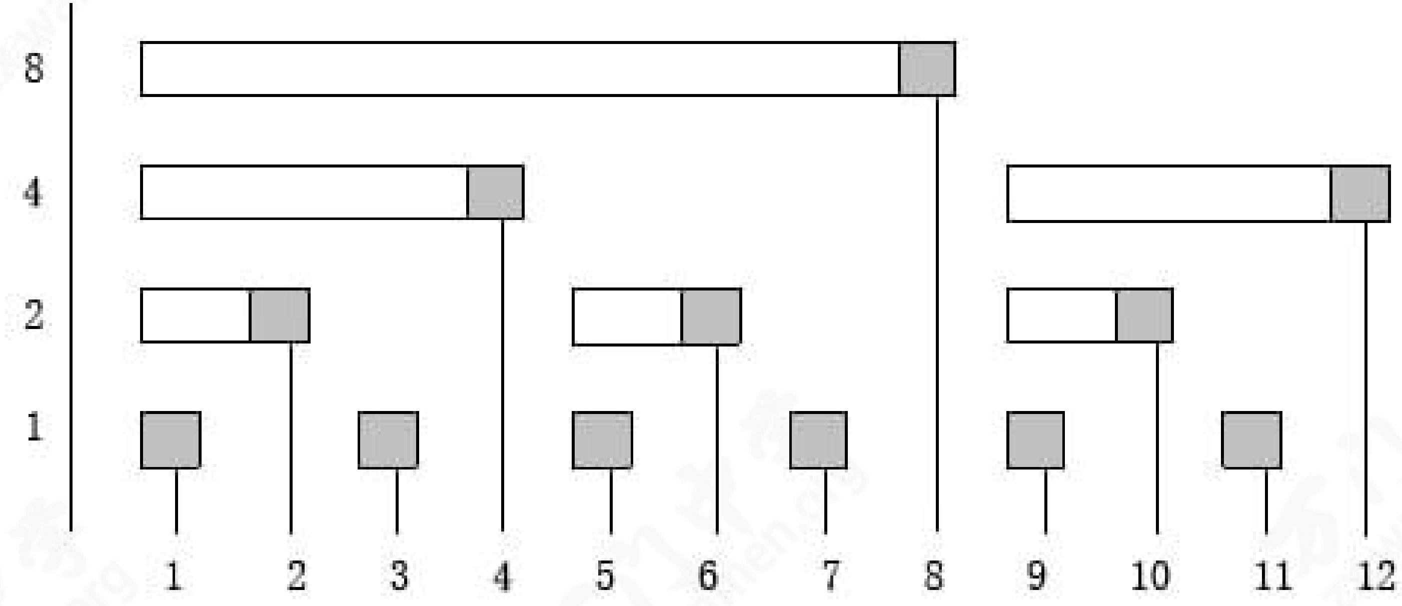
树状数组的操作

主讲人：邓哲也

树状数组的查询

根据下图可以得到查询 a[1..i] 前缀和的方法：

```
ans = 0
while (i > 0) ans += sum[i], i -= C(i);
```

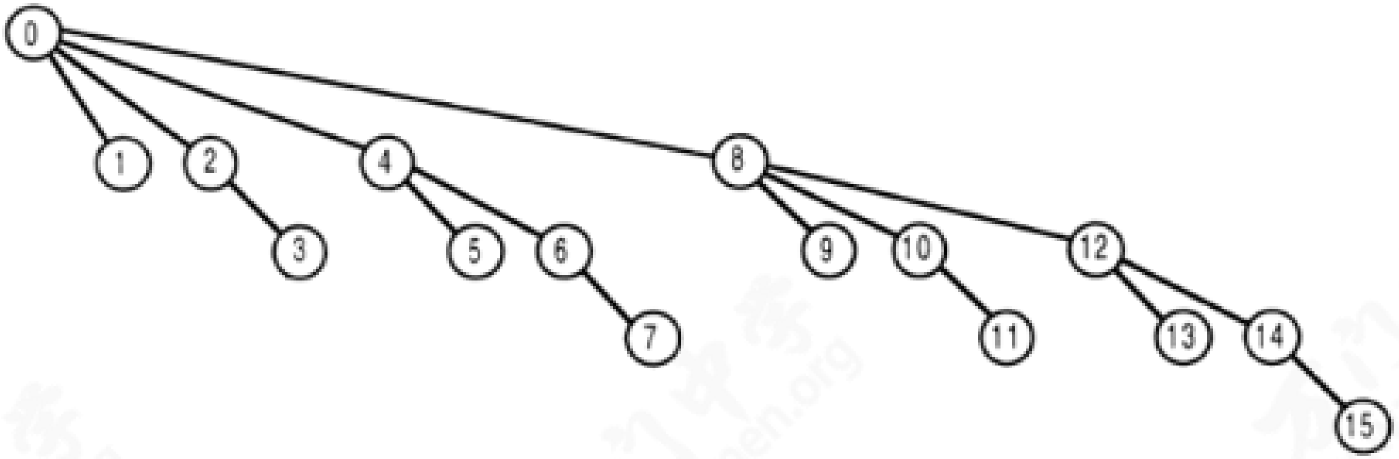


树状数组的查询

根据下图可以得到查询 a[1..i] 前缀和的方法：

```
ans = 0
while (i > 0) ans += sum[i], i -= C(i);
```

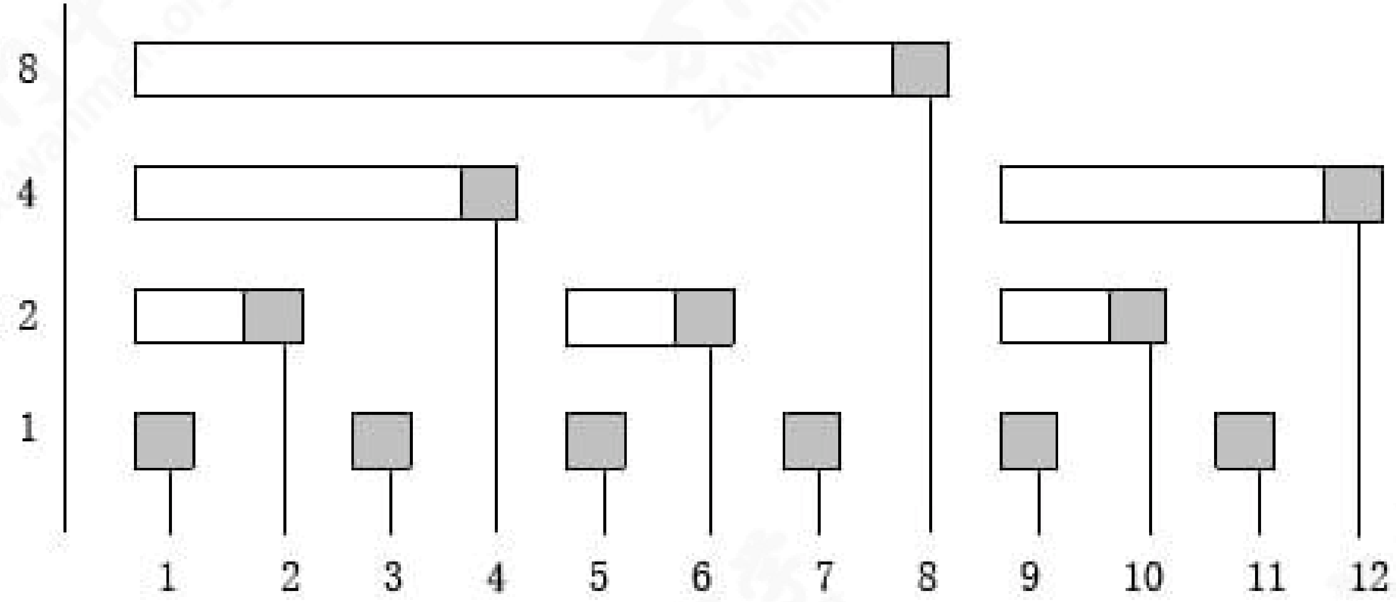
下图为查询树



树状数组的查询

需要相加的项的个数为 i 的二进制表示中包含的 1 的个数。

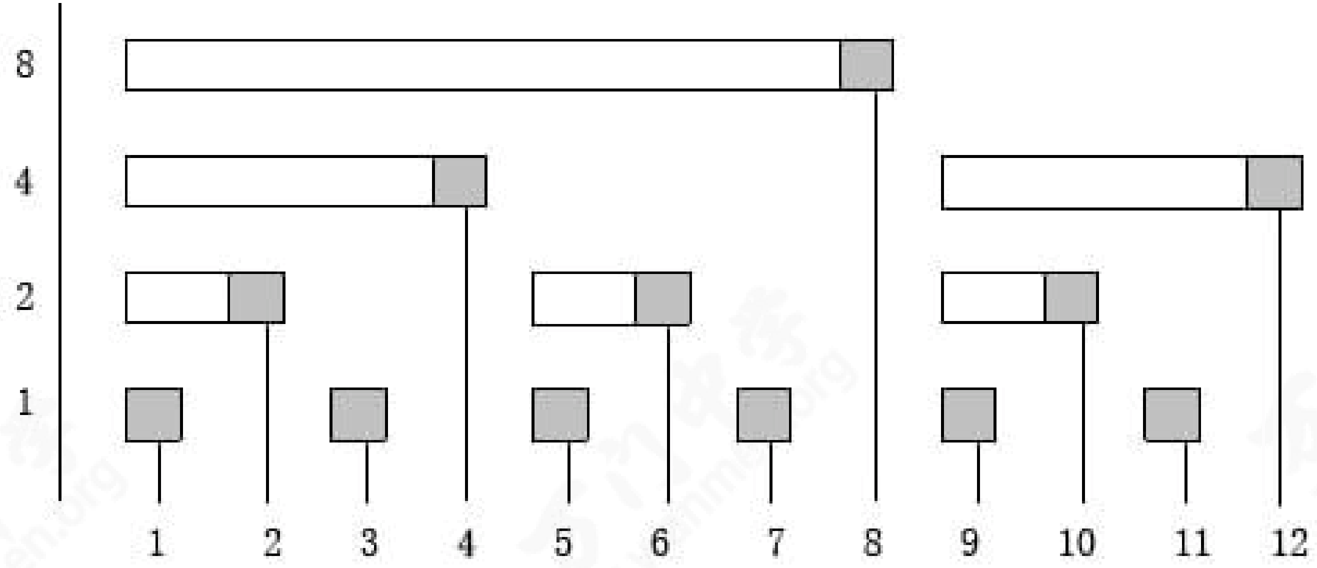
时间复杂度为 $O(\log_2 i)$。



树状数组的修改

当 a[i] 要加上 v 时，我们需要关心有哪些子集和包含了这个需要被修改的元素 a[i]。

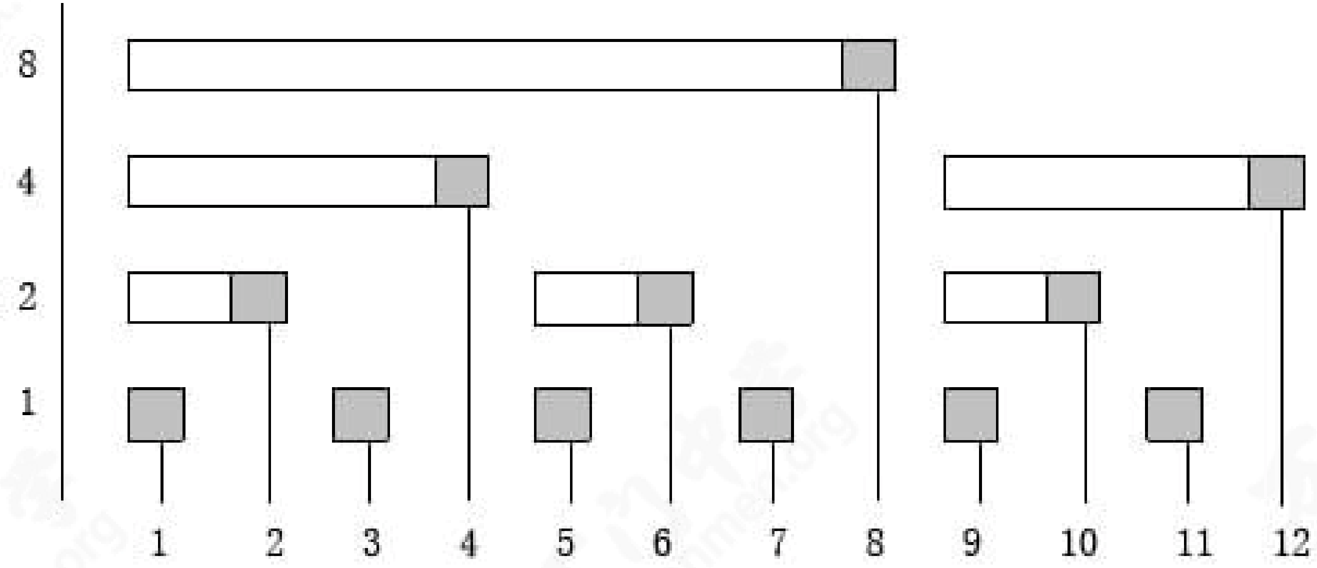
可以发现，每个深色方块上面的那些长方形都代表了包含该元素的子集。



树状数组的修改

与查询不同的是，这里是每一步循环给下标加上 C(i)

```
change(i, v):
    while(i <= n)    sum[i] += v, i += C(i)
```



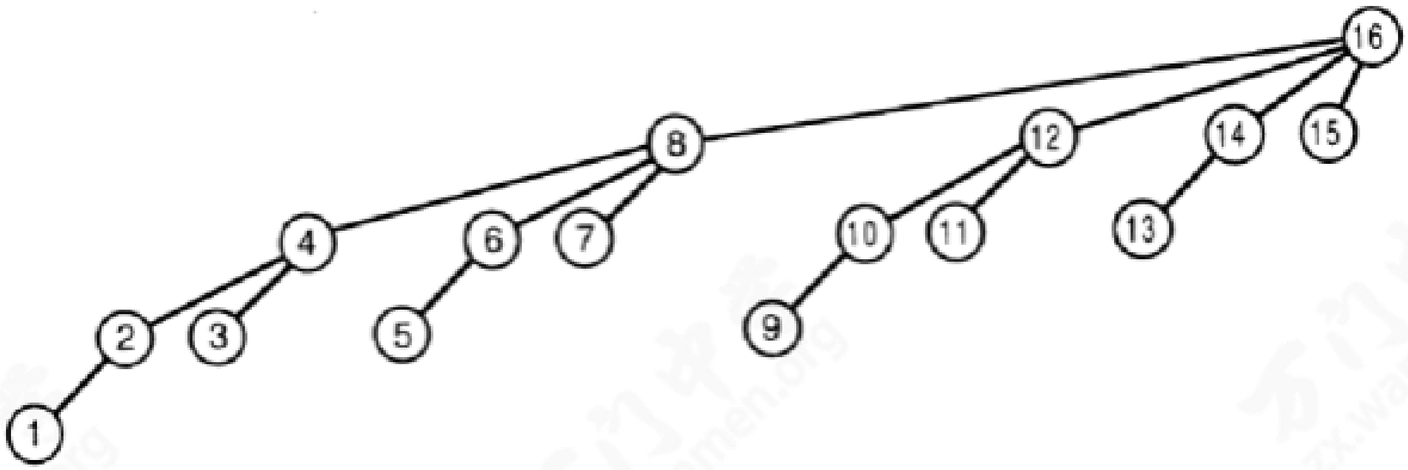
树状数组的修改

与查询不同的是，这里是每一步循环给下标加上 C(i)

change(i, v):

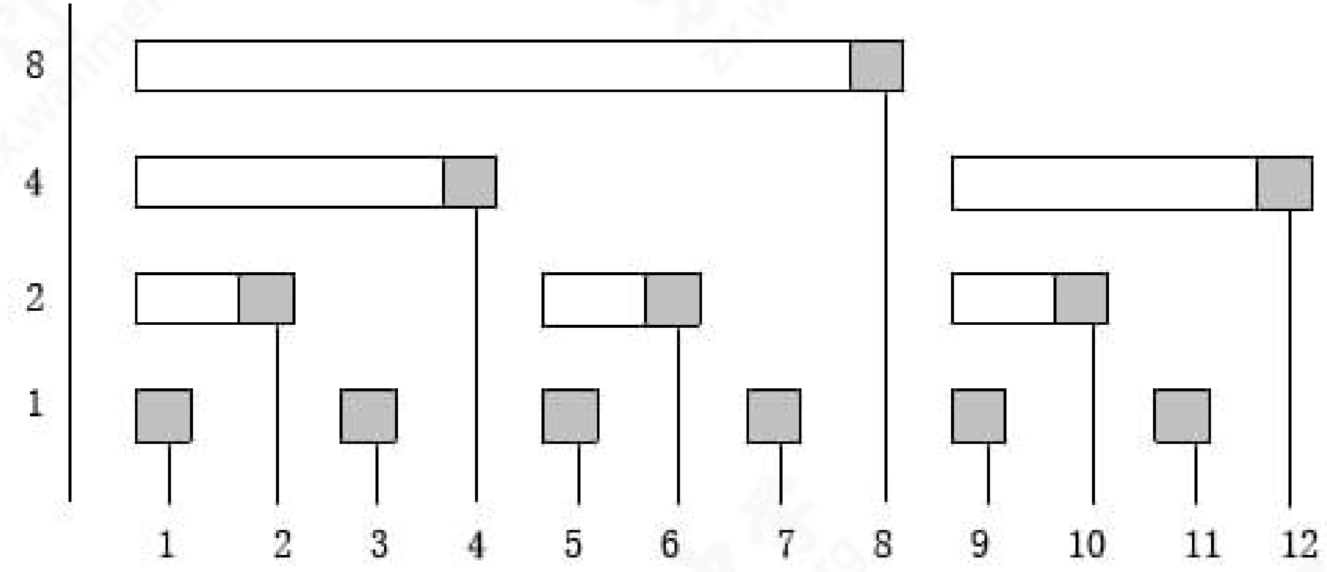
    while(i <= n)   sum[i] += v, i += C(i)

下图是更新树



树状数组的修改

由于树的深度至多是 $\log_2 n$，所以修改操作的时间复杂度也是 $O(\log_2 n)$



树状数组的常用技巧

查询区间和：a[x..y]

只需要查询Query(y) - Query(x - 1)

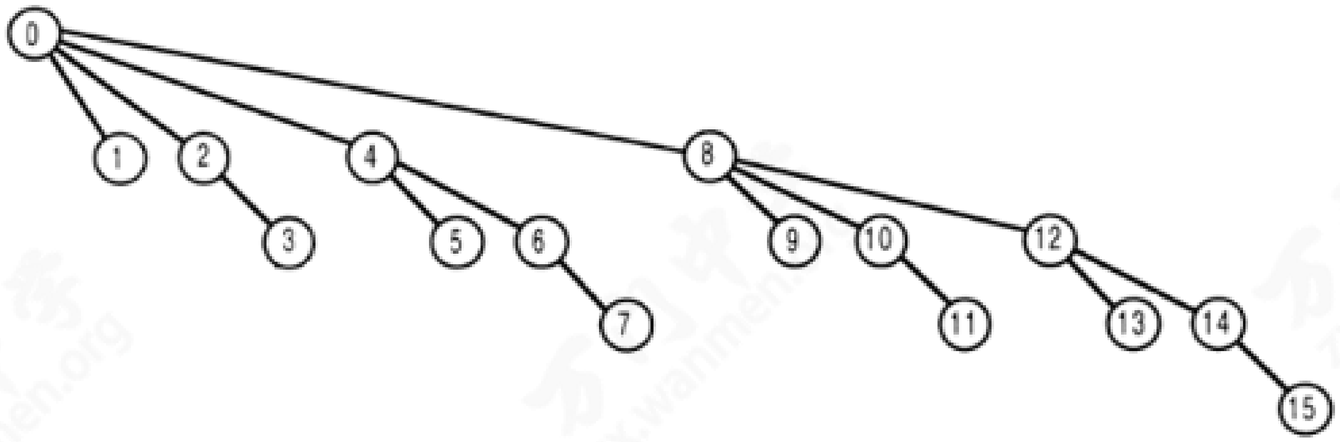
树状数组的常用技巧

单点查询：a[x]

最简单的方法：Query(x) - Query(x - 1)

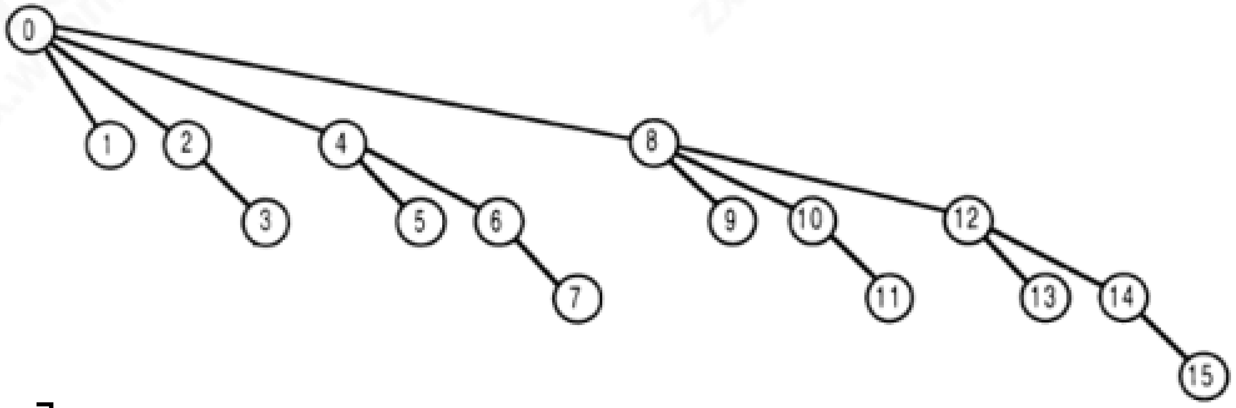
但是要执行两次查询。

观察查询树，a[x] = sum[x] - (Query(x - 1) - Query(LCA(x, x - 1))



树状数组的常用技巧

```
get_value(i):
    ans = sum[i]
    lca = i - C(i)
    i --
    while i != lca:
        ans -= sum[i]
        i -= C[i]
    return ans
```

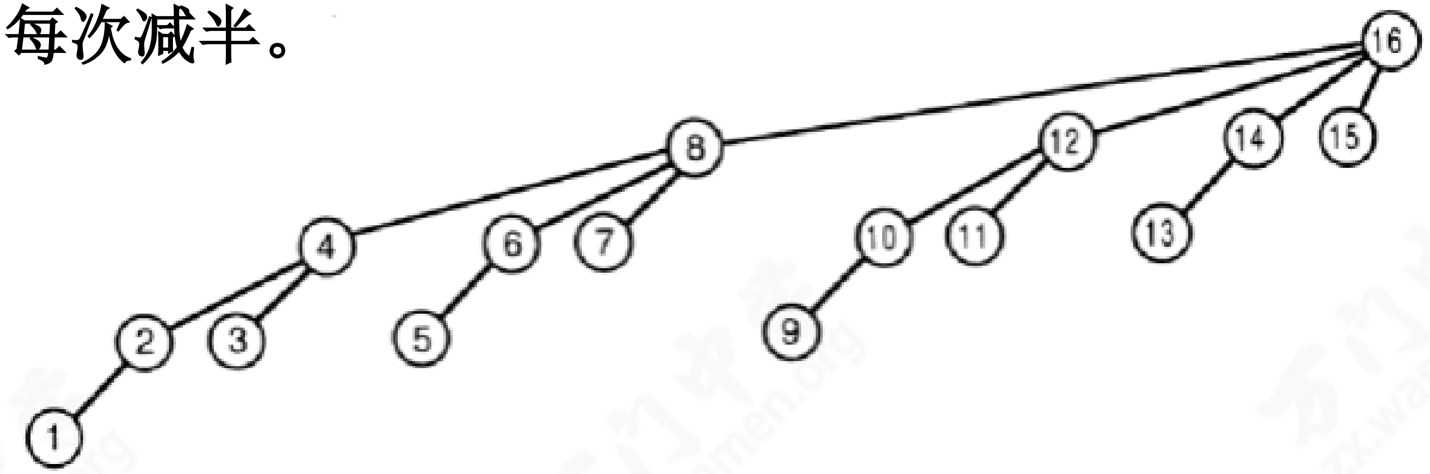


树状数组的常用技巧

假设元素非负，查询某个前缀和对应的前缀下标 i

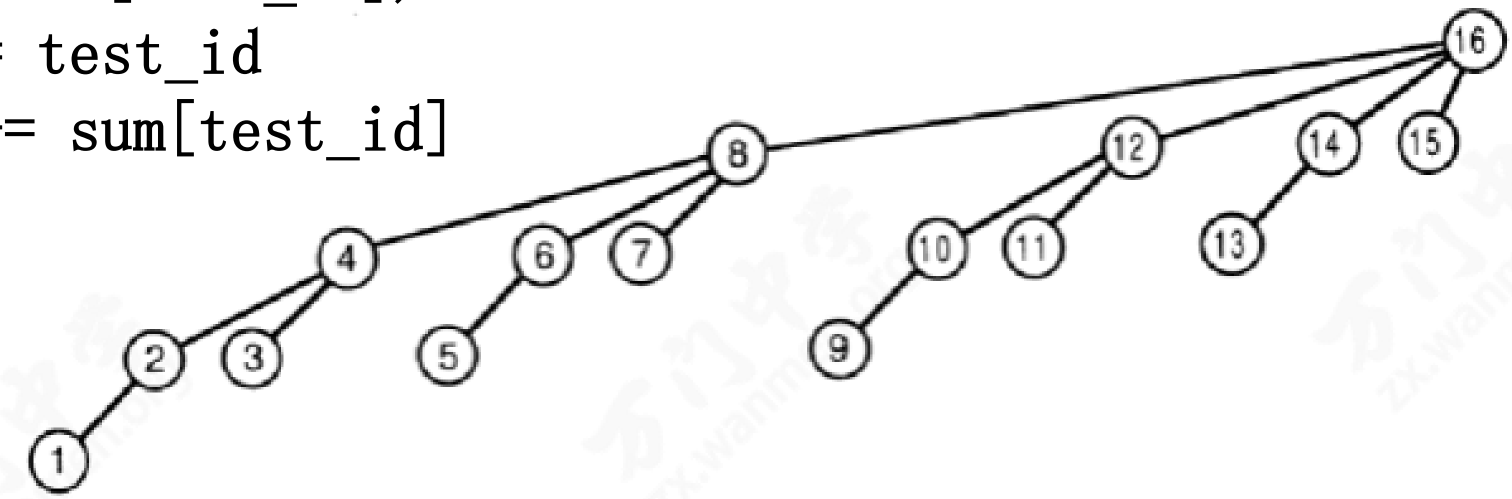
因为下标为 2的幂次的子集包含了从开始的元素到自己的所有元素。

所以我们可以根据更新树进行二分，初始步长为 n，之后每次减半。



树状数组的常用技巧

```
get_index(v):
        i = 0
        len = n
        while len != 0:
                test_id = i + len
                if (v >= sum[test_id]):
                        i = test_id
                        v -= sum[test_id]
                len /= 2
        return i
```



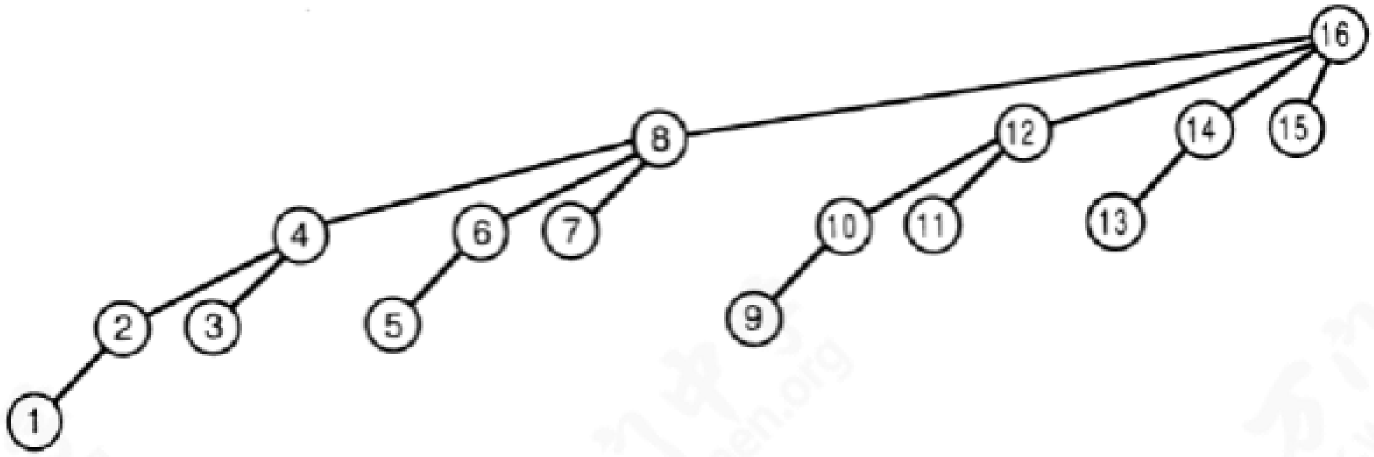
树状数组的常用技巧

初始化树状数组：

朴素做法，一个一个插入：O(n log n)

维护一个前缀和数组 pre[x] = sum(a[1 .. x])

sum[x] = pre[x] - pre[x - C(x)]



下节课再见