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 10

Выполнил:

Говоров П. И.

К3140

Проверил:

Афанасьев А. В.

Санкт-Петербург

2024 г.

Содержание отчета

Оглавление

[Содержание отчета 2](#_Toc182054457)

[1 задача. Улучшение Quick sort 3](#_Toc182054458)

[2 задача. Анти-quick sort 5](#_Toc182054459)

[4 задача. Точки и отрезки 6](#_Toc182054460)

[6 задача. Сортировка целых чисел 9](#_Toc182054461)

[8 задача. K ближайших точек к началу координат 11](#_Toc182054462)

[9 задача. Ближайшие точки 12](#_Toc182054463)

[Вывод 15](#_Toc182054464)

Задачи по варианту

**1 задача. Улучшение Quick sort**

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:

A[k] < x для всех ℓ + 1 ≤ k ≤ m1 – 1

A[k] = x для всех m1 ≤ k ≤ m2

A[k] > x для всех m2 + 1 ≤ k ≤ r

Листинг кода

def quick\_sort\_p3(arr,l,r):  
 if l < r:  
 lt, gt = partition3(arr, l, r)  
 quick\_sort\_p3(arr, l, lt - 1)  
 quick\_sort\_p3(arr, gt + 1,r)  
  
  
def partition3(arr, l, r):  
 x\_ind = random.randint(l, r)  
 arr[l], arr[x\_ind] = arr[x\_ind], arr[l]  
 x = arr[l]  
 lt = l  
 eq = l  
 gt = r  
 while eq <= gt:  
 if arr[eq] < x:  
 arr[lt], arr[eq] = arr[eq], arr[lt]  
 lt += 1  
 eq += 1  
 elif arr[eq] > x:  
 arr[gt], arr[eq] = arr[eq], arr[gt]  
 gt -= 1  
 else:  
 eq += 1  
  
 return lt, gt

Текстовое объяснение решения.

Разделяем массив на числа большие, меньшие, равные заданному “pivot” и находим границы разделения. Рекурсивно делим массив по этим разделителям.

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях: Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0094 sec | 30.43 Mb |
| Пример из задачи | 0.00894 sec | 30.25 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.16941 sec | 31.46 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(nlogn) и используемую память.

2 задача. Анти-quick sort

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.

• Формат входного файла (input.txt). В первой строке находится единственное число n (1 ≤ n ≤ 10^6).

• Формат выходного файла (output.txt). Вывести перестановку чисел от 1 до

n, на которой быстрая сортировка выполнит максимальное число сравнений.

Если таких перестановок несколько, вывести любую из них.

• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

Листинг кода

def generate\_worst\_case(ln):  
 arr = []  
 for i in range(0, ln):  
 arr += [i + 1]  
 if i > 1:  
 arr[-1], arr[i // 2] = arr[i // 2], arr[-1]  
 return arr

Текстовое объяснение решения.

По очереди добавляем элемент и меняем его с центральным

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00451 sec | 30.24 Mb |
| Пример из задачи | 0.00173 sec | 30.23 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.08889 sec | 31.90 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

4 задача. Точки и отрезки

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.

.

Цель. Вам дается набор точек и набор отрезков. Цель состоит в том, чтобы

вычислить для каждой точки количество отрезков, содержащих эту точку.

Формат входного файла (input.txt). Первая строка содержит два неотрицательных целых числа s и p. s - количество отрезков, p - количество

точек. Следующие s строк содержат 2 целых числа ai, bi, которые определяют i-ый отрезок [ai, bi]. Последняя строка определяет p целых чисел – точек x1, x2, ..., xp. Ограничения: 1 ≤ s, p ≤ 50000; −108 ≤ ai ≤ bi ≤ 108 для всех 0 ≤ i < s; −108 ≤ xi ≤ 108 для всех 0 ≤ j < p.

Формат выходного файла (output.txt).Выведите p неотрицательных целых

чисел k0, k1..., kp−1, где ki - это число отрезков, которые содержат xi

То есть, ki = |j : aj ≤ xi ≤ bj |.

Листинг кода

def points\_and\_segments(segments, points):  
 start\_seg\_dct = {}  
 end\_seg\_dct = {}  
 for a,b in segments:  
 start\_seg\_dct[a] = start\_seg\_dct.get(a, 0) + 1  
 end\_seg\_dct[b+1] = end\_seg\_dct.get(b+1, 0) + 1  
 points\_c = points.copy()  
 points = sorted(set(points))  
 start\_seg\_arr = sorted(start\_seg\_dct.keys())  
 end\_seg\_arr = sorted(end\_seg\_dct.keys())  
  
 i = j = 0  
 cur\_layers = 0  
 ans\_arr = {}  
 for k in range(len(points)):  
 if j >= len(end\_seg\_arr):  
 ans\_arr[points[k]] = cur\_layers  
 elif i < len(start\_seg\_arr) and j < len(end\_seg\_arr) and points[k] < start\_seg\_arr[i] and points[k] < end\_seg\_arr[j]:  
 ans\_arr[points[k]] = cur\_layers  
 else:  
 while i < len(start\_seg\_arr) and points[k] >= start\_seg\_arr[i]:  
 cur\_layers += start\_seg\_dct[start\_seg\_arr[i]]  
 i += 1  
  
 while j < len(end\_seg\_arr) and points[k] >= end\_seg\_arr[j]:  
 cur\_layers -= end\_seg\_dct[end\_seg\_arr[j]]  
 j += 1  
  
 ans\_arr[points[k]] = cur\_layers  
  
 return [ans\_arr[i] for i in points\_c]

Текстовое объяснение решения.

Сортируем все отрезки, сохраняем сортированный массив начал и концов отрезков, через два указателя для этих массивов поддерживаем текущее количество слоев отрезков.

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях: Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0055 sec | 30.16 Mb |
| Пример из задачи | 0.00554 sec | 30.25 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.77409 sec | 31.79 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости времени работы O(n), а также используемую память.

6 задача. Сортировка целых чисел

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 Ai

· Bj , где 1 ≤ i ≤ n и 1 ≤ j ≤ m. Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива.

Пусть из этих чисел получится отсортированная последовательность C длиной n · m. Выведите сумму каждого десятого элемента этой последовательности (то есть, C1 + C11 + C21 + ...).

• Формат входного файла (input.txt). В первой строке содержатся числа n

и m (1 ≤ n, m ≤ 6000) – размеры массивов. Во второй строке содержится

6n чисел – элементы массива A. Аналогично, в третьей строке содержится

m чисел — элементы массива B. Элементы массива неотрицательны и не

превосходят 40000.

• Формат выходного файла (output.txt). Выведите одно число — сумму

каждого десятого элемента последовательности, полученной сортировкой

попарных произведенй элементов массивов A и B.

• Ограничение по времени. 2 сек Листинг кода

def sort\_integer\_nums(arr\_a, arr\_b):  
 arr = [i\*j for i in arr\_a for j in arr\_b]  
 cnt\_s\_a = [0]\*40001  
  
 for i in arr:  
 cnt\_s\_a[i] += 1  
 sorted\_arr = []  
 for i in range(len(cnt\_s\_a)):  
 if cnt\_s\_a[i] != 0:  
 sorted\_arr.extend([i]\*cnt\_s\_a[i])  
 sm = sum(sorted\_arr[i] for i in range(0, len(sorted\_arr), 10))  
 return sm

Текстовое объяснение решения.

Сохраняем в массив пары произведения двух массивов. Добавляем в индекс значения элемента +1, потом соединяем по очереди все элементы, таким образом сортируя

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0055 sec | 30.16 Mb |
| Пример из задачи | 0.00563 sec | 30.57 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.77409 sec | 31.79 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

8 задача. K ближайших точек к началу координат

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 (x1, y1) и (x2, y2) равно p (x1 − x2) 2 + (y1 − y2) 2. 8 • Формат ввода или входного файла (input.txt). Первая строка содержит n - общее количество точек на плоскости и через пробел K - количество ближайший точек к началу координат, которые надо найти. Каждая следующая из n строк содержит 2 целых числа xi , yi , определяющие точку (xi , yi). Ограничения: 1 ≤ n ≤ 105 ; −109 ≤ xi , yi ≤ 109 - целые числа. • Формат выхода или выходного файла (output.txt). Выведите K ближайших точек к началу координат в строчку в квадратных скобках через запятую. Ответ вывести в порядке возрастания расстояния до начала координат. Если оно равно, порядок произвольный. • Ограничение по времени. 10 сек. • Ограничение по памяти. 256 мб

Листинг кода:

def find\_k\_closest\_points(points, k):  
 points = sorted([[a,b,find\_dist(a,b)] for a,b in points], key=lambda x:x[2])  
 points = points[:k]  
 ans = ''  
 for i in range(len(points)-1):  
 a, b, c = points[i]  
 ans += f'[{a},{b}],'  
 a, b, c = points[-1]  
 ans += f'[{a},{b}]'  
 return ans  
  
def find\_dist(dot1, dot2):  
 return (dot1 \*\* 2 + dot2 \*\* 2) \*\* 0.5

Текстовое объяснение решения.

Подсчитаем для каждой точки расстояние до (0,0) и отсортируем по этому ключу, обрежем до k элементов

Результат работы кода на примере из задачи:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00312 sec | 30.20 Mb |
| Пример из задачи | 0.00264 sec | 30.13 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.75601 sec | 30.20 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(nlogn) и используемую память.

9 задача. Ближайшие точки

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. • Цель. Заданы n точек на поверхности, найти наименьшее расстояние между двумя (разными) точками. Напомним, что расстояние между двумя точками (x1, y1) и (x2, y2) равно p (x1 − x2) 2 + (y1 − y2) 2. • Формат ввода или входного файла (input.txt). Первая строка содержит n - количество точек. Каждая следующая из n строк содержит 2 целых числа xi , yi , определяющие точку (xi , yi). Ограничения: 1 ≤ n ≤ 105 ; −109 ≤ xi , yi ≤ 109 - целые числа. • Формат выхода или выходного файла (output.txt). Выведите минимальное расстояние. Абсолютная погрешность между вашим ответом и оптимальным решением должна быть не более 10−3 . Чтобы это обеспечить, выведите ответ с 4 знаками после запятой. 9 • Ограничение по времени. 10 сек. • Ограничение по памяти. 256 мб.

Листинг кода

def recursion\_pair\_closest(arr\_s\_x, arr\_s\_y):  
 if len(arr\_s\_x) <= 3:  
 min\_dist = 10\*\*18  
 best\_pair = ()  
 for i in range(len(arr\_s\_x)):  
 for j in range(i + 1, len(arr\_s\_x)):  
 min\_d = find\_dist(arr\_s\_x[i], arr\_s\_x[j])  
 if min\_d < min\_dist:  
 min\_dist = min\_d  
 best\_pair = (arr\_s\_x[i], arr\_s\_x[j])  
  
 return min\_dist, best\_pair  
  
 mid = len(arr\_s\_x) // 2  
 mid\_sep = arr\_s\_x[mid][0]  
  
 left\_arr\_x, right\_arr\_x = arr\_s\_x[:mid], arr\_s\_x[mid:]  
 left\_arr\_y = [d for d in arr\_s\_y if d[0] < mid\_sep]  
 right\_arr\_y = [d for d in arr\_s\_y if d[0] > mid\_sep]  
  
 dist\_l, pair\_l = recursion\_pair\_closest(left\_arr\_x, left\_arr\_y)  
 dist\_r, pair\_r = recursion\_pair\_closest(right\_arr\_x, right\_arr\_y)  
  
 min\_d = min(dist\_l, dist\_r)  
 if dist\_l < dist\_r:  
 best\_pair = pair\_l  
 else:  
 best\_pair = pair\_r  
  
 strip\_line\_arr = []  
 for dot in arr\_s\_y:  
 if abs(dot[0] - mid\_sep) < min\_d:  
 strip\_line\_arr.append(dot)  
  
 for i in range(len(strip\_line\_arr)):  
 for j in range(i + 1, min(i + 7, len(strip\_line\_arr))):  
 ds = find\_dist(strip\_line\_arr[i], strip\_line\_arr[j])  
 if ds < min\_d:  
 min\_d = ds  
 best\_pair = (strip\_line\_arr[i], strip\_line\_arr[j])  
  
 return min\_d, best\_pair  
  
def find\_dist(dot1, dot2):  
 return ((dot1[0] - dot2[0]) \*\* 2 + (dot1[1] - dot2[1]) \*\* 2) \*\* 0.5

Текстовое объяснение решения.

Через метод разделяй и властвуй, делим сортированный массив по x координате и рекурсивно запускаем для разделенных массивов, на возврате проверяем дистанцию на пересечении двух разделений.

Результат работы на примере:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00153 sec | 30.19 Mb |
| Пример из задачи | 0.00253 sec | 30.49 Mb |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.81082 sec | 30.41 Mb |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(nlogn) и используемую память.

Вывод

1. В этой лабораторной мы научились делать quick sort, quick sort partition 3 и count sort, а также находить минимальную дистанцию между точками за O(n logn)
2. Алгоритм O(n) и O(n logn) выполняется достаточно быстро, относительно квадратичной сложности, затраты памяти также прямо пропорциональны линейной.
3. С помощью методов *time.perf\_counter()* и *psutil.Process().memory\_info().rss* можно отслеживать ресурсозатратность алгоритмов.