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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировки

Студент гр. 9304 _____ Афанасьев А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием бинарного дерева, реализовать его, решить поставленную задачу на его основе.

Постановка задачи.

Вариант *12.*

Реализовать итеративную быструю сортировку контейнеров. Быстрая сортировка –итеративная реализация.

Выполнение работы.

Программа на вход ожидает строку для сортировки. Вообще функция iterQSort() шаблонная и поддерживает не только строковый контейнер. Функция ничего не возвращает и принимает итератор на начало и итератор на элемент, следующий за концом контейнера. Алгоритм повторяет рекурсивный алгоритм быстрой сортировки, но работает без рекурсии, используя стек. Просто каждую итерацию на стек кладутся границы участка, который будет обрабатываться в будущем в цикле. Каждый шаг выбирается опорный элемент pivot (точнее соответствующий ему итератор), в данной реализации по умолчанию таким является последний элемент контейнера. Создается итератор iterToCompareWithPivot, который является копией первого элемента в контейнере. Он нам нужен для перемещения элементов относительно опорного. Берем из стека начало и конец и начинаем двигаться по участку контейнера в этом отрезке. Всякий раз, когда значение внутри итератора it (то, что перемещается по контейнеру в цикле) меньше значения в pivot, мы меняем значение it со значением iterToCompareWithPivot местами и передвигаем iterToCompareWithPivot вправо. Таким образом, элементы, что меньше опорного, окажутся слева от него, а те, что больше, - справа. Сам опорный пока находится все еще на последнем месте, после цикла мы поменяем его значение со значением iterToCompareWithPivot местами, так опорный окажется в центре.

Потом мы поделим отрезок на два и добавим границы полученных отрезков на стек.

Исходный код находится в приложении А.

Тестирование.

Программу можно собрать командой *make*, после этого создается исполняемый файл *lab4*. Его можно запустить, передав в него строку. Также можно запустить тестирующий скрипт *testScript.py,* конфигурационный файл которого лежит в папке с исполняемым файлом. В конфигурационном файле можно настроить многие параметры, включая количество тестов и директорию, в которой они находятся. В тестовом файле должна находиться только лишь одна строка – сам тест. Программа возвращает сообщение об синтаксической ошибке ввода,если такая была, либо ответ. Тестирующий скрипт выводит на экран поданную строку, результат работы программы, правильный ответ и *success* или *fail* в зависимости от совпадения того, что вернула программа, и правильного ответа. Пример его работы можно посмотреть на рисунке 1. А в таблице 1 можно посмотреть примеры строк-тестов.

```
strx@strxpc:~/gitreps/main/Programs/ETU/3SEM/AaDS/lb4$ python testScript.py
Make sure that this script is in the same directory as the program execute file.

test0:
Input: "12"
CorrectAnswer: 1 2
Answer: 1 2
Result: success

test1:
Input: "3251784"
CorrectAnswer: 1 2 3 4 5 7 8
Answer: 1 2 3 4 5 7 8
Result: success

Total: Successes: 2. Fails: 0
```

Рисунок 1 - Пример вызова скрипта

Таблица 1. Примеры входных и выходных данных

№	Входные данные	Выходные данные
1	1 2	1 2
2	3 2 5 1 7 8 4	1 2 3 4 5 7 8
3	2 5 5 4 1 2 3 1 2 5	1 1 2 2 2 3 4 5 5 5
4	4 6 5 6 8 7 8 9 0 9 0 8 9 7 5 4 3 6 7 8	0 0 3 4 4 5 5 6 6 6 7 7 7 8 8 8 8 9 9 9
5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
6	9 8 7 6 5 4 3 2 1	1 2 3 4 5 6 7 8 9
7	a b c d z x y	a b c d x y z

Выводы.

В ходе выполнения лабораторной работы был реализован итеративный алгоритм быстрой сортировки, который использует меньше памяти, чем рекурсивный аналог. В среднем алгоритм работает за $n * \log_2 n$, но в худшем случае за n^2.

ПРИЛОЖЕНИЕ А

main.cpp

```
#include "../libs/IterQSort.h"

#include <iostream>
#include <string>
#include <algorithm>

int main(int argc, char const *argv[])
{
    std::string str(argv[1]), strCopy = str;

    iterQSort<std::string::iterator>(std::begin(str),
std::end(str));

    std::sort(strCopy.begin(), strCopy.end());

    std::cout << "std::sort:\n";
    for (auto it = strCopy.begin(); it != strCopy.end(); +
+it)
        std::cout << *it << ' ';

    std::cout << '\n';
    return 0;
}
```

IterQSort.h

```
#ifndef __ITERQSORT__H__
#define __ITERQSORT__H__

#include <stack>
#include <iostream>

template <typename RandomIt>
void swapForSort(RandomIt left, RandomIt right)
{
    auto tmpSwap = *left;
    *left = *right;
    *right = tmpSwap;
}

template <typename RandomIt>
void iterQSort(RandomIt start, RandomIt end)
{
    if (start < end)
    {

        std::stack<RandomIt> stck;
        stck.push(end);
        stck.push(start);
```

```
        int counter = 0;

        std::cout << "Step " + std::to_string(counter) + ":\
n";
        for (auto it = start; it != end; ++it)
            std::cout << *it << ' ';
        std::cout << '\n';
        while (!stck.empty())
        {
            ++counter;
            RandomIt newStart = stck.top();
            stck.pop();

            RandomIt newEnd = stck.top();
            stck.pop();

            RandomIt iterToCompareWithPivot = newStart;
            RandomIt pivot = newEnd - 1;

            for (auto it = newStart; it != newEnd; ++it)
            {
                if (*it < *pivot)
                {
                    swapForSort<RandomIt>(it,
iterToCompareWithPivot);
                    ++iterToCompareWithPivot;
                }
            }
            swapForSort<RandomIt>(pivot,
iterToCompareWithPivot);
            std::cout << "Step " + std::to_string(counter) +
":\n";
            for (auto it = start; it != end; ++it)
                std::cout << *it << ' ';
            std::cout << '\n';

            if (iterToCompareWithPivot - 1 > newStart)
            {
                stck.push(iterToCompareWithPivot);
                stck.push(newStart);
            }
            if (iterToCompareWithPivot + 1 < newEnd)
            {
                stck.push(newEnd);
                stck.push(iterToCompareWithPivot + 1);
            }
        }
    }
}

#endif //!__ITERQSORT__H__
```

Makefile

```
compiler = g++
```

```
flags = -c -g -std=c++17 -Wall
appname = lab4
lib_dir = Sources/libs/
src_dir = Sources/srcs/

src_files := $(wildcard $(src_dir)*)
obj_files := $(addsuffix .o, $(basename $(notdir $
(src_files))))

define compile
    $(compiler) $(flags) $<
endef

programbuild: $(obj_files)
    $(compiler) $^ -o $(appname)

%.o: $(src_dir)/%.cpp $(lib_dir)/*.h
    $(call compile)

clean:
    rm -f *.o $(appname)
```