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 выполнила

Студентка 46 группы

Баева Д.Н.

Преподаватель:

Нигодин Е.А.

Краснодар 2024

**Цель работы:** написать процедуру, отыскивающую в бинарном дереве ключ, следующий за данным (в смысле отношения порядка). Найти функцию (рекурсивную, не рекурсивную) сложности алгоритма (программы). Произвести экспериментальные расчеты, подтверждающие правильность аналитических результатов.

Рассмотрим каждый из алгоритмов в отдельности и оценим их сложность.

Функции располагаются внутри класса TreeNode. Он представляет узел бинарного дерева. У каждого узла есть значение (value), а также ссылки на левого (left) и правого (right) потомков. Также содержится метод create\_random\_tree(self, num\_nodes), который создаёт случайное бинарное упорядоченное дерево с заданным количеством вершин. Значения вершин генерируются случайно в диапазоне от 1 до 100.

**1. Рекурсивная процедура отыскания следующего в отношении порядка ключа за заданным.**

Функция find\_next\_key\_recursive рекурсивно ищет следующий ключ в бинарном дереве по отношению к заданному ключу.

Параметры функции: Функция принимает два параметра: root и key. root — это корень дерева, в котором мы ищем следующий ключ, а key — это ключ, следующий за которым мы хотим найти.

Базовый случай рекурсии: если root равен None, то есть мы достигли конца дерева и не нашли следующий ключ, функция возвращает None.

Рекурсивный случай: если значение в текущем узле (root.value) меньше или равно заданному ключу (key), то следующий ключ, если он существует, должен быть в правом поддереве. Поэтому функция рекурсивно вызывается для правого поддерева (root.right).

Если значение в текущем узле больше заданного ключа, то следующий ключ может быть либо в левом поддереве, либо это текущий узел. Функция рекурсивно вызывается для левого поддерева (root.left). Если левое поддерево возвращает None, это означает, что в левом поддереве нет ключей, больших заданного, и следующим ключом является значение текущего узла (root.value). Если левое поддерево возвращает не None, это означает, что в левом поддереве есть ключ, больший заданного, и он возвращается как следующий ключ.

**Временная сложность:**

Составляет **O(h),** где **h** — это высота дерева. Это связано с тем, что в худшем случае алгоритму придется пройти от корня до одного из листьев дерева, что соответствует высоте дерева.

**Пространственная сложность:**

Также составляет **O(h),** поскольку в худшем случае глубина рекурсивного стека может достигать высоты дерева.

**Количество операций:**

В лучшем случае, когда следующий ключ является непосредственным потомком текущего узла, временная сложность составляет O(**1**), и алгоритму требуется всего одна операция.

В худшем случае, каждая вершина посещается ровно один раз, поэтому количество операций также будет **O(n).**

Важно отметить, что эти оценки сложности предполагают, что дерево является сбалансированным. Если дерево несбалансированное, то есть, например, вырождается в связный список, то временная и пространственная сложности в худшем случае могут достигать **O(n),** где **n** — это количество узлов в дереве. В этом случае каждый узел дерева посещается ровно один раз.

Функция, рекурсивно отыскивающая в бинарном дереве ключ, следующий за данным, представлена на рисунке 1.

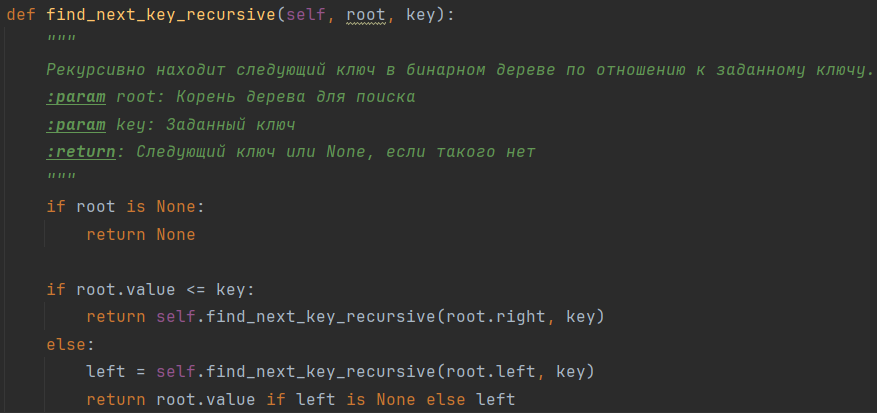


Рисунок 1 – Рекурсивное процедура.

**2. Не рекурсивная процедура отыскивающую в бинарном дереве ключ, следующий за данным.**

Функция find\_next\_key\_iterative ищет следующий ключ в бинарном дереве по отношению к заданному ключу с использованием итеративного подхода.

Параметры функции: функция принимает один параметр: key, где key — это ключ, следующий за которым мы хотим найти.

Инициализация: функция инициализирует две переменные: current и successor. current — это текущий узел, который мы рассматриваем, и он изначально устанавливается на корень дерева (self). successor — это следующий ключ, который мы ищем, и он изначально устанавливается на None.

Цикл while: функция входит в цикл while, который продолжается, пока current не станет None. В каждой итерации цикла функция проверяет, является ли значение текущего узла меньше или равным заданному ключу. Если это так, то следующий ключ, если он существует, должен быть в правом поддереве, и current обновляется на current.right. В противном случае, следующий ключ может быть либо в левом поддереве, либо это текущий узел. В этом случае successor обновляется на current.value, а current обновляется на current.left.

Возвращаемое значение: после выхода из цикла функция возвращает successor, который представляет следующий ключ после заданного ключа в бинарном дереве. Если такого ключа нет, функция возвращает None.

**Временная сложность**, **пространственная сложность** **и** **количество операций** аналогичны предыдущему случаю.

Функция, итеративно отыскивающая в бинарном дереве ключ, следующий за данным, представлена на рисунке 2.

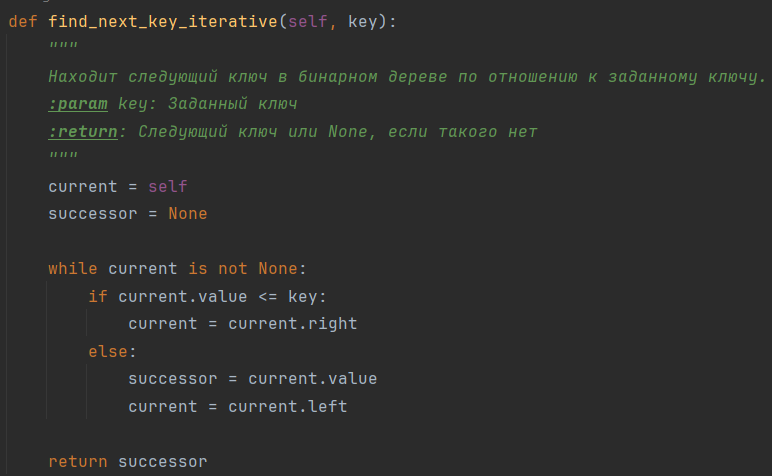


Рисунок 2 – Не рекурсивная процедура.

**Результаты работы программы:**

На рисунке 3 представлен результат работы программы.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – Результат поиска ключа, следующего за данным, в бинарном дереве.

**Экспериментальная сложность алгоритма:**

В таблице 1 представлены результаты запусков алгоритма на разном количестве вершин и время, за которое отыщется в бинарном дереве ключ, следующий за данным.

|  |  |  |
| --- | --- | --- |
| Количество вершин (шт.) | Время рекурсивной процедуры (сек.) | Время итеративной процедуры (сек.) |
| 50 | 0.000000000 | 0.000000000 |
| 100 | 0.000000000 | 0.000000000 |
| 500 | 0.000000000 | 0.000000000 |
| 1000 | 0.000000000 | 0.000000000 |
| 5000 | 0.000000000 | 0.000000000 |
| 10000 | 0.000000000 | 0.000000000 |
| 50000 | 0.000000000 | 0.000000000 |
| 70000 | 0.001001596 | 0.001001596 |
| 100000 | 0.001004457 | 0.001004457 |

**Листинг программ:**

Файл Algorithm.py:

import random

import time

class TreeNode:

def \_\_init\_\_(self, value):

self.value = value

self.left = None

self.right = None

def create\_random\_tree(self, num\_nodes):

"""

Создает случайное бинарное упорядоченное дерево с заданным количеством вершин.

Значения вершин генерируются случайно в диапазоне от 1 до 100.

:param num\_nodes: Количество вершин в дереве

:return: Корень дерева

"""

def insert\_node(root, value):

if root is None:

return TreeNode(value)

if value < root.value:

root.left = insert\_node(root.left, value)

else:

root.right = insert\_node(root.right, value)

return root

root = None

for \_ in range(num\_nodes):

value = random.randint(1, 100) # генерируем случайное значение

root = insert\_node(root, value)

return root

def find\_next\_key\_recursive(self, root, key):

"""

Рекурсивно находит следующий ключ в бинарном дереве по отношению к заданному ключу.

:param root: Корень дерева для поиска

:param key: Заданный ключ

:return: Следующий ключ или None, если такого нет

"""

if root is None:

return None

if root.value <= key:

return self.find\_next\_key\_recursive(root.right, key)

else:

left = self.find\_next\_key\_recursive(root.left, key)

return root.value if left is None else left

def find\_next\_key\_iterative(self, key):

"""

Находит следующий ключ в бинарном дереве по отношению к заданному ключу.

:param key: Заданный ключ

:return: Следующий ключ или None, если такого нет

"""

current = self

successor = None

while current is not None:

if current.value <= key:

current = current.right

else:

successor = current.value

current = current.left

return successor

if \_\_name\_\_ == "\_\_main\_\_":

# создание простого бинарного дерева

num\_nodes = 10000

root = TreeNode(None)

root = root.create\_random\_tree(num\_nodes)

# измерение времени выполнения рекурсивного метода

start\_time\_recursive = time.time()

target\_key = random.randint(1, 100)

next\_key\_1 = root.find\_next\_key\_recursive(root,target\_key)

end\_time\_recursive = time.time()

# измерение времени выполнения итеративного метода

start\_time\_iterative = time.time()

next\_key\_2 = root.find\_next\_key\_iterative(target\_key)

end\_time\_iterative = time.time()

print(f"Заданный ключ: {target\_key}, следующий ключ: {next\_key\_1}")

print(f"Время выполнения (рекурсивно): {end\_time\_recursive - start\_time\_recursive:.10f} секунд")

print(f"Заданный ключ: {target\_key}, следующий ключ: {next\_key\_2}")

print(f"Время выполнения (итеративно): {end\_time\_iterative - start\_time\_iterative:.10f} секунд")