Львівський національний університет імені Івана Франка
Факультет електроніки та комп’ютерних технологій

Звіт про виконання лабораторної роботи №7

Сортування підрахунком. Сортування за розрядами

Виконав:
Студент групи ФЕП-22
Серафим Д.В.

Львів-2022

Хід роботи:

Частина 1. Сортування підрахунком.

1. У бібліотеці Sort (створеній раніше), згідно описаного в теоретичній частині алгоритму, створити функцію CountingSort(…) для реалізації сортування методом підрахунку.

```
 def counting_sort(main_array):  # Функція CountingSort(…) для реалізації
сортування методом підрахунку.
    buffer_array = []
    array_exp = []
    for j in range(max(main_array) + 1):
        buffer_array.append(0)

    # print(buffer_array)

    for i in range(len(main_array)):
        buffer array[main array[i - 1]] += 1

    # print(buffer_array)

    for i in range(len(buffer_array)):
        if buffer_array[i] != 0:
            for j in range(buffer_array[i]):
                array_exp.append(i)
        else:
            continue

    return array_exp
```

2. Створити новий проект Lab_7_1 та підключити до нього бібліотеку Sort. У функції main() проекту реалізувати можливість введення одномірного масиву цілих чисел з обмеженнями та відображення результатів сортування.

```
from Sort import counting_sort, counting_sort_task_3, radix_sort,
radix_sort_task_3
from random import randint


print(10 // 3)

"""
Автоматичне введення в масив
"""


def auto_input():
    main_array = []
    number_of_elements = (input(f"Вкажіть кількість елементів в масиві: "))
    frame_min_number = 0
    frame_max_number = int(input(f"Вкажіть максимальне число яке може входити в
масив: "))
    for i in range(int(number_of_elements)):
        main_array.append(randint(frame_min_number, frame_max_number))

    return main_array


main_array = auto_input()
```

```
"""
PART 1
"""

print(f"Вхідний масив                                    : {main_array}")

print(f"Після сортування методом Підрахунку             :
{counting_sort(main_array)}")

print(f"Після сортування методом Підрахунку обернений:
{counting_sort_task_3(main_array)}")


"""
PART 2
"""
print("-----------------------------------------------")


print(f"Вхідний масив                                : {main_array}")

print(f"Після сортування за розрядами                : {radix_sort(main_array)}")

print(f"Після сортування за розрядами обернений:
{radix_sort_task_3(main_array)}")
```

3. Відкомпілювати проект та продемонструвати його роботу викладачеві. У функції CountingSort(…) змінити порядок проходження останнього циклу (зі спадання на зростання). Пояснити отриманий результат.

```
def counting_sort_task_3(main_array):  # Функція CountingSort(…) для реалізації
сортування методом підрахунку.
    # Зі зміненим порядком проходження останнього циклу (зі спадання на
зростання)
    buffer_array = []
    array_exp = []
    for j in range(max(main_array) + 1):
        buffer_array.append(0)

    # print(buffer_array)

    for i in range(len(main_array)):
        buffer_array[main_array[i - 1]] += 1

    # print(buffer_array)

    i = len(buffer_array) - 1
    while i >= 0:

        if buffer_array[i] != 0:

            for j in range(buffer_array[i]):
                array_exp.append(i)
                i -= 1

        else:
            i -= 1
            continue

    return array_exp
```

```
Вкажіть кількість елементів в масиві: 15
Вкажіть максимальне число яке може входити в масив: 25
Вхідний масив                              : [17, 13, 3, 13, 21, 20, 14, 5, 11, 16, 17, 13, 14, 8, 20]
Після сортування методом Підрахунку        : [3, 5, 8, 11, 13, 13, 13, 14, 14, 16, 17, 17, 20, 20, 21]
Після сортування методом Підрахунку обернений: [21, 20, 19, 17, 16, 14, 13, 11, 8, 5, 3]
```

В данному випадку якщо змінити порядок проходження останнього циклу , міняється напрям зростання масиву.

Частина 2. Сортування за розрядами.

1. У бібліотеці Sort, згідно описаного в теоретичній частині алгоритму,створити функцію RadixSort(…) для реалізації процедури сортування за розрядами. В якості алгоритму сортування окремого розряду обрати метод сортування підрахунками. Сортування здійснювати в циклі для кожного розряду окремо. Щоб отримати і-ий розряд n-розрядного числа потрібно цілочисельно поділити його на 10і-1та взяти остачу від ділення на 10.

```python
def counting_sort_for_radix(main_array, place_value):

    count_array = [0] * 10
    main_array_size = len(main_array)

    for i in range(main_array_size):
        place_element = (main_array[i] // place_value) % 10
        count_array[place_element] += 1

    for i in range(1, 10):
        count_array[i] += count_array[i - 1]

    array_exp = [0] * main_array_size  # Вихідний масив

    i = main_array_size - 1
    while i >= 0:
        current_element = main_array[i]
        place_element = (main_array[i] // place_value) % 10
        count_array[place_element] -= 1
        new_position = count_array[place_element]
        array_exp[new_position] = current_element
        i -= 1

    return array_exp


def radix_sort(main_array):

    max_element = max(main_array)  # Знаходить максимальний елемент у вхідному
масиві

    number_of_digits_in_max = len(str(max_element))  # Кількість цифр в
максимальному елементі

    place_value = 1

    output_array = main_array
    while number_of_digits_in_max > 0:
        output_array = counting_sort_for_radix(output_array, place_value)
        place_value *= 10
        number_of_digits_in_max -= 1
```

```
    return output_array
```

2. Створити новий проект Lab_7_2 та підключити до нього бібліотеку Sort. У функції main() проекту реалізувати можливість введення одномірного масиву n-розрядних цілих чисел та відображення результатів сортування.

```
from Sort import counting_sort, counting_sort_task_3, radix_sort,
radix_sort_task_3
from random import randint


print(10 // 3)

"""
Автоматичне введення в масив
"""


def auto_input():
    main_array = []
    number_of_elements = (input(f"Вкажіть кількість елементів в масиві: "))
    frame_min_number = 0
    frame_max_number = int(input(f"Вкажіть максимальне число яке може входити в
масив: "))
    for i in range(int(number_of_elements)):
        main_array.append(randint(frame_min_number, frame_max_number))

    return main_array


main_array = auto_input()

"""
PART 1
"""

print(f"Вхідний масив                                    : {main_array}")

print(f"Після сортування методом Підрахунку              :
{counting_sort(main_array)}")

print(f"Після сортування методом Підрахунку обернений:
{counting_sort_task_3(main_array)}")


"""
PART 2
"""
print("-------------------------------------------")


print(f"Вхідний масив                          : {main_array}")

print(f"Після сортування за розрядами          : {radix_sort(main_array)}")

print(f"Після сортування за розрядами обернений:
{radix_sort_task_3(main_array)}")
```

3. Відкомпілювати проект та продемонструвати його роботу викладачеві.

```
Вхідний масив                          : [17, 13, 3, 13, 21, 20, 14, 5, 11, 16, 17, 13, 14, 8, 20]
Після сортування за розрядами          : [3, 5, 8, 11, 13, 13, 13, 14, 14, 16, 17, 17, 20, 20, 21]
```

3.1.У процедурі сортування підрахунками змінити порядок проходження останнього циклу (зі спадання на зростання). Пояснити отриманий результат.

```
-------------------------------------------
Вхідний масив                          : [2, 13, 13, 8, 25, 22, 11, 17, 9, 10, 19, 1, 15, 17, 13]
Після сортування за розрядами          : [1, 2, 8, 9, 10, 11, 13, 13, 13, 15, 17, 17, 19, 22, 25]
Після сортування за розрядами обернений 3.1: [9, 8, 2, 1, 19, 17, 17, 15, 13, 13, 13, 11, 10, 25, 22]
Після сортування за розрядами обернений 3.2: [1, 2, 8, 9, 10, 11, 13, 13, 13, 15, 17, 17, 19, 22, 25]
-------------------------------------------
```

В данному випадку якщо змінити порядок проходження останнього циклу , міняється напрям зростання елементів кожного розряду

3.2.У функції RadixSort(…) змінити порядок проходження циклу вибору розряду (зі зростання на спадання). Пояснити отриманий результат.

```
-------------------------------------------
Вхідний масив                          : [2, 13, 13, 8, 25, 22, 11, 17, 9, 10, 19, 1, 15, 17, 13]
Після сортування за розрядами          : [1, 2, 8, 9, 10, 11, 13, 13, 13, 15, 17, 17, 19, 22, 25]
Після сортування за розрядами обернений 3.1: [9, 8, 2, 1, 19, 17, 17, 15, 13, 13, 13, 11, 10, 25, 22]
Після сортування за розрядами обернений 3.2: [1, 2, 8, 9, 10, 11, 13, 13, 13, 15, 17, 17, 19, 22, 25]
-------------------------------------------
```

В цьому випадку нічого не помінялось

Висновок: На цій лабораторній роботі я ознайомився з такими методами сортування: Сортування підрахунком. Сортування за розрядами

Весь код:

Main.py

```python
from Sort import counting_sort, counting_sort_task_3, radix_sort,
radix_sort_task_3, radix_sort_task_3_2
from random import randint


print(10 // 3)

"""
Автоматичне введення в масив
"""


def auto_input():
    main_array = []
    number_of_elements = (input(f"Вкажіть кількість елементів в масиві: "))
    frame_min_number = 0
    frame_max_number = int(input(f"Вкажіть максимальне число яке може входити в
масив: "))
    for i in range(int(number_of_elements)):
        main_array.append(randint(frame_min_number, frame_max_number))
```

```
    return main_array


main_array = auto_input()

"""
PART 1
"""

print(f"Вхідний масив                                  : {main_array}")

print(f"Після сортування методом Підрахунку            :
{counting_sort(main_array)}")

print(f"Після сортування методом Підрахунку обернений:
{counting_sort_task_3(main_array)}")


"""
PART 2
"""
print("-----------------------------------------------")


print(f"Вхідний масив                                : {main_array}")

print(f"Після сортування за розрядами                : {radix_sort(main_array)}")

print(f"Після сортування за розрядами обернений 3.1:
{radix_sort_task_3(main_array)}")

print(f"Після сортування за розрядами обернений 3.2:
{radix_sort_task_3_2(main_array)}")

print("-----------------------------------------------")
```

Sort.py

```
"""
PART 1
"""


def counting_sort(main_array):  # Функція CountingSort(…) для реалізації
сортування методом підрахунку.
    buffer_array = []
    array_exp = []
    for j in range(max(main_array) + 1):
        buffer_array.append(0)

    # print(buffer_array)

    for i in range(len(main_array)):
        buffer_array[main_array[i - 1]] += 1

    # print(buffer_array)

    for i in range(len(buffer array)):
        if buffer_array[i] != 0:
            for j in range(buffer_array[i]):
                array_exp.append(i)
        else:
            continue
```

```
    return array_exp


def counting_sort_task_3(main_array):  # Функція CountingSort(…) для реалізації сортування методом підрахунку.
    # Зі зміненим порядком проходження останнього циклу (зі спадання на зростання)
    buffer_array = []
    array_exp = []
    for j in range(max(main_array) + 1):
        buffer_array.append(0)

    # print(buffer_array)

    for i in range(len(main_array)):
        buffer_array[main_array[i - 1]] += 1

    # print(buffer_array)

    i = len(buffer_array) - 1
    while i >= 0:

        if buffer_array[i] != 0:

            for j in range(buffer_array[i]):
                array_exp.append(i)
                i -= 1

        else:
            i -= 1
            continue

    return array_exp


"""
PART 2
"""


def counting_sort_for_radix(main_array, place_value):

    count_array = [0] * 10
    main_array_size = len(main_array)

    for i in range(main_array_size):
        place element = (main array[i] // place value) % 10
        count_array[place_element] += 1

    for i in range(1, 10):
        count_array[i] += count_array[i - 1]

    array_exp = [0] * main_array_size  # Вихідний масив

    i = main_array_size - 1
    while i >= 0:
        current_element = main_array[i]
        place_element = (main_array[i] // place_value) % 10
        count_array[place_element] -= 1
        new_position = count_array[place_element]
        array_exp[new_position] = current_element
        i -= 1

    return array_exp
```

```python
def radix_sort(main_array):

    max_element = max(main_array)  # Знаходить максимальний елемент у вхідному масиві

    number_of_digits_in_max = len(str(max_element))  # Кількість цифр в максимальному елементі

    place_value = 1

    output_array = main_array
    while number_of_digits_in_max > 0:
        output_array = counting_sort_for_radix(output_array, place_value)
        place_value *= 10
        number_of_digits_in_max -= 1

    return output_array


def counting_sort_for_radix_task_3(main_array, place_value):

    count_array = [0] * 10
    main_array_size = len(main_array)

    for i in range(main_array_size):
        place_element = (main_array[i] // place_value) % 10
        count_array[place_element] += 1

    for i in range(1, 10):
        count_array[i] += count_array[i - 1]

    array_exp = [0] * main_array_size  # Вихідний масив

    i = 0
    while i <= main_array_size - 1:
        current_element = main_array[i]
        place_element = (main_array[i] // place_value) % 10
        count_array[place_element] -= 1
        new_position = count_array[place_element]
        array_exp[new_position] = current_element
        i += 1

    return array_exp


def radix_sort_task_3(main_array):

    max_element = max(main_array)  # Знаходить максимальний елемент у вхідному масиві

    number_of_digits_in_max = len(str(max_element))  # Кількість цифр в максимальному елементі

    place_value = 1

    output_array = main_array
    while number_of_digits_in_max > 0:
        output_array = counting_sort_for_radix_task_3(output_array, place_value)
        place_value *= 10
        number_of_digits_in_max -= 1

    return output_array


def counting_sort_for_radix_task_3_2(main_array, place_value):
```

```
    count_array = [0] * 10
    main_array_size = len(main_array)

    for i in range(main_array_size):
        place_element = (main_array[i] // place_value) % 10
        count_array[place_element] += 1

    for i in range(1, 10):
        count_array[i] += count_array[i - 1]

    array_exp = [0] * main_array_size  # Вихідний масив

    i = main_array_size - 1
    while i >= 0:
        current_element = main_array[i]
        place_element = (main_array[i] // place_value) % 10
        count_array[place_element] -= 1
        new_position = count_array[place_element]
        array_exp[new_position] = current_element
        i -= 1

    return array_exp


def radix_sort_task_3_2(main_array):

    max_element = max(main_array)  # Знаходить максимальний елемент у вхідному
масиві

    number_of_digits_in_max = len(str(max_element))  # Кількість цифр в
максимальному елементі

    place_value = 1

    output_array = main_array
    i = 0
    while i < number_of_digits_in_max:
        output_array = counting_sort_for_radix_task_3_2(output_array,
place_value)
        place_value *= 10
        i += 1

    return output_array
```