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. 9304 Ламбин А.В.

Преподаватель Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с методами сортировки, реализовать с помощью языка программирования C++ один из методов сортировки для любых типов данных.

Задание.

Вариант 22.

Реализовать алгоритм пасьянсной сортировки.

Выполнение работы.

При запуске программы из стандартного потока ввода считывается строка – набор целых чисел, разделённых пробелами. В случае, если строка или её начальная часть валидна, то в вектор *array* записываются считанные числа. В противном случае, выводится сообщение о некорректных входных данных, и программа завершается. В стандартный поток вывода записывается изначальный массив, затем он и его копия сортируются с помощью функций *patienceSort()* и *std::sort()*, соответственно, и также записываются в стандартный поток вывода.

Для удобства был перегружен оператор вывода вектора в поток. При попытке записи вектора в стандартный поток вывода или файл будут выводиться элементы вектора, разделённые запятой и заключённые в общие квадратные скобки.

В шаблонной функции *patienceSort()* объявляются вектор стеков *arrStack*, применяющийся в реализации алгоритма сортировки, его размер *size*, размер входного вектора элементов *count* и булевая переменная *isStart*, показывающая, был ли начат первый этап алгоритма.

Для начала необходимо перенести элементы из вектора в стеки по следующему принципу: начиная с первого стека, проходим по всем имеющимся, если в текущем стеке есть элемент, больший данного, то помещаем его туда, в противном случае – переходим к следующему; если элемент не был положен ни в один из стеков, создаём новый и помещаем его туда.

Вторым шагом необходимо верхние элементы всех стеков расположить отдельно в том же порядке, в котором находились их стеки. На каждом шаге алгоритма записываем первый имеющийся в новом стеке элемент в исходный вектор, и проверяем первые стеки включительно до того, элемент которого был записан в вектор. Если среди верхних элементов их есть наименьший, который будет так же меньше верхнего отдельного стека, то мы его помещаем туда. Затем алгоритм повторяется.

Разработанный программный код см. в приложении А.

Тестирование.

Запуск программы начинается с ввода команды “make”, что приведёт к компиляции и линковки программы, после чего будет создан исполняемый файл lab4. Запуск программы возможен двумя способами:

- командой “./lab4”, после запуска которой необходимо ввести выражение; результат будет напечатан на экране;
- “./lab4 «input»”, где input – входная строка в кавычках.

Тестирование программы производится с помощью скрипта script.py, написанного на языке программирования Python3. Запуск скрипта осуществляется командой “python3 script.py”.

Результаты тестирования см. в приложении Б.

Выводы.

Было проведено ознакомление с методами сортировки, был реализован метод пасьянсной сортировки с помощью языка программирования C++.

Была разработана программа, сортирующая массив элементов методом пасьянсной сортировки. Использование данного метода, на мой взгляд, оправдано ввиду быстроты работы (сложность алгоритма в лучшем случае $O(n)$, в худшем – $O(n \log n)$).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <string>
#include <vector>
#include <stack>
#include <algorithm>
#include <regex>

template<typename type>
std::ostream &operator << (std::ostream &out, const std::vector<type>
&arr) {
    if (arr.size() == 0) {
        out << "[]";
    } else {
        out << '[';
        for (int i = 0; i < arr.size() - 1; i++)
            out << arr[i] << ", ";
        out << arr[arr.size() - 1] << ']';
    }
    return out;
}

template<typename type>
void patienceSort (std::vector<type> &array) {
    std::vector<std::stack<type>> arrStack;
    int size = 0;
    int count = array.size();
    bool isStart = false;

    // first stage
    for (type elem : array) {
        if (!isStart) {
            std::stack<type> stack;
            arrStack.push_back(stack);
            arrStack[0].push(elem);
            size++;
            isStart = true;
            continue;
        }
        bool flag = false;
        for (int i = 0; i < size; i++) {
            if (arrStack[i].top() >= elem) {
                arrStack[i].push(elem);
                flag = true;
                break;
            }
        }
        if (!flag) {
            std::stack<type> stack;
            arrStack.push_back(stack);
            arrStack[size].push(elem);
            size++;
        }
    }
```

```
    array.clear();
    std::cout << '\t' << array << '\n';

    // second stage
    std::stack<type> queue;
    for (int i = size - 1; i >= 0; i--) {
        queue.push(arrStack[i].top());
        arrStack[i].pop();
    }
    int queueCount = size;
    for (int k = 0; k < count; k++) {
        array.push_back(queue.top());
        queue.pop();
        std::cout << '\t' << array << '\n';
        queueCount--;
        if (k == count - 1)
            break;
        type min;
        int numStack;
        if (queue.empty()) {
            for (int i = 0; i < size; i++) {
                if (!arrStack[i].empty()) {
                    min = arrStack[i].top();
                    numStack = i;
                    break;
                }
            }
        } else {
            min = queue.top();
            numStack = -1;
        }
        for (int i = 0; i < size - queueCount; i++) {
            if (!arrStack[i].empty() && arrStack[i].top() < min) {
                min = arrStack[i].top();
                numStack = i;
            }
        }
        if (numStack > -1) {
            queue.push(arrStack[numStack].top());
            queueCount++;
            arrStack[numStack].pop();
        }
    }
}

bool checkString (std::string &str) {
    std::regex target("( )+");
    str = std::regex_replace(str, target, " ");
    if (str[0] == ' ')
        str.erase(0, 1);
    if (str[str.size() - 1] == ' ')
        str.erase(str.size() - 1, 1);
    if (str.size() == 0)
        return false;
    return true;
}

int main (int argc, char **argv) {
```

```
    std::vector<int> array;
    std::string input;
    if (argc < 2)
        std::getline(std::cin, input);
    else
        input = argv[1];
    if (!checkString(input)) {
        std::cerr << "Error: wrong input string\n";
        return 0;
    }
    std::istringstream stream(input);
    int data;
    while (stream >> data)
        array.push_back(data);
    if (array.empty()) {
        std::cerr << "Error: wrong input string\n";
        return 0;
    }
    std::cout << "Input array: " << array << '\n';
    std::vector<int> copyArray = array;
    std::sort(copyArray.begin(), copyArray.end());
    patienceSort<int>(array);
    std::cout << "Array sorted with patience sort: " << array << '\n';
    std::cout << "Array sorted with std::sort: " << copyArray << '\n';
    return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 – Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	76 56 98 5 34 23 98 48 12 38 64 100 24 66	Input array: [76, 56, 98, 5, 34, 23, 98, 48, 12, 38, 64, 100, 24, 66] [] [5] [5, 12] [5, 12, 23] [5, 12, 23, 24] [5, 12, 23, 24, 34] [5, 12, 23, 24, 34, 38] [5, 12, 23, 24, 34, 38, 48] [5, 12, 23, 24, 34, 38, 48, 56] [5, 12, 23, 24, 34, 38, 48, 56, 64] [5, 12, 23, 24, 34, 38, 48, 56, 64, 66] [5, 12, 23, 24, 34, 38, 48, 56, 64, 66, 76] [5, 12, 23, 24, 34, 38, 48, 56, 64, 66, 76, 98] [5, 12, 23, 24, 34, 38, 48, 56, 64, 66, 76, 98, 98] [5, 12, 23, 24, 34, 38, 48, 56, 64, 66, 76, 98, 98, 100] Array sorted with patience sort: [5, 12, 23, 24, 34, 38, 48, 56, 64, 66, 76, 98, 98, 100] Array sorted with std::sort: [5, 12, 23, 24, 34, 38, 48, 56, 64, 66, 76, 98, 98, 100]	
2.	23 95 23 76 86 32 38 28 21 8 39	Input array: [23, 95, 23, 76, 86, 32, 38, 28, 21, 8, 39] [] [8] [8, 21] [8, 21, 23] [8, 21, 23, 23] [8, 21, 23, 23, 28] [8, 21, 23, 23, 28, 32]	

		[8, 21, 23, 23, 28, 32, 38] [8, 21, 23, 23, 28, 32, 38, 39] [8, 21, 23, 23, 28, 32, 38, 39, 76] [8, 21, 23, 23, 28, 32, 38, 39, 76, 86] [8, 21, 23, 23, 28, 32, 38, 39, 76, 86, 95] Array sorted with patience sort: [8, 21, 23, 23, 28, 32, 38, 39, 76, 86, 95] Array sorted with std::sort: [8, 21, 23, 23, 28, 32, 38, 39, 76, 86, 95]	
3.	9 -6 -9 0 7 1 -5	Input array: [9, -6, -9, 0, 7, 1, -5] [] [-9] [-9, -6] [-9, -6, -5] [-9, -6, -5, 0] [-9, -6, -5, 0, 1] [-9, -6, -5, 0, 1, 7] [-9, -6, -5, 0, 1, 7, 9] Array sorted with patience sort: [-9, -6, -5, 0, 1, 7, 9] Array sorted with std::sort: [-9, -6, -5, 0, 1, 7, 9]	
4.	-3520 3604 -3397 -6968 9304 1026 4715	Input array: [-3520, 3604, -3397, -6968, 9304, 1026, 4715] [] [-6968] [-6968, -3520] [-6968, -3520, -3397] [-6968, -3520, -3397, 1026] [-6968, -3520, -3397, 1026, 3604] [-6968, -3520, -3397, 1026, 3604, 4715] [-6968, -3520, -3397, 1026, 3604, 4715, 9304] Array sorted with patience sort: [-6968, -3520, -3397, 1026, 3604, 4715, 9304] Array sorted with std::sort: [-6968, -3520, -3397, 1026, 3604, 4715, 9304]	

5.	32 31 30 29 28 27 26	Input array: [32, 31, 30, 29, 28, 27, 26] [] [26] [26, 27] [26, 27, 28] [26, 27, 28, 29] [26, 27, 28, 29, 30] [26, 27, 28, 29, 30, 31] [26, 27, 28, 29, 30, 31, 32] Array sorted with patience sort: [26, 27, 28, 29, 30, 31, 32] Array sorted with std::sort: [26, 27, 28, 29, 30, 31, 32]	
6.	20 21 22 23 24	Input array: [20, 21, 22, 23, 24] [] [20] [20, 21] [20, 21, 22] [20, 21, 22, 23] [20, 21, 22, 23, 24] Array sorted with patience sort: [20, 21, 22, 23, 24] Array sorted with std::sort: [20, 21, 22, 23, 24]	
7.		Error: wrong input string	
8.	qwe	Error: wrong input string	
9.	d a c b	Error: wrong input string	