МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования

**«Ижевский государственный технический университет имени М.Т.Калашникова»**

(ФГБОУ ВПО ИжГТУ имени М.Т.Калашникова)

Лабораторная работа №1

# по дисциплине «Алгоритмы и структуры данных»

**Сбалансированные деревья**

Выполнил: студент гр. Б01-191-1зт

Шайхиев А.Ф.

Проверил(а): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Еланцев М.О.

# Постановка задачи:

Реализовать В-дерево.

Дерево должно поддерживать следующие операции:

1) Вставка элемента (целочисленный ключ + значение)

2) Получение значения элемента по ключу

3) Удаление элемента по ключу

4) Печать всех элементов дерева

# 1.В-дерево:

Исходный код:

class TreeNode:

    def \_\_init\_\_(self,key,val,left=None,right=None,parent=None):

        self.key = key

        self.payload = val

        self.leftChild = left

        self.rightChild = right

        self.parent = parent

    def hasLeftChild(self):

        return self.leftChild

    def hasRightChild(self):

        return self.rightChild

    def isLeftChild(self):

        return self.parent and self.parent.leftChild == self

    def isRightChild(self):

        return self.parent and self.parent.rightChild == self

    def isRoot(self):

        return not self.parent

    def isLeaf(self):

        return not (self.rightChild or self.leftChild)

    def hasAnyChildren(self):

        return self.rightChild or self.leftChild

    def hasBothChildren(self):

        return self.rightChild and self.leftChild

    def replaceNodeData(self,key,value,lc,rc):

        self.key = key

        self.payload = value

        self.leftChild = lc

        self.rightChild = rc

        if self.hasLeftChild():

            self.leftChild.parent = self

        if self.hasRightChild():

            self.rightChild.parent = self

class BinaryTree:

    def \_\_init\_\_(self):

        self.root = None

        self.size = 0

    def length(self):

        return self.size

    def \_\_len\_\_(self):

        return self.size

    #Вставка эллемента

    def put(self,key,val):

        if self.root:

            self.\_put(key,val,self.root)

        else:

            self.root = TreeNode(key,val)

        self.size = self.size + 1

    def \_put(self,key,val,currentNode):

        if key < currentNode.key:

            if currentNode.hasLeftChild():

                   self.\_put(key,val,currentNode.leftChild)

            else:

                   currentNode.leftChild = TreeNode(key,val,parent=currentNode)

        else:

            if currentNode.hasRightChild():

                   self.\_put(key,val,currentNode.rightChild)

            else:

                   currentNode.rightChild = TreeNode(key,val,parent=currentNode)

    def \_\_setitem\_\_(self,k,v):

       self.put(k,v)

    #получение значения эллемента по ключу

    def get(self,key):

       if self.root:

           res = self.\_get(key,self.root)

           if res:

                  print(res.payload)

           else:

                  print(None)

       else:

           return None

    def \_get(self,key,currentNode):

       if not currentNode:

           return None

       elif currentNode.key == key:

           return currentNode

       elif key < currentNode.key:

           return self.\_get(key,currentNode.leftChild)

       else:

           return self.\_get(key,currentNode.rightChild)

    def \_\_getitem\_\_(self,key):

       return self.get(key)

    def \_\_contains\_\_(self,key):

       if self.\_get(key,self.root):

           return True

       else:

           return False

    #удаление эллемента по ключу

    def delete(self,key):

      if self.size > 1:

         nodeToRemove = self.\_get(key,self.root)

         if nodeToRemove:

             self.remove(nodeToRemove)

             self.size = self.size-1

         else:

             raise KeyError('Error, key not in tree')

      elif self.size == 1 and self.root.key == key:

         self.root = None

         self.size = self.size - 1

      else:

         raise KeyError('Error, key not in tree')

    def \_\_delitem\_\_(self,key):

       self.delete(key)

    def remove(self,currentNode):

         if currentNode.isLeaf(): #leaf

           if currentNode == currentNode.parent.leftChild:

               currentNode.parent.leftChild = None

           else:

               currentNode.parent.rightChild = None

         elif currentNode.hasBothChildren(): #interior

           succ = currentNode.findSuccessor()

           succ.spliceOut()

           currentNode.key = succ.key

           currentNode.payload = succ.payload

         else: # this node has one child

           if currentNode.hasLeftChild():

             if currentNode.isLeftChild():

                 currentNode.leftChild.parent = currentNode.parent

                 currentNode.parent.leftChild = currentNode.leftChild

             elif currentNode.isRightChild():

                 currentNode.leftChild.parent = currentNode.parent

                 currentNode.parent.rightChild = currentNode.leftChild

             else:

                 currentNode.replaceNodeData(currentNode.leftChild.key,

                                    currentNode.leftChild.payload,

                                    currentNode.leftChild.leftChild,

                                    currentNode.leftChild.rightChild)

           else:

             if currentNode.isLeftChild():

                 currentNode.rightChild.parent = currentNode.parent

                 currentNode.parent.leftChild = currentNode.rightChild

             elif currentNode.isRightChild():

                 currentNode.rightChild.parent = currentNode.parent

                 currentNode.parent.rightChild = currentNode.rightChild

             else:

                 currentNode.replaceNodeData(currentNode.rightChild.key,

                                    currentNode.rightChild.payload,

                                    currentNode.rightChild.leftChild,

                                    currentNode.rightChild.rightChild)

    #печать всех эллементов дерева

    def printGivenLevel(self, root, level):

      if root == None:

        return

      if level == 1:

        print (root.payload)

      elif level > 1:

          self.printGivenLevel(root.leftChild, level-1);

          self.printGivenLevel(root.rightChild, level-1);

    def printLevelOrder(self):

      h = self.size

      i=1

      while(i<=h):

        self.printGivenLevel(self.root, i)

        i +=1

mytree = BinaryTree() #создаем экземпляр дерева (инициализируем)

Контрольный пример:

.put() – принимает в качестве аргументов ключ элемента и его значение, затем добавляет элемент в дерево.

.get() – принимает в качестве аргумента ключ элемента, и выводит его значение как результат работы.

.delete() – принимает в качестве аргумента ключ элемента, затем удаляет его значение.

.printLevelOrder() – выводит на печать все элементы дерева.

mytree = BinaryTree() #создаем экземпляр дерева (инициализируем)

#создаем начальную структуру дерева

mytree[1]="red"

mytree[2]="blue"

mytree[8]="yellow"

mytree[4]="grey"

print("Начальная структура дерева:")

mytree.printLevelOrder()

#вставка элемента (ключ + значение)

mytree.put(7,"black")

print("Вставка нового элемента:")

mytree.printLevelOrder()

#получение элемента по ключу

print("Получение элемента по ключу:")

mytree.get(7)

#удалениe элемента по ключу

print("Удаление элемента по ключу:")

mytree.delete(2)

mytree.printLevelOrder()

#печать всех элементов дерева

print("Финальное дерево,после изменений:")

mytree.printLevelOrder()

Результат работы программы:

