

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

Выполнил:
Авдиенко Данила Андреевич
Группа К3140

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Оглавление

Оглавление

<i>Содержание отчета.....</i>	<i>2</i>
<i>Задачи по варианту.....</i>	<i>3</i>
Задание № 1. Улучшение Quick_sort.....	3
Задание №4. Точки и отрезки	5
Задание № 7. Цифровая сортировка	7

Задачи по варианту

Задание № 1. Улучшение Quick_sort

Текст задачи.

1. Используя псевдокод процедуры Randomized - QuickSort, а также 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:

Код:

```
def quick_sort(arr: list):
    """Обычный алгоритм быстрой сортировки"""
    if len(arr) <= 1:
        return arr # base case
    pivot = arr[0]
    gt = []
    mn = []
    for i in arr:
        if i > pivot:
            gt.append(i)
        else:
            mn.append(i)
    return quick_sort(mn) + [pivot] + quick_sort(gt)

def quick_sort_upgrade(arr: list):
    """Улучшенный алгоритм быстрой сортировки"""
    if len(arr) <= 1:
        return arr # base case
    pivot = arr[0]
    gt = []
    eq = []
    mn = []
    for i in arr:
        if i > pivot:
            gt.append(i)
        elif i < pivot:
            mn.append(i)
        else:
            eq.append(i)
    return quick_sort_upgrade(mn) + eq + quick_sort_upgrade(gt)

if __name__ == "__main__":
    time_start = time.perf_counter()
    n, arr = read_input("../txtf/input.txt")
    arr = quick_sort_upgrade(arr)
    write_output(arr, "../txtf/output.txt")
    time_memory_tracking(time_start)
```

Текстовое объяснение функции.

Функция quick_sort сортирует список, разделяя его на меньшие и большие относительно первого элемента, а затем рекурсивно объединяет отсортированные части. quick_sort_upgrade делает то же самое, но

дополнительно выделяет элементы, равные опорному, для ускорения работы.

Примеры работы кода:

Quick_sort_upgrade
Время: 0.000330000000000000043
Память: 8.9 МБ
Process finished with exit code 0

Для функции написаны модульные тесты при помощи unittest.

Вывод: реализован алгоритм быстрой сортировки в двух версиях — базовой и улучшенной. Базовая версия делит элементы на меньшие и большие, улучшенная дополнительно выделяет равные опорному для ускорения. Обе функции протестированы и работают корректно.

Задание №4. Точки и отрезки

Текст задачи.

Допустим, вы организовываете онлайн-лотерею. Для участия нужно сделать ставку на одно целое число. При этом у вас есть несколько интервалов последовательных целых чисел. В этом случае выигрыш участника пропорционален количеству интервалов, содержащих номер участника, минус количество интервалов, которые его не содержат. (В нашем случае для начала - подсчет только количества интервалов, содержащих номер участника). Вам нужен эффективный алгоритм для расчета выигрышей для всех участников. Наивный способ сделать это - просто просканировать для всех участников список всех интервалов. Однако ваша лотерея очень популярна: у вас тысячи участников и тысячи интервалов. По этой причине вы не можете позволить себе медленный наивный алгоритм.

Код:

```
def lottery(data):
    # Распаковка входных данных
    s, p = data[0] # количество отрезков и точек
    array = data[1:s + 1] # список всех списков
    dot_list = data[s + 1] # список всех точек

    events = []

    for ul in array:
        events.append((ul[0], -1))
        events.append((ul[-1], 1)) # добавили конец и начало каждого
# отрезка, пометив их индексами -1 и 1

    for idx, dot in enumerate(dot_list):
        events.append((dot, 0, idx))

    events = sorted(events, key=lambda x: (x[0], x[1]))
    cntr = 0
    list_peresecheniy = []
    for event in events:
        if event[1] == -1:
            cntr += 1
        elif event[1] == 1:
            cntr -= 1
        elif event[1] == 0:
            list_peresecheniy.append((cntr, event[2]))
    list_peresecheniy = sorted(list_peresecheniy, key=lambda x: x[1])
    result = [count for count, _ in list_peresecheniy]
    return result

if __name__ == "__main__":
    time_start = time.perf_counter()
    data = read_input_for_lottery("../txtf/input.txt")
    write_output(lottery(data), "../txtf/output.txt")
    time_memory_tracking(time_start)
```

Текстовое объяснение функции lottery()

Функция lottery принимает на вход данные о количестве отрезков и точек, а также список координат концов отрезков и точек. Она

преобразует данные в набор событий: начало и конец отрезков помечаются как -1 и 1 соответственно, а точки — как 0. Все события сортируются по их координатам (и типу события). После этого функция подсчитывает пересечения каждого отрезка с точками, изменяя счетчик пересечений на основе типа события. Результаты подсчета пересечений для каждой точки возвращаются в виде списка.

Затраты на работу кода:

lottery()
Время: 0.00069449999999999989
Память: 8.6 МБ
Process finished with exit code 0

Для функции написаны модульные тесты при помощи unittest.

Вывод: Был реализован алгоритм подсчета пересечений точек с отрезками, основанный на сортировке событий и однократном их прохождении с использованием счетчика. Алгоритм протестирован на входных данных и корректно возвращает количество пересечений для каждой точки в порядке их появления.

Задание № 7. Цифровая сортировка

Текст задачи:

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

Код:

```
def digital_sorting(n: int, m: int, k: int, matrix: list):
    # Инициализация индексов
    indices = [i for i in range(1, n + 1)]

    # Выполнение k фаз сортировки
    for phase in range(1, k + 1):
        # Определяем строку для сортировки
        row_to_sort_by = m - phase

        # Сортируем индексы на основе символа в строке `row_to_sort_by`
        indices.sort(key=lambda i: matrix[row_to_sort_by][i - 1]) # `i - 1` из-за индексации с 1

    # Возвращаем итоговый порядок индексов
    return indices

if __name__ == "__main__":
    time_start = time.perf_counter()
    with open("../txtf/input.txt", "r") as inp:
        n, m, k = map(int, inp.readline().split())
        matrix = [inp.readline().strip() for i in range(m)]
    result_indices = digital_sorting(n, m, k, matrix)
    time_memory_tracking(time_start)
    write_output(result_indices, "../txtf/output.txt")
```

Текстовое объяснение решения.

Функция `digital_sorting` выполняет поразрядную сортировку индексов столбцов матрицы. На вход подаются размеры матрицы n и m , количество фаз сортировки k , и сама матрица в виде списка строк. На каждой фазе выбирается строка, по которой производится сортировка, начиная с нижней ($m - \text{phase}$). Индексы сортируются по символам в указанной строке, и после выполнения всех фаз возвращается итоговый порядок индексов.

Затраты на работу кода:

Время: 0.0005701669999999999

Память: 8.4 МБ

Process finished with exit code 0

Для функции написаны модульные тесты при помощи `unittest`.

Вывод по задаче: Был реализован алгоритм поразрядной сортировки, который выполняет k фаз сортировки, учитывая символы в строках матрицы снизу вверх. Алгоритм корректно определяет итоговый порядок индексов, протестирован на входных данных и сохраняет результат в файл.

Вспомогательные файлы. Utils и runall

Utils – содержит в себе переиспользуемые блоки кода для работы с файлами
runall – содержит в себе скрипт для запуска всех тестов лабораторной работы и вывода результата в консоль

Вывод:

Реализованы и протестированы при помощи unittest алгоритмы сортировки: быстрая, поразрядная и подсчет пересечений. Алгоритмы работают корректно и эффективно.