САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5
по курсу «Алгоритмы и структуры данных»
Тема: Деревья. Пирамида, пирамидальная сортировка. Очередь с приоритетами.
Вариант 17

Выполнил:
Останин А. С.
К3140

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №4. Построение пирамиды

В этой задаче вы преобразуете массив целых чисел в пирамиду. Это важнейший шаг алгоритма сортировки под названием HeapSort. Гарантированное время работы в худшем случае составляет O(n log n), в отличие от среднего времени работы QuickSort, равного O(n log n). QuickSort обычно используется на практике, потому что обычно он быстрее, но HeapSort используется для внешней сортировки, когда вам нужно отсортировать огромные файлы, которые не помещаются в памяти вашего компьютера. Первым шагом алгоритма HeapSort является создание пирамиды (heap) из массива, который вы хотите отсортировать. Ваша задача - реализовать этот первый шаг и преобразовать заданный массив целых чисел в пирамиду. Вы сделаете это, применив к массиву определенное количество перестановок (swaps). Перестановка - это операция, как вы помните, при которой элементы ai и aj массива меняются местами для некоторых i и j. Вам нужно будет преобразовать массив в пирамиду, используя только O(n) перестановок. Обратите внимание, что в этой задаче вам нужно будет использовать min-heap вместо max-heap.

```python
from lab5.utils import read_data, write_data


def min_heap(array, len_ar):
    """
    Возвращает индексы перестановок для пирамиды min-heap

    Пример: [5, 4, 3, 2, 1] -> [(0, 4), (1, 4), (2, 4), (3, 4)]
    """
    perms = []
    c = 0
    for i in range(1, len_ar):
        perm_i = i - 1
        perm_j = array.index(min(array[i:]))
        if array[i - 1] > array[perm_j]:
            array[perm_i], array[perm_j] = array[perm_j], array[perm_i]
            perms.append((perm_i, perm_j))
            c += 1
    return [perms, c]


def task1():
    PATH_INPUT = '../txtf/input.txt'
    PATH_OUTPUT = '../txtf/output.txt'

    data = read_data(PATH_INPUT)
    array_ = [int(i) for i in data[1].split(' ')]
    n = int(data[0])
    result = min_heap(array_, n)

    # Записывает результат в файл, используя форматирование
    # Пример: [[(0, 4), (1, 3)], 2] -> '2\n 0 4\n 1 3'
```

```
    write_data(PATH_OUTPUT, f'{result[1]}\n' + '\n'.join([str(key) + ' ' +
str(value) for key, value in result[0]]))
    print(result)


if __name__ == "__main__":
    task1()
```

Для построения min-heap пирамиды мы будем менять местами элементы из начала и конца так, чтобы в начале были все меньшие числа, чем в конце в этой строчке `array[perm_i], array[perm_j] = array[perm_j], array[perm_i]` При этом будем добавлять в список perms каждую перестановку

В функции task1() будем считывать, обрабатывать и выводить данные.

```
Input:
5
5 4 3 2 1
Output:
2
0 4
1 3
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	`4.689997876994312e-05 s`	`0.00016021728515625 Mb`
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: написан алгоритм, строящий min-heap пирамиду

Задача №6. Очередь с приоритетами

Реализуйте очередь с приоритетами. Ваша очередь должна поддерживать следующие операции: добавить элемент, извлечь минимальный элемент, уменьшить элемент, добавленный во время одной из операций.

```python
from lab5.utils import read_data, write_data


def data_format(data):
    """
    Форматирует данные под задачу

    Пример: ['A 1', 'A 5', 'X', 'D 2 4'] -> ['A 1', 'A 4', 'X']
    """
    data_ = []
    for i in data:
        i = i.split()
        if i[0] == 'A':
            data_.append(f'A {i[1]}')
        elif i[0] == 'D':
            data_[int(i[1]) - 1] = f'A {i[2]}'
        elif i[0] == 'X':
            data_.append('X')
    return data_



class Queue:
    def __init__(self, queue: list):
        self.queue = queue

    def get_queue(self):
        """
        Возвращает очередь
        """
        return self.queue

    def queue_take(self):
        """
        Берёт элемент из очереди, при этом удаляя его

        Пример: [1, 2, 3] -> 3 и [1, 2]
        """
        last_element = self.queue[-1]
        self.queue = self.queue[:-1]
        return last_element

    def queue_add(self, number):
        """
        Добавляет элемент в очередь
        """
        self.queue.append(number)

    def get_min_element(self):
        """
        Возвращает минимальный элемент из очереди
        """
        min_el = 10**10
        # В этом цикле находим наименьшее число в очереди
        for i in range(len(self.queue)):
            el = self.queue_take()
            min_el = min(min_el, int(el))
            self.queue_add(el)

        # В этом цикле берём наименьшее число из очереди
        for i in range(len(self.queue)):
            el = self.queue_take()
            if int(el) == min_el:
                return el
```

```
def main(commands_raw):
    """
    Главная функция, которая выполняет задачу, получает данные, форматирует
и возвращает ответ

    Пример: ['A 1', 'A 5', 'X', 'D 2 4', 'X'] -> ['1', '4']
    """
    commands = data_format(commands_raw)
    queue = Queue([])
    minimal_elements = []
    for i in commands:
        i = i.split()
        if i[0] == 'A':
            queue.queue_add(i[1])
        else:
            min_el = queue.get_min_element()
            if min_el:
                minimal_elements.append(min_el)
            else:
                minimal_elements.append('*')
    return minimal_elements


def task3():
    PATH_INPUT = '../txtf/input.txt'
    PATH_OUTPUT = '../txtf/output.txt'

    commands = read_data(PATH_INPUT)[1:]
    result = main(commands)

    write_data(PATH_OUTPUT, '\n'.join([i for i in result]))
    print(result)


if __name__ == "__main__":
    task3()
```

Напишем класс Queue, который будет представлен как очередь, добавим в него методы:

get_queue – получить очередь

queue_take – взять элемент из очереди

queue_add – добавить элемент в очередь

get_min_element – получение минимального элемента

В функции main будем считывать, обрабатывать и выводить данные.

```
Input: 8
A 3
A 4
A 2
X
D 2 1
X
X
X
```

```
Output:
2
1
3
*
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	`0.00020589999621734023 s`	`0.000713348388671875 mb`
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод по задаче: реализован алгоритм работы очереди, а именно основные действия (взять, добавить элемент), как и получение минимального элемента.

Вывод

Реализованы алгоритмы построения min-heap пирамиды и работы очереди.