

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 8

Выполнил:  
Макунина А.А.  
К32421

Проверила:  
Артамонова В.Е.

Санкт-Петербург  
2022 г.

## Содержание отчета

Задачи по варианту	3
Задача №2. Высота дерева	3
Задача №4. Построение пирамиды	5
Вывод	9

## 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  $n$  ( $1 \leq n \leq 10^5$ ). 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.

На вход получаем количество узлов, а далее ссылки на родительские узлы. Создаём массив из отрицательных элементов, в конце добавляем нуль. При подсчёте узлов мы заменяем каждое отрицательное значение на высоту и ищем среди них максимальное.

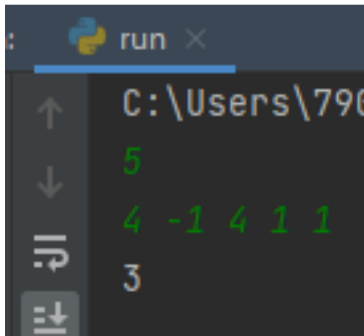
```
def height(parents):  
    depth = [-1] * len(parents) + [0]  
  
    def count_depth(i):  
        if depth[i] == -1:  
            depth[i] = count_depth(parents[i]) + 1  
        return depth[i]  
  
    return max(count_depth(i) for i in  
range(len(parents)))
```

```

n = input()
parents = [int(i) for i in input().split()]
t_start = time.perf_counter()
tracemalloc.start()
print(height(parents))

```

Результат работы кода на примере из текста задачи:

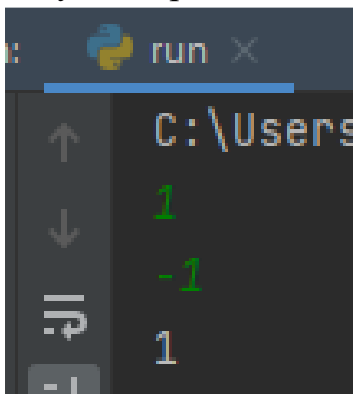


```

C:\Users\79006>python run.py
5
4 -1 4 1 1
3
3

```

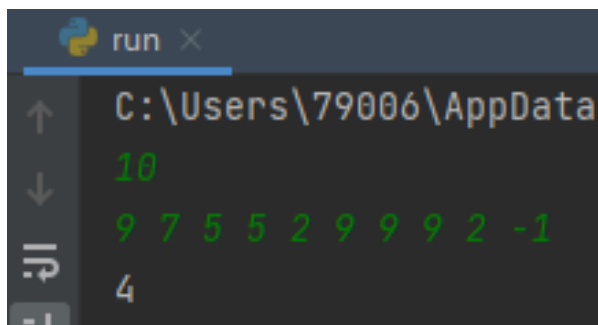
Результат работы кода максимальном и минимальном значениях:



```

C:\Users\79006>python run.py
1
-1
1
1

```



```

C:\Users\79006\AppData\Local>python run.py
10
9 7 5 5 2 9 9 9 2 -1
4
4

```

	Время выполнения, с	Максимальные затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.00015770000000037143	0.0018444061279296875
Пример из задачи	0.00022910000000031516	0.0027790069580078125
Верхняя граница диапазона значений входных данных из текста задачи	0.000181900000000117834	0.0023651123046875

Вывод по задаче: ура, мы распределили детей и посчитали глубину семейной кучи!

#### 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  $a_i$  и  $a_j$  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.

- **Формат ввода или входного файла (input.txt).** Первая строка содержит целое число  $n$  ( $1 \leq n \leq 10^5$ ), вторая содержит  $n$  целых чисел  $a_i$  входного массива, разделенных пробелом ( $0 \leq a_i \leq 10^9$ , все  $a_i$  - различны.)
- **Формат выходного файла (output.txt).** Первая строка ответа должна содержать целое число  $m$  - количество сделанных свопов. Число  $m$  должно удовлетворять условию  $0 \leq m \leq 4n$ . 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  $0 \leq i \leq n-1$  должны выполняться условия:

1. если  $2i + 1 \leq n - 1$ , то  $a_i < a_{2i+1}$ ,
2. если  $2i + 2 \leq n - 1$ , то  $a_i < a_{2i+2}$ .

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.

Две функции: сортировка кучи по наименьшему элементу и построение кучи + swap() - простой способ обменять значения двух переменных, которые содержат одинаковые типы данных.

```
def swap(arr1, i, arr2, j):
    arr1[i], arr2[j] = arr1[j], arr2[i]

def min_heapify(arr, index):
    global count
    left = 2 * index + 1
    right = 2 * index + 2
    if left < len(arr) and arr[left] < arr[index]:
        min = left
    else:
        min = index
    if right < len(arr) and arr[right] < arr[min]:
        min = right
    if min != index:
        count += 1
        swaps.append(str(index) + ' ' + str(min))
        swap(arr, index, arr, min)
        min_heapify(arr, min)

def make_heap(arr):
    n = int(len(arr) - 1)
    for k in range(n, -1, -1):
        min_heapify(arr, k)

n = input()
l = list(map(int, input().split()))
t_start = time.perf_counter()
tracemalloc.start()
count = 0
swaps = []
make_heap(l)
print(count)
for j in swaps:
    print(j)
```

Результат работы кода на примере из текста задачи:

Результат работы кода максимальном и минимальном значениях:

	Время выполнения, с	Максимальные затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	7.160000000000605e-05	0.0085906982421875
Пример из задачи	0.00011029999999934148	0.0085906982421875

Пример из задачи	5.539999999903955e-05	0.0085906982421875
Верхняя граница диапазона значений входных данных из текста задачи	0.1409054999999988	0.2840852737426758

Вывод по задаче: намного приятнее деревьев.



## **Вывод**

Спасибо, что в этой лабораторной всего две задачи. Но в семье не без урода. Первая – фу, вторая – класс. Знакомство с деревьями оцениваю негативно, а пирамидки ничего так.