信息学

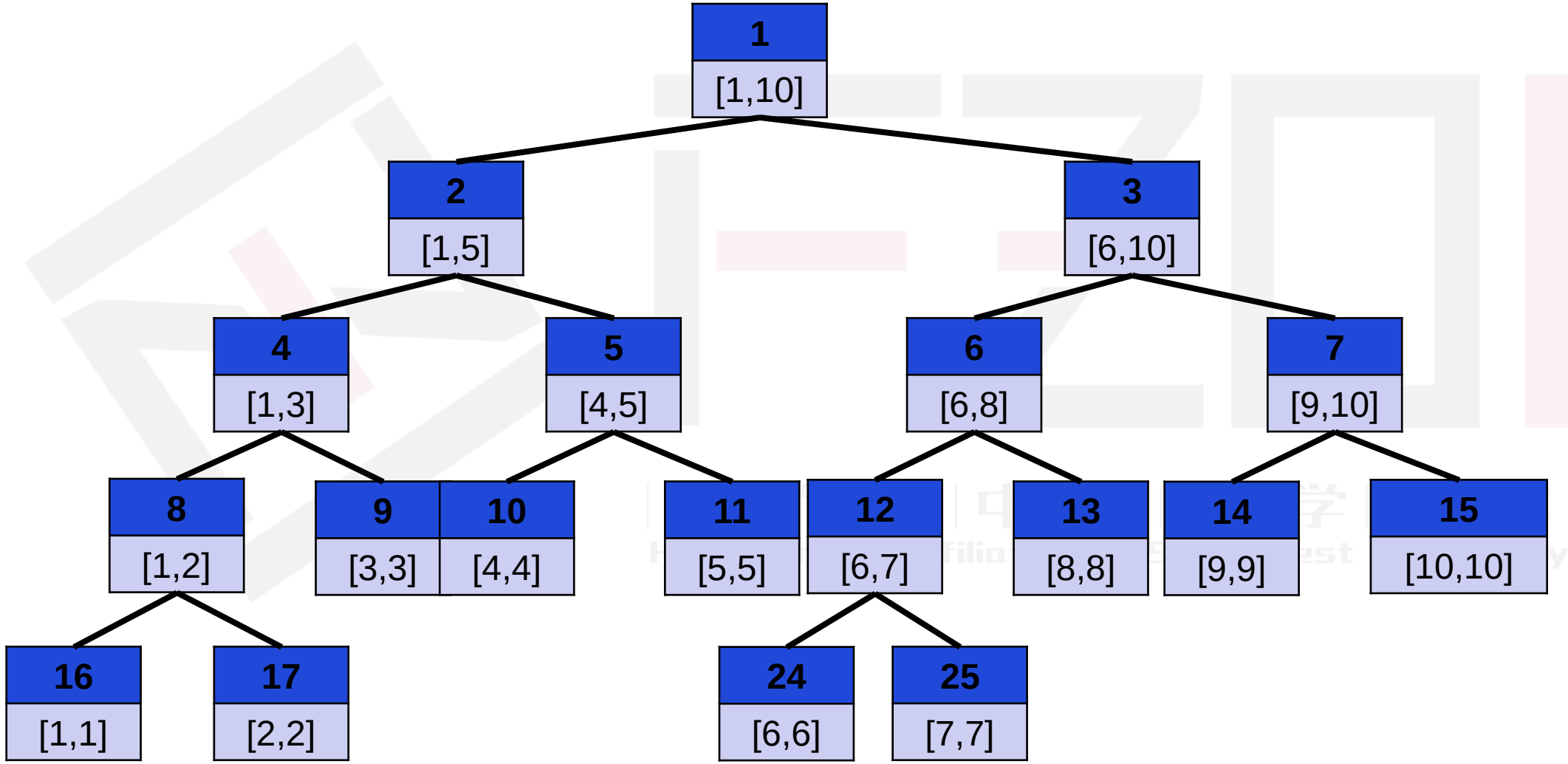
线段树

西南大学附属中学校

信息奥赛教练组

线段树

线段树是基于**分治**思想的**二叉树结构**，用于进行**区间上的信息统计**。



线段树

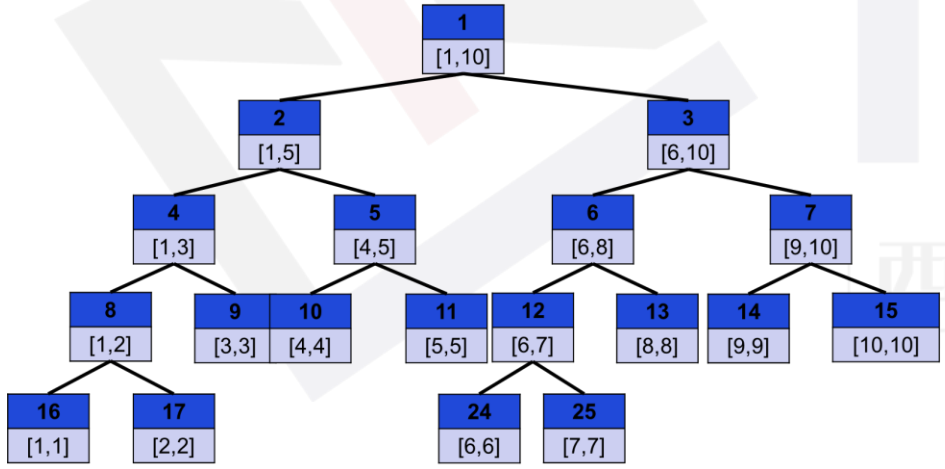
① 线段树的每个节点都代表一个区间。

② 线段树具有唯一的根节点，代表整个区间的统计范围：[1,N]。

③ 线段树的每个叶结点都代表一个长度为1的区间[x,x]。

④ 根节点编号为1，其他编号节点为x的左子节点为x*2，右子节点为x*2+1。

⑤ 对于每个内部结点[l,r]，它的左子节点是[l,mid]，右子节点是[mid+1,r]，其中mid=(l+r)/2。



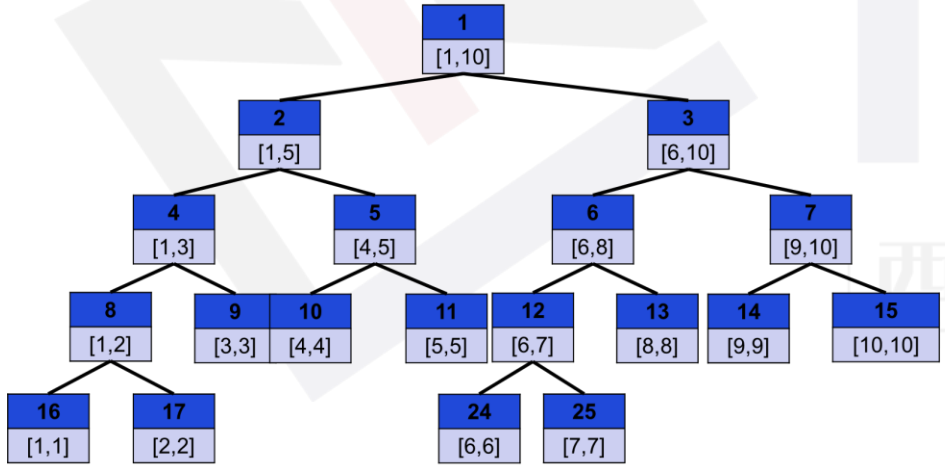
线段树

N个叶结点需要多大空间存储？

N个叶结点的满二叉树共有结点：

$$N+N/2+N/4+\cdots+1=2N-1$$

有时候最后一层产生了空余，所有**保存线段树的数组长度要不小于4N，才能保证不会越界**。

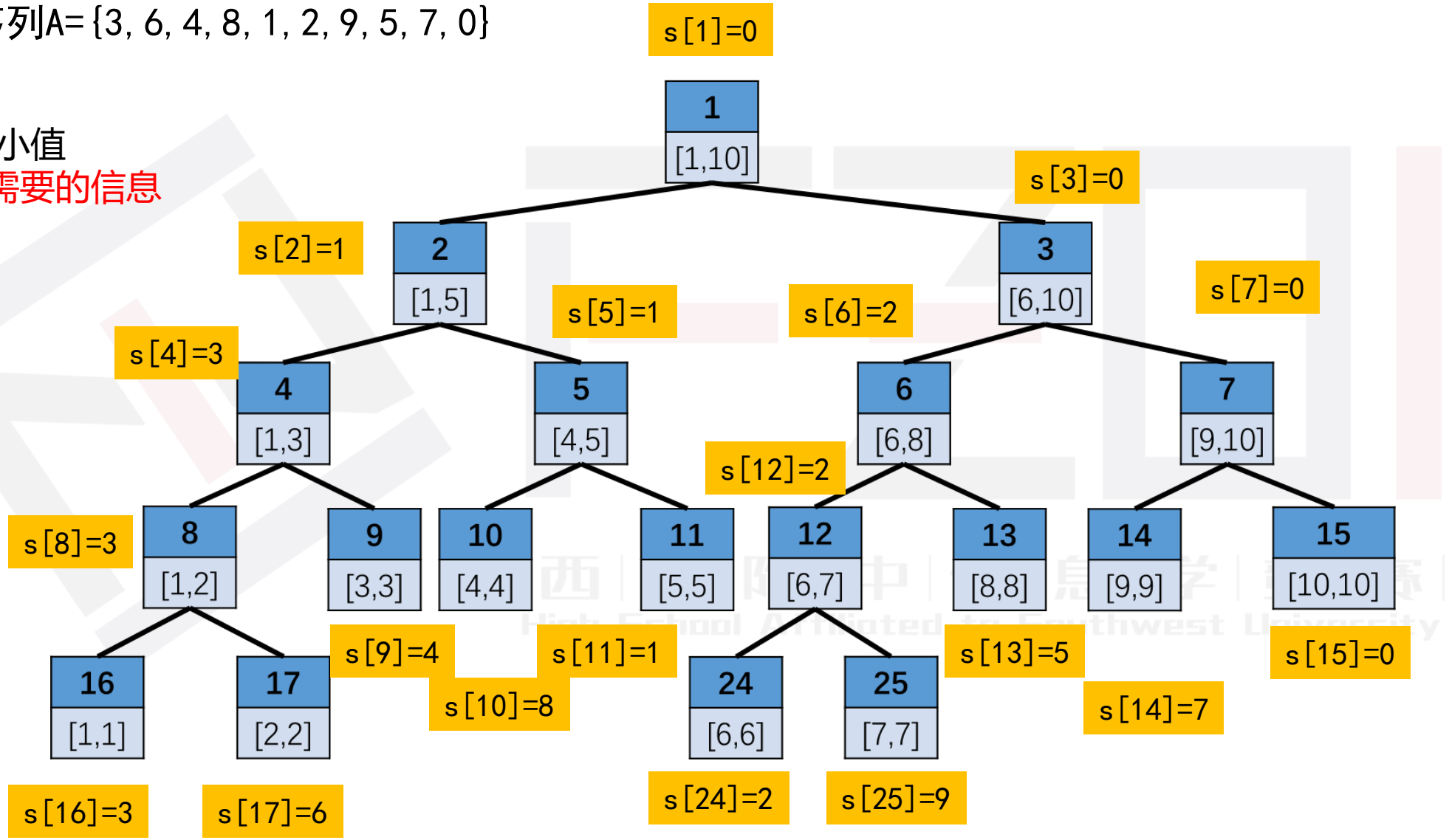


线段树—建树

序列A={3, 6, 4, 8, 1, 2, 9, 5, 7, 0}

s[i]存储最小值

结点存储需要的信息



线段树—建树


```
void built(int k,int l,int r)//第k个节点代表的区间为[l,r]
{
	if(l==r)
	{
		s[k]=a[l];
		return;
	}
	int mid=(l+r)/2;
	built(k*2,l,mid);
	built(k*2+1,mid+1,r);
	s[k]=min(s[k*2],s[k*2+1]);//维护最小值
}
//主函数main()里
for(int i=1;i<=n;i++)
	cin>>a[i];
built(1,1,n);//调用
```

```
s[k]=max(s[k*2],s[k*2+1]); //维护最大值
s[k]=s[k*2]+s[k*2+1]; //维护区间和
```

时间复杂度:O(N)

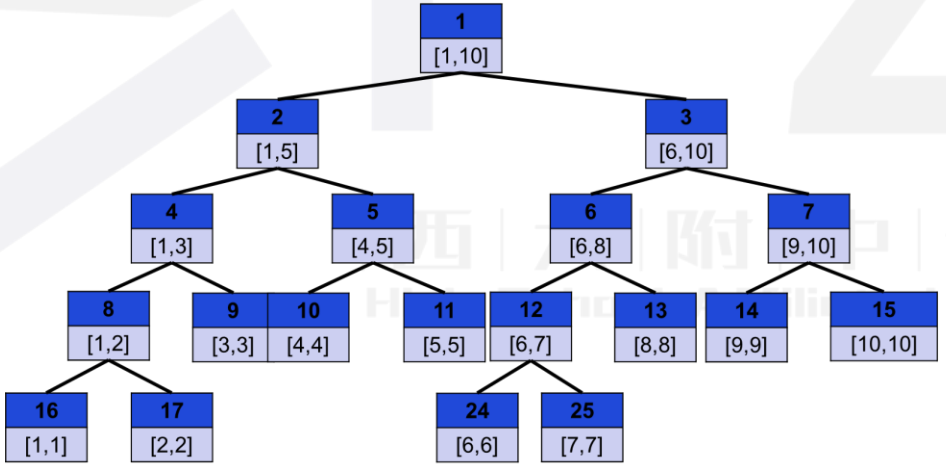
线段树—单点修改

将A[x]的值修改为v。

在线段树中，根节点(编号1的节点)是执行各种命令的入口。

我们从根节点出发，递归找到区间[x,x]的叶节点进行修改，同时从下往上更新节点维护的最小值(或最大值，区间和等)。

时间复杂度O(logN)



线段树—单点修改


```
void update(int k,int l,int r,int x,int v)   //将a[x]修改为v
{
      if(x<l||x>r) return;          //a[x]不在区间内
      if(l==r&&x==l)                //到达对应的叶结点
      {
            s[k]=v;
            return;
      }
      int mid=(l+r)/2;
      update(k*2,l,mid,x,v);              //修改左区间
      update(k*2+1,mid+1,r,x,v);//修改右区间
      s[k]=min(s[k*2],s[k*2+1]);//维护最小值
}
//主函数main()里
update(1,1,n,x,v);  //调用
```

线段树—区间查询

查询区间[x,y]的最小值(或最大值，和等)。

从根节点出发，查询区间[x,y]和节点表示区间[l,r]存在三种关系：

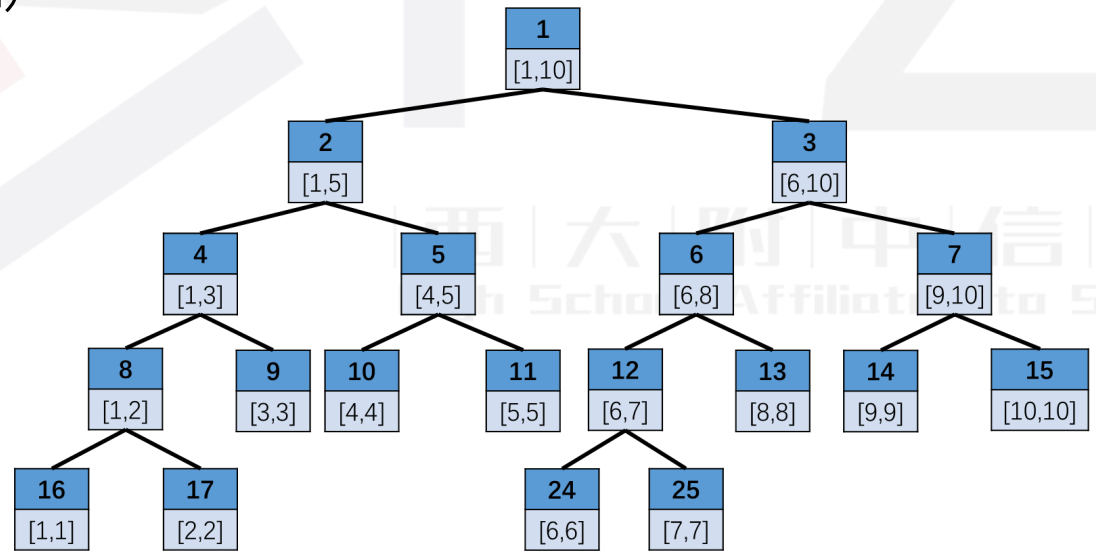
① 查询区间[x,y]与表示区间[l,r]无交集，返回一个不影响结果的值。

② 查询区间[x,y]包含表示区间[l,r]，直接返回当前节点维护的最小值，作为候选答案。

③ 除了上面两种情况，继续对两个子节点进行递归，返回结果中的较小值。

时间复杂度:O(logN)

求最小值，就返回一个极大值，求最大值，就返回极小值，求和就返回0



线段树—区间查询

询问区间最小值

```
int query(int k,int l,int r,int x,int y)      //询问区间[x,y]
{
      if(r<x||l>y) return 2100000000;  //查询区间与当前区间无交集
      if(x<=l&&r<=y)      return s[k]; //查询区间包含当前区间，返回维护好的
最小值
      int mid=(l+r)/2;
      return min(query(k*2,l,mid,x,y), query(k*2+1,mid+1,r,x,y));
}
//主函数main()里
query(1,1,n,x,y);   //调用
```

一个问题

Q：如果需要对某个区间进行修改怎么办？

显然如果每个都去修改，时间复杂度为O(nlogn)

树状数组借助差分数组实现O(1)修改

线段树借助lazy tag实现，其实就是偷懒标记

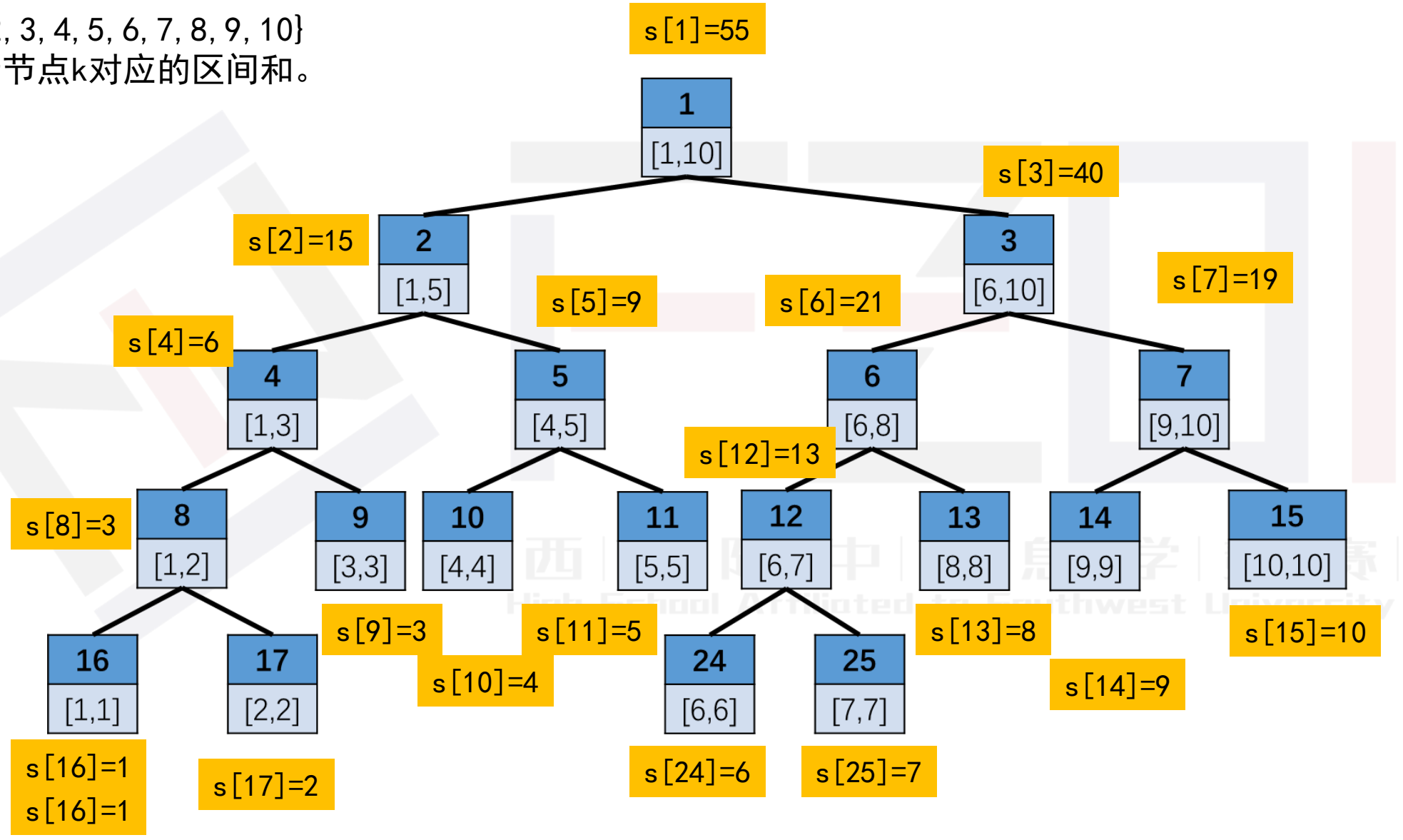
线段树—区间修改

lazy标记下传：

从某个节点递归下去时，将当前节点的lazy下传，更新两个子节点的s值和lazy值，并将当前节点的lazy值清零。

线段树—区间修改

a[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
s[k]表示节点k对应的区间和。



线段树—区间修改

将区间[4,8]每个元素增加5

s[5]=s[5]+(5-4+1)*5=19

lazy[5]=lazy[5]+5=5

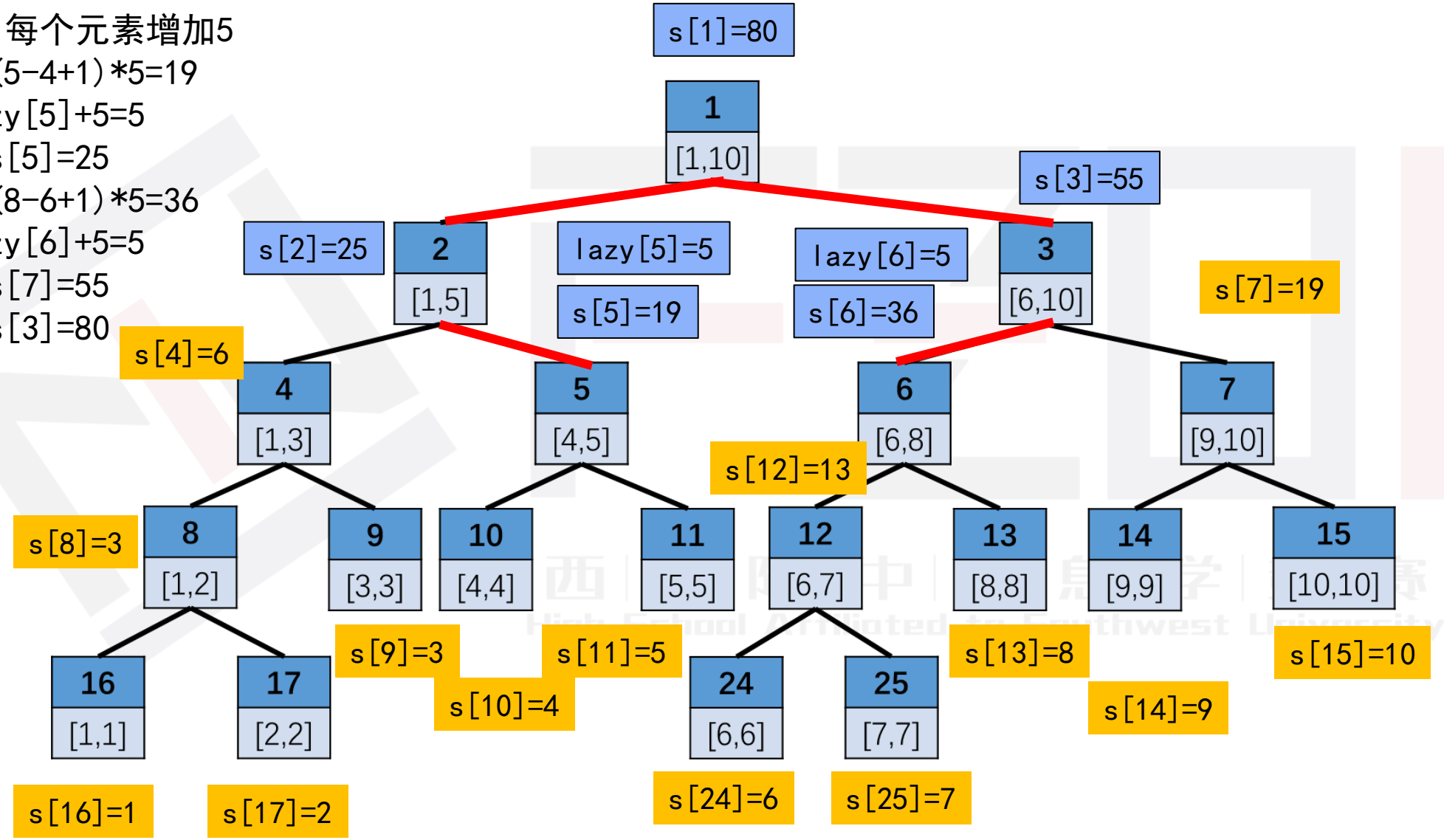
s[2]=s[4]+s[5]=25

s[6]=s[6]+(8-6+1)*5=36

lazy[6]=lazy[6]+5=5

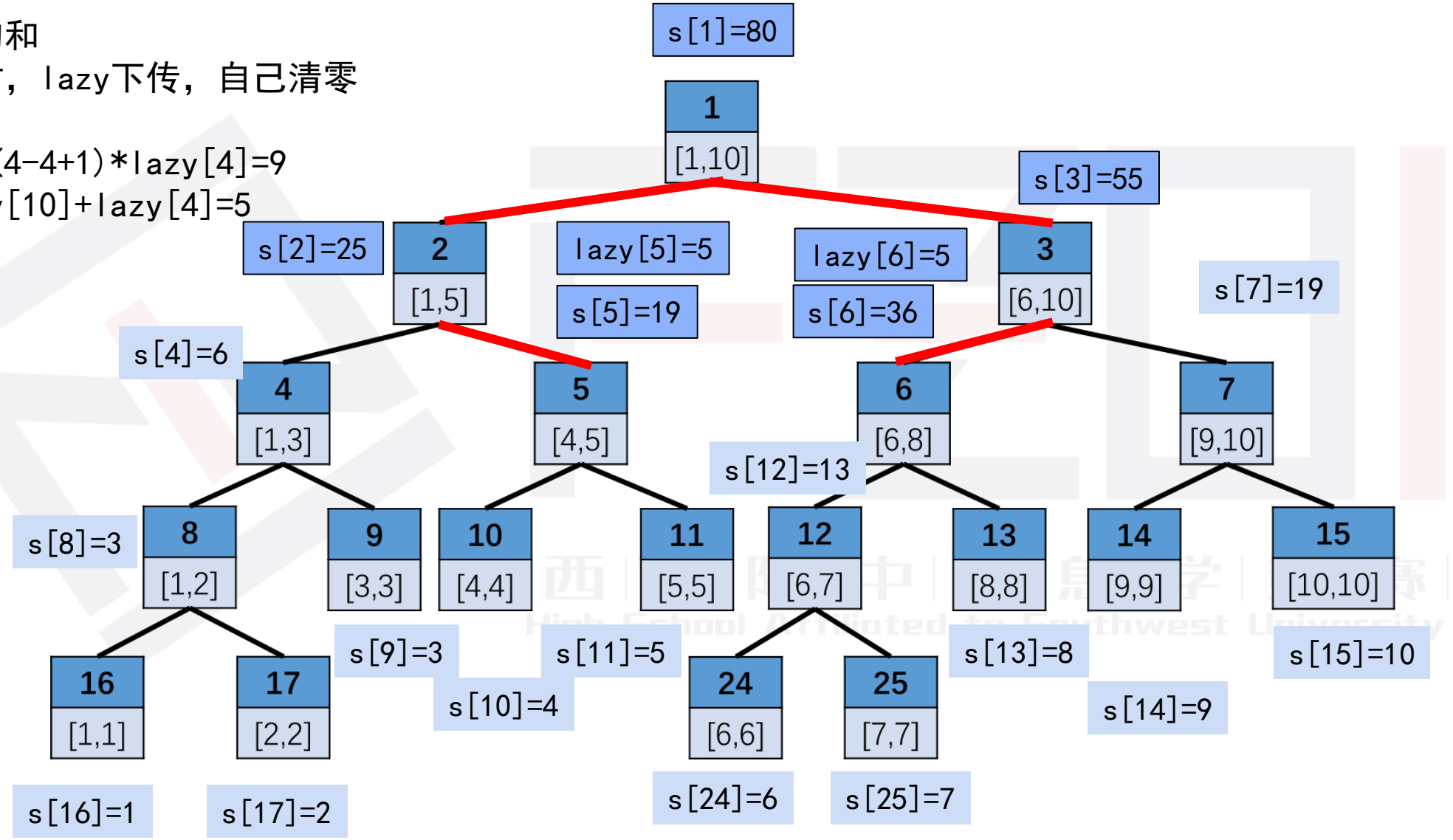
s[3]=s[6]+s[7]=55

s[1]=s[2]+s[3]=80



线段树—区间修改

求区间[2,4]的和
访问到节点5时，lazy下传，自己清零
lazy[5]=0
s[10]=s[10]+(4-4+1)*lazy[4]=9
lazy[10]=lazy[10]+lazy[4]=5



线段树—区间修改


```
void Add(int k,int l,int r,int v) //给定区间[l,r]所有数加上v
{
        lazy[k]+=v;                    //打标记
        s[k]=(r-l+1)*v;     //维护区间和
        return;
}
void pushdown(int k,int l,int r)  //将节点k的标记下传给子节点
{
        if(lazy[k]==0) return;
        int mid=(l+r)/2;
        Add(k*2,l,mid,lazy[k]);              //下传左子树
        Add(k*2+1,mid+1,r,lazy[k]);          //下传右子树
        lazy[k]=0;                           //下传自身清零
 }
```

线段树—区间修改


```
void modify(int k,int l,int r,int x,int y,int v) //给区间[x,y]所有数加上v
{
        if(r<x||l>y) return;                    //不包含不处理
        if(x<=l&&r<=y) {Add(k,l,r,v);return;}   //完全包含直接处理
        int mid=(l+r)/2;
        pushdown(k,l,r);            //不存在完全包含就将k节点标记下传，继续找
        modify(k,l,mid,x,y,v);      //找左子树
        modify(k,mid+1,r,x,y,v);    //找右子树
        s[k]=s[k*2]+s[k*2+1];               //下传后更新区间和s
}
```

线段树—区间查询

询问区间和(带lazy下传)

```
int query(int k,int l,int r,int x,int y)  //询问区间[x,y]的和
{
        if(r<x||l>y) return 0;                    //不包含
        if(x<=l&&r<=y) return s[k];                    //包含直接返回
        int mid=(l+r)/2;
        pushdown(k,l,r);            //下传标记，将未更新的节点用标记更新
        return query(2*k,l,mid,x,y)+ query(2*k+1,mid+1,r,x,y);
}//这里不需要更新区间和，因为子节点未改变
```

总结

线段树和树状数组的基本功能都是在某一满足结合律的操作(比如加法，乘法，最大值，最小值)下，0(logn)的时间复杂度内修改单个元素并且维护区间信息。

不同的是，树状数组只能维护前缀“操作和”(前缀和，前缀积，前缀最大最小)，而线段树可以维护区间操作和。

线段树的适用范围更广，线段树常用于一维区间问题，但不仅局限于区间问题。

Thanks

For Your Watching