前置知识 - lowbit()运算

得到非负整数 x 在二进制表示下**最低位** 1 及其后面 0 构成的数值

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

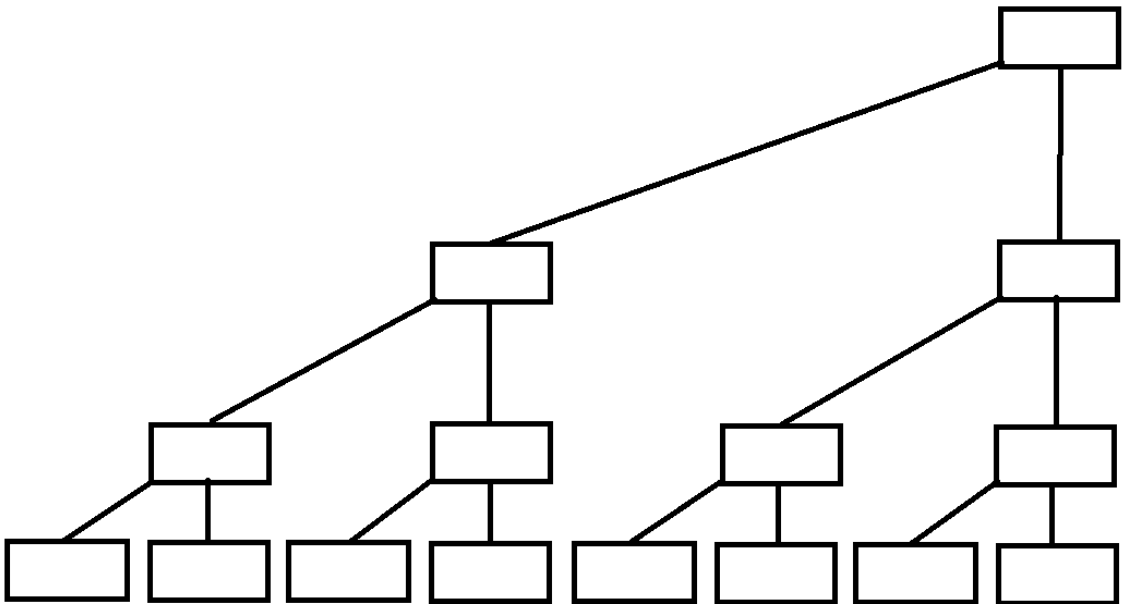
树状数组

又称为二叉索引树(Binary Indexed Tree，BIT)

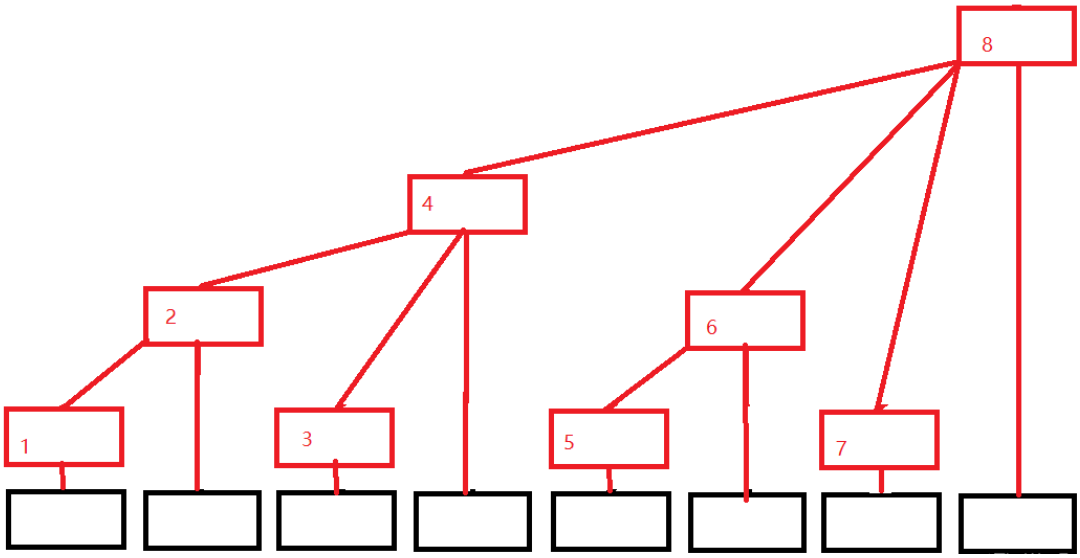
一种用来解决**动态前缀和**问题的数据结构

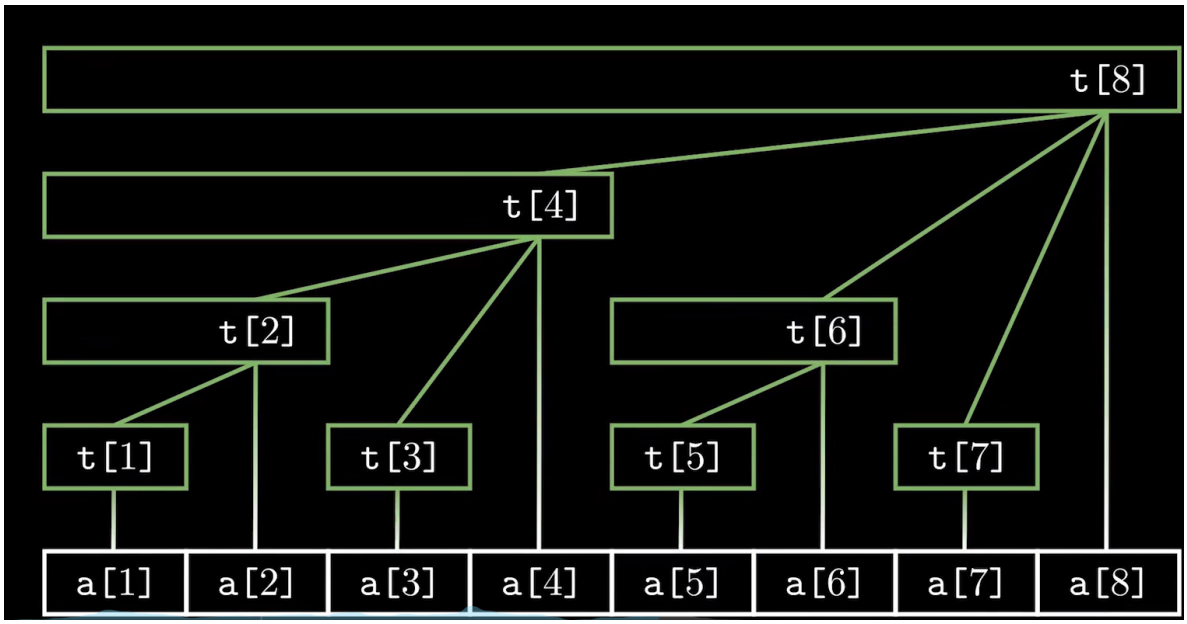
原理

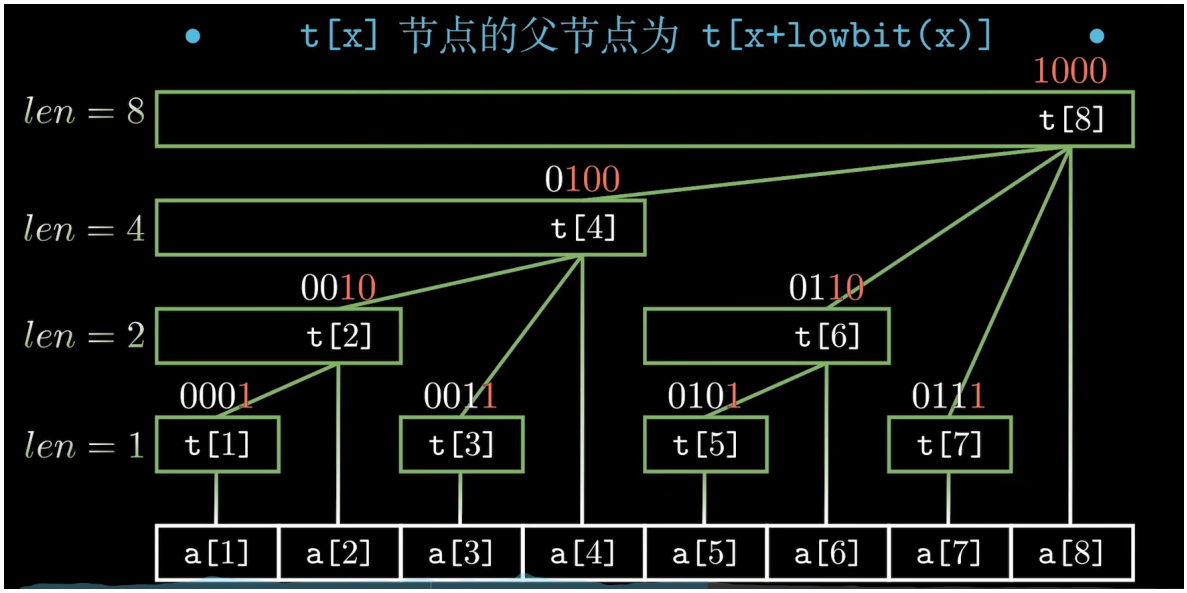
二叉树结构



树状数组结构


8
4
2
6
1
3
5
7


t[8]
t[4]
t[2]
t[6]
t[1]
t[3]
t[5]
t[7]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]



1.t 数组就是所谓的 树状数组

2.每个节点 $t[x]$ 表示以 x 为根的子树中叶节点值的和

3.把每个节点 $t[x]$ 中的下标 x 转化为二进制的形式， 我们会发现每一层的末尾的 0 的数量是相同的 (也就是说同一层的 $lowbit(x)$ 返回值相同)

4.$t[x]$ 表示的是 $\sum_{i=x-lowbit(x)+1}^{x} a_i$, 它的父节点是 $t[x+lowbit(x)]$

5.整棵树的深度为 $log(n)+1$

操作

修改

```
void add(int pos, int val){  //  c为树状数组
    while(pos <= n)  // 不能越界
    {
        c[pos] += val;
        pos += lowbit(pos);
    }
}
```

前缀求和

```
int getsum(int x){  // a[1]..a[x]的和
    int ans = 0;
    while (x >= 1)
    {
        ans = ans + c[x];
        x = x - lowbit(x);
    }
    return ans;
}
```

区间求和

```
int query(int l, int r){  //  前缀相减
    return getsum(r) - getsum(l);
}
```

小结

1.时间复杂度

对于每次修改和查询的操作，本质是 x 的变化，最差情况是每次移动一个二进制位，所以单次操作复杂度为$logn$

2.局限性

可以发现树状数组的查询是通过前缀和做差实现的，所以只能维护部分可以满足前缀和的信息，如区间和，区间 xor，而如区间最值这些问题是无法维护的

区间修改

考虑引入**差分数组** b

$b[1] = a[1]$

$b[i] = a[i] - a[i-1]$

修改

对 $[L,R]$所有 $a[i]+val$ 相当于$b_L+=val$，$b_{R+1}-=val$ ，这样区间修改就被转化为两次单点修改

单点查询

$a[i] = (a[i] - a[i-1]) + (a[i-1] - a[i-2]) + \cdots + (a[2] - a[1]) + (a[1]) = \sum_{j=1}^{i} b_j$

单点查询就是就是对差分数组求前缀和，参考前面的前缀求和即可

区间查询

$\sum_{i=1}^{r} a_i$

$= \sum_{i=1}^{r} \sum_{j=1}^{i} b_j$

$= \sum_{i=1}^{r} b_i * (r - i + 1)$

$= (r+1) \sum_{i=1}^{r} b_i - \sum_{i=1}^{r} b_i * i$

对于一个前缀和 我们需要额外的维护一个 $b_i * i$ 的树状数组

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;

int n;
ll a[1000050], b[1000050];
ll t1[1000050], t2[1000050];

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

void add(int x, int val)
{
    int val1 = val;
    ll val2 = 1ll * val * x;
    while(x <= n)
    {
        t1[x] += val1;
        t2[x] += val2;
        x += lowbit(x);
    }
}

ll getsum(int x)//sum[1, x]
{
    ll sum1 = 0;
    ll sum2 = 0;
    int xx = x;
    while(x > 0)
    {
        sum1 += t1[x];
        sum2 += t2[x];
        x -= lowbit(x);
    }
    return 1ll * sum1 * (xx + 1) - sum2;
}

ll query(int l, int r)
{
    return getsum(r) - getsum(l - 1);
```

```
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    int q;
    cin >> n >> q;

    for(int i = 1; i <= n; i++) cin >> a[i];

    for(int i = 1; i <= n; i++)
        b[i] = a[i] - a[i - 1];

    for(int i = 1; i <= n; i++)
        add(i, b[i]);

    while(q--)
    {
        int op, l, r;
        cin >> op >> l >> r;
        if(op == 1)
        {
            int val;
            cin >> val;
            add(l, val);
            add(r + 1, -val);
        }
        else
            cout << query(l, r) << '\n';
    }

    return 0;
}
```