初识线段树

引入

线段树是算法竞赛中常用的用来维护**区间信息**的数据结构。

线段树可以在 $O(logN)$ 的时间复杂度内实现单点修改、区间修改、区间查询（区间求和，求区间最大值，求区间最小值）等操作。

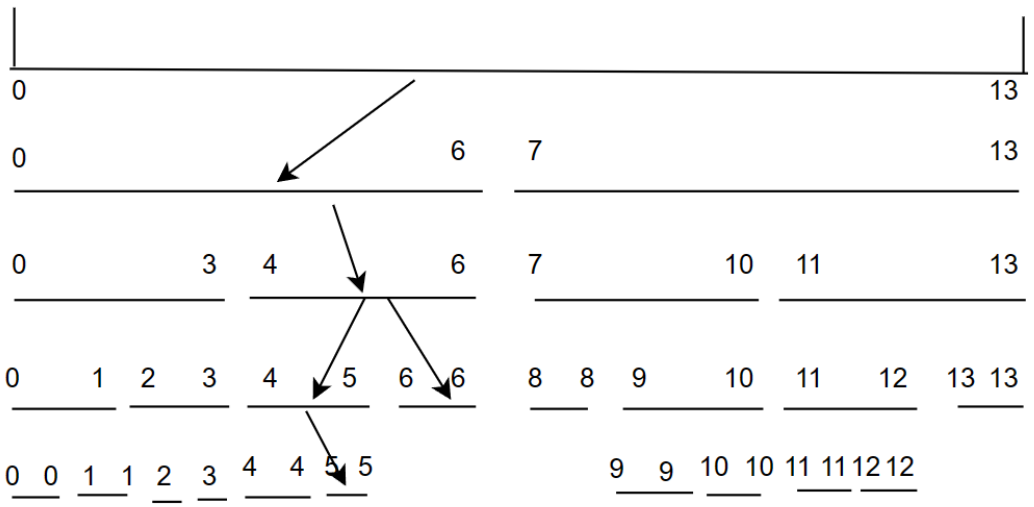
过程

更正： 图片画得有点问题，下标应该从 1 开始到 n 结束，不过意思能懂就行了

例如我们要求某一区间的最大值，首先每一个线段树节点包含的信息有：区间左值 l、区间右值 r、区间最大值 max。

```
class Node{
    int l, r, max;
    public Node(){};
    public Node(int l, int r) {
        this.l = l;
        this.r = r;
    }
    public Node(int l, int r, int max) {
        this.l = l;
        this.r = r;
        this.max = max;
    }
}
```

现在我们查询区间5-6的最大值



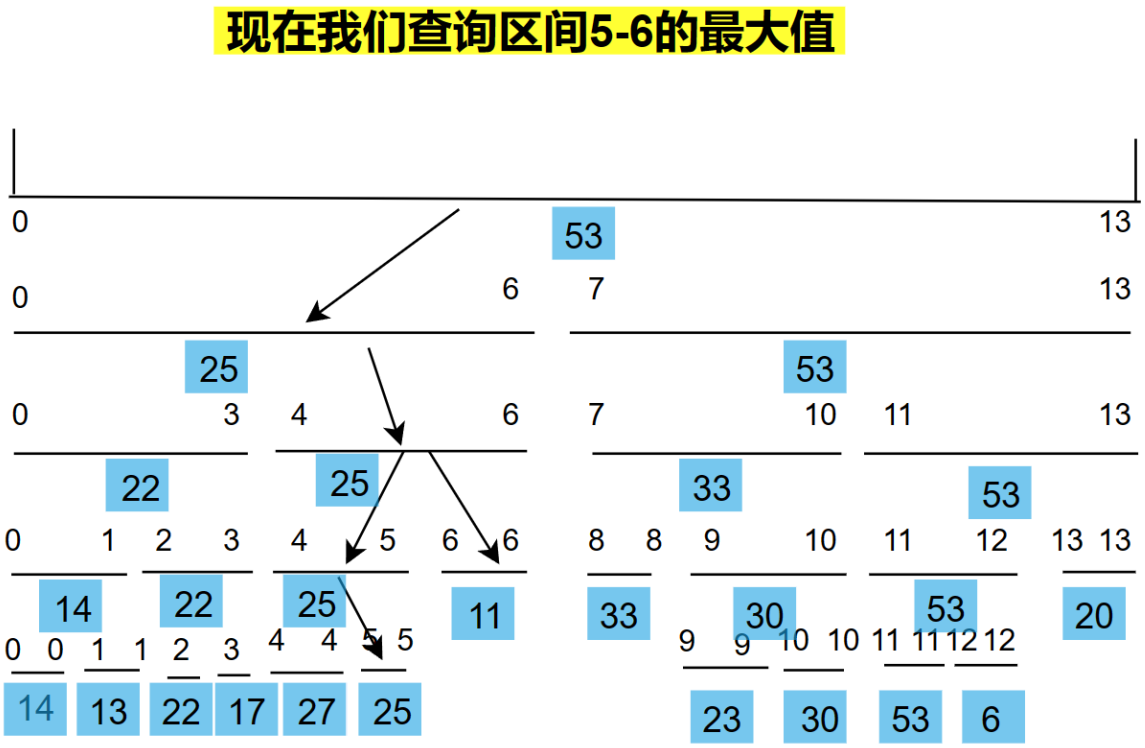
首先我们要做的是建立这个线段树，即 $build$ 操作。

```
static void build(int u, int l, int r) {
    tr[u] = new Node();
    tr[u].l = l, tr[u].r = r;
    if (l == r){
        tr[u].max = arr[l];//走到了叶子节点
        return;
    }
    int mid = (l + r) >> 1;
    build(u<<1, l, mid);
    build(u<<1 | 1, mid+1, r);
    pushup(u);//回溯算每个区间的最大值
}
```

$pushup$ 其实就是回溯的时候由子节点更新父节点的属性值

```
static void pushup(int u) {
    tr[u].max = Math.max(tr[u<<1].max, tr[u<<1|1].max);
}
```

现在我们的线段树长这样了



然后我们进行 $query$ 操作

```
static void query(int u, int l, int r) {
    if (l <= tr[u].l && tr[u].r <= r) return tr[u].max;//如果这个区间被完全包含了,则直接返回其属性（最大值）
    int mid = (tr[u].l + tr[u].r) >> 1;
    int res = 0;
    if (l <= mid) res = query(u<<1, l, r);
    if (r > mid) res = Math.max(res, query(u<<1|1, l, r));
    return res;
}
```

那么一个最基本的线段树就已经成形了，接下来证明一下节点数需要开到 $4 * N$ 个节点数。

假设一共有 N 个叶子节点，那么我们可以算一下倒数第二层的节点数：$2^{(log_2^N+1)}$，那么最坏的情况下一共会有 $2^0 + 2^1 + 2^2 + ... + 2^{log_2^n+1} + N$，即 $2N - 1$ 个节点，为了防止数据越界，我们可以在往下走一层，则一共开 $4N$ 个节点。

最后完整代码如下：

```
import java.util.*;
import java.io.*;

public class Main{
    static class Node {
        int l, r, max;
    }

    static int n;
    static final int N = 100010;
    static int[] arr = new int[N];
    static Node[] tr = new Node[N*4];

    public static void main(String[] args) throws Exception {
        BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
        PrintWriter pw = new PrintWriter(System.out);
        String[] s = bf.readLine().split(" ");
        n = Integer.parseInt(s[0]);
        s = bf.readLine().split(" ");
        for (int i = 1; i <= n; i ++ ) {
            arr[i] = Integer.parseInt(s[i-1]);
        }
        build(1, 1, n);
        s = bf.readLine().split(" ");
        int l = Integer.parseInt(s[0]), r = Integer.parseInt(s[1]);
        pw.println(query(1, l, r));
        pw.close();
    }

    static int query(int u, int l, int r) {
        if (l <= tr[u].l && tr[u].r <= r) return tr[u].max;
        int res = 0;
        int mid = (tr[u].l + tr[u].r) >> 1;
```

```
        if (l <= mid) res = query(u<<1, l, r);
        if (r > mid) res = Math.max(res, query(u<<1|1, l, r));
        return res;
    }

    static void build(int u, int l, int r) {
        tr[u] = new Node();
        tr[u].l = l;
        tr[u].r = r;
        if (l == r) {
            tr[u].max = arr[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(u<<1, l, mid);
        build(u<<1|1, mid+1, r);
        pushup(u);
    }

    static void pushup(int u) {
        tr[u].max = Math.max(tr[u<<1].max, tr[u<<1|1].max);
    }
}
```

来看一道单点修改的题

最大数

就是比上面讲的多了一个修改操作 $update$

```
static void update(int u, int x, int max) {//x表示待修改的节点
        if (tr[u].l == x && tr[u].r == x) {
            tr[u].max = max;
            return;
        } else {
            int mid = (tr[u].l + tr[u].r) >> 1;
            if (x <= mid) update(u<<1, x ,max);
            else update(u<<1|1, x, max);
            pushup(u);
        }
    }
```

完整代码如下：

```
import java.util.*;
import java.io.*;

public class Main{
    static class Node {
        int l, r, max;
```

```
    }

    static int m, p;
    static final int N = 200010;
    static Node[] tr = new Node[N*4];

    public static void main(String[] args) throws Exception {
        BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
        PrintWriter pw = new PrintWriter(System.out);
        String[] s = bf.readLine().split(" ");
        m = Integer.parseInt(s[0]);
        p = Integer.parseInt(s[1]);
        int res = 0;
        int cnt = 0;
        build(1, 1, m);
        while (m -- > 0) {
            s = bf.readLine().split(" ");
            String ch = s[0];
            int num = Integer.parseInt(s[1]);
            if (ch.equals("Q")) {
                res = query(1, cnt-num+1, cnt);
                pw.println(res);
            }
            else {
                update(1, ++cnt, (int)(((long)num+res)%p));
            }
        }
        pw.close();
    }

    static int query(int u, int l, int r) {
        if (l <= tr[u].l && tr[u].r <= r) return tr[u].max;
        int res = 0;
        int mid = (tr[u].l + tr[u].r) >> 1;
        if (l <= mid) res = query(u<<1, l, r);
        if (r > mid) res = Math.max(res, query(u<<1|1, l, r));
        return res;
    }

    static void build(int u, int l, int r) {
        tr[u] = new Node();
        tr[u].l = l;
        tr[u].r = r;
        if (l == r) {
            return;
        }
        int mid = (l + r) >> 1;
        build(u<<1, l, mid);
        build(u<<1|1, mid+1, r);
    }

    static void update(int u, int x, int max) {
```

```
        if (tr[u].l == x && tr[u].r == x) {
            tr[u].max = max;
            return;
        } else {
            int mid = (tr[u].l + tr[u].r) >> 1;
            if (x <= mid) update(u<<1, x ,max);
            else update(u<<1|1, x, max);
            pushup(u);
        }
    }

    static void pushup(int u) {
        tr[u].max = Math.max(tr[u<<1].max, tr[u<<1|1].max);
    }
}
```