МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по индивидуальному заданию №1**

**«ДЕРЕВЬЯ»**

**по курсу**

**«ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ»**

Работу выполнил

Студент 39 группы

Зайцев А.С.

Преподаватель:

Нигодин Е.А.

**Рекуррентная функция, определяющая является ли бинарное дерево сбалансированным.**

Метод возвращает булево значение.

Оценка сложности:

Пространственная сложность совпадает с количеством вершин в дереве – это V.

Теперь определим временную сложность,

Первая операция – обход дерева в глубину для поиска макс. Глубины D

Далее идёт цикл от 1 до D – 1, внутри цикла производится подсчёт количества вершин на уровне обходом в глубину .

Оператор сравнения: 1.

В случае положительности цикл прерывается и возвращает False.

Иначе цикл переходит к следующей итерации.

Вовзрат True

Итого сложность будет V – Количество вершин, D - глубина

**Итеративная функция, определяющая является ли бинарное дерево сбалансированным.**

Метод возвращает булево значение.

Оценка сложности:

Пространственная сложность совпадает с количеством вершин в дереве – это V.

Теперь определим временную сложность,

Первая операция – обход дерева в ширину для поиска макс. Глубины D

Далее идёт цикл от 1 до D – 1, внутри цикла производится подсчёт количества вершин на уровне обходом в ширину .

Оператор сравнения: 1.

В случае положительности цикл прерывается и возвращает False.

Иначе цикл переходит к следующей итерации.

Вовзрат True

Итого сложность будет V – Количество вершин, D - глубина

**Экспериментальная сложность алгоритма**

Таблица №1

|  |  |  |
| --- | --- | --- |
| Количество | Рекурсия | Итерация |
| 10 | 0.0000069 | 0.0000071 |
| 100 | 0.00045 | 0.000043 |
| 1000 | 0.0054 | 0.0046 |
| 10000 | 0.43 | 0.52 |

**Результаты работы программы:**

Результат работы программы представлен на рисунке 1.

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

Рисунок 1 – Работа программы

**Текст программы:**

# Узел бинарного дерева

class Node:

value = None

left = None

right = None

def \_\_init\_\_(self, value=None, left=None, right=None):

self.value = value

self.left = left

self.right = right

class BinaryTree:

root = None

def \_append(self, value, currentNode: Node):

if value > currentNode.value:

if currentNode.right is None:

currentNode.right = Node(value)

else:

self.\_append(value, currentNode.right)

elif value < currentNode.value:

if currentNode.left is None:

currentNode.left = Node(value)

else:

self.\_append(value, currentNode.left)

def append(self, value):

if self.root is None:

self.root = Node(value)

else:

self.\_append(value, self.root)

def \_deep(self, currentNode: Node, deepValue=0) -> int:

if currentNode.right is not None and currentNode.left is not None:

return max(self.\_deep(currentNode.right, deepValue + 1), self.\_deep(currentNode.left, deepValue + 1))

elif currentNode.left is not None:

return self.\_deep(currentNode.left, deepValue + 1)

elif currentNode.right is not None:

return self.\_deep(currentNode.right, deepValue + 1)

else:

return deepValue

def \_deep\_iterative(self) -> int:

deepValue = 0

queue = [(self.root, 0)]

while queue:

curNodeWithLevel = queue.pop()

if curNodeWithLevel[1] > deepValue:

deepValue = curNodeWithLevel[1]

if curNodeWithLevel[0].right is not None:

queue.append((curNodeWithLevel[0].right, curNodeWithLevel[1] + 1))

if curNodeWithLevel[1].left is not None:

queue.append((curNodeWithLevel[0].left, curNodeWithLevel[1] + 1))

return deepValue

def \_countNodesOnLevel\_iterative(self, levelToCheck: int) -> int:

sum = 0

queue = [(self.root, 0)]

while queue:

curNodeWithLevel = queue.pop()

if curNodeWithLevel[1] == levelToCheck:

sum += 1

else:

if curNodeWithLevel[0].right is not None:

queue.append((curNodeWithLevel[0].right, curNodeWithLevel[1] + 1))

if curNodeWithLevel[1].left is not None:

queue.append((curNodeWithLevel[0].left, curNodeWithLevel[1] + 1))

return sum

def \_countNodesOnLevel(self, currentNode: Node, levelToCheck: int, currentLevel: int = 0) -> int:

if levelToCheck == currentLevel:

return 1

else:

if currentNode.right is not None and currentNode.left is not None:

return (

self.\_countNodesOnLevel(currentNode.right, levelToCheck, currentLevel + 1)

+ self.\_countNodesOnLevel(currentNode.left, levelToCheck, currentLevel + 1)

)

elif currentNode.left is not None:

return self.\_countNodesOnLevel(currentNode.left, levelToCheck, currentLevel + 1)

elif currentNode.right is not None:

return self.\_countNodesOnLevel(currentNode.right, levelToCheck, currentLevel + 1)

else:

return 0

def isBalanced(self) -> bool:

# Поиск глубины дерева

deep = self.\_deep(self.root)

print("deep", deep)

for i in range(1, deep - 1):

nodesOnLevel = self.\_countNodesOnLevel(self.root, i)

print(nodesOnLevel, i)

if nodesOnLevel < 2 \*\* i:

return False

return True

def isBalanced\_iterative(self) -> bool:

# Поиск глубины дерева

deep = self.\_deep\_iterative()

for i in range(1, deep - 1):

nodesOnLevel = self.\_countNodesOnLevel\_iterative(self.root, i)

if nodesOnLevel < 2 \*\* i:

return False

return True

def main():

arr = [4, 3, 2, 6, 1, 7]

tree = BinaryTree()

for i in arr:

tree.append(i)

print(tree.isBalanced())

arr = [1, 2, 3, 4]

tree = BinaryTree()

for i in arr:

tree.append(i)

print(tree.isBalanced())

arr = [1]

tree = BinaryTree()

for i in arr:

tree.append(i)

print(tree.isBalanced())

main()