САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5

по курсу «Алгоритмы и структуры данных»

Тема: Тема работы

Вариант 23

Выполнил:

Юшков А.М.

К3139

Санкт-Петербург

2025г.

# Содержание отчета

[**Содержание отчета**](#_10udeakjagvs) **2**

[**Задачи по варианту**](#_gf7yxvsg0zb) **3**

[Задача №2.](#_pgzbaj56cf0o) Высота дерева 3

[Задача №7.](#_pgzbaj56cf0o) Снова сортировка 4

[**Вывод**](#_fu90fuyk873) **5**

# Задачи по варианту

## Задача №2. Высота дерева

from lab5.utils import file\_read, file\_write, measure  
  
def calculate\_height(n, parents):  
 children = [[] for \_ in range(n)]  
 root = -1  
  
 for child\_index, parent\_index in enumerate(parents):  
 if parent\_index == -1:  
 root = child\_index  
 else:  
 children[parent\_index].append(child\_index)  
  
 def height(node):  
 if not children[node]:  
 return 1  
 return 1 + max(height(child) for child in children[node])  
  
 return height(root)  
  
def process\_tree\_height():  
 data = file\_read()  
 n = int(data[0][0])  
 parents = list(map(int, data[1]))  
 height = calculate\_height(n, parents)  
 file\_write([height])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 measure(process\_tree\_height)

Текстовое объяснение решения.

Программа эффективно вычисляет высоту дерева, используя рекурсивный подход и представление дерева в виде списка дочерних узлов.

Описание алгоритма

**Входные данные**:

n — количество узлов в дереве.

parents — массив, где каждый элемент указывает на родителя текущего узла. Корень дерева обозначается значением -1.

**Построение дерева**:

Создаётся список children, где для каждого узла хранятся его дочерние узлы.

Корень дерева определяется по значению -1 в массиве parents.

**Рекурсивный расчёт высоты**:

Для каждого узла вычисляется высота его поддерева.

Высота узла равна 1 (текущий узел) плюс максимальная высота среди его дочерних узлов.

Если узел не имеет дочерних узлов (лист), его высота равна 1.

**Результат**:

Программа возвращает высоту дерева, начиная с корня.

Результат работы кода на примере из текста задачи

Input.txt

5  
-1 0 4 0 3

Output.txt

4

## Задача №7. Снова сортировка

from lab5.utils import file\_read, file\_write, measure  
  
  
def max\_heapify(arr, n, i):  
 largest = i  
 left = 2 \* i + 1  
 right = 2 \* i + 2  
  
 while True:  
 if left < n and arr[left] > arr[largest]:  
 largest = left  
 if right < n and arr[right] > arr[largest]:  
 largest = right  
 if largest != i:  
 arr[i], arr[largest] = arr[largest], arr[i]  
 i = largest  
 left = 2 \* i + 1  
 right = 2 \* i + 2  
 else:  
 break  
  
  
def heap\_sort(arr):  
 n = len(arr)  
  
 for i in range(n // 2 - 1, -1, -1):  
 max\_heapify(arr, n, i)  
  
 for i in range(n - 1, 0, -1):  
 arr[0], arr[i] = arr[i], arr[0]  
 max\_heapify(arr, i, 0)  
  
 return arr[::-1]  
  
  
def process\_heap\_sort():  
 data = file\_read()  
 n = int(data[0][0])  
 arr = list(map(int, data[1]))  
 sorted\_arr = heap\_sort(arr)  
 file\_write([" ".join(map(str, sorted\_arr))])  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 measure(process\_heap\_sort)

Программа эффективно сортирует массив чисел в порядке убывания, используя алгоритм сортировки кучей. Этот метод обладает временной сложностью **O(n log n)** и является устойчивым и эффективным для больших объёмов данных.

Алгоритм:

**Входные данные**:

n — количество элементов в массиве.

arr — массив чисел, который нужно отсортировать.

**Построение max-кучи**:

Функция max\_heapify преобразует поддерево с корнем в узле i в max-кучу (где родительский узел всегда больше дочерних).

Для каждого узла, начиная с середины массива и до корня, вызывается max\_heapify, чтобы построить max-кучу для всего массива.

**Сортировка**:

Максимальный элемент (корень кучи) перемещается в конец массива.

Размер кучи уменьшается на 1, и max\_heapify вызывается для нового корня, чтобы восстановить свойства max-кучи.

Процесс повторяется, пока весь массив не будет отсортирован.

**Результат**:

Массив возвращается в обратном порядке (отсортированный по убыванию).

Результат работы кода на примере из текста задачи

Input.txt

5  
3 5 1 4 2

Output.txt

5 4 3 2 1

# Вывод

В процессе выполнения лабораторной работы, посвящённой сортировке и изучению деревьев, были разработаны и протестированы несколько алгоритмов сортировки и алгоритмов сортировки за линейное время. Каждая программа решает конкретную задачу, демонстрируя эффективное использование структур данных и алгоритмических подходов. Результаты работы программ проверены на корректность, а их производительность оценена с помощью модуля measure.