九章算法

九章算法基础班

第六讲　宽度优先遍历和BST

课程版本：v2.0　　张三疯 老师

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

版权声明

九章课程不提供视频，也严禁录制视频的侵权行为

否则将追求法律责任和经济赔偿

请不要缺课

本节重点

- 宽度优先遍历（Breadth-first traverse）
- 什么是二分搜索树（BST）
- BST的常用操作

课程回顾

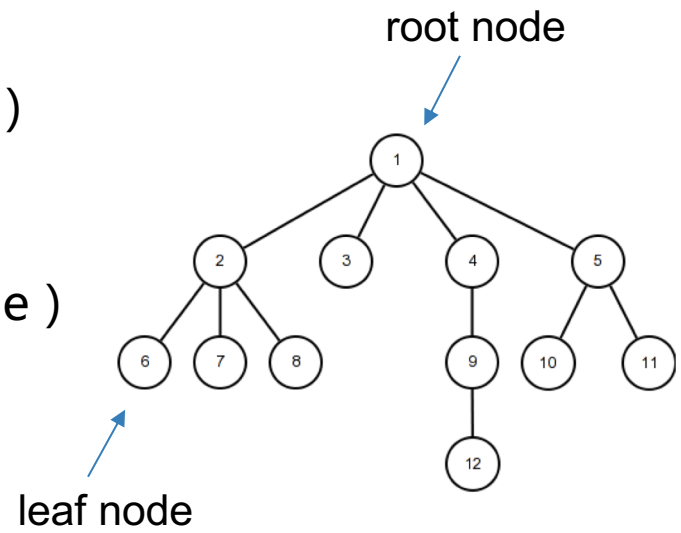
树（Tree）

九章算法

- 树的定义
  - 由节点（node）构成
  - 每个节点有零个或多个子节点（child node）

    没有子节点的是叶子节点（leaf node）
  - 每个非根节点只有一个父节点（parent node）

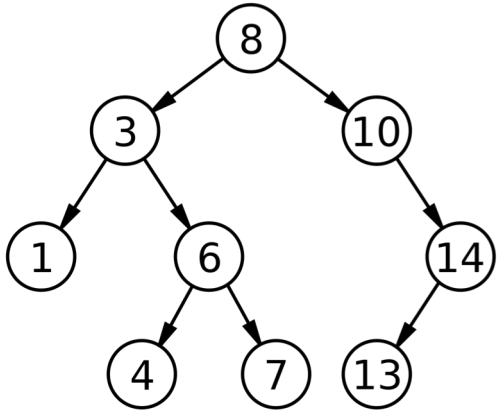
    没有父节点的是根节点（root node）



二叉树（Binary tree）

- 二叉树（Binary tree）
  - 每个节点最多有两个子节点
  - 两个子节点分别被称为左孩子（left child）和右孩子（right child）
  - 叶子节点：没有孩子的节点

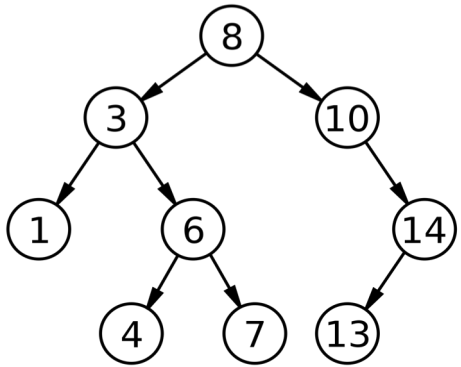
- 不特别说明的话，我们提到的树都是指二叉树



深度优先遍历

- 先序遍历（preorder）
  - root – left – right
  - 8 3 1 6 4 7 10 14 13
- 中序遍历（inorder）
- 后序遍历（postorder）

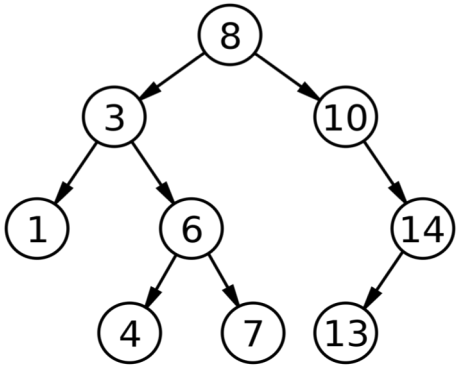
```
48  def preorder_traverse(root):
49      if root is None:
50          return
51
52      print(root.val, end=' ')
53      preorder_traverse(root.left)
54      preorder_traverse(root.right)
```



深度优先遍历

- 中序遍历（inorder）
  - left – root – right
  - 1 3 4 6 7 8 10 13 14

- 后序遍历（postorder）
  - left – right – root
  - 1 4 7 6 3 13 14 10 8



递归（Recursive）

- 什么是递归 (recursive)？
  - 数据结构的递归

    树就是一种递归的数据结构
  - 算法（程序）的递归

    函数自己调用自己

递归三要素

- 递归的定义
  - 首先这个问题或者数据结构需要是递归定义的
- 递归的出口
  - 什么时候递归终止
- 递归的拆解
  - 递归不终止的时候，如何分解问题

递归（Recursive）

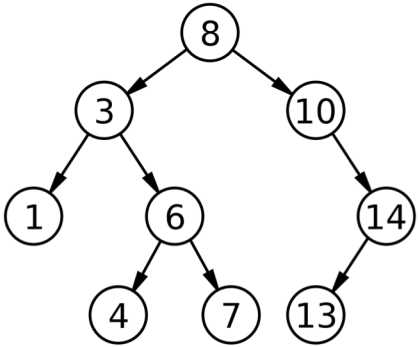
- 时间复杂度（Time complexity）
  - 程序执行的时间和输入问题规模之间的关系
  - 函数调用的次数 × 每次函数调用的时间消耗

- 空间复杂度（Space complexity）
  - 程序占用的空间和输入问题规模之间的关系
  - 堆（heap）空间 + 栈（stack）空间

宽度优先遍历

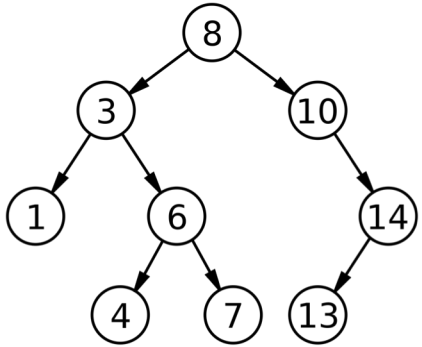
宽度优先遍历

- 二叉树的宽（广）度优先遍历
  - 访问每一个节点，不重不漏
  - 按层次的顺序遍历二叉树
  - 8 3 10 1 6 14 4 7 13



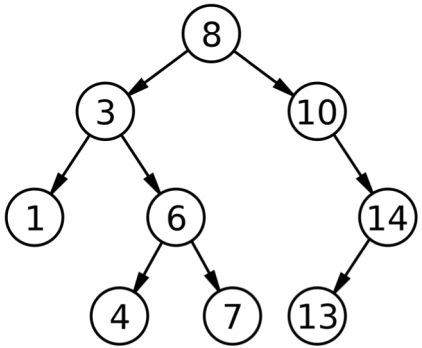
宽度优先遍历

- 二叉树的宽度优先遍历的实现
  - 使用队列（Queue）作为主要的数据结构
  - 代码演示



宽度优先遍历

- 二叉树的宽度优先遍历



8

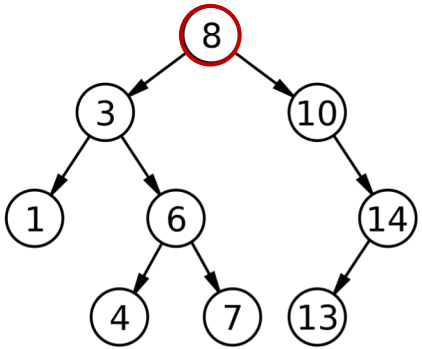
队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

九章算法

- 二叉树的宽度优先遍历



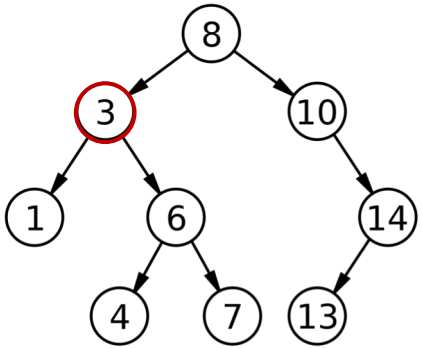
3	10

队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

- 二叉树的宽度优先遍历



10	1	6

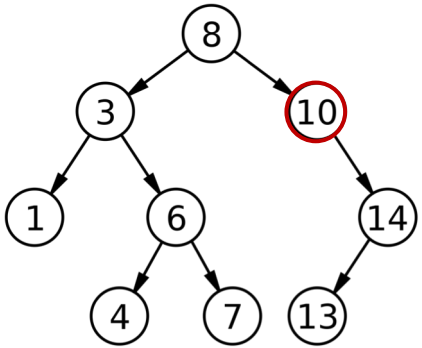
队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

九章算法

- 二叉树的宽度优先遍历



1	6	14

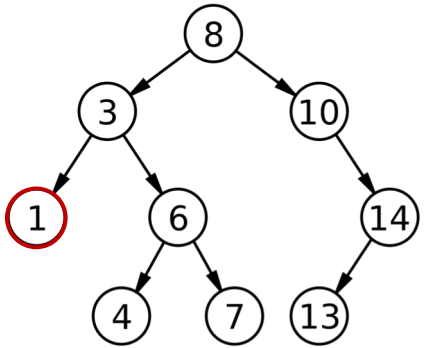
队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

九章算法

- 二叉树的宽度优先遍历



6	14

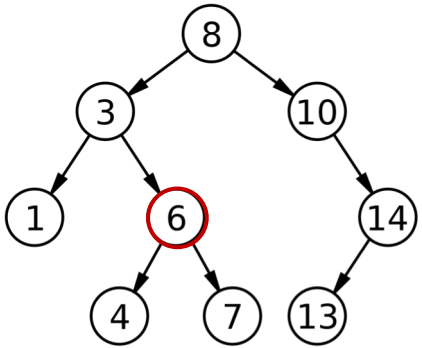
队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

九章算法

- 二叉树的宽度优先遍历



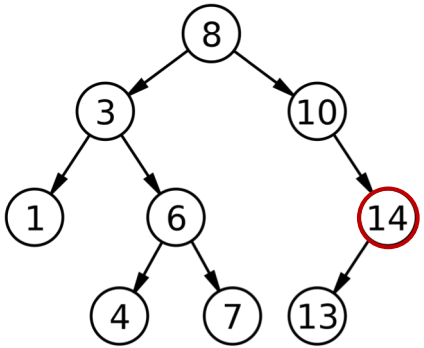
14 4 7

队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

- 二叉树的宽度优先遍历



4	7	13

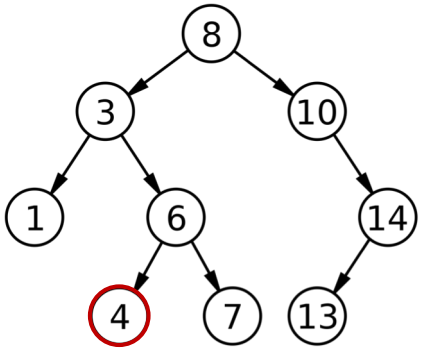
队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

九章算法

- 二叉树的宽度优先遍历



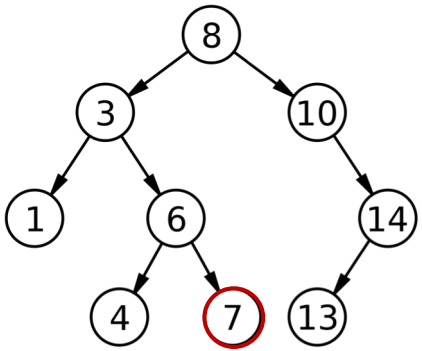
7	13

队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

- 二叉树的宽度优先遍历



13

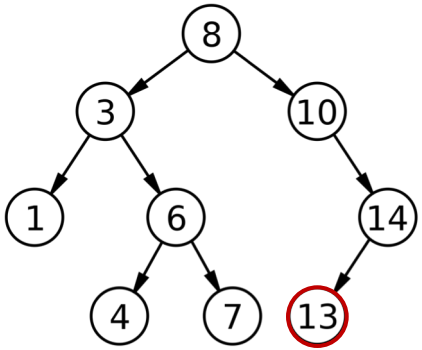
队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

九章算法

- 二叉树的宽度优先遍历



队列

```
40  def breadth_first_traverse(root):
41      if not root:
42          return
43
44      que = Queue(maxsize=0)
45      que.put(root)
46
47      while not que.empty():
48          cur = que.get()
49          print(cur.val, end=' ')
50          if cur.left:
51              que.put(cur.left)
52          if cur.right:
53              que.put(cur.right)
```

宽度优先遍历

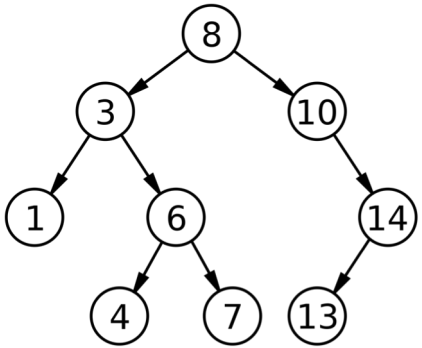
- 二叉树的宽度优先遍历
  - 分层遍历
  - 多一个循环
  - 代码演示

- 层数记录了根节点到当前节点的路径长度

宽度优先遍历

九章算法

- 二叉树的宽度优先遍历：分层遍历



8

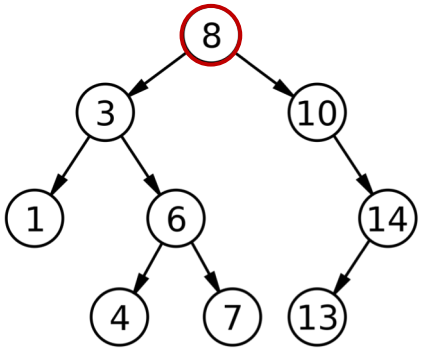
队列

```
56  def breadth_first_traverse_by_level(root):
57      if not root:
58          return
59
60      que = Queue(maxsize=0)
61      que.put(root)
62
63      while not que.empty():
64          n = que.qsize()
65          for i in range(n):
66              cur = que.get()
67              print(cur.val, end=' ')
68              if cur.left:
69                  que.put(cur.left)
70              if cur.right:
71                  que.put(cur.right)
72          print()
```

宽度优先遍历

九章算法

- 二叉树的宽度优先遍历：分层遍历



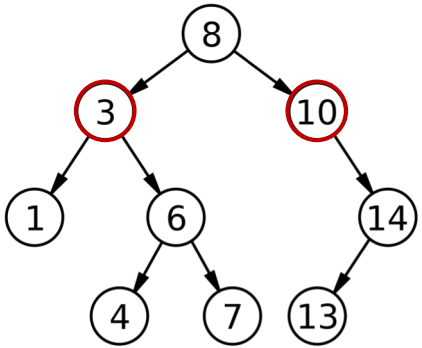
3	10

队列

```
56  def breadth_first_traverse_by_level(root):
57      if not root:
58          return
59
60      que = Queue(maxsize=0)
61      que.put(root)
62
63      while not que.empty():
64          n = que.qsize()
65          for i in range(n):
66              cur = que.get()
67              print(cur.val, end=' ')
68              if cur.left:
69                  que.put(cur.left)
70              if cur.right:
71                  que.put(cur.right)
72          print()
```

宽度优先遍历

- 二叉树的宽度优先遍历：分层遍历



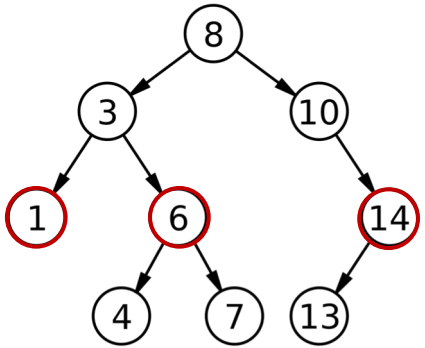
1	6	14

队列

```
56  def breadth_first_traverse_by_level(root):
57      if not root:
58          return
59
60      que = Queue(maxsize=0)
61      que.put(root)
62
63      while not que.empty():
64          n = que.qsize()
65          for i in range(n):
66              cur = que.get()
67              print(cur.val, end=' ')
68              if cur.left:
69                  que.put(cur.left)
70              if cur.right:
71                  que.put(cur.right)
72          print()
```

宽度优先遍历

- 二叉树的宽度优先遍历：分层遍历



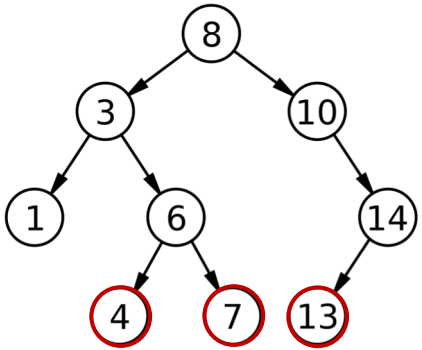
4	7	13

队列

```
56  def breadth_first_traverse_by_level(root):
57      if not root:
58          return
59
60      que = Queue(maxsize=0)
61      que.put(root)
62
63      while not que.empty():
64          n = que.qsize()
65          for i in range(n):
66              cur = que.get()
67              print(cur.val, end=' ')
68              if cur.left:
69                  que.put(cur.left)
70              if cur.right:
71                  que.put(cur.right)
72          print()
```

宽度优先遍历

- 二叉树的宽度优先遍历：分层遍历



队列

```
56  def breadth_first_traverse_by_level(root):
57      if not root:
58          return
59
60      que = Queue(maxsize=0)
61      que.put(root)
62
63      while not que.empty():
64          n = que.qsize()
65          for i in range(n):
66              cur = que.get()
67              print(cur.val, end=' ')
68              if cur.left:
69                  que.put(cur.left)
70              if cur.right:
71                  que.put(cur.right)
72          print()
```

宽度优先遍历

- 二叉树的宽度优先遍历
  - 时间复杂度：O(n)
  - 空间复杂度：由节点最多的层的节点数决定，O(n)

宽度优先遍历

- 练习一
  - Binary Tree Level Order Traversal
  - https://lintcode.com/en/problem/binary-tree-level-order-traversal/
  - https://www.jiuzhang.com/solution/binary-tree-level-order-traversal/

宽度优先遍历

- 练习二
  - Convert Binary Tree to Linked Lists by Depth
  - https://www.lintcode.com/en/problem/convert-binary-tree-to-linked-lists-by-depth/
  - https://www.jiuzhang.com/solution/convert-binary-tree-to-linked-lists-by-depth/

BST及其常用操作

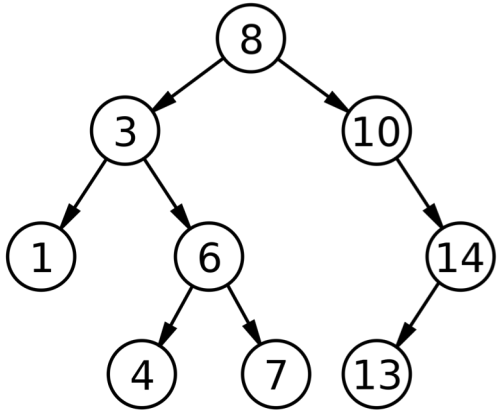
BST

- 什么是BST（Binary Search Tree）
  - 满足以下性质的Binary tree
  - 对于每个节点，他的左子树的所有节点都比它小
  - 对于每个节点，他的右子树的所有节点都比它大

- 以上是严格的说法，不允许BST有重复的节点，实际算法中可以允许重复

BST

- BST的特性有哪些应用？
  - 对BST进行中序遍历，得到的是一个非降序的序列
  - 插入操作（普通的Binary tree没有插入操作的概念）
  - 高效的查找



BST

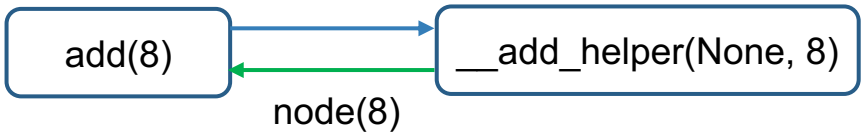
- BST的常用操作
  - 插入（insert）
  - 查找（find）
  - 删除（delete）

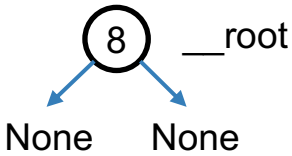
BST

- BST的插入操作

add(8)

```
20  def __add_helper(self, root, val):
21      if not root:
22          return TreeNode(val)
23      if val < root.val:
24          root.left = self.__add_helper(root.left, val)
25      else:
26          root.right = self.__add_helper(root.right, val)
27
28      return root
```



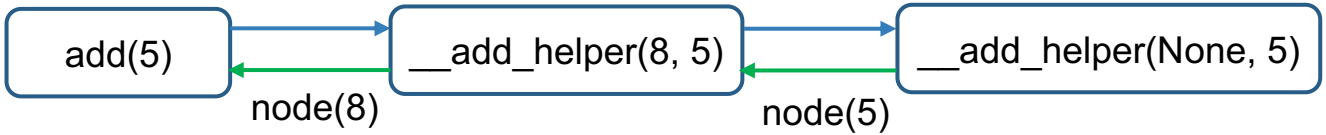


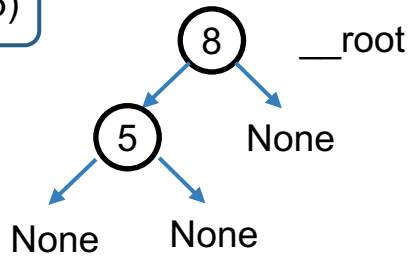
BST

- BST的插入操作

```
add(5)
```

```
20  def __add_helper(self, root, val):
21      if not root:
22          return TreeNode(val)
23      if val < root.val:
24          root.left = self.__add_helper(root.left, val)
25      else:
26          root.right = self.__add_helper(root.right, val)
27
28      return root
```



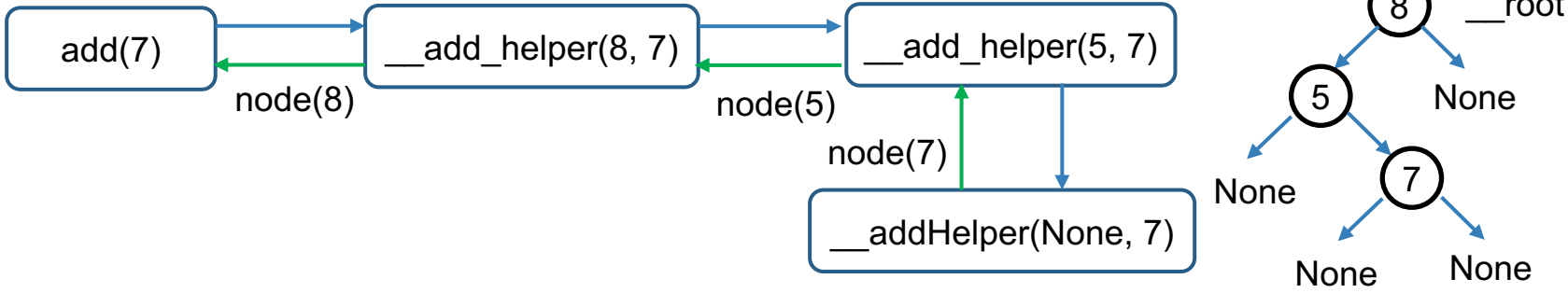


BST

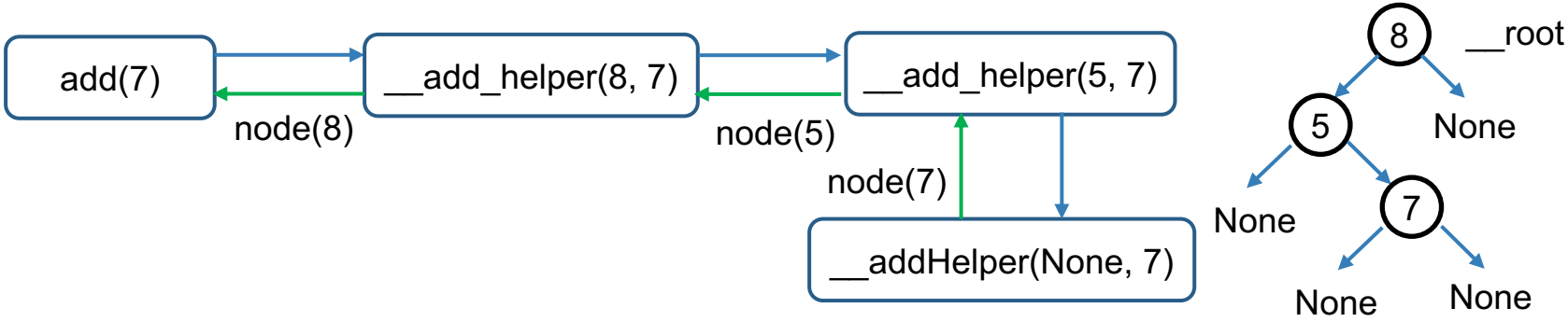
- BST的插入操作

add(7)

```
20  def __add_helper(self, root, val):
21      if not root:
22          return TreeNode(val)
23      if val < root.val:
24          root.left = self.__add_helper(root.left, val)
25      else:
26          root.right = self.__add_helper(root.right, val)
27
28      return root
```



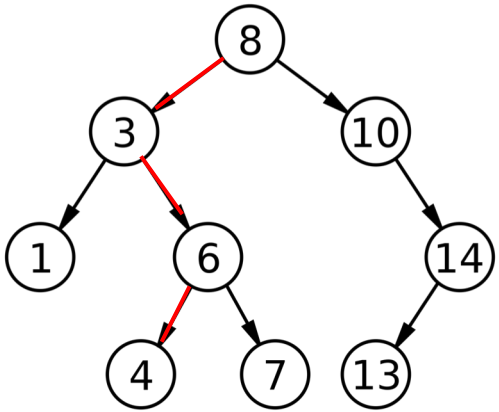
- BST的插入操作
  - 时间复杂度：O(h)
  - 空间复杂度：O(h)



BST

- BST的查找操作

```
contains(5)
```

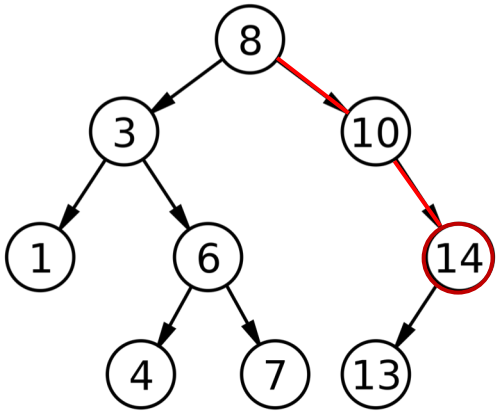



```python
33  def __contains_helper(self, root, val):
34      if not root:
35          return False
36
37      if root.val == val:
38          return True
39      elif val < root.val:
40          return self.__contains_helper(root.left, val)
41      else:
42          return self.__contains_helper(root.right, val)
```

BST

- BST的查找操作

```
contains(14)
```

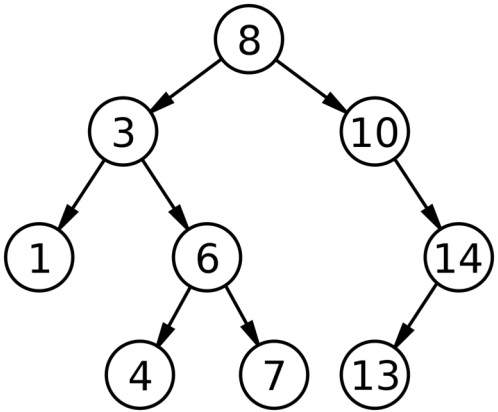



```python
def __contains_helper(self, root, val):
    if not root:
        return False

    if root.val == val:
        return True
    elif val < root.val:
        return self.__contains_helper(root.left, val)
    else:
        return self.__contains_helper(root.right, val)
```

BST

- BST的查找操作
  - 时间复杂度：O(h)
  - 空间复杂度：O(h)




```
33  def __contains_helper(self, root, val):
34      if not root:
35          return False
36
37      if root.val == val:
38          return True
39      elif val < root.val:
40          return self.__contains_helper(root.left, val)
41      else:
42          return self.__contains_helper(root.right, val)
```

BST

- 树的高度h与节点数n的关系
  - 最坏情况：n

    退化为线性
  - 最好情况：logn

    平衡的BST

平衡BST的应用

- 高效插入（insert）和查找（find）元素

- 在实际系统中广泛应用
  - 数据库
  - 搜索引擎

Bag

- 练习三
  - Insert Node in a Binary Search Tree
  - https://lintcode.com/en/problem/insert-node-in-a-binary-search-tree/
  - https://www.jiuzhang.com/solution/insert-node-in-a-binary-search-tree/

- 练习四
  - Validate Binary Search Tree
  - https://lintcode.com/en/problem/validate-binary-search-tree/
  - https://www.jiuzhang.com/solution/validate-binary-search-tree/

总结

- 二叉树的宽度优先遍历
  - 使用队列（Queue）作为主要的数据结构
  - 分层遍历：多一个循环

- 二分搜索树（BST）
  - 内部有序的二叉树
  - 可以高效插入和查找元素

Q&A

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
官网: www.jiuzhang.com

谢谢大家