

python实现二叉查找树

原创Joe-Han2017-05-01 11:28:4712400收藏12

分类专栏：算法Python文章标签：二叉树查找python算法

1. 定义

二叉查找树（Binary Search Tree），又称为二叉搜索树、二叉排序树。其或者是一棵空树；或者是具有以下性质的二叉树：

- 若左子树不空，则左子树上所有结点的值均小于或等于它的根结点的值
- 若右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值
- 左、右子树也分别为二叉排序树

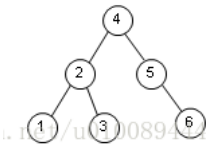


图 1

树中节点的定义如下：

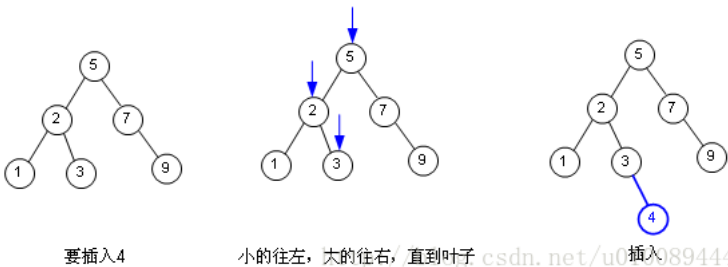
```
1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.lchild = None
5         self.rchild = None
```

2. 查找与插入

当二叉查找树不为空时：

- 首先将给定值与根结点的关键字比较，若相等，则查找成功
- 若小于根结点的关键字值，递归查左子树
- 若大于根结点的关键字值，递归查右子树
- 若子树为空，查找不成功

二叉排序树是一种动态树表。其特点是：树的结构通常不是一次生成的，而是在查找过程中，当树中不存在关键字等于给定值的结点时再进行插入。新插入的结点一定是一个新添加的叶子结点，并且是查找不成功时查找路径上访问的最后一个结点的左孩子或右孩子结点。如下图所示：



```

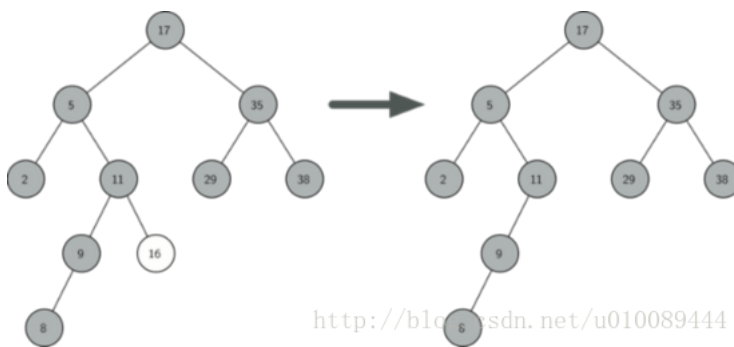
1  # 搜索
2  def search(self, node, parent, data):
3      if node is None:
4          return False, node, parent
5      if node.data == data:
6          return True, node, parent
7      if node.data > data:
8          return self.search(node.lchild, node, data)
9      else:
10         return self.search(node.rchild, node, data)
11
12 # 插入
13 def insert(self, data):
14     flag, n, p = self.search(self.root, self.root, data)
15     if not flag:
16         new_node = Node(data)
17         if data > p.data:
18             p.rchild = new_node
19         else:
20             p.lchild = new_node

```

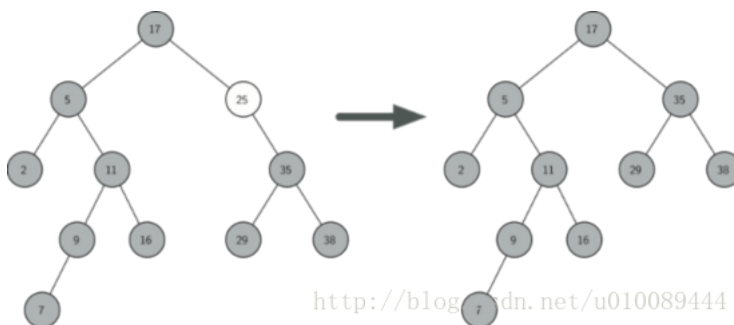
3. 删除

二叉查找树的删除操作分为三种情况：

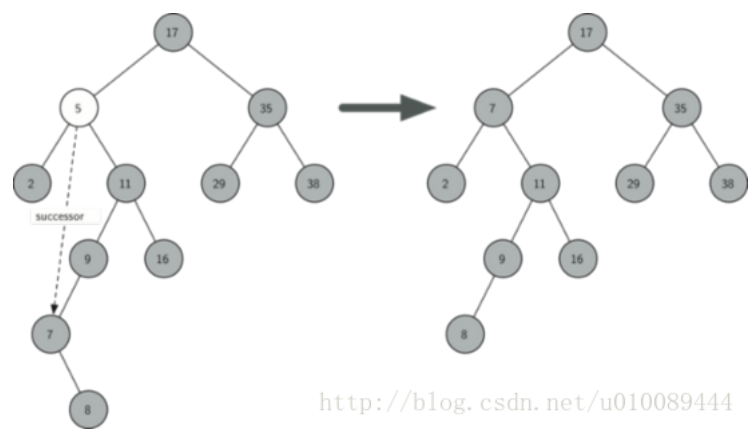
1. 如果待删除的节点是叶子节点，那么可以立即被删除，如下图所示：



2. 如果节点只有一个儿子，则将此节点parent的指针指向此节点的儿子，然后删除节点，如下图所示：



3. 如果节点有两个儿子，则将其右子树的最小数据代替此节点的数据，并将其右子树的最小数据删除，如下图所示：



<http://blog.csdn.net/u010089444>

删除节点的代码：

```
1  # 删除
2  def delete(self, root, data):
3      flag, n, p = self.search(root, root, data)
4      if flag is False:
5          print "无该关键字，删除失败"
6      else:
7          if n.lchild is None:
8              if n == p.lchild:
9                  p.lchild = n.rchild
10             else:
11                 p.rchild = n.rchild
12             del n
13         elif n.rchild is None:
14             if n == p.lchild:
15                 p.lchild = n.lchild
16             else:
17                 p.rchild = n.lchild
18             del n
19         else: # 左右子树均不为空
20             pre = n.rchild
21             if pre.lchild is None:
22                 n.data = pre.data
23                 n.rchild = pre.rchild
24                 del pre
25             else:
26                 next = pre.lchild
27                 while next.lchild is not None:
28                     pre = next
29                     next = next.lchild
30                 n.data = next.data
31                 pre.lchild = next.rchild
32                 del next
```

4. 遍历

1. 先序遍历

访问顺序如下：

- 访问根节点
- 先序遍历左子树
- 先序遍历右子树

以图 1 为例，访问顺序为：4， 2， 1， 3， 5， 6

```
1  # 先序遍历
2  def preOrderTraverse(self, node):
3      if node is not None:
```

```
4         print node.data,
5         self.preOrderTraverse(node.lchild)
6         self.preOrderTraverse(node.rchild)
```

2. 中序遍历

访问顺序如下：

- 中序遍历左子树
- 访问根节点
- 中序遍历右子树

以图 1 为例，访问顺序为：1, 2, 3, 5, 7, 9

```
1     # 中序遍历
2     def inOrderTraverse(self, node):
3         if node is not None:
4             self.inOrderTraverse(node.lchild)
5             print node.data,
6             self.inOrderTraverse(node.rchild)
```

3. 后序遍历

访问顺序如下：

- 后序遍历左子树
- 后序遍历右子树
- 访问根节点

以图 1 为例，访问顺序为：1, 3, 2, 9, 7, 5

```
1     # 后序遍历
2     def postOrderTraverse(self, node):
3         if node is not None:
4             self.postOrderTraverse(node.lchild)
5             self.postOrderTraverse(node.rchild)
6             print node.data,
```

5. 完整代码

```
1     # encoding: utf-8
2     class Node:
3         def __init__(self, data):
4             self.data = data
5             self.lchild = None
6             self.rchild = None
7
8     class BST:
9         def __init__(self, node_list):
10            self.root = Node(node_list[0])
11            for data in node_list[1:]:
12                self.insert(data)
13
14     # 搜索
15     def search(self, node, parent, data):
16         if node is None:
17             return False, node, parent
18         if node.data == data:
19             return True, node, parent
20         if node.data > data:
21             return self.search(node.lchild, node, data)
22         else:
23             return self.search(node.rchild, node, data)
24
25     # 插入
```

```
26 def insert(self, data):
27     flag, n, p = self.search(self.root, self.root, data)
28     if not flag:
29         new_node = Node(data)
30         if data > p.data:
31             p.rchild = new_node
32         else:
33             p.lchild = new_node
34
35 # 删除
36 def delete(self, root, data):
37     flag, n, p = self.search(root, root, data)
38     if flag is False:
39         print "无该关键字, 删除失败"
40     else:
41         if n.lchild is None:
42             if n == p.lchild:
43                 p.lchild = n.rchild
44             else:
45                 p.rchild = n.rchild
46             del p
47         elif n.rchild is None:
48             if n == p.lchild:
49                 p.lchild = n.lchild
50             else:
51                 p.rchild = n.lchild
52             del p
53         else: # 左右子树均不为空
54             pre = n.rchild
55             if pre.lchild is None:
56                 n.data = pre.data
57                 n.rchild = pre.rchild
58                 del pre
59             else:
60                 next = pre.lchild
61                 while next.lchild is not None:
62                     pre = next
63                     next = next.lchild
64                 n.data = next.data
65                 pre.lchild = next.rchild
66                 del p
67
68 # 先序遍历
69 def preOrderTraverse(self, node):
70     if node is not None:
71         print node.data,
72         self.preOrderTraverse(node.lchild)
73         self.preOrderTraverse(node.rchild)
74
75 # 中序遍历
76 def inOrderTraverse(self, node):
77     if node is not None:
78         self.inOrderTraverse(node.lchild)
79         print node.data,
80         self.inOrderTraverse(node.rchild)
81
82 # 后序遍历
83 def postOrderTraverse(self, node):
84     if node is not None:
85         self.postOrderTraverse(node.lchild)
86         self.postOrderTraverse(node.rchild)
87         print node.data,
88
89
90 a = [49, 38, 65, 97, 60, 76, 13, 27, 5, 1]
91 bst = BST(a) # 创建二叉查找树
92 bst.inOrderTraverse(bst.root) # 中序遍历
93
94 bst.delete(bst.root, 49)
95 print
96 bst.inOrderTraverse(bst.root)
```

精选python思维导图，源文件价值千元！09-30

自己购买的几十个思维导图文件，免费分享给大家。全部是可以修改、编辑和打印的思维导图。通过脑图，可以...

二叉搜索树--LeetCode(python)扣扣biubiubiu~ 101

验证二叉搜索树 给定一个二叉树，判断其是否是一个有效的二叉搜索树。假设一个二叉搜索树具有如下特征：节...

优质评论可以帮助作者获得更高权重

评论

Eudaimonia_: 为什么是del p而不是del n 6月前 回复 ...

1

码哥 *Xavier学长: 写的蛮好的 1年前 回复 ...

1

LeetCode.938——Python二叉搜索树深度遍历（DFS）迭代及递归（栈... zhr1030635594的博客 236

二叉搜索树 二叉排序树，每个节点的 left < node < right 递归实现深度优先搜索 版本一 对于每个节点进行递归，...

【二叉树】用python实现二叉搜索树guofei_fly的博客 193

二叉搜索树(BTS, binary search tree)是满足左子树节点小于根节点、右子树节点大于根节点的二叉树，是B-树的...

二叉查找树的Python实现_yeshankuangrenaaaaa的博客-CSDN博客9-24

由于二叉查找树是进入AVL平衡二叉树的学习的前提,因此先进行这个的实现。可以通过对节点的info属性进行赋...

python实现二叉查找树_zheng2485951710的博客-CSDN博客9-12

二叉树用python实现二叉树相关功能,有助于理解二叉树(这里实现的均为二叉查找树功能)概念二叉树是指计算机...

python实现vip视频解析爬取joe_niu的博客 3万+

https://blog.csdn.net/u013589137/article/details/80683905?utm_source=blogxgwz0 转载请注明作者和出处: http...

利用Python创建二叉树。acarsar的博客 2030

记录下用Python如何创建一个二叉树 1.定义一个节点类，为每一个节点都赋予左右孩子的属性，然后后续才能往...

使用python实现二叉搜索树_q1403539144的博客-CSDN博客9-27

具体的介绍可参照python官方collections模块docs 二叉搜索树主要有两个分支:一个分支用于存放小于当前节点的...

python3 实现二叉查找树的搜索、插入、删除_python_lia..._CSDN博客5-27

个人分类: 算法 Python 版权声明:本文为博主原创文章,转载需注明出处。 https://blog.csdn.net/u010089444/article...

二叉树和查找树的算法详解及Python3实现liangjiubujiu的博客 2569

树的介绍 1. 树的定义 树是一种数据结构，它是由n (n>=1) 个有限节点组成一个具有层次关系的集合。把它...

分段和分页复兴之路 2万+

一. 分页存储管理 1.基本思想 用户程序的地址空间被划分成若干固定大小的区域，称为“页”，相应地，内存空间...

python实现二叉搜索树 - qq_41386300的博客12-23

python实现二叉查找树 阅读数 9660 1. 定义二叉查找树(Binary Search Tree),又称为二叉搜索树、二叉排序树。其...

python实现二叉搜索树_your_answer的博客-CSDN博客9-25

#貌似纯二叉树没什么卵用,就不写示列了。 #二叉搜索树就不一样了,下面会有示列。 classBSTNode(Node): def_...

二叉搜索树python实现——python——系列1woshilsh的博客 468

这几天讲了二叉搜索树的相关算法，好像教材没给出完整的实现过程，都是缺胳膊少腿，学生自学的过程中老碰...

python二叉查找树CRUD工程师 2119

二叉查找树是二叉树的进化型，特点就是前序遍历得到的数组是递增关系，通俗点就是左孩子最小，父节点次之...

Python实现二叉搜索树-查找树-排序树【通俗易懂】_weix..._CSDN博客9-12

Python 实现二叉搜索树插入、查找、删除结点算法 二叉搜索树(Binary Search Tree),又称为“二叉排序树”、“二...

二叉搜索树python实现——python——系列3_woshilsh的博客-CSDN博客9-18

接下来实现修剪,修剪的时候使用后序遍历的思路,在树上进行遍历查找符合条件的留下,不符合的修剪掉,最后返回...

树--四种遍历一个人的学习和碎碎念 1万+

1、四种遍历概念 (1) 先序遍历：先访问根节点，再访问左子树，最后访问右子树。(2) 后序遍历：先...

©2020 CSDN 皮肤主题: 编程工作室 设计师:CSDN官方博客 返回首页


关于我们 招聘 广告服务 网站地图 kefu@csdn.net 客服论坛 400-660-0108 QQ客服 (8:30-22:00)

公安备案号 11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 版权与免责声明 版权申诉 网络110报警服务

中国互联网举报中心 家长监护 版权申诉 北京互联网违法和不良信息举报中心 ©1999-2020 北京创新乐知网络技术有限公司

点赞14 评论2 分享 收藏12 手机看 打赏 ... 关注

https://blog.csdn.net/u010089444/article/details/708545106/8



Joe-Han


码龄7年 暂无认证

50

7万+

1万+

66万+



原创

周排名

总排名

访问

等级

4390

323

761

93

599


积分

粉丝

获赞

评论

收藏




TA的主页


私信


关注


搜博文文章





热门文章

优化方法总结：SGD，Momentum，AdaGrad，RMSProp，Adam  86559


Numpy数组的保存与读取  66833


循环神经网络与LSTM  50968


生成式对抗网络（Generative Adversarial Networks，GANs）  38902


自动编码器（Autoencoder）  38825


分类专栏


 TensorFlow 11篇


 深度学习 8篇

 Python 7篇

 NLP 6篇

 写作 1篇

 eclipse 2篇



最新评论

卷积神经网络(CNN)

qq_39554370: 有引用文献就好了

循环神经网络与LSTM

L_Chao14: 想问一下，W[ht, xi].这里面隐状态和输入x是怎么组合在一起，直接拼接 ...

优化方法总结：SGD，Momentum，Ada...

weixin_43584808: GD可能过拟合

优化方法总结：SGD，Momentum，Ada...

wangqing1234567890: Adagrad公式有问题

判断有向图是否有环及环中元素


小葱抹抹酱: 楼主，请问如何输出含有公共边的环


最新文章


强化学习笔记(2)：Sarsa 与 Sarsa(lambda)


强化学习笔记(1)：Q-Learning


遗传算法 (Genetic Algorithm)


 点赞14


 评论2

 分享

 收藏12

 手机看

 打赏



关注

https://blog.csdn.net/u010089444/article/details/70854510

7/8

2016年 32篇

目录

- 1. 定义
- 2. 查找与插入
- 3. 删除
- 4. 遍历
 - 1. 先序遍历
 - 2. 中序遍历
 - 3. 后序遍历
- 5. 完整代码

