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. 9381 Щеглов Д.А.

Преподаватель Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить и реализовать сортировку данных, обозначить её плюсы и минусы.

Задание.

17. Нитевидная сортировка.

Основные теоретические положения.

В данном методе приходится постоянно удалять и вставлять элементы, поэтому он достаточно оптимален при работе с двусвязными списками. Функция сортировки получает на вход список элементов, которые изначально расположены в произвольном порядке. В самой функции создаётся ещё 2 списка. Один предназначен для временного хранения части списка, а другой для записи в него отсортированного списка, который уже будет являться результатом нитевидной сортировки. Далее, в зависимости от содержимого изначального списка, в функции происходит некоторое количество итераций: 1 – если изначальный список оказался уже отсортирован, n – если отсортирован в обратном порядке. Поэтому, временная сложность нитевидной сортировки достаточно скромна – в среднем O(n2). Однако весьма эффективна при работе с почти упорядоченными списками – O(n).

Описание алгоритма.

На каждой итерации происходят следующее действия:

1. Первый элемент изначального списка переносится в конец списка для хранения элементов и удаляется из изначального списка.
2. Далее следует обход обновленного изначального списка. Если там удаётся найти элемент, который больше или равен последнему элементу в списке хранения, то этот элемент переносится в конец списка хранения и удаляется из текущего изначального списка. После чего происходит переход на следующую итерацию.

3. Если итератор достигает конца изначального списка, то упорядоченный список хранения присоединяется к итоговому списку и очищается. После этого начинается новая итерация с изменённым начальным списком.

Описание функций.

template <typename T>
void print_list(list<T> started_list)

Функция для вывода элементов списка.

template <typename T>
list<T> strandSort(list<T> lst)

Функция нитевидной сортировки. Работает с произвольным типом данных. Возвращает отсортированный список.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	4 2 6 8 9	Начало сортировки Новая итерация: Изначальный список = { 2, 6, 8, 9, } Помещаем '2' в упорядоченный подсписок Упорядоченный подсписок = { 2, } '6' >= '2' поэтому перемещаем '6' в упорядоченный подсписок Упорядоченный подсписок = { 2, 6, } '8' >= '6' поэтому перемещаем '8' в упорядоченный подсписок	Если передать изначально отсортированный список, то алгоритм проходит всего одну итерацию.

		Упорядоченный подсписок = { 2, 6, 8, } '9' >= '8' поэтому перемещаем '9' в упорядоченный подсписок Упорядоченный подсписок = { 2, 6, 8, 9, } Конец итерации Соединяем списки: Получаем список { 2, 6, 8, 9, } Конец сортировки Список до сортировки: { 2, 6, 8, 9, } Список после сортировки: { 2, 6, 8, 9, }	
2.	4 17 11 3 -2	Начало сортировки Новая итерация: Изначальный список = { 17, 11, 3, -2, } Помещаем '17' в упорядоченный подсписок Упорядоченный подсписок = { 17, } Конец итерации Соединяем списки: Получаем список { 17, }	Если передать изначально отсортированный в обратном порядке список, то алгоритм пройдёт n итераций. В данном тесте n = 4.

		Новая итерация: Изначальный список = { 11, 3, -2, } Помещаем '11' в упорядоченный подсписок Упорядоченный подсписок = { 11, } Конец итерации Соединяем списки: Получаем список { 11, 17, } Новая итерация: Изначальный список = { 3, -2, } Помещаем '3' в упорядоченный подсписок Упорядоченный подсписок = { 3, } Конец итерации Соединяем списки: Получаем список { 3, 11, 17, } Новая итерация: Изначальный список = { -2, } Помещаем '-2' в упорядоченный подсписок Упорядоченный подсписок = { -2, } Конец итерации Соединяем списки:	

		Получаем список { -2, 3, 11, 17, } Конец сортировки Список до сортировки: { 17, 11, 3, -2, } Список после сортировки: { -2, 3, 11, 17, }	
3.	1 800	Начало сортировки Новая итерация: Изначальный список = { 800, } Помещаем '800' в упорядоченный подсписок Упорядоченный подсписок = { 800, } Конец итерации Соединяем списки: Получаем список { 800, } Конец сортировки Список до сортировки: { 800, } Список после сортировки: { 800, }	Список из одного элемента выводится в консоль.
4	10 78 97 87	Начало сортировки Новая итерация:	Проверяем работу алгоритма на

	0 19 200 -7 -32 568 4	Изначальный список = { 78, 97, 87, 0, 19, 200, -7, -32, 5, 68, } Помещаем '78' в упорядоченный подсписок Упорядоченный подсписок = { 78, } '97' >= '78' поэтому перемещаем '97' в упорядоченный подсписок Упорядоченный подсписок = { 78, 97, } '200' >= '97' поэтому перемещаем '200' в упорядоченный подсписок Упорядоченный подсписок = { 78, 97, 200, } Конец итерации Соединяем списки: Получаем список { 78, 97, 200, } Новая итерация: Изначальный список = { 87, 0, 19, -7, -32, 5, 68, } Помещаем '87' в упорядоченный подсписок Упорядоченный подсписок = { 87, } Конец итерации Соединяем списки: Получаем список { 78, 87, 97, 200, }	обычном списке, в котором элементы расставлены в случайном порядке.

		Новая итерация: Изначальный список = { 0, 19, -7, -32, 5, 68, } Помещаем '0' в упорядоченный подсписок Упорядоченный подсписок = { 0, } '19' >= '0' поэтому перемещаем '19' в упорядоченный подсписок Упорядоченный подсписок = { 0, 19, } '68' >= '19' поэтому перемещаем '68' в упорядоченный подсписок Упорядоченный подсписок = { 0, 19, 68, } Конец итерации Соединяем списки: Получаем список { 0, 19, 68, 78, 87, 97, 200, } Новая итерация: Изначальный список = { -7, -32, 5, } Помещаем '-7' в упорядоченный подсписок Упорядоченный подсписок = { -7, } '5' >= '-7' поэтому перемещаем '5' в упорядоченный подсписок Упорядоченный подсписок = { -7, 5, }	

		Конец итерации Соединяем списки: Получаем список { -7, 0, 5, 19, 68, 78, 87, 97, 200, } Новая итерация: Изначальный список = { -32, } Помещаем '-32' в упорядоченный подсписок Упорядоченный подсписок = { -32, } Конец итерации Соединяем списки: Получаем список { -32, -7, 0, 5, 19, 68, 78, 87, 97, 200, } Конец сортировки Список до сортировки: { 78, 97, 87, 0, 19, 200, -7, -32, 5, 68, } Список после сортировки: { -32, -7, 0, 5, 19, 68, 78, 87, 97, 200, }	

Выводы.

Была изучена и реализована нитевидная сортировка. Разобраны преимущества и недостатки данного метода сортировки.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: source.cpp

```
#include<iostream>
#include<fstream>
#include<list>
#include<algorithm>
#include<stdlib.h>
#include<locale>

using namespace std;

template <typename T>
void print_list(list<T> started_list) // функция для вывода всех
элементов списка
{
    cout << "{ ";
    for (auto start_pos = started_list.begin(); start_pos !=
started_list.end(); start_pos++)
        cout << *start_pos << ", ";
    cout << '}';
}

template <typename T>
list<T> strandSort(list<T> lst) // Нитевидная сортировка
{

    cout << "Начало сортировки\n";

    list<T> result_list;    // отсортированный список
    list<T> sorted_list;       // промежуточный результат

    while (!lst.empty())
    {
        { // отладочная информация
            cout << "Новая итерация:\n Изначальный список = ";
            print_list(lst);
            cout << "\n Помещаем \'" << *(lst.begin()) << "\' в
упорядоченный подсписок\n";
        }

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

        {  // отладочная информация
            cout << " Упорядоченный подсписок = ";
            print_list(sorted_list);
            cout << "\n";
        }
```

```
        for (typename list<T>::iterator iter = lst.begin(); iter
!= lst.end(); )
        {
            if (sorted_list.back() <= *iter)  // если последний
добавленный элемент меньше или
            {                                 // равен элементу на
котором находится итератор
                { // отладка
                    cout << " \'" << *iter << "\' >= \'" <<
sorted_list.back() << "\' поэтому перемещаем" << " '"<<*iter <<"'
" << "в упорядоченный подсписок" << endl;
                }
                sorted_list.push_back(*iter); // добавляем этот
элемент в конец
                iter = lst.erase(iter);        // и удаляем его из
начального массива

                {  // отладка
                    cout << " Упорядоченный подсписок = ";
                    print_list(sorted_list);
                    cout << "\n" ;
                }
            }
            else
                iter++;                  // не забываем увеличить
итератор
        }
        result_list.merge(sorted_list); // объединяем два
отсортированных списка в один
        //отладка
        {
            cout << "Конец итерации\n\n Соединяем списки:\n
Получаем список ";
            print_list(result_list);
            cout << "\n\n";
        }
    }
    cout << "Конец сортировки\n" << endl;
    return result_list;
}

int main()
{
    setlocale(LC_ALL, "rus");
    list<int> start_list;
    int elem = 0;
    int count_of_elem = 0;
    bool flag = true;

    cout << "Введите количество элементов списка:" << endl;
    while (flag == true) { //считываем количество элементов списка
        cin >> count_of_elem;
```

```
        if (count_of_elem > 30 || count_of_elem < 1 )
        {
            cout << "Введите корректное количество элементов
списка!" << endl;
            continue;
        }
        flag = false;
    }

    for (int i = 0; i < count_of_elem; i++) // считываем элементы
списка
    {
        if (i == 0)
            cout << "Через пробел введите все элементы списка:\n";
        cin >> elem;    // считываем число
        start_list.push_back(elem);
    }

    cout <<"\n";
    list<int> strand_sorted_list = strandSort(start_list);
    cout << "Список до сортировки:\n";
    print_list(start_list);
    cout << "\n\n";
    cout << "Список после сортировки:\n";
    print_list(strand_sorted_list);
    return 0;
     }
```