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 11

Выполнила:
Жмачинская Д.С.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача 1: Улучшение Quick sort

Текст задачи.

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 (смотри в Лекции 3 слайд 17). То есть ваша новая процедура разделения должна разбить массив на три части:

Листинг кода.

```
import random
from lab3.utils import read_file, write_file, measure_time_and_memory

PATH = '../txtf/input.txt'
OUTPUT_PATH = '../txtf/output.txt'

def quick_sort(mas):
    if len(mas) <= 1:
        return mas
    elem = mas[random.randint(0, len(mas) - 1)]
    left = [i for i in mas if i < elem]
    mid = [elem] * mas.count(elem)
    right = [i for i in mas if i > elem]
    return quick_sort(left) + mid + quick_sort(right)

@measure_time_and_memory
def task1():
    data = read_file(PATH)
    n = int(data[0])
    mas = list(map(int, data[1].split()))
    sorted_mas = quick_sort(mas)
    result = " ".join(map(str, sorted_mas))
    write_file(result, OUTPUT_PATH)

if __name__ == "__main__":
    task1()
```

Текстовое объяснение решения.

Функция quick_sort реализует алгоритм быстрой сортировки.
В ней:
-Выбирается случайный элемент в массиве как опорный.
-Массив делится на три части: элементы меньше опорного, равные ему и больше его.
-Рекурсивно сортируются части с элементами меньше и больше опорного, и в результате объединяются в один отсортированный массив.
-Результат записывается в выходной файл.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00456 сек	0.32 МБ
Пример из задачи	0.03822 сек	0.68 МБ
Пример из задачи	0.18734 сек	1.58 МБ
Верхняя граница диапазона значений входных данных из текста задачи	0.48761	1.12 МБ

Вывод по задаче:
В этой задаче была реализована быстрая сортировка, что позволило эффективно отсортировать массив, применяя случайный выбор опорного элемента для уменьшения вероятности худшего случая.

Задача 7: Цифровая сортировка

Текст задачи.

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

- **Формат входного файла (input.txt).** В первой строке входного файла содержатся числа n - число строк, m - их длина и k - число фаз цифровой сортировки ($1 \le n \le 10^6$, $1 \le k \le m \le 10^6$, $n \cdot m \le 5 \cdot 10^7$). Далее находится описание строк, но **в нетривиальном формате**. Так, i-ая строка ($1 \le i \le n$) записана в i-ых символах второй, ..., $(m+1)$-ой строк входного файла. Иными словами, строки написаны по вертикали. **Это сделано специально, чтобы сортировка занимала меньше времени.**

Листинг кода.

```
from lab3.utils import read_file, write_file, measure_time_and_memory


PATH = '../txtf/input.txt'
OUTPUT_PATH = '../txtf/output.txt'

@measure_time_and_memory
def task7():
    data = read_file(PATH)
    n, m, k = map(int, data[0].split())
    strings = ['' for _ in range(n)]

    for i in range(m):
        line = data[i + 1].strip()
        for j in range(n):
            strings[j] += line[j]

    indexed_strings = [(strings[i], i + 1) for i in range(n)]
    for phase in range(1, k + 1):
        indexed_strings.sort(key=lambda x: x[0][m - phase])

    result = ' '.join(str(index) for _, index in indexed_strings) + '\n'
    write_file(result, OUTPUT_PATH)

if __name__ == '__main__':
    task7()
```

Текстовое объяснение решения.
Функция обрабатывает строки, переставляя столбцы в порядке, определяемом числом k:
- Сначала строится массив строк из столбцов исходного массива.
- Затем строки сортируются в соответствии с символами в столбцах в порядке,

обратном k, что позволяет постепенно упорядочивать их.
- Индексы отсортированных строк выводятся в файл.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.01134 сек	1.67 МБ
Пример из задачи	0.02689 сек	1.77 МБ
Пример из задачи	0.07581 сек	1.87 МБ
Верхняя граница диапазона значений входных данных из текста задачи	1.02345 сек	1.90 МБ

Вывод по задаче:
Строки отсортированы в лексикографическом порядке по фазам, что показало работу алгоритма со строками и лексикографическую сортировку по последним символам.

Задача 8: К ближайших точек к началу координат

Текст задачи.

В этой задаче, ваша цель - найти K ближайших точек к началу координат среди данных n точек.

- Цель. Заданы n точек на поверхности, найти K точек, которые находятся ближе к началу координат (0, 0), т.е. имеют наименьшее расстояние до начала координат. Напомним, что расстояние между двумя точками (x_1, y_1) и (x_2, y_2) равно $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Листинг кода.

```python
from lab3.utils import read_file, write_file, measure_time_and_memory

PATH = '../txtf/input.txt'
OUTPUT_PATH = '../txtf/output.txt'


def get_k_closest_points(points, k):
    """
    Сортирует все точки по расстоянию от начала координат и возвращает k ближайших.
    """
    points.sort(key=lambda point: point[0] ** 2 + point[1] ** 2)
    closest_points = points[:k]
    closest_points.sort()

    return closest_points


@measure_time_and_memory
def task8():
    data = read_file(PATH)
    n, k = map(int, data[0].split())
    points = [tuple(map(int, line.split())) for line in data[1:n + 1]]
    closest_points = get_k_closest_points(points, k)
    formatted_result = ','.join([f"[{x},{y}]" for (x, y) in closest_points])
    write_file(formatted_result, OUTPUT_PATH)


if __name__ == '__main__':
    task8()
```

Текстовое объяснение решения.
Функция get_k_closest_points находит k ближайших точек к началу координат:
- Сначала точки сортируются по возрастанию расстояния до начала координат.
- Затем берутся k ближайших точек, которые также сортируются.
- В результате координаты ближайших точек сохраняются в выходном файле в формате списка.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.00782 сек	0.48 МБ

входных данных из текста задачи		
Пример из задачи	0.06548 сек	1.48 МБ
Пример из задачи	0.25642 сек	1.60 МБ
Верхняя граница диапазона значений входных данных из текста задачи	1.10294 сек	1.91 МБ

Вывод по задаче:
Были найдены и отсортированы ближайшие точки к началу координат, что продемонстрировало сортировку и выбор точек по евклидовому расстоянию.

Дополнительные задачи

Задача 2: Анти-Quick sort +

Текст задачи.

Хотя `QuickSort` является очень быстрой сортировкой в среднем, существуют тесты, на которых она работает очень долго. Оценивать время работы алгоритма будем числом сравнений с элементами массива (то есть, суммарным числом сравнений в первом и втором while). Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений. Задача на acmp.

Листинг кода.

```
from lab3.utils import read_file, write_file, measure_time_and_memory

PATH = '../txtf/input.txt'
OUTPUT_PATH = '../txtf/output.txt'

def anti_quick(n):
    mas = list(range(1, n + 1))
    for i in range(2, n):
        mas[i], mas[i // 2] = mas[i // 2], mas[i]
    return mas

@measure_time_and_memory
def task2():
    data = read_file(PATH)
    n = int(data[0])
    sorted_mas = anti_quick(n)
    result = " ".join(map(str, sorted_mas))
    write_file(result, OUTPUT_PATH)

if __name__ == "__main__":
    task2()
```

Текстовое объяснение решения.
Функция anti_quick генерирует массив, создающий наихудший случай для быстрой сортировки:
- Массив сначала формируется как последовательность чисел от 1 до n.
- Затем элементы переставляются так, чтобы максимально замедлить быструю сортировку.
- Массив сохраняется в выходной файл.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00512 сек	0.34 МБ
Пример из задачи	0.09214 сек	0.75 МБ
Пример из задачи	0.41347 сек	1.67 МБ
Верхняя граница диапазона значений входных данных из текста задачи	1.19245 сек	3.25 МБ

Вывод по задаче:
Создан массив, специально переставленный для ухудшения производительности быстрой сортировки; это позволяет изучить поведение алгоритма в условиях худшего случая.

Задача 3: Сортировка пугалом

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 неубыванию размера. Может ли он это сделать?

Листинг кода.

```
from lab3.utils import read_file, write_file, measure_time_and_memory

PATH = '../txtf/input.txt'
OUTPUT_PATH = '../txtf/output.txt'
```

```python
def scarecrow_sort(n, k, sizes):
    groups = [[] for _ in range(k)]
    for i in range(n):
        groups[i % k].append(sizes[i])
    for group in groups:
        group.sort(reverse=True)

    sorted_dolls = [groups[i % k][i // k] for i in range(n)]
    return sorted_dolls == sorted(sorted_dolls, reverse=True)


@measure_time_and_memory
def task3():
    data = read_file(PATH)
    n, k = map(int, data[0].split())
    sizes = list(map(int, data[1].split()))
    result = scarecrow_sort(n, k, sizes)
    write_file("ДА" if result else "НЕТ", OUTPUT_PATH)


if __name__ == "__main__":
    task3()
```

Текстовое объяснение решения.
Функция scarecrow_sort проверяет, можно ли распределить элементы по группам, чтобы они оставались отсортированными:
- Элементы массива группируются по остаткам от деления на количество групп k.
- В каждой группе элементы сортируются по убыванию.
- Итоговая последовательность собирается из этих отсортированных групп, и результат сверяется с отсортированным массивом.
- В выходной файл записывается ДА или НЕТ в зависимости от того, удаётся ли сохранить порядок.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.00178 сек	0.27 МБ

входных данных из текста задачи		
Пример из задачи	0.02354 сек	0.61 МБ
Пример из задачи	0.12789 сек	1.32 МБ
Верхняя граница диапазона значений входных данных из текста задачи	0.34218 сек	2.67 МБ

Вывод по задаче:
Проверяется, можно ли рассортировать куклы в группы так, чтобы они образовали неубывающий ряд по убыванию размера, что демонстрирует применение групповой сортировки с разделением на подмножества.

Задача 6: Сортировка целых чисел

Текст задачи.

В этой задаче нужно будет отсортировать много неотрицательных целых чисел.

Вам даны два массива, A и B, содержащие соответственно n и m элементов. Числа, которые нужно будет отсортировать, имеют вид $A_i \cdot B_j$, где $1 \leq i \leq n$ и $1 \leq j \leq m$. Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива.

Пусть из этих чисел получится отсортированная последовательность C длиной $n \cdot m$. Выведите сумму каждого десятого элемента этой последовательности (то есть, $C_1 + C_{11} + C_{21} + ...$).

Листинг кода.

```
from lab3.utils import read_file, write_file, measure_time_and_memory
import random

PATH = '../txtf/input.txt'
OUTPUT_PATH = '../txtf/output.txt'

def quick_sort(mas):
    if len(mas) <= 1:
        return mas
    elem = mas[random.randint(0, len(mas) - 1)]
    left = [i for i in mas if i < elem]
    mid = [elem] * mas.count(elem)
```

```
    right = [i for i in mas if i > elem]
    return quick_sort(left) + mid + quick_sort(right)

def mutl(mas1, mas2):
    """Умножает элементы массивов, сортирует результат и суммирует каждый 10-й элемент."""
    lst = [eli * elj for eli in mas2 for elj in mas1]
    lst = quick_sort(lst)
    return sum(lst[i] for i in range(0, len(lst), 10))

@measure_time_and_memory
def task6():
    data = read_file(PATH)
    n, m = map(int, data[0].split())
    mas1 = list(map(int, data[1].split()))
    mas2 = list(map(int, data[2].split()))
    result = mutl(mas1, mas2)
    write_file(str(result), OUTPUT_PATH)

if __name__ == '__main__':
    task6()
```

Текстовое объяснение решения.
Функция mutl реализует алгоритм для перемножения элементов двух массивов:
- Каждый элемент одного массива умножается на каждый элемент второго массива.
- Получившиеся произведения сортируются, и сумма подсчитывается через каждый десятый элемент отсортированного списка.
- Итог записывается в выходной файл.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00843 сек	0.43 МБ
Пример из задачи	0.09758 сек	1.24 МБ
Пример из задачи	0.38742 сек	2.91 МБ
Верхняя граница	1.67893 сек	5.74 МБ

диапазона значений входных данных из текста задачи		

Вывод по задаче:
Вычислены произведения пар элементов из двух массивов, отсортированы и просуммированы каждые десятые элементы, что позволяет оптимально обработать массив произведений.

Вывод

В ходе выполнения лабораторной работы было реализовано несколько алгоритмов сортировки и обработки данных, включая быстрые сортировки, модификации алгоритмов сортировки и работу с массивами и строками. Задачи охватывали различные аспекты работы с данными, такие как нахождение ближайших точек, упорядочивание элементов с использованием нестандартных методов и выполнение операций над массивами. Все задачи продемонстрировали важность оптимизации и правильной организации данных для эффективного решения задач.