САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2

по курсу «Алгоритмы и структуры данных»

Тема: Сортировка слиянием. Метод декомпозиции

Вариант 10

Выполнил:

Абакар Иссака Малли

К3141

Проверил:

Афанасьев А. В.

Санкт-Петербург

2024 г.

Содержание отчета

Оглавление

Содержание отчета 2

Задачи по варианту 3

1 задача. Сортировка слиянием 3

3 задача. Число инверсий 5

4 задача. Бинарный поиск 7

5 задача. Представитель большинства 9

6 задача. Поиск максимальной прибыли 11

7 задача. Поиск максимального подмассива за линейное время 14

Вывод 16

Задачи по варианту

1 задача. Сортировка слияниемИспользуя псевдокод процедур Merge и Merge-sort из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и

проверьте сортировку, создав несколько рандомных массивов, подходящих

под параметры:

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

содержится число n (1 ≤ n ≤ 2 · 104) — 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. 2сек.

• Ограничение по памяти. 256 мб.

Листинг кода

def merge\_sort(arr, p, r):

    if p < r:

        q = (p+r)//2

        merge\_sort(arr,p,q)

        merge\_sort(arr,q+1,r)

        merge(arr, p, q, r)

def merge(arr, p, q, r):

    arr1 = arr[p:q+1]

    arr2 = arr[q+1:r+1]

    i = j = 0

    while i < len(arr1) and j < len(arr2):

        if arr1[i] < arr2[j]:

            arr[p+i+j] = arr1[i]

            i += 1

        else:

            arr[p+i+j] = arr2[j]

            j += 1

    while i < len(arr1):

        arr[p+i+j] = arr1[i]

        i += 1

    while j < len(arr2):

        arr[p+i+j] = arr2[j]

        j += 1

Текстовое объяснение решения.

Метод разделяй и властвуй: рекурсивно делим массив пополам и выполняем на возврате слияние двух рекурсивно отсортированных массивов в один.

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00022 sec

15.54 Mb

Пример из задачи

0.00016 sec

15.54 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.02904 sec

16.51 Mb

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

3 задача. Число инверсий

Инверсией в последовательности чисел A называется такая ситуация, когда

i < j, а Ai > Aj . Количество инверсий в последовательности в некотором роде определяет, насколько близка данная последовательность к отсортированной.

Например, в сортированном массиве число инверсий равно 0, а в массиве, сортированном наоборот - каждые два элемента будут составлять инверсию (всего n(n − 1)/2).

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

• Формат входного файла (input.txt). В первой строке входного файла содержится число n (1 ≤ n ≤ 105) — число элементов в массиве. Во второй

строке находятся n различных целых чисел, по модулю не превосходящих

109

• Формат выходного файла (output.txt). В выходной файл надо вывести

число инверсий в массиве.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб.

Листинг кода

def merge\_sort(arr, p, r):

    global k

    if p < r:

        q = (p+r)//2

        merge\_sort(arr,p,q)

        merge\_sort(arr,q+1,r)

        merge(arr, p, q, r)

def merge(arr, p, q, r):

    global k

    arr1 = arr[p:q+1]

    arr2 = arr[q+1:r+1]

    i = j = 0

    while i < len(arr1) and j < len(arr2):

        if arr1[i] <= arr2[j]:

            arr[p+i+j] = arr1[i]

            i += 1

        else:

            arr[p+i+j] = arr2[j]

            j += 1

            k += len(arr1)-i

    while i < len(arr1):

        arr[p+i+j] = arr1[i]

        i += 1

        k += len(arr2)-j

    while j < len(arr2):

        arr[p+i+j] = arr2[j]

        j += 1

Текстовое объяснение решения.

Во время процесса слияния во время сортировки добавим к счетчику количество оставшихся элементов в смежном массиве

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.0001 sec

15.55 Mb

Пример из задачи

0.00015 sec

15.54 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.18916 sec

16.18 Mb

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

4 задача. Бинарный поиск

В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет

очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска.

• Формат входного файла (input.txt). В первой строке входного файла содержится число n (1 ≤ n ≤ 105) — число элементов в массиве, и последовательность a0 < a1 < ... < an−1 из n различных положительных целых чисел в порядке возрастания, 1 ≤ ai ≤ 109 для всех 0 ≤ i < n. Следующая строка содержит число k, 1 ≤ k ≤ 105 и k положительных целых чисел b0, ...bk−1, 1 ≤ bj ≤ 109 для всех 0 ≤ j < k.

• Формат выходного файла (output.txt). Для всех i от 0 до k − 1 вывести

индекс 0 ≤ j ≤ n − 1, такой что ai = bj или -1, если такого числа в массиве

нет.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб.

Листинг кода

def bs(arr, key):

    lt = -1

    rt = len(arr)

    while rt - lt > 1:

        m = (rt+lt) // 2

        if key > arr[m]:

            lt = m

        else:

            rt = m

    if rt < len(arr) and arr[rt] == key:

        return rt

    else:

        return -1

Текстовое объяснение решения.

Делим массив пополам и сравниваем центральное значение с ключом, а затем передвигаем границы поиска в зависимости от результата

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00015 sec

15.29 Mb

Пример из задачи

0.00025 sec

15.32 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.00028 sec

17.35 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости времени работы бинарного поиска O(logn) и длины массива k, O(k), а также используемую память.

5 задача. Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов a1, a2, ...an, и

нужно проверить, содержит ли она элемент, который появляется больше, чем n/2

Формат входного файла (input.txt). В первой строке входного файла содержится число n (1 ≤ n ≤ 105) — число элементов в массиве. Во второй

строке находятся n положительных целых чисел, по модулю не превосходящих 109, 0 ≤ ai ≤ 109

• Формат выходного файла (output.txt). Выведите 1, если во входной последовательности есть элемент, который встречается строго больше половины

раз; в противном случае - 0.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб

Листинг кода

def majority(arr, lt, rt):

    if lt == rt:

        return arr[lt]

    m = (lt+rt) // 2

    lt\_item = majority(arr, lt, m)

    rt\_item = majority(arr, m+1, rt)

    cnt\_lt\_item = arr[lt:m+1].count(lt\_item)

    cnt\_rt\_item = arr[m+1:rt+1].count(rt\_item)

    if lt == 0 and rt == len(arr)-1:

        if lt\_item != rt\_item:

            if max(cnt\_lt\_item,cnt\_rt\_item) > len(arr)//2:

                return 1

            else:

                return 0

        else:

            return 1

    else:

        if cnt\_lt\_item >= cnt\_rt\_item:

            return lt\_item

        else:

            return rt\_item

Текстовое объяснение решения.

Рекурсивно разбиваем массив на 2 части и проверяем на возврате каких элементов больше и каждый раз сохраняя “победителя” для следующего уровня рекурсии.

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00022 sec

15.40 Mb

Пример из задачи

0.00014 sec

15.54 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.03461 sec

16.92 Mb

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

6 задача. Поиск максимальной прибыли

Используя псевдокод процедур Find Maximum Subarray и Find Max Crossing

Subarray из презентации к Лекции 2 (страницы 25-26), напишите программу поиска максимального подмассива.

Примените ваш алгоритм для ответа на следующий вопрос. Допустим, у нас

есть данные по акциям какой-либо фирмы за последний месяц (год, или иной срок).

Проанализируйте этот срок и выдайте ответ, в какой из дней при покупке единицы

акции данной фирмы, и в какой из дней продажи, вы бы получили максимальную

прибыль? Выдайте дату покупки, дату продажи и максимальную прибыль.

Вы можете использовать любые данные для своего анализа. Например, я набрала в Google "акции" и мне поиск выдал акции Газпрома, тут - можно скачать информацию по стоимости акций за любой период. (Перейдя по ссылке, нажмите на вкладку "Настройки"→ "Скачать")

Соответственно, вам нужно только выбрать данные, посчитать изменение цены и применить алгоритм поиска максимального подмассива.

• Формат входного файла в данном случае на ваше усмотрение.

• Формат выходного файла (output.txt). Выведите название фирмы, рассматриваемый вами срок изменения акций, дату покупки и дату продажи

единицы акции, чтобы получилась максимальная выгода; и сумма этой прибыли.Текстовое объяснение решения.

Сохраняем изначальные данные с индексом, сортируем и выводим изначальные индексы полученных нулевого, последнего, среднего элементов.

Листинг кода:

def max\_subarray(arr, low, high):

    if low == high:

        return (low,high, arr[low])

    else:

        m = (low+high)//2

        lt\_low, lt\_high, lt\_sum = max\_subarray(arr,low,m)

        rt\_low, rt\_high, rt\_sum = max\_subarray(arr,m+1,high)

        crs\_low, crs\_high, crs\_sum = max\_cross\_subarray(arr,low,m,high)

        if lt\_sum >= rt\_sum and lt\_sum >= crs\_sum:

            return (lt\_low, lt\_high, lt\_sum)

        elif rt\_sum >= lt\_sum and rt\_sum >= crs\_sum:

            return (rt\_low, rt\_high, rt\_sum)

        else:

            return (crs\_low, crs\_high, crs\_sum)

def max\_cross\_subarray(arr, low, m, high):

    lt\_sum = rt\_sum = float('-inf')

    temp\_sum = 0

    for i in range(m, low-1,-1):

        temp\_sum += arr[i]

        if temp\_sum > lt\_sum:

            lt\_sum = temp\_sum

            mx\_lt = i

    temp\_sum = 0

    for i in range(m+1, high+1):

        temp\_sum += arr[i]

        if temp\_sum > rt\_sum:

            rt\_sum = temp\_sum

            mx\_rt = i

    return (mx\_lt, mx\_rt, lt\_sum + rt\_sum)

Текстовое объяснение решения.

Рекурсивно на возврате проверяем в какой части массива больше сумме: только в левой, на пересечении центра, только в правой.

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00017 sec

15.62 Mb

Пример из задачи

0.00014 sec

15.55 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.11067 sec

23.22 Mb

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

7 задача. Поиск максимального подмассива за линейное время

Можно найти максимальный подмассив за линейное время, воспользовавшись следующими идеями. Начните с левого конца массива и двигайтесь вправо, отслеживая найденный к данному моменту максимальный подмассив. Зная максимальный подмассив массива A[1..j], распространите ответ на поиск максимального подмассива, заканчивающегося индексом j + 1, воспользовавшись следующим наблюдением: максимальный подмассив массива A[1..j + 1] представляет собой либо максимальный подмассив массива A[1..j], либо подмассив A[i..j + 1] для некоторого 1 ≤ i ≤ j + 1. Определите максимальный подмассив вида A[i..j + 1] за константное время, зная максимальный подмассив, заканчивающийся индексом j.

Листинг кода

def max\_subarray\_linear(arr):  
 min\_pref = min(arr[0], 0)  
 max\_sum = arr[0]  
 cur\_sum = arr[0]  
 for i in range(1,len(arr)):  
 cur\_sum += arr[i]  
 max\_sum = max(max\_sum, cur\_sum - min\_pref)  
 min\_pref = min(cur\_sum, min\_pref)  
 return max\_sum

Текстовое объяснение решения.

Каждый раз находим минимальный префикс массива, а также перепроверяем максимум, как текущий префикс минус минимальный.

Результат работы на примере:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00808 sec

29.70 Mb

Пример из задачи

0.00916 sec

29.62 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.10914 sec

31.95 Mb

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

**Вывод**

• В этой лабораторной мы научились делать merge sort, binary search и различные вариации, а также находить max\_subarray несколькими способами

• Алгоритм O(n) выполняется достаточно быстро, относительно квадратичной сложности, затраты памяти также прямо пропорциональны линейной.

• С помощью методов *time.perf\_counter()* и *psutil.Process().memory\_info().rss* можно отслеживать ресурсозатратность алгоритмов.