

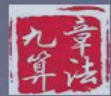
# 课程尚未开始 请大家耐心等待

关注微信公共账号，获得最新面试题信息及解答



Weibo:

<http://www.weibo.com/ninechapter>



# 第二讲 高级数据结构(上)

九章高级算法班 第2章  
[www.ninechapter.com](http://www.ninechapter.com)



# Overview

1. Segment Tree
2. Union Find



# 1. Segment Tree



# Segment Tree

线段树的性质：

1. 二叉树
2. Parent 区间被平分  
Leftson = 左区间  
Rightson = 右区间
3. 每个节点表示区间



# Segment Tree

查找操作：

节点区间和要查找区间的关系

四种情况：

1. 节点区间**包含**查找区间—>查找区间递归向下
2. 节点区间**不相交于**查找区间 ->查找区间停止搜索
3. 节点区间**相交不包含于**查找区间->查找区间分裂成两段区间，一段于被节点区间包含，另一段不相交
4. 节点区间**相等于**查找区间—> 返回值查找的结果



# Segment Tree

Build tree

自上而下递归分裂

自下而上回溯更新

Modify tree :

自上而下递归查找

自下而上回溯更新



# Segment Tree 形态

1. 求最大、最小型
2. 求和型
3. 计数型





# Interval sum(Build,Query)

<http://www.lintcode.com/en/problem/interval-sum/>



# Interval sum ii(Modify)

<http://www.lintcode.com/en/problem/interval-sum-ii/>



# Segment Tree 不同构建方式

Build tree:

1.以数组的下标来建立线段树

2.以数值来建立线段树

下标型线段树 和 值型的线段树的区别是什么呢？



# Segment Tree - Count segment tree query ii

<http://www.lintcode.com/en/problem/segment-tree-query-ii/>



# Count of Smaller Number

<http://www.lintcode.com/en/problem/count-of-smaller-number/>



# Count of Smaller Number

<http://www.lintcode.com/en/problem/count-of-smaller-number/>

[1,4,3] , 3

- 1. for 循环遍历  $O(n)$
- 2. 排序二分  $O(\log(n))$
- 3. 线段树  $O(\log(m))$



# Count of Smaller Number before itself

<http://www.lintcode.com/en/problem/count-of-smaller-number-before-itself/>

1. 两层for 循环遍历  $O(n^2)$
2. 线段树。
  1. insert an element.
  2. query a range



# Count of Smaller Number before itself

<http://www.lintcode.com/en/problem/count-of-smaller-number-before-itself/>

难点：

1. 怎么样把问题转换为区间问题？
2. 怎么样把区间问题联想到线段树？
3. 怎么样把把问题分解为插入和查询两步？





# 线段树

操作:

1. Query
2. Modify
3. Build

应用:

1. Sum
2. Maximum/ Minimum
3. Count

构建形式:

1. 下标作为建立区间
2. 值作为建立区间



# Union Find



# 两大操作

1. 查询 Find  
递归 > 非递归

2. 合并 Union



# 两大操作

## 1. 查询 Find

```
int find(int x){  
    int parent = father.get(x);  
    while(parent != father.get(parent)) {  
        parent = father.get(parent);  
    }  
    return parent;  
}
```

```
HashMap<Integer, Integer> father = new HashMap<Integer, Integer>()
```

# 两大操作

## 2. 合并 Union

```
void union(int x, int y){  
    int fa_x = find(x);  
    int fa_y = find(y);  
    if(fa_x != fa_y)  
        father.put(fa_x, fa_y);  
}
```

```
HashMap<Integer, Integer> father = new HashMap<Integer, Integer>()
```



# Find the Connected Component in the Undirected Graph

<http://www.lintcode.com/en/problem/find-the-connected-component-in-the-undirected-graph/>



# Find the Connected Set in the Directed Graph

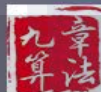
<http://www.lintcode.com/zh-cn/problem/find-the-connected-set-in-the-directed-graph/>



# 两大操作

## 1. 查询 compressed\_find

```
int compressed_find(int x){  
    int parent = father.get(x);  
    while(parent != father.get(parent))  
        parent = father.get(parent);  
}  
int temp = -1;  
int fa = father.get(x);  
while(fa != father.get(fa)) {  
    temp = father.get(fa);  
    father.put(fa, parent);  
    fa = temp;  
}  
return parent;  
}
```

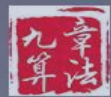




# Number of Islands

[www.lintcode.com/zh-cn/problem/number-of-islands](http://www.lintcode.com/zh-cn/problem/number-of-islands)

二维和一维标号转换小技巧



# Number of Islands II

<http://www.lintcode.com/zh-cn/problem/number-of-islands-ii/>



# Summary

数据结构的题目：

通过分析需要什么操作来找到适合的数据结构进行使用。

线段树：1. 区间Get max/min, sum, count

2. 线段树的三个性质 (I. 这个是一个二叉树, II. 每个节点表示一段区间的max或者sum. III. 非叶子节点：左右儿子等于叶子区间平分后的左右区间, 值等于左右儿子的值的更新。)

3. 了解线段树三个接口 build, query, modify.

4. 下标型和值型线段树

并查集：1. 合并

2. 查找

