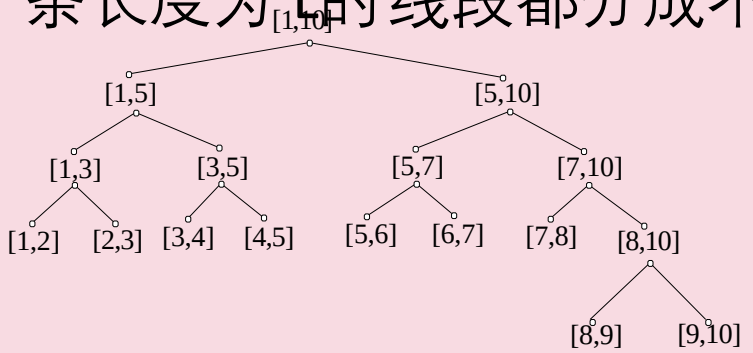
3. 线段树的性质

- 除最后一层外是满二叉树
- 长度范围为[1,L]的线段树，深度不超过$log(L-1)$
- 线段树上的结点个数不超过$2L$个
- 线段树把区间上的任意一条长度为L的线段都分成不超过$2logL$条线段



4. 线段树的存储方式

```
struct node {
    int a,b,left,right;
    int cover;
};
node tree[maxn];
```

- a、b表示结点的左右端点；left、right表示左右儿子的编号。一般这四个域一定要有。
- cover表示某线段被覆盖的次数，初始值应该是0。

5.1 线段树的建立

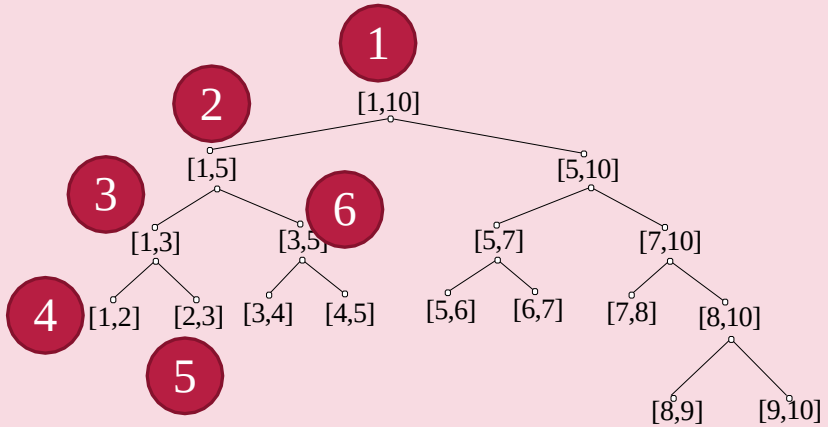
- 从树根开始构造线段树，维护一个全局变量total，表示结点编号：

```
void build_tree(int a, int b)
{
  int now, mid;
  total++;
  now = total;
  tree[now].a = a;
  tree[now].b = b;
  tree[now].cover = 0;
  if (a+1<b)
  {
    mid = (a+b) << 1;
    _____________________________;
    build_tree(a, mid);
    _____________________________;
    build_tree(mid, b);
  }

}
```

tree[now].left = total+1

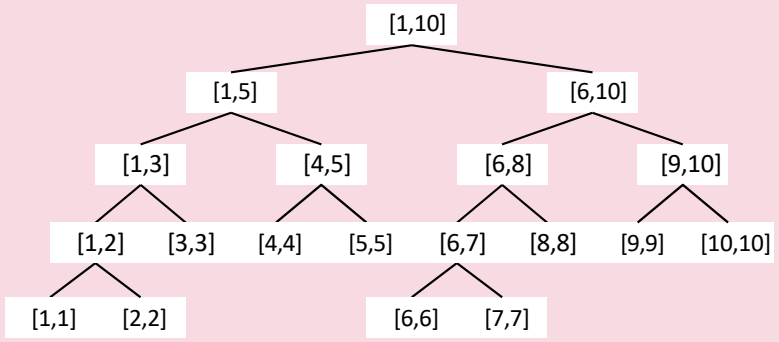
tree[now].right = total+1



仅仅给线段树安排好了位置，线段树还没入住!

5.1.2 用于数据统计的线段树

```
struct node
{
   int left, right;
   int tag, sum;
}
 void push_up(int x)
 {
   tree[x].sum =
     tree[x*2].sum +
     tree[x*2+1].sum;
 }
```




```
void build_tree(int x, int a, int b)
{
   tree[x].left = a;
   tree[x].right = b;
   tree[x].tag = 0;
   if (a==b)
   {
        tree[x].sum = s[a];
        return;
   }
   int mid = (a+b)>>1;
   build_tree(x*2, a, mid);
   build_tree(x*2+1, mid+1, b);
   push_up(x);
}
```

将孩子结点的信息更新到父亲结点

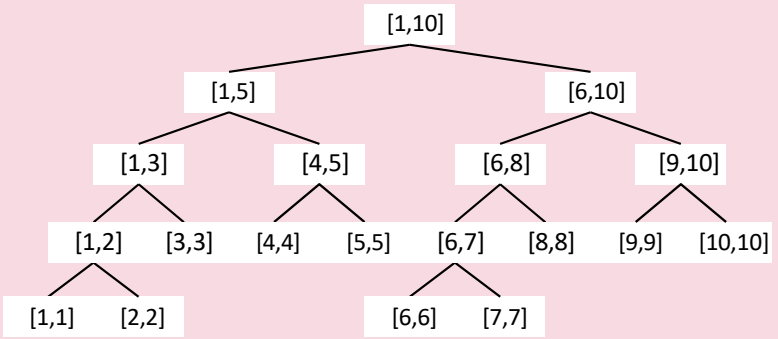
5.2 线段树的插入

- 对线段[a,b]每个值增加num。从第x结点开始找，找到区段后增加
- 如：插入[3,6]，update(1,3,6,num)

```
void update(int x, int a, int b, int num)
{
    if ((a<=tree[x].left) && (tree[x].right<=b))
    {
        _____________________________________________
        _____________________________________________;
        return;
    }
    int mid = (tree[x].left+tree[x].right)>>1;
    if (a<=mid)
        update(x*2, a, b, num);
    if (mid<b)
        update(x*2+1, a, b, num);
    ____________________;
}
```

tree[x].sum += (tree[x].right-tree[x].left+1)*num

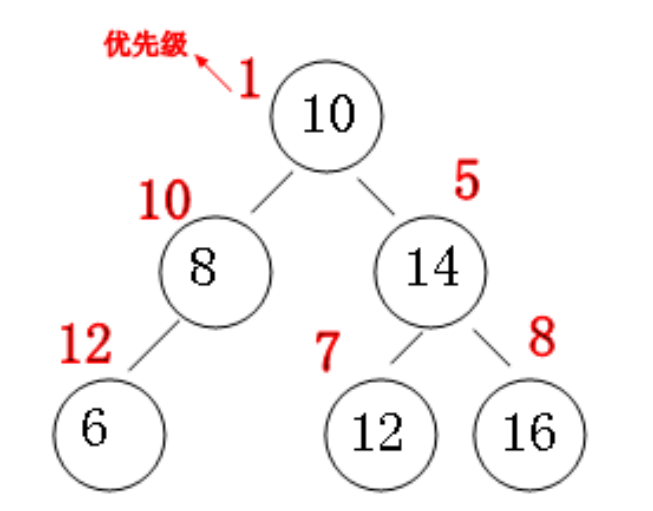
push_up(x)



找到[a,b]能全包含的区间就插入，而有交集的区间则进入该区间，去找能全包含的区间，然后插入。

TREAP

- Tree+ Heap
- 引入一个新的权值，要求树在满足二叉搜索树性质的同时新的权值满足堆的性质。



以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/776042033140010135