САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3
по курсу «Алгоритмы и структуры данных»
Тема: Быстрая сортировка, сортировки за линейное время
Вариант 12

Выполнила:
Мкртчян.К.Г.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Оглавление

Задачи по варианту

Задача №1. Улучшение Quick sort. [N баллов]

Текст задачи: Цель задачи - переделать данную реализацию рандомизированного алгоритма быстрой сортировки, чтобы она работала быстро даже с последовательностями, содержащими много одинаковых элементов.

Решение:

```
def quick_sort(m: list[int]) -> list[int]:
    if len(m) > 1:
        pivot = m[randint(0, len(m) - 1)]
        start = [i for i in m if i < pivot]
        equal = [i for i in m if i == pivot]
        end = [i for i in m if i > pivot]
        m = quick_sort(start) + equal + quick_sort(end)
    return m
```

Объяснение решения:

Эта функция реализует алгоритм быстрой сортировки (Quick Sort). Она принимает список целых чисел `m` и сортирует его:

1. Если длина списка больше 1, выбирается опорный элемент (pivot) случайным образом.
2. Список разделяется на три подсписка:
   - `start`: элементы меньше опорного.
   - `equal`: элементы равные опорному.
   - `end`: элементы больше опорного.
3. Функция рекурсивно вызывается для `start` и `end`, а затем объединяет отсортированные подсписки в один.
4. Возвращается отсортированный список.

Если длина списка меньше или равна 1, он возвращается без изменений.

Время выполнения, затраты памяти и результат:

	данные	время, сек.	память, МБ	результат
Минимальные значения	1 -9	0.0	31.73	-9
Значения из примера	5 2 3 9 2 2	0.0	31.76	2 2 2 3 9
Значения из примера	10 -73 -100 78 -99 -56 43 4 -68 12 19	0.0	31.77	-100 -99 -73 -68 -56 4 12 19 43 78
Максимальные значения	10000 -54801043 -213205967 -971742931...	0.03	32.57	-999594006 -999223889 -999179747...

Вывод по задаче: мы улучшили алгоритм быстрой сортировки за счет разделения на 3 части.

Задача №7. Цифровая сортировка. [N баллов]

Текст задачи: Дано n строк, выведите их порядок после k фаз цифровой сортировки.

Решение:

```
def strings_sort(n: int, m: int, k: int, strings: tp.List[str]):
    strings = reformat(n, strings)
    for ind in range(m-1, m-k-1, -1):
        strings = radix_sort(ind, strings)
    return [s[0] for s in strings]


def radix_sort(k: int, strings: list[tp.List[str]]) -> list[int]:
    alf = {i:[] for i in range(97, 123)}
    for s in strings:
        alf[ord(s[1][k])] += [s]
    return sum(alf.values(), [])
```

Объяснение решения:

Эти функции сортируют список строк по конкретным символам, используя алгоритм сортировки, основанный на основании разрядов (Radix Sort).

1. Функция `strings_sort(n: int, m: int, k: int, strings: tp.List[str])`:
   - Принимает параметры: `n` (количество строк), `m` (длина строк), `k` (количество символов для сортировки) и `strings` (список строк).
   - Вызывает функцию `reformat`, которая, вероятно, изменяет формат или структуру строки.
   - Затем в цикле сортирует строки по `k` последним символам (с конца) с помощью `radix_sort`.
   - Возвращает список первых элементов отсортированных строк.

2. Функция `radix_sort(k: int, strings: list[tp.List[str]])`:
   - Принимает индекс `k` и список строк `strings`.
   - Создает словарь `alf`, где ключи — это коды символов (от 97 до 122, соответствующие маленьким латинским буквам 'a' - 'z').
   - Для каждой строки добавляет ее в список в словаре `alf` по коду символа на позиции `k`.
   - Возвращает объединенные значения словаря, что приводит к сортировке строк по заданному символу.

Таким образом, эти функции в совокупности сортируют строки на основе заданных символов с использованием сортировки по разрядам.

Время выполнения, затраты памяти и результат:

	данные	время, сек.	память, МБ	результат
Минимальные значения	1 1 1 w	0.0	31.67	1
Значения из примера	3 3 2 bbb aba baa	0.0	31.7	3 2 1
Максимальные значения	5000 10000 1000 mvy tun agj	0.75	82.62	3407 1597 3119

Вывод по задаче: мы научились сортировать строки с учетом количества фаз сортировки, используя алгоритм поразрядной сортировки.

Задача №9. Ближайшие точки. [N баллов]

Текст задачи: В этой задаче, ваша цель - найти пару ближайших точек среди данных n точек (между собой). Это базовая задача вычислительной геометрии, которая находит применение в компьютерном зрении, систем управления трафиком.

Решение:

```
class Dot:
    def __init__(self, x, y):
        self.x = x
        self.y = y


def shortest_distance(n: int, dots: list[Dot]) -> float:
    dots = sorted(dots, key=lambda point: point.x)
    return separation(n, dots)


def separation(n: int, dots: list[Dot]):
    if n <= 3:
        return slow_shortest_distance(n, dots)
    middle = n // 2
    mid_dot = dots[middle]
    dl = separation(middle, dots[:middle])
    dr = separation(middle, dots[middle:])
```

```
    d = min(dl, dr)
    strip = []
    for i in range(n):
        if abs(dots[i].x - mid_dot.x) < d:
            strip.append(dots[i])
    return min(d, centre_dots(strip, len(strip), d))


def centre_dots(strip, size, d):
    min_dist = d
    strip = sorted(strip, key=lambda dot: dot.y)
    for i in range(size):
        for j in range(i + 1, size):
            if (strip[j].y - strip[i].y) >= min_dist:
                break
            min_dist = min(min_dist, euclidean_dist(strip[i], strip[j]))
    return min_dist


def euclidean_dist(dot1: Dot, dot2: Dot) -> float:
    return round(((dot1.x - dot2.x) ** 2 + (dot1.y - dot2.y) ** 2)**0.5, 4)
```

Объяснение решения:

Этот код реализует алгоритм для нахождения кратчайшего расстояния между парой точек в двумерном пространстве с использованием метода "Разделяй и властвуй".

1. Класс `Dot`:
   - Конструктор `__init__(self, x, y)` инициализирует объект точки с координатами `x` и `y`.
2. Функция `shortest_distance(n: int, dots: list[Dot]) -> float`:
   - Принимает количество точек `n` и список объектов `dots` типа `Dot`.
   - Сортирует точки по координате `x`.
   - Вызывает функцию `separation`, передавая отсортированный список точек.
3. Функция `separation(n: int, dots: list[Dot])`:
   - Если количество точек меньше или равно 3, вызывает функцию `slow_shortest_distance`, которая предположительно использует наивный метод для нахождения расстояния.
   - Разделяет список на две половины и рекурсивно находит минимальное расстояние `dl` в левой половине и `dr` в правой половине.
   - Объединяет результаты, находит точки, близкие к средней вертикали, и вызывает `centre_dots` для поиска минимального расстояния в этой "полосе".
   - Возвращает минимальное расстояние между точками.

4. Функция `centre_dots(strip, size, d)`:
- Принимает список точек в полосе `strip`, их количество `size` и минимальное расстояние `d`.
- Сортирует точки по координате `y`.
- Ищет минимальное расстояние среди точек из `strip`, находясь в пределах `d` по оси `y`.
5. Функция `euclidean_dist(dot1: Dot, dot2: Dot) -> float`:
- Расчитывает евклидово расстояние между двумя точками `dot1` и `dot2` и возвращает его округленным до четырех знаков после запятой.
Таким образом, в совокупности эти функции позволяют эффективно находить кратчайшее расстояние между парой точек с использованием подхода "разделяй и властвуй".

Время выполнения, затраты памяти и результат:

	данные	время, сек.	память, МБ	результат
Минимальные значения	1 (7, 7)	0.0	31.41	inf
Значения из примера	4 (7, 7) (1, 100) (4, 8) (7, 7)	0.0	31.44	0.0
Значения из примера	2 (0, 0) (3, 4)	0.0	31.44	5.0
Максимальные значения	100000 (215216437, 978020239) (-758313786, -619381219)...	0.41	52.75	16629.7315

Вывод по задаче: мы научились искать кратчайшее расстояние между двумя точками на плоскости, используя принцип «разделяй и властвуй».

Дополнительные задачи

Задача №2. Анти-quick sort. [N баллов]

Текст задачи: требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений.

Решение:

```
def worst_case(n: int) -> list[int]:
    worst_arr = [i + 1 for i in range(n)]
    for i in range(2, len(worst_arr)):
        worst_arr[i], worst_arr[i // 2] = worst_arr[i // 2], worst_arr[i]
    return worst_arr
```

Объяснение решения:

Функция `worst_case(n: int) -> list[int]` создает список, который можно считать "худшим случаем" для алгоритма быстрой сортировки. Давайте по шагам разберем работу этой функции:

1. Создание первоначального массива:

worst_arr = [i + 1 for i in range(n)]

Эта строка создает список `worst_arr`, содержащий числа от 1 до `n`. Например, если `n = 5`, то `worst_arr` будет равен `[1, 2, 3, 4, 5]`.

2. Перестановка элементов:

```
for i in range(2, len(worst_arr)):
    worst_arr[i], worst_arr[i // 2] = worst_arr[i // 2], worst_arr[i]
```

Этот цикл начинается с `i = 2` и проходит до конца списка. На каждой итерации элементы на позициях `i` и `i // 2` (целочисленное деление) меняются местами.

Например:

- Для `i = 2`: меняются местами `worst_arr[2]` и `worst_arr[1]`.
- Для `i = 3`: меняются местами `worst_arr[3]` и `worst_arr[1]`.
- Для `i = 4`: меняются местами `worst_arr[4]` и `worst_arr[2]`, и так далее.

Эта перестановка элементов создает структуру, которая может быть "неудобной" для алгоритма быстрой сортировки, поскольку обеспечивает определенную сложность при дальнейшем упорядочивании или выполнении операций над массивом.

3. Возврат результата:

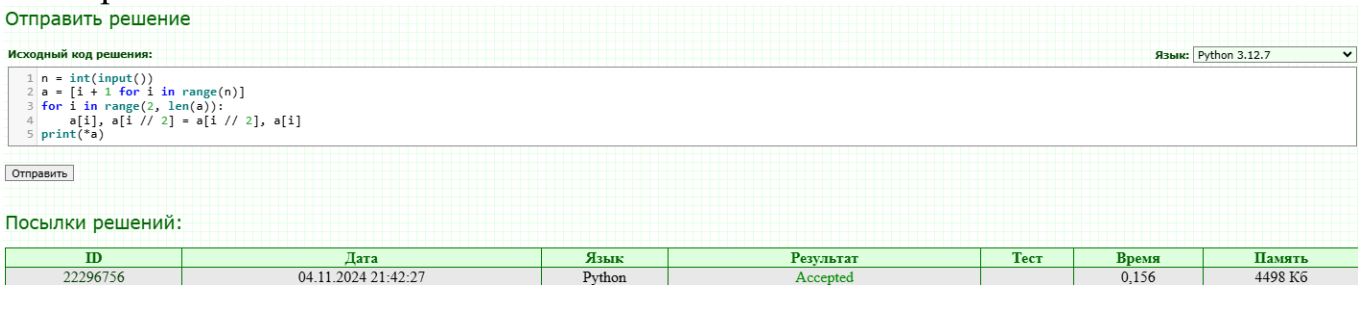
return worst_arr

Функция возвращает получившийся массив после всех перестановок.

Время выполнения, затраты памяти и результат:

	данные	время, сек.	память, МБ	результат
Минимальные значения	1	0.0	31.52	1
Значения из примера	3	0.0	31.54	1 3 2
Минимальные значения	10	0.0	31.54	1 4 6 8 10 5 3 7 2 9
Минимальные значения	1000000	0.11	71.25	1 4 6 8 10 12 14 16 18 20...

Скрин с ACMP:



Вывод по задаче: на основе работы алгоритма быстрой сортировки мы смогли написать программу, которая будет генерировать худший случай для этого алгоритма.

Задача №3. Сортировка пугалом. [N баллов]

Текст задачи: «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 матрешки 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 не убыванию размера. Может ли он это сделать?

Решение:

```
def scarecrow_sort(n: int, k: int, array: list) -> str:
    for i in range(k):
        for j in range(i+k, n, k):
            if array[i] > array[j]:
                array[i], array[j] = array[j], array[i]

    if all(array[i] <= array[i+1] for i in range(n-1)):
        return "ДА"
    else:
        return "НЕТ"
```

Объяснение решения:

Функция `scarecrow_sort(n: int, k: int, array: list) -> str` реализует специфический алгоритм сортировки и проверяет, отсортирован ли массив после применения этого алгоритма. Давайте подробно разберем, как она работает шаг за шагом.

Шаг 1: Итерация по подмассивам

1. Внешний цикл (`for i in range(k)`): Этот цикл проходит от `0` до `k-1`. Значение `i` указывает на индекс элемента в `array`, который будет использоваться для сравнения и потенциального обмена.
2. Внутренний цикл (`for j in range(i+k, n, k)`): Этот цикл проходит по элементам массива с шагом `k`, начиная с индекса `i + k`. То есть для каждого `i`, внутренний цикл сравнивает элемент `array[i]` с элементами, которые находятся `k` позиций дальше (например, `i + k`, `i + 2k`, и т.д.). Это позволяет сортировать элементы массива с учетом их индекса в зависимости от `k`.
3. Сравнение и обмен:

Если `array[i]` больше, чем `array[j]`, они меняются местами. Это приводит к тому, что элементы, находящиеся на расстоянии `k` друг от друга, сравниваются и упорядочиваются. В результате этого процесса по каждому из `k` подмассивов происходит сортировка.
Шаг 2: Проверка на отсортированность
4. Проверка сортировки: После применения алгоритма сортировки, функция проверяет, отсортирован ли массив. Для этого используется выражение с `all`, которое проверяет, что каждый элемент `array[i]` меньше или равен следующему элементу `array[i+1]`.
5. Возврат результата:
- Если массив отсортирован, функция возвращает строку `"ДА"`.
- Если нет, она возвращает строку `"НЕТ"`.
Итог
Функция `scarecrow_sort` сортирует массив по "группам" с шагом `k` и проверяет, стал ли массив отсортированным после этого процесса. Возвращает "ДА" или "НЕТ" в зависимости от результата. Это подходит для случаев, когда мы имеем дело с массивами, которые можно разбить на подмассивы, которые можно сортировать независимо друг от друга.
Время выполнения, затраты памяти и результат:

	данные	время, сек.	память, МБ	результат
Минимальные значения	2	0.0	31.55	ДА
Значения из примера	2 1 3	0.0	31.59	НЕТ
Значения из примера	1 5 3 4 1 7 6 8 2	0.0	31.59	НЕТ
Значения из примера	1 1 3 4 5	0.0	31.59	ДА
Максимальные значения	-999923669 -999961344 -999309402 -999893871	0.0	36.41	НЕТ

Вывод по задаче: мы научились проверять массив на возможность сортировки специфическим алгоритмом «сортировка пугалом».

Задача №5. Индекс Хирша. [N баллов]
Текст задачи: для заданного массива целых чисел citations, где каждое из этих чисел - число цитирований i-ой статьи ученого-исследователя, посчитайте индекс Хирша этого ученого.

Решение:

```
def h_index(citations: list[int]) -> int:
    res = 0
    citations = sorted(citations)[::-1]
```

```
    for i in range(len(citations)):
        if citations[i] >= i + 1:
            res += 1
    return res
```

Объяснение решения:
Функция `h_index(citations: list[int]) -> int` вычисляет индекс H (или H-index) для списка цитирований научных публикаций. H-index — это метрика, которая используется для оценки продуктивности и влияния ученого на основе количества его публикаций и количества цитирований этих публикаций.
Давайте разберем, как именно работает эта функция:

1. Сортировка списка на основе цитирований:
citations = sorted(citations)[::-1]
Эта строка сортирует входной список `citations` в порядке убывания. То есть, наиболее цитируемые статьи будут первыми в новом виде списка.
2. Инициализация переменной результата:
res = 0
Параметр `res` используется для подсчета значения H-index.
3. Цикл для вычисления H-index:
for i in range(len(citations)):
Этот цикл проходит по каждому элементу `citations`:
if citations[i] >= i + 1:
Здесь выполняется проверка: если количество цитирований публикации, соответствующей индексу `i`, больше или равно `i + 1`, это означает, что существует как минимум `i + 1` публикаций, имеющих по крайней мере `i + 1` цитирований.
- Если условие выполняется, то
res += 1
увеличивается на 1. То есть мы фиксируем, что у нас найден еще один кандидат для увеличения H-index.
4. Возврат значения H-index:
return res
В конце функции возвращается значение `res`, которое соответствует вычисленному H-index. Таким образом, функция `h_index` позволяет вычислить H-index на основе списка цитирований и применяется для оценки продуктивности исследователей или научных публикаций.

Время выполнения, затраты памяти и результат:

	данные	время, сек.	память, МБ	результат
Минимальные значения	4	0.0	40.93	1
Значения из примера	1 3 1	0.0	41.0	1
Значения из примера	3 0 6 1 5	0.0	41.02	3
Максимальные значения	4204 2282 4071 4615...	0.0	41.04	819

Вывод по задаче: мы научились считать индекс Хирша в массиве с количеством цитирований каждой статьи.

Вывод по лабораторной работе

В ходе лабораторной работы мы научились реализовывать алгоритм быстрой сортировки, а также алгоритмы сортировки за линейное время (например, сортировка пугалом).