【H4】AVL 树作业

代码部分：

```python
#uuid_share#  f7840f32-1d41-4517-9a19-bebf09b1f517  #
# DSA'21课程上机作业
# 【H4】AVL树作业
#
# 说明：为方便批改作业，请同学们在完成作业时注意并遵守下面规则：
# （1）直接在本文件中指定部位编写代码
# （2）如果作业中对相关类有明确命名/参数/返回值要求的，请严格按照要求执行
# （3）有些习题会对代码的编写进行特殊限制，请注意这些限制并遵守
# （4）作业代码部分在5月26日18:00之前提交到Canvas系统

# ---- 用AVL树实现字典类型 ----
# 用AVL树来实现字典类型，使得其put/get/in/del操作均达到对数性能
# 采用如下的类定义，至少实现下列的方法
# key至少支持整数、浮点数、字符串
# 请调用hash(key)来作为AVL树的节点key
# 【注意】涉及到输出的__str__, keys, values这些方法的输出次序是AVL树中序遍历次序
#    也就是按照hash(key)来排序的，这个跟Python 3.7中的dict输出次序不一样。

# 请在此编写你的代码
class TreeNode():
    """
    二叉树节点
    请自行完成节点内部的实现，并实现给出的接口
    """
    def __init__(self, key, value=None):
        self.preKey = key
        self.key = hash(key)
        self.value = value
        self.left = None
        self.right = None
        self.height = 0

    def rightRotation(self):
        if self.left == None:
            return
        else:
```

```
            ret = self.left
        cache = ret.right
        self.left = cache
        self.heightUpdate()
        ret.right = self
        ret.heightUpdate()
        return ret

    def leftRotation(self):
        if self.right == None:
            return
        else:
            ret = self.right
        cache = ret.left
        self.right = cache
        self.heightUpdate()
        ret.left = self
        ret.heightUpdate()
        return ret

    def heightUpdate(self):
        leftHight = -1 if self.left == None else self.left.height
        rightHight = -1 if self.right == None else self.right.height
        self.height = 1 + max(leftHight, rightHight)

    def leftRightRotation(self):
        if self.left != None:
            self.left = self.left.leftRotation()
        return self.rightRotation()

    def rightLeftRotation(self):
        if self.right != None:
            self.right = self.right.rightRotation()
        return self.leftRotation()

```

```
72      def isLeaf(self):
73          if self.left == None and self.right == None:
74              return True
75          else:
76              return False
77
78      def onlyHaveLeft(self):
79          if self.left != None and self.right == None:
80              return True
81          else:
82              return False
83
84      def onlyHaveRight(self):
85          if self.left == None and self.right != None:
86              return True
87          else:
88              return False
89
90      def findRightMost(self):
91          root = self
92          while True:
93              right = root.right
94              if right == None:
95                  break
96              root = right
97          return (root.preKey, root.key, root.value)
98
99      def findLeftMost(self):
100         root = self
101         while True:
102             left = root.left
103             if left == None:
104                 break
105             root = left
106         return (root.preKey, root.key, root.value)
```

```python
class AVL:
    def __init__(self, node: TreeNode = None):
        self.root = node

    def insert(self, root, node):
        def getHeight(root):
            if root == None:
                return -1
            else:
                return root.height
        if root == None:
            return node
        if node.key < root.key:
            root.left = self.insert(root.left, node)
            root.left.heightUpdate()
            if getHeight(root.left) - getHeight(root.right) == 2:
                left = root.left
                if getHeight(left.right) > getHeight(left.left):
                    root = root.leftRightRotation()
                else:
                    root = root.rightRotation()
        else:
            root.right = self.insert(root.right, node)
            root.right.heightUpdate()
            if getHeight(root.left) - getHeight(root.right) == -2:
                right = root.right
                if getHeight(right.left) > getHeight(right.right):
                    root = root.rightLeftRotation()
                else:
                    root = root.leftRotation()
        root.heightUpdate()
        return root

    def find(self, root, key):
        if root == None:
```

```
    def find(self, root, key):
        if root == None:
            return None
        if root.key == key:
            return root
        elif root.key > key:
            return self.find(root.left, key)
        elif root.key < key:
            return self.find(root.right, key)

    def remove(self, root, key):
        def getHeight(root):
            if root == None:
                return -1
            else:
                return root.height
        left = root.left
        right = root.right
        if root.key == key:
            if root.isLeaf():
                return None
            elif root.onlyHaveLeft():
                root = left
            elif root.onlyHaveRight():
                root = right
            else:
                cachePreKey, cacheKey, cacheValue = right.findLeftMost()
                root.preKey = cachePreKey
                root.key = cacheKey
                root.value = cacheValue
                root.right = self.remove(right, cacheKey)
        elif root.key < key:
            root.right = self.remove(right, key)
        else:
            root.left = self.remove(left, key)
```

```
            else:
                root.left = self.remove(left, key)
            root.heightUpdate()
            left = root.left
            right = root.right
            if getHeight(right) - getHeight(left) == 2:
                if getHeight(right.left) > getHeight(right.right):
                    root = root.rightLeftRotation()
                else:
                    root = root.leftRotation()
            elif getHeight(right) - getHeight(left) == -2:
                if getHeight(left.right) > getHeight(left.left):
                    root = root.leftRightRotation()
                else:
                    root = root.rightRotation()
            root.heightUpdate()
            return root


class mydict(AVL):
    """
    以AVL树作为内部实现的字典
    """
    def getRoot(self):  # 返回内部的AVL树根
        return self.root

    def __init__(self, root = None):  # 创建一个空字典
        AVL.__init__(self, root)

    def __setitem__(self, key, value):  # 将key:value保存到字典
        node = TreeNode(key, value)
        self.root = self.insert(self.root, node)

    def __getitem__(self, key):  # 从字典中根据key获取value
        # v = md[key]
```

```
207
208     def __getitem__(self, key):  # 从字典中根据key获取value
209         # v = md[key]
210         # key在字典中不存在的话，请raise KeyError
211         key = hash(key)
212         res = self.find(self.root, key)
213         if res == None:
214             raise KeyError
215         else:
216             return res.value
217
218     def __delitem__(self, key):  # 删除字典中的key
219         # del md[key]
220         # key在字典中不存在的话，请raise KeyError
221         key = hash(key)
222         res = self.find(self.root, key)
223         if res == None:
224             raise KeyError
225         else:
226             self.root = self.remove(self.root, key)
227
228     def __len__(self):
229         return self.keys().__len__()
230
231     def __contains__(self, key):
232         key = hash(key)
233         res = self.find(self.root, key)
234         if res == None:
235             return False
236         else:
237             return True
238
239     def clear(self):  # 清除字典
240         self.root = None
241
242     def __str__(self):
```

```
    def __str__(self):
        keys = self.keys()
        values = self.values()
        return dict(zip(keys, values)).__str__()
    __repr__ = __str__

    def keys(self):  # 返回所有的key，类型是列表，按照AVL树中序遍历次序
        def in_order(tree, res_ = []):
            if tree != None:
                res = in_order(tree.left, res)
                res.append(tree.preKey)
                res = in_order(tree.right, res)
            return res
        return in_order(self.root)

    def values(self):  # 返回所有的value，类型是列表，按照AVL树中序遍历次序
        def in_order(tree, res_ = []):
            if tree != None:
                res = in_order(tree.left, res)
                res.append(tree.value)
                res = in_order(tree.right, res)
            return res
        return in_order(self.root)
# 代码结束
# 检验
print("========== AVL树实现字典 ==========")
md = mydict()
md['hello'] = 'world'
md['name'] = 'sessdsa'
print(md)  # {'name': 'sessdsa', 'hello': 'world'}

for f in range(1000):
    md[f**0.5] = f

for i in range(1000, 2000):
```

```python
for i in range(1000, 2000):
    md[i] = i**2

print(len(md))  # 2002
print(md[2.0])  # 4
print(md[1000])  # 1000000
print(md['hello'])  # world
print(20.0 in md)  # True
print(99 in md)  # False

del md['hello']
print('hello' in md)  # False
for i in range(1000, 2000):
    del md[i]
print(len(md))  # 1001
for f in range(1000):
    del md[f**0.5]
print(len(md))  # 1
print(md.keys())  # ['name']
print(md.values())  # ['sessdsa']
for a in md.keys():
    print(md[a])  # sessdsa
md.clear()
print(md)  # {}
```

结果:

```
========= AVL树实现字典 =========
{'hello': 'world', 'name': 'sessdsa'}
2002
4
1000000
world
True
False
False
1001
1
['name']
['sessdsa']
sessdsa
{}

Process finished with exit code 0
```