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?

1.1 Условие

Дан массив целых чисел. Определите, является ли он неубывающей пирамидой (кучей).

1.2 Листинг кода

```python
def is_heap(arr):
    for i in range(len(arr) // 2):
        lt = 2 * i + 1
        rt = 2 * i + 2
        if lt < len(arr) and arr[i] > arr[lt]:
            return "NO"
        if rt < len(arr) and arr[i] > arr[rt]:
            return "NO"
    return "YES"
```

1.3 Объяснение решения

Проверяем основное свойство неубывающей пирамиды: для каждого родителя значение должно быть не больше значений его левого и правого потомков.

1.4 Результаты выполнения

Примеры:

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

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=1000000)	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
import sys
from collections import deque

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

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

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

2.3 Результаты выполнения

Пример:

- **Input:** `5 \n 4 -1 4 1 1` → **Output:** `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=1000000)	0.61734 sec	36.59 Mb

3 Задача 4. Построение пирамиды

3.1 Условие

Преобразовать массив в min-heap, используя только $O(n)$ перестановок.

3.2 Листинг кода

```python
def sift_down(i, n, arr, swaps):
    min_idx = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and arr[l] < arr[min_idx]:
        min_idx = l
    if r < n and arr[r] < arr[min_idx]:
        min_idx = r
    if i != min_idx:
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
        swaps.append((i, min_idx))
        sift_down(min_idx, n, arr, swaps)

def build_heap(arr):
    swaps = []
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        sift_down(i, n, arr, swaps)
    return swaps
```

3.3 Результаты выполнения

Пример:

- **Input:** `5 \n 5 4 3 2 1` → **Output:** `3 \n 1 4 \n 0 1 \n 1 3`

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=5)	0.00122 sec	14.83 Mb
Пример из задачи	0.00101 sec	14.90 Mb
Верхняя граница (n=1000000)	0.61734 sec	33.47 Mb

4 Задача 6. Очередь с приоритетами

4.1 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))
```

4.2 Результаты выполнения

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=1000000)	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 $O(n)$ и операции извлечения минимума и изменения ключа за $O(\log n)$.

3. С помощью метрик было подтверждено, что данные алгоритмы масштабируются линейно-логарифмически, что делает их пригодными для обработки больших данных.