Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 5

з дисципліни «Теорія алгоритмів»

на тему:

**"Стійке сортування"**

Виконав:

студент гр. ІС-02

Плостак Ілля

Викладач:

Cт. Вик. Новікова П. А.

Київ – 2021

**1. Завдання**

Відсортувати числа, які записані у десятковій системі та які мають d розрядів. Процес сортування виконувати за розрядами цих чисел. в Реалізувати алгоритм RedixSort.

Нехай є n d-значних чисел, в яких кожна цифра приймає одне з k можливих значень. Розробити алгоритм RadixSort, який дозволить виконати коректне сортування цих чисел за час Θ(d(n + k)). Використовувати стійке сортування, яке використовуються цим алгоритмом, який має час роботи Θ(n + k).Надати можливість вибору числа d.

**2. Програмний код:**

[main.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab5/main.py):

import random  
import algorithms as alg  
  
arr = [i for i in range(121, 141)]  
random.shuffle(arr)  
print("Stock:", arr)  
  
out = alg.RadixSort(arr, 3)  
print("Sorted:", out)

[algorithms.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab5/algorithms.py):

import math  
  
def RadixSort(A, d): # функция RadixSort принимает сортируемый массив и число разрядов  
 for i in range(1, d + 1): # от первого до последнего разряда  
 B = [0 for i in range(0, len(A))] # создаём массив в который будет возвращаться сортированный массив на каждом  
 k = MaxRankNum(A, i) # получаем длину будущего дополнительного массива  
 CountingSort(A, B, k, i) # выполняем стойкую сортировку на основе CountingSort  
 A = B.copy() # копируем сортированный массив в массив А  
 return A # возвращаем сортированный массив  
  
def CountingSort(A, B, k, d): # стойкая сортировка на основе CountingSort  
 C = [0 for i in range(0, k + 1)] # создаём дополнительный массив для подсчёта количества одинаковых элементов  
 for j in range(0, len(A)): # от 0 до длины массива A  
 C[GetNumOfRank(A[j], d)] += 1 # прибавляем 1 за каждый одинаковый элемент  
 for i in range(1, k + 1): # от 1 длины временного массива  
 C[i] += C[i - 1] # к каждому следующему прибавляем предыдущее число,  
 # поскольку каждому индексу соответствует элемент с тем же значением  
 for j in range(len(A) - 1, -1, -1): # от последнего элемента до нулевого в сортируемом массиве A  
 B[C[GetNumOfRank(A[j], d)] - 1] = A[j] # ставим на нужное место элемент A[j]  
 C[GetNumOfRank(A[j], d)] -= 1 # отнимаем от текущего массива индексов 1  
 # что бы если встретится ещё один такой же элемент  
 # поставить его не на то же место, а на соседнее  
  
def MaxRankNum(A, rank):  
 Max = GetNumOfRank(A[0], rank)  
 for i in range(1, len(A)):  
 if GetNumOfRank(A[i], rank) > Max:  
 Max = GetNumOfRank(A[i], rank)  
 return Max  
  
def GetNumOfRank(num, rank):  
 rank -= 1  
 num -= math.floor(num / math.pow(10, rank + 1)) \* math.pow(10, rank + 1)  
 num /= math.floor(math.pow(10, rank))  
 return int(num)

**3. Приклад вхідного файлу, результат запуску програми та приклад вихідного файлу:**

Результат запуску програми для перевірки чи сортують розроблені алгоритми:



**4. Висновок:**

В цій лабораторній роботі ми вивчили алгоритм стійкого сортування (на основі Counting Sort) та Radix Sort. Стійкий алгоритм має складність Θ(n+k), де k це довжина тимчасового масиву (додаткова пам’ять), а загалом увесь Radix Sort має складність Θ(d(n+k)), де d це кількість розрядів чисел у масиві. Ця комбінація алгоритмів сильно скорочує час роботи порівняно зі звичайним Counting Sort. У Counting Sort розмір додаткової пам’яті завжди залежить від висхідного масиву та може бути дуже великим, а у цій комбінації k <=9 завжди. На жаль на відміну від алгоритмів які базуються на порівняннях цей алгоритм не працює для чисел менших за 0 тому що сортує за розрядами не враховуючи наявність мінусу.

Вивід у програмі здійснюється у консоль.