

**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]	6
Задача №6. Очередь с приоритетами.....	10
Задача №7. Снова сортировка.....	17

Задача №1. Куча ли?

Текст задачи.

Структуру данных «куча», или, более конкретно, «неубывающая пирамида», можно реализовать на основе массива.

Для этого должно выполняться основное свойство неубывающей пирамиды, которое заключается в том, что для каждого $1 \leq i \leq n$ выполняются условия:

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

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')
```

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

```
5.1.py  input.txt  output.txt
1      5
2      1 0 1 2 0
```

```
5.1.py  input.txt  output.txt
1      No
```

```
5.1.py  input.txt  output.txt
1      5
2      1 3 2 5 4
```

```
5.1.py  input.txt  output.txt
1      YES
```

Результат работы кода на максимальных и минимальных значениях

```
5.1.py  input.txt  output.txt
1      240000
2      950 121 385 615 381 878 169 482 127 178 138 492 410 472 872 310 865 425 480 392 395 603 325 814 487 119 669 298 59 408 835 482
    688 128 500 984 544 827 963 668 569 319 728 292 969 523 961 486 208 975 117 430 477 702 964 613 205 981 772 576 890 232 361 2
    526 671 12 942 186 793 357 161 147 911 753 965 609 190 775 390 400 538 984 205 549 129 735 291 947 139 615 681 317 711 500 1
    749 116 459 387 245 953 917 733 856 149 960 226 569 285 169 664 819 355 454 216 101 802 395 923 445 193 669 890 612 993 966
    205 2 300 877 455 412 908 25 55 921 829 360 834 481 656 281 354 524 783 682 876 98 591 415 909 152 762 763 340 687 292 698 97
    174 490 169 449 970 722 291 918 768 789 834 885 147 393 208 853 15 337 616 577 532 440 190 386 273 141 24 649 759 640 18 97 5
    66 979 226 33 567 760 346 911 627 206 375 260 732 605 212 418 829 768 178 619 376 513 967 973 206 148 593 442 137 832 705 430
    580 874 638 966 863 54 206 710 374 12 470 427 853 851 789 501 869 231 663 253 753 972 971 84 747 344 864 372 423 762 629 742
    840 348 251 1000 691 719 734 751 917 519 402 366 64 883 853 385 222 937 991 853 345 830 551 312 365 915 512 393 51 786 758 64
    772 28 943 839 18 990 376 231 595 714 967 526 623 450 250 80 478 668 46 123 219 951 684 427 319 841 164 7 101 724 349 906 913
```

```

5.1.py  input.txt  output.txt x
1  10

```

```

5.1.py  input.txt x  output.txt
1  1
2  1

```

	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 $O(n \log n)$, в отличие от среднего време- ни работы QuickSort, равного $O(n \log n)$. 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 мас- сив 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 $O(n)$ 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]
```

```

        c += 1
        h.append(str(i - 1) + ' ' + str(smallest -
1))
        min_heapify(A, smallest - 1, heap_size)

def build_min_heap(A): # создание неубывающей
пирамиды
    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()
arr = ([int(i) for i in s.split()])
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 `n`. 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 `s` 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.py		input.txt	output.txt
1	5		
2	5 4 3 2 1		

5.4.py		input.txt	output.txt
1	3		
2	1 4		
3	0 1		
4	1 3		
5			


```

5.4.py
input.txt
output.txt

1 1
2 3

```

```

5.4.py
input.txt
output.txt

1 0
2

```

	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):
    i = i + 1
    _l = 2 * i
    r = 2 * i + 1
    if _l <= len(A) and A[_l - 1] > A[i - 1]:
        largest = _l
    else:
        largest = i
    if r <= len(A) and A[r - 1] > A[largest - 1]:
        largest = r
    if largest != i:
        A[i - 1], A[largest - 1] = A[largest - 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):
    i = i + 1
    _l = 2 * i
    r = 2 * i + 1
    if _l <= len(A) and A[_l - 1] < A[i - 1]:
        smallest = _l
    else:
        smallest = i
    if r <= 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):
    i = i + 1
    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])
        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:
        return '*'
    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)
        S_D =
list(f_i.readlines()[int(S[1])].split())
        S_D[1] = int(S_D[1])
        i = A.index(S_D[1])
        heap_decrease_key(A, i, int(S[2]))
        f_i.seek(0)
        for k in range(j + 2):
            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 `i` в списке `A`.
- `build_max_heap`: функция для построения макс-кучи из списка `A`.
- `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 `i` в списке `A`.
- `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 n считывается из первой строки файла 'input.txt' и используется для определения количества операций с кучей. Затем создается пустой список A , 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 A .
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:

input.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	X				
9	X				

input.txt		output.txt ×		5.6.py	
1	2				
2	1				
3	3				
4	*				
5					

≡

input.txt

×

≡

output.txt

×

Python 5.6.py

1	2
2	A 1
3	x

≡

input.txt

×

≡

output.txt

×

Python 5.6.py

1	1
2	

	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 случайных массивов, подходящих под параметры:

Листинг кода:

```
def heap(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and arr[l] < arr[largest]:
        largest = l
    if r < n and arr[r] < arr[largest]:
        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 `hear`, которая принимает на вход массив `arr`, его размер `n` и индекс `i`. Функция `hear` проверяет, является ли элемент с индексом `i` корнем поддерева неубывающей пирамидой. Если нет, то функция находит индекс наименьшего из дочерних элементов и меняет местами значения элементов с индексами `i` и `largest`. Затем функция вызывает себя рекурсивно для поддерева с корнем в `largest`. Затем определяется функция `heapsort`, которая принимает на вход массив `arr`. Функция `heapsort` сначала определяет размер массива `n` и вызывает функцию `hear` для каждого узла дерева (т.е. для индексов от $n/2 - 1$ до 0). Затем функция `heapsort` проходит по всем элементам массива `arr`, начиная с конца, и меняет местами первый (т.е. наименьший) элемент с текущим последним элементом. Затем функция вызывает функцию `hear` для поддерева с корнем в 0 и размером `i` (т.е. без последнего элемента).

Результат работы кода на случайном примере

 5.1.py	 5.7.py	 input.txt ×	 output.txt
1	5		
2	5 88 9 21 33		

 5.1.py	 5.7.py	 input.txt	 output.txt ×
1	88 33 21 9 5		

Результат работы кода на максимальных и минимальных значениях

```
5.1.py 5.7.py input.txt x output.txt
1 100000
2 599 664 821 383 912 500 11 936 358 543 388 863 920 687 274 987 931 950
   455 594 603 743 573 908 885 409 740 305 623 854 264 910 497 527 833 974
   154 32 221 150 718 882 535 914 328 371 552 837 790 689 202 260 74 530
   743 392 343 649 155 793 150 204 204 58 221 992 79 444 771 966 344 332
   965 965 587 96 960 912 719 446 843 130 279 622 521 649 991 769 894 268
   695 115 633 687 518 623 603 314 200 41 488 773 343 743 986 846 716 471
   788 417 185 452 921 323 951 848 53 379 317 574 643 616 265 802 368 940
```

```
5.1.py 5.7.py input.txt output.txt x
1 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
   1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
   1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
   1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
   1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
   1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
   1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
   1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 999 999 999 999
   999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
   999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
   999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
   999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
```

```
5.1.py 5.7.py input.txt x output.txt
1 1
2 1
```

```
5.1.py 5.7.py input.txt output.txt x
1 1
```

	Время выполнения (сек)	Затраты памяти (Мб)
--	------------------------------	---------------------

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