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 практическому заданию №2**

**«Нахождение функции сложности алгоритмов»**

**по курсу**

**«ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ»**

Работу выполнила

Студентка 46 группы

Баева Д.Н.

Преподаватель:

Нигодин Е.А.

Краснодар 2024

**Цель работы:** найти функцию сложности алгоритмов для:

1. Рекурсивной процедуры подсчёта вершин в бинарном упорядоченном дереве;
2. Не рекурсивной процедуры подсчёта вершин в бинарном упорядоченном дереве;
3. Косвенной рекурсии подсчёте вершин в бинарном упорядоченном дереве.

Рассмотрим каждый из алгоритмов в отдельности и оценим их сложность.

Все три 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 подсчёта вершин в бинарном упорядоченном дереве.**

Функцияcount\_nodes\_recursive(root). Этот метод рекурсивно подсчитывает количество вершин в бинарном упорядоченном дереве. Алгоритм:

* + - Принимает корень дерева (root) в качестве аргумента;
  + Рекурсивно подсчитывает количество вершин в дереве;
  + Если root равен None, возвращает 0 (базовый случай);
  + В противном случае, рекурсивно вызывает себя для левого и правого поддеревьев, добавляя 1 к результату;
  + В итоге возвращает общее количество вершин в дереве.

**Временная сложность:**

В худшем случае каждая вершина посещается ровно один раз, поэтому время выполнения будет **O(n),** где n - количество вершин в дереве.

**Пространственная сложность:**

Рекурсивный вызов функции использует стек, поэтому дополнительная память будет использована пропорционально глубине дерева. В худшем случае это также **O(n).**

**Количество операций:**

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

Функция, вычисляющая рекурсивно количество вершин бинарного дерева, представлена на рисунке 1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 – Рекурсивное процедура.

**2. Не рекурсивная процедура подсчёта вершин в бинарном упорядоченном дереве**

Итеративный методcount\_nodes\_iterative(root). Этот метод итеративно подсчитывает количество вершин в бинарном упорядоченном дереве. Алгоритм:

* Используем стек для обхода дерева;
* Инициализируем счетчик count равным 0 и добавляем корень root в стек;
* Пока стек не пуст:
  + Извлекаем текущий узел из стека;
  + Увеличиваем счетчик на 1;
  + Добавляем правого и левого потомков в стек (если они существуют);
* Возвращаем общее количество вершин в дереве.

**Временная сложность**:

Итеративная версия также будет использовать **O(n)** времени выполнения.

**Пространственная сложность**:

Итеративный обход также использует дополнительную память (стек), поэтому пространственная сложность также **O(n)**.

**Количество операций**:

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

Функция, выполняющая итеративный подсчёт количества вершин бинарного дереве, представлена на рисунке 2.

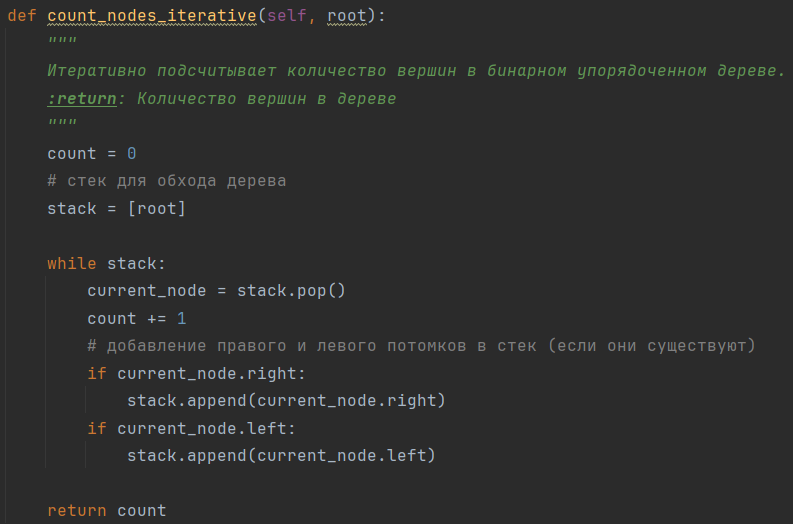


Рисунок 2 – Не рекурсивная процедура.

**3. Косвенная рекурсия подсчёта вершин в бинарном упорядоченном дереве.**

В косвенной рекурсии одна функция вызывает другую, которая в свою очередь вызывает первую.

В данном случае создаются две функции: одну для обхода дерева и другую для рекурсивного подсчета вершин.

Метод create\_random\_tree(num\_nodes) создает бинарное упорядоченное дерево с заданным количеством вершин. Значения вершин генерируются случайно в диапазоне от 1 до 100. Метод count\_nodes\_recursive(root) рекурсивно подсчитывает количество вершин в дереве.

**Временная сложность:**

Временная сложность косвенной рекурсии также остается **O(n)**.

**Пространственная сложность:**

Пространственная сложность алгоритма косвенной рекурсии зависит от использования стека вызовов во время рекурсивных вызовов функций. В данном случае у нас есть две функции, которые вызывают друг друга:рекурсивная функция count\_nodes\_recursive(root):каждый рекурсивный вызов функции создает новый экземпляр функции в стеке вызовов.Пространственная сложность будет равна глубине дерева (максимальному количеству одновременных рекурсивных вызовов).В худшем случае, когда дерево является несбалансированным (например, линейное дерево), глубина будет **O(n)**. Итеративная функция count\_nodes\_iterative(root) использует стек для обхода дерева.Пространственная сложность также будет **O(n)** из-за использования стека.В худшем случае, когда дерево является несбалансированным, глубина стека будет **O(n).**

В итоге, обе функции имеют пространственную сложность **O(n).**

**Количество итераций:**

В косвенной рекурсии каждый вызов функции создает новый экземпляр функции в стеке вызовов. Количество итераций будет равно количеству вершин в дереве.

Функция, выполняющая косвенную рекурсию подсчёта вершин бинарного дерева, представлена на рисунке 3.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – Функция сортировки вставкой.

**Результаты работы программы:**

На рисунке 4 представлен результат работы программы.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 4 – Результат скалярного перемножение векторов.

**Экспериментальная сложность алгоритма:**

В таблице 1 представлены результаты запусков алгоритма на разном количестве вершин и время, за которое выполнялся подсчёт количества вершин бинарного упорядоченного дерева.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество вершин (шт.) | Время рекурсивной процедуры (сек.) | Время итеративной процедуры (сек.) | Время косвенной рекурсии (сек.) |
| 50 | 0.000000000 | 0.000000000 с | 0.000000000 |
| 100 | 0.000000000 | 0.000000000 | 0.000000000 |
| 500 | 0.000000000 | 0.000000000 с | 0.000000000 |
| 1000 | 0.000000000 | 0.000000000 | 0.000000000 |
| 3000 | 0.001001596 | 0.001004457 | 0.000000000 |
| 5000 | 0.001004457 | 0.001001596 | 0.001005411 |
| 5500 | 0.003004789 | 0.001003265 | 0.001003027 |
| 7000 | 0.002006769 | 0.002393723 | 0.002002239 |
| 100000 | 0.003073215 | 0.004009962 | 0.003062725 |

**Листинг программ:**

Файл Algorithms.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 count\_nodes\_recursive(self, root):

"""

Рекурсивно подсчитывает количество вершин в бинарном упорядоченном дереве.

:param root: Корень дерева

:return: Количество вершин в дереве

"""

if root is None:

return 0

return 1 + self.count\_nodes\_recursive(root.left) + self.count\_nodes\_recursive(root.right)

def count\_nodes\_iterative(self, root):

"""

Итеративно подсчитывает количество вершин в бинарном упорядоченном дереве.

:return: Количество вершин в дереве

"""

count = 0

# стек для обхода дерева

stack = [root]

while stack:

current\_node = stack.pop()

count += 1

# добавление правого и левого потомков в стек (если они существуют)

if current\_node.right:

stack.append(current\_node.right)

if current\_node.left:

stack.append(current\_node.left)

return count

def count\_nodes\_indirect(self):

"""

Косвенно подсчитывает количество вершин в бинарном упорядоченном дереве.

:return: Количество вершин в дереве

"""

def count\_recursive(node):

if node is None:

return 0

return 1 + count\_recursive(node.left) + count\_recursive(node.right)

return count\_recursive(self)

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

# создание простого бинарного дерева

num\_nodes = 50

root = TreeNode(None)

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

# измерение времени выполнения рекурсивного метода

start\_time\_recursive = time.time()

num\_nodes\_recursive = root.count\_nodes\_recursive(root)

end\_time\_recursive = time.time()

# измерение времени выполнения итеративного метода

start\_time\_iterative = time.time()

num\_nodes\_iterative = root.count\_nodes\_iterative(root)

end\_time\_iterative = time.time()

# измерение времени выполнения косвенной рекурсии

start\_time\_indirect = time.time()

num\_nodes\_indirect = root.count\_nodes\_indirect()

end\_time\_indirect = time.time()

print(f"Количество вершин в дереве (рекурсивно): {num\_nodes\_recursive}")

print(f"Время выполнения (рекурсивно): {end\_time\_recursive - start\_time\_recursive:.9f} секунд")

print(f"Количество вершин в дереве (итеративно): {num\_nodes\_iterative}")

print(f"Время выполнения (итеративно): {end\_time\_iterative - start\_time\_iterative:.9f} секунд")

print(f"Количество вершин в дереве (косвенно): {num\_nodes\_indirect}")

print(f"Время выполнения (косвенная рекурсия): {end\_time\_indirect - start\_time\_indirect:.9f} секунд")