树状数组问题

什么是树状数组：
对于数组a，存在一个数组c，满足：

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

且：

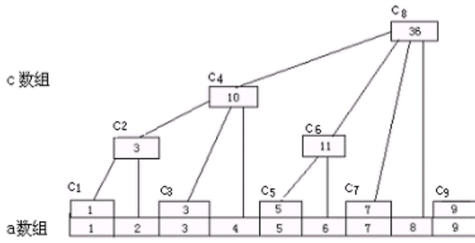
- k为i在二进制下末尾0的个数；
- i 从1开始计算；

那么就把C数组称为A数组的树状数组。

现在的问题就是知道，如何求得k?
在这里直接给出结果，不进行数学推导: k = lowBits(i) = i & (-i)
所以可以得到树状数组的递推公式：

$$C[i] = A[i - lowBits(i) + 1] + \ldots + A[i]$$

如下图：



树状数组的好处在于能快速求出区间和。比如说求得i, j之间和，因为

$$sum[i] = A[1] + A[2] + \ldots + A[i]$$

所以$[i, j]$之间的和是：

$$sum = sum[j] - sum[i - 1]$$

因为树状数组是由原始数组构成的，因此它也必定可以用来求出区间之和。所以存在公式：

$$sum(k) = C[n_1] + C[n_2] + \ldots + C[n_m]$$

其中$n_m = k$**, 且**

$$n_{i-1} = n_i - lowBits(n_i)$$

所以可以得到，

$$sum(6) = C[6] + C[4]\text{ , 其中}$$

$$C[6] = A[5] + A[6];$$

$$C[4] = A[1] + A[2] + A[3] + A[4]$$

求和的操作大体就是这样。

如果要对数组a中的某个位置的元素进行更新，那么树状数组c应该如何更新。
根据上述的描述，如果数组a中位置i元素更新，那么对于树状数组中：

$$C[n_1], C[n_2], \ldots C[n_m]$$

都需要更新，其中：

$$n_1 = i$$

$$n_{i+1} = n_i + lowBits(n_i)$$

这就是树状数组的更新操作。

下述问题来自于Leetcode 307: Range Sum Query - Mutable.
看到这个题目，第一反应就是进行预处理，维护一个累加数组然后再调用，这种方法对于immutable题目是合理的，但是对于这个题目来说，由于牵扯到更新操作，因此操作超时。
现在利用树状数组处理。

```
public class NumArray {
    int[] nums;
    int[] tree;
    int size;
    public NumArray(int[] nums) {
        this.size = nums.length;
        this.nums = nums;
        this.tree = new int[size + 1];
        for (int i = 0; i < size; i++) {
            updateTree(i, nums[i]);
        }
    }

    void updateTree(int i, int val) {
        i = i + 1;
        while (i <= size) {
            tree[i] += val;
            i = i + lowBits(i);
        }
    }


    void update(int i, int val) {
        updateTree(i, val - nums[i]);
        nums[i] = val;
    }

    public int sumRange(int i, int j) {
        if (i == 0) return getSum(j);
        return getSum(j) - getSum(i - 1);
    }

    private int getSum(int n) {
        n = n + 1;
        int sum = 0;
        while (n > 0) {
            sum += tree[n];
            n = n - lowBits(n);
        }
        return sum;
    }
    private int lowBits(int n) {
        return n & (-n);
    }
}


// Your NumArray object will be instantiated and called as such:
// NumArray numArray = new NumArray(nums);
// numArray.sumRange(0, 1);
// numArray.update(1, 10);
// numArray.sumRange(1, 2);
```

AC通过。