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.

Выполнил:
Мезенцев Богдан
К 3139
Проверил:
Афанасьев Антон Владимирович

Санкт-Петербург
2024 г.

Содержание отчёта

Задачи по варианту

Задание 1. Куча ли?

Реализация функции check_array(), которая проверяет структуру массива:

```
def check_array(n, A):
    """Проверяет, является ли массив кучей или нет"""
    A = [0] + A
    for i in range(1, n + 1):
        if 2 * i <= n and A[i] > A[2 * i]:
            return 'NO'
        if ((2 * i) + 1) <= n and A[i] > A[(2 * i) + 1]:
            return 'NO'
    return 'YES'
```

Данная функция проверяет два условия, при выполнении которых массив будет являться кучей и функция вернет значение ‘YES’. В противном случае функция вернет значение ‘NO’.

Задание 2. Высота дерева

Реализация функции hight_of_tree():

```
def hight_of_tree(n, parents):
    """Вычесляет высоту дерева"""

    children = [[] for _ in range(n)]  # Создаем список детей для каждого узла
    root = None

    for child, parent in enumerate(parents):
        if parent == -1:
            root = child  # Узел без родителя является корневым
        else:
            children[parent].append(child)

    queue = deque([(root, 1)])
    height = 0

    while queue:
        node, level = queue.popleft()
        height = max(height, level)
        for child in children[node]:
            queue.append((child, level + 1))

    return str(height)
```

Данная функция строит список для каждого узла, где находятся дети.
Далее с помощью цикла проверяется каждый узел на наличие детей.
Узел без родителя является называется корневым.
Далее для создания очереди используется импортируемы класс queue, с помощью которого создается очередь с корневым элементом.
Высота самого дерева находится после полного прохода по очереди и проверки ее элементов.

Задание 4. Построение пирамиды

Реализация функции create_heap():

```python
def create_heap(n, A):

    swaps = []  # массив с перестановками

    def min_heap(i):
        min_index = i
        left = 2 * i + 1  # Левый ребенок
        right = 2 * i + 2  # Правый ребенок

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

        # Если ребенок меньше текущего узла, то меняем их местами
        if i != min_index:
            A[i], A[min_index] = A[min_index], A[i]
            swaps.append((i, min_index))
            min_heap(min_index)

    for i in range(n // 2 - 1, -1, -1):
        min_heap(i)

    return swaps
```

Данная функция строит дерево(пирамиду) и подсчитывает количество перестановок элементов. Все перестановки будут хранится в списке swaps. Для каждого элемента полученного списка A, вызывается функция min_heap(), которая строит минимальное дерево для каждого элемента пирамид. Также, если ребенок этого элемента меньше ее самого, то они меняются местами и список swaps пополняется. В итоге функция возвращает данный список перестановок.

Задание 7. Снова сортировка

Реализация функции heap_sort_reverse():

```python
def heap_sort_reverse(n, A):
    """Пирамидальная сортировка в убывающем порядке"""

    # Построение max-heap для каждого корня i
    for i in range(n // 2 - 1, -1, -1):
        heapify(A, n, i)

    # Извлечение элементов из кучи в порядке убывания
    for i in range(n - 1, 0, -1):
        A[0], A[i] = A[i], A[0]
        heapify(A, i, 0)

    # Перестановка элементов в убывающем порядке
    for i in range(n // 2):
        A[i], A[n - 1 - i] = A[n - 1 - i], A[i]

    return A
```

Данная функция реализует алгоритм пирамидальной сортировки в убывающем порядке. Для каждого элемента массива вызывается функция heapify(), которая строит дерево для этого элемента. Далее элементы в куче меняются местами, пока они не будут отсортированными в обратном порядке.