САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5
по курсу «Алгоритмы и структуры данных»
Деревья. Пирамида, пирамидальная сортировка. Очередь с приоритетами.

Вариант 15

Выполнил:
Левахин Лев Александрович
К3140

Проверил:
--

Санкт-Петербург
2024 г.

Содержание отчета

Задания по варианту: 3, 7

Задача 3 – Обработка сетевых пакетов

Задача 7 – Снова сортировка?

Задания по выбору: 1, 2

Задача 1 - Куча ли?

Задача 2 - Высота дерева

Задачи по выбору

Задача №1. Куча ли?

Листинг кода.

```
"""
Куча ли?
"""

from lab5 import utils
import os


CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def is_heap(n:int, lst:list) -> str:
    for i in range(1, n + 1):
        left_child_index = 2 * i
        right_child_index = 2 * i + 1

        if left_child_index <= n:
            if lst[i - 1] > lst[left_child_index - 1]:
                return "NO"

        if right_child_index <= n:
            if lst[i - 1] > lst[right_child_index - 1]:
                return "NO"

    return "YES"


def main():
    n, lst = input_data()
    result = is_heap(n, lst)
    return result


def input_data() -> (int, list):
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    n, lst = int(data[0]), utils.str_to_list(data[1])
    return n, lst


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, [result])
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab4 import utils
import datetime
import tracemalloc
import unittest

from lab5.task1.src.task1 import is_heap


class TaskTest1(unittest.TestCase):

    def test_is_heap_performance(self):
        """Тест функции на время и память"""
        # given
        n, lst = 5, utils.str_to_list("4 -1 4 1 1")
        max_allowed_time = datetime.timedelta(seconds=2)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = is_heap(n, lst)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, "NO")
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 256)


    def test_is_heap_correctly(self):
        """Тест на корректность работы"""
        # given
        n1, lst1 = 5, utils.str_to_list("4 -1 4 1 1")
        n2, lst2 = 5, utils.str_to_list("-1 0 4 0 3")

        # when
        result1 = is_heap(n1, lst1)
        result2 = is_heap(n2, lst2)

        # then
        self.assertEqual(result1, "NO")
        self.assertEqual(result2, "YES")


if __name__ == "__main__":
    unittest.main()
```

Скрины работы тестов:

Задача №2. Высота дерева

Листинг кода.

```
"""
Высота дерева
```

```
"""

from lab5 import utils
import os


CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def build_tree(n:int, parents:list):
    """
    Создает список смежности для дерева
    Каждый индекс в списке представляет узел, а значения — его дочерние
узлы
    Находит корень дерева
    """
    tree = [[] for i in range(n)]
    root = None
    for child in range(n):
        parent = parents[child]
        if parent == -1:
            root = child
        else:
            tree[parent].append(child)
    return tree, root


def find_tree_height(tree:list[list], node:int):
    """
    Рекурсивно вычисляет высоту дерева
     - Если узел не имеет детей - возвращает 1
     - Для каждого дочернего узла рекурсивно вызывается функция и находится
максимальная высота из дочерних узлов
     """
    if not tree[node]:  # Если у узла нет детей
        return 1
    else:
        height = 0
        for child in tree[node]:
            height = max(height, find_tree_height(tree, child))
        return height + 1


def input_data() -> (int, list):
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    cnt_nodes, parents = int(data[0]), utils.str_to_list(data[1])
    return cnt_nodes, parents


def main():
    cnt_nodes, parents = input_data()
    tree, root = build_tree(cnt_nodes, parents)
    tree_height = find_tree_height(tree, root)

    return tree_height


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, [result])
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab5.task2.src.task2 import *
from lab5 import utils
import datetime
import tracemalloc
import unittest


class TaskTest2(unittest.TestCase):

    def test_is_heap_performance(self):
        """Тест функции на время и память"""
        # given
        n, parents = 5, utils.str_to_list("4 -1 4 1 1")
        max_allowed_time = datetime.timedelta(seconds=3)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        tree, root = build_tree(n, parents)
        tree_height = find_tree_height(tree, root)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(tree_height, 3)
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 512)

    def test_is_heap_correctly(self):
        """Тест на корректность работы"""
        # given
        n1, parents1 = 5, utils.str_to_list("4 -1 4 1 1")
        n2, parents2 = 5, utils.str_to_list("-1 0 4 0 3")

        # when
        tree1, root1 = build_tree(n1, parents1)
        tree_height1 = find_tree_height(tree1, root1)
        tree2, root2 = build_tree(n2, parents2)
        tree_height2 = find_tree_height(tree2, root2)

        # then
        self.assertEqual(tree_height1, 3)
        self.assertEqual(tree_height2, 4)


if __name__ == "__main__":
    unittest.main()
```

Задачи по варианту

Задача №3. Обработка сетевых пакетов

Листинг кода.

```
from lab5 import utils
import os


CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def process_packets(s:int, n:int, packets_info:list) -> list:
    buffer = []  # очередь для хранения пакетов
    current_time = 0  # текущее время
    results = []

    for packet in packets_info:
        arrival_time = packet[0]
        processing_time = packet[1]

        # Удаляем обработанные пакеты из буфера
        while buffer and buffer[0] <= arrival_time:
            buffer.pop(0)

        # Проверяем, можем ли мы добавить пакет в буфер
        if len(buffer) < s:
            # Если процессор свободен, обновляем текущее время до времени
прибытия пакета
            if current_time < arrival_time:
                current_time = arrival_time

            # Начинаем обработку нового пакета
            results.append(current_time)  # время начала обработки
            current_time += processing_time  # обновляем текущее время на
время обработки
            buffer.append(current_time)
        else:
            results.append(-1)  # пакет отброшен

    return results


def input_data_handler(data:list[list]) -> (int, int, list):
    """
    Обрабатывает входные данные, разделяя их
    - Принимает: список со всеми данными
    - Возвращает: (s - размер буфера, n - количество пакетов, packets_info
- информация о передаваемых пакетах)
    """
    s, n = data[0][0], data[0][1]
    packets_info = data[1:]
    return s, n, packets_info


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    result = []
    for s in data:
        if s == '':
            break
        s, n = map(int, s.split())
        result.append([s, n])
```

```
    return result


def main():
    data = input_data()
    s, n, packets_info = input_data_handler(data)
    result = process_packets(s, n, packets_info)

    return result


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, result)
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab5.task3.src.task3 import *
from lab5 import utils
import datetime
import tracemalloc
import unittest


class TaskTest3(unittest.TestCase):

    def test_func_performance(self):
        """Тест на время и память"""
        # given
        s, n, packets_info = 3, 6, [[0, 2], [1, 2], [2, 2], [3, 2], [4, 2],
[5, 2]]
        max_allowed_time = datetime.timedelta(seconds=10)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = process_packets(s, n, packets_info)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, [0, 2, 4, 6, 8, -1])
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 512)

    def test_func_correctly(self):
        """Тест на корректность работы"""
```

```
        # given
        s1, n1, packets_info1 = 1, 0, []
        s2, n2, packets_info2 = 1, 1, [[0,0]]
        s3, n3, packets_info3 = 1, 2, [[0,1], [0,1]]
        s4, n4, packets_info4 = 1, 2, [[0,1], [1,1]]
        s12, n12, packets_info12 = 3, 6, [[0, 2], [1, 2], [2, 2], [3, 2],
[4, 2], [5, 2]]

        # when
        result1 = process_packets(s1, n1, packets_info1)
        result2 = process_packets(s2, n2, packets_info2)
        result3 = process_packets(s3, n3, packets_info3)
        result4 = process_packets(s4, n4, packets_info4)
        result12 = process_packets(s12, n12, packets_info12)

        # then
        self.assertEqual(result1, [])
        self.assertEqual(result2, [0])
        self.assertEqual(result3, [0, -1])
        self.assertEqual(result4, [0, 1])
        self.assertEqual(result12, [0, 2, 4, 6, 8, -1])


if __name__ == "__main__":
    unittest.main()
```

Скрины работы тестов:

Задача №7. Снова сортировка?

Листинг кода.

```
"""
Пирамидальная сортировка в невозрастающем порядке
"""

from lab5 import utils
import os


CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def heapify(lst: list, n: int, i: int):
    """
    Проверяет, является ли поддерево с корнем в i кучей
    Если нет, она перестраивает его так, чтобы оно стало кучей
    Это делается рекурсивно для дочерних элементов
    """
    smallest = i  # Инициализируем корень дерева как самый маленький
элемент
    left = 2 * i + 1  # левый элемент
    right = 2 * i + 2  # правый элемент

    # Если левый элемент меньше корня
    if left < n and lst[left] < lst[smallest]:
        smallest = left

    # Если правый элемент меньше самого маленького элемента
    if right < n and lst[right] < lst[smallest]:
```

```
        smallest = right

    # Если самый маленький элемент не корень
    if smallest != i:
        lst[i], lst[smallest] = lst[smallest], lst[i]

        # Рекурсивно хипифай для затронутого поддерева
        heapify(lst, n, smallest)


def heap_sort(n: int, lst: list) -> list:
    """
    Пирамидальная сортировка или сортировка кучей
     - Сначала строится куча из массива
     - Извлекаются элементы из кучи
     - Самый маленький элемент (корень) перемещается в конец массива и
вызывается heapify для уменьшенной кучи.
    """
    # Построение кучи
    for i in range(n // 2 - 1, -1, -1):
        heapify(lst, n, i)

    # Извлекаем элементы из кучи
    for i in range(n - 1, 0, -1):
        lst[i], lst[0] = lst[0], lst[i]  # Перемещаем текущий корень в
конец
        heapify(lst, i, 0)  # Вызываем хипифай на уменьшенной куче

    return lst


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    n, lst = int(data[0]), utils.str_to_list(data[1])

    return n, lst


def main():
    n, lst = input_data()
    result = heap_sort(n, lst)

    return result


if __name__ == "__main__":
    result = main()

    utils.write_file(CURRENT_SCRIPT_DIR_PATH, [result])
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab5.task7.src.task7 import *
from lab5 import utils
```

```
import datetime
import tracemalloc
import unittest
import random


class TaskTest7(unittest.TestCase):

    def test_func_performance(self):
        """Тест на время и память"""
        # given
        n = 21
        lst = utils.str_to_list("42 23 16 15 8 4 6 5 4 3 2 1 2 3 1000000000
100000000 10000000 1000000 100 1000 12873891")

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = heap_sort(n, lst)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result,
[1000000000,100000000,12873891,10000000,1000000,1000,100,42,23,16,15,8,6,5,
4,4,3,3,2,2,1])

    # Тест сортировки на худших данных
    def test_heap_sort_hard(self):
        """Тест сортировки на самых больших данных"""
        # given
        n_hud = 10 ** 5
        lst_hud = [random.randint(10**3, 10 ** 9) for i in range(n_hud)]

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = heap_sort(n_hud, lst_hud)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(lst_hud, sorted(lst_hud, reverse=True))
        print(f"\nНа самых больших данных сортировка отработала за
{spent_time} с, использовав памяти {memory_used} Мб")

    # Тест сортировки на средних данных
    def test_heap_sort_middle(self):
        """Тест сортировки на средних данных"""
        # given
        n_mid = 10 ** 3
        lst_mid = [random.randint(1, 10 ** 9) for i in range(n_mid)]

        # when
```

```
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = heap_sort(n_mid, lst_mid)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(lst_mid, sorted(lst_mid, reverse=True))
        print(f"\nНа средних данных сортировка отработала за {spent_time}
с, использовав памяти {memory_used} Мб")


if __name__ == "__main__":
    unittest.main()
```

Вывод

На языке Python можно решать сложные задачи и реализовывать разные типы данных.