线段树上二分

主讲人：邓哲也

POJ 2828 Buy Tickets

有 N 个人来排队，第 i 个人来的时候会排在第 p[i](0 <= p[i] < i) 个人的后面，它会被分配一个数字 v[i]。

现在告诉你 n 对 (p[i], v[i])，请你按照队伍顺序输出每个人的数字。

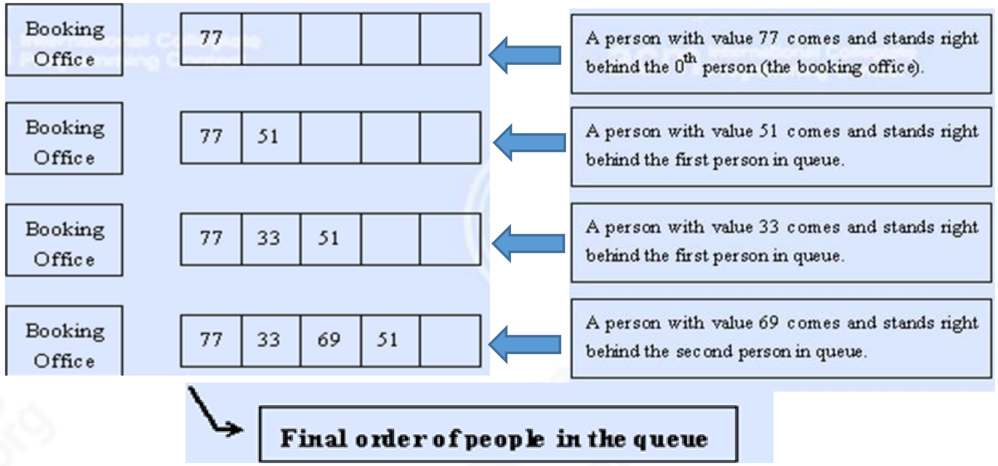
n <= 200000

Sample Input

```
4
0 77
1 51
1 33
2 69
```

Sample Output

```
77 33 69 51
```



POJ 2828 Buy Tickets

考虑模拟做法，维护一个数组 Q 表示队列

第 i 个人进来时，需要把 Q[p[i] + 1 .. i] 的人都往后移动一位。

然后把 Q[p[i] + 1] 赋值为 v[i]

这个向后移动一位操作非常困难。

用数组和指针维护都是 $O(n^2)$ 的。

POJ 2828 Buy Tickets

因为第 i 个人进入队伍之后，它的位置还会发生改变，相对位置也会发生改变。

正难则反。

倒着考虑每个人，就可以确定下每个人的位置了。

比如第 n 个人，一定就在第 p[n] + 1 的位置上。

思考如何判断第 n-1 个人的位置。

POJ 2828 Buy Tickets

如果 p[n-1]+1 < p[n]+1，那么也就是说，第 n-1 个人的位置没有因为第 n 个人的进入而发生改变。

此时要第 n-1 个人的位置在 p[n-1]+1 处。

如果 p[n-1]+1 >= p[n] + 1，那么也就是说，第 n-1 个人的位置因为第 n 个人的进入而发生改变，且向后移了一位。

此时要第 n-1 个人的位置在 p[n-1]+2 处。

POJ 2828 Buy Tickets

总结成一句话，就是第 n-1 个人的位置，是除了第 n 个人所在的位置以外的 n-1 个位置中的第 p[n-1]+1 个。

继续考虑第 n-2 个人的位置。

对 p[n-2]+1 和前两个人的位置进行讨论。

得出第 n-2 个人的位置，就是除了第 n 和第 n-1 个人所在的位置以外的 n-2 个位置中的第 p[n-2]+1 个。

POJ 2828 Buy Tickets

当问题规模为 n 时，确定 n 所在位置，删除这个位置后，问题就可以递归成规模为 n-1 的问题，因为这 n-1 个人的相对位置关系是与 n 无关的。

只是要在考虑绝对位置的时候，留出 n 所在的空位。

那么问题就变成了，一开始有 n 个位置，每次我们查询从左往右数第 k 个空位置，然后放置一个人（即把它变为非空）。

POJ 2828 Buy Tickets

一个简单的想法就是用线段树维护一个全是 1 的数组 a。

查找第 k 个空位置，就是找一个下标 i 使得 a[1 ..i] 的前缀和等于 k。

由于 a 是单调递增的，所以我们可以二分下标 i。

用线段树维护区间和，每次二分的时候查询即可。

这样的时间复杂度是 $O(n \log^2 n)$

POJ 2828 Buy Tickets

注意线段树本身就有分治的性质。

在使用分治套分治的时候，就要思考，能不能减少冗余的计算，尽量用一层分治解决问题。

我们考虑直接在线段树上进行二分。

POJ 2828 Buy Tickets

注意线段树本身就有分治的性质。

在使用分治套分治的时候，就要思考，能不能减少冗余的计算，尽量用一层分治解决问题。

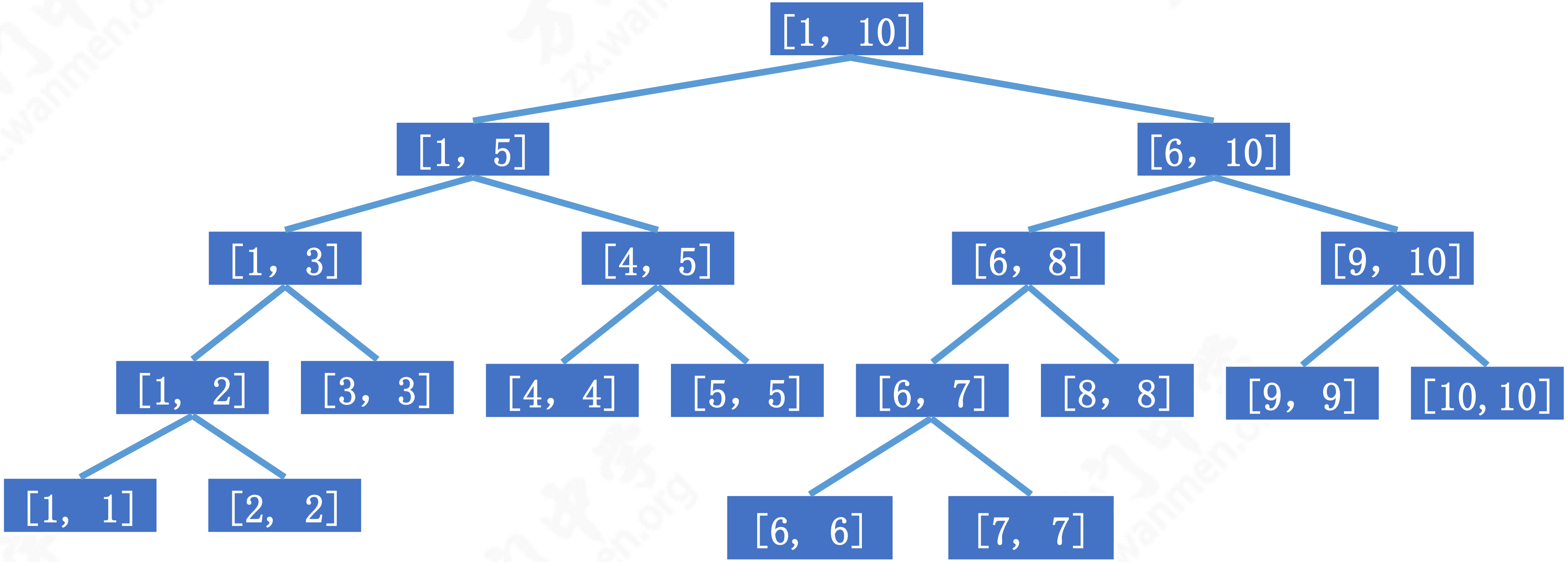
我们考虑直接在线段树上进行二分。

假设每个节点记录的是区间和 sum.

POJ 2828 Buy Tickets

如果 sum[ls] >= k ，进入左子树；

否则 k -= sum[ls]，进入右子树。



POJ 2828 Buy Tickets

因此只要在树上从根走到叶节点，就能确定第 k 个位置所在的叶子。

把叶子的值修改为 0，再回溯上来。

POJ 2828 Buy Tickets

```
void change(int k, int l, int r, int x){
      if (l == r){
            sum[x] = 0;
            return;
      }
      int mid = (l + r) >> 1;
      if (sum[ls] >= k) change(k, l, mid, ls);
      else change(k - sum[ls], mid + 1, r, rs);
      update(x);
}
```

POJ 2828 Buy Tickets

```
void update(int x){
        sum[x] = sum[ls] + sum[rs];
}
void build(int l, int r, int x){
        if (l == r){
                sum[x] = 1;
                return 0;
        }
        int mid = (l + r) >> 1;
        build(l, mid, ls);
        build(mid + 1, r, rs);
        update(x);
}
```

POJ 2828 Buy Tickets

因此只要在树上从根走到叶节点，就能确定第 k 个位置所在的叶子。

把叶子的值修改为 0，再回溯上来。

每次的时间复杂度为 O(log n)

总的时间复杂度就是 O(n log n).

代码实现

```
#define N 200010
int sum[N << 2], n, ans[N], pos[N], val[N];
void upd(int x){
    sum[x] = sum[ls] + sum[rs];
}
int find(int k, int l, int r, int x){
    if (l == r){
        sum[x] = 0;
        return l;
    }
    int mid = (l + r) >> 1, ret = 0;
    if (k <= sum[ls]) ret = find(k, l, mid, ls);
    else ret = find(k - sum[ls], mid + 1, r, rs);
    upd(x);
    return ret;
}
```

代码实现

```
void build(int l, int r, int x){
    if (l == r){
        sum[x] = 1;
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, ls);
    build(mid + 1, r, rs);
    upd(x);
}
```

代码实现

```
int main() {
    while(scanf("%d", &n) != EOF){
        for (int i = 1;i <= n;i ++) scanf("%d%d", &pos[i], &val[i]);
        build(1, n, 1);
        for (int i = n;i >= 1;i --){
            int k = find(pos[i] + 1, 1, n, 1);
            ans[k] = val[i];
        }
        for(int i = 1;i <= n;i ++) printf("%d%c", ans[i], (i < n) ? ' ' : '\n');
    }
    return 0;
}
```

下节课再见