树微课Part2

许

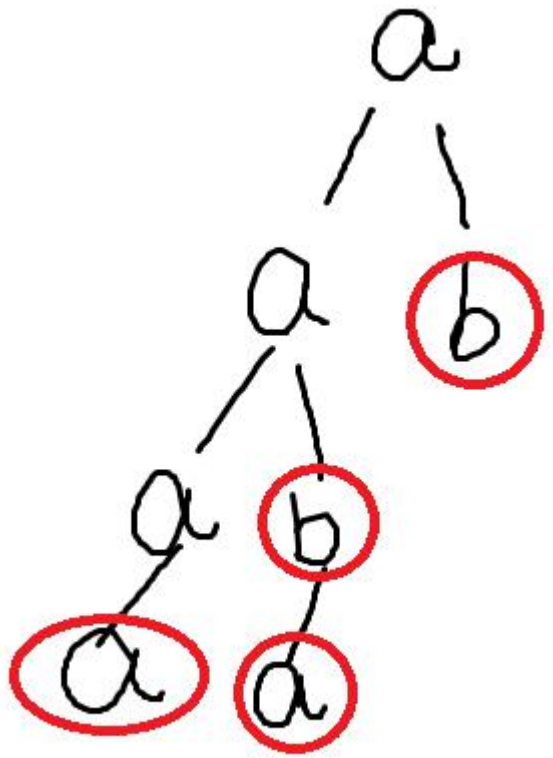
内容回顾

- 树的基本概念
- 树的遍历
  - 前序、中序、后序递归遍历
  - 前序+中序->构造树
  - 非递归遍历
- 二叉查找树
- 左儿子右兄弟表示

- 字典树
- 线段树
- 平衡二叉树
- 空间索引树

字典树

- aaaa
- aab
- aaba
- ab



字典树-结构定义

- struct TreeNode{
  bool isWord;

  TreeNode* child[26] //假设只有26个小写字母
- }
- 查找快速，空间占用交大

字典树-结构定义2

- struct TreeNode{
  - char ch;
  - bool isWord;
  - int childst;
  - int childnum;}
- vector<TreeNode> nodelist;
- vector<int> childindex
- node的儿子则为 nodelist[childindex[node.childst..node.childst+node.childnum-1]]

字典树的构造

```
void insert(TreeNode * r, const string & word)
{
    for (int i = 0; i<word.length(); i++)
    {
        int id = word[i] - 'a';
        if (r->child[id] == NULL)
        {
            r->child[id] = new TreeNode();
        }
        r = r->child[id];
    }
    r->isword = true;
}
```

字典树的查找

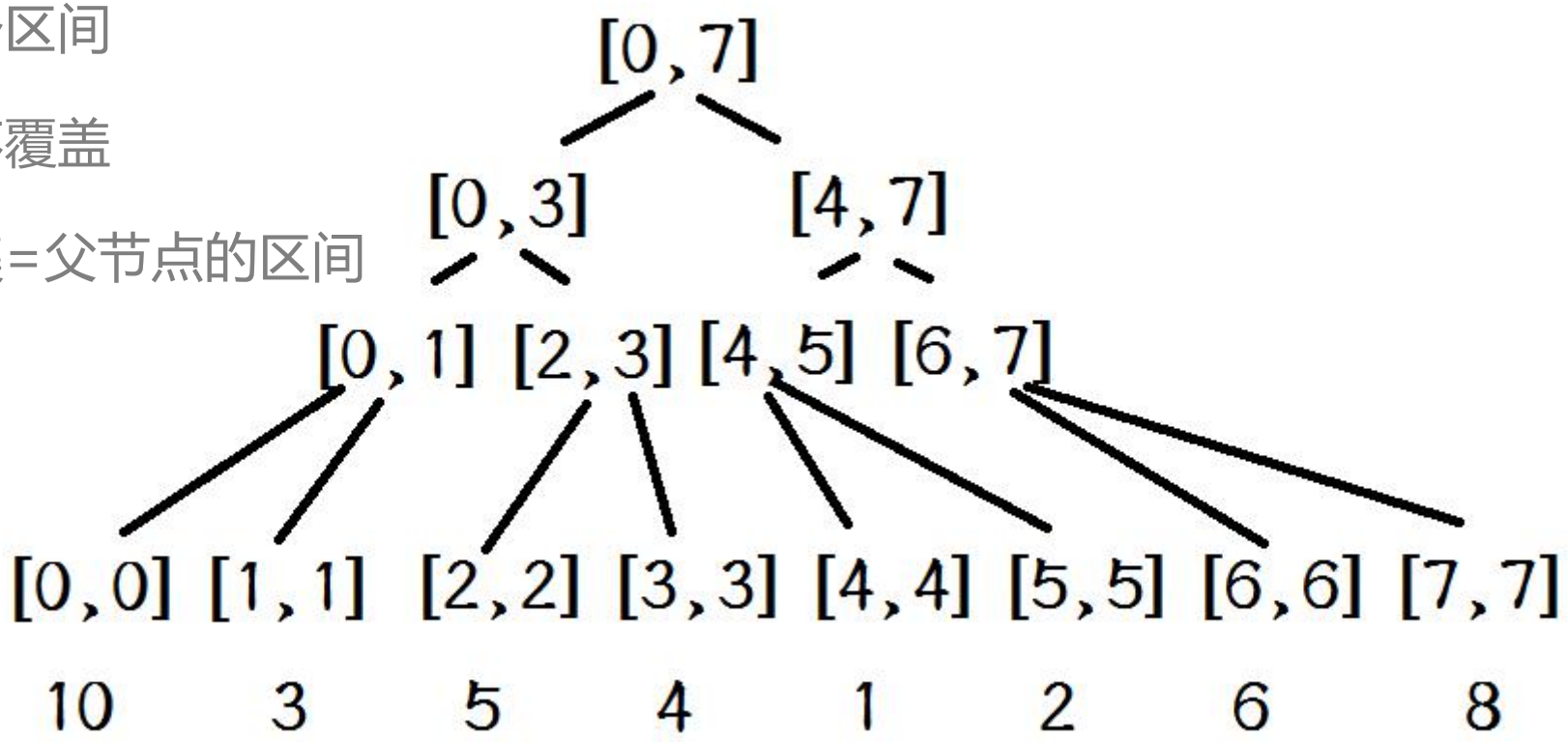
```
bool find(TreeNode * r, const string & word)
{
    for (int i = 0; i < word.length(); i++)
    {
        int id = word[i] - 'a';
        if (r->child[id] == NULL)
        {
            return false;
        }
        r = r->child[id];
    }
    return r->isword;
}
```

字典树-示例

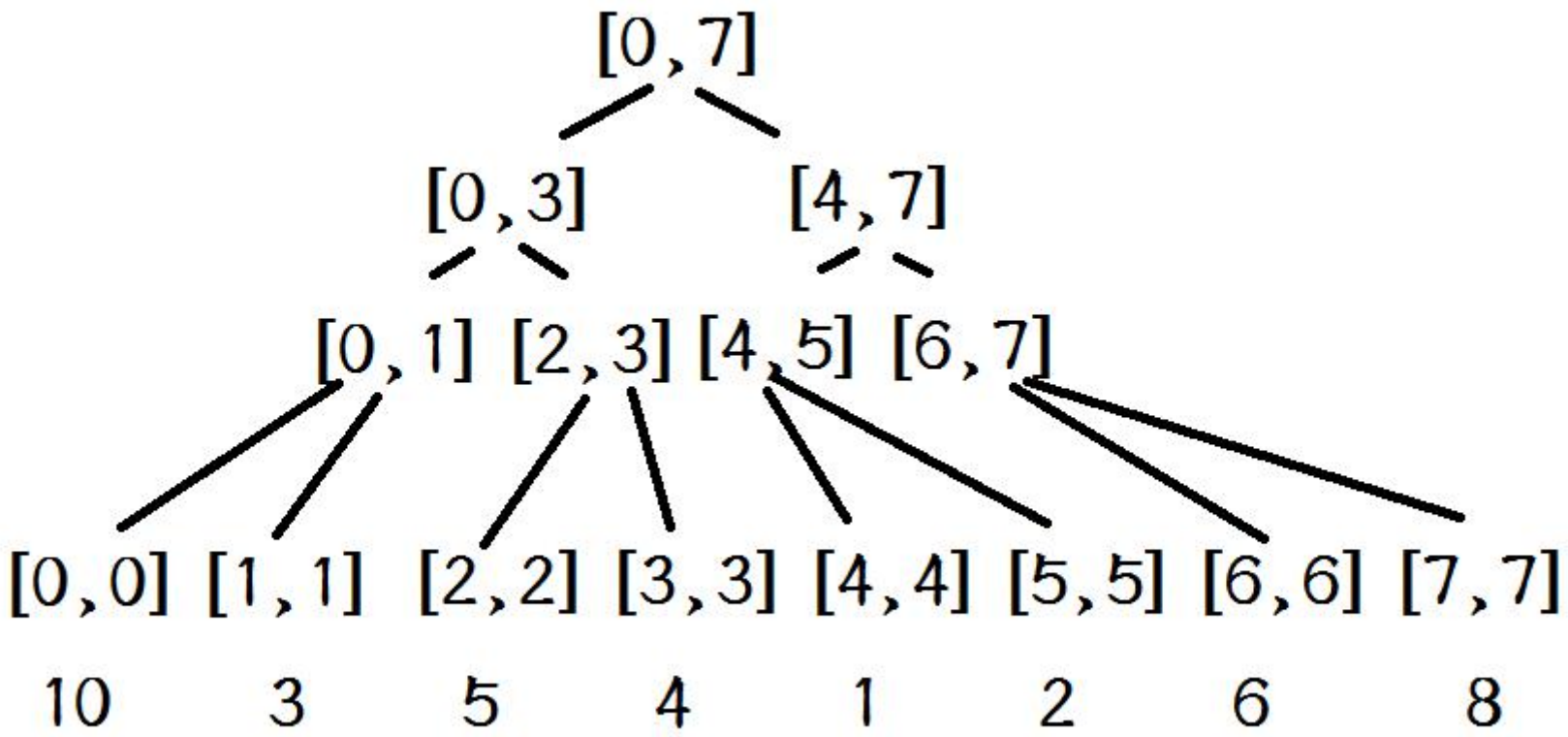
- LeetCode 139. Word Break
- LeetCode 140. Word Break2
- 需要储存字典，频繁查找一个串是不是落在字典里面都很适用

线段树-形式1

- 每个节点表示一个区间
- 子树的区间彼此不覆盖
- 子树结点区间并集=父节点的区间



- 求最小最大元素
- [0,1] ->(3,10)
- [2,3]->(5,4)
- [4,5]->(1,2)
- [6,7]->(6,8)
- [0,3]->(3,10)
- [4,7]->(1,8)
- [0,7]->(1,10)



线段树-更新

- 找到发生更改的叶子结点
- 重新计算父节点值
- 如果父节点值发生改变，继续计算再上一层结点值

线段树-形式1

- 典型题目
- 307. Range Sum Query - Mutable
  - Given nums = [1, 3, 5] 初始给定
  - 两种操作：

    sumRange(int i,int j)返回 nums[i..j]的和

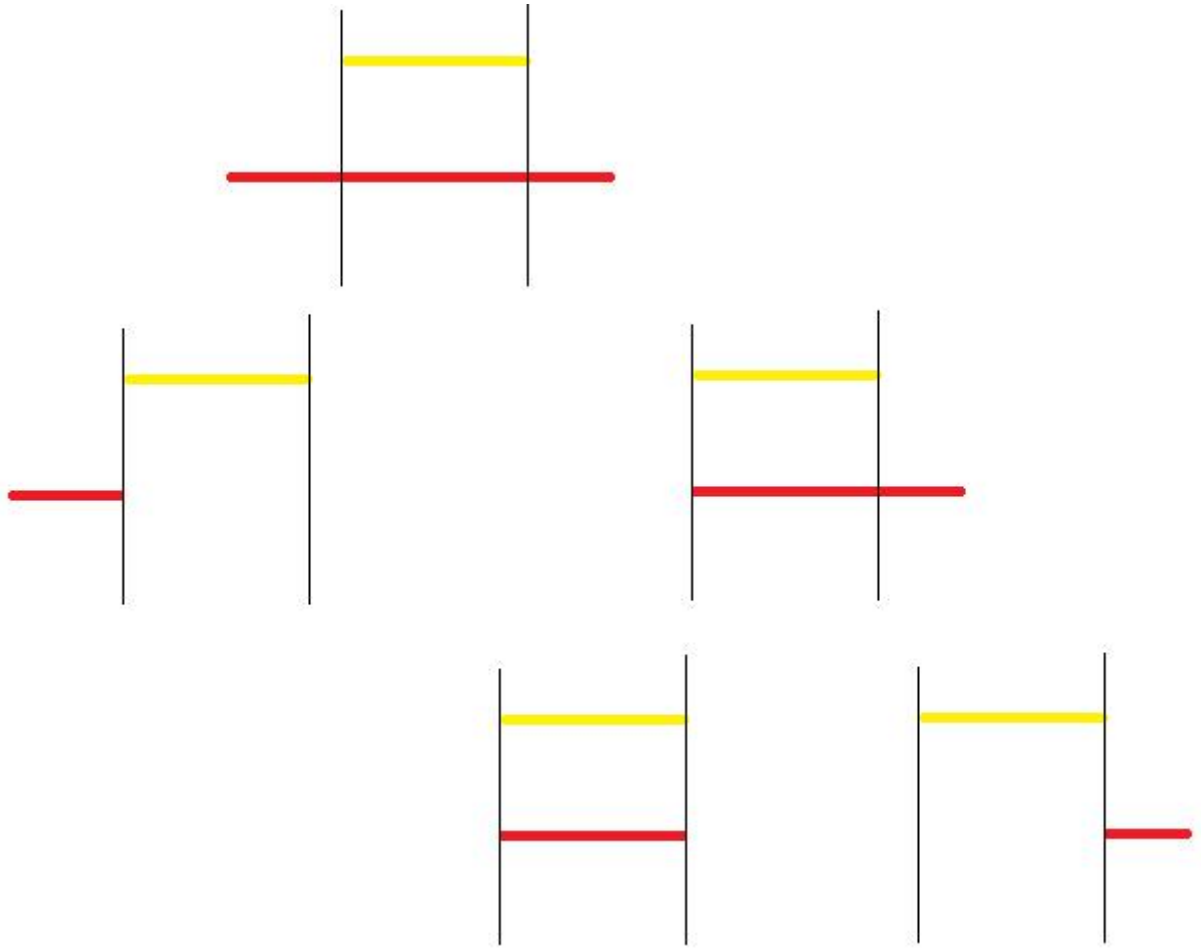
    update(int i,int val) 修改nums[i]=val;

线段树-形式2

- 一维坐标上有若干条线段 起点st,终点ed 0<=st<ed<=maxint
- 问一个点被多少条线段覆盖
- 操作insert(int st,int ed);
- 查询query(int x)
- 多次插入与查询

线段树-形式2-插入

- 坐标都是正整数
- 初始构造一个单结点树 [0,maxint]
- 插入线段(st,ed)



线段树-形式2-插入

- 不是叶子结点，则递归插入子节点

```
if (r->left != NULL && r->right != NULL)
{
    insert(r->left, st, ed);
    insert(r->right, st, ed);
}
```

- 叶子节点 st<=r.st && r.ed<=ed

  不需要分裂count++

```
if (st <= r->st && r->ed <= ed)
{
    r->count++;
}
```

线段树-形式2-插入

- 叶子节点r.st<st

  r分裂成[r.st..st-1][st..r->ed]

- 上条不成立且叶子节点r.ed>ed

  r分裂成[r.st..ed][ed+1..r->ed]

```
if (r->st < st )
{
    r->left = new SegTreeNode(r->count);
    r->right = new SegTreeNode(r->count);

    r->left->st = r->st;
    r->left->ed = st - 1;

    r->right->st = st;
    r->right->ed = r->ed;
    insert(r->right, st, ed);
}
else
if(r->ed>ed)
{
    r->left = new SegTreeNode(r->count);
    r->right = new SegTreeNode(r->count);

    r->left->st = r->st;
    r->left->ed = ed;

    r->right->st = ed+1;
    r->right->ed = r->ed;
    insert(r->left, st, ed);
}
```

线段树-形式2-查询

- int quert(SegTreeNode * r,int x)
- 如果r非叶子结点，则查询子节点，、

  根据x值与r->left.ed值的关系决定查询左子树还是右子树
- 如果r=叶子结点，则返回count

```
int query(SegTreeNode * r, int x)
{
    if (r != NULL)
    {
        if (r->left != NULL && r->right != NULL)
        {
            if (x <= r->left->ed)
            {
                return query(r->left, x);
            }
            else
            {
                return query(r->right, x);
            }
        }
        else
        {
            return r->count;
        }
    }
    return 0;
}
```

线段树-形式2-扩展

- 如果问题变成了query只需要返回

  x是否被线段覆盖了 不需要知道被多少线段覆盖了，应该如何修改

- 当插入过程中发现

  存在r的左右子结点都是叶子结点，且r->left->count==r->right->count

  可以如何优化

线段树-形式2-城市天际线

- [0..INT_MAX]=0;
- INSERT [ST,ED] -> H
- 最后求x处的最大H
- 按坐标输出整个线段树
- LeetCode 218

  注意原题的输入为 [Li,Ri) 转成闭区间插入的时候应该插入[Li,Ri-1]

红黑树

- 二叉查找树
- 保持了O(logn)的高度
- 所以查找插入删除的最坏代价也是O(logn)
- 查询过程同二叉查找树
- 难点在 如何保持删除与插入的时候树的平衡
- c++标准库的map使用的红黑树
- 细节请看 结构之法

B树

- 每个非叶子结点最多有M个子节点
- 根结点最少有2个子节点
- 非根非叶子结点最少有 m/2个子节点
- 所以叶子结点在同一层
- 扩展有B+ B*

空间索引树

- 在N维空间内查询距离点v<=d的点集合
  - 2维空间
    - 查询落在一个圆形内的点集合
    - 查询落在一个矩形范围内的点集合

R树

- 叶子结点=点
- 父节点=所有子节点的最小外接矩形
- R树区别于普通树的一个显著特点是，子树表示的范围可能会重叠
- 减少子树的矩形重叠，能够让R树更为优化
- 同B树，可以设计每层最多有M个子节点，最少有M/2个
- 大部分R树都能离线插入构造好树，在线查询，不进行动态插入。
- 所以建树可以进行更多的计算以优化 从而提高查询性能
- 大规模数据往往有自然属性可以很容易的分成不同的子集

总结

- 思考清楚每个结点代表什么
- 非叶子结点的值是否可以解决问题？
  - 是需要递归查询子节点呢
  - 还是直接可以返回当前结点信息
- 把大问题分成子问题 子树的问题解决了，当前树的问题也就解决了
- m叉树的分裂与收缩其实原理类似，一通百通

- 微博：小小喵78
- Q&A