207. 区间求和 II

在类的构造函数中给一个整数数组，实现两个方法 `query(start, end)` 和 `modify(index, value)`:

- 对于 query(*start*, *end*), 返回数组中下标 *start* 到 *end* 的 和。
- 对于 modify(*index*, *value*), 修改数组中下标为 *index* 上的数为 *value*.

注意事项

在做此题前，建议先完成以下三题：

- 线段树的构造
- 线段树的查询
- 线段树的修改

样例，给定数组 A = `[1,2,7,8,5]`.

- `query(0, 2)`, 返回 `10`.
- `modify(0, 4)`, 将 A[0] 修改为 *4*.
- `query(0, 1)`, 返回 `6`.
- `modify(2, 1)`, 将 A[2] 修改为 *1*.
- `query(2, 4)`, 返回 `14`.

线段树的构造、修改、查询

```
public class Solution {
  class SegmentTreeNode {
      public int start, end;
      public int sum;
      public SegmentTreeNode left, right;
      public SegmentTreeNode(int start, int end, int sum) {
            this.start = start;
            this.end = end;
            this.sum = sum;
            this.left = this.right = null;
      }
  }
  SegmentTreeNode root;
  public Solution(int[] A) {
      root = build(0, A.length - 1, A );
  }
  /*
   * @return: The sum from start to end
   */
  public long query(int start, int end) {
      return querySegmentTree(root, start, end);
  }

  public void modify(int index, int value) {
      modifySegmentTree(root, index, value);
  }

```

```
    // query
    public long querySegmentTree(SegmentTreeNode root, int start, int end){
        // if( null == root || start > end || start > root.end || end < root.start )
        //     return 0;
        if( start == root.start && root.end == end )
            return root.sum;
        int mid = (root.start + root.end)/2;
        long leftsum = 0, rightsum = 0;
        if( start <= mid )
            leftsum = querySegmentTree(root.left, start, Math.min(mid,end) );
        if( mid < end )
            rightsum = querySegmentTree(root.right, Math.max(mid+1,start), end);
        return leftsum + rightsum;
    }

    // modify
    private void modifySegmentTree(SegmentTreeNode root, int index, int value){
        if( null == root || index < root.start || index > root.end)
            return;
        if( index == root.start && index == root.end ){
            root.sum = value;
            return;
        }
        int mid = (root.start + root.end)/2;
        if( index <= mid ){
            modifySegmentTree(root.left, index, value);
        }else{
            modifySegmentTree(root.right, index, value);
        }
        root.sum = root.left.sum + root.right.sum;
    }

    // build
    private SegmentTreeNode build(int start, int end, int[] A){
        if( start > end )
            return null;

        SegmentTreeNode node = new SegmentTreeNode( start, end, A[start] );
        if( start == end )
            return node;

        int mid = (start + end)/2;
        node.left = build(start, mid, A);
        node.right = build(mid + 1, end, A);
        node.sum = node.left.sum + node.right.sum;
        return node;
    }
}
```