МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Сортировки

Студент гр. 9382 _____ Юрьев С.Ю.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить и реализовать сортировку данных, оценить ее достоинства и недостатки.

Задание.

17. Нитевидная сортировка.

Описание алгоритма.

В функцию сортировки передается неотсортированный список элементов. Внутри функции создаются 2 дополнительных списка: для временного хранения части списка (sub_list) и хранения итогового списка (result_list), который эта функция в итоге выведет. В функции происходит некоторое количество итераций по переданному списку (от 1 при уже отсортированном списке, вплоть до n итераций при списке отсортированном в обратную сторону).

На каждой итерации повторяются одни и те же действия:

1) Если начальный список не пустой: перенести 1-й элемент из начального списка (с удалением из него) в конец sub_list;

Иначе: закончить итерацию по начальному списку и закончить сортировку.

2) Начать обход по обновленному начальному списку с поиском элементов, которые >= последнего элемента из sub_list. При нахождении такового перенести этот элемент из начального списка (с удалением из него) в конец sub_list и продолжить итерацию на шаге 2).

3) При достижении конца начального списка присоединить упорядоченный sub_list к упорядоченному result_list (с очищением sub_list), чтобы получить упорядоченный result_list большей длины.

4) Начать новую итерацию с измененным начальным списком.

У нитевидной сортировки:

худшая сложность по времени = O(n^2)

средняя сложность по времени = O(n^2)

лучшая сложность по времени = O(n)

общая сложность по памяти = O(n)

Достоинства:

- стабильность сортировки (не меняет относительный порядок сортируемых элементов, имеющих одинаковые ключи, по которым происходит сортировка)
- маленькая лучшая сложность по времени (имеет минимальное значение для любой сортировки)

Недостатки:

- большая средняя и максимальная сложность по времени
- большая общая сложность по памяти

(очень простая и понятная сортировка, но все же лучше использовать иные виды, т.к. существуют сортировки в которых при тех же достоинствах меньше недостатков (cocktail shaker sort, gnome sort, cubesort и др.))

Описание функций.

template <typename T>
void print_list(std::list<T> entered_list)
Принимает на вход список, хранящий произвольный тип данных, ничего не возвращает.
Выводит элементы list.

template <typename T>
std::list<T> strandSort(std::list<T> lst)
Принимает на вход список, хранящий произвольный тип данных, возвращает так же список, хранящий произвольный тип данных
Сортирует переданный список и возвращает отсортированный

void print_empty_depth(int depth)
Принимает на вход число, ничего не возвращает.
Выводит табуляцию указанное количество раз

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 — Результаты тестирования

No	Входные данные	Выходные данные	Комментарии
1	5 4 3 2 1 0	Enter how many symbols will be in the list, OR enter 'q' for exit: 5 Enter all elements, OR enter 'q'	Список сортированный в обратном порядке (результаты

		for exit: 4 3 2 1 0 START sorting NEW ITERATION Start list = { 4 3 2 1 0 } Move first: '4' in sub_list. Now sub_list = { 4 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 4 } NEW ITERATION Start list = { 3 2 1 0 } Move first: '3' in sub_list. Now sub_list = { 3 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 3 4 } NEW ITERATION Start list = { 2 1 0 } Move first: '2' in sub_list. Now sub_list = { 2 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 2 3 4 } NEW ITERATION Start list = { 1 0 } Move first: '1' in sub_list. Now sub_list = { 1 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 1 2 3 4 } NEW ITERATION Start list = { 0 } Move first: '0' in sub_list. Now sub_list = { 0 } END of iteration. Merge sub_list	библиотечной и реализованной сортировок совпадают)

		with result_lisp. Now result_list = { 0 1 2 3 4 } STOP sorting List before strand sorting: { 4 3 2 1 0 } List after strand sorting: { 0 1 2 3 4 } List after std::list::sorting: { 0 1 2 3 4 } END OF PROGRAM	
2	1 2	Enter how many symbols will be in the list, OR enter 'q' for exit: 1 Enter all elements, OR enter 'q' for exit: 2 Only one element in array. List before strand sorting: { 2 } List after strand sorting: { 2 } List after std::list::sorting: { 2 } END OF PROGRAM	Список из одного элемента (результаты библиотечной и реализованной сортировок совпадают)
3	0	Enter how many symbols will be in the list, OR enter 'q' for exit: 0 Array is empty. List before strand sorting: { } List after strand sorting: { } List after std::list::sorting: { }	Список пуст (результаты библиотечной и реализованной сортировок совпадают)

		END OF PROGRAM	
4	4 2 0 5 3	Enter how many symbols will be in the list, OR enter 'q' for exit: 4 Enter all elements, OR enter 'q' for exit: 2 0 5 3 START sorting NEW ITERATION Start list = { 2 0 5 3 } Move first: '2' in sub_list. Now sub_list = { 2 } '5' >= '2' so move it in sub_list. Now sub_list = { 2 5 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 2 5 } NEW ITERATION Start list = { 0 3 } Move first: '0' in sub_list. Now sub_list = { 0 } '3' >= '0' so move it in sub_list. Now sub_list = { 0 3 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 0 2 3 5 } STOP sorting List before strand sorting: { 2 0 5 3 } List after strand sorting: { 0 2 3 5 } List after std::list::sorting: { 0 2 3 5 } END OF PROGRAM	Обычный случайный список (результаты библиотечной и реализованной сортировок совпадают)

4	3 567 50000 hghjghj 4378 53 9 -5 3 5676 432 101 293	Enter how many symbols will be in the list, OR enter 'q' for exit: 3 567 50000 hghjghj 4378 53 Enter all elements, OR enter 'q'	Ввод чрезмерного количества значений

		for exit: 9 -5 3 5676 432 101 293 START sorting NEW ITERATION Start list = { 9 -5 3 } Move first: '9' in sub_list. Now sub_list = { 9 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 9 } NEW ITERATION Start list = { -5 3 } Move first: '-5' in sub_list. Now sub_list = { -5 } '3' >= '-5' so move it in sub_list. Now sub_list = { -5 3 } END of iteration. Merge sub_list with result_lisp. Now result_list = { -5 3 9 } STOP sorting List before strand sorting: { 9 -5 3 } List after strand sorting: { -5 3 9 } List after std::list::sorting: { -5 3 9 } END OF PROGRAM	
5	1.999999 2.99999 3.9999	Enter how many symbols will be in the list, OR enter 'q' for exit: 1.999999 Enter all elements, OR enter 'q' for exit: 2.99999 3.9999 Only one element in array. List before strand sorting: { 2 } List after strand sorting: { 2 }	Ввод значений с плавающей точкой

		List after std::list::sorting: { 2 } END OF PROGRAM	
6	Char 5 1 k 2 3 4 5 2 3 k 5 90 0	Enter how many symbols will be in the list, OR enter 'q' for exit: char Input is invalid. Please, try again. Enter how many symbols will be in the list, OR enter 'q' for exit: 5 Enter all elements, OR enter 'q' for exit: 1 k 2 3 4 5 Element №1 is invalid. Please, try again. Enter remaining elements, OR enter 'q' for exit: 2 3 k 5 Element №3 is invalid. Please, try again. Enter remaining elements, OR enter 'q' for exit: 90 0 START sorting NEW ITERATION Start list = { 1 2 3 90 0 } Move first: '1' in sub_list. Now sub_list = { 1 } '2' >= '1' so move it in sub_list. Now sub_list = { 1 2 } '3' >= '2' so move it in sub_list. Now sub_list = { 1 2 3 } '90' >= '3' so move it in sub_list. Now sub_list = { 1 2 3 90 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 1 2 3 90 } NEW ITERATION	Ввод некорректных значений

		Start list = { 0 } Move first: '0' in sub_list. Now sub_list = { 0 } END of iteration. Merge sub_list with result_lisp. Now result_list = { 0 1 2 3 90 } STOP sorting List before strand sorting: { 1 2 3 90 0 } List after strand sorting: { 0 1 2 3 90 } List after std::list::sorting: { 0 1 2 3 90 } END OF PROGRAM	
7	2 q	Enter how many symbols will be in the list, OR enter 'q' for exit: 2 Enter all elements, OR enter 'q' for exit: q 'q' was entered. Finishing program...	Ввод значения для преждевременного завершения программы
8	-20 q	Enter how many symbols will be in the list, OR enter 'q' for exit: -20 Sorry, count of elements can`t be < 0. Please, try again. Enter how many symbols will be in the list, OR enter 'q' for exit: q 'q' was entered. Finishing program...	Ввод некорректной длины массива

Выводы.

Была изучена и реализована сортировка данных, были оценены ее достоинства и недостатки.

ПРИЛОЖЕНИЕ С КОДОМ

main.cpp :

```
#include "headers.h"

int main()
{
    using namespace std;
    setlocale(LC_ALL, "rus");

    list<int> start_list;
    int element = 0;
    short int count_of_elements = 0;
    char ch;

    { // считывание количества элементов и самих элементов
    while(true) // считывание, сколько будет введено элементов
    {
         cout << "Enter how many symbols will be in the list, OR enter 'q' for
exit:" << endl;

        ch = cin.peek();    // записываем следующий знак из cin в переменную, не
изменяя cin
        if (ch == 'q')      // проверка на ввод 'q'
        {
            cout << "\'q\' was entered. Finishing program..." << endl;
            return 0;       // выход из программы
        }

        cin >> count_of_elements; // пытаемся считать значение в int

        if (cin.fail()) // если это не 'q' и не число или же это слишком большое
число
        {
            cin.clear();            // переводим cin в обычный режим работы
            cin.ignore(32767,'\n'); // очищаем cin от остатков
            cout << "Input is invalid.  Please, try again.\n" << endl;
        }
        else
        {
            std::cin.ignore(32767,'\n');// удаляем лишние значения

               if (count_of_elements > 50) // проверяем значение на адекватные
размеры
                cout << "Sorry, count of elements can`t be more than 50. Please,
try again.\n" << endl;
            else if (count_of_elements < 0)
                   cout << "Sorry, count of elements can`t be < 0. Please, try
again.\n" << endl;
            else
                break;
        }
    }

    for (int i = 0; i < count_of_elements; ) // считывание самих элементов
    {
        if (i == 0)
            cout << "Enter all elements, OR enter 'q' for exit:" << endl;
```

```
        ch = cin.peek();    // записываем следующий знак из cin в переменную, не
изменяя cin
        if (ch == 'q')      // проверка на ввод 'q'
        {
            cout << "\'q\' was entered. Finishing program..." << endl;
            return 0;       // выход из программы
        }

        cin >> element;     // пытаемся получить число

        if (cin.fail())     // ошибка при получении числа
        {
            cin.clear();
            cin.ignore(32767,'\n');

             cout << "Element №" << i << " is invalid.  Please, try again.\n" <<
endl;
            if (i > 0)
                  cout << "Enter remaining elements, OR enter 'q' for exit:" <<
endl;
            continue;
        }
        else
        {
            start_list.push_back(element);
            i += 1;
        }

    }

    cout << endl;
    }

    list<int> strand_sorted_list = strandSort(start_list);

    cout << "List before strand sorting:\n";
    print_list(start_list);
    cout << '\n' << endl;

    cout << "List after strand sorting:\n";
    print_list(strand_sorted_list);
    cout << '\n' << endl;

    list<int> c = start_list;
    c.sort(); // стандартная сортировка std::list::sort

    cout << "List after std::list::sorting:\n";
    print_list(c);
    cout << '\n' << endl;

    cout << "END OF PROGRAM" << endl;
    return 0;

}
```

funcs.inl :

```
template <typename T>
void print_list(std::list<T> entered_list) // выведет все элементы list через
пробел
{
    using namespace std;
    cout << "{ ";
```

```
    for (auto pos_begin = entered_list.begin(); pos_begin != entered_list.end();
++pos_begin)
        cout << *pos_begin << " ";
    cout << '}';
    return;
}

template <typename T>
std::list<T> strandSort(std::list<T> lst) // нитевидная сортировка
{
    using namespace std;

    if (lst.size() == 0) // если передан пустой массив
    {
        cerr << "Array is empty." << endl;
        return lst;
    }
    else if (lst.size() == 1) // если передан массив из одного элемента
    {
        cerr << "Only one element in array." << endl;
        return lst;
    }

    cout << "START sorting" << endl;

    list<T> result_list;    // для готового результата
    list<T> sub_list;       // для вычислений
    int depth_counter = 1;  // для вывода отладочной информации

    while (!lst.empty())
    {
        { // отладочная информация
        print_empty_depth(depth_counter);
        cout << "NEW ITERATION" << endl;

        print_empty_depth(depth_counter);
        cout << "Start list = ";
        print_list(lst);
        cout << endl;

        print_empty_depth(depth_counter);
        cout << "Move first: \'" << *(lst.begin()) << "\' in sub_list." << endl;
        }

              sub_list.push_back(lst.front());//  выписываем  первый  элемент  из
начального списка
        lst.pop_front();                // удаляем первый элемент из начального
списка

        {  // отладочная информация
            print_empty_depth(depth_counter);
            cout << "Now sub_list = ";
            print_list(sub_list);
            cout << endl;
            depth_counter += 1;
        }

         for (typename list<T>::iterator it = lst.begin(); it != lst.end(); ) //
итератор будет увеличиваться внутри
        {
             if (sub_list.back() <= *it)  // если последний добавленный элемент
<= элемента на котором находится итератор
            {
```

```
                { // отладочная информация
                print_empty_depth(depth_counter);
                 cout << "\'" << *it << "\' >= \'" << sub_list.back() << "\' so
move it in sub_list." << endl;
                }

                sub_list.push_back(*it); // то добавляем его в конец
                it = lst.erase(it);      // и удаляем этот элемент из начального
массива

                {  // отладочная информация
                print_empty_depth(depth_counter);
                cout << "Now sub_list = ";
                print_list(sub_list);
                cout << endl;
                depth_counter += 1;
                }
            }
            else
                it++;
        }
        result_list.merge(sub_list); // смержили sub_list с result_list и теперь
sub_list пуст (merge объединяет 2 сортированных списка в один)

        { // отладочная информация
        print_empty_depth(depth_counter);
        cout << "END of iteration." << endl;

        print_empty_depth(depth_counter);
        cout << "Merge sub_list with result_lisp." << endl;

        print_empty_depth(depth_counter);
        cout << "Now result_list = ";
        print_list(result_list);
        cout << endl;
        depth_counter += 1;
        }
    }

    cout << "STOP sorting\n\n" << endl;
    return result_list;
}

void print_empty_depth(int depth) // выведет "  " указанное кол-во раз
{
    for (int j = 0; j < depth; j++)
        std::cout << "   ";
    return;
}
```

headers.h :

```
#ifndef HEADERS_H
#define HEADERS_H


#include <list>
#include <iostream>
#include <algorithm>
```

```
#include <fstream>

void print_empty_depth(int depth);

#include "funcs.inl"

#endif
```