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 _____ Игнашов В.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Реализовать алгоритм сортировки. Использовать шаблоны для сортировки разных типов данных.

Задание.

21. Соломонова сортировка.

Выполнение работы.

В коде используется структура данных вектора, подключаемая библиотекой vector, которая является аналогом динамического массива с методами взаимодействия с ним.

Алгоритм использует принципы приблизительного распределения элемента с последующей вставкой:

1. Найти минимум, максимум в массиве, на их базе высчитывается дельта:
2. «Время разбрасывать камни» - С помощью вычисленного значения дельты приблизительно высчитывается должное положение элемента: $NewIndex(A_i) = \frac{A_i - A_{min}}{delta} + 1$
3. «Время собирать камни» - После предыдущего шага на каждой позиции в итоговом массиве могут лежать несколько элементов. В таким случае необходимо отсортировать между собой и последовательно склеить все получившиеся мини-блоки.

Сложность алгоритма(в худшем случае) – $O(N^2)$

Сложность алгоритма(в лучшем случае) – $O(N)$

*void sortSolomon(vector<T>*arr):*

Входные данные - вектор шаблонных значений arr.

Этап 1: Создание дельты.

В цикле, проходясь по всему массиву вычисляем наименьшее и наибольшее значение, сравнивая с прошлыми значениями в шаблонных min и max.

Вычисляем значение дельты по формуле.

Создаем вектор векторов, для записи в каждый элемент нескольких элементов исходного массива и переходим к этапу 2.

Этап 2: Время раскидывать камни

Задаем массиву размер исходного массива+2, чтобы не выйти за границы при неожиданных значениях в массиве.

Для каждого элемента в исходном массиве высчитываем его примерную позицию в конечном массиве и записываем его в вектор, содержащийся на этой позиции, фиксируя все промежуточные результаты.

Переходим к этапу 3.

Этап 3: Время собирать камни.

Очищаем исходный массив, чтобы заново перезаписать в него значения в правильном порядке.

Двигаемся по временному вектору и для каждой позиции сортируем стандартно пузырьком те значения, что лежат на этой позиции. Тем самым пройдя по всему массиву мы будем иметь мини-блоки, в котором начальный элемент следующего блока больше последнего в предыдущем, а сами блоки отсортированы, в таком случае склеиваем получившиеся блоки, записывая подряд все значения в главный массив.

В главной функции программы происходит выбор ввода и считывание из него исходного массива, он записывается в два вектора, один – используемый в нашей сортировке, второй – используемый в сортировке методом стандартной библиотеки.

Исходный массив выводится. После чего при условии, что в исходном массиве 1 или 0 элементов – выводим, что нет необходимости запускать сортировку, иначе запускаем сортировку с помощью функции void sortSolomon(vector<T>*). Выводим получившийся после сортировки массив и, отсортировав такой же второй массив стандартным методом – выводим для сравнения.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении В.

Выводы.

Был реализован алгоритм Соломоновой сортировки, также доступной для всех типов значений, которые могут быть преобразованы в числовые значения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
#include <fstream>

using namespace std;

template<typename T>
void sortSolomon(vector<T> *arr){//Функция сортировки
    //ЭТАП 1 создание дельты
    T min=(*arr)[0];
    T max=(*arr)[0];

    for(int i=0;i<arr->size();i++){//Находим наименьший и наибольший элемент
        if((*arr)[i]<min)
            min=(*arr)[i];
        if((*arr)[i]>max)
            max=(*arr)[i];
    }
    cout << "Найдены минимальное и максимальное значения в массиве - " << min <<
" и " << max <<"\n";
    float delta=(float)(max-min)/arr->size();//Высчитываем дельту
    cout << "Высчитана delta по формуле (max-min)/N = " << delta << "\n";

    vector<vector<T>> tmp;
    //Этап 2, время разбрасывать камни
    tmp.resize(arr->size()+2);
    for(int i=0;i<arr->size();i++){
        int curPos=((*arr)[i]-min)/delta+1;//Высчитываем дельту для каждого
элемента и распределяем в соответсвующие клетки
        cout << i << "-й элемент распределен в " << curPos << " клетку\n";
        tmp[curPos].push_back((*arr)[i]);
        cout << "Количество элементов во временном векторе векторов:\n";
        for(int j=1;j<tmp.size();j++){
            cout << tmp[j].size() << " ";
        }
        cout << "\n\n";
    }
    cout << "По итогу имеем следующий временный вектор:\n";
    for(int i=1;i<tmp.size();i++){//Итоговый вектор векторов
        cout << tmp[i].size() << ": ";
        for(int j=0;j<tmp[i].size();j++){
            cout << tmp[i][j] << " ";
        }
        cout << "\n";
    }
    //Этап 3, время собирать камни
    arr->clear();
    cout << "Последовательно загоняем блоки, отсортировав их:\n";
    for(int i=0;i<tmp.size();i++){//Загоняем блоки, сортируя их, ибо по-разному
могли добавиться в список определенного элемента
        if(tmp[i].size()==0)
            continue;
        //Стандартная сортировка пузырьком мини-блоков
        for(int j=0;j<tmp[i].size();j++)
            for(int k=j;k<tmp[i].size();k++)
                if(tmp[i][j]>tmp[i][k]) {
                    T t = tmp[i][j];
```

```
                    tmp[i][j] = tmp[i][k];
                    tmp[i][k] = t;
                }
        for(int j=0;j<tmp[i].size();j++){
            arr->push_back(tmp[i][j]);
        }
        for(int j=0;j<arr->size();j++){
            cout << (*arr)[j] << " ";//Промежуточные прогоны блоков внутрь
массива
        }
        cout << "\n";
    }
}

int main() {
    //Ввод входной строки
    int choose=0;
    cout << "Choose input:\n0.CIN\n1.FILE\n";
    cin >> choose;//Выбираем между вводом с консоли и из файла
    getchar();
    vector<int> arr;
    vector<int> arr2;
    string x;
    if(choose==0){
        getline(cin,x);
        if(!std::atoi(x.c_str())){
            cout << "Wrong input!";
            return 0;
        }
        while(!x.empty()){
            arr.push_back(std::atoi(x.c_str()));
            arr2.push_back(std::atoi(x.c_str()));
            getline(cin,x);
        }
    }else {
        ifstream ff;
        ff.open("Lab4Input.txt");
        if (ff.is_open()) {
            getline(ff, x);
            if(!std::atoi(x.c_str())){
                cout << "Wrong input!";
                return 0;
            }
            while(!x.empty()){
                arr.push_back(std::atoi(x.c_str()));
                arr2.push_back(std::atoi(x.c_str()));
                getline(ff, x);
            }
        }
        else
            return 0;
    }

    cout << "Исходный порядок:\n";
    for(int i=0;i<arr.size();i++){
        cout << arr[i] << " ";
    }
    if(arr.size()>=2) {
        cout << "\nЗапуск сортировки:\n\n";
        sortSolomon(&arr);
    }else{
        cout << "Нет необходимости запускать сортировку\n";
    }

    cout << "Отсортированный порядок:\n";
    for(int i=0;i<arr.size();i++){
```

```
        cout << arr[i] << " ";
    }

    sort(arr2.begin(),arr2.end());
    cout << "\nОтсортированный библиотечным методом порядок:\n";
    for(int i=0;i<arr2.size();i++) {
        cout << arr2[i] << " ";
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные
1.	3 5 3 7 8 4 5 0 3 8 2 6	Исходный порядок: 3 5 3 7 8 4 5 5 0 3 8 2 6 Запуск сортировки: Найдены минимальное и максимальное значения в массиве - 0 и 8 Высчитана delta по формуле (max-min)/N = 0.615385 0-й элемент распределен в 5 клетку Количество элементов во временном векторе векторов: 0 0 0 0 1 0 0 0 0 0 0 0 0 1-й элемент распределен в 9 клетку Количество элементов во временном векторе векторов: 0 0 0 0 1 0 0 0 1 0 0 0 0 2-й элемент распределен в 5 клетку Количество элементов во временном векторе векторов: 0 0 0 0 2 0 0 0 1 0 0 0 0 3-й элемент распределен в 12 клетку Количество элементов во временном векторе векторов: 0 0 0 0 2 0 0 0 1 0 0 1 0 0 4-й элемент распределен в 13 клетку Количество элементов во временном векторе векторов: 0 0 0 0 2 0 0 0 1 0 0 1 1 0 5-й элемент распределен в 7 клетку Количество элементов во временном векторе векторов: 0 0 0 0 2 0 1 0 1 0 0 1 1 0 6-й элемент распределен в 9 клетку

		Количество элементов во временном векторе векторов: 0 0 0 0 2 0 1 0 2 0 0 1 1 0 7-й элемент распределен в 9 клетку Количество элементов во временном векторе векторов: 0 0 0 0 2 0 1 0 3 0 0 1 1 0 8-й элемент распределен в 1 клетку Количество элементов во временном векторе векторов: 1 0 0 0 2 0 1 0 3 0 0 1 1 0 9-й элемент распределен в 5 клетку Количество элементов во временном векторе векторов: 1 0 0 0 3 0 1 0 3 0 0 1 1 0 10-й элемент распределен в 13 клетку Количество элементов во временном векторе векторов: 1 0 0 0 3 0 1 0 3 0 0 1 2 0 11-й элемент распределен в 4 клетку Количество элементов во временном векторе векторов: 1 0 0 1 3 0 1 0 3 0 0 1 2 0 12-й элемент распределен в 10 клетку Количество элементов во временном векторе векторов: 1 0 0 1 3 0 1 0 3 1 0 1 2 0 По итогу имеем следующий временный вектор: 1: 0 0: 0: 1: 2 3: 3 3 3 0: 1: 4 0:

		3: 5 5 5 1: 6 0: 1: 7 2: 8 8 0: Последовательно загоняем блоки, отсортировав их: 0 0 2 0 2 3 3 3 0 2 3 3 3 4 0 2 3 3 3 4 5 5 5 0 2 3 3 3 4 5 5 5 6 0 2 3 3 3 4 5 5 5 6 7 0 2 3 3 3 4 5 5 5 6 7 8 8 Отсортированный порядок: 0 2 3 3 3 4 5 5 5 6 7 8 8 Отсортированный библиотечным методом порядок: 0 2 3 3 3 4 5 5 5 6 7 8 8
2.	-1 1 0	Исходный порядок: -1 1 0 Запуск сортировки: Найдены минимальное и максимальное значения в массиве - -1 и 1 Высчитана delta по формуле (max-min)/N = 0.666667 0-й элемент распределен в 1 клетку Количество элементов во временном векторе векторов: 1 0 0 0 1-й элемент распределен в 4 клетку Количество элементов во временном векторе векторов: 1 0 0 1 2-й элемент распределен в 2 клетку Количество элементов во временном векторе векторов: 1 1 0 1

		По итогу имеем следующий временный вектор: 1: -1 1: 0 0: 1: 1 Последовательно загоняем блоки, отсортировав их: -1 -1 0 -1 0 1 Отсортированный порядок: -1 0 1 Отсортированный библиотечным методом порядок: -1 0 1	
3.		Исходный порядок: Нет необходимости запускать сортировку Отсортированный порядок: Отсортированный библиотечным методом порядок:	
4.	1	Исходный порядок: 1 Нет необходимости запускать сортировку Отсортированный порядок: 1 Отсортированный библиотечным методом порядок: 1	
5.	0.34534 0.213 0.3	Исходный порядок: 0.34534 0.213 0.3 Запуск сортировки: Найдены минимальное и максимальное значения в массиве - 0.213 и 0.34534 Высчитана delta по формуле (max-min)/N = 0.0441133 0-й элемент распределен в 4 клетку Количество элементов во временном векторе векторов: 0 0 0 1 1-й элемент распределен в 1 клетку	Изменены типы для входных значений в главной функции, функция solomonSort(vector<T>*) осталась неизменна

		Количество элементов во временном векторе векторов: 1 0 0 1 2-й элемент распределен в 2 клетку Количество элементов во временном векторе векторов: 1 1 0 1 По итогу имеем следующий временный вектор: 1: 0.213 1: 0.3 0: 1: 0.34534 Последовательно загоняем блоки, отсортировав их: 0.213 0.213 0.3 0.213 0.3 0.34534 Отсортированный порядок: 0.213 0.3 0.34534 Отсортированный библиотечным методом порядок: 0.213 0.3 0.34534	
6.	s	Wrong input!	В случае с целочисленными вводимыми данными добавлена проверка если std::atoi() некорректно выполняется