树状数组

游坦之老师

1、树状数组课程介绍

树状数组，binary index tree，用于维护前缀信息的结构。

对前缀信息的处理也是非常高效的。

北美常见面试题。

熟练掌握树状数组类似问题的解决，可以加深初学者对于逻辑分层的理解。

2、树状数组问题举例

给定一个整数数组 nums，然后你需要实现两个函数：

update(i, val) 将数组下标为 i 的元素修改为val

sumRange(l, r) 返回数组下标在 [l, r] 区间的元素的和

暴力求解：update时间复杂度O（1）、sumRange时间复杂度O（n）

如果用树状数组来求解呢?

3、树状数组与区间和的联系

树状数组是通过前缀和思想，用来完成单点更新和区间查询的数据结构。它比之线段树，所用空间更小，速度更快。

如何用前缀和求解sumRange(i, j)呢?

那么树状数组具体如何实现单点更新以及区间求和呢?

4、树状数组算法分析

注意：

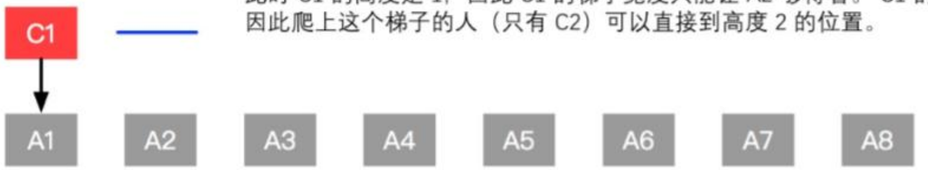
树状数组的下标从 1 开始计数。

定义：

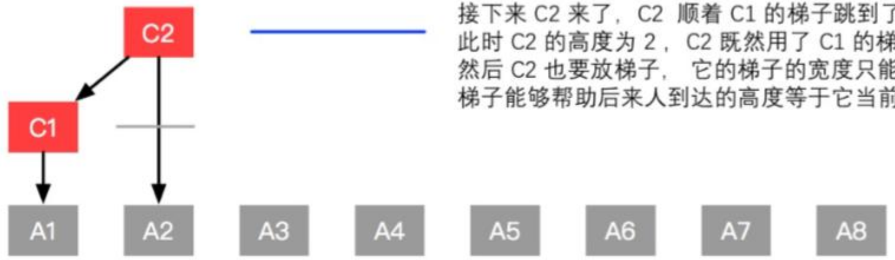
数组 C 是一个对原始数组 A 的预处理数组。

4、树状数组算法分析

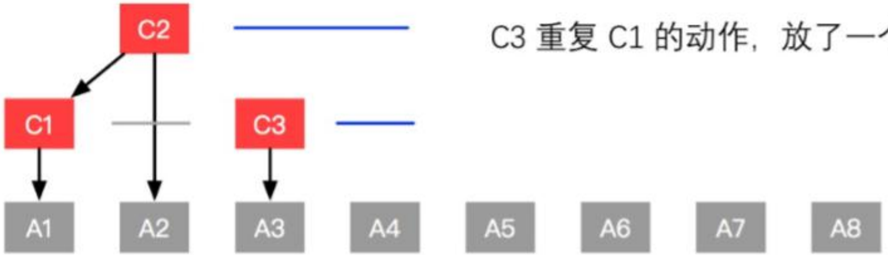
前人栽树，后人乘凉，前人搭梯子，后人跳更高，后人不忘恩。
C1 放置的梯子，梯子的规则是：梯子的宽度和高度相当，梯子能够帮助后人登上的高度也和梯子的高度相当。
此时 C1 的高度是 1，因此 C1 的梯子宽度只能让 A2 够得着。 C1 的梯子能够让别人跳上的高度也和梯子的高度相当。
因此爬上这个梯子的人（只有 C2）可以直接到高度 2 的位置。



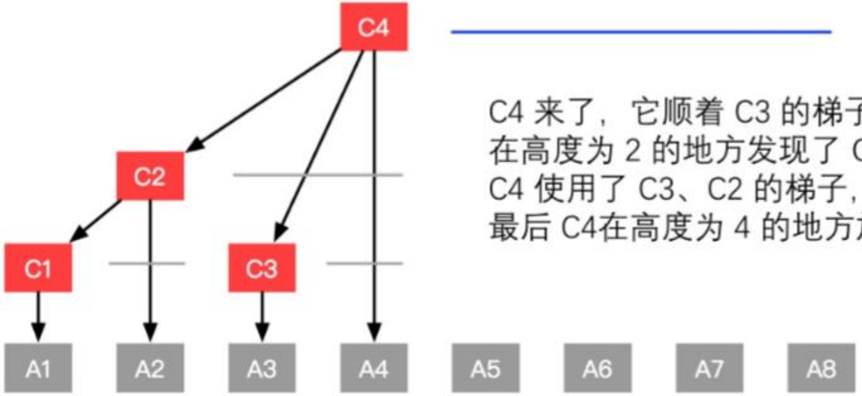
接下来 C2 来了，C2 顺着 C1 的梯子跳到了高度为 2 的地方，在这里放了个梯子，供后面的人使用。
此时 C2 的高度为 2，C2 既然用了 C1 的梯子，就得 “关照” C1，因此，它连接了 C1。
然后 C2 也要放梯子，它的梯子的宽度只能是 2，所以，只有 A3 和 A4 能够得着。
梯子能够帮助后来人到达的高度等于它当前的高度，因此爬上它的人能直接到高度 4。



4、树状数组算法分析

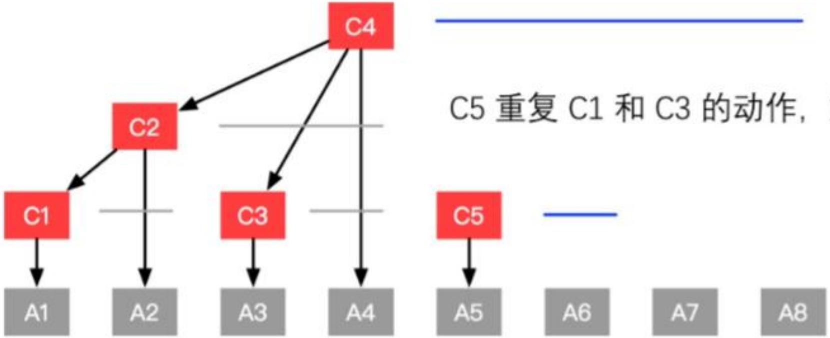


C3 重复 C1 的动作，放了一个宽度为 1 能帮助后来人跳到高度 2 的梯子。

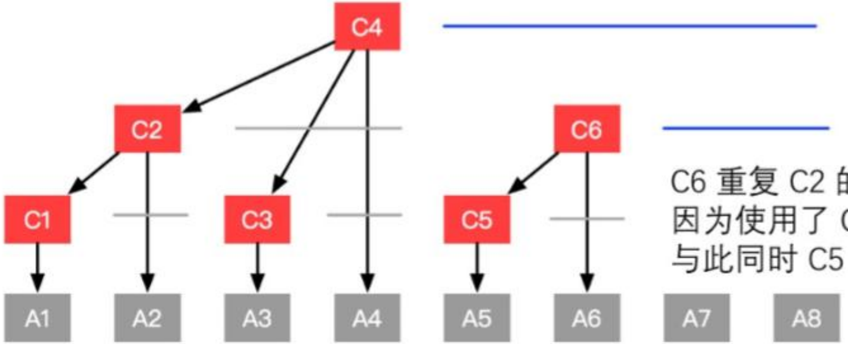


C4 来了，它顺着 C3 的梯子来到了高度为 2 的梯子。
在高度为 2 的地方发现了 C2 的梯子，于是跳到了高度为 4 的地方。
C4 使用了 C3、C2 的梯子，所以要“关照”它们，于是连接 C3 和 C2。
最后 C4在高度为 4 的地方放置了高度为 4 能帮助后来人跳到 8 的梯子。

4、树状数组算法分析

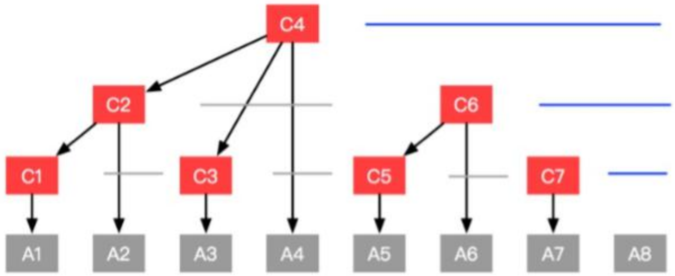


C5 重复 C1 和 C3 的动作，放了一个高度为 1 能帮助后来人跳到 2 的梯子。

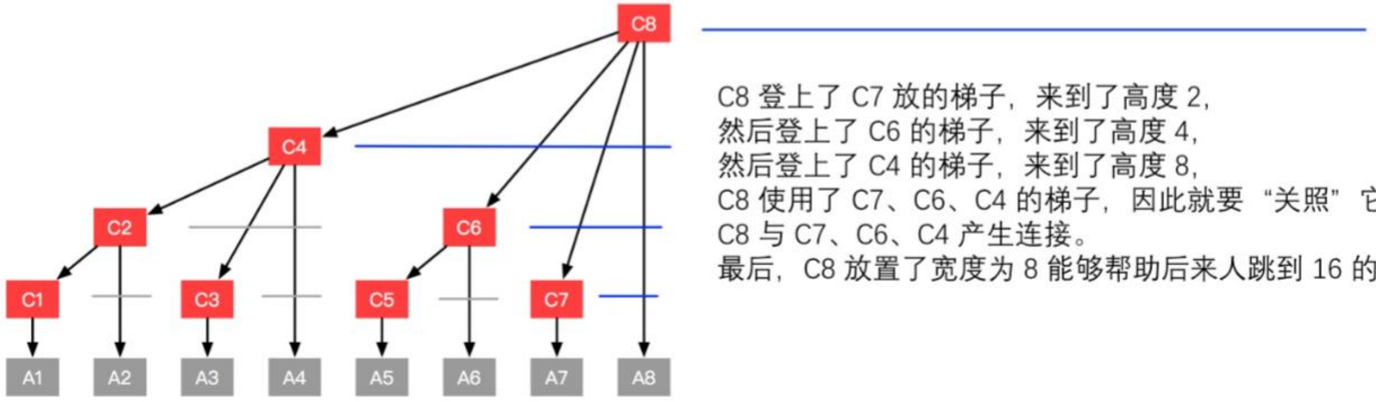


C6 重复 C2 的动作，它登上了 C5 的梯子，来到了高度 2。
因为使用了 C5 的梯子，因此要和 C5 产生连接。
与此同时 C5 放置了宽度为 2，能帮助后来人跳到高度 4 的梯子。

4、树状数组算法分析



C7 重复 C1 、 C3 和 C5 的动作，
放了一个高度为 1 能帮助后来人跳到 2 的梯子。



C8 登上了 C7 放的梯子，来到了高度 2，
然后登上了 C6 的梯子，来到了高度 4，
然后登上了 C4 的梯子，来到了高度 8，
C8 使用了 C7、C6、C4 的梯子，因此就要 “关照” 它们，
C8 与 C7、C6、C4 产生连接。
最后，C8 放置了宽度为 8 能够帮助后来人跳到 16 的梯子。

4、树状数组算法分析

九章算法

数组 C 的索引 i	数组 C 的和定义由数组 A 的哪些元素而来	数组 C 中的元素来自数组 A 的个数
1	C[1] = A[1]	1
2	C[2] = A[1] + A[2]	2
3	C[3] = A[3]	1
4	C[4] = A[1] + A[2] + A[3] + A[4]	4
5	C[5] = A[5]	1
6	C[6] = A[5] + A[6]	2
7	C[7] = A[7]	1
8	C[8] = A[1] + A[2] + A[3] + A[4] + A[5] + A[6] + A[7] + A[8]	8

C[i]来自几个数组A中的元素：取决于i的二进制末尾有几个连续的0。比如有k个0，那么C[i]来自2^k个A中的元素。

4、树状数组算法分析

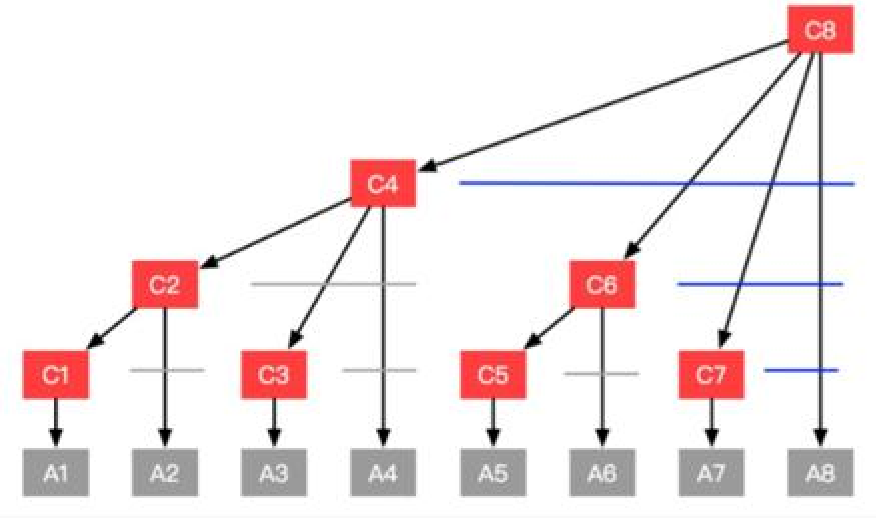
九章算法

索引 i	i 的二进制表示	k	2^k	数组 C 的定义由数组 A 的哪些元素而来
1	0000 0001	0	1	C[1] = A[1]
2	0000 0010	1	2	C[2] = A[1] + A[2]
3	0000 0011	0	1	C[3] = A[3]
4	0000 0100	2	4	C[4] = A[1] + A[2] + A[3] + A[4]
5	0000 0101	0	1	C[5] = A[5]
6	0000 0110	1	2	C[6] = A[5] + A[6]
7	0000 0111	0	1	C[7] = A[7]
8	0000 1000	3	8	C[8] = A[1] + A[2] + A[3] + A[4] + A[5] + A[6] + A[7] + A[8]

定义一个lowbit函数：lowbit(i) = 2 ^ k。

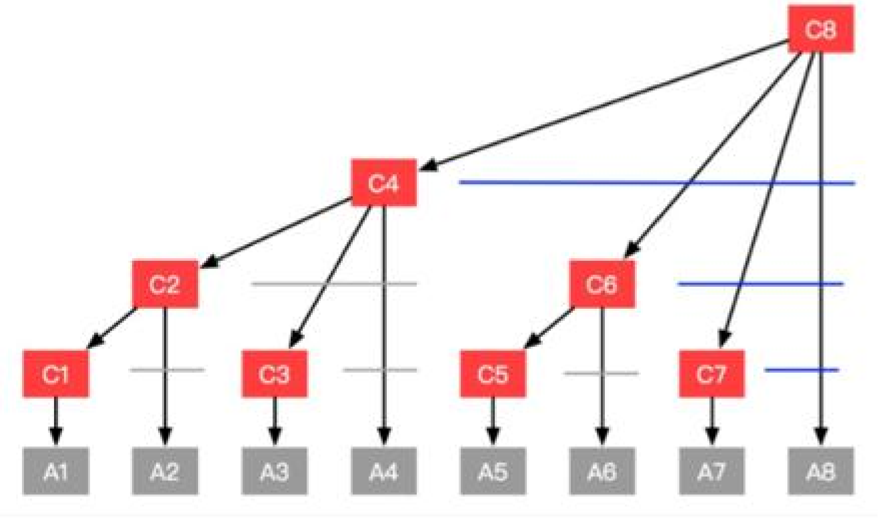
4、树状数组算法分析

根据lowbit函数，可以知道C[i]代表几个A中元素相加以及i的父亲在哪。



5、树状数组的构建

先都初始化为0，然后再更新为相应的值 == 构建



5、树状数组的构建

进行区间和查询 == 进行两次前缀和查询

sum(i) = sum{ A[j] | 1 <= j <= i }

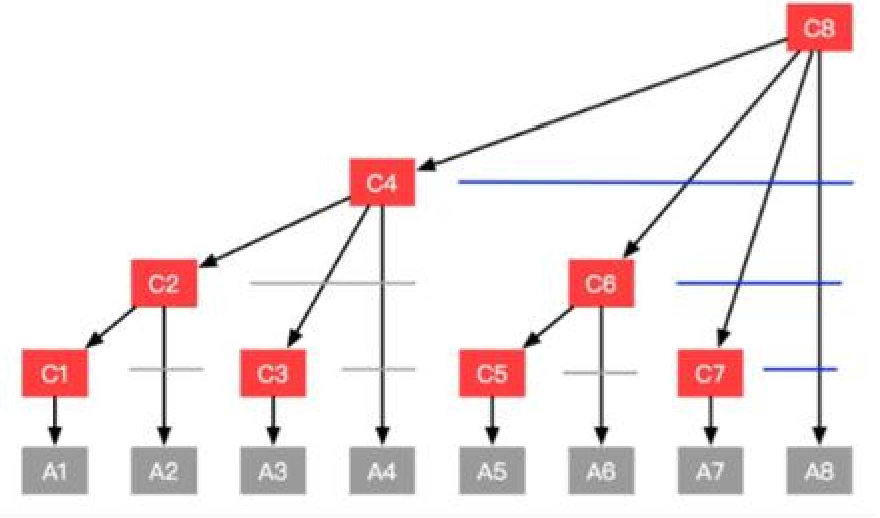
= A[1] + A[2] + ... + A[i]

= A[1] + A[2] + A[i-2^k] + A[i-2^k+1] + ... + A[i]

= A[1] + A[2] + A[i-2^k] + C[i]

= sum(i - 2^k) + C[i]

= sum(i - lowbit(i)) + C[i]



5、树状数组的构建

神秘的lowbit!

位运算&： 1&1=1 0&1=0 1&0=0 0&0=0

3 & 11 = ？

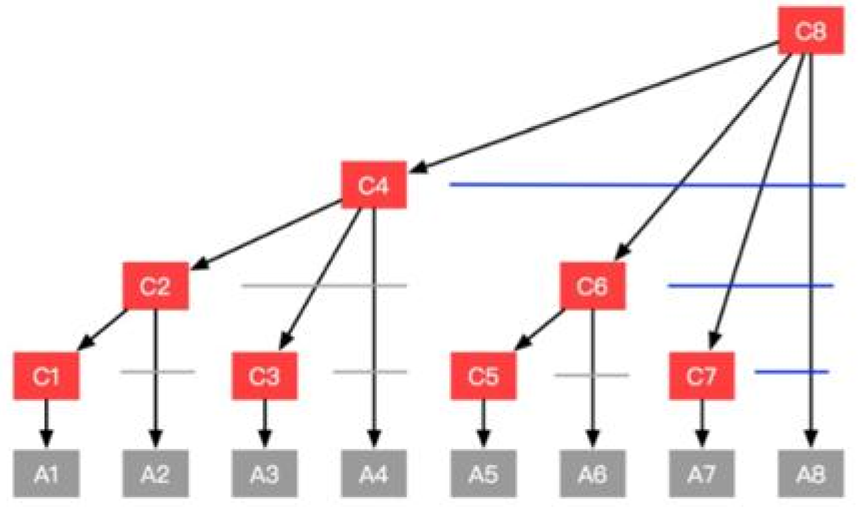
正数和负数的二进制。

num & (-num) == 2 ^ k

6、树状数组算法程序实现——lintcode 840 range-sum

```java
private int[] arr, bit;
public NumArray(int[] nums) {
    arr = new int[nums.length];
    bit = new int[nums.length + 1];

    for (int i = 0; i < nums.length; i++) {
        update(i, nums[i]);
    }
}
public void update(int index, int val) {
    int delta = val - arr[index];
    arr[index] = val;
    for (int i = index + 1; i <= arr.length; i = i + lowbit(i)) {
        bit[i] += delta;
    }
}
public int sumRange(int left, int right) {
    return getPrefixSum(right) - getPrefixSum(left - 1);
}
public int getPrefixSum(int index) {
    int sum = 0;
    for (int i = index + 1; i > 0; i = i - lowbit(i)) {
        sum += bit[i];
    }
    return sum;
}
private int lowbit(int x) {
    return x & (-x);
}
```



Q & A

九章算法

扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: ninechapter
知乎专栏： http://zhuanlan.zhihu.com/jiuzhang
微博: http://www.weibo.com/ninechapter
官网: www.jiuzhang.com