

LintCode领扣题解 (/problem) / 二叉搜索树中最接近的值 II · Closest Binary Search Tree Value II

二叉搜索树中最接近的值 II · Closest Binary Search Tree Value II

中文

谷歌 (/problem/?tags=google)

二叉查找树 (/problem/?tags=binary-search-tree)

栈 (/problem/?tags=stack)

描述

给定一棵非空二叉搜索树以及一个target值, 找到 BST 中最接近给定值的 k 个数。

❗ * 给出的target值为浮点数 * 你可以假设 `k` 总是合理的, 即 `k ≤ 总节点数` * 我们可以保证给出的 BST 中只有`唯一`一个最接近给定值的 k 个值的集合

样例

样例 1:

输入:
{1}
0.000000
1
输出:
[1]
解释:
二叉树 {1}, 表示如下的树结构:
1

样例 2:

输入:
{3,1,4,#,2}
0.275000
2
输出:
[1,2]
解释:
二叉树 {3,1,4,#,2}, 表示如下的树结构:
3
/ \
1 4
 \
2

挑战

假设是一棵平衡二叉搜索树, 你可以用时间复杂度低于 $O(n)$ 的算法解决问题吗(n 为节点个数)?

在线评测地址: <https://www.lintcode.com/problem/closest-binary-search-tree-value-ii/> (<https://www.lintcode.com/problem/closest-binary-search-tree-value-ii/>)

收起题目描述 ^

语言类型

ALL (48)

python (24)

java (16)

cpp (7)

javascript (1)

上传题解



令狐冲

更新于 7/27/2020, 3:27:35 AM

最优算法, 时间复杂度 $O(k + \log n)$, 空间复杂度 $O(\log n)$

实现如下的子函数:

1. `getStack()` => 在假装插入 `target` 的时候, 看看一路走过的节点都是哪些, 放到 `stack` 里, 用于 `iterate`
2. `moveUpper(stack)` => 根据 `stack`, 挪动到 `next node`
3. `moveLower(stack)` => 根据 `stack`, 挪动到 `prev node`

有了这些函数之后, 就可以把整个树当作一个数组一样来处理, 只不过每次 `i++` 的时候要用 `moveUpper`, `i--` 的时候要用 `moveLower`

java

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ /
Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public List<Integer> closestKValues(TreeNode root, double target, int k) {
        List<Integer> values = new ArrayList<>();

        if (k == 0 || root == null) {
            return values;
        }

        Stack<TreeNode> lowerStack = getStack(root, target);
        Stack<TreeNode> upperStack = new Stack<>();
        upperStack.addAll(lowerStack);
        if (target < lowerStack.peek().val) {
            moveLower(lowerStack);
        } else {
            moveUpper(upperStack);
        }

        for (int i = 0; i < k; i++) {
            if (lowerStack.isEmpty() ||
                !upperStack.isEmpty() && target - lowerStack.peek().val > upperStack.peek().val - target) {
                values.add(upperStack.peek().val);
                moveUpper(upperStack);
            } else {
                values.add(lowerStack.peek().val);
                moveLower(lowerStack);
            }
        }

        return values;
    }

    private Stack<TreeNode> getStack(TreeNode root, double target) {
        Stack<TreeNode> stack = new Stack<>();

        while (root != null) {
            stack.push(root);
        }
    }
}
```

```
        if (target < root.val) {
            root = root.left;
        } else {
            root = root.right;
        }
    }

    return stack;
}

public void moveUpper(Stack<TreeNode> stack) {
    TreeNode node = stack.peek();
    if (node.right == null) {
        node = stack.pop();
        while (!stack.isEmpty() && stack.peek().right == node) {
            node = stack.pop();
        }
        return;
    }

    node = node.right;
    while (node != null) {
        stack.push(node);
        node = node.left;
    }
}

public void moveLower(Stack<TreeNode> stack) {
    TreeNode node = stack.peek();
    if (node.left == null) {
        node = stack.pop();
        while (!stack.isEmpty() && stack.peek().left == node) {
            node = stack.pop();
        }
        return;
    }

    node = node.left;
    while (node != null) {
        stack.push(node);
        node = node.right;
    }
}
}
```

👍 获赞 29

💬 17 条评论

你的口袋题库

2000+ 算法真题、国内外名企题库免费开放



九章算法APP



令狐冲

更新于 6/23/2020, 10:49:35 AM

暴力方法。时间复杂度 $O(n)$, 空间复杂度也是 $O(n)$

- 先用 inorder traversal 求出中序遍历
- 找到第一个 $\geq \text{target}$ 的位置 index
- 从 index-1 和 index 出发, 设置两根指针一左一右, 获得最近的 k 个整数

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ /
Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm\_source=code
 */
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public List<Integer> closestKValues(TreeNode root, double target, int k) {
        List<Integer> values = new ArrayList<>();

        traverse(root, values);

        int i = 0, n = values.size();
        for (; i < n; i++) {
            if (values.get(i) >= target) {
                break;
            }
        }

        if (i >= n) {
            return values.subList(n - k, n);
        }

        int left = i - 1, right = i;
        List<Integer> result = new ArrayList<>();
        for (i = 0; i < k; i++) {
            if (left >= 0 && (right >= n || target - values.get(left) < values.get(right) - target)) {
                result.add(values.get(left));
                left--;
            } else {
                result.add(values.get(right));
                right++;
            }
        }

        return result;
    }

    private void traverse(TreeNode root, List<Integer> values) {
        if (root == null) {
            return;
        }

        traverse(root.left, values);
        values.add(root.val);
        traverse(root.right, values);
    }
}
```

👍 获赞 9

💬 6 条评论



令狐冲

更新于 9/5/2020, 1:44:42 AM

move_upper 和 move_lower 是镜像操作, left 和 right 互相换一下就行。相当于在 bst 里 get next node & get previous node

```

/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
"""
Definition of TreeNode:
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left, self.right = None, None
"""

class Solution:
    """
    @param root: the given BST
    @param target: the given target
    @param k: the given k
    @return: k values in the BST that are closest to the target
    """
    def closestKValues(self, root, target, k):
        if root is None or k == 0:
            return []

        lower_stack = self.get_stack(root, target)
        upper_stack = list(lower_stack)
        if lower_stack[-1].val < target:
            self.move_upper(upper_stack)
        else:
            self.move_lower(lower_stack)

        result = []
        for i in range(k):
            if self.is_lower_closer(lower_stack, upper_stack, target):
                result.append(lower_stack[-1].val)
                self.move_lower(lower_stack)
            else:
                result.append(upper_stack[-1].val)
                self.move_upper(upper_stack)

        return result

    def get_stack(self, root, target):
        stack = []
        while root:
            stack.append(root)
            if target < root.val:
                root = root.left
            else:
                root = root.right

        return stack

    def move_upper(self, stack):
        if stack[-1].right:
            node = stack[-1].right

```

```

        while node:
            stack.append(node)
            node = node.left
        else:
            node = stack.pop()
            while stack and stack[-1].right == node:
                node = stack.pop()

def move_lower(self, stack):
    if stack[-1].left:
        node = stack[-1].left
        while node:
            stack.append(node)
            node = node.right
    else:
        node = stack.pop()
        while stack and stack[-1].left == node:
            node = stack.pop()

def is_lower_closer(self, lower_stack, upper_stack, target):
    if not lower_stack:
        return False

    if not upper_stack:
        return True

    return target - lower_stack[-1].val < upper_stack[-1].val - target

```

👍 获赞 4

💬 5 条评论



李助教

更新于 7/11/2020, 4:34:01 AM

```

/**
 * 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括：系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
 * - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
 */
"""
Definition of TreeNode:
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left, self.right = None, None
"""

class Solution:
    """
    @param root: the given BST
    @param target: the given target
    @param k: the given k
    @return: k values in the BST that are closest to the target
    """
    def closestKValues(self, root, target, k):
        # write your code here
        stack_upper = []
        stack_lower = []
        cur = root
        while cur:
            stack_upper.append(cur)

```

```

        cur = cur.left
    cur = root
    while cur:
        stack_lower.append(cur)
        cur = cur.right

    while len(stack_upper) > 0 and stack_upper[-1].val < target:
        self.move_upper(stack_upper)
    while len(stack_lower) > 0 and stack_lower[-1].val >= target:
        self.move_lower(stack_lower)

    ans = []
    for i in range(k):
        if len(stack_lower) == 0:
            upper = stack_upper[-1].val
            ans.append(upper)
            self.move_upper(stack_upper)
        elif len(stack_upper) == 0:
            lower = stack_lower[-1].val
            ans.append(lower)
            self.move_lower(stack_lower)
        else:
            upper, lower = stack_upper[-1].val, stack_lower[-1].val
            if upper - target < target - lower:
                ans.append(upper)
                self.move_upper(stack_upper)
            else:
                ans.append(lower)
                self.move_lower(stack_lower)

    return ans

def move_upper(self, stack):
    cur = stack.pop()
    if cur.right:
        cur = cur.right
    while cur:
        stack.append(cur)
        cur = cur.left

def move_lower(self, stack):
    cur = stack.pop()
    if cur.left:
        cur = cur.left
    while cur:
        stack.append(cur)
        cur = cur.right

```

👍 获赞 3

💬 1 条评论



令狐冲

更新于 7/15/2020, 9:59:04 AM

使用类似于 Find K Closest Elements 的做法。先求得 inorder, 然后再二分到一个接近的位置, 然后 Two pointers 向两边走。时间复杂度 $O(n)$

```

/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课

```

```
* - 更多详情请见官方网站: http://www.jiuzhang.com/?utm\_source=code
*/
"""
Definition of TreeNode:
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left, self.right = None, None
"""

class Solution:
    """
    @param root: the given BST
    @param target: the given target
    @param k: the given k
    @return: k values in the BST that are closest to the target
    """
    def closestKValues(self, root, target, k):
        if root is None or k == 0:
            return []

        nums = self.get_inorder(root)
        left = self.find_lower_index(nums, target)
        right = left + 1
        results = []
        for _ in range(k):
            if self.is_left_closer(nums, left, right, target):
                results.append(nums[left])
                left += 1
            else:
                results.append(nums[right])
                right += 1
        return results

    def get_inorder(self, root):
        dummy = TreeNode(0)
        dummy.right = root
        stack = [dummy]
        inorder = []

        while stack:
            node = stack.pop()
            if node.right:
                node = node.right
                while node:
                    stack.append(node)
                    node = node.left
            if stack:
                inorder.append(stack[-1].val)

        return inorder

    def find_lower_index(self, nums, target):
        """
        find the largest number < target, return the index
        """
        start, end = 0, len(nums) - 1
        while start + 1 < end:
            mid = (start + end) // 2
            if nums[mid] < target:
                start = mid
            else:
                end = mid

        if nums[end] < target:
            return end

        if nums[start] < target:
```



```

        return start

    return -1

def is_left_closer(self, nums, left, right, target):
    if left < 0:
        return False
    if right >= len(nums):
        return True
    return target - nums[left] < nums[right] - target

```

👍 获赞 1

💬 1 条评论



李助教

更新于 6/9/2020, 7:03:54 AM

```

/**
 * 本参考程序由九章算法用户提供。版权所有，转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作，授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括：九章算法班 2020升级版，算法强化班，算法基础班，北美算法面试高频题班，Java 高级工程师 P6+ 小班课，面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括：系统设计 System Design，面向对象设计 OOD
 * - 专题及项目类课程包括：动态规划专题班，Big Data - Spark 项目实战，Django 开发项目课
 * - 更多详情请见官方网站：http://www.jiuzhang.com/?utm_source=code
 */
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

public class Solution {
    /**
     * @param root: the given BST
     * @param target: the given target
     * @param k: the given k
     * @return: k values in the BST that are closest to the target
     */
    private Stack<TreeNode> upperStack = new Stack<>();
    private Stack<TreeNode> lowerStack = new Stack<>();
    public List<Integer> closestKValues(TreeNode root, double target, int k) {
        // write your code here

        List<Integer> ans = new ArrayList<>();

        TreeNode cur;
        cur = root;
        while(cur != null){
            upperStack.push(cur);
            cur = cur.left;
        }
        cur = root;
        while(cur != null){
            lowerStack.push(cur);
            cur = cur.right;
        }

        while(!upperStack.isEmpty() && getUpperPeakVal() < target) moveUpper();

```

```

while(!lowerStack.isEmpty() && getLowerPeakVal() >= target) moveLower();

for(int i = 0; i < k; i++){
    if(!upperStack.isEmpty() && !lowerStack.isEmpty()){
        if(getUpperPeakVal() - target < target - getLowerPeakVal()){
            ans.add(getUpperPeakVal());
            moveUpper();
        }else{
            ans.add(getLowerPeakVal());
            moveLower();
        }
    }else if(!upperStack.isEmpty()){
        ans.add(getUpperPeakVal());
        moveUpper();
    }else{
        ans.add(getLowerPeakVal());
        moveLower();
    }
}

return ans;
}

int getUpperPeakVal(){
    return upperStack.peek().val;
}

int getLowerPeakVal(){
    return lowerStack.peek().val;
}

void moveUpper(){
    TreeNode cur = upperStack.pop();
    cur = cur.right;
    while(cur != null){
        upperStack.push(cur);
        cur = cur.left;
    }
}

void moveLower(){
    TreeNode cur = lowerStack.pop();
    cur = cur.left;
    while(cur != null){
        lowerStack.push(cur);
        cur = cur.right;
    }
}
}

```

👍 获赞 1

💬 1 条评论



九章算法助教团队

更新于 6/9/2020, 7:04:12 AM

最优算法, 时间复杂度 $O(k + \log n)$ $O(k + \log n)$, 空间复杂度 $O(\log n)$ $O(\log n)$

实现如下的子函数:



getStack() => 在假装插入 target 的时候, 看看一路走过的节点都是哪些, 放到 stack 里, 用于 iterate moveUpper(stack) => 根据 stack, 挪动到 next node
 moveLower(stack) => 根据 stack, 挪动到 prev node 有了这些函数之后, 就可以把整个树当作一个数组一样来处理, 只不过每次 i++ 的时候要用 moveUpper, i-- 的时候要用 moveLower

/**

* 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */

```
class Solution {
public:
    void helper(TreeNode* root, vector<int>& inorder) {
        if(root == NULL)
            return;
        helper(root->left, inorder);
        inorder.push_back(root->val);
        helper(root->right, inorder);
    }
    vector<int> closestKValues(TreeNode * root, double target, int k) {
        if(k == 1)
            return {root->val};
        vector<int> inorder;
        helper(root, inorder);
        vector<int> res;
        int start = 0;
        int end = inorder.size() - 1;
        while(start + 1 < end)
        {
            int mid = start + (end - start) / 2;
            if(inorder[mid] == target)
            {
                res.push_back(inorder[mid]);
                k--;
                break;
            }
            else if(inorder[mid] < target)
                start = mid;
            else
                end = mid;
        }
        end = start + 1;
        while(k --)
        {
            if(start >= 0 && end < inorder.size())
            {
                if(abs(inorder[start] - target) < abs(inorder[end] - target))
                {
                    res.push_back(inorder[start]);
                    start--;
                }
                else
                {
                    res.push_back(inorder[end]);
                    end++;
                }
            }
            else if(start >= 0)
            {
                res.push_back(inorder[start]);
                start--;
            }
            else
            {
                res.push_back(inorder[end]);
                end++;
            }
        }
        return res;
    }
}
```

```
};
```

 获赞 0 4 条评论**zxqiu**

更新于 8/5/2020, 3:51:19 AM

使用令狐老师的基本思路重写，让代码更易读。思路等同于从指定节点开始分别向前和向后遍历，直到找到k个最接近target的节点。使用prev和next两个栈分别记录前驱和后继，goPrev相当于反向中序遍历，goNext相当于正向中序遍历。

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ /
Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm\_source=code
 */
public class Solution {
    public List<Integer> closestKValues(TreeNode root, double target, int k) {
        Stack<TreeNode> next = new Stack<TreeNode>();
        Stack<TreeNode> prev = new Stack<TreeNode>();
        TreeNode node = root;

        // find the nodes closest to target
        while (node != null) {
            if (node.val < target) {
                prev.push(node);
                node = node.right;
            } else {
                next.push(node);
                node = node.left;
            }
        }

        List<Integer> ret = new LinkedList<Integer>();

        while (ret.size() < k) {
            double distp = prev.isEmpty() ? Integer.MAX_VALUE : Math.abs(prev.peek().val - target);
            double distn = next.isEmpty() ? Integer.MAX_VALUE : Math.abs(next.peek().val - target);

            // compare and find the closest node, and move the corresponding stack.
            if (distp < distn) {
                ret.add(0, prev.peek().val);
                goPrev(prev);
            } else {
                ret.add(next.peek().val);
                goNext(next);
            }
        }

        return ret;
    }

    private void goNext(Stack<TreeNode> st) {
        TreeNode r = st.pop().right;

        while (r != null) {
            st.push(r);
            r = r.left;
        }
    }

    private void goPrev(Stack<TreeNode> st) {
        TreeNode l = st.pop().left;

        while (l != null) {
            st.push(l);
            l = l.right;
        }
    }
}
```

👍 获赞 54

💬 9 条评论

**zxqiu**

更新于 12/19/2020, 11:44:15 AM

中序遍历暴力解法简单易懂。队没满遇到一个node就塞进去;满了就把距离远的删了,距离近的塞进去。

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
public class Solution {
    public List<Integer> closestKValues(TreeNode root, double target, int k) {
        List<Integer> ret = new LinkedList<Integer>();

        dfs(root, target, k, ret);

        return ret;
    }

    private void dfs(TreeNode root, double target, int k, List<Integer> ret) {
        if (root == null) {
            return;
        }

        dfs(root.left, target, k, ret);

        if (ret.size() < k) {
            ret.add(root.val);
        } else if (Math.abs(root.val - target) < Math.abs(ret.get(0) - target)) {
            ret.remove(0);
            ret.add(root.val);
        }

        dfs(root.right, target, k, ret);
    }
}
```

👍 获赞 13

💬 5 条评论

**明尼苏达大角羊**

更新于 8/5/2020, 11:19:26 PM

对令狐老师和其他同学的解法小小总结一下, $O(h + k)$ 的时间复杂度。 h 为树的高度, 平均为 $\log n$ 。

【确认条件】 (1) 沟通BST的定义。 (2) 确认是否需要判断tree和k是否valid。 (3) 确认不会存在两个与target距离相等的值, 否则输出list的时候还得判断哪一个放在前面。 (4) 确认k是否小于等于tree中的节点数 (虽然解法中遇到这种情况会通过break跳出)。

【解题思路】 (1) 通过get_stacks()虚拟寻找target的插入位置, 并将一路上经过的点根据值的大小分别放入prev_stack和next_stack。用两个栈的好处是: 之后在实现get_next()和get_prev()的时候会相对简单一些, 不需要像完整版BST iterator那么复杂。

(2) 实现get_next(), 利用next_stack寻找next_value。在一般的BST iterator中, 寻找下一节点的算法是: 如果当前点存在右子树, 那么就是右子树中一直向左走到底的那个点; 如果当前点不存在右子树, 则对到达当前点的路径进行反向遍历 (一直pop stack), 寻找第一个 (离当前点最近的) 左拐的点。然而在本题中, 因为已经分离prev_stack和next_stack, 所以在当前节点不存在右子树的情况下, 当前节点在next_stack中的前一个位置自然就是要找的下一个点。因此代码中只需处理当前节点存在右子树时的情况, 即先取当前节点的右子树, 再一路向左走到底。

(3) 实现get_prev(), 利用prev_stack寻找prev_value。对get_next()的处理方式取反, 即先取当前节点的左子树, 再一路向右走到底。若不存在左子树, 在pop出当前节点后, stack-1 ()自然处于下一个prev节点的位置。

(4) for循环k次, 每次比较prev_stack和next_stack栈顶节点的值, 把与target距离近的那个放进results中。

【实现要点】 (1) 实现get_stacks()的时候, 在把节点分入两个栈的时候注意思考一下, 别把大小写, 左右子树弄反了。另外对于本题, 不需要对root.val == target的情况专门处理。 (2) 实现get_next()和get_prev()注意细节 (完整版BST iterator其实需要背诵, 本题中再对其简化)。 (3) 比较大小的时候引入sys.maxsize作为异常情况处理。

(/) 课程 (/course/) 旗舰课 (/premium-course/) 1对1私教 (/1on1/) 免费课 (/seminar/) 刷题题解 (/shatitong/) 成功案 (更多... (/accounts/profile/)) 我的课程 (/accounts/)

【复杂度】 时间复杂度: $O(h + k)$, $O(h)$ 来自于对树的搜索, $O(k)$ 是获取k个结果。空间复杂度: $O(h)$

```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
class Solution:
    def closestKValues(self, root, target, k):
        results = []
        if root is None or k == 0:
            return results
        next_stack, prev_stack = self.get_stacks(root, target)

        for _ in range(k):
            if len(next_stack) == 0 and len(prev_stack) == 0:
                break
            next_diff = sys.maxsize if len(next_stack) == 0 else abs(next_stack[-1].val - target)
            prev_diff = sys.maxsize if len(prev_stack) == 0 else abs(prev_stack[-1].val - target)

            if next_diff < prev_diff:
                results.append(self.get_next(next_stack))
            else:
                results.append(self.get_prev(prev_stack))

        return results

    def get_stacks(self, root, target):
        next_stack, prev_stack = [], []
        while root:
            if root.val < target:
                prev_stack.append(root)
                root = root.right
            else:
                next_stack.append(root)
                root = root.left

        return next_stack, prev_stack

    def get_next(self, next_stack):
        value = next_stack[-1].val
        node = next_stack.pop().right
        while node:
            next_stack.append(node)
            node = node.left

        return value

    def get_prev(self, prev_stack):
        value = prev_stack[-1].val
        node = prev_stack.pop().left
        while node:
            prev_stack.append(node)
            node = node.right

        return value
```

👍 获赞 11 😊 添加评论

**Cccus**

更新于 11/3/2020, 4:27:48 AM

建议大家还是掌握这个时间复杂度 $O(k + \lg n)$ 的解法 要不然面试的评级肯定不会高，即使做出来 $O(n)$ 又有什么用呢 毕竟题目中出现 k 就是希望你能用它来减少时间复杂度的

方法如下： 首先建立两个数组 `prev` 和 `next` 用来储存比 `target` 小的 `node` 和比它大的 `node` 再用 `while k`: 去遍历 k 个数，提前取出他们的值，比较大小，再调用方法 `getNext` 或 `getPrev` 即可


```
/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ /
Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm\_source=code
 */
class Solution:
    """
    @param root: the given BST
    @param target: the given target
    @param k: the given k
    @return: k values in the BST that are closest to the target
    """
    def closestKValues(self, root, target, k):
        # write your code here
        prev = []
        next = []

        while root:
            if root.val > target:
                next.append(root)
                root = root.left
            elif root.val < target:
                prev.append(root)
                root = root.right
            else:
                next.append(root)
                break

        res = []
        while k:
            next_val = sys.maxsize if len(next) == 0 else abs(next[-1].val - target)
            prev_val = sys.maxsize if len(prev) == 0 else abs(prev[-1].val - target)

            if next_val == sys.maxsize and prev_val == sys.maxsize:
                break
            if next_val < prev_val:
                res.append(next[-1].val)
                self.getNext(next)
                k -= 1
            else:
                res.append(prev[-1].val)
                self.getPrev(prev)
                k -= 1

        return res

    def getNext(self, next):
        node = next.pop()
        node = node.right
        while node:
            next.append(node)
            node = node.left

    def getPrev(self, prev):
        node = prev.pop()
        node = node.left
        while node:
            prev.append(node)
```

👍 获赞 9

💬 7 条评论



九章用户PJGZHC

更新于 6/9/2020, 7:03:47 AM

要流氓法, 先来一个inorder 再把find k 个接近点的方法照抄一遍。事实证明, 题还是需要背的

```

/**
 * 本参考程序由九章算法用户提供。版权所有, 转发请注明出处。
 * - 九章算法致力于帮助更多中国人找到好的工作, 授课老师均来自硅谷和国内的一线大公司在职工程师。
 * - 现有的求职课程包括: 九章算法班 2020升级版, 算法强化班, 算法基础班, 北美算法面试高频题班, Java 高级工程师 P6+ 小班课, 面试软技能指导 - BQ / Resume / Project 2020版
 * - Design类课程包括: 系统设计 System Design, 面向对象设计 OOD
 * - 专题及项目类课程包括: 动态规划专题班, Big Data - Spark 项目实战, Django 开发项目课
 * - 更多详情请见官方网站: http://www.jiuzhang.com/?utm_source=code
 */
"""
Definition of TreeNode:
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left, self.right = None, None
"""

class Solution:
    """
    @param root: the given BST
    @param target: the given target
    @param k: the given k
    @return: k values in the BST that are closest to the target
    """
    def closestKValues(self, root, target, k):
        # write your code here
        if root is None or k == 0:
            return []
        self.list = []
        self.inOrder(root)
        res = self.findKth(self.list, target, k)

        return res

    def findKth(self, list, target, k):
        start, end = 0, len(list) - 1

        while start + 1 < end:
            mid = (start + end) / 2
            if list[mid] < target:
                start = mid
            else:
                end = mid
        out = []
        while len(out) < k:
            leftDiff = abs(target - list[start]) if start >= 0 else None
            rightDiff = abs(target - list[end]) if end < len(list) else None


            if rightDiff != None and leftDiff != None:
                if rightDiff < leftDiff:
                    out.append(list[end])
                    end += 1
                else:
                    out.append(list[start])
                    start -= 1
            elif leftDiff != None:
                out.append(list[start])
                start -= 1
            elif rightDiff != None:

```

```
        out.append(list[end])
        end += 1

    return out

def inOrder(self, root):
    if root is None:
        return
    self.inOrder(root.left)
    self.list.append(root.val)
    self.inOrder(root.right)
```

 获赞 4 3 条评论[加载更多题解](#)

进阶课程

视频+互动

直播+互动

直播+互动

互动课

九章算法班 2021 版

8周时间精通 57 个核心高频考点, 9 招击破 FLAG、BATJ 算法面试。22....

系统架构设计 System Design 2021 版

成为百万架构师必上。30 课时带你快速掌握 18 大系统架构设计知识点与面...

九章算法面试高频题冲刺班

每期更新 15% 题目, 考前押题, 一举拿下 FLAG & BATJ Offer



面向对象设计 OOD

应届生及亚马逊面试必考, IT 求职必备基础

[首页 \(/?skip_redirect=true\)](#) | [联系我们 \(mailto:info@jiuzhang.com\)](mailto:info@jiuzhang.com) | [加入我们 \(/joinus\)](#)

Copyright © 2013-2020 九章算法 浙ICP备19045946号-1
(<http://www.miibeian.gov.cn/>)

商务合作: [fukesu@jiuzhang.com \(mailto:fukesu@jiuzhang.com\)](mailto:fukesu@jiuzhang.com)

 (<http://weibo.com/ninechapter>)  知
(<https://www.zhihu.com/people/crackinterview/>)

(/)