

Национальный исследовательский Университет ИТМО
Мегафакультет информационных и трансляционных технологий
Факультет инфокоммуникационных технологий

Алгоритмы и структуры данных

Лабораторная работа №3

Работу

выполнил:

И.В. Гуторова

Группа: К3141

Преподаватель:

В.Е. Артамонова

Санкт-Петербург
2022

Содержание

1. Задачи по варианту	3
1.1. Улучшение Quick sort	3
1.2. Индекс Хирша	6
2. Дополнительные задачи	8
2.1. Сортировка пугалом	8
Вывод	11
Список использованных источников	12

1. Задачи по варианту

1.1. Улучшение Quick sort

1. Используя псевдокод процедуры Randomized - QuickSort, а так же Partition из презентации к Лекции 3 (страницы 8 и 12), напишите программу быстрой сортировки на Python и проверьте ее, создав несколько рандомных массивов, подходящих под параметры:

- Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 10^4$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .
- Формат выходного файла (output.txt). Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел
- Ограничение по времени: 2 сек.
- Ограничение по памяти: 256 мб.
- Для проверки можно выбрать наихудший случай, когда сортируется массив размера 10^3 , 10^4 , 10^5 чисел порядка 10^9 , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний - случайный. Сравните на данных сетах Randomized-QuickSort и простой QuickSort. (А также есть Tail-Recursive-QuickSort [1])

2. Основное задание. Цель задачи - переделать данную реализацию рандомизированного алгоритма быстрой сортировки, чтобы она работала быстро даже с последовательностями, содержащими много одинаковых элементов. Чтобы заставить алгоритм быстрой сортировки эффективно обрабатывать последовательности с несколькими уникальными элементами, нужно заменить двухстороннее разделение на трехстороннее [2]. То есть ваша новая процедура разделения должна разбить массив на три части:

- $A[k] < x$ для всех $l + 1 \leq k \leq m_1 - 1$
- $A[k] = x$ для всех $m_1 \leq k \leq m_2$
- $A[k] > x$ для всех $m_2 + 1 \leq k \leq r$
- Формат входного и выходного файла аналогичен п.1.
- Аналогично п.1 этого задания сравните Randomized-QuickSort+ с Partition и ее с Partition3 на сетах случайных данных, в которых содержатся всего несколько уникальных элементов при $n = 10^3, 10^4, 10^5$. Что быстрее, Randomized-QuickSort+ с Partition3 или Merge-Sort?

Таблица 1.1

Пример

input.txt	output.txt
5 2 3 9 2 2	2 2 2 3 9

```

1  fin = open('input.txt')
2  n = int(fin.readline())
3  num = fin.readline().split()
4  fout = open('output.txt', 'w')
5  def quick_sort(nums, start, end):
6      if start < end - 1:
7          m = partition(nums, start, end)
8          quick_sort(nums, start, m)
9          quick_sort(nums, m + 1, end)
10 def randomized_quick_sort(nums, start, end):
11     if start < end - 1:
12         k = randint(start, end - 1)
13         nums[k], nums[start] = nums[start], nums[k]
14         m = partition(nums, start, end)
15         randomized_quick_sort(nums, start, m)
16         randomized_quick_sort(nums, m + 1, end)
17 def partition(nums, start, end):
18     x = nums[start]
19     j = start
20     for i in range(start + 1, end):
21         if nums[i] ≤ x:
22             j = j + 1
23             nums[j], nums[i] = nums[i], nums[j]
24     nums[j], nums[start] = nums[start], nums[j]
25     return j
26 def quick_sort3(nums, start, end):
27     if start < end - 1:
28         m1, m2 = partition3(nums, start, end)
29         quick_sort3(nums, start, m1)
30         quick_sort3(nums, m2 + 1, end)
31 def randomized_quick_sort3(nums, start, end):
32     if start < end - 1:
33         k = randint(start, end - 1)
34         nums[k], nums[start] = nums[start], nums[k]
35         m1, m2 = partition3(nums, start, end)
36         randomized_quick_sort3(nums, start, m1)
37         randomized_quick_sort3(nums, m2 + 1, end)
38 def partition3(nums, start, end):
39     x = nums[start]
40     j = start
41     k = 0
42     for i in range(start + 1, end):
43         if nums[i] ≤ x:
44             j = j + 1
45             if nums[i] == x:
46                 k += 1
47             nums[j], nums[i] = nums[i], nums[j]
48     nums[j], nums[start] = nums[start], nums[j]
49     return j - k + 1, j
50 quick_sort(num, 0, len(num))
51 fout.write(str(num).replace('[', '').replace(']', '').replace(',', '
↵ ')).replace('"', ''))
52 fin.close()
53 fout.close()

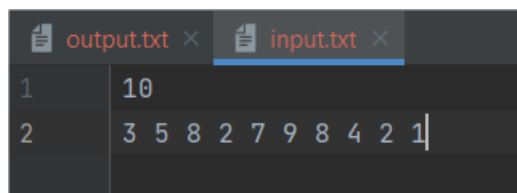
```

Листинг 1: Код первой задачи

Текстовое объяснение решения:

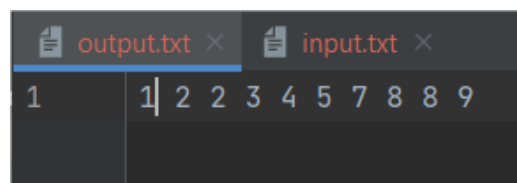
Импортируем модули для измерения времени и памяти, запускаем счетчик времени. Открываем файл `input.txt`, считываем число элементов массива, преобразовывая строку в целочисленный тип данных, считываем строку с числами, разбиваем ее по пробелам в массив. Далее у нас описаны 6 функций. `Quicksort()` принимает на вход массив чисел и индексы начала и конца сортировки+1. Если индекс начала меньше индекса конца, то есть длина сортируемой части массива больше 1, то выполняется: вызываем `partition()`, индекс числа, с которым сравнивали все, записываем в переменную, вызываем рекурсивно функцию для части массива меньше числа и для части массива больше числа. В `randomizedquicksort` все аналогично, но в начале мы выбираем случайный элемент массива и меняем его местами с первым. В `partition()` за опорное значение принимается первый элемент массива, мы проходимся по всему массиву. Если элемент меньше опорного значения, то увеличиваем счетчик таких чисел. И меняем местами этот элемент с первым, который больше опорного. В конце вне цикла меняем местами опорный элемент и последний, который меньше его. `Quicksort3()` и `randomizedquicksort3()` работают аналогично с предыдущими функциями, но принимают от `partition3()` два числа – индекс первого числа, который равен опорному и последнего. И сортировка происходит тогда уже только с начала до первого числа и от второго числа до конца. `Partition3()` высчитывает эти два числа. В целом работает аналогично `partition()`, но еще запоминает количество чисел равных опорному, благодаря которому считает индекс первого числа, равного опорному. Уже вне функций вызываем `quicksort()` для массива от начала до его длины. В дальнейшем для проведения сравнения эта строчка будет заменяться на `quicksort3()`, `randomizedquicksort()`, `randomizedquicksort3()`. Остальной код будет оставаться прежним. Открываем файл `output.txt` для записи, записываем в него отсортированный по возрастанию массив, преобразовав в строку и удалив лишние знаки. Закрываем файлы `input.txt` и `output.txt`. Выводим в консоль время работы с начала. Подсчитываем затраты памяти и также выводим их в консоль.

Результат работы кода на примере:



1	10
2	3 5 8 2 7 9 8 4 2 1

Рисунок 1.1. Пример input файла для первой задачи



1	1 2 2 3 4 5 7 8 8 9
---	---------------------

Рисунок 1.2. Результат работы кода

Вывод по задаче:

В этой задаче мы научились сортировать массив чисел по возрастанию путем быстрой сортировки и улучшили алгоритм, а также сравнили различные варианты выполнения сортировки и сравнили с сортировкой слиянием.

1.2. Индекс Хирша

Для заданного массива целых чисел `citations`, где каждое из этих чисел – число цитирований i -ой статьи ученого-исследователя, посчитайте индекс Хирша этого ученого.

По определению Индекса Хирша на Википедии: Учёный имеет индекс h , если h из его/её N_p статей цитируются как минимум h раз каждая, в то время как оставшиеся ($N_p - h$) статей цитируются не более чем h раз каждая. Иными словами, учёный с индексом h опубликовал как минимум h статей, на каждую из которых сослались как минимум h раз.

Если существует несколько возможных значений h , в качестве h -индекса принимается максимальное из них.

- Формат ввода или входного файла (`input.txt`). Одна строка `citations`, содержащая n целых чисел, по количеству статей ученого (длина `citations`), разделенных пробелом или запятой.
- Формат выхода или выходного файла (`output.txt`). Одно число - индекс Хирша (h -индекс).
- Ограничения: $1 \leq n \leq 5000$, $0 \leq citations[i] \leq 1000$.

Таблица 1.2

Пример

input.txt	output.txt
3,0,6,1,5	3

Пояснение. `citations = [3,0,6,1,5]` означает, что ученый опубликовал 5 статей в целом, и каждая из них оказалась процитирована 3, 0, 6, 1, 5 раз соответственно. Поскольку у ученого есть 3 статьи с минимум тремя цитированиями, а у оставшихся двух - не более 3 цитирований, его индекс Хирша равен 3.

Таблица 1.3

Пример

input.txt	output.txt
1,3,1	1

- Ограничений по времени (и памяти) не предусмотрено, проверьте максимальный случай при заданных ограничениях на данные, и оцените асимптотическое время.
- Подумайте, если бы массив `citations` был бы изначально отсортирован по возрастанию, можно было бы еще ускорить алгоритм?

```
1 fin = open('input.txt')
2 temp = fin.readline().split(',')
3 num = []
4 for i in range(len(temp)):
5     num.append(int(temp[i]))
6 fout = open('output.txt', 'w')
7
```

```

8  def randomized_quick_sort3(nums, start, end):
9      if start < end - 1:
10         k = randint(start, end - 1)
11         nums[k], nums[start] = nums[start], nums[k]
12         m1, m2 = partition3(nums, start, end, nums[start])
13         randomized_quick_sort3(nums, start, m1)
14         randomized_quick_sort3(nums, m2 + 1, end)
15
16  def partition3(nums, start, end, x):
17      j = start
18      k = 0
19      for i in range(start + 1, end):
20         if nums[i] ≤ x:
21             j = j + 1
22             if nums[i] == x:
23                 k += 1
24             nums[j], nums[i] = nums[i], nums[j]
25      nums[j], nums[start] = nums[start], nums[j]
26      return j - k + 1, j
27
28  randomized_quick_sort3(num, 0, len(num))
29  hirsh = 0
30  for h in range(num[-1]):
31      m1, m2 = partition3(num, 0, len(num), h)
32      if len(num) - m1 ≥ h:
33          hirsh = h
34      else:
35          break
36  fout.write(str(hirsh))
37  fin.close()
38  fout.close()

```

Листинг 2: Код второй задачи

Текстовое объяснение решения:

Импортируем модули для измерения времени и памяти, запускаем счетчик времени. Открываем файл input.txt, считываем число элементов массива, преобразовывая строку в целочисленный тип данных, считываем строку с числами, разбиваем ее по запятым в массив. Используем функции из первой задачи, чтобы отсортировать массив. Также немного изменим potition(). Теперь она будет принимать на вход еще и опорный элемент. Сортируем массив. Создаем переменную для индекса, изначально равную 0. В цикле проходим по всем числам от 0 до максимального в массиве. Вызываем partition() для этого числа, таким образом находим индекс первого числа, большего рассматриваемого. Если количество чисел, которые больше данного числа, больше данного числа, приравниваем индекс Хирша к нему. Если нет, выходим из цикла, дальше ничего быть не может. Открываем файл output.txt для записи, записываем в него индекс Хирша. Закрываем файлы input.txt и output.txt. Выводим в консоль время работы с начала. Подсчитываем затраты памяти и также выводим их в консоль.

Результат работы кода на примере:

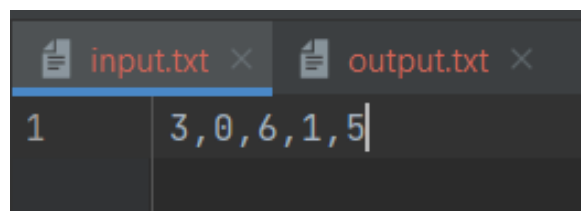


Рисунок 1.3. Пример input файла

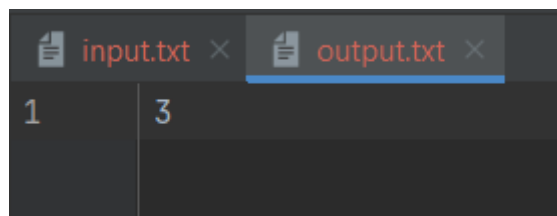


Рисунок 1.4. Результат работы кода

Вывод по задаче:

В этой задаче мы научились высчитывать индекс Хирша и применили быструю сортировку на практике.

2. Дополнительные задачи

2.1. Сортировка пугалом

«Сортировка пугалом» — это давно забытая народная потешка. Участнику под верхнюю одежду продевают деревянную палку, так что у него оказываются растопырены руки, как у огородного пугала. Перед ним ставятся n матрёшек в ряд. Из-за палки единственное, что он может сделать — это взять в руки две матрёшки на расстоянии k друг от друга (то есть i -ую и $i + k$ -ую), развернуться и поставить их обратно в ряд, таким образом поменяв их местами. Задача участника — расположить матрёшки по неубыванию размера. Может ли он это сделать?

- Формат входного файла (input.txt). В первой строчке содержатся числа n и k ($1 \leq n, k \leq 10^5$) — число матрёшек и размах рук. Во второй строчке содержится n целых чисел, которые по модулю не превосходят 10^9 — размеры матрёшек.
- Формат выходного файла (output.txt). Выведите «ДА», если возможно отсортировать матрёшки по неубыванию размера, и «НЕТ» в противном случае.
- Ограничение по времени: 2 сек.
- Ограничение по памяти: 256 мб.

Таблица 2.1

Пример

input.txt	output.txt
3 2 2 1 3	НЕТ
5 3 1 5 3 4 1	ДА


```

1
2 def scarecrow_sort(num, k):
3     for i in range(0, len(num) - k):
4         if num[i] > num[i + k]:
5             num[i], num[i + k] = num[i + k], num[i]
6
7 fin = open('input.txt')
8 n, k = fin.readline().split()
9 temp = fin.readline().split()
10 num = []
11 for i in range(len(temp)):
12     num.append(int(temp[i]))
13 n = int(n)
14 k = int(k)
15 fout = open('output.txt', 'w')
16
17 scarecrow_sort(num, k)
18 res = 'ДА'
19 for i in range(0, len(num) - 1):
20     if num[i] > num[i + 1]:
21         res = 'НЕТ'
22         break
23
24 fout.write(res)
25 fin.close()
26 fout.close()

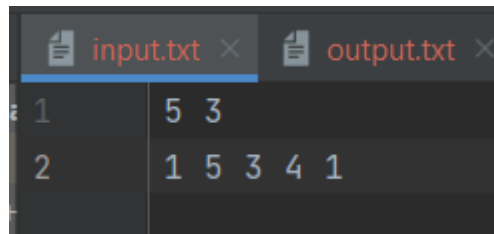
```

Листинг 3: Код первой дополнительной задачи

Текстовое объяснение решения:

Функция выполняет сортировку пугалом: в цикле проходим по каждому элементу массива до $n-k$. Сравниваем текущий элемент и элемент, который на k впереди. Если первый больше, меняем их местами. Запускаем счетчик времени. Открываем файл `input.txt`, считываем входные данные. Открываем файл `output.txt` для записи. Вызываем функцию сортировки. Создаем переменную для результата со значением «да». В цикле проходим по каждому элементу отсортированного массива, если элемент больше следующего, присваиваем переменной значение «нет». Записываем в `output.txt` результат. Закрываем файлы `input.txt` и `output.txt`.

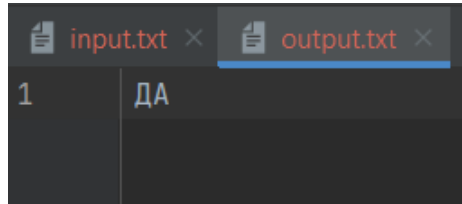
Результат работы кода на примере:



The screenshot shows a code editor with two tabs: 'input.txt' and 'output.txt'. The 'input.txt' tab is active and contains two lines of text: '1' and '2'. The 'output.txt' tab contains two lines of text: '5 3' and '1 5 3 4 1'.

input.txt	output.txt
1	5 3
2	1 5 3 4 1

Рисунок 2.1. Пример input файла



The screenshot shows a code editor with two tabs: 'input.txt' and 'output.txt'. The 'output.txt' tab is active and contains one line of text: 'ДА'. The 'input.txt' tab contains one line of text: '1'.

input.txt	output.txt
1	ДА

Рисунок 2.2. Результат работы кода

Вывод по задаче:

В этой задаче мы осуществлять сортировку пугалом и определять, когда она работает.

Вывод

В этой лабораторной работе мы изучили способы быстрой сортировки и сортировку пугалом. Также решили прикладную задачу - расчет индекса Хирша.

Список использованных источников

1. Алгоритмы: построение и анализ. — URL: https://www.google.com/books/edition/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B_%D0%9F%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B8_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7/UVg1LPacgRcC?kptab=editions&gbpv=0.
2. Лекции по предмету "Алгоритмы и структуры данных". — URL: https://drive.google.com/drive/folders/1PtZ6U_928G3DnjFn0_Hv3Rplo-iFvQa-?usp=share_link.