线段树的区间修改（一）

主讲人：邓哲也

万门中学 zx.wanmen.org

线段树的区间修改

问题：有一个长度为 n 的序列，a[1]，a[2]，…，a[n]。

现在执行 m 次操作，每次可以执行以下两种操作之一：

1. 将下标在区间［l，r］的数都修改为 v（v>0）。
2. 询问一个下标区间［l，r］中所有数的和。、

线段树的区间修改

如果把区间修改拆成 r - l + 1 个单点修改，甚至不如模拟。

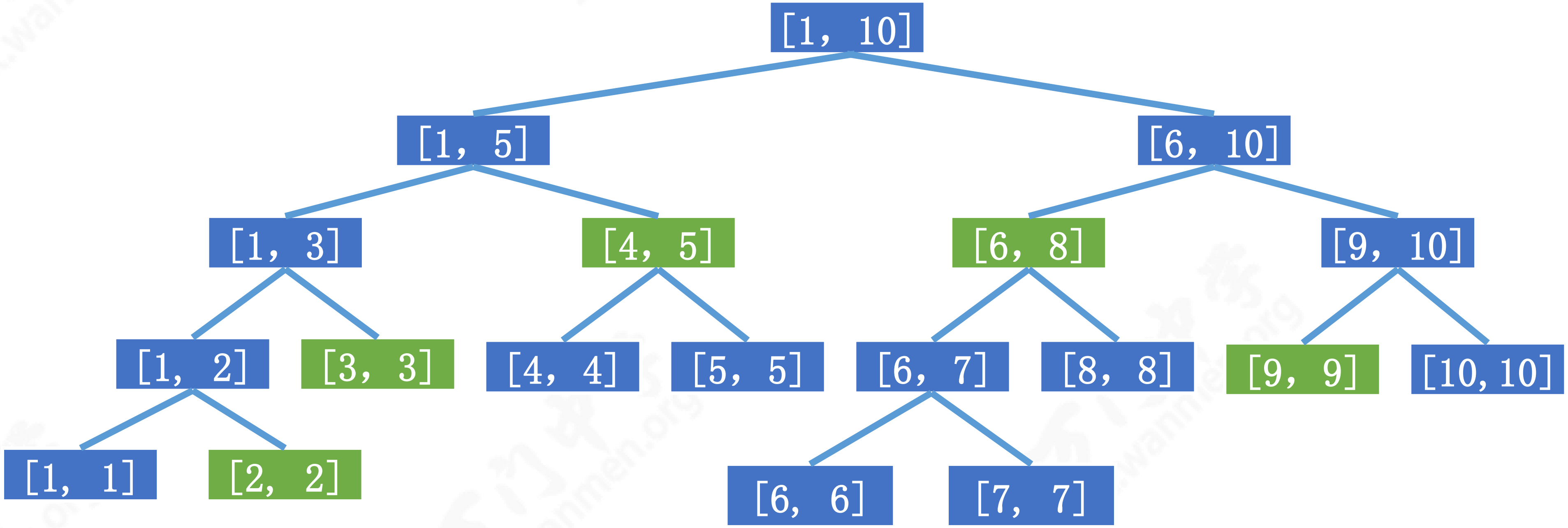
我们希望区间修改和区间查询一样，先把区间分成线段树上的

若干个区间。然后分别修改这几个区间。

延迟修改技术

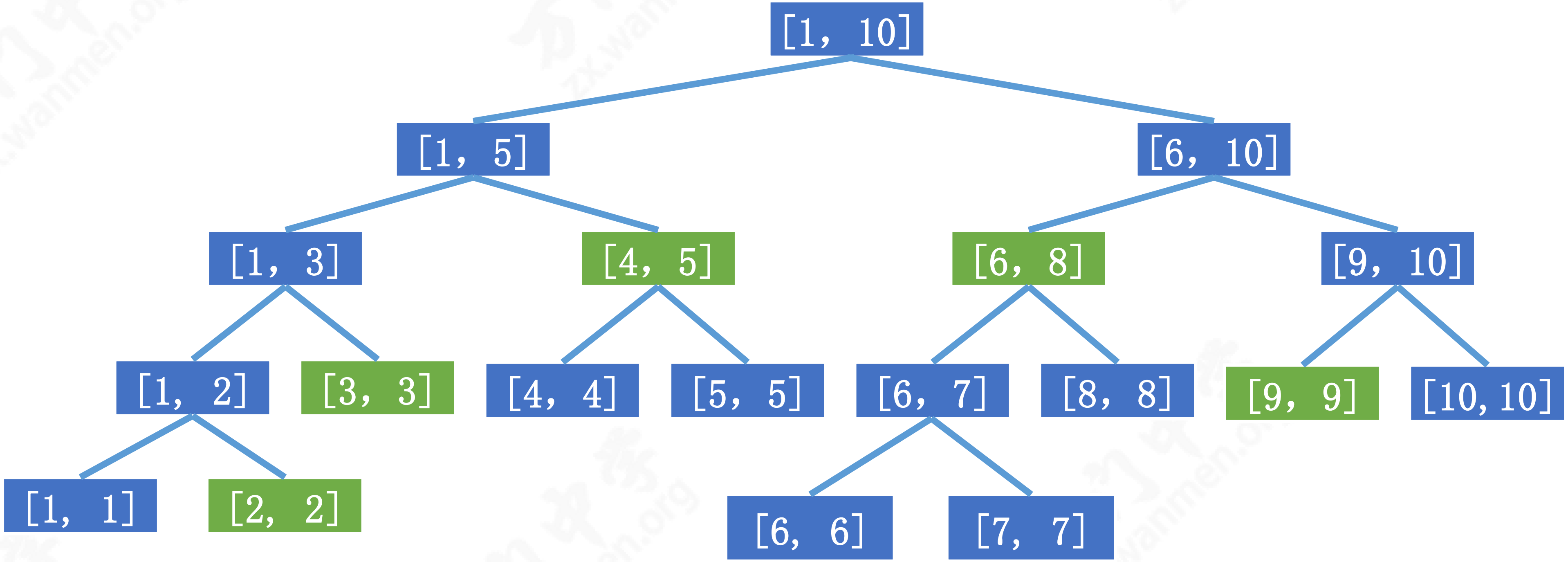
把［2，9］里的数都改成 v。

把绿色的节点打上一个标记 tag，表示这个子树里的值都是 v，对应的 sum 要改成 v * (r - l + 1)



延迟修改技术

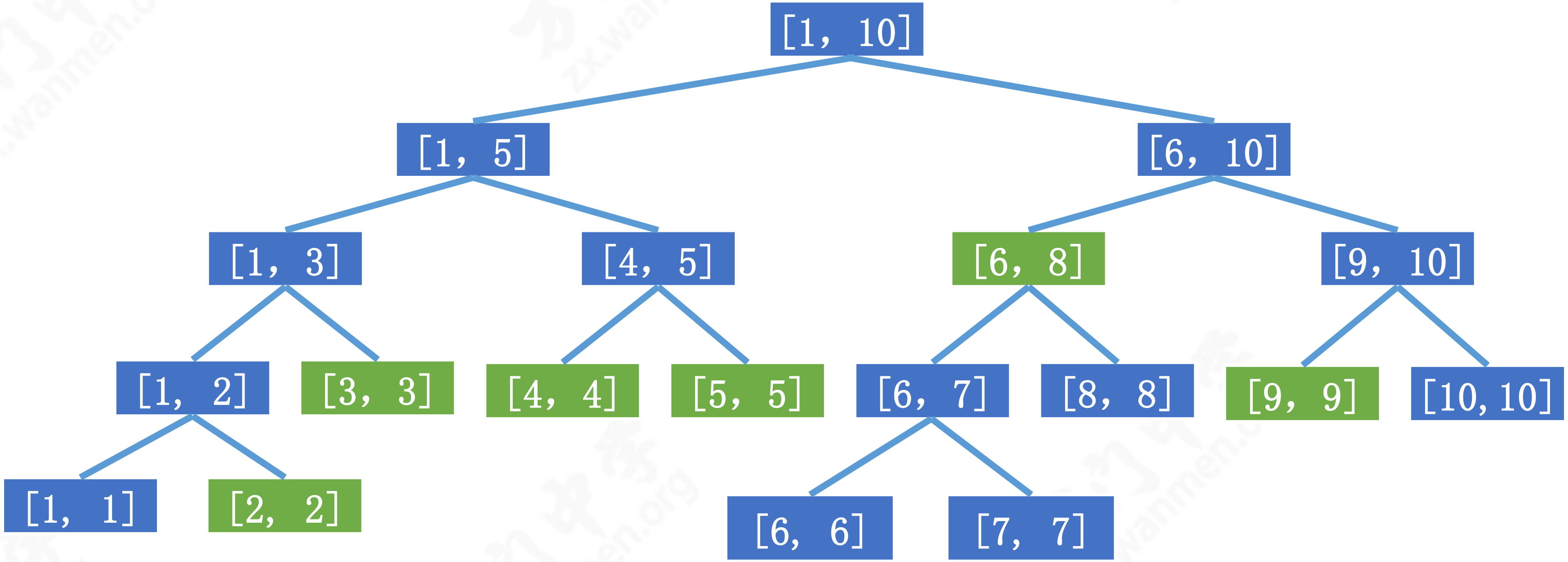
只要查询不到 [4, 4], [5, 5], [6, 7], [8, 8]，我们就不用递归下去修改它们的 tag 和 sum。



延迟修改技术

但是，如果此时查询［5，5］，查询的路上会碰到［4，5］。

需要把［4，5］的 tag 清空，传给［4，4］和［5，5］。



延迟修改技术

此时查询［5，5］，查询的路上会碰到［4，5］。

发现［4，5］的 tag 是 v，说明这颗子树里的值发生了改变，

但是子树里的值还没有进行更改。

为了保证接下来的查询正确，我们需要把这个标记下传。

比如［l，r］的 tag 非负，我们就需要把标记传给［l，mid］

和［mid+1，r］，以保证递归下去的查询正确。

延迟修改技术

对于[l, mid]：tag[x * 2] = tag[x]，sum[x * 2] = (mid - l + 1) * tag[x]；

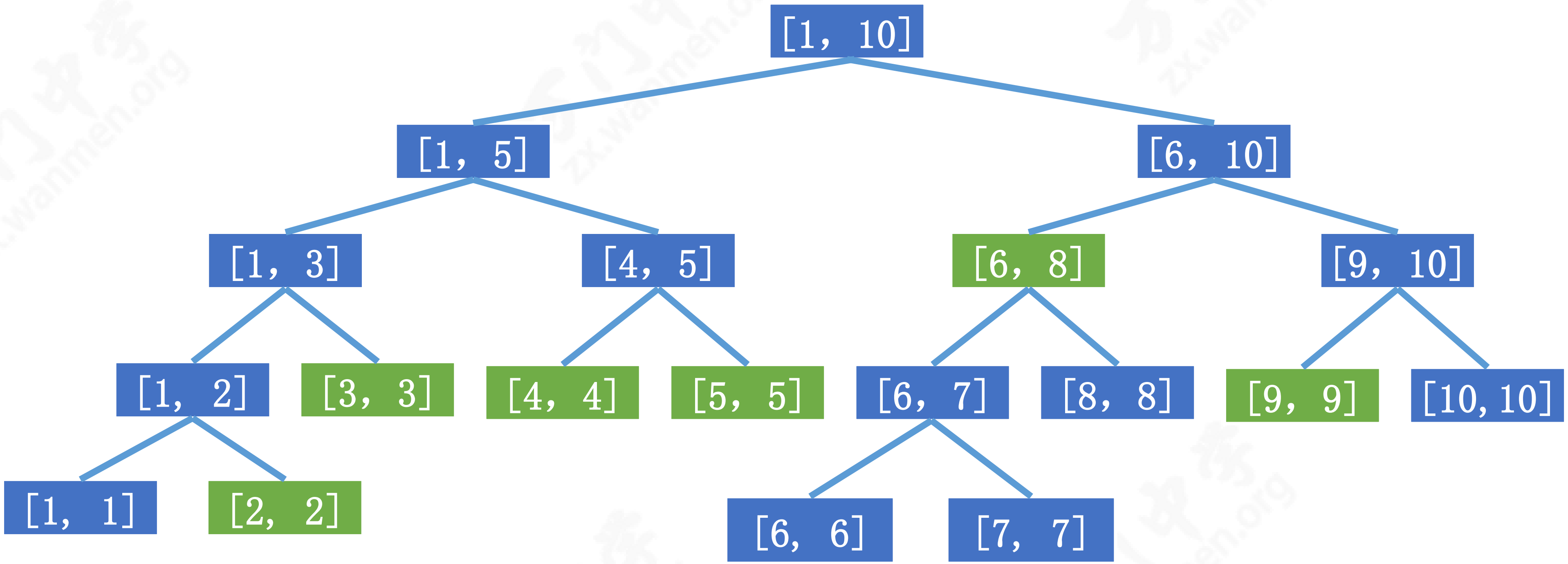
对于[mid+1, r]：tag[x * 2 + 1] = tag[x]，sum[x * 2 + 1] = (r - mid) * tag[x]

然后把 tag[x] 置为 0，表示这个点上已经没有待下传的标记了。

对于查询和修改操作，都需要检查当前节点是否需要下传标记。

延迟修改技术

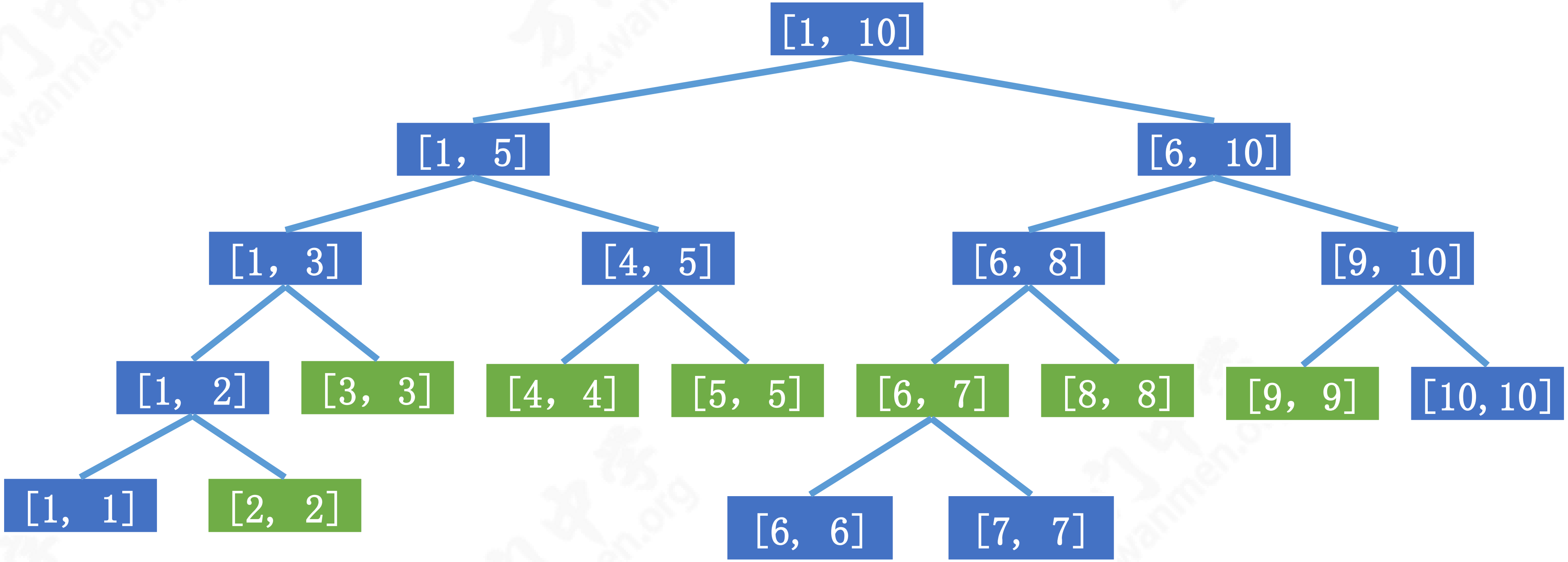
思考如果此时查询［8，8］，标记会如何下传？



延迟修改技术

思考如果此时查询［8，8］，标记会如何下传？

［6，8］下传到子节点，［6，7］不用再往下传了。



延迟修改技术代码实现

为了方便，我们定义 ls, rs 分别为左右子树的编号：

```
#define ls (x << 1)
#define rs (x << 1 | 1)
```

更新函数：

```
void update(int x){
	sum[x] = sum[ls] + sum[rs];
}
```

延迟修改技术代码实现

标记下传函数：

```
void down(int l, int r, int x){
      int mid = (l + r) >> 1;
      if (tag[x] > 0){
            tag[ls] = tag[rs] = tag[x];
            sum[ls] = (mid - l + 1) * tag[x];
            sum[rs] = (r - mid) * tag[x];
            tag[x] = 0;
      }
}
```

延迟修改技术代码实现

修改[A, B]区间改为 v：

```
void change(int A, int B, int v, int l, int r, int x) {
	if (A <= l && r <= B) {
		tag[x] = v;
		sum[x] = v * (r - l + 1);
		return;
	}
	down(l, r, x);			// 在继续修改之前，先检查是否要下传标记
	int mid = (l + r) >> 1;
	if (A <= mid) change(A, B, v, l, mid, ls);
	if (mid < B)		change(A, B, v, mid + 1, r, rs);
	update(x);		// 回溯的时候要更新每个节点的sum，因为子节点的值
改变了
}
```

延迟修改技术代码实现

查询[A, B]的区间和

```
int query(int A, int B, int l, int r, int x) {
      if (A <= l && r <= B)
            return sum[x];
      down(l, r, x);              // 在继续查询之前，先检查是否要下传标记
      int mid = (l + r) >> 1, ret = 0;
      if (A <= mid) ret += query(A, B, l, mid, ls);
      if (mid < B)  ret += query(A, B, mid + 1, r, rs);
      //update(x);          // 这里是不用update(x)的，思考为什么？
      return ret;
}
```

延迟修改技术时间复杂度

这种到需要的时候才进行标记下传的方法，使我们整体处理的时间复杂度仍然维持在 $O(\log_2 n)$

下节课再见

万门中学 zx.wanmen.org