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
К3239

Проверил:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание отчета

Оглавление

Задачи по варианту

1 задача. Куча ли?

Структуру данных «куча», или, более конкретно, «неубывающая пирамида», можно реализовать на основе массива.
Для этого должно выполнятся основное свойство неубывающей пирамиды, которое заключается в том, что для каждого $1 \le i \le n$ выполняются условия:
1. если $2i \le n$, то $a_i \le a_{2i}$,
2. если $2i + 1 \le n$, то $a_i \le a_{2i+1}$.
Дан массив целых чисел. Определите, является ли он неубывающей пирамидой.

• Формат входного файла (input.txt). Первая строка входного файла содержит целое число n ($1 \le n \le 106$). Вторая строка содержит n целых чисел, по модулю не превосходящих $2 \cdot 109$

• Формат выходного файла (output.txt). Выведите «YES», если массив является неубывающей пирамидой, и «NO» в противном случае.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

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"
```

Текстовое объяснение решения.
Проверяем, что дети имеют меньшее значение, чем родитель

Результат работы кода на примере из задачи:

```
Memory used: 14.96 МБ
Elapsed time: 0.00084 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 5

Memory used: 14.63 МБ
Elapsed time: 0.00211 sec
```

```
n = 1000000

Memory used: 33.12 МБ
Elapsed time: 0.44345 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00211 sec	14.63 Mb
Пример из задачи	0.00084 sec	14.63 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.44345 sec	33.12 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

2 задача. Высота дерева

В этой задаче ваша цель - привыкнуть к деревьям. Вам нужно будет прочитать описание дерева из входных данных, реализовать структуру данных, сохранить дерево и вычислить его высоту. • Вам дается корневое дерево. Ваша задача - вычислить и вывести его высоту. Напомним, что высота (корневого) дерева - это максимальная глубина узла или максимальное расстояние от листа до корня. Вам дано произвольное дерево, не обязательно бинарное дерево. • Формат ввода или входного файла (input.txt). Первая строка содержит число узлов n ($1 \le n \le 105$). Вторая строка содержит n целых чисел от-1 до $n-1$ – указание на родительский узел. Если i-ое значение равно -1, значит, что узел i - корневой, иначе это число является обозначением индекса родительского узла этого i-го узла ($0 \le i \le n - 1$). Индексы считать с 0. Гарантируется, что дан только один корневой узел, и что входные данные представляют дерево. • Формат вывода или выходного файла (output.txt). Выведите целое число – высоту данного дерева. • Ограничение по времени. 3 сек. • Ограничение по памяти. 512 мб.

Листинг кода

```
class TreeNode:
    def __init__(self, key):
        self.key = key
        self.parent = None
        self.children = []

    def add_child(self, child):
        child.parent = self
        self.children.append(child)



def recursive_find_height(node):
    if not node.children:
        return 1

    heights = [recursive_find_height(child) for child in node.children]
    return max(heights) + 1


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

    queue =
deque(zip(tree_dict[root_id],[root_id]*len(tree_dict[root_id])))
    checked = [0] * len(arr)

    while queue:
```

```
        node, node_root = queue.popleft()
        if checked[node] == 0:
            nodes_arr[node_root].add_child(nodes_arr[node])
            checked[node] = 1
            for item in tree_dict[node]:
                if not checked[item]:
                    queue.append((item, node))

    return recursive_find_height(nodes_arr[root_id])
```

Текстовое объяснение решения.

Создаем дерево и с помощью BFS находим высоту

Результат работы кода на примере из задачи:

```
Memory used: 15.18 МБ
Elapsed time: 0.00108 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 3

Memory used: 14.33 МБ
Elapsed time: 0.00543 sec
```

```
n = 1000000

Memory used: 36.59 МБ
Elapsed time: 0.61734 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00543 sec	14.33 Mb
Пример из задачи	0.00108 sec	15.18 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.61734 sec	36.59 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

4 задача. 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 ij.
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.

• Формат ввода или входного файла (input.txt). Первая строка содержит целое число n ($1 \le n \le 105$), вторая содержит n целых чисел ai входного массива, разделенных пробелом ($0 \le ai \le 109$, все ai - различны.)

• Формат выходного файла (output.txt). Первая строка ответа должна содержать целое число m - количество сделанных свопов. Число m должно удовлетворять условию $0 \le m \le 4n$. Следующие m строк должны содержать по 2 числа: индексы i и j сделанной перестановки двух элементов,
индексы считаются с 0. После всех перестановок в нужном порядке массив должен стать пирамидой, то есть для каждого i при $0 \le i \le n-1$ должны выполняться условия:
1. если $2i + 1 \le n - 1$, то $ai < a2i+1$,
2. если $2i + 2 \le n - 1$, то $ai < a2i+2$.
Обратите внимание, что все элементы входного массива различны. Любая последовательность свопов, которая менее 4n и после которой входной массив становится корректной пирамидой, считается верной.

• Ограничение по времени. 3 сек.

• Ограничение по памяти. 512 мб.

Листинг кода

```python
def get_swap_arr(data):
    global ans_swap
    ans_swap = []

    for i in range(len(data) // 2 - 1, -1, -1):
        recursion(i, len(data), data)

    return ans_swap


def recursion(ind, n, data):
    global 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)
```

Текстовое объяснение решения.

Рекурсивно проверяем больше ли родитель, чем дети, если да то меняем родителя и ребенка и запускаем рекурсию для ребенка

Результат работы кода на примере из задачи:

```
Memory used: 14.90 МБ
Elapsed time: 0.00101 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 5

Memory used: 14.83 МБ
Elapsed time: 0.00122 sec
```

```
n = 1000000

Memory used: 33.47 МБ
Elapsed time: 0.61734 sec
```

	Время выполнения	Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи	0.00122 sec	14.83 Mb
Пример из задачи	0.00101 sec	14.9 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.61734 sec	33.47 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости времени работы O(n), а также используемую память.

6 задача. 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. • Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 106$) - число операций с очередью. 10 Следующие n строк содержат описание операций с очередью, по одному описанию в строке. Операции могут быть следующими: – A x – требуется добавить элемент x в очередь. – X – требуется удалить из очереди минимальный элемент и вывести его в выходной файл. Если очередь пуста, в выходной файл требуется вывести звездочку «*». – D x y – требуется заменить значение элемента, добавленного в очередь операцией A в строке входного файла номер x + 1, на y. Гарантируется, что в строке x + 1 действительно находится операция A, что этот элемент не был ранее удален операцией X, и что y меньше, чем предыдущее значение этого элемента. В очередь помещаются и извлекаются только целые числа, не превышающие по модулю 109 . • Формат выходного файла (output.txt). Выведите последовательно результат выполнения всех операций X, по одному в каждой строке выходного файла. Если перед очередной операцией X очередь пуста, выведите вместо числа звездочку «*». • Ограничение по времени. 2 сек. • Ограничение по памяти. 256 мб.

Листинг кода

```
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, index = heapq.heappop(self.heap)
            self.ans_arr.append(str(value))
            del self.position[index]
        else:
            self.ans_arr.append('*')
        self.index += 1

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

    def get_ans(self):
        return self.ans_arr
```

Текстовое объяснение решения.
Использую кучу и словарь для поддерживания минимального элемента в очереди

Результат работы кода на примере из задачи:

```
Memory used: 15.21 МБ
Elapsed time: 0.00088 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n = 3

Memory used: 15.31 МБ
Elapsed time: 0.00238 sec
```

```
n = 1000000

Memory used: 37.32 МБ
Elapsed time: 0.73464 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00238 sec	15.31 Mb
Пример из задачи	0.00088 sec	15.21 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.73464 sec	37.32 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

Вывод

1. В этой лабораторной мы научились создавать деревья, пирамида или двоичная куча, очередь с приоритетами, а также еще одному виду сортировки за n log n: пирамидальной (heapsort), а также реализовывать некоторые алгоритмы для этих структур
2. Алгоритм O(n) и O(n logn) выполняется достаточно быстро, относительно квадратичной сложности, затраты памяти также прямо пропорциональны линейной.
3. С помощью методов *time.perf_counter()* и *psutil.Process().memory_info().rss* можно отслеживать ресурсозатратность алгоритмов.