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: Деревья. Пирамида, пирамидальная сортировка. Очередь с приоритетами
Вариант 1

Выполнил:
Бен Шамех Абделазиз
Группа: K3239

Проверила:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание

1 Задача 1. Куча ли? 2
1.1 Условие . . . 2
1.2 Листинг кода . . . 2
1.3 Текстовое объяснение . . . 2
1.4 Результаты выполнения . . . 2

2 Задача 2. Высота дерева 3
2.1 Условие . . . 3
2.2 Листинг кода . . . 3
2.3 Текстовое объяснение . . . 3
2.4 Результаты выполнения . . . 3

3 Задача 4. Построение пирамиды 4
3.1 Условие . . . 4
3.2 Листинг кода . . . 4
3.3 Текстовое объяснение . . . 4
3.4 Результаты выполнения . . . 4

4 Задача 6. Очередь с приоритетами 5
4.1 Условие . . . 5
4.2 Листинг кода . . . 5
4.3 Текстовое объяснение . . . 5
4.4 Результаты выполнения . . . 5

5 Вывод 7

1 Задача 1. Куча ли?

1.1 Условие

Дан массив целых чисел. Определите, является ли он неубывающей пирамидой. Основное свойство: для каждого i должны выполняться условия $a_i \leq a_{2i}$ и $a_i \leq a_{2i+1}$.

1.2 Листинг кода

```
def is_heap(arr):
    for i in range(len(arr) // 2):
        rt = 2 * i + 2
        lt = 2 * i + 1

        if (rt < len(arr) and arr[i] > arr[rt]) or (lt < len(arr) and
   arr[i] > arr[lt]):
            return "NO"

    return "YES"
```

1.3 Текстовое объяснение

Проверяем основное свойство неубывающей кучи: проходим по родительским узлам и сравниваем их значения со значениями детей. Если хотя бы один ребенок меньше родителя, возвращаем "NO".

1.4 Результаты выполнения

Примеры:

- **Input:** `5, 1 0 1 2 0` → **Output:** `NO`
- **Input:** `5, 1 3 2 5 4` → **Output:** `YES`

Метрики эффективности:

Сценарий	**Время выполнения**	**Затраты памяти**
Нижняя граница (n=5)	0.00211 sec	14.63 Mb
Пример из задачи	0.00084 sec	14.63 Mb
Верхняя граница (n=10^6)	0.44345 sec	33.12 Mb

2 Задача 2. Высота дерева

2.1 Условие

Вычислить высоту произвольного корневого дерева, представленного в виде массива родительских связей.

2.2 Листинг кода

```python
from collections import deque

class TreeNode:
    def __init__(self, key):
        self.key = key
        self.children = []

def get_tree_height(arr):
    nodes_arr = [TreeNode(i) for i in range(len(arr))]
    root_id = -1
    for i in range(len(arr)):
        if arr[i] != -1:
            nodes_arr[arr[i]].children.append(nodes_arr[i])
        else:
            root_id = i

    # BFS find height
    queue = deque([(nodes_arr[root_id], 1)])
    max_h = 0
    while queue:
        node, h = queue.popleft()
        max_h = max(max_h, h)
        for child in node.children:
            queue.append((child, h + 1))
    return max_h
```

2.3 Текстовое объяснение

Строим дерево на основе массива родителей. С помощью алгоритма поиска в ширину (BFS) обходим дерево уровень за уровнем, чтобы найти максимальное расстояние от корня до самого глубокого листа.

2.4 Результаты выполнения

Пример: Ввод: `5, 4 -1 4 1 1` → Вывод: `3`.

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=3)	0.00543 sec	14.33 Mb
Пример из задачи	0.00108 sec	15.18 Mb
Верхняя граница ($n=10^6$)	0.61734 sec	36.59 Mb

3 Задача 4. Построение пирамиды

3.1 Условие

Преобразовать массив в min-heap (неубывающая пирамида), используя только $O(n)$ перестановок. Вывести список произведенных свопов.

3.2 Листинг кода

```python
def recursion(ind, n, data, ans_swap):
    min_index = ind
    left = 2 * ind + 1
    right = 2 * ind + 2

    if left < n and data[left] < data[min_index]:
        min_index = left
    if right < n and data[right] < data[min_index]:
        min_index = right

    if ind != min_index:
        data[ind], data[min_index] = data[min_index], data[ind]
        ans_swap.append((ind, min_index))
        recursion(min_index, n, data, ans_swap)

def get_swap_arr(data):
    ans_swap = []
    for i in range(len(data) // 2 - 1, -1, -1):
        recursion(i, len(data), data, ans_swap)
    return ans_swap
```

3.3 Текстовое объяснение

Рекурсивно проверяем, больше ли родитель, чем дети. Если да, то меняем родителя и минимального ребенка местами, запуская процесс «просеивания вниз» (sift-down). Это позволяет построить кучу за линейное время.

3.4 Результаты выполнения

Пример: Ввод: `5, 5 4 3 2 1` → Вывод свопов: `(1 4), (0 1), (1 3)`.

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница	0.00122 sec	14.83 Mb
Пример из задачи	0.00101 sec	14.90 Mb
Верхняя граница	0.61734 sec	33.47 Mb

4 Задача 6. Очередь с приоритетами

4.1 Условие

Реализовать структуру «очередь с приоритетами» с поддержкой операций добавления элемента, извлечения минимума и уменьшения ключа.

4.2 Листинг кода

```
import heapq

class PriorityQueue:
    def __init__(self):
        self.heap = []
        self.position = {}
        self.index = 0
        self.ans_arr = []

    def add_element(self, item):
        heapq.heappush(self.heap, (item, self.index))
        self.position[self.index] = item
        self.index += 1

    def pop_minimum(self):
        while self.heap and (self.heap[0][1] not in self.position or
                             self.position[self.heap[0][1]] != self.heap
   [0][0]):
            heapq.heappop(self.heap)
        if self.heap:
            value, idx = heapq.heappop(self.heap)
            self.ans_arr.append(str(value))
            del self.position[idx]
        else:
            self.ans_arr.append('*')

    def replace_item_with_index(self, idx, item):
        idx -= 1
        if idx in self.position:
            self.position[idx] = item
            heapq.heappush(self.heap, (item, idx))
```

4.3 Текстовое объяснение

Используем стандартную библиотеку `heapq` для реализации кучи. Для поддержки операции изменения ключа используем словарь `position`, который позволяет отслеживать актуальные значения элементов по их порядковому номеру добавления.

4.4 Результаты выполнения

Пример: Ввод: `A 3, A 4, A 2, X, D 2 1, X` → Вывод: `2, 1`.

Метрики эффективности:

Сценарий	**Время выполнения**	**Затраты памяти**
Нижняя граница (n=3)	0.00238 sec	15.31 Mb
Пример из задачи	0.00088 sec	15.21 Mb
Верхняя граница (n=10^6)	0.73464 sec	37.32 Mb

5 Вывод

1. В данной лабораторной работе мы изучили структуру данных пирамида (двоичная куча) и её применение в очередях с приоритетами и пирамидальной сортировке (Heapsort).

2. Реализовали алгоритмы построения кучи за линейное время $O(n)$ и операции извлечения минимума за $O(\log n)$.

3. С помощью метрик было подтверждено, что данные алгоритмы масштабируются эффективно, позволяя обрабатывать до 10^6 элементов в рамках временных ограничений.