树状数组

概念原理

应用

- $O(\log n)$时间**求前缀和与修改某数**，解决**区间和**问题，区间极值不可
- 对于1~n的数，**维护未被使用的数的排名**，`findKthNum()`
    - 即初始每个数对应的值都是1,前缀和即为排名；被用的数对应值-1,即为0
    - 对于排名k的数，使用二分查找
- 通常和差分、前缀和搞在一起

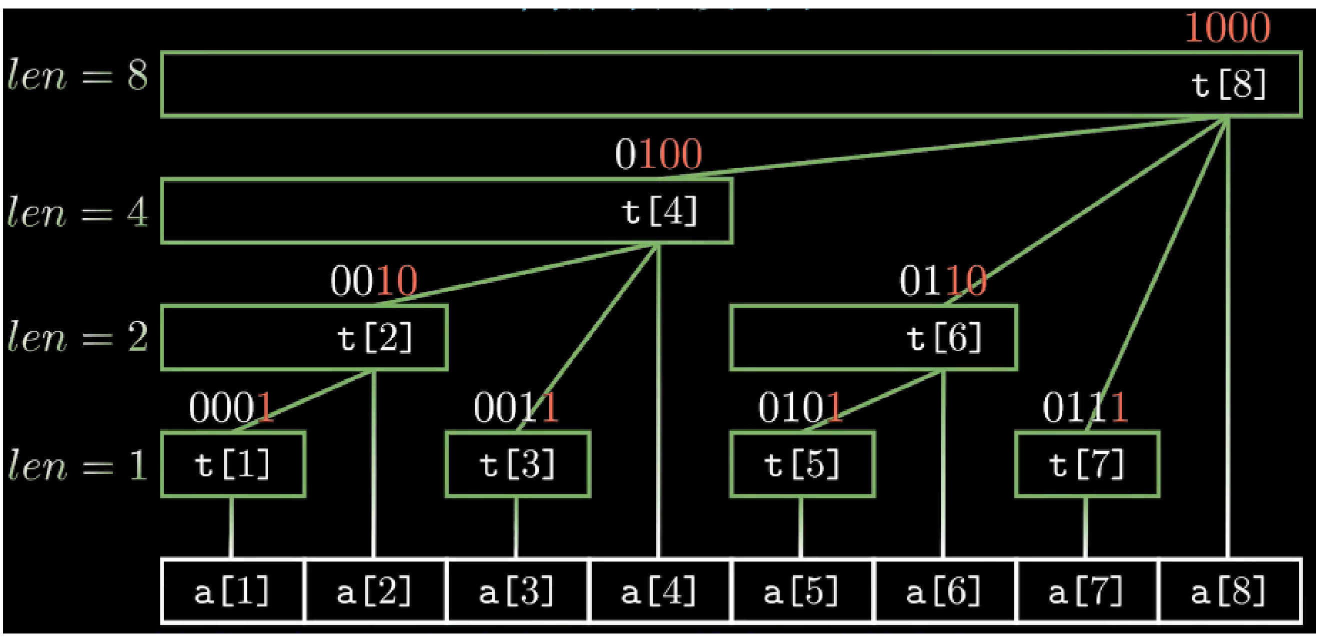
基本原理

二进制思想：$x = 2^{i_k} + 2^{i_{k-1}} + \cdots + 2^{i_1}, k \le \log x, i_k > i_{k-1} > \ldots$，则对于$[1, x]$ 的区间划分成k部分：

- (1) $(x - 2^{i_1}, x]$ 包含2^{i_1}个数，其中2^{i_1}是x的二进制表示的最后一位1
- (2) $(x - 2^{i_1} - 2^{i_2}, x - 2^{i_1}]$ 包含2^{i_2}个数，其中2^{i_2}是$x - 2^{i_1}$的二进制表示的最后一位1
- (3) $(x - 2^{i_1} - 2^{i_2} - 2^{i_3}, x - 2^{i_1} - 2^{i_2}]$ 包含2^{i_3}个数, ...
- ...
- (k) $(0, x - 2^{i_1} - 2^{i_2} - \cdots - 2^{i_{k-1}}]$ 包含2^{i_k}个数

(L, R] 区间的长度一定是R的二进制表示的最后一位1

求最后一位1可以用 `lowbit(x) { return x & -x }`，表示仅保留最后一位1后对应的值。于是最终区间可以表示为 `[R - lowbit(R) + 1, R]` 。于是可用 `c[x]` 记录以x为右端点，长度为lowbit(x)的区间



连线是构成关系，然后也可看做树边

```
(1) c[16] = a[16] + c[15] + c[14] + c[12] + c[8];
(2) c[8]  = a[8]  + c[7]  + c[6]  + c[4];
(3) c[12] = a[12] + c[11] + c[10];          c[4] = a[4] + c[3] + c[2];
...
```

对于x>0，均可一般记为 x = XX100⋯0(k个0)； c[x]则为以x结尾的，长度是2^k的区间和。计算c[x]

- 首先必有a[x]，剩下需计算$[x-2^k+1, x-1]$的内容，x-1 = XX011⋯1(k个1)
- 然后可分为k段计算，即每一个1都对应一个儿子。**通过父节点找到所有子节点**
  - e.g. X0|1111 可分为： (1110, **1111**], (1100, **1110**], (1000, **1100**], (0000, **1000**]
  - `c[x] = a[x] + c[x-1] + c[x-1-lowbit(x-1)] + ...` 直到变为0；即每次去掉最后一个1
- 反向，**通过子节点找到父节点**：重要，对应于修改某个数
  - 即找到所有包含子节点的父节点
  - 修改完一次，直接影响的节点只有1个(说明是树)： `parent(x) = x + lowbit(x)`
    - 因而迭代向上即可，每迭代一层，末尾0的数量增加一个

操作

修改：子节点找父节点 `for(int i = x; i <= n; i += lowbit(i)) 更新 tr[i];`
查询：拆分成每一段 `for(int i = x; i > 0; i -= lowbit(i)) res += tr[i];`
初始化：

- 直接一个个加进去 `for(int i = 1; i <= n; i++) add(i, a[i])` ，好比暴力建堆O(nlogn)；但其实这个也够了，一般用到树状数组的复杂度都是O(mlogn)，所以没有增加多少
- 类似Floyd建堆，也有O(n)的方式
  - 对每个x，找所有子节点，只加“树边”： `for (int i = x-1; i; i -= lowbit(i)) c[x] += c[i]` 这样就是n-1次；加上原数组的点 `a[i]` ，就是 `2n-1`
  - 对于x，求x的前缀和 `s[x]` ，那么根据定义有 `c[x] = s[x] - s[x-lowbit(i)]` ，求前缀和O(n)，然后再求c[x]也是O(n)，总共也是O(n)

基本模板

原始的

```
const int N = xx;

int n;
int a[N];   // 原始数组
int tr[N]; // 对应的树状数组
```

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

// 维护 a[x] += c
void add(int x, int c) {
    for (int i = x; i <= n; i += lowbit(i)) tr[i] += c;
}

// 查询 a[1] + ... + a[x] 的前缀和
int sum(int x) {
    int res = 0;
    for (int i = x; i; i -= lowbit(i)) res += tr[i];
    return res;
}

// 初始化
void init() {
    // 最简单： 一个一个加
    for (int i = 1; i <= n; ++i) add(i, a[i]);
    // 类似floyd建堆
}
```

维护排名

```cpp
// 此时a[]原始数组中存的是0或1,表示是否存在
int findKthNumMin(int k) {
    int l = 1, r = n, mid;
    int res;
    while (l <= r) {
        mid = l + r >> 1;
        if (k > sum(mid)) l = mid + 1;
        else r = mid - 1;
    }
    return l;
}
```