c[x] += y;
}

LL ask(LL c[], int x)//求差分树状数组c[x]的前缀和
{
    LL res = 0;
    for (; x; x -= lowbit(x)) res += c[x];
    return res;
}

LL prefix_ask(int x)//求原数组a[x]的前缀和
{
    return ask(c1, x) * (x + 1) - ask(c2, x);
}


int main()
```

```cpp
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
        add(c1, i, a[i] - a[i - 1]);
        add(c2, i, (LL)i * (a[i] - a[i - 1]));
    }
    while (m--)
    {
        string op;
        int l, r, d;
        cin >> op >> l >> r;
        if (op == "C")
        {
            cin >> d;
            add(c1, l, d), add(c1, r + 1, -d);
            add(c2, l, l * d), add(c2, r + 1, (r + 1) * -d);
        }
        else cout << prefix_ask(r) - prefix_ask(l - 1) << endl;
    }
    return 0;
}
```

四、AcWing 244. 谜一样的牛（树状数组+二分）

【题目描述】

有n头奶牛，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头奶牛站成一列，已知第i头牛前面有A_i头牛比它低，求每头奶牛的身高。

【输入格式】

第1行：输入整数n。

第$2\sim n$行：每行输入一个整数A_i，第i行表示第i头牛前面有A_i头牛比它低。

（注意：因为第1头牛前面没有牛，所以并没有将它列出）

【输出格式】

输出包含n行，每行输出一个整数表示牛的身高。

第i行输出第i头牛的身高。

【数据范围】

$1 \leq n \leq 10^5$

【输入样例】

```
5
1
2
1
0
```

【输出样例】

```
2
4
5
3
1
```

【分析】

我们发现，如果说第K头牛的前面有A_k头牛比它矮，那么它的身高H_k就是数值$1 \sim n$中第A_k+1小的没有在$H_{k+1}, H_{k+2}, \sim, H_n$中出现过的数。

所以说，我们需要建立一个长度为n的1序列b，刚开始都是1，表示n种身高都有1次使用机会，然后从n到1倒序扫描每一个A_i，对于每个A_i执行查询和修改操作。

也就是说这道题目的题意就是让我们动态维护一个01序列，支持查询第k个1所在的位置，以及修改序列中的一个数值。第k个1所在的位置是第一个前缀和大于等于k的位置，因此我们可以二分查找出这个位置，时间复杂度为$O(logn)$，每次判断mid的前缀和$ask(mid)$是否大于等于k，时间复杂度也为$O(logn)$，因此总的时间复杂度为$O(nlog^2n)$。

【代码】

```
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;

const int N = 100010;
int h[N], c[N], res[N];
int n;
```

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

void add(int x, int y)
{
    for (; x <= n; x += lowbit(x)) c[x] += y;
}

int ask(int x)
{
    int res = 0;
    for (; x; x -= lowbit(x)) res += c[x];
    return res;
}

int main()
{
    cin >> n;
    for (int i = 2; i <= n; i++) cin >> h[i];
    for (int i = 1; i <= n; i++) add(i, 1);//初始每种身高都有1次使用机会
    for (int i = n; i; i--)
    {
        int k = h[i] + 1;//第i头牛前面有h[i]头牛比它低,因此第i头为剩下的身高中第h[i]+1高的
        int l = 1, r = n;
        while (l < r)
        {
            int mid = l + r >> 1;
            if (ask(mid) >= k) r = mid;//剩余身高中前缀和第一个大于等于k的即为第k高的身高
            else l = mid + 1;
        }
        res[i] = r;
        add(r, -1);//身高用过后使用机会-1
    }
    for (int i = 1; i <= n; i++) cout << res[i] << endl;
    return 0;
}
```