

# 树状数组

可它跟树又有多大关系呢？

主要应用于大部分基于区间上的更新以及求和问题。

- 1.单点修改+区间查询
- 2.区间修改+单点查询
- 3.区间修改+区间查询

优点：修改查询 $O(\log n)$ ，码量少常数小

缺点：功能有限

但是避开线段树它不香吗

## 前置知识点：（一阶）差分思想（简）

首先大家一定都知道差分，那么差分究竟是怎么一回事呢？就让小编带大家了解一下吧！

好了不玩了

首先大家一定都知道前缀和，那么（没玩梗，真的）给定  $n$  个元素的数组  $A$  前缀和数组  $B$  有  $B[i] = A[i] + B[i-1]$

也就是  $B[1] = A[1]; B[2] = A[1] + A[2]; B[3] = A[1] + A[2] + A[3]; \dots$

那么所谓的（一阶）差分，就是前缀和的逆运算。设其数组为  $C$  则  $C[i] = A[i] - A[i-1]$  也就是

- $C[1] = A[1]$
- $C[2] = A[2] - A[1]$
- $C[3] = A[3] - A[2]$
- $\dots$
- $C[i] = A[i] - A[i-1]$

则将  $C$  取前缀和，便得到原始数组  $A$

主要用途  $O(1)$  处理区间值（加减）修改

如将区间  $(l, r)$  加上  $val$  只需差分数组  $C$  中

```
C[l] += val;  
C[r+1] -= val;
```

求多次变更后某项的值，只需求其差分数组C中该项的前缀和即可

## 下面是正题

### 先说灵魂

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

返回x的二进制从低到高位的一个'1'代表的数，例如12的二进制为1100， $\text{lowbit}(12) = 4$

### 再说原理

设原始数组为A，树状数组为C，则

- $C[1] = A[1];$
- $C[2] = A[1] + A[2];$
- $C[3] = A[3];$
- $C[4] = A[1] + A[2] + A[3] + A[4];$
- $C[5] = A[5];$
- $C[6] = A[5] + A[6];$
- $C[7] = A[7];$
- $C[8] = A[1] + A[2] + A[3] + A[4] + A[5] + A[6] + A[7] + A[8];$
- □□□□□□

不难发现是有规律的， $C[i] = A[i-2^k+1] + A[i-2^k+2] + \dots + A[i]$ ，其中k为i的二进制中从最低位到高位连续零的长度

那么怎么求和呢？如  $\sum_{i=1}^7 A[i] = C[7] + C[6] + C[4]$ ；

而7在二进制下为111，减去最低位的'1'后为110，对应6；再减去最低位的'1'后为100，对应4；正好对应上式的三个下标

那么实现方法也就一目了然了：

```
int getsum(int x){//区间查询 1-x
    int ans = 0;
    while(x){
        ans += c[x];
        x -= lowbit(x);
    }
    return ans;
}
```

相应地，建立n个元素的树状数组：

```
void update(int x, int val){//单点修改,也是建立过程
```

```

while(x <= n){
    c[x] += val;
    x += lowbit(x);
}
}

```

```

for(int i = 1; i <= n; ++i){
    scanf("%d", &tmp);
    update(i, tmp);
}

```

以上为基础版树状数组实现，即单点修改+区间查询。

而区间修改+单点查询只需用A的差分数组建立树状数组即可。

```
update(i, tmp - last);
```

区间修改(x, y, val)[]

```
update(x, val);
update(y + 1, -val);
```

## 最后是类似于基础线段树的区间修改+区间查询

这里我们还是利用差分（差分数组为C[]

$$\sum_{i=1}^n A[i] = (C[1]) + (C[1]+C[2]) + \dots + (C[1]+C[2]+\dots+C[n])$$

$$= n * C[1] + (n-1) * C[2] + \dots + C[n]$$

$$= n * (C[1]+C[2]+\dots+C[n]) - (0 * C[1] + 1 * C[2] + \dots + (n-1) * C[n])$$

$$\text{所以上式可以变为} \sum_{i=1}^n A[i] = n * \sum_{i=1}^n C[i] - \sum_{i=1}^n (C[i] * (i-1))$$

如果理解前面的都比较轻松的话，这里也就知道要干嘛了，维护两个数状数组[] $\sum1[i] = C[i]$  [] $\sum2[i] = C[i] * (i-1)$

下面完整代码（稍作修改可适用于下方例题）

```

#include<bits/stdc++.h>
#define namespace namespace
#define namespace namespace
#define tsd std
using namespace std; //传统艺能
//using namespace std;
//using namespace tsd;

int sum1[1000086], sum2[1000086];

```

```
int n, m;

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

void update(int x, int val){
    int tmp = x;
    while(x <= n){
        sum1[x] += val;
        sum2[x] += val * (tmp - 1);
        x += lowbit(x);
    }
}

int getsum(int x){
    int ans = 0, tmp = x;
    while(x){
        ans += tmp * sum1[x] - sum2[x];
        x -= lowbit(x);
    }
    return ans;
}

int main(){
    scanf("%d%d", &n, &m);
    int tmp, last = 0;
    for(int i = 2; i <= n; ++i){
        scanf("%d", &tmp);
        update(i, tmp - last);
        last = tmp;
    }
    int op, x, y, z;
    while(m--){
        scanf("%d", &op);
        if(op == 1){
            scanf("%d %d %d", &x, &y, &z);
            update(x, z);
            update(y + 1, -z);
        }
        else{
            scanf("%d %d", &x, &y);
            printf("%d\n", getsum(y) - getsum(x-1));
        }
    }
}
```

## 板子例题

<https://www.luogu.com.cn/problem/P3374>

<https://www.luogu.com.cn/problem/P3368>

<https://www.luogu.com.cn/problem/P3372>

## 率 级应用：维护-查询-修改区间最值

线段树：我不要面子的吗

### 区间修改

既然是维护最值，那么树状数组  $C[i]$  中保存的就是  $\max(A[i-2^k+1], A[i-2^k+2], \dots, A[i])$  如下：

- $C[1] = A[1];$
- $C[2] = \max(A[1], A[2]);$
- $C[3] = A[3];$
- $C[4] = \max(A[1], A[2], A[3], A[4]);$
- $C[5] = A[5];$
- $C[6] = \max(A[5], A[6]);$
- $C[7] = A[7];$
- $C[8] = \max(A[1], A[2], A[3], A[4], A[5], A[6], A[7], A[8]);$
- □□□□□□

但修改  $A[i]$  需要将所有包含  $A[i]$  的  $C[j]$  全部重算

可以发现，对于  $x$  可以转移到  $x$  的只有  $x-2^0, x-2^1, x-2^2, \dots, x-2^k$   $k$  满足  $2^k < \text{lowbit}(x)$  且  $2^{(k+1)} \geq \text{lowbit}(x)$

例  $x = 1010000 (80)$

$$= 1001000 + \text{lowbit}(1001000) = 1001000 + 1000 = 1001000 + 2^3$$

$$= 1001100 + \text{lowbit}(1001100) = 1001100 + 100 = 1001100 + 2^2$$

$$= 1001110 + \text{lowbit}(1001110) = 1001110 + 10 = 1001110 + 2^1$$

$$= 1001111 + \text{lowbit}(1001111) = 1001111 + 1 = 1001111 + 2^0$$

所以对于每个  $C[i]$  重算的复杂度为  $O(\log n)$  总复杂度  $O((\log n)^2)$  若维护最大值且修改只大不小，则复杂度  $O(\log n)$

与维护区间和类似的维护最值代码：

```
void update(int x){
    int lb;
    while(x <= n){
        c[x] = a[x];
        lb = lowbit(x);
        for(int j = 1; j < lb; j <= 1)    c[x] = max(c[x], c[x-j]);
        x += lb;
    }
}
```

```
}  
}
```

此应用部分以上部分内容引自CSDN博主LbyG文章，原文链接：<https://blog.csdn.net/u010598215/article/details/48206959>

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: <https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:manespace:%E6%A0%91%E7%8A%B6%E6%95%B0%E7%BB%84&rev=1591173749>

Last update: 2020/06/03 16:42