【第三十三周】从前缀和到树状数组（Binary-Indexed-Tree）

1、1109. 航班预订统计

1. 我们可以遍历给定的预定记录数组，每次 O(1) 地完成对差分数组的修改即可。当我们完成了差分数组的修改，只需要最后求出差分数组的前缀和即可得到目标数组。
2. 本题中日期从 1 开始，因此我们需要相应的调整数组下标对应关系，对于预定记录 booking=[l,r,inc]，我们需要让 d[l−1] 增加 inc，d[r] 减少 inc。特别地，当 r 为 n 时，我们无需修改 d[r]，因为这个位置溢出了下标范围。如果求前缀和时考虑该位置，那么该位置对应的前缀和值必定为 0。

```
var corpFlightBookings = function(bookings, n) {
    const nums = new Array(n).fill(0);
    for (const booking of bookings) {
        nums[booking[0] - 1] += booking[2];
        if (booking[1] < n) {
            nums[booking[1]] -= booking[2];
        }
    }
    for (let i = 1; i < n; i++) {
        nums[i] += nums[i - 1];
    }
    return nums;
};
```

2、307. 区域和检索 - 数组可修改

1. 树状数组裸题，需要根据输入数组进行初始化，这里通过计算前缀和初始化
2. 在查找数组中某个范围内元素的总和时，分为三种情况：

   情况一：若mid刚好在sLeft的左侧

   情况二：若mid刚好在sLeft的右侧

   情况三：若mid刚好在 sLeft和sRight中间

   然后分别对于三种情况进行递归查找

```
/**
 * @param {number[]} nums
```

```
 */
var NumArray = function(nums) {
    this.arrTree = Array(2*nums.length);
    this.numsArr = nums;
    this.setTree(0,0,this.numsArr.length-1);
    // console.log(this.arrTree)
};
NumArray.prototype.setTree =  function (i,left, right){//构建线段树
        if(left>right) return 0;
        if(left==right){
            this.arrTree[i] = this.numsArr[left];
            return this.arrTree[i];
        }
        let mid = ((left+right)-(left+right)%2)/2;
        // 构建i节点的左子线段树
        let leftVal = this.setTree(2*i+1,left,mid);
        //构建i节点的右子线段树
        let rightVal = this.setTree(2*i+2,mid+1,right);
        this.arrTree[i] = leftVal + rightVal;
        return this.arrTree[i];
    }

/**
 * @param {number} index
 * @param {number} val
 * @return {void}
 */
NumArray.prototype.update = function(index, val) {// 更新线段树中的值
    this.diff = val - this.numsArr[index];
    this.numsArr[index] = val;
    this.index = index;
    this.updateArrTree(0,0,this.numsArr.length-1);
    // console.log(this.arrTree)
};

NumArray.prototype.updateArrTree = function(i,left,right){
        if(left>right) return;
        if(left==right && left===this.index){
            this.arrTree[i] += this.diff;
            return;
        }


        // this.arrTree[i] += this.diff;
        this.arrTree[i] += this.diff;

        let mid = ((left+right)-(left+right)%2)/2;
        // 如果index小于等于mid顺着左子树查找,
        if(this.index<=mid){
```

```
            this.updateArrTree(2*i+1,left,mid);

        }else{// 否则顺着右子树查找
            this.updateArrTree(2*i+2,mid+1,right);
        }
    }

/**
 * @param {number} left
 * @param {number} right
 * @return {number}
 */
NumArray.prototype.sumRange = function(left, right) {

    return this.sumArrTree(0,0,this.numsArr.length-1,left,right);

};

NumArray.prototype.sumArrTree = function(i,left,right,sLeft,sRight){
        if(left>sRight || right<sLeft){
            return 0;
        }
        if(sLeft<=left && sRight>=right ){
            // console.log('a,',this.arrTree[i])
            return this.arrTree[i];
        }

        let mid = ((left+right)-(left+right)%2)/2;

        if(sRight<=mid){// 若mid刚好在sLeft的左侧
            return this.sumArrTree(2*i+1,left,mid,sLeft,sRight);
        }else if(sLeft>mid){// 若mid刚好在sLeft的右侧
            return  this.sumArrTree(2*i+2,mid+1,right,sLeft,sRight)
        }else{// 若mid刚好在 sLeft和sRight中间
            return this.sumArrTree(2*i+1,left,mid,sLeft,mid) +
this.sumArrTree(2*i+2,mid+1,right,mid+1,sRight);
        }
    }

/**
 * Your NumArray object will be instantiated and called as such:
 * var obj = new NumArray(nums)
 * obj.update(index,val)
 * var param_2 = obj.sumRange(left,right)
 */
```

3、面试题 10.10. 数字流的秩

1. 每次输入一个新x值，假设有arr[i]<x，对于统计小于等于x的数量时，都需要将arr[i]的值加一

```
var StreamRank = function() {
  this.arr = new Array(50001).fill(0);
};

/**
 * @param {number} x
 * @return {void}
 */
StreamRank.prototype.track = function(x) {
  this.arr[x] += 1;
};

/**
 * @param {number} x
 * @return {number}
 */
StreamRank.prototype.getRankOfNumber = function(x) {
  let total = 0;
  for (let i = 0; i <= x; i++) {
    total += this.arr[i];
  }
  return total;
};

/**
 * Your StreamRank object will be instantiated and called as such:
 * var obj = new StreamRank()
 * obj.track(x)
 * var param_2 = obj.getRankOfNumber(x)
 */
```

4、1310. 子数组异或查询

1. 题目提供了一个 queries 数组，其中每一个 query 其实就是在给定的 arr 数组中划定一个范围，然后我们需要做的计算就是把这个范围内的所有数字进行异或（xor）运算，最终得到这个 query 的结果。
2. 如果数组 arr 的长度为 n，数组 queries 的长度为 m（即有 m 个查询），则最坏情况下每个查询都需要 O(n)的时间计算结果，总时间复杂度是 O(n*m)，会超出时间限制，因此必须优化。
3. 查询[left,right](left≤right)，用 Q(left,right) 表示该查询的结果
4. 考虑到异或运算的性质 $x \oplus x=0$。

5. 当 left=0 时， xors[left]=0，因此 Q(left,right)=xors[left]⊕xors[right+1] 也成立。
6. 因此对任意 0≤left≤right<n，都有Q(left,right), Q(left,right)=xors[left]⊕xors[right+1]，即可在 O(1) 的时间内完成一个查询的计算。
7. 所以我们整体的操作流程是
8. 计算前缀异或数组 xors；
9. 计算每个查询的结果，第 i 个查询的结果为 xors[queries[i][0]] ⊕ xors[queries[i][1]+1]。

```
  var xorQueries = function(arr, queries) {
    const n = arr.length;
    const xors = new Array(n + 1).fill(0);
    for (let i = 0; i < n; i++) {
        xors[i + 1] = xors[i] ^ arr[i];
    }
    const m = queries.length;
    const ans = new Array(m).fill(0);
    for (let i = 0; i < m; i++) {
        ans[i] = xors[queries[i][0]] ^ xors[queries[i][1] + 1];
    }
    return ans;
};
```

5、1409. 查询带键的排列

1. 对于数组 queries 中的每一个询问项 query，我们在排列 P 中找到 query 所在的位置，并把它加入答案。随后，我们需要将 query 移动到排列 P 的首部。具体地，我们首先将 query 从排列 P 中移除，再添加到排列 P 的首部即可。

```
/**
 * @param {number[]} queries
 * @param {number} m
 * @return {number[]}
 */
var processQueries = function(queries, m) {
  let ans = [], n = queries.length;

  // 构建 p 排列的数组
  let arrp = new Array(m).fill(0).map((v, k) => k + 1);

  for (let i = 0; i < n; i++) {
    let curr = queries[i];
    let index = arrp.indexOf( curr );
    ans.push( index );
    arrp.splice(index, 1);
    arrp.unshift( curr );
  }
```

```
  return ans;
};
```