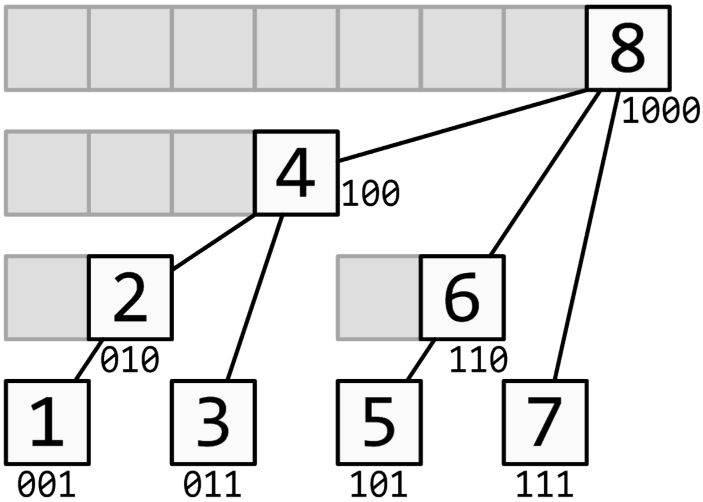
树状数组

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 $a[1\dots4]$，$c[6]$ 管理了 $a[5\dots6]$，$c[7]$ 只管理了 $a[7]$。

Complexity：$O(\lg n)$

Code：

```
int c[N];
inline int lowbit(int x){
    return x & -x;
}
int querySum(int x){
    int res = 0;
    while(x){
        res += c[x];
        x -= lowbit(x);
    }
    return res;
}
void add(int x, int v){
    while(x <= n){
        c[x] += v;
        x += lowbit(x);
    }
}
```

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：$O(\lg^2 n)$

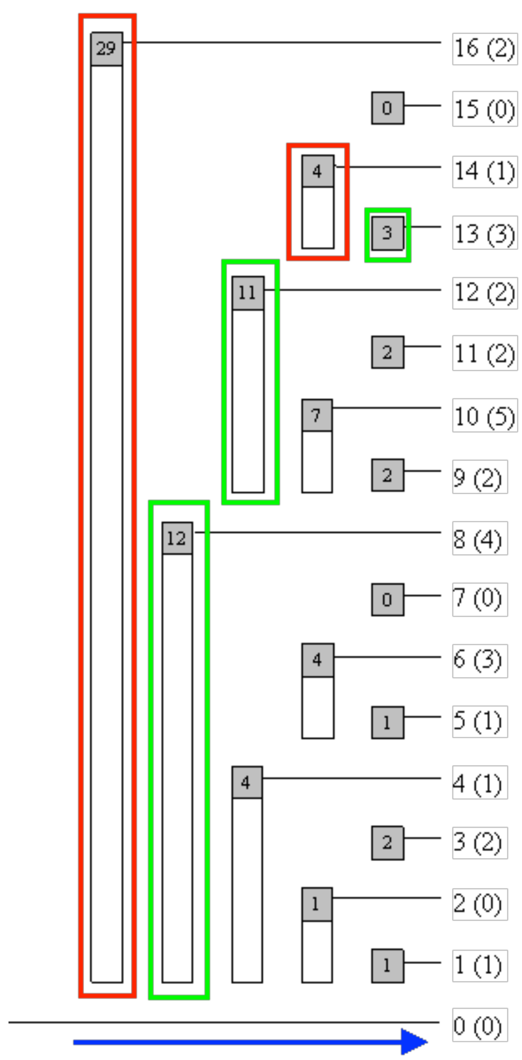
Code：

```
int c[N][N];
inline int lowbit(int x){ return x & -x; }
inline void addy(int x, int y, int val){ while(y <= m){ c[x][y] += val; y += lowbit(y); } }
inline void addx(int x, int y, int val){ while(x <= n){ addy(x, y, val, c); x += lowbit(x); } }
inline int sumy(int x, int y){ int res = 0; while(y){ res += c[x][y]; y -= lowbit(y); } return res; }
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：假设我们想要搜索前缀和为 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。

<table>
<tr><th>树状数组</th><th>各变量变化情况</th></tr>
<tr><td>


</td><td>
<table>
<tr><td>Step</td><td>pos</td><td>sum</td><td>bit[pos + 2^i]</td><td>sum + bit</td><td>Lift</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>29</td><td>29</td><td>No</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>12</td><td>12</td><td>Yes</td></tr>
<tr><td>2</td><td>8</td><td>12</td><td>11</td><td>23</td><td>Yes</td></tr>
<tr><td>3</td><td>12</td><td>23</td><td>4</td><td>27</td><td>No</td></tr>
<tr><td>4</td><td>12</td><td>23</td><td>3</td><td>26</td><td>Yes</td></tr>
</table>
</td></tr>
</table>

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：

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