解决99%二叉树问题的算法——分治法 Divide & Conquer

主讲人 令狐冲
课程版本 v7.0

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像

一经发现，将被追究法律责任和赔偿经济损失

分治法 Divide Conquer

将大规模问题拆分为若干个小规模的**同类型问题**去处理的算法

分治法和二分法（Binary Search）有什么区别?

什么样的数据结构适合二分法？

数组：一个大数组可以拆分为若干个不相交的子数组

归并排序，快速排序，都是基于数组的分治法

二叉树：整棵树的左子树和右子树都是二叉树

二叉树的大部分题都可以使用分治法解决

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

二叉树的高度是多少?

A: O(n)

B: O(logn)

C: O(h)

二叉树的高度是多少?

最坏 O(n) 最好 O(logn)

一般用 O(h) 来表示更合适

二叉树考点剖析

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/

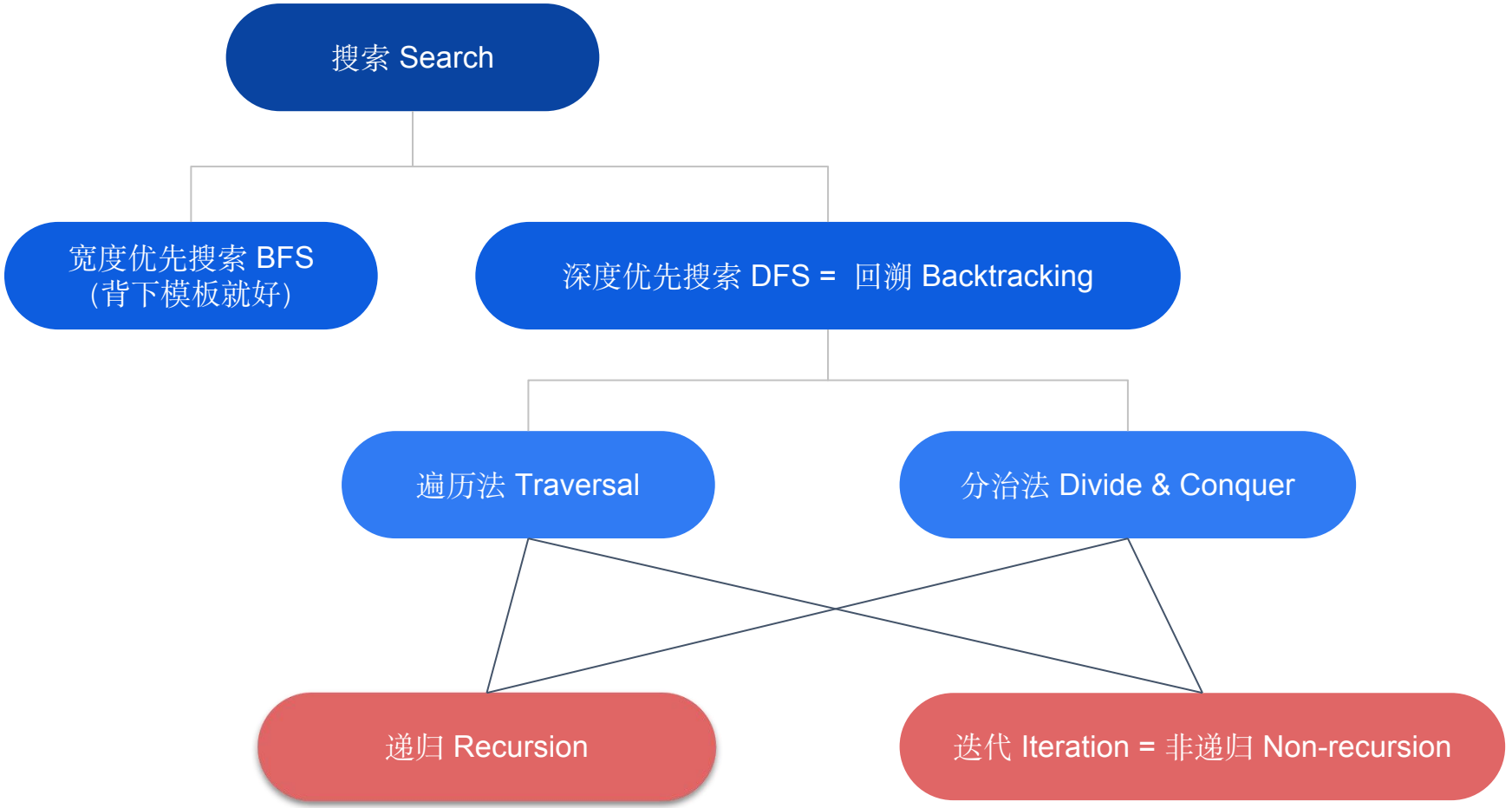
考点本质：深度优先搜索（Depth First Search）

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

一张图搞明白：递归，DFS，回溯，遍历，分治，迭代



将递归和非递归理解为算法的一种实现方式而不是算法

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

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

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/

尽可能避免使用全局变量

容易写出 BUG

可以把需要修改的变量作为参数传入到函数里

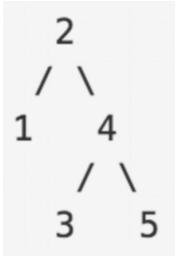
第三类考察形态

二叉搜索树
Binary Search Tree

BST 基本性质

- 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

BST的高度是多少

A: O(n)

B: O(logn)

C: O(h)

BST的高度是多少

同样是最坏 O(n) 最好 O(logn)

用 O(h) 表示更合适

只有 Balanced Binary Tree (平衡二叉树) 才是 O(logn)

红黑树 Red-black Tree

红黑树是一种 Balanced BST

C++/Java中有一个数据结构的实现用的是 Balanced BST

你知道是什么？

Java: TreeMap / TreeSet

C++: map / set

Python: 不好意思不提供

有一定要用 TreeMap 来做的面试题么？——没有

关于红黑树，你需要掌握的是

- 知道他是一个 Balanced BST
- 知道他能干嘛
  - O(LogN) 的时间内实现增删查改
  - O(LogN) 的时间内实现找最大找最小
  - O(LogN) 的时间内实现找比某个数大的最小(upperBound)和比某个数小的最大(lowerBound)
- 知道他的工程应用价值
  - Java 1.8 中的 HashMap 的实现里同时用到了 TreeMap 和 LinkedList

就面试而言，你不需要掌握他的代码实现，因为写出来200+行，没人能在面试的时候写完

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?

Follow up: 二叉树经常被修改

如何优化 kthSmallest 这个操作?

优化方法

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

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？

Closest Binary Search Tree Value 评分标准

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 + logn) 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(klogn) 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(klogn) 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

在互动课中继续学习如下二叉树的内容

- 第 22 章 后序遍历非递归与 Morris 算法
  - 不作为必须要掌握的知识点，但是学了可以提高 Coding 能力
- 第 23 章 二叉查找树的增删查改
  - 必须掌握增查改，删除操作不作要求