线段树的构建和查询

主讲人：邓哲也

线段树的链表存储

```
struct node{
    int left, right, value;
    node *lchild, *rchild;
};
```

left 和 right 用来表示该结点代表的区间。

value 维护这个区间的信息，比如区间和等等。

每个节点同时维护两个孩子的指针。

坏处是指针要动态申请，由于寻址不连续，所以效率不高。

线段树的数组存储

回忆堆的储存方式。

根存在 1号位置。

对于存储在 i 号位置的节点，它的左孩子存在 2i 号位置，

右孩子 2i+1 号位置。

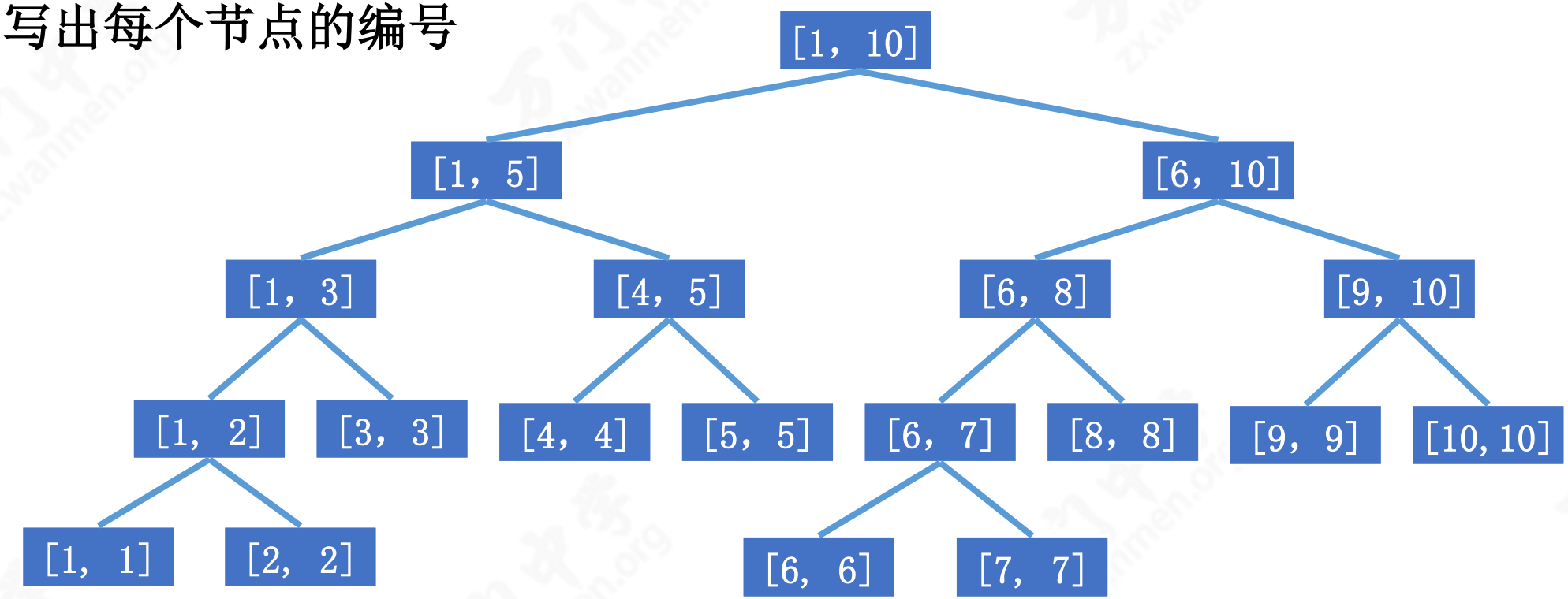
同时我们也不需要记录每个位置对应的区间，只要在递归的找

这个点的时候边找边修改即可。

线段树的结构

大小为 10 的线段树

写出每个节点的编号



建立线段树

```
void build(int l, int r, int x) {
        if (l == r) {                          //叶子结点
                sum[x] = a[l];
                return;
        }
        int mid = (l + r) >> 1;
        build(l, mid, x * 2);
        build(mid + 1, r, x * 2 + 1);
        update(x);                             //更新信息
}
```

建立线段树

线段树的建立过程是自顶向下的。

根据定义如果当前节点代表的区间[l，r]没有满足 l = r，那它一定有两个孩子，递归构建左右两部分。

如果l = r，说明到达了叶节点，可以停止向下构建，同时记录叶节点的信息。

之后每一步，构建完左右孩子的节点后，要把它们俩的信息合并到一起，保存在当前节点。

线段树的数组存储

数组存储也有一点不好，因为线段树并不是真正的完全二叉树。

最后一层可能很空。且空节点的数量可以达到 2n 个。

因此维护长度为 n 的序列，用数组存线段树的话，最好要开到 4*n 的长度，才能保证数组不越界。

线段树的查询

在构建完线段树之后，现在对于节点 [l, r]，它就记录了区间[l, r]的区间和sum。

现在以查询区间 [l, r] 的子段和为例，来看看如何实现区间划分。

线段树的查询

查询的过程也是自顶向下的。

假设询问区间是 [A, B]，现在所在的节点表示的区间为[l, r]

- 如果 A <= l <= r <= B，那么直接返回当前节点的 sum，
  不用递归下去。
- 否则，我们需要判断要不要递归到两个子节点上询问。

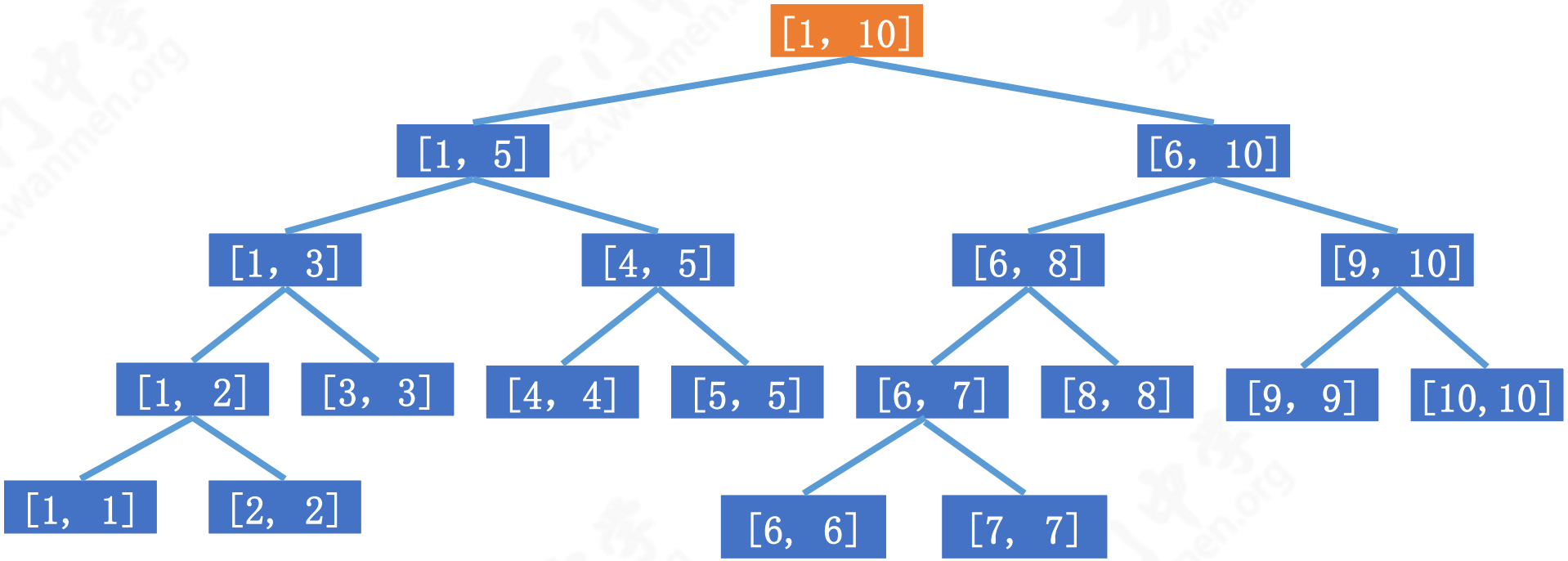
线段树的查询

假设询问区间是 [A, B]，现在所在的节点表示的区间为[l, r]

- 计算 mid = (l + r) / 2，左子节点的区间为 [l, mid]，右子节点的区间为 [mid+1, r].
- 如果 A <= mid，即询问区间与左子节点有重合，需要递归到左子节点。
- 如果 B >= mid + 1，即询问区间与右子节点有重合，需要递归到右子节点。
- 递归完之后，需要把两个孩子询问的结果加起来作为返回值。

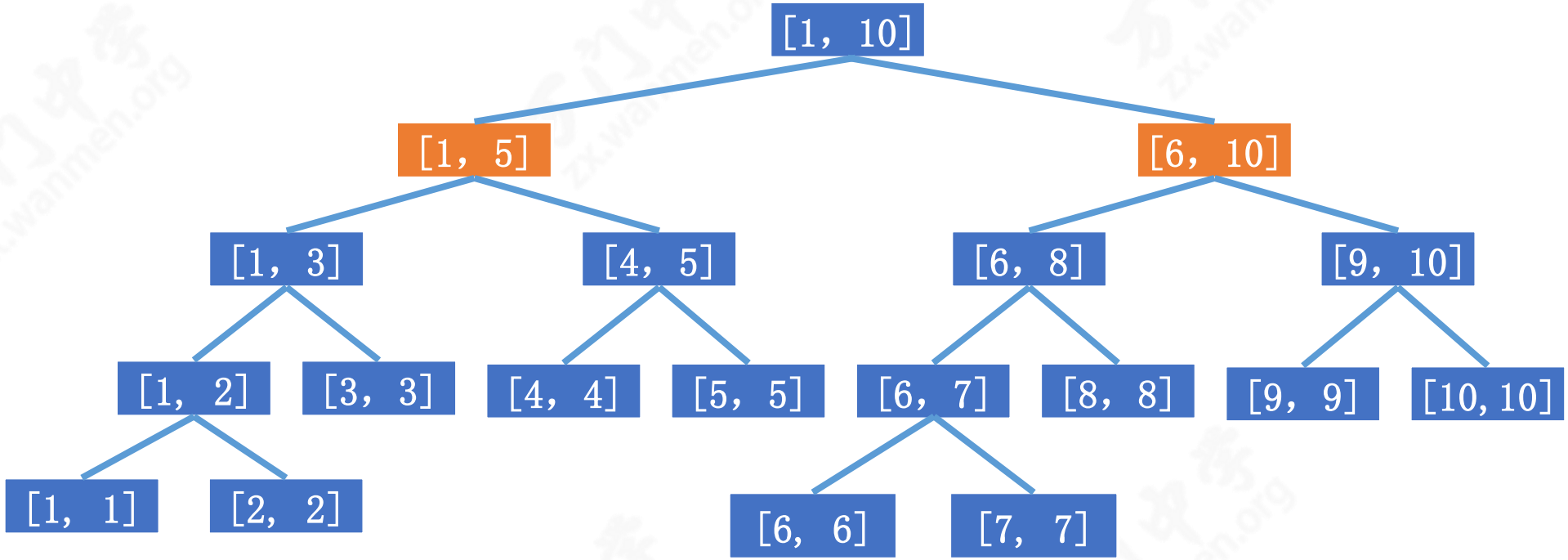
线段树上的查询

例如查询[2，9]的区间和。



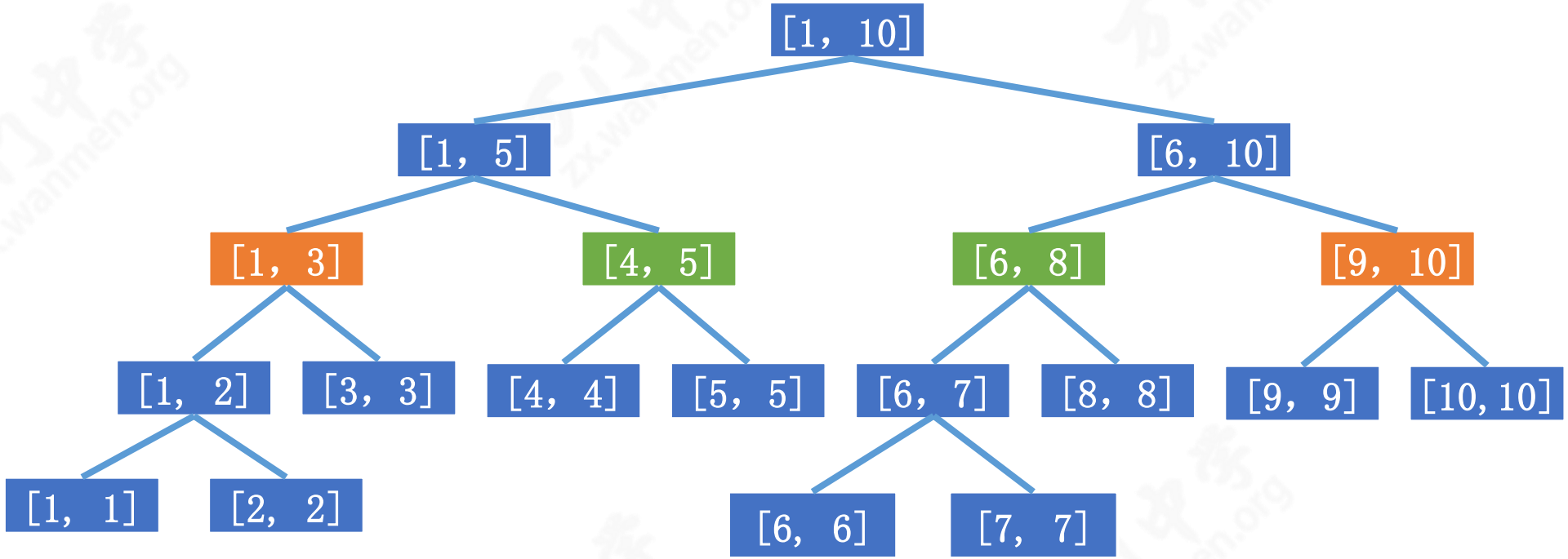
线段树上的查询

例如查询[2，9]的区间和。



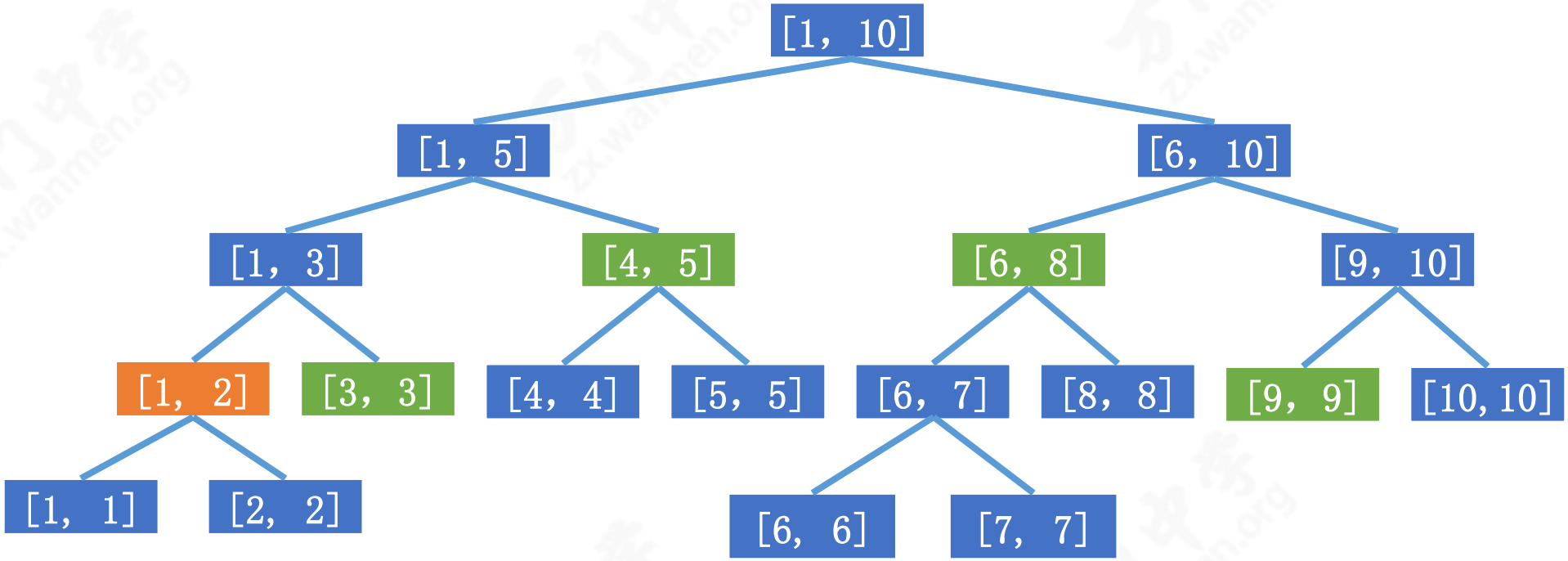
线段树上的查询

例如查询[2，9]的区间和。



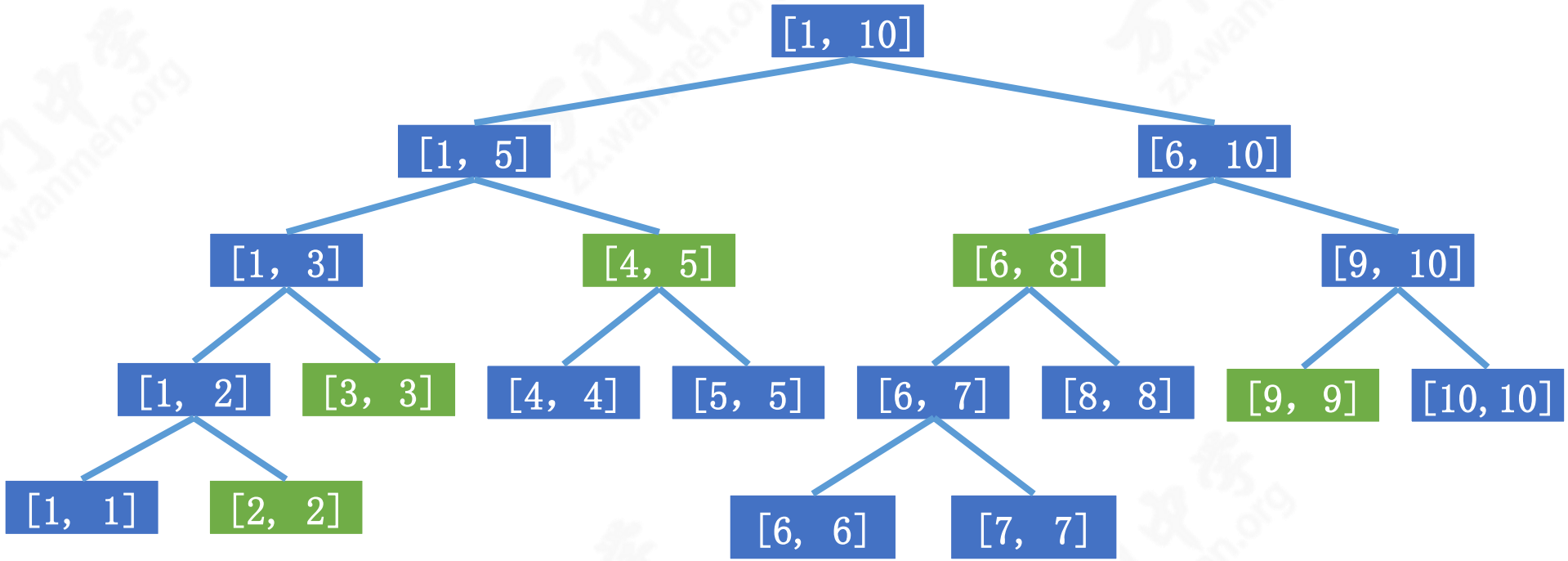
线段树上的查询

例如查询[2，9]的区间和。



线段树上的查询

例如查询[2，9]的区间和。



线段树的查询

```
int query(int A, int B, int l, int r, int x){
        if (A <= l && r <= B)
                return sum[x];
        int mid = (l + r) >> 1, ans = 0;
        if (A <= mid)
                ans += query(A, B, l, mid, x * 2);
        if (mid < B)
                ans += query(A, B, mid + 1, r, x * 2 + 1);
        return ans;
}
```

下节课再见