САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5 по курсу «Алгоритмы и структуры данных»

Вариант 7

Выполнила:

Пожидаева Е.Р.

Санкт-Петербург 2024 г.

Оглавление

Задача №1. Куча ли?	3
Задание №4. Построение пирамиды [N баллов]	
Задача №6. Очередь с приоритетами	
Задача №7. Снова сортировка	

Задача №1. Куча ли?

Текст задачи.

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 \leq n, то a_i \leq 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.

Листинг кода.

```
def ifheap(arr, n):
    for i in range(1, n // 2 + 1):
        if not arr[i - 1] <= arr[2 * i - 1] or not
arr[i - 1] <= arr[2 * i]:
            return False
    return True

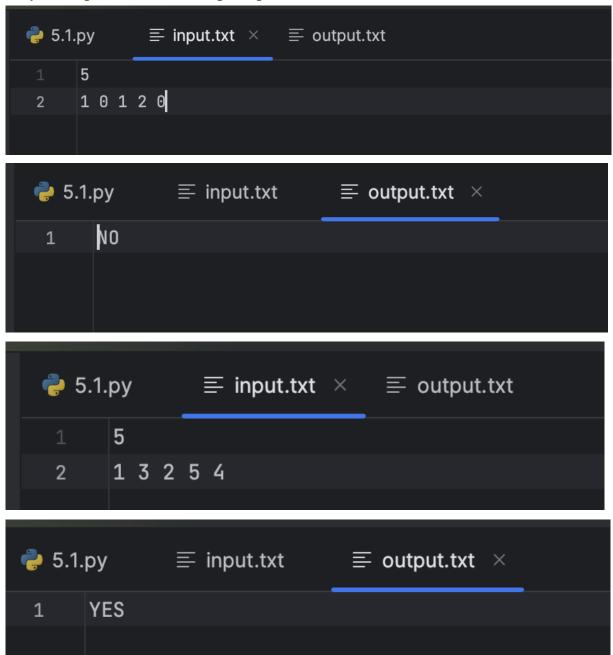
with open('input.txt') as f:
    n = int(f.readline())
    arr = [int(i) for i in f.readline().split()]

with open('output.txt', 'w') as f:
    if ifheap(arr, n):
        f.write('YES')
    else:
        f.write('NO')</pre>
```

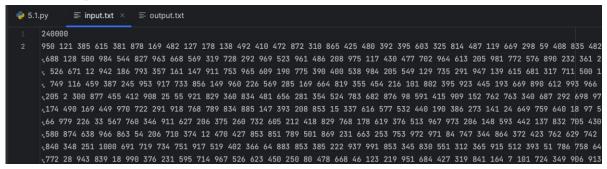
Текстовое объяснение решения.

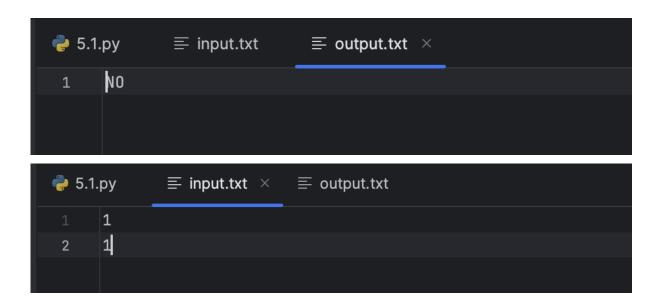
Здесь мы проверяем, является ли данный массив неубывающей пирамидой. Функция ifheap принимает на вход массив arr и его размер n. Затем она проходит по всем узлам дерева (т.е. по индексам от 1 до n//2) и проверяет, что значение текущего узла меньше или равно значению его потомков (если они есть). Если это не так, то функция возвращает False. Если все узлы удовлетворяют условию кучи, функция возвращает True.

Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных и минимальных значениях





	Время выполнения (сек)	Затраты памяти (Мб)
Нижняя граница диапазона значений входных данных из текста задачи	0.00152969360351562 5	6131712
Пример из задачи	0.00052380561828613 28	6127616
Пример из задачи	0.06354069709777832	6846656
Верхняя граница диапазона значений входных данных из текста задачи	0.14455318450927734	5984256

Вывод по задаче: Я научилась работать с неубывающей пирамидой на основе массива.

Задание №4. Построение пирамиды [N баллов]

В этой задаче вы преобразуете массив целых чисел в пирамиду. Это важней- ший шаг алгоритма сортировки под названием HeapSort. Гарантированное время работы в худшем случае составляет O(nlogn), в отличие от среднего време- ни работы QuickSort, равного O(nlogn). QuickSort обычно используется на практике, потому что обычно он быстрее, но HeapSort используется для внеш- ней 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 аі и ај массива меняются местами для некоторых і и ј. Вам нужно будет преобразовать массив в пирамиду, используя только O(n) пе- рестановок. Обратите внимание, что в этой задаче вам нужно будет использовать min-heap вместо max-heap.

Листинг кода:

```
import math

def min_heapify(A, i, heap_size):
    global c
    global h
    i = i + 1
        l = 2 * i
        r = 2 * i + 1
    if _l <= heap_size and A[_l - 1] < A[i - 1]:
        smallest = _l
    else:
        smallest = i
    if r <= heap_size and A[r - 1] < A[smallest - 1]:
        smallest = r
    if smallest != i:
        A[i - 1], A[smallest - 1] = A[smallest - 1],
A[i - 1]</pre>
```

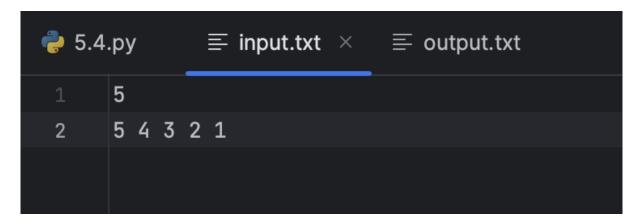
```
c += 1
      h.append(str(i - 1) + ' ' + str(smallest -
1))
      min heapify(A, smallest - 1, heap size)
def build min heap(A): # создание неубывающей
  heap size = len(A)
  global c
  for i in range (math.ceil (heap size // 2), -1,
-1):
      min heapify(A, i, heap size)
  return c
f = open('input.txt')
n = int(f.readline())
s = f.readline()
h = []
f.close()
f = open('output.txt', 'w')
C = 0
build min heap(arr)
f.write(str(c) + '\n')
for line in h:
  f.write(line + '\n')
f.close()
```

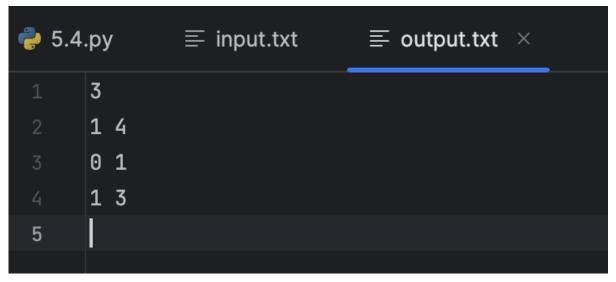
Текстовое объяснение решения

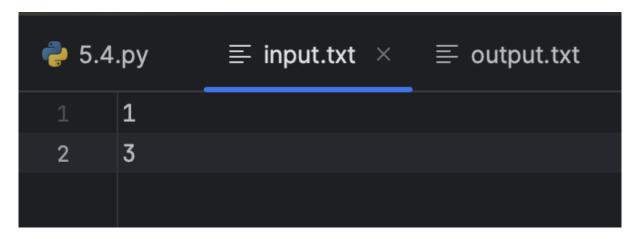
Файл открывается, из него считывается первая строка, содержащая число чисел в массиве, записывается в п. Далее в s записывается строка, содержащая элементы массива и элементы этой строки записываются в массив arr. Создается пустой массив h, Открывается файл output для записи. Функция build_min_heap вызывает функцию min_heapify для каждого внутреннего узла. Эта функция в свою очередь проверяет является ли узел наименьшим среди себя и двух потомков. Если не является, то узел меняется местами с наименьшим потомком. Цикл проходит снизу вверх. В глобальной

переменной с ведется подсчет перестановок, а в h записываются индексы элементов, которые программа поменяла местами. Все эти данные записываются в файл output.

Результат работы кода на примерах







? 5.4	l.py	≡ input.txt	≡ output.txt ×
1	0		
2	1		

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.002939238921803929	567849
Пример 1	0.00726172687372838	583921

Вывод: Благодаря этой задаче я узнала, как работать с кучами и создавать их из списков.

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.

Листинг кода

```
import math
def max heapify(A, i):
  if l \le len(A) and A[l-1] > A[i-1]:
       largest = 1
  else:
       largest = i
  if r \le len(A) and A[r - 1] > A[largest - 1]:
      largest = r
       if largest != i:
          A[i - 1], A[largest - 1] = A[largest - 1]
1], A[i - 1]
           max heapify(A, largest - 1)
def build max heap(A):
   for i in range (math.ceil(len(A) // 2), -1, -1):
       max heapify(A, i)
def min heapify(A, i):
  if l \le len(A) and A[l-1] \le A[i-1]:
       smallest = 1
   else:
       smallest = i
   if r \le len(A) and A[r - 1] < A[smallest - 1]:
       smallest = r
   if smallest != i:
```

```
A[i - 1], A[smallest - 1] = A[smallest - 1],

A[i - 1]

min_heapify(A, smallest - 1)
```

```
def build min heap(A):
   for i in range (math.ceil(len(A) // 2), -1, -1):
       min heapify(A, i)
def heap increase key(A, i, key):
   parent = math.ceil(i // 2)
   A[i - 1] = key
   while i > 1 and A[parent - 1] < A[i - 1]:
       A[i-1], A[parent-1] = (A[parent-1], A[i-1]
- 1])
       i = parent
def heap insert(A, key):
   A.append(None)
   heap increase key(A, len(A) - 1, key)
def heap extract min(A):
  if len(A) < 1:
   build min heap(A)
   min = A[0]
   A.pop(0)
   build max heap(A)
   return min
def heap decrease key(A, i, key):
   A[i] = key
   max heapify(A, i)
f i = open('input.txt')
n = int(f i.readline())
f o = open('output.txt', 'w')
A = list()
```

```
for j in range(n):
   S = list(f i.readline().split())
   if S[0] == 'A':
       heap insert (A, int(S[1]))
   if S[0] == 'X':
       f o.write(str(heap extract min(A)))
       f o.write('''
   if S[0] == 'D':
       f i.seek(0)
       SD =
list(f i.readlines()[int(S[1])].split())
       SD[1] = int(SD[1])
       i = A.index(S D[1])
       heap decrease key(A, i, int(S[2]))
       f i.seek(0)
           f i.readline()
f i.close()
f o.close()
```

Текстовое объяснение решения:

Этот код реализует различные функции для работы с кучей. Импортируется модуль math, который предоставляет математические функции для работы с числами. В данном случае, используется функция ceil для округления числа вверх. Затем идут определения функций:

- max_heapify: функция для поддержания свойства кучи в виде макс-кучи (значение каждого узла больше или равно значения его потомков). Исходный узел передается в виде индекса і в списке А.
- build_max_heap: функция для построения макс-кучи из списка А.
- min_heapify: функция для поддержания свойства кучи в виде мин-кучи (значение каждого узла меньше или равно значения его потомков). Исходный узел передается в виде индекса і в списке А.
- heap_insert: функция для вставки нового значения key в кучу A.
- heap_extract_min: функция для извлечения минимального значения из кучи A.
- heap_decrease_key: функция для уменьшения значения элемента кучи A по индексу i.

Далее открывается файл 'input.txt' для чтения и файл 'output.txt' для записи. Переменная п считывается из первой строки файла 'input.txt' и используется для определения количества операций с кучей. Затем создается пустой список А, в который будут добавляться значения элементов кучи.Запускается цикл для выполнения операций с кучей. В каждой итерации считывается строка из файла 'input.txt' и разделяется на отдельные элементы. Если первый элемент строки равен 'A', то вызывается функция heap_insert и в кучу добавляется новое значение, преобразованное в целое число.

Если первый элемент строки равен 'X', то вызывается функция heap_extract_min и минимальное значение из кучи записывается в файл 'output.txt'. Если первый элемент строки равен 'D', то выполняется следующая последовательность действий:

- 1. Считывается строка из файла 'input.txt' по индексу, переданному вторым элементом текущей строки.
- 2. В строке разделяются элементы и второй элемент преобразуется в целое число.
- 3. Индекс элемента с таким значением находится в списке А.
- 4. Функция heap_decrease_key вызывается для уменьшения значения этого элемента в куче A до значения, переданного третьим элементом строки.
- 5. Чтобы корректно прочитать следующие строки из файла, позиция в файле 'input.txt' сбрасывается в начало до строки с индексом j+2.

После завершения цикла производится закрытие файлов 'input.txt' и 'output.txt'.

Результат работы кода на примерах из текста задачи:

≡ inp	ut.txt ×	≡ output.txt	🥏 5.6.py
1	8		
2	A 3		
3	A 4		
4	A 2		
5	X		
6	D 2 1		
7	X		
8	Х		
9	X		

≣ inp	ut.txt	≡ output.txt ×	🥏 5.6.py
1	2		
2	1		
3	3		
4	*		
5			

≣ inpu	ut.txt ×	≡ output.txt	🥏 5.6.py
1	2		
2	A 1		
3	x		
≣ inp	ut.txt	≡ output.txt ⇒	< 🥏 5.6.py
≡ inp	ut.txt 1	≡ output.txt >	× 🕏 5.6.py
		≡ output.txt >	< 🕏 5.6.py

	Время выполнения (сек)	Затраты памяти (Мб)
Нижняя граница диапазона значений входных данных из текста задачи	0.0034561	6505585
Пример из задачи	0.0035677	6738291

Вывод по задаче: в ходе решения этой задачи я познакомилась с реализацией очереди с приоритетами.

Задача №7. Снова сортировка

Текст задачи.

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

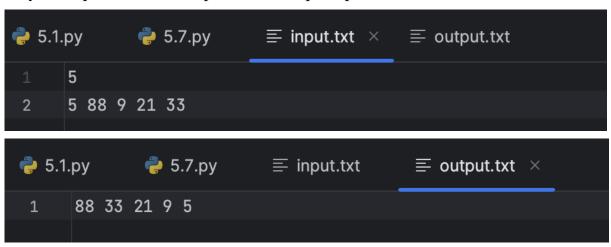
Листинг кода:

```
def heap(arr, n, i):
   largest = i
   if l < n and arr[l] < arr[largest]:</pre>
       largest = 1
   if r < n and arr[r] < arr[largest]:</pre>
       largest = r
   if largest != i:
       arr[i], arr[largest] =
arr[largest], arr[i]
       heap(arr, n, largest)
def heapsort(arr):
   n = len(arr)
   for i in range (n // 2 - 1, - 1, - 1):
       heap(arr, n, i)
   for i in range (n - 1, 0, - 1):
       arr[i], arr[0] = arr[0], arr[i]
       heap(arr, i, 0)
f = open('input.txt', 'r')
n = int(f.readline())
s = list(map(int, f.readline().split()))
f.close()
heapsort(s)
f = open('output.txt', 'w')
f.write(' '.join(list(map(str, s))))
```

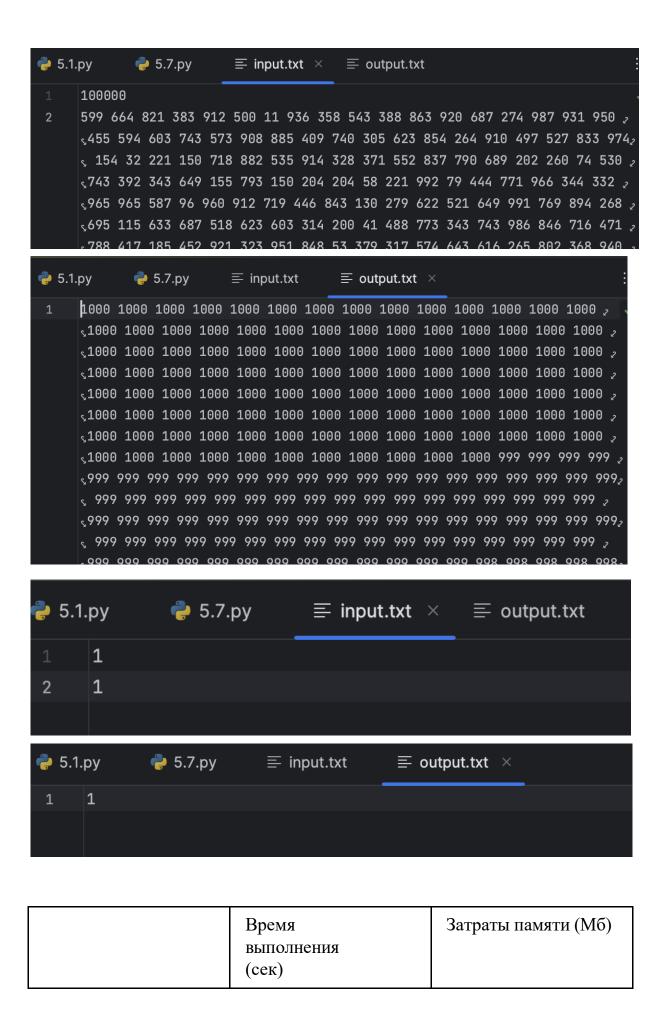
Текстовое объяснение решения.

Определяется функция heap, которая принимает на вход массив агг, его размер п и индекс і. Функция heap проверяет, является ли элемент с индексом і корнем поддерева неубывающей пирамидой. Если нет, то функция находит индекс наименьшего из дочерних элементов и меняет местами значения элементов с индексами і и largest. Затем функция вызывает себя рекурсивно для поддерева с корнем в largest. Затем определяется функция heapsort, которая принимает на вход массив агг. Функция heapsort сначала определяет размер массива п и вызывает функцию heap для каждого узла дерева (т.е. для индексов от n//2 - 1 до 0). Затем функция heapsort проходит по всем элементам массива агг, начиная с конца, и меняет местами первый (т.е. наименьший) элемент с текущим последним элементом. Затем функция вызывает функцию heap для поддерева с корнем в 0 и размером і (т.е. без последнего элемента).

Результат работы кода на рандомном примере



Результат работы кода на максимальных и минимальных значениях



Нижняя граница диапазона значений входных данных из текста задачи	0.00083136558532714 84	6057984
Рандомный пример	0.00067496299743652 34	6180864
Верхняя граница диапазона значений входных данных из текста задачи	0.000474789220098765 44	6082560

Вывод по задаче:

Я поработала с пирамидальной сортировкой