AVL tree

定义

平衡因子为1，-1或0的平衡二叉树

插入

步骤一 正常BST插入
步骤二 回溯，分情况fixup（两种大情况，四种小情况）
代码如下：

```
    def insert(self,key,val):
        newNode=self.__insert(self.root,key,val)
        if self.root==None:self.root=newNode
    def __insert(self,h,key,val):
        ##Normal BST insertio
        if self.__isEmpty(h):
            newNode=self.NODE(key,val)##Do not forget 'self.'
            return newNode
        elif h.key==key:h.val=val
        elif key>h.key:h.right=self.__insert(h.right,key,val)
        elif key<h.key:h.left=self.__insert(h.left,key,val)
        ##AVL fixing balance
        if self.__getHeight(h.left)-self.__getHeight(h.right)==2:
            if key<h.left.key:
                h=self.__rotateRight(h)
            elif key>h.left.key:
                h.left=self.__rotateLeft(h.left)
                h=self.__rotateRight(h)
        if self.__getHeight(h.right)-self.__getHeight(h.left)==2:
            if key>h.right.key:
                h=self.__rotateLeft(h)
            elif key<h.right.key:
                h.right=self.__rotateRight(h.right)
                h=self.__rotateLeft(h)
        self.__updateHeight(h)##This sentence is necessary!
        return h
```

删除

步骤一 普通BST删除 步骤二 回溯，fixup，基本上与插入差不多
代码如下:

```
def delete(self,key):
        self.__delete(self.root,key)
    def __delete(self,h,key):
        if self.__isEmpty(h):return None
        elif key>h.key:h.right=self.__delete(h.right,key)
        elif key<h.key:h.left=self.__delete(h.left,key)
        elif key==h.key:
            if self.__isEmpty(h.left) and self.__isEmpty(h.right):
                if h==self.root:self.root=None
                del(h)
                return None
            elif self.__isEmpty(h.left):
                save=h.right
                if h==self.root:self.root=save
                del(h)
                return save
            elif self.__isEmpty(h.right):
                save=h.left
                if h==self.root:self.root=save
                del(h)
                return save
            else:
                min=self.__getMin(h.right)
                h.key=min.key
                h.val=min.val
                h.right=self.__delete(h.right,min.key)
        if self.__getHeight(h.left)-self.__getHeight(h.right)==2:
            if self.__getHeight(h.left.left)>=self.__getHeight(h.left.right):
                h=self.__rotateRight(h)
            else:
                h.left=self.__rotateLeft(h.left)
                h=self.__rotateRight(h)
        if self.__getHeight(h.right)-self.__getHeight(h.left)==2:
            if self.__getHeight(h.right.right)>=self.__getHeight(h.right.right):
                h=self.__rotateLeft(h)
            else:
                h.right=self.__rotateRight(h.right)
                h=self.__rotateLeft(h)
        self.__updateHeight(h)
        return h
```

改进

用平衡因子代替树高，从而节省存储空间

其它

......