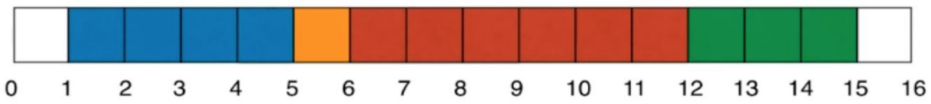
数据结构与算法 10- 线段树

笔记本： 我的笔记
创建时间： 2020/10/9 22:25 **更新时间：** 2020/10/9 22:50
作者： liuhouer
标签： 算法

为什么要使用线段树

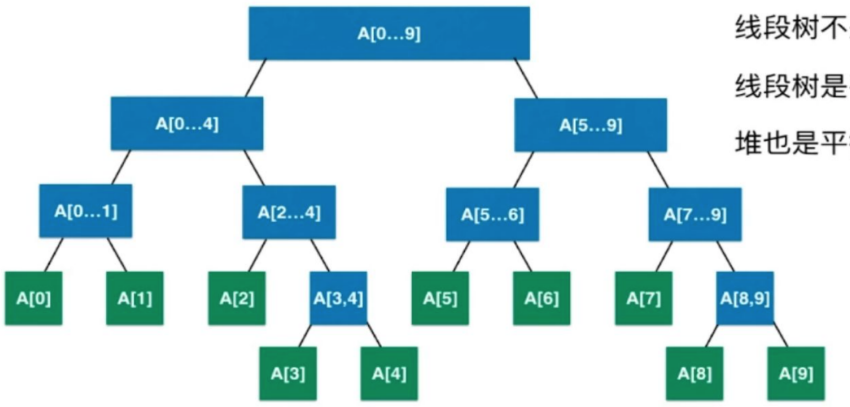
最经典的线段树问题：区间染色

有一面墙，长度为n，每次选择一段儿墙进行染色



m次操作后，我们可以看见多少种颜色？

什么是线段树



线段树不是完全二叉树

线段树是平衡二叉树

堆也是平衡二叉树

为什么要使用线段树

另一类经典问题：区间查询

32	26	17	55	72	19	8	46	22	68	28	33	62	92	53	16
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

查询一个区间[i, j]的最大值，最小值，或者区间数字和

实质：基于区间的统计查询

适应场景：求解区间内容，或者统计区间内容，动态的区间内容

1.实现线段树

```
public class SegmentTree<E> {


    private E[] tree;
    private E[] data;
    private Merger<E> merger;


    public SegmentTree(E[] arr, Merger<E> merger){


        this.merger = merger;


        data = (E[])new Object[arr.length];
        for(int i = 0 ; i < arr.length ; i ++)
            data[i] = arr[i];


        tree = (E[])new Object[4 * arr.length];
        buildSegmentTree(0, 0, arr.length - 1);
    }


    // 在treeIndex的位置创建表示区间[l...r]的线段树
    private void buildSegmentTree(int treeIndex, int l, int r){


        if(l == r){
            tree[treeIndex] = data[l];
            return;
        }


        int leftTreeIndex = leftChild(treeIndex);
        int rightTreeIndex = rightChild(treeIndex);


        // int mid = (l + r) / 2;
        int mid = l + (r - l) / 2;
        buildSegmentTree(leftTreeIndex, l, mid);
        buildSegmentTree(rightTreeIndex, mid + 1, r);
```

```
        tree[treeIndex] = merger.merge(tree[leftTreeIndex],
tree[rightTreeIndex]);
    }


    public int getSize(){
        return data.length;
    }


    public E get(int index){
        if(index < 0 || index >= data.length)
            throw new IllegalArgumentException("Index is illegal.");
        return data[index];
    }


    // 返回完全二叉树的数组表示中，一个索引所表示的元素的左孩子节点的索引
    private int leftChild(int index){
        return 2*index + 1;
    }


    // 返回完全二叉树的数组表示中，一个索引所表示的元素的右孩子节点的索引
    private int rightChild(int index){
        return 2*index + 2;
    }


    // 返回区间[queryL, queryR]的值
    public E query(int queryL, int queryR){


        if(queryL < 0 || queryL >= data.length ||
                queryR < 0 || queryR >= data.length || queryL > queryR)
            throw new IllegalArgumentException("Index is illegal.");


        return query(0, 0, data.length - 1, queryL, queryR);
    }


    // 在以treeIndex为根的线段树中[l...r]的范围里，搜索区间[queryL...queryR]的值
    private E query(int treeIndex, int l, int r, int queryL, int queryR){


        if(l == queryL && r == queryR)
            return tree[treeIndex];


        int mid = l + (r - l) / 2;
        // treeIndex的节点分为[l...mid]和[mid+1...r]两部分


        int leftTreeIndex = leftChild(treeIndex);
        int rightTreeIndex = rightChild(treeIndex);
        if(queryL >= mid + 1)
            return query(rightTreeIndex, mid + 1, r, queryL, queryR);
        else if(queryR <= mid)
            return query(leftTreeIndex, l, mid, queryL, queryR);


        E leftResult = query(leftTreeIndex, l, mid, queryL, mid);
        E rightResult = query(rightTreeIndex, mid + 1, r, mid + 1, queryR);
        return merger.merge(leftResult, rightResult);
    }


    // 将index位置的值，更新为e
    public void set(int index, E e){


        if(index < 0 || index >= data.length)
            throw new IllegalArgumentException("Index is illegal");
```

```
        data[index] = e;
        set(0, 0, data.length - 1, index, e);
    }


    // 在以treeIndex为根的线段树中更新index的值为e
    private void set(int treeIndex, int l, int r, int index, E e){


        if(l == r){
            tree[treeIndex] = e;
            return;
        }


        int mid = l + (r - l) / 2;
        // treeIndex的节点分为[l...mid]和[mid+1...r]两部分


        int leftTreeIndex = leftChild(treeIndex);
        int rightTreeIndex = rightChild(treeIndex);
        if(index >= mid + 1)
            set(rightTreeIndex, mid + 1, r, index, e);
        else // index <= mid
            set(leftTreeIndex, l, mid, index, e);


        tree[treeIndex] = merger.merge(tree[leftTreeIndex],
tree[rightTreeIndex]);
    }


    @Override
    public String toString(){
        StringBuilder res = new StringBuilder();
        res.append('[');
        for(int i = 0 ; i < tree.length ; i ++){
            if(tree[i] != null)
                res.append(tree[i]);
            else
                res.append("null");


            if(i != tree.length - 1)
                res.append(", ");
        }
        res.append(']');
        return res.toString();
    }
}
```

2.定义一个merge接口，实现对区间的操作\求和\max\min\等等

```
public interface Merger<E> {


    E merge(E a, E b);

}
```

3.实现leetcode代码

```
/// Leetcode 307. Range Sum Query - Mutable
/// https://leetcode.com/problems/range-sum-query-mutable/description/
class NumArray {
```

```
    private SegmentTree<Integer> segTree;
    public NumArray(int[] nums) {


        if(nums.length != 0){
            Integer[] data = new Integer[nums.length];
            for(int i = 0 ; i < nums.length ; i ++)
                data[i] = nums[i];
            segTree = new SegmentTree<>(data, (a, b) -> a + b);
        }
    }


    public void update(int i, int val) {
        if(segTree == null)
            throw new IllegalArgumentException("Error");
        segTree.set(i, val);
    }


    public int sumRange(int i, int j) {
        if(segTree == null)
            throw new IllegalArgumentException("Error");
        return segTree.query(i, j);
    }
}
```

4.动态的线段树求和

```
class NumArrayComplete {


    private interface Merger<E> {
        E merge(E a, E b);
    }


    private class SegmentTree<E> {


        private E[] tree;
        private E[] data;
        private Merger<E> merger;


        public SegmentTree(E[] arr, Merger<E> merger){


            this.merger = merger;


            data = (E[])new Object[arr.length];
            for(int i = 0 ; i < arr.length ; i ++)
                data[i] = arr[i];


            tree = (E[])new Object[4 * arr.length];
            buildSegmentTree(0, 0, arr.length - 1);
        }


        // 在treeIndex的位置创建表示区间[l...r]的线段树
        private void buildSegmentTree(int treeIndex, int l, int r){


            if(l == r){
                tree[treeIndex] = data[l];
                return;
            }


            int leftTreeIndex = leftChild(treeIndex);
```

```
            int rightTreeIndex = rightChild(treeIndex);


            // int mid = (l + r) / 2;
            int mid = l + (r - l) / 2;
            buildSegmentTree(leftTreeIndex, l, mid);
            buildSegmentTree(rightTreeIndex, mid + 1, r);


            tree[treeIndex] = merger.merge(tree[leftTreeIndex],
tree[rightTreeIndex]);
        }


        public int getSize(){
            return data.length;
        }


        public E get(int index){
            if(index < 0 || index >= data.length)
                throw new IllegalArgumentException("Index is illegal.");
            return data[index];
        }


        // 返回完全二叉树的数组表示中，一个索引所表示的元素的左孩子节点的索引
        private int leftChild(int index){
            return 2*index + 1;
        }


        // 返回完全二叉树的数组表示中，一个索引所表示的元素的右孩子节点的索引
        private int rightChild(int index){
            return 2*index + 2;
        }


        // 返回区间[queryL, queryR]的值
        public E query(int queryL, int queryR){


            if(queryL < 0 || queryL >= data.length ||
                    queryR < 0 || queryR >= data.length || queryL > queryR)
                throw new IllegalArgumentException("Index is illegal.");


            return query(0, 0, data.length - 1, queryL, queryR);
        }


        // 在以treeIndex为根的线段树中[l...r]的范围里，搜索区间[queryL...queryR]
的值
        private E query(int treeIndex, int l, int r, int queryL, int queryR){


            if(l == queryL && r == queryR)
                return tree[treeIndex];


            int mid = l + (r - l) / 2;
            // treeIndex的节点分为[l...mid]和[mid+1...r]两部分


            int leftTreeIndex = leftChild(treeIndex);
            int rightTreeIndex = rightChild(treeIndex);
            if(queryL >= mid + 1)
                return query(rightTreeIndex, mid + 1, r, queryL, queryR);
            else if(queryR <= mid)
                return query(leftTreeIndex, l, mid, queryL, queryR);


            E leftResult = query(leftTreeIndex, l, mid, queryL, mid);
            E rightResult = query(rightTreeIndex, mid + 1, r, mid + 1,
queryR);
```

```
            return merger.merge(leftResult, rightResult);
        }


        // 将index位置的值，更新为e
        public void set(int index, E e){


            if(index < 0 || index >= data.length)
                throw new IllegalArgumentException("Index is illegal");


            data[index] = e;
            set(0, 0, data.length - 1, index, e);
        }


        // 在以treeIndex为根的线段树中更新index的值为e
        private void set(int treeIndex, int l, int r, int index, E e){


            if(l == r){
                tree[treeIndex] = e;
                return;
            }


            int mid = l + (r - l) / 2;
            // treeIndex的节点分为[l...mid]和[mid+1...r]两部分


            int leftTreeIndex = leftChild(treeIndex);
            int rightTreeIndex = rightChild(treeIndex);
            if(index >= mid + 1)
                set(rightTreeIndex, mid + 1, r, index, e);
            else // index <= mid
                set(leftTreeIndex, l, mid, index, e);


            tree[treeIndex] = merger.merge(tree[leftTreeIndex],
tree[rightTreeIndex]);
        }


        @Override
        public String toString(){
            StringBuilder res = new StringBuilder();
            res.append('[');
            for(int i = 0 ; i < tree.length ; i ++){
                if(tree[i] != null)
                    res.append(tree[i]);
                else
                    res.append("null");


                if(i != tree.length - 1)
                    res.append(", ");
            }
            res.append(']');
            return res.toString();
        }
    }


    private SegmentTree<Integer> segTree;


    public NumArrayComplete(int[] nums) {


        if(nums.length != 0){
            Integer[] data = new Integer[nums.length];
            for(int i = 0 ; i < nums.length ; i ++)
                data[i] = nums[i];
            segTree = new SegmentTree<>(data, (a, b) -> a + b);
```

```
        }
    }


    public void update(int i, int val) {
        if(segTree == null)
            throw new IllegalArgumentException("Error");
        segTree.set(i, val);
    }


    public int sumRange(int i, int j) {
        if(segTree == null)
            throw new IllegalArgumentException("Error");
        return segTree.query(i, j);
    }
}
```