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 11

Выполнил:
Тимаков Е.П. (фамилия имя)
К3141 (номер группы)

Проверила:
Артамонова В.Е.

Санкт-Петербург 2024 г.

Содержание отчета

Задачи по варианту

Задача №2. 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 ($1 \leq n \leq 105$). 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 ($0 \leq i \leq n - 1$). 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. (именно листинг, а не скрины)

```
def main():
    with open("input.txt") as f:
        n = int(f.readline())
        nums = list(map(int, f.readline().split()))
    x = max(nums)
    k = 1
    while x != -1:
        x = nums[x]
        k += 1
    with open("output.txt", "w", encoding="utf-8") as f:
        f.write(str(k))


main()
```

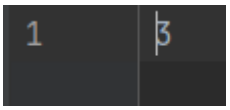
Текстовое объяснение решения:

Первая строка файла содержит число "n", а вторая строка содержит список чисел. Затем код находит максимальное значение в списке и использует его в качестве индекса для доступа к другому элементу в списке. Этот процесс продолжается, пока не будет найден элемент со значением -1.

Результат работы кода на примерах из текста задачи:

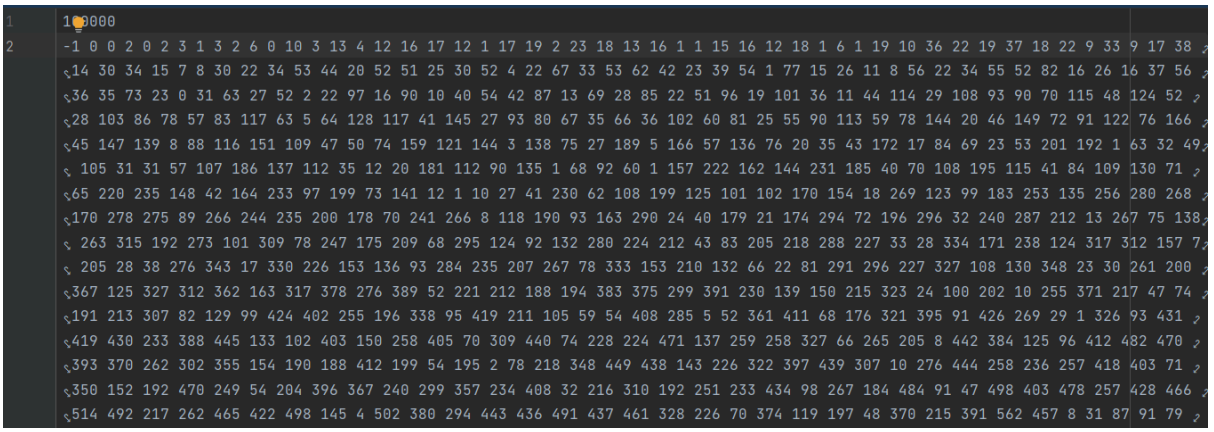
input.txt:

output.txt:

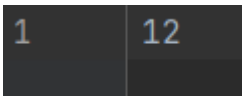


Результат работы кода на максимальных:

input.txt:

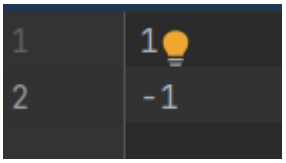


output.txt:



Результат работы кода на минимальных значениях:

input.txt:



output.txt:

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001	25
Пример из задачи	0.002	25
Верхняя граница диапазона значений входных данных из текста задачи	0.019	30

Вывод по задаче:

В данной задаче, мы ознакомились с структурой дерева. Также мы реализовали алгоритм подсчета высоты дерева.

Задача №7. Снова сортировка

Напишите программу пирамидальной сортировки на Python для последовательности в убывающем порядке. Проверьте ее, создав несколько рандомных массивов, подходящих под параметры:

• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 105$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 109 .

• Формат выходного файла (output.txt). Одна строка выходного файла с отсортированным по невозрастанию массивом. Между любыми двумя числами должен стоять ровно один пробел.

• Для проверки можно выбрать случай, когда сортируется массив размера 103 , 104 , 105 чисел порядка 109 , отсортированных в обратном порядке; когда массив уже отсортирован в нужном порядке; когда много одинаковых элементов, всего 4-5 уникальных; средний - случайный. Сравните на данных сетах Randomized-QuickSort, MergeSort, HeapSort, InsertionSort.

Листинг кода. (именно листинг, а не скрины)

```
class Heap:
    def __init__(self, array=None):
        if array is None:
```

```
        array = []
    self.array = array
    self.length = len(array)
    self.heapify_full()

def add(self, value):
    self.array.append(value)
    self.length += 1
    if self.length > 1:
        self.heapify_full()

def pop_minimal(self):
    if self.length > 0:
        value = self.array.pop(0)
        self.length -= 1
        self.heapify_full()
        return value
    return None

def heapify(self, arr, n, i):
    left_ch = i * 2 + 1
    right_ch = i * 2 + 2
    smallest = i
    if left_ch < n and arr[smallest] > arr[left_ch]:
        smallest = left_ch
    if right_ch < n and arr[smallest] > arr[right_ch]:
        smallest = right_ch
    if smallest != i:
        arr[i], arr[smallest] = arr[smallest], arr[i]
        self.heapify(arr, n, smallest)

def heapify_full(self):
    for i in range(self.length - 1, -1, -1):
        self.heapify(self.array, self.length, i)

def heap_sort(self):
    arr = self.array[:]
    for i in range(self.length - 1, 0, -1):
        (arr[i], arr[0]) = (arr[0], arr[i])
```

```
            self.heapify(arr, i, 0)
        return arr

    def change(self, value_to_change, value_to_change_to):
        if value_to_change in self.array:
            to_change = self.array.index(value_to_change)
            self.array[to_change] = value_to_change_to
            self.heapify_full()


def main():
    with open("input.txt") as f:
        _ = f.readline()
        nums = list(map(int, f.readline().split()))
    heap = Heap(nums)
    sorted_nums = heap.heap_sort()

    with open("output.txt", "w", encoding="utf-8") as f:
        f.write(" ".join(map(str, sorted_nums)))


main()
```

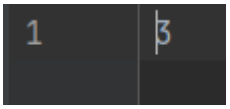
Текстовое объяснение решения:

Данный код реализует класс Heap (куча), который представляет структуру данных для хранения элементов с возможностью эффективного извлечения наименьшего элемента и сортировки. В классе реализованы методы для инициализации кучи, добавления элемента, извлечения минимального элемента, сортировки кучи, изменения значения элемента в куче.

Функция main читает входные данные из файла "input.txt", создает объект класса Heap, сортирует числа с помощью метода heap_sort.

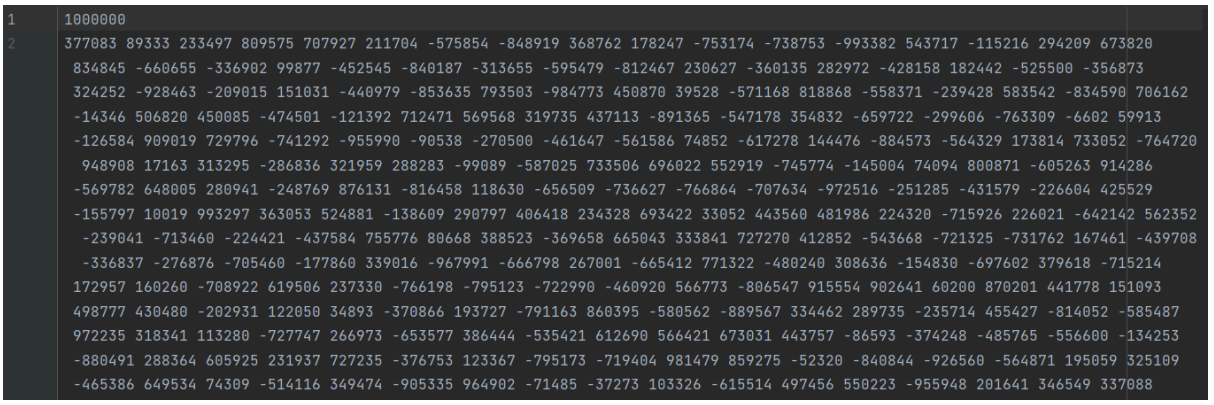
Результат работы кода на примерах из текста задачи:
input.txt:

output.txt:

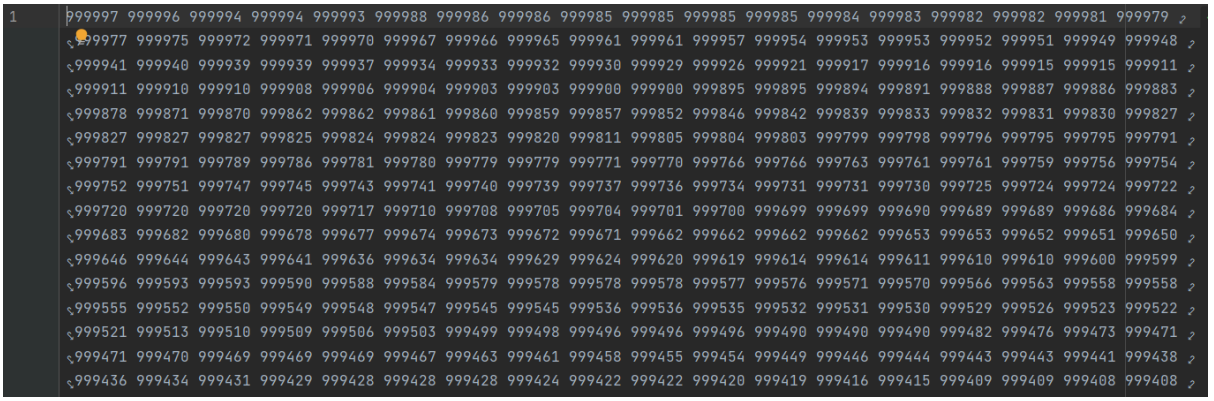


Результат работы кода на максимальных:

input.txt:

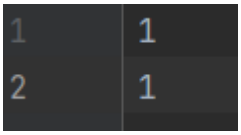


output.txt:



Результат работы кода на минимальных значениях:

input.txt:



output.txt:

	Время выполнения	Затраты памяти
Нижняя граница	0.002	25

диапазона значений входных данных из текста задачи		
Верхняя граница диапазона значений входных данных из текста задачи	0.019	30

Вывод по задаче:

В данной задаче мы изучили и реализовали алгоритм сортировки кучей.

Вывод

В данной лабораторной работе мы изучили структуру деревьев и сортировку кучей.