

二叉树中的分治法与遍历法

Divide Conquer & Traversal

课程版本 v6.1

主讲 令狐冲



扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: [ninechapter](#)

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

知乎: <http://zhuankan.zhihu.com/jiuzhang>

官网: <http://www.jiuzhang.com>

- 二叉树（**Binary Tree**）问题的考点剖析
- 第一类考察形态：求值，求路径类二叉树问题
- 第二类考察形态：结构变化类二叉树问题
- 第三类考察形态：二叉查找树（**Binary Search Tree**）类问题
 - 非递归（**Non-recursion or Iteration**）版本的中序遍历（**Inorder Traversal**）

请在随课教程中先修如下内容：

什么是遍历（**Traverse**）

什么是分治（**Divide Conquer**）

递归三要素是什么

递归（**Recursion**），搜索（**Search**），遍历（**Traverse**），分治（**Divide Conquer**）之间的联系和区别是什么

什么是先序遍历（**Pre-order**），中序遍历（**In-order**），后序遍历（**Post-order**）

什么是结果类 **ResultType**，什么时候使用 **ResultType**

什么是二叉查找树（**Binary Search Tree**），他有什么特性？他有什么作用？

二叉树的高度是多少？

A: $O(n)$

B: $O(\log n)$

C: $O(h)$

二叉树的高度是多少？

最坏 $O(n)$ 最好 $O(\log n)$
一般用 $O(h)$ 来表示更合适

考察形态：二叉树上求值，求路径

代表例题：<http://www.lintcode.com/problem/subtree-with-maximum-average/>

考点本质：深度优先搜索（Depth First Search）

考察形态：二叉树结构变化

代表例题：<http://www.lintcode.com/problem/invert-binary-tree/>

考点本质：深度优先搜索（Depth First Search）

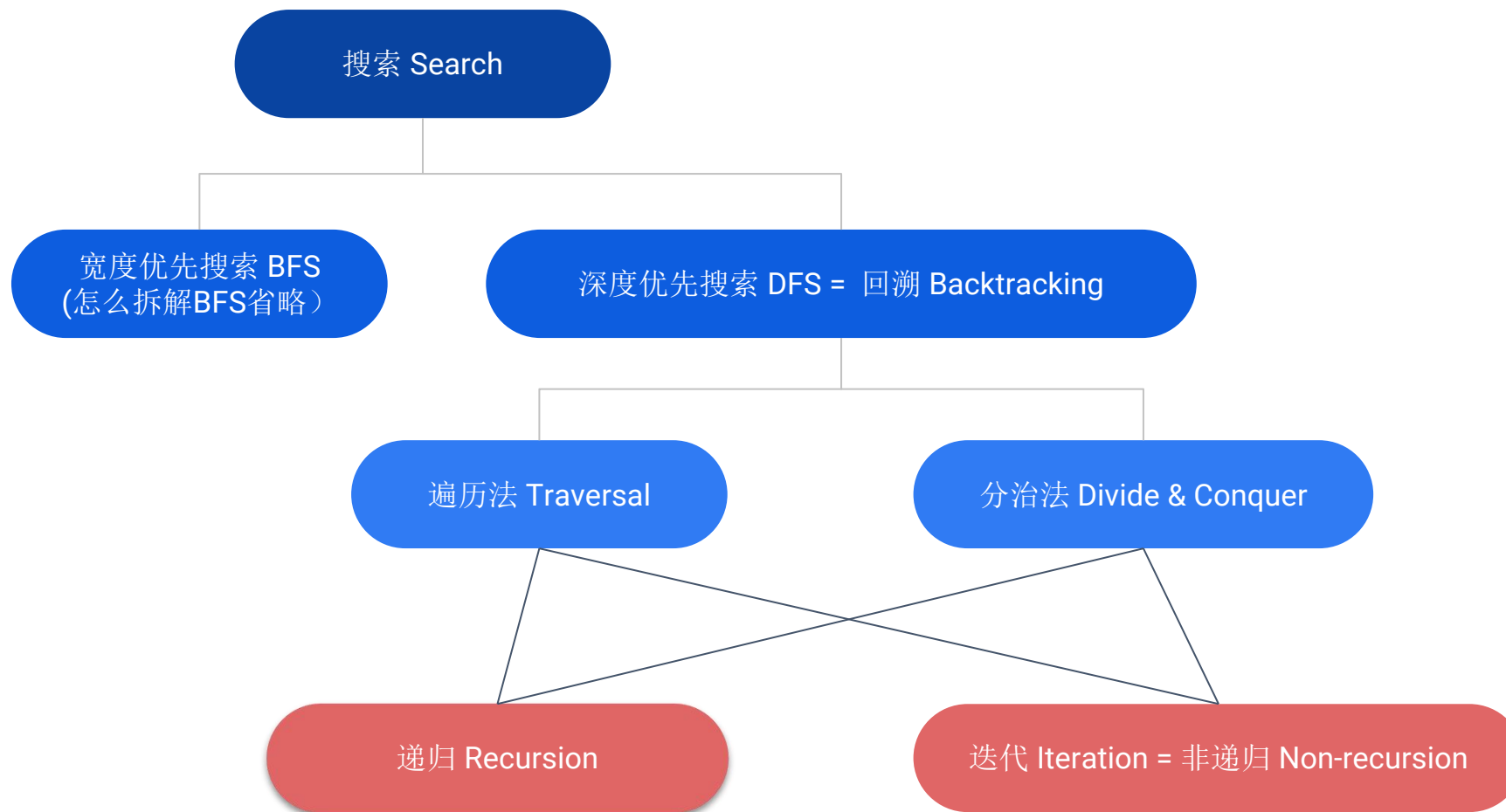
考察形态：二叉查找树（Binary Search Tree）

代表例题：<http://www.lintcode.com/problem/validate-binary-search-tree/>

考点本质：深度优先搜索（Depth First Search）

Tree-based Depth First Search

不管二叉树的题型如何变化
考点都是基于树的深度优先搜索



将递归和非递归理解为算法的一种实现方式而不是算法

独孤九剑 —— 破枪式

碰到二叉树的问题，就想想整棵树在该问题上的结果
和左右儿子在该问题上的结果之间的联系是什么

第一类考察形态

二叉树上求值，求路径

Maximum / Minimum / Average / Sum / Paths

Minimum Subtree

<http://www.lintcode.com/problem/minimum-subtree/>

<http://www.jiuzhang.com/solutions/minimum-subtree/>

求和最小的子树

Binary Tree Paths

<http://www.lintcode.com/problem/binary-tree-paths/>

<http://www.jiuzhang.com/solutions/binary-tree-paths/>

求从根（root）到叶（leaf）的所有路径

Lowest Common Ancestor

<http://www.lintcode.com/problem/lowest-common-ancestor/>

<http://www.jiuzhang.com/solutions/lowest-common-ancestor/>

with parent pointer vs no parent pointer

follow up: LCA II & III

休息5分钟

Take a break

第二类考察形态

二叉树结构变化

Flattern Binary Tree to Linked List

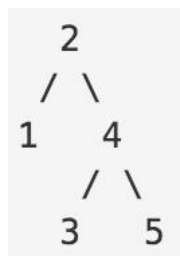
<http://www.lintcode.com/problem/flatten-binary-tree-to-linked-list/>

<http://www.jiuzhang.com/solutions/flatten-binary-tree-to-linked-list/>

第三类考察形态

二叉搜索树
Binary Search Tree

- 从定义出发：
 - 左子树都比根节点小
 - 右子树都不小于根节点
- 从效果出发：
 - 中序遍历 in-order traversal 是“不下降”序列
 - 如图，中序遍历为 1 2 3 4 5



- 性质：
 - 如果一棵二叉树的中序遍历不是“不下降”序列，则一定不是BST
 - 如果一棵二叉树的中序遍历是不下降，也未必是BST
 - 比如下面这棵树就不是 BST，但是它的中序遍历是不下降序列。
 - 1
 - / \
 - 1 1

Kth Smallest Element in BST

<https://www.lintcode.com/problem/kth-smallest-element-in-a-bst/>

<https://www.jiuzhang.com/solution/kth-smallest-element-in-a-bst/>

时间复杂度如何分析？

BST的高度是多少

A: $O(n)$

B: $O(\log n)$

C: $O(h)$

BST的高度是多少

同样是最坏 $O(n)$ 最好 $O(\log n)$

用 $O(h)$ 表示更合适

只有 **Balanced Binary Tree** (平衡二叉树) 才是 $O(\log n)$

Follow up: 二叉树经常被修改

如何优化 kthSmallest 这个操作？

在 `TreeNode` 中增加一个 `counter`，代表整个树的节点个数

也可以用一个 `HashMap<TreeNode, Integer>` 来存储某个节点为代表的子树的节点个数

在增删查改的过程中记录不断更新受影响节点的 `counter`

在 `kthSmallest` 的实现中用类似 `Quick Select` 的算法去找到 `kth smallest element`

时间复杂度为 $O(h)$ ， h 为树的高度。

Strong Hire: 能够答出 `Follow Up` 的算法，并写出 `kthSmallest` 核心代码（不需要写增删查改，45分钟写不完的）， `bug free or minor bug`，不需要提示

Hire / Weak Hire : 能够答出 `Follow up` 的算法，大致写出 `kthSmallest` 核心代码，存在一定bug，或者需要提示

No Hire: 答不出 `follow up`

Strong No: 连第一问的 `Inorder traversal` 都不会写

Binary Search Tree Iterator

<http://www.lintcode.com/problem/binary-search-tree-iterator/>

<http://www.jiuzhang.com/solution/binary-search-tree-iterator/>

阅读全文并背诵

该 Iterator 算法即 non-recursion 的 inorder traversal，不仅仅适用于 BST，任何 Binary Tree 都可以

- stack 中保存一路走到当前节点的所有节点
- stack 的栈顶 一直存储 iterator 指向的当前节点
- hasNext() 只需要判断 stack 是否为空
- next() 只需要返回 stack 的栈顶值，并将 iterator 挪到下一个点，对 stack 进行相应的变化

挪到下一个点的算法如下：

- 如果当前点存在右子树，那么就是右子树中“一路向左”最左边的那个点
- 如果当前点不存在右子树，则是走到当前点的路径中，第一个左拐的点

相关题：

<http://www.lintcode.com/problem/inorder-successor-in-binary-search-tree/>

<http://www.lintcode.com/problem/validate-binary-search-tree/> 不用递归

<http://www.lintcode.com/problem/binary-tree-inorder-traversal/> 不用递归

Closest Binary Search Tree Value

<https://www.lintcode.com/problem/closest-binary-search-tree-value/>

<http://www.jiuzhang.com/solution/closest-binary-search-tree-value/>

如果使用中序遍历，时间复杂度是多少？

如果使用 lowerBound / upperBound 的方法，时间复杂度是多少？

Follow up: 寻找 k 个最接近的值

<https://www.lintcode.com/problem/closest-binary-search-tree-value-ii/>

<https://www.jiuzhang.com/solution/closest-binary-search-tree-value-ii/>

如果是用中序遍历得到从小到大的所有值，接下来的问题相当于之前学过的哪个题？

有没有更快的办法？

Strong Hire

找1个点和找k个点都答出来，且找 k 个点的能用 $O(k + \log n)$ 的时间复杂度

Hire

找1个点和找k个点都答出来，且找 k 个点的能用 $O(k \log n)$ 的时间复杂度完成，少 bug，无需提示

Weak Hire:

找1个点和找k个点都答出来，且找 k 个点的能分别用 $O(k \log n)$ 和 $O(n)$ 的时间复杂度完成，bug 多，需要提示

No Hire

答出1个点，答不出 k 个点非 $O(n)$ 的算法

Strong No Hire

啥都答不出来

Related Questions

- Search Range in Binary Search Tree
- <http://www.lintcode.com/problem/search-range-in-binary-search-tree/>
- Insert Node in a Binary Search Tree
- <http://www.lintcode.com/problem/insert-node-in-a-binary-search-tree/>
- Remove Node in a Binary Search Tree
- <http://www.lintcode.com/problem/remove-node-in-binary-search-tree/>
- <http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/9-BinTree/BST-delete.html>

请在随课教程中自学如下内容：

- **Morris 算法**：使用 $O(1)$ 的额外空间复杂度对二叉树进行先序遍历（**Preorder Traversal**）
- 用非递归的方法实现先序遍历，中序遍历和后序遍历
- 二叉查找树（**Binary Search Tree**）的增删查改
- **Java** 自带的平衡排序二叉树 **TreeMap / TreeSet** 的介绍和面试中的应用