Segment Tree

#G面试准备

- 每个节点里存储着区间和max
- 对数组[1,3,4,2]
- 叶子节点的区间就是[0,0] [1,1] [2,2] [3,3] max值分别为1 3 4 2
- 求数组区间内最大值 很快得到答案

线段树查询

Segment Tree Query

O(logn)

查找操作: 节点区间和要查找区间的关系 四种情况:

1. 节点区间包含查找区间–>查找区间递归向下
2. 节点区间不相交于查找区间 ->查找区间停止搜索
3. 节点区间相交不包含于查找区间->查找区间分裂成两 段区间，一段于被节点区间包含，另一段不相交
4. 节点区间相等于查找区间–> 返回值查找的结果

画图比较root的起止和当前要查找的起止

```java
public class Solution {
    /*
     * @param root: The root of segment tree.
     * @param start: start value.
     * @param end: end value.
     * @return: The maximum number in the interval [start, end]
     */
    public int query(SegmentTreeNode root, int start, int end) {
        // write your code here
        if (start == root.start && end == root.end) {
            return root.max;
        }
        int leftmax = Integer.MIN_VALUE;
        int rightmax = Integer.MIN_VALUE;
        int mid = (root.start + root.end) / 2;
        // 左区
        if (start <= mid) {
            if (mid < end) { // 分裂
                leftmax = query(root.left, start, mid);
            } else { // 包含
                leftmax = query(root.left, start, end);
            }
        }
        if (mid < end) {
            if (start <= mid) { // 分裂
                rightmax = query(root.right, mid + 1, end);
            } else {
                rightmax = query(root.right, start, end);
            }
        }
        return Math.max(leftmax, rightmax);
    }
}
```

建立

Build tree:

字诀:

自上而下递归分裂

- 建立当前节点
- 得到mid
- start, mid | mid+1, end 左右递归下去建立

自下而上回溯更新

- 建好之后从下面开始往上更新max值
  - 左区间和右区间两个max的最大值

```java
public class Solution {
    /*
     * @param A: a list of integer
     * @return: The root of Segment Tree
     */
    public SegmentTreeNode build(int[] A) {
        // write your code here
        if (A == null || A.length == 0) {
            return null;
        }
        int n = A.length - 1;
        SegmentTreeNode root = helper(A, 0, n);
        return root;
    }
    private SegmentTreeNode helper(int[] A, int start, int end) {
        if (start > end) {
            return null;
        }
        int mid = (start + end) / 2;
        SegmentTreeNode root = new SegmentTreeNode(start, end, A[start]);
        if (start == end) {
            return root;
        }
        root.left = helper(A, start, mid);
        root.right = helper(A, mid + 1, end);
        // 要记得加判断是否null再去更新max
        if (root.left != null) {
            root.max = Math.max(root.max, root.left.max);
        }
        if (root.right != null) {
            root.max = Math.max(root.max, root.right.max);
        }
        return root;
    }
}
```

建树时间复杂度 O(n)

修改

O(logn)

自上而下查询节点 找到[0,0]区间节点

自下而上更新max

```java
public class Solution {
    /*
     * @param root: The root of segment tree.
     * @param index: index.
     * @param value: value
     * @return:
     */
    public void modify(SegmentTreeNode root, int index, int value) {
        // write your code here
        if (root.start == index && root.end == index) {
            root.max = value;
            return;
        }
        int mid = (root.start + root.end) / 2;
        if (root.start <= index && index <= mid) {
            modify(root.left, index, value);
        }
        if (mid < index && index <= root.end) {
            modify(root.right, index, value);
        }
        root.max = Math.max(root.left.max, root.right.max);
    }
}
```

- 判断是进入左边还是右边 找[index, index]这个node
- 左右子节点都会在的

Range Sum Query - Immutable

Range Sum Query - Immutable - LeetCode

- 建 Prefix Sum 数组
- 要比原数组size大1 0的位置存个0
- prefixSum数组里的下标表示加到第几个数
- 所以要求从i到j的和 就是从第i+1个数加到第j+1个数
- prefixSum[j+1] - prefixSum[i] 后面是i 这样第i+1个数还包括在里面

Range Sum Query - Mutable

Range Sum Query - Mutable - LeetCode

- 线段树实现
- 需要用到以上所有三个操作的实现 build query update
- 唯一一次错：查询的时候忘记在开头加上退出递归条件：如果node的range和查询的range一模一样 输出当前node存的sum

```
if (node.start == start && node.end == end) {
      return node.sum;
}
```

- 每一个函数都要写一个private的函数来做helper 因为比提供的函数声明还多需要传入一个当前node作为参数