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»

Студентка гр. 9381 Андрух И.А.

Преподаватель Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Ознакомиться с алгоритмом трёхпутевой сортировки и создать программу для реализации этого алгоритма.

Задание

Вариант 10.

Быстрая сортировка, рекурсивная реализация. Процедура трёхчастного разделения. Деление производится не на две группы, а на три: <x, =x, >x.

Описание алгоритма.

Программа запрашивает у пользователя путь до файла. Если файл не может быть открыт, выводится предупреждение. Из файла считывается строка чисел, числа записываются в элемент типа array_list list, в поле array. Введенные числа выводятся на экран для проверки.

Далее вызывается функция сортировки qsort3way(s, list, 0, list.size() - 1,0).

Трёхпутевая сортировка осуществляется следующим образом: выбирается опорный элемент(в нашем случае - pivot, генерируется случайным образом через rand()), массив чисел делится на три части. В центральной части находятся элементы, равные опорному, в левой – элементы, меньшие, чем опорный, а в правой – большие. Функция сортировки вызывается рекурсивно для левой и правой частей.

Перед рекурсивным вызовом выводятся промежуточные значения: крайние индексы сортируемого интервала, выбранный опорный элемент, глубина рекурсии.

Сложность алгоритма быстрой сортировки с процедурой трёхчастного разделения в худшем случае такая же, как и у обычной быстрой сортировки – O(n^2). Преимущество данной сортировки в том, что при малой частоте вариативности элементов сокращается количество итераций относительно быстрой сортировки с делением пополам.

В наиболее сбалансированном варианте при каждой операции разделения массив делится на две примерно одинаковые части, следовательно, сложность алгоритма будет равна O(n*$log_2 n$).

В самом несбалансированном варианте каждое разделение дает два подмассива размерами 1 и n-1, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых. Общее время работы составит O(n^2).

Описание структур данных и функций.

Для хранения чисел была создана структура array_list. Поля структуры:

- T* array – массив чисел;
- int capacity – размерность массива array;
- int count – количество элементов, записанных в массив array;

Для работы со списком использовались функции:

- void resize(int new_capacity) – изменяет размерность массива;
- array_list<T>::array_list<T>(int start_capacity) – создает массив заданной размерности без записанных туда значений;
- T& array_list<T>::operator[] (int index) – позволяет обращаться к элементам поля array через объект типа array_list по индексу;
- void array_list<T>::push_back(T element) – добавление элемента в конец массива;
- int array_list<T>::size() – получение длины списка

Для логгирования, вывода промежуточных данных, была создана функция string log(array_list &list, int min, int max, T pivot, int depth).

Для реализации трёхпутевой сортировки чисел была создана функция void qsort3way(string& s, array_list<T>& list, int l, int r, int depth).

Для вывода результата сортировки создана функция std::string print_list(array_list<T>& list).

Тестирование программы.

Было создано несколько тестов для проверки работы программы. (см. Приложение А).

Вывод.

Был изучен принцип трёхпутевой быстрой сортировки и реализован алгоритм этой сортировки на языке с++.

ПРИЛОЖЕНИЕ А

Тестирование.

Входные данные	Выходные данные
12 355 1 34 87 14 0 -56 32 71	12 355 1 34 87 14 0 -56 32 71 Работа алгоритма: Логгирование 12 1 34 87 14 0 -56 32 71 355 индекс левого элемента: 0 индекс правого элемента: 9 опорный элемент: 355 глубина рекурсии: 0 -56 34 87 14 0 1 32 71 12 355 индекс левого элемента: 0 индекс правого элемента: 8 опорный элемент: -56 глубина рекурсии: 1 -56 12 1 0 14 32 71 87 34 355 индекс левого элемента: 1 индекс правого элемента: 8 опорный элемент: 14 глубина рекурсии: 2 -56 0 1 12 14 32 71 87 34 355 индекс левого элемента: 1 индекс правого элемента: 3 опорный элемент: 1 глубина рекурсии: 3 -56 0 1 12 14 32 71 34 87 355 индекс левого элемента: 5 индекс правого элемента: 8 опорный элемент: 87 глубина рекурсии: 3 -56 0 1 12 14 32 34 71 87 355 индекс левого элемента: 5 индекс правого элемента: 7 опорный элемент: 71 глубина рекурсии: 4 -56 0 1 12 14 32 34 71 87 355 индекс левого элемента: 5

	индекс правого элемента: 6 опорный элемент: 32 глубина рекурсии: 5 Конец промежуточных значений Ответ: -56 0 1 12 14 32 34 71 87 355
6.14 6.13 6.22 6.12	6.140000 6.130000 6.220000 6.120000 Работа алгоритма: Логгирование 6.120000 6.130000 6.220000 6.140000 индекс левого элемента: 0 индекс правого элемента: 3 опорный элемент: 6.130000 глубина рекурсии: 0 6.120000 6.130000 6.140000 6.220000 индекс левого элемента: 2 индекс правого элемента: 3 опорный элемент: 6.220000 глубина рекурсии: 1 Конец промежуточных значений Ответ: 6.120000 6.130000 6.140000 6.220000
23 12	Работа алгоритма: Логгирование 12 23 индекс левого элемента: 0 индекс правого элемента: 1 опорный элемент: 23 глубина рекурсии: 0 Конец промежуточных значений

	Ответ: 12 23
	Работа алгоритма: Логгирование Конец промежуточных значений Ответ:

ПРИЛОЖЕНИЕ Б

Исходный код программы.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <sstream>
using  namespace std;

template<typename T>
struct array_list
{
    T* array;  //массив чисел
    int capacity;//размерность массива array
    int count; //количество непустых элементов массива array

    void resize(int new_capacity);
    array_list(int start_capacity=1);
    T& operator[] (int index);

    void push_back(T element);
    int size();
};
template<typename T>
void array_list<T>::resize(int new_capacity) //изменение размерности
массива
{
    T *arr = new T[count];
    for (int i = 0; i < count; ++i)
    {
        arr[i] = array[i];  //переписываем значения массива в
вспомогательный
    }
    delete [] array;         //удаляем старый массив
    array = new T[new_capacity]; //выделяем массив нужного размера
    for (int i = 0 ; i < count; ++i)
    {
        array[i] = arr[i];    //переписываем значения в нужный массив
    }
    delete [] arr;        //очищаем память под вспомогательный массив
    capacity = new_capacity;    //изменяем поле класса - размерность
массива
}
template<typename T>
array_list<T>::array_list(int start_capacity)  //создание списка чисел
{
    capacity = start_capacity;
    count = 0;
    array = new T[capacity];
}
template<typename T>
```

```
T& array_list<T>::operator[] (int index)  //функция позволяет обращаться к
элементам поля array через объект типа array_list по индексу
{
    return array[index];
}

template<typename T>
void array_list<T>::push_back(T element) //добавление элемента в конец
массива
{
    if (capacity == count)
    {
        resize(count + 8);  //изменяем размер массива
    }
    array[count] = element;  //вставляем элемент
    count++;
}
template<typename T>
int array_list<T>::size() //получение длины списка
{
    return count;
}


template<typename T>
string log(array_list<T> &list, int min, int max, T pivot, int depth)
//функция для вывода промежуточного результата
{
    string s = ""; //строка для вывода промежуточных данных
    for (int i = 0; i < list.size(); i++)
    {
        s += to_string(list[i]) + ' ';//вывод элементов
    }
    s += '\n';
    s += "\tиндекс левого элемента: " + to_string(min) + "\n\tиндекс
правого элемента: " + to_string(max) + "\n\tопорный элемент: " +
to_string(pivot) + "\n\tглубина рекурсии: " + to_string(depth) +"\n\n";
//добавляем к строке промежуточные сведения
    return s;
}

template<typename T>
void qsort3way(string& s, array_list<T>& list, int l, int r, int depth)
//функция трёхпутевой сортировки
{
    if (l >= r)
    {
        return;
    }
    int lt = l; // lt - индекс, по которому запишется очередное значение,
меньше опорного
    int gt = r;  //gt - индекс, по которому запишется очередное значение,
больше опорного
    T pivot = list[l + (rand() % (r - l))];//опорный элемент генерируется
рандомным образом
    int i = l; //проходим по массиву слева направо
    while (i <= gt)
    {
        if (list[i] < pivot)  //если значение меньше чем опорный элемент,
записываем его по индексу lt, lt смещаем вправо
        {
            T t = list[lt];
            list[lt] = list[i];
            list[i] = t;
            lt += 1;
            i += 1;
        }
        else if (list[i] > pivot)  //если значение больше чем опорный
элемент, записываем его по индексу gt, gt смещаем влево
        {
            T t = list[gt];
            list[gt] = list[i];
            list[i] = t;
            gt -= 1;
        }
        else
        {
            i += 1;
        }
```

```
    }
    s += log(list, l, r, pivot, depth);
    qsort3way(s, list, l, lt - 1, depth + 1);  //рекурсивный вызов для
левой части
    qsort3way(s, list, gt + 1, r, depth + 1);  //рекурсивный вызов для
правой части
}



template<typename T>
std::string print_list(array_list<T>& list)  //вывод итогового результата
{
    string s;
    for (int i = 0; i < list.size(); i++)
    {
        s += std::to_string(list[i]) + " ";
    }
    return s;
}

int main() {
    string input;
    ifstream file;
    string filename;
    cout << "Введите путь до файла\n";
    cin >> filename;
    file.open(filename); //открываем файл
    if (!file.is_open()){
        cout << "Файл не может быть открыт!\n";
    }
    getline(file,input); //считываем строку из файла
    file.close();            //закрываем файл
    array_list<int> list = array_list<int>();
    int elem;
    istringstream str(input);
    while(str >> elem) list.push_back(elem);
    cout << print_list(list);
    cout << endl << "Работа алгоритма:" << endl;
    string s;
    qsort3way(s, list, 0, list.size() - 1,0);
    cout << "Логгирование" << endl << s << endl << "Конец промежуточных
значений" << endl;
    cout << endl << "Ответ: " << endl << print_list(list) << endl;
    return 0;
}
```