

基础算法和数据结构高频题 II



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](#)

知乎专栏: <http://zhuanlan.zhihu.com/jiuzhang>

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

官网: www.jiuzhang.com

- Missing Interval 这题，我们遇到了什么样的corner case？（多选）
 - a. 溢出MaxInt b. 空二叉树 c. 输入数组为空
- Sliding window 类问题通用的套路是？
 - _____减少一个，_____增加一个（填空）
- 第三节课讲到计算时间复杂度时，哪一句话所代表的方法很常用？
 - 每个元素只会被访问_____遍（填空）

- 北京时间:2017-10-31 10:00 - 11:30 下周二
美西时间:2017-10-30 19:00 - 20:30 下周一
美东时间:2017-10-30 22:00 - 23:30 下周一
- <http://www.jiuzhang.com/qa/6328/>

- 二分查找回顾
- BST类问题 (2题)
- 二叉树类问题 (3题)

二分查找回顾

- 二分查找：
 - 给 n 个数，从小到大排序，数字不重复，再给一个待查找的数 x ，问 x 是否存在于这 n 个数中，存在返回下标，不存在返回-1

- 二分查找算法：
 - 找到位于数组中点的数mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$

0	1	2	3	4	5	6	7	8
2	5	7	9	10	13	20	32	35

- 二分查找算法：
 1. 找到中点mid
 2. x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 3. 重复以上两步

找到9的位置 $x = 9$

0	1	2	3	4	5	6	7	8
2	5	7	9	10	13	20	32	35



- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$

0	1	2	3	4	5	6	7	8
2	5	7	9	10	13	20	32	35



- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$

0	1	2	3
2	5	7	9



- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$

0	1	2	3
2	5	7	9



- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$

0	1	2	3
2	5	7	9



- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$


2	3
7	9



- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$

2	3
7	9


mid

- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

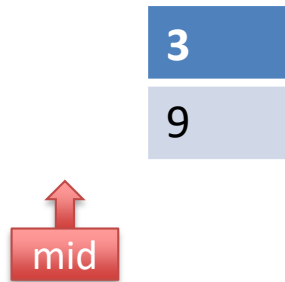
找到9的位置 $x = 9$

2	3
7	9

↑
mid

- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$



- 二分查找算法：
 - 找到中点mid
 - x和中点mid比较：
 - $x < \text{mid}$: 砍掉mid右边
 - $x > \text{mid}$: 砍掉mid左边
 - $x = \text{mid}$: 退出
 - 重复以上两步

找到9的位置 $x = 9$



时间复杂度 $O(\log N)$

二分查找类题目：

Guess Number Game	Search for a Range	Search a 2D Matrix
Maximum Number in Mountain Sequence	Find Minimum in Rotated Sorted Array	Smallest Rectangle Enclosing Black Pixels
Search in a Big Sorted Array	First Bad Version	Find Peak Element
Rectangle Enclosing Black Pixels	Last Position of Target	Closest Number in Sorted Array

BST类问题

Convert BST to Greater Tree

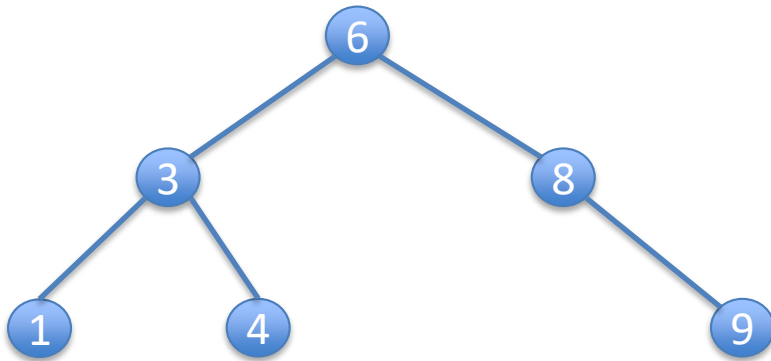
<http://www.lintcode.com/zh-cn/problem/convert-bst-to-greater-tree/>

<http://www.jiuzhang.com/solutions/convert-bst-to-greater-tree/>

Convert BST to Greater Tree

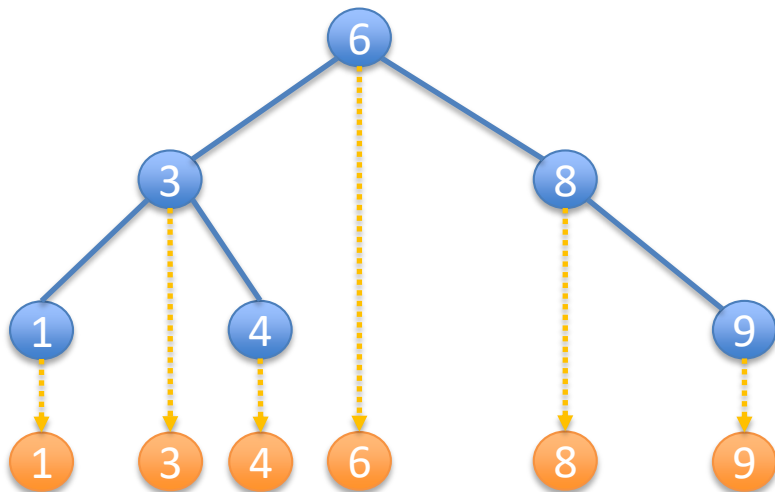
BST的性质:

- BST的定义?
 - 左边都比root小，右边都比root大

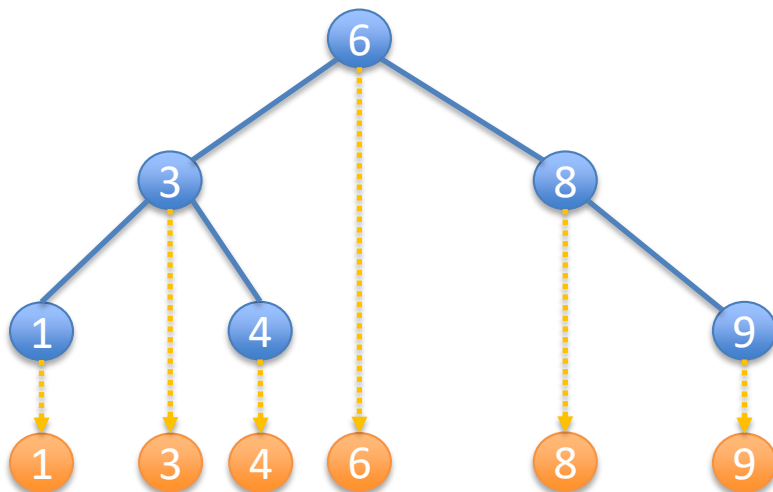


Convert BST to Greater Tree

- 把BST规范的画出来是什么样子？
 - 向下投影出的序列是从小到大的顺序



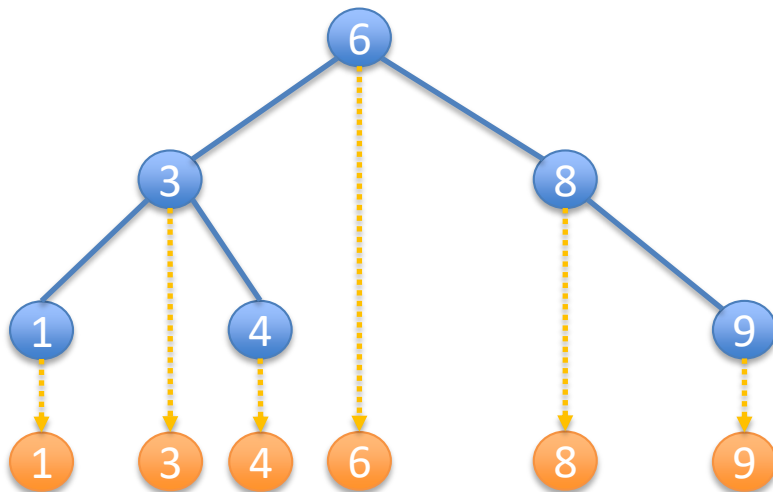
- BST的中根遍历（Inorder 左中右），节点访问顺序是什么样的？
 - 小到大的顺序



Convert BST to Greater Tree

此题思路：

- 如果是右中左这样的“反中根遍历”，节点访问顺序会是什么样的？
 - 从大到小的顺序



Convert BST to Greater Tree

算法流程:

1. 定义累计求和变量 $\text{sum} = 0$
2. 按照右中左这样的“反中根遍历”，依次访问每个节点
 - ① 更新累计求和变量 $\text{sum} = \text{sum} + \text{当前节点的值}$
 - ② 更新当前节点的值 $= \text{sum}$

时间复杂度 $O(N)$

Convert BST to Greater Tree

代码实现:

```
12 public class Solution {
13     /**
14      * @param root the root of binary tree
15      * @return the new root
16      */
17     int sum = 0;
18
19     void dfs(TreeNode cur) {
20         if (cur == null) {
21             return;
22         }
23         dfs(cur.right);
24         sum += cur.val;
25         cur.val = sum;
26         dfs(cur.left);
27     }
28
29     public TreeNode convertBST(TreeNode root) {
30         // Write your code here
31         dfs(root);
32         return root;
33     }
34 }
```

- 怎么理解这一段DFS (Depth First Search 深度优先搜索)?

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

DFS的两种理解方式:

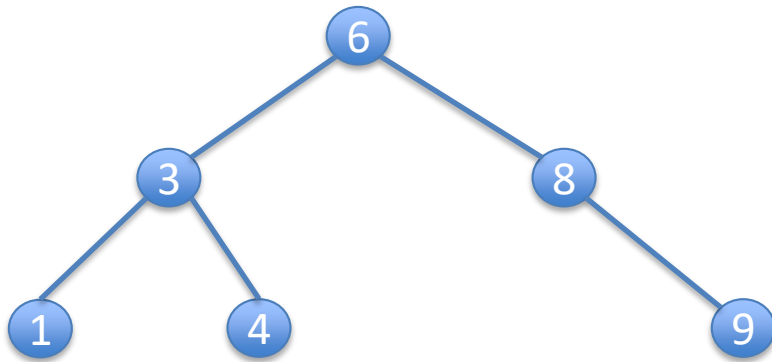
1. 按照实际执行顺序模拟
2. 按照DFS的定义宏观理解

Convert BST to Greater Tree

- 怎么理解这一段DFS (Depth First Search 深度优先搜索)?

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

- 按照实际执行顺序模拟



- 怎么理解这一段DFS (Depth First Search 深度优先搜索)?

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

2. 按照DFS的定义宏观理解:

我们这里**DFS**函数的定义是什么? 任务是什么?

1. “反中根遍历”的顺序访问**cur**这颗树每个节点
2. **sum**要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和

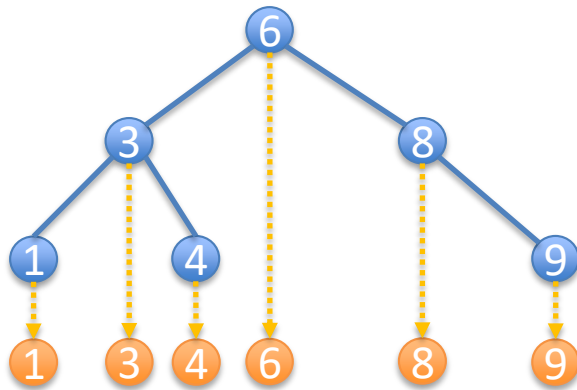
Convert BST to Greater Tree

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }  
28  
29 public TreeNode convertBST(TreeNode root) {  
30     // Write your code here  
31     dfs(root);  
32     return root;  
33 }  
34 }
```

convertBST(board) : 给你root=6这棵树，帮我把这三件事干了
dfs (CEO): 好的

任务:

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和



Convert BST to Greater Tree



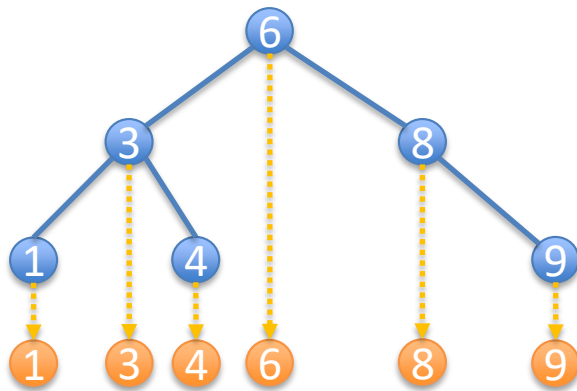
九章算法

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

dfs (CEO): 我的任务是对
cur=6 这棵树做三件事

任务:

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和



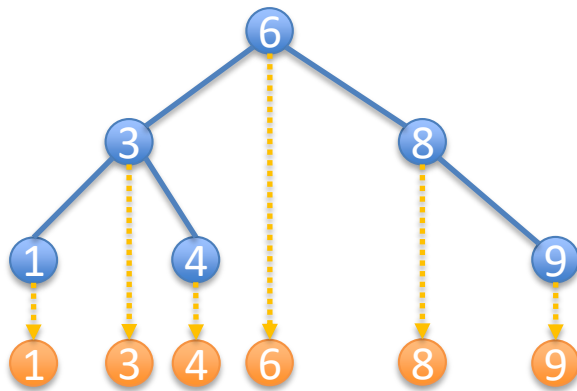
Convert BST to Greater Tree

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

dfs (CEO): 我先检查下cur=6
这颗树是否是空的，空的话
我就完事交差了

任务：

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和



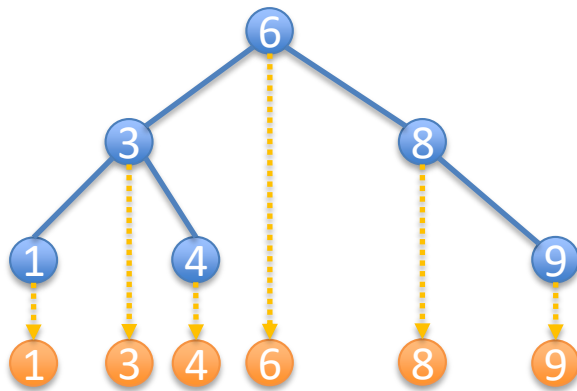
Convert BST to Greater Tree

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

dfs (CEO): 既然cur=6不为空, 那我们来分配下任务吧: 23行的dfs, 你是总监, 你先帮我把8为根的这棵右子树做了, 任务还是这三样, 等你结果
dfs@23: 🙌

任务:

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和



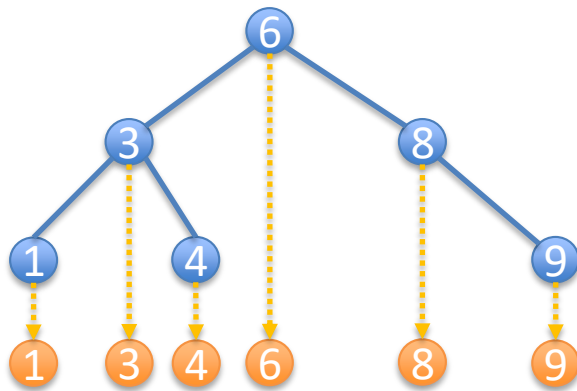
Convert BST to Greater Tree

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

dfs@23: 报告老大，任务已完成，sum 已经加上了访问过的元素的值，所有元素的已经变成大于等于它的元素之和
dfs (CEO): 👍

任务：

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和



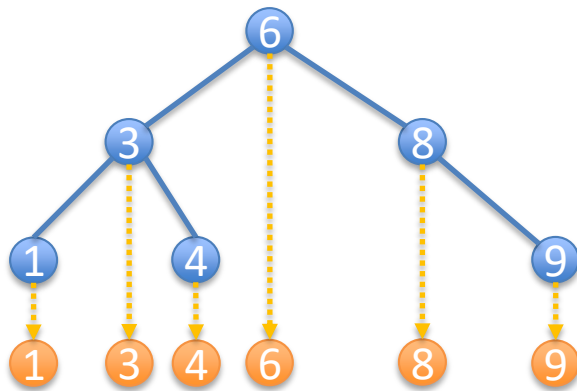
Convert BST to Greater Tree

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

dfs (CEO): 总监dfs@23已经完成了他的工作, 我要在他工作的基础之上完成自己的工作@24 25: sum 加上当前的值, 当前的值变为所有比它的大元素之和, 也就是sum

任务:

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和



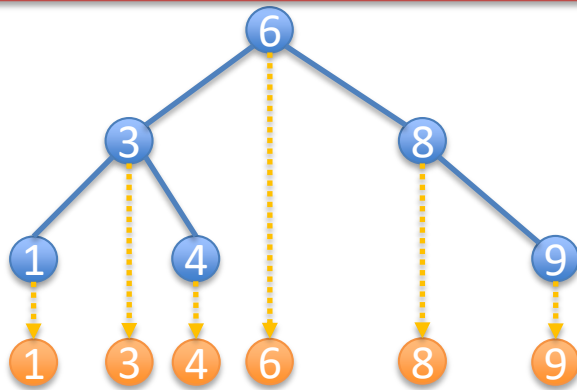
Convert BST to Greater Tree

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

dfs (CEO): 好了, dfs@26, 你也是总监, dfs@23和我的工作都做完了, sum现在是有所有大于等于6的元素的和, 条件已经给你创造好了, 剩下的以3为根的这棵左子树就交给你了
dfs@26: 放心, 包在我身上 😎

任务:

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和



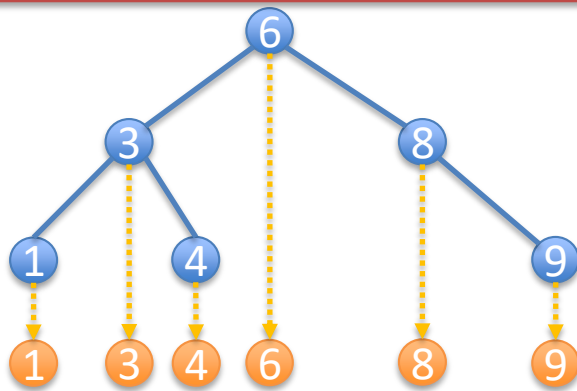
Convert BST to Greater Tree

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

dfs@26: 报告老大，全部做完
dfs(CEO): 不错，所有工作都做完了，
可以向董事会汇报了
全体: 🙌

任务:

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和



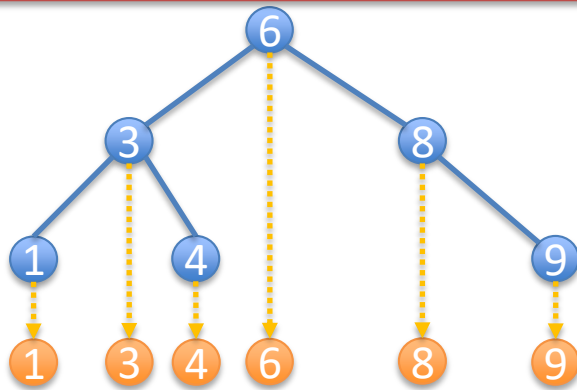
Convert BST to Greater Tree

```
19 void dfs(TreeNode cur) {  
20     if (cur == null) {  
21         return;  
22     }  
23     dfs(cur.right);  
24     sum += cur.val;  
25     cur.val = sum;  
26     dfs(cur.left);  
27 }
```

任务:

1. “反中根遍历”的顺序访问cur这颗树每个节点
2. sum要在当前的基础上要加上所有访问元素的值
3. 所有访问元素要变成大于等于它的元素之和

其实在做的过程中，总监把任务分给了经理，经理又分配给了主管，主管又分配给了各级干活的。。。他们的任务都是这三件事，直到某个干活的发现分给他的树是空的，他就啥也没干，高兴的直接交差了



Convert BST to Greater Tree

考点:

- 基础数据结构上的灵活操作
- BST遍历的性质

能力维度:

3. 基础数据结构/算法

◆ 小技巧总结:

- DFS的两种理解方式:

1. 按照实际执行顺序模拟 (适合枚举型DFS, 下节课内容)
2. 按照DFS的定义宏观理解 (适合分治型DFS, 本节课内容)

Inorder Successor in Binary Search Tree

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

<http://www.jiuzhang.com/solutions/inorder-successor-in-binary-search-tree/>

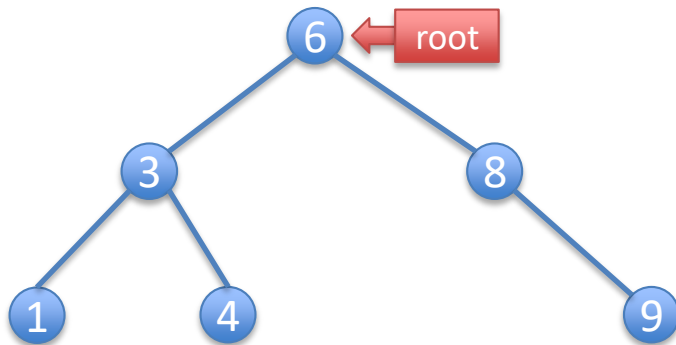
名词解释:

- 节点的**successor** 后继就是比给定节点大的所有节点中最小的那个

Inorder Successor in Binary Search Tree

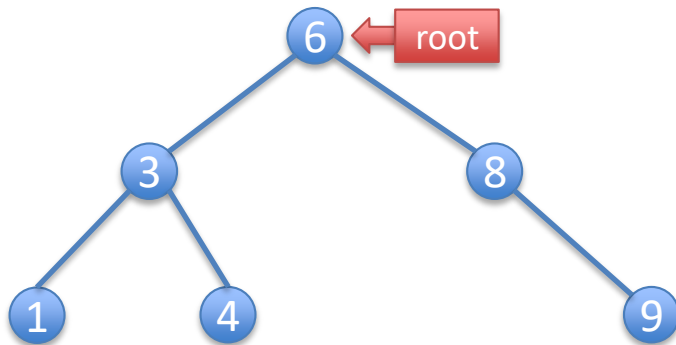
思路:

- 参考二分法, 考虑p和root之间的关系 (p 为给定的节点, 要找到p节点的后继)



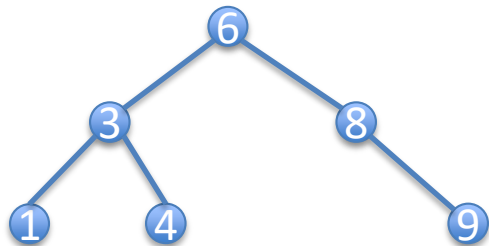
Inorder Successor in Binary Search Tree

- 两种情况：
 1. root的值 \leq p的值 答案就在右子树中
 2. root的值 $>$ p的值 答案有两个备选：
 - a) 就是root
 - b) 左子树中找（如果找到就一定是它，因为左子树中的元素都比根小）

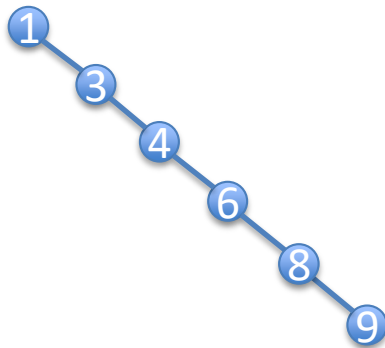


Inorder Successor in Binary Search Tree

- 时间复杂度是多少？
 - $O(h)$ h 是树的高度
- 树高 h 和元素个数 n 之间的关系？

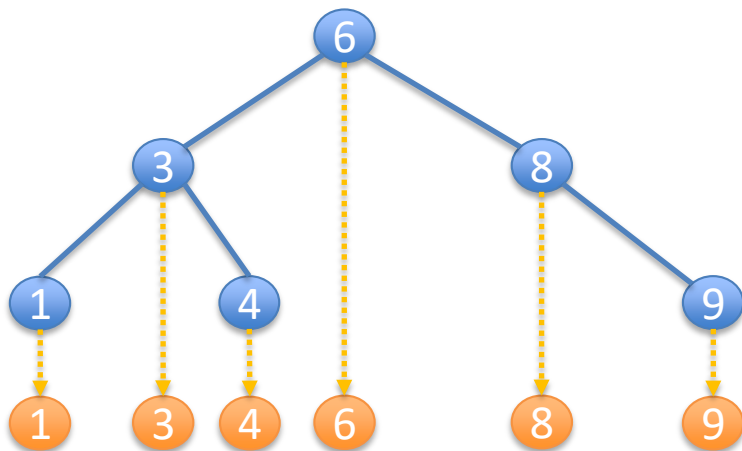


树比较平衡的时候 $h \approx \log(n)$

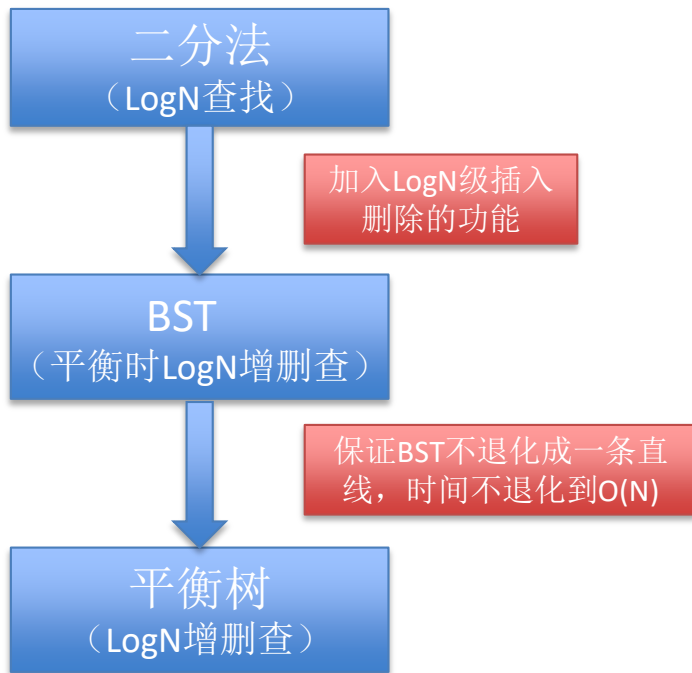


树极不平衡的时候 $h \approx n$

- BST与二分法的关系：
 - 将BST规范的画出来，可以看到BST和二分查找非常类似，几乎是等价的，BST上有着和二分查找一样的思考方式，root就是二分查找中每次查找的mid，每次和root分情况讨论就等价于二分查找中和mid分情况讨论



- BST与二分法的关系:



◆ 小技巧总结:

- 遇到BST上操作的问题，可以拿给定的节点（区间）与root做比较，分类讨论、分而治之

扩展问题:

- BST上求一段的和

- Company Tags: Facebook

考点:

- 基础数据结构上的灵活操作
- **BST**对根比较后的分类讨论

能力维度:

3. 基础数据结构/算法
4. 逻辑思维/算法优化能力

- BST类问题以及其他树、二叉树上的问题常考数据结构上的灵活操作
- 解决BST类问题要从BST的性质着手（画出来从小到大、类似二分查找），经常与root比较后分类讨论
- BST类题目：
 - [Validate binary search tree](#)
 - [Binary search tree iterator](#)

二叉树类问题

Binary Tree Flipping

<http://www.lintcode.com/zh-cn/problem/binary-tree-flipping/>

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

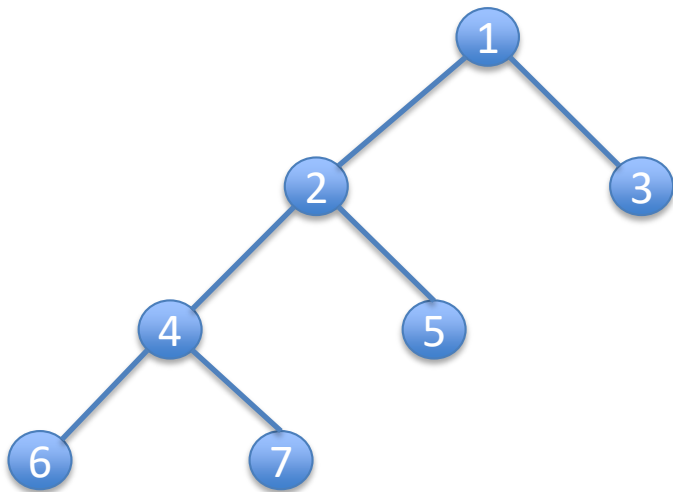
思路:

- 问题一：这到底是一棵什么样的树？
 - 试着把这个树画出来

Binary Tree Flipping

思路：

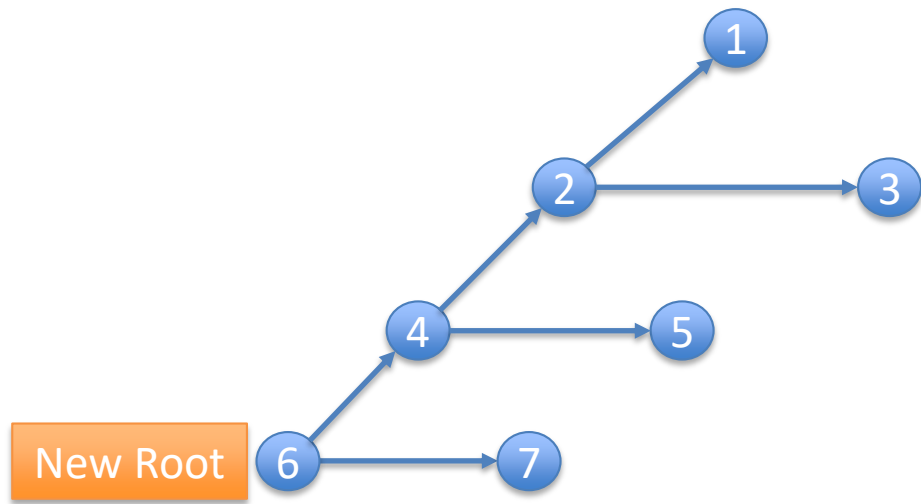
- 问题一：这到底是一棵什么样的树？
 - 试着把这个树画出来



Binary Tree Flipping

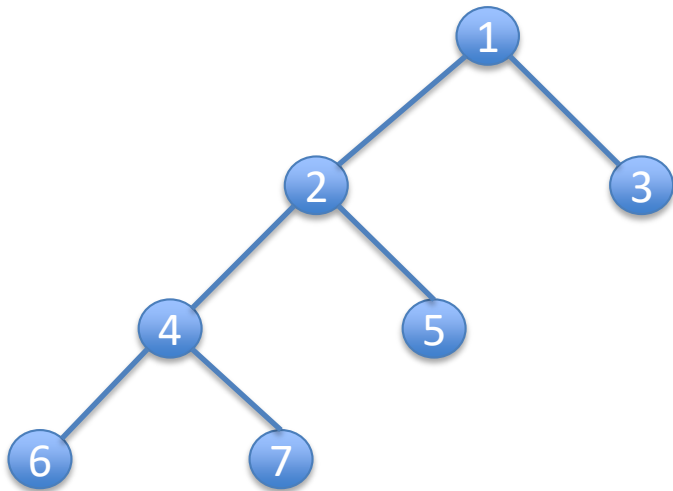
思路:

- 问题二: 按题目要求翻转后是一棵什么样的树?

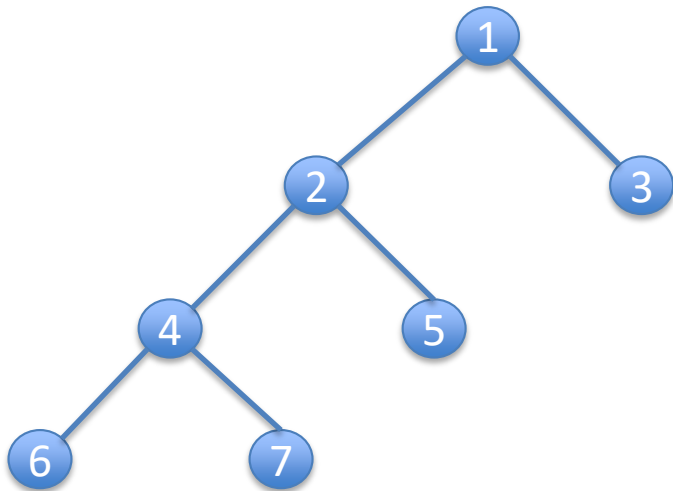


- 问题三：用什么算法实现？
 - DFS 递归翻转

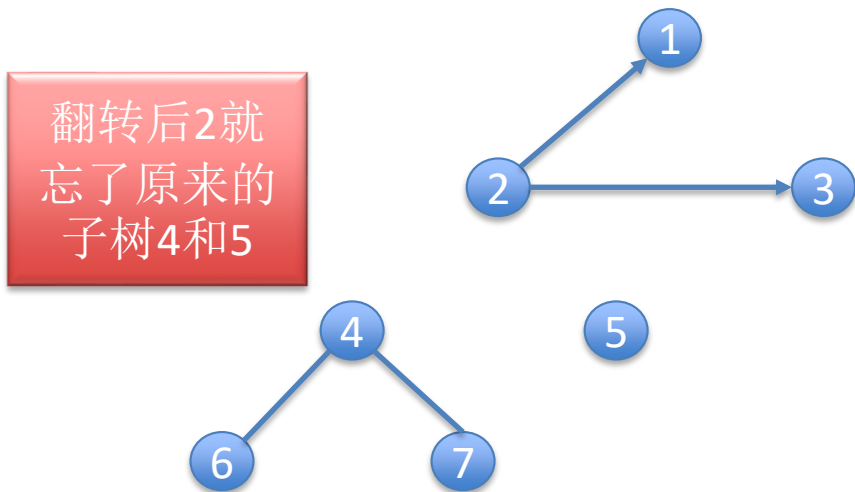
- 问题四：是先翻转当前节点再递归(a)，还是先递归再翻转当前节点(b)?
 - 想想怎样做使得上下节点互不影响
 - 同学们会怎么选?



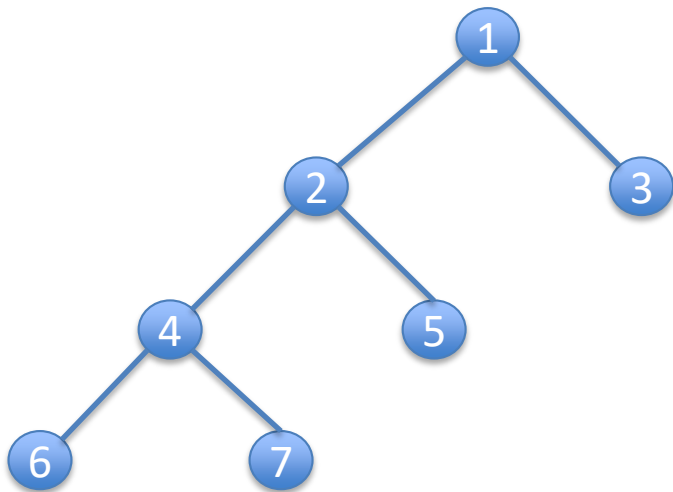
- 问题四：是先翻转当前节点再递归(a)，还是先递归再翻转当前节点(b)?
 - 想想怎样做使得上下节点互不影响
 - 假如是先翻转再递归(a)?



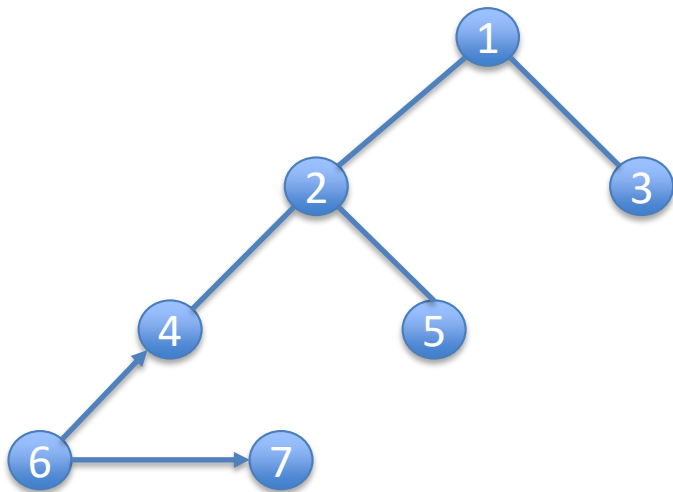
- 问题四：是先翻转当前节点再递归(a)，还是先递归再翻转当前节点(b)?
 - 想想怎样做使得上下节点互不影响
 - 假如是先翻转再递归(a)?



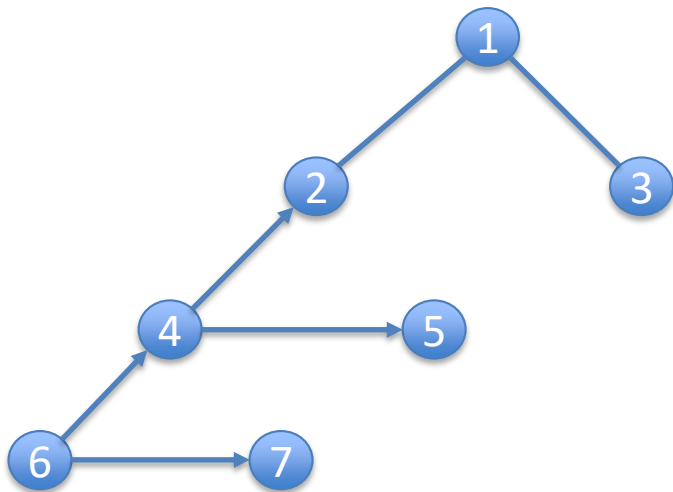
- 问题四：是先翻转当前节点再递归(a)，还是先递归再翻转当前节点(b)?
 - 想想怎样做使得上下节点互不影响
 - 所以是？



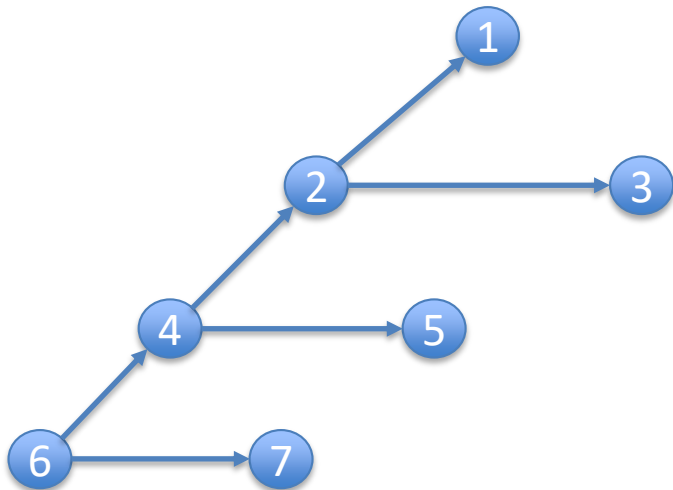
- 问题四：是先翻转当前节点再递归(a)，还是先递归再翻转当前节点(b)?
 - 想想怎样做使得上下节点互不影响
 - 所以是？



- 问题四：是先翻转当前节点再递归(a)，还是先递归再翻转当前节点(b)?
 - 想想怎样做使得上下节点互不影响
 - 所以是？



- 问题四：是先翻转当前节点再递归(a)，还是先递归再翻转当前节点(b)?
 - 想想怎样做使得上下节点互不影响
 - 所以是？
 - 代码实现？



时间复杂度 $O(N)$

- Company Tags: LinkedIn

考点:

- 基础数据结构上的灵活操作
- 二叉树的遍历，遍历后的形状改变

能力维度:

1. 理解问题
3. 基础数据结构/算法

Binary Tree Leaves Order Traversal

<http://www.lintcode.com/zh-cn/problem/binary-tree-leaves-order-traversal/>

<http://www.jiuzhang.com/solutions/binary-tree-leaves-order-traversal/>

思路:

- 一层一层的剥开二叉树，如何确定这一层包含哪些节点？
 - 画出来看看

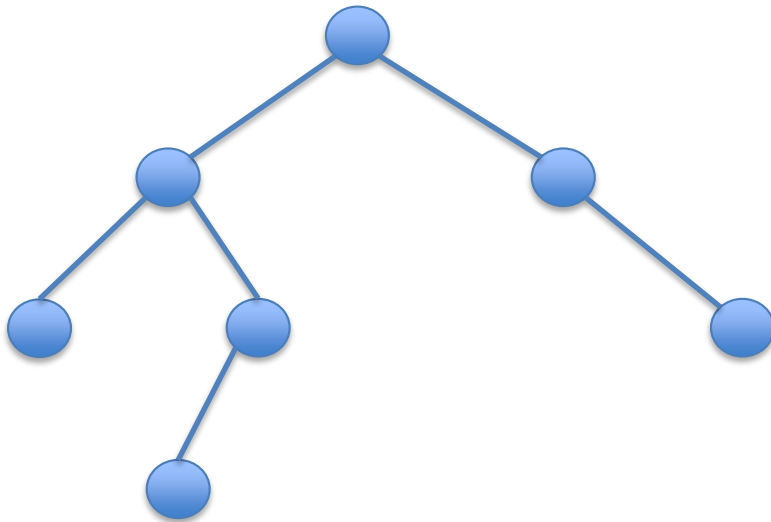
Binary Tree Leaves Order Traversal



九章算法

思路:

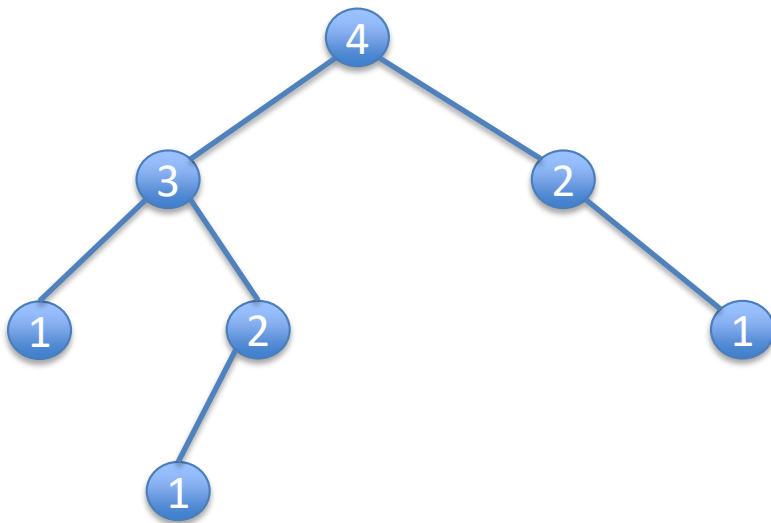
- 一层一层的剥开二叉树，如何确定这一层包含哪些节点？
 - 画出来看看



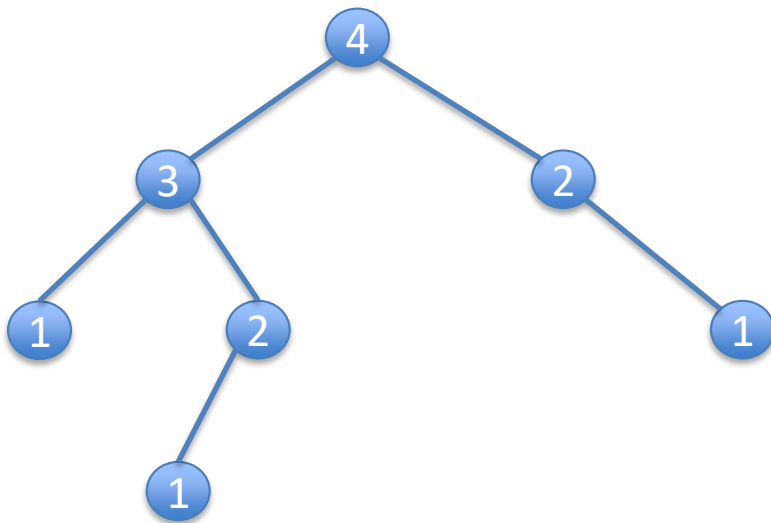
Binary Tree Leaves Order Traversal

思路:

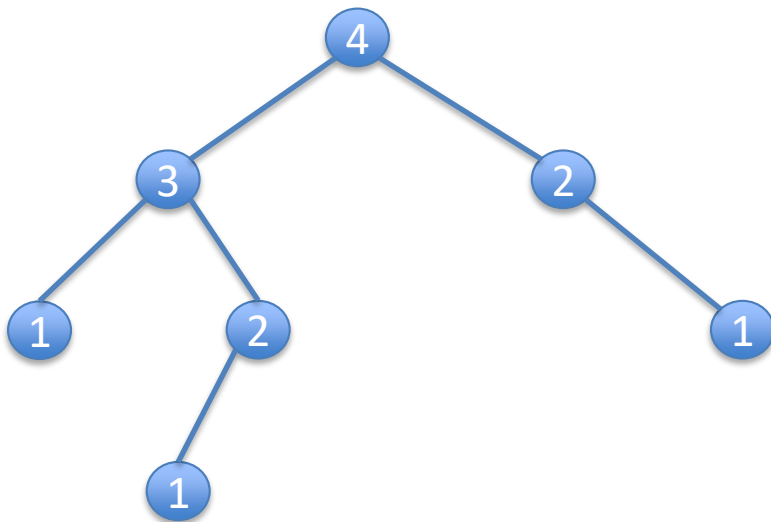
- 一层一层的剥开二叉树，如何确定这一层包含哪些节点？
 - 画出来看看



- 剥开的顺序有什么规律呢？
 - 其实第 k 层包含的就是所有高度为 k 的节点

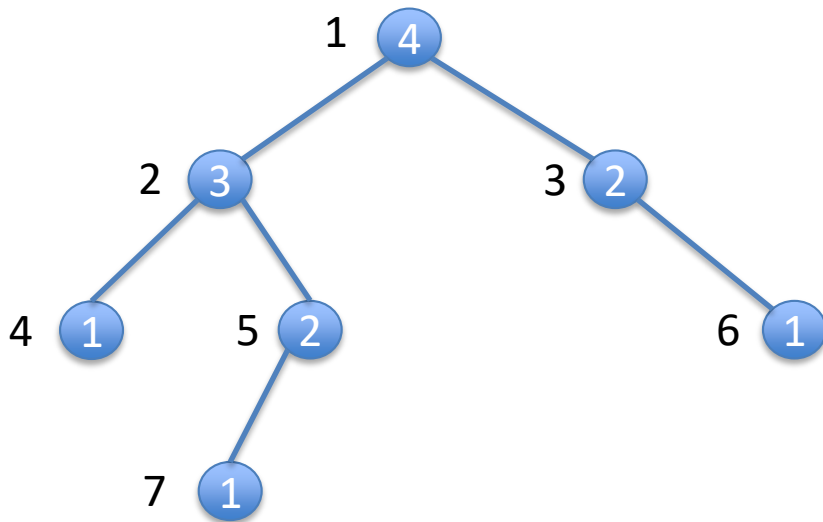


- 所以怎么求高度？怎么保存答案？
 - DFS计算节点高度，hash 保存答案



Binary Tree Leaves Order Traversal

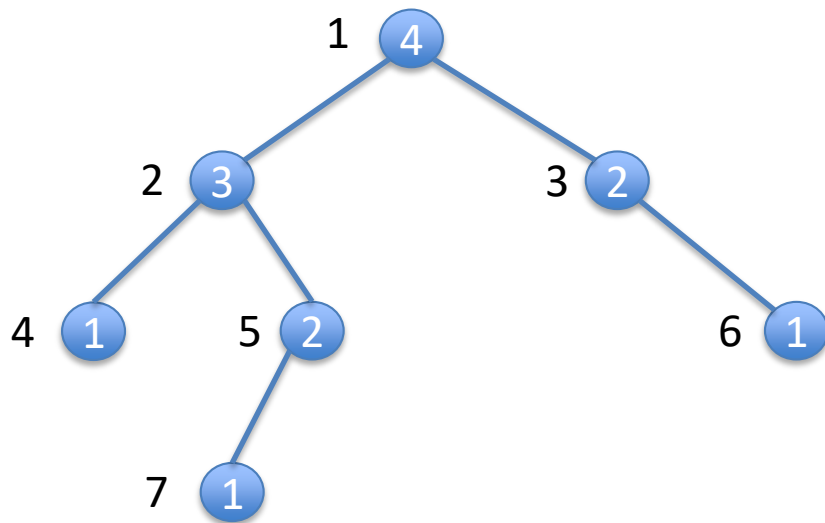
- 所以怎么求高度？怎么保存答案？
 - DFS计算节点高度，hash 保存答案



1	2	3	4

Binary Tree Leaves Order Traversal

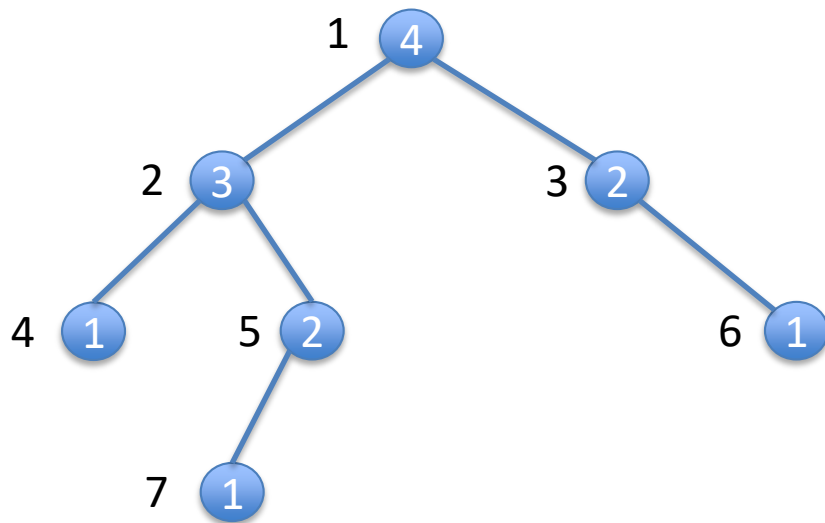
- 所以怎么求高度？怎么保存答案？
 - DFS计算节点高度，hash 保存答案



4	5	2	1
7	3		
6			
1	2	3	4

Binary Tree Leaves Order Traversal

- 怎么实现?
 - 见代码



时间复杂度 $O(N)$

4	5	2	1
7	3		
6			
1	2	3	4

- Company Tags: LinkedIn

考点:

- 基础数据结构上的灵活操作
- 二叉树的遍历

类似题:

- <http://www.lintcode.com/zh-cn/problem/maximum-subtree/>

能力维度:

3. 基础数据结构/算法

Binary Tree Vertical Order Traversal

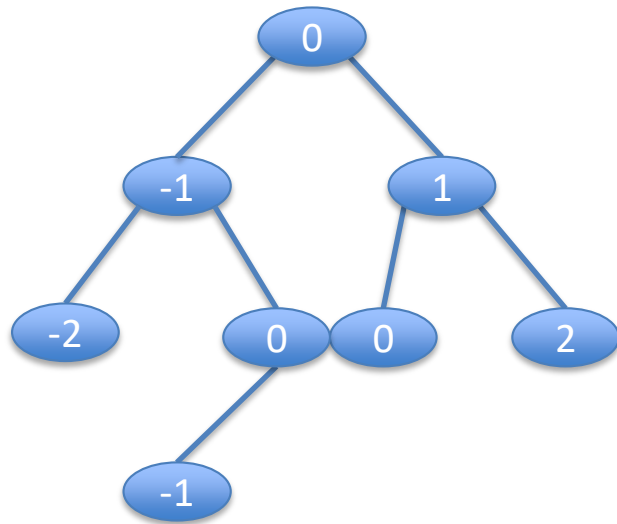
<http://www.lintcode.com/zh-cn/problem/binary-tree-vertical-order-traversal/>

<http://www.jiuzhang.com/solutions/binary-tree-vertical-order-traversal/>

Binary Tree Vertical Order Traversal

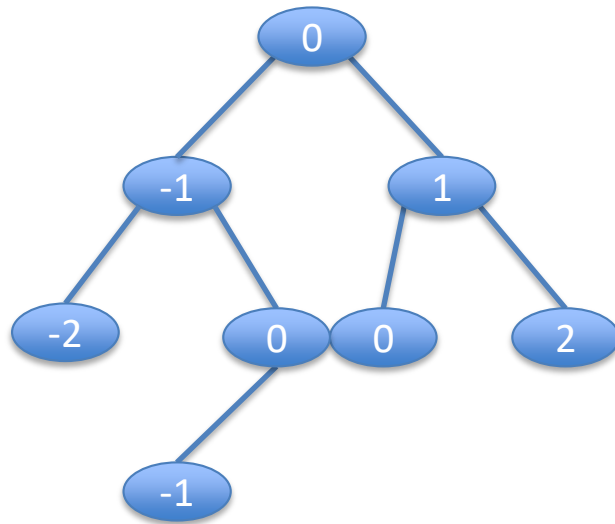
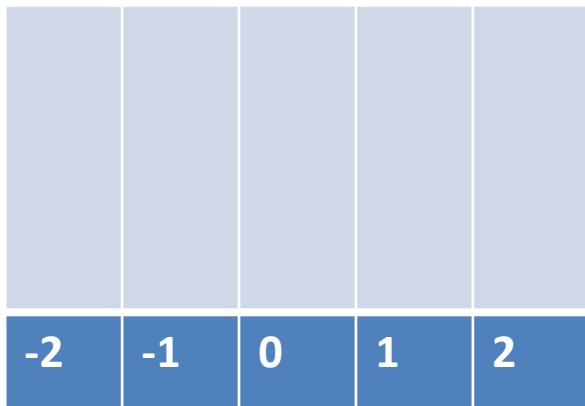
思路:

- 这一题有三个顺序:
 - 第一, 列数从小到大
 - 第二, 列数相同时行数从小到大
 - 第三, 列数行数都相同时, 从左到右



Binary Tree Vertical Order Traversal

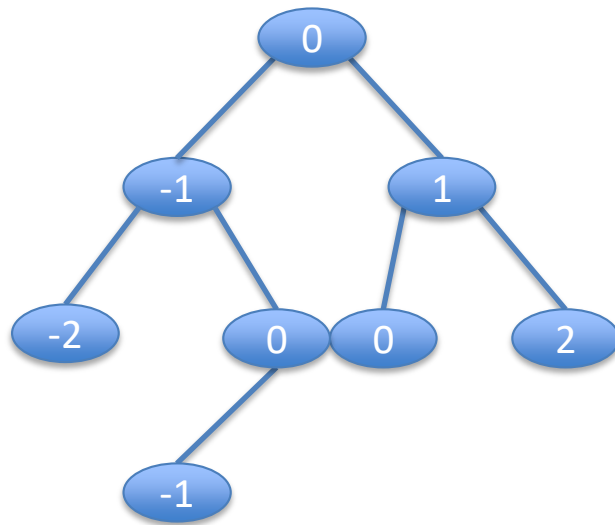
- 怎样计算列数？
 - Root为0 向左-1 向右+1
- 怎样保证第一个顺序，列数从小到大
 - Hash



Binary Tree Vertical Order Traversal

- 怎样保证第二个顺序，行数从小到大
 - BFS 一行一行的访问

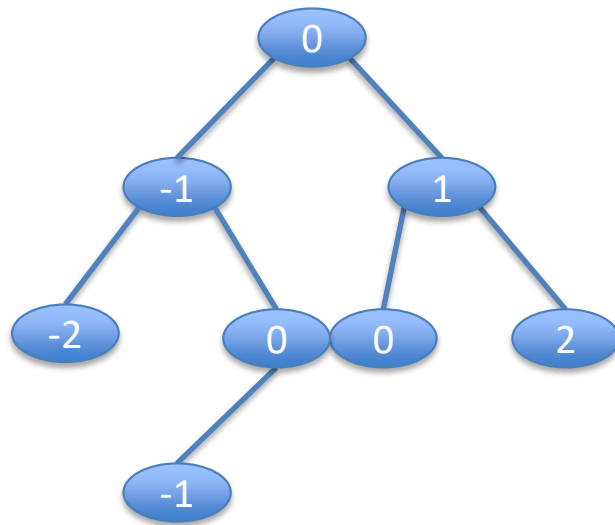
-2	-1	0	1	2



Binary Tree Vertical Order Traversal

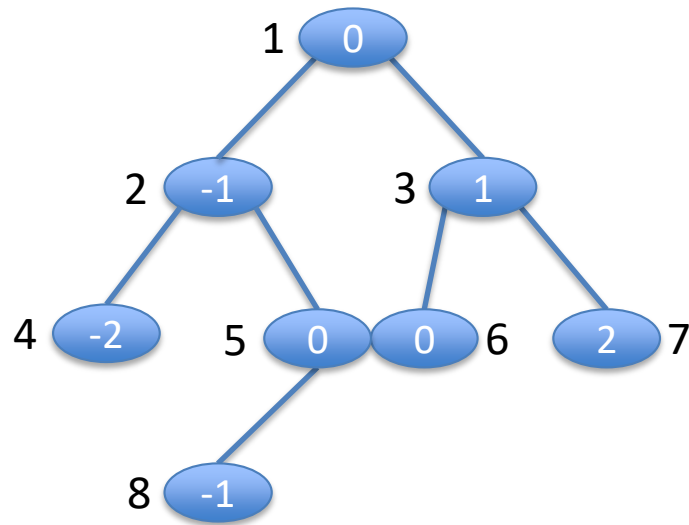
- 怎样保证第三个顺序
 - BFS已经保证了

-2	-1	0	1	2



Binary Tree Vertical Order Traversal

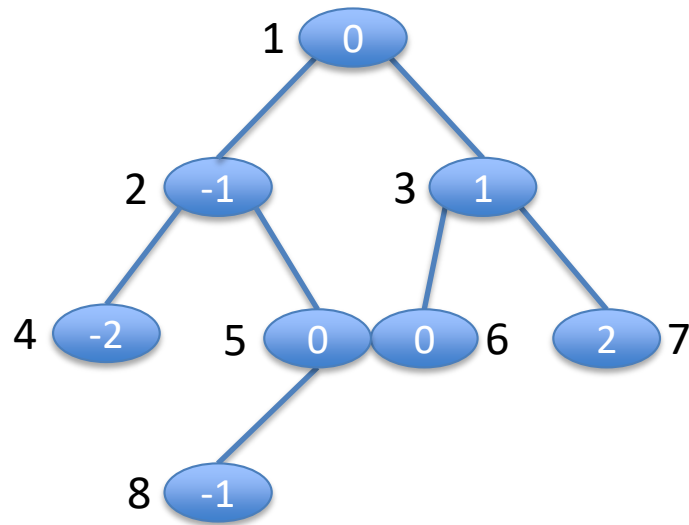
- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

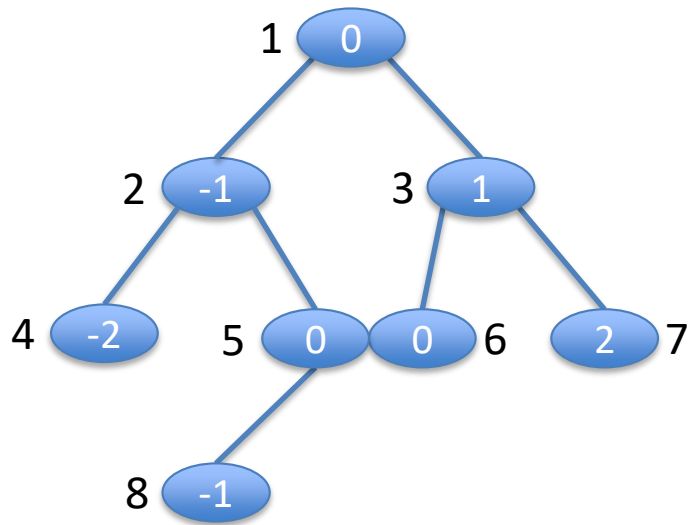
访问顺序：



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

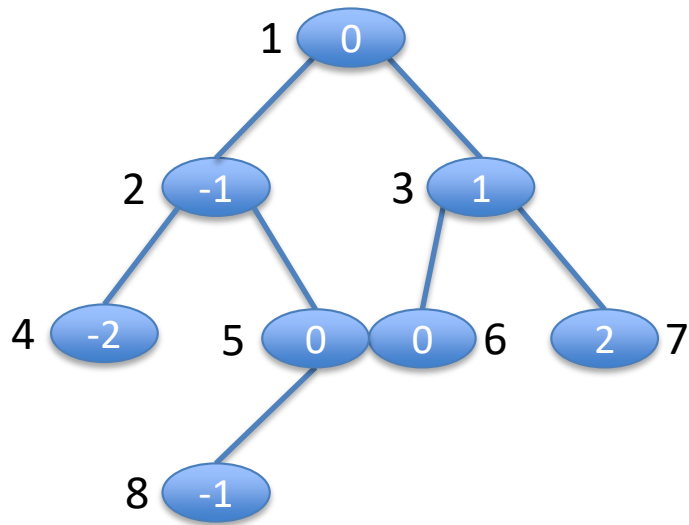
访问顺序：1



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

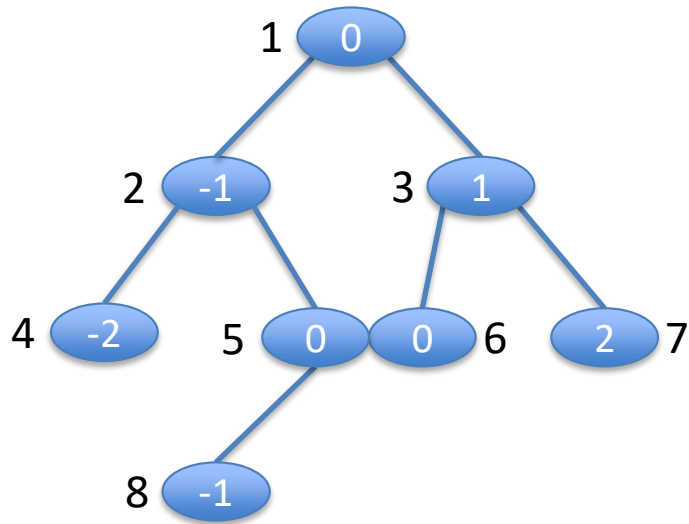
访问顺序：1



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

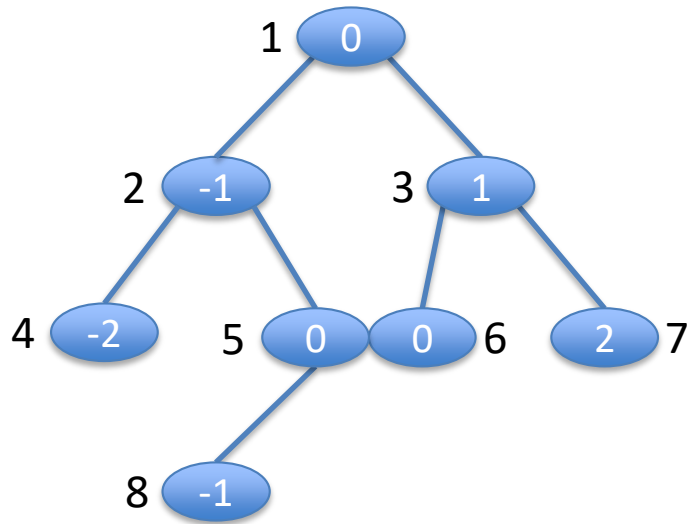
访问顺序：1 2



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

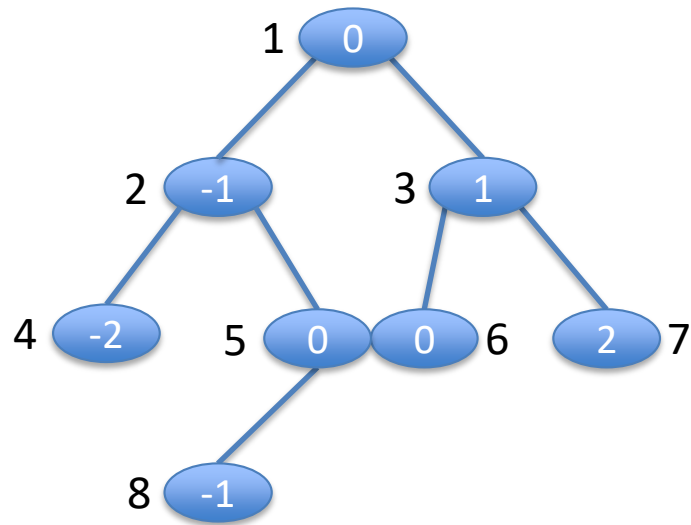
访问顺序：1 2



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

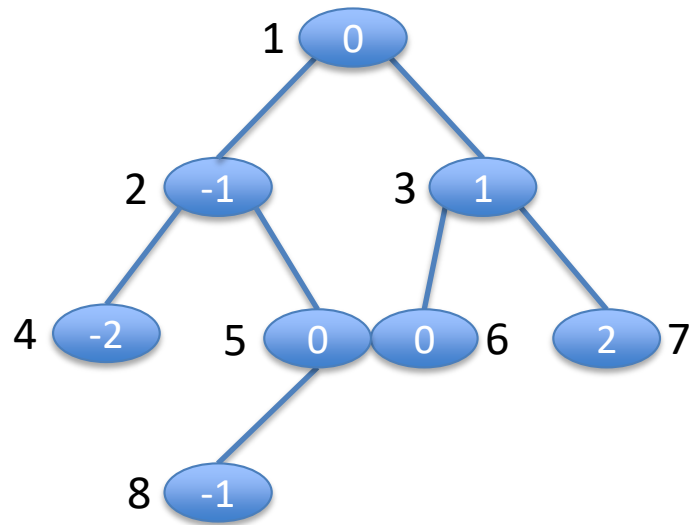
访问顺序：1 2 3



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

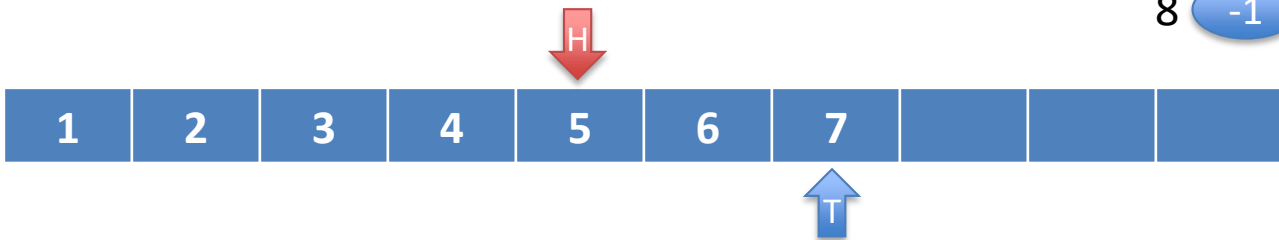
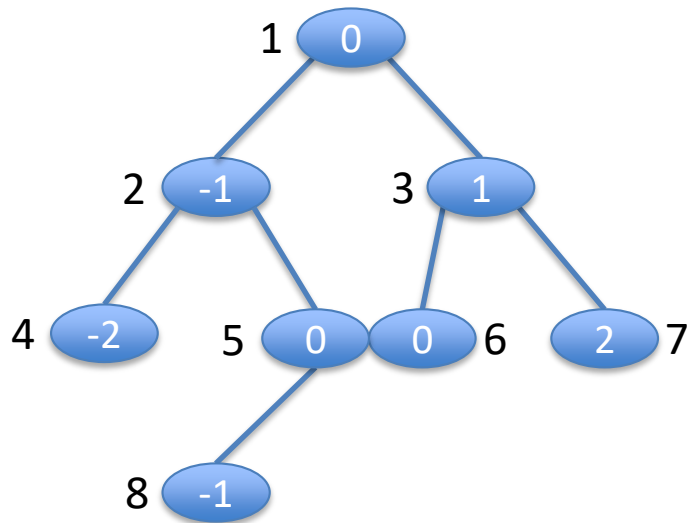
访问顺序：1 2 3



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

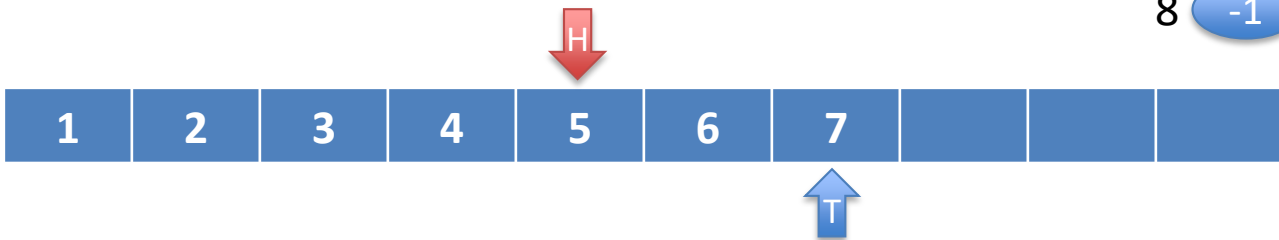
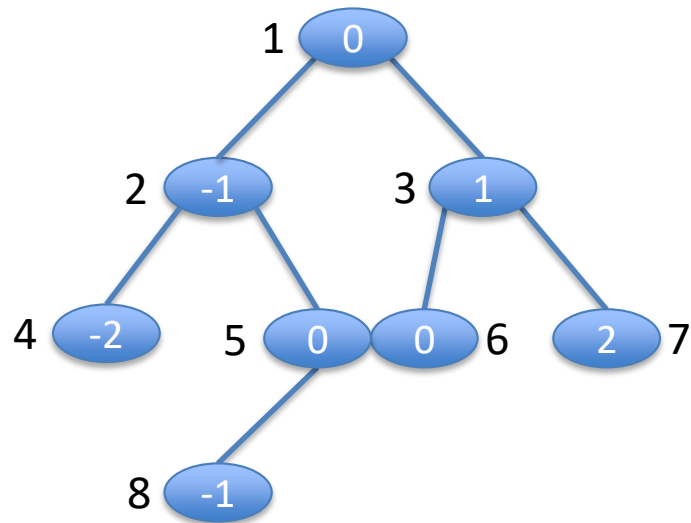
访问顺序：1 2 3 4



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

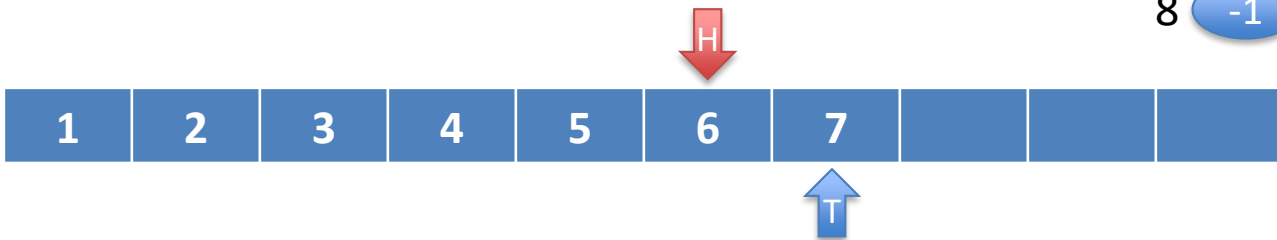
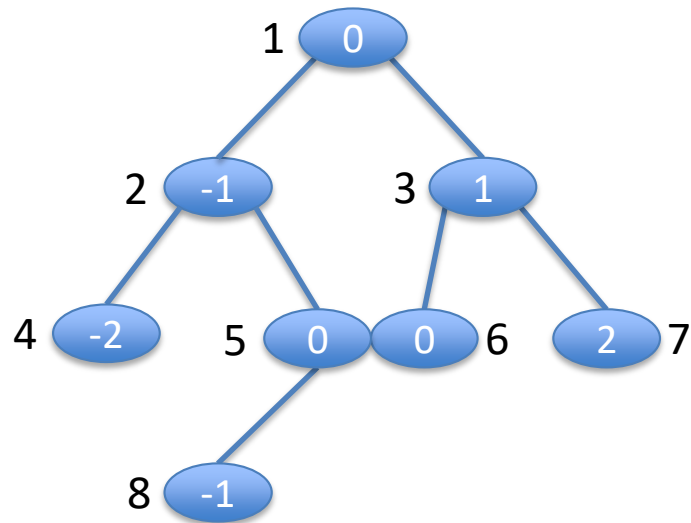
访问顺序：1 2 3 4



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

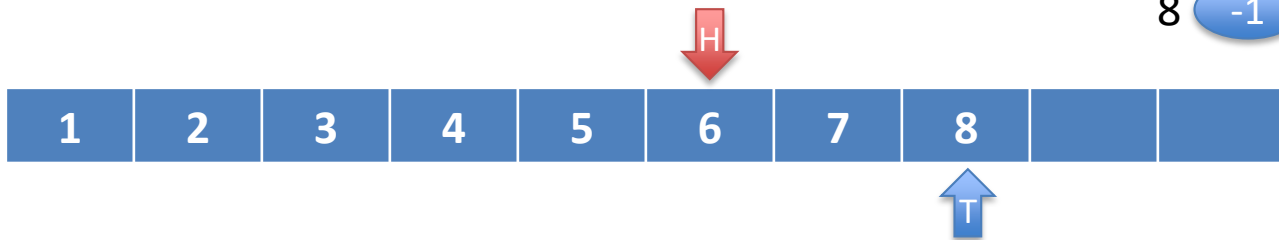
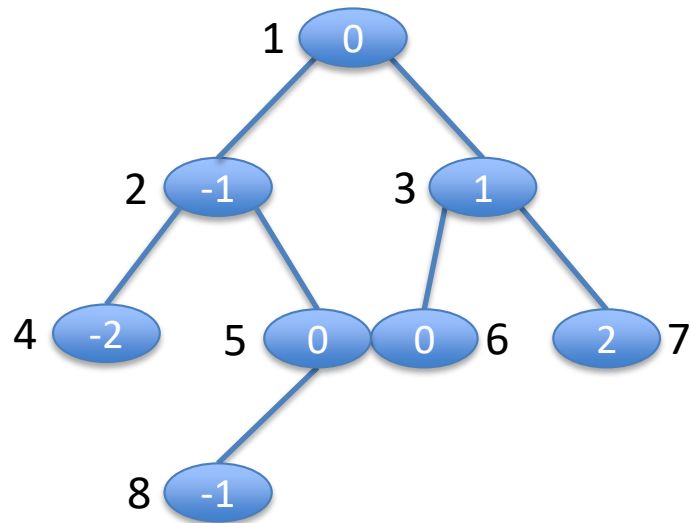
访问顺序：1 2 3 4 5



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

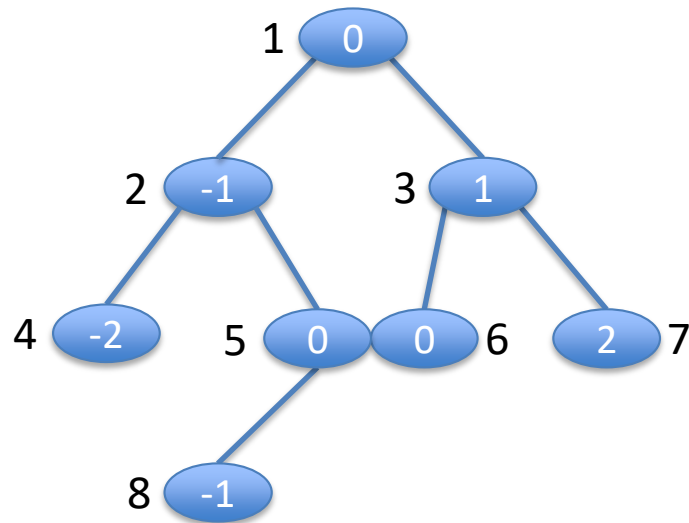
访问顺序：1 2 3 4 5



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

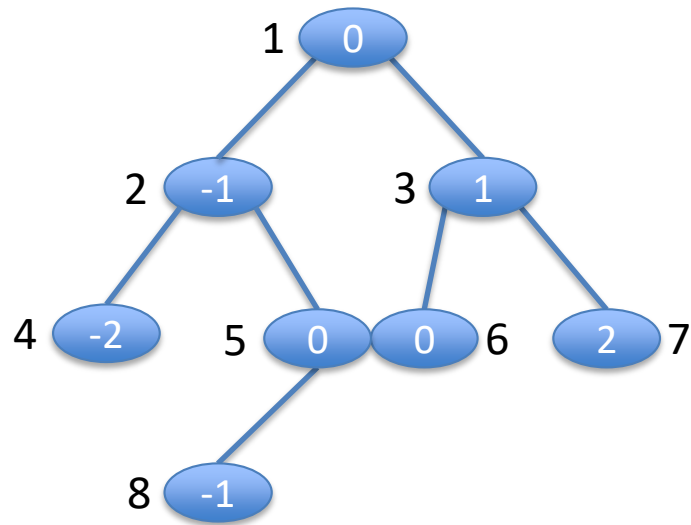
访问顺序：1 2 3 4 5 6



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

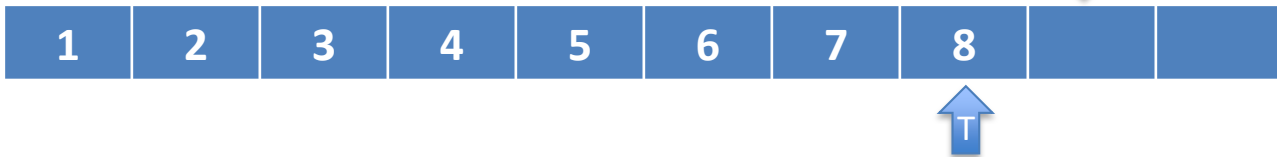
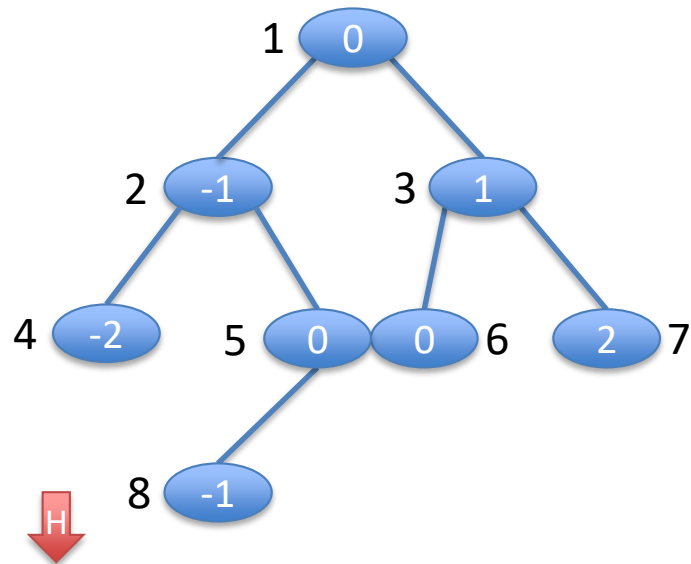
访问顺序：1 2 3 4 5 6 7



Binary Tree Vertical Order Traversal

- 怎样实现BFS?
 - 数据结构：队列
 - 实现步骤：
 1. 起始元素进队
 2. 队首元素出队（访问该当前元素）
 3. 当前元素扩展，相应元素进队
 4. 重复2,3两步，直到队列为空

访问顺序：1 2 3 4 5 6 7 8



时间复杂度：O(N)

- Company Tags: Google Facebook

考点:

- 基础数据结构上的灵活操作
- 二叉树的遍历

能力维度:

3. 基础数据结构/算法
4. 逻辑思维/算法优化能力

二叉树类题目：

Binary Tree Preorder Traversal	Binary Tree Inorder Traversal	Binary Tree Postorder Traversal
Maximum Depth of Binary Tree	Balanced Binary Tree	Lowest Common Ancestor
Binary Tree Maximum Path Sum II	Binary Tree Maximum Path Sum	Binary Tree Level Order Traversal

- Convert BST to Greater Tree

- ◆ 小技巧总结:

- DFS的两种理解方式:

1. 按照实际执行顺序模拟 (适合枚举型DFS, 下节课内容)
2. 按照DFS的定义宏观理解 (适合分治型DFS, 本节课内容)

- Inorder Successor in Binary Search Tree

- ◆ 小技巧总结:

- 遇到BST上操作的问题, 可以拿给定的节点(区间)与root做比较, 分类讨论、分而治之

- Binary Tree Flipping
- Binary Tree Leaves Order Traversal
- Binary Tree Vertical Order Traversal



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](#)

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

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

官网: www.jiuzhang.com