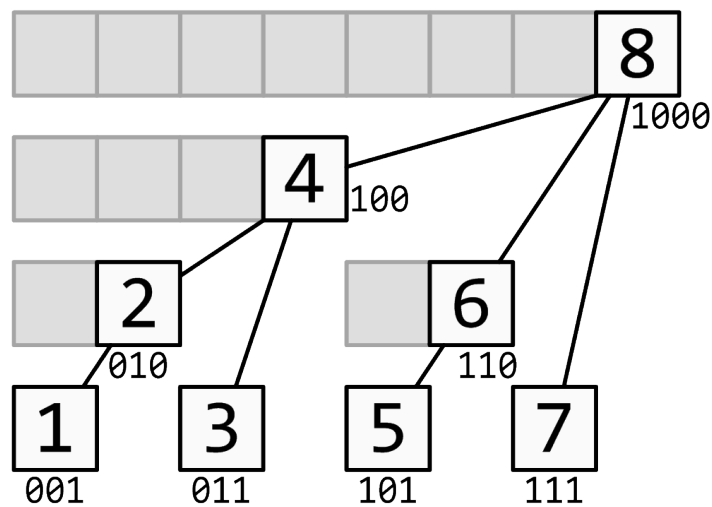


# 树状数组

## Fenwick Tree / Binary Indexed Tree

### 树状数组

**Idea:** 通过神奇的 `lowbit()` 操作使得  $c[x]$  包含了其前一系列  $c[]$  的和，管理一系列  $a[]$ 。



例如， $c[4]$  管理了  $a[1...4]$ ， $c[6]$  管理了  $a[5...6]$ ， $c[7]$  只管理了  $a[7]$ 。

**Complexity:**  $O(\lg n)$

**Code:**

```
1  int c[N];
2  inline int lowbit(int x){
3      return x & -x;
4  }
5  int querySum(int x){
6      int res = 0;
7      while(x){
8          res += c[x];
9          x -= lowbit(x);
10     }
11     return res;
12 }
13 void add(int x, int v){
14     while(x <= n){
```

```

15         c[x] += v;
16         x += lowbit(x);
17     }
18 }

```

## 二维树状数组

**Idea:** 树状数组扩展成二维，解决二维的单点/区间修改/求和问题。

**Complexity:**  $O(\lg^2 n)$

**Code:**

```

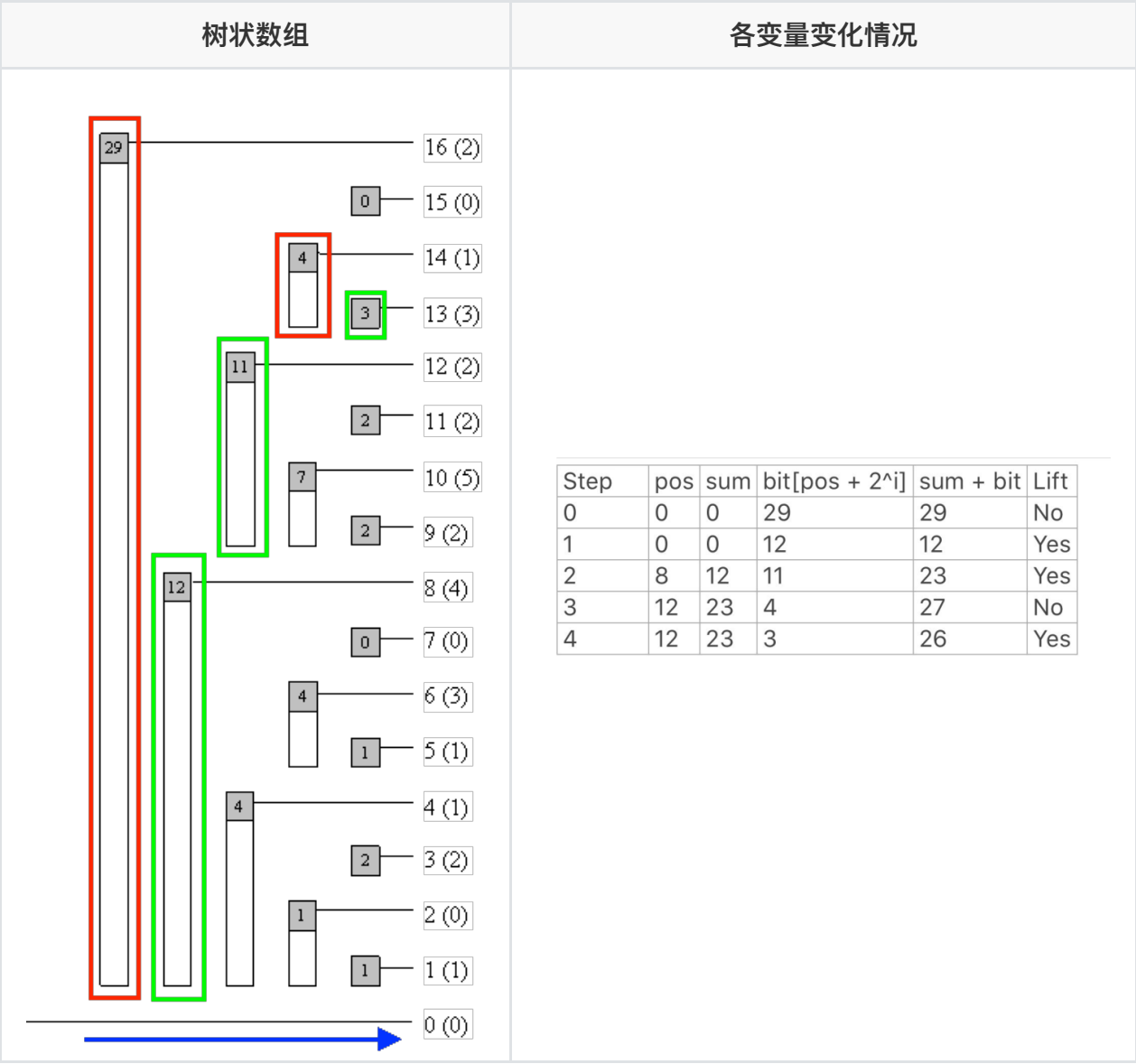
1  int c[N][N];
2  inline int lowbit(int x){ return x & -x; }
3  inline void addy(int x, int y, int val){ while(y <= m){ c[x][y] +=
    val; y += lowbit(y); } }
4  inline void addx(int x, int y, int val){ while(x <= n){ addy(x, y,
    val, c); x += lowbit(x); } }
5  inline int sumy(int x, int y){ int res = 0; while(y){ res += c[x][y];
    y -= lowbit(y); } return res; }
6  inline int sumx(int x, int y){ int res = 0; while(x){ res += sumy(x,
    y, c); x -= lowbit(x); } return res; }

```

## 树状数组倍增

**Problem:** 给定某个序列，要求维护的操作有：单点修改，求前缀和，搜索某个前缀和（类似于在前缀和数组上求 `lower_bound`）。

**Idea:** 假设我们要搜索前缀和为  $val$  的地方，设定一个 `pos` 指针，它初始为 0，最终将指向最大的前缀和小于  $val$  的位置；再设置一个变量 `sum`，存储 `pos` 处的前缀和；设置倍增的长度 `i`，最初为  $\lg n$ （为了代码方便，一般取 20 即可），在倍增的过程中不断减小至 0。每一个状态 (`pos`, `sum`, `i`) 表示我们现在考虑的是位置 `pos+(1<<i)` 的前缀和，这个前缀和的值是 `sum+c[pos+(1<<i)]`，如果它大于等于了  $val$ ，那么我们减小倍增的长度 `i`；否则，我们把 `pos` 提到 `pos+(1<<i)` 处。



**What's more:** 只要我们维护的信息具有单调性，就可以用这个方法。

**Compelxity:**  $O(n \lg n)$

**Code:**

```
1 int search(int val){
2     int pos = 0, sum = 0;
3     for(int i = 20; i >= 0; i--){
4         if(pos + (1<<i) <= n && sum + c[pos+(1<<i)] < val)
5             pos += (1<<i), sum += c[pos];
6     }
7     return pos + 1;
}
```