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 _____ Арутюнян В.В.

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 *15.*

Реализовать сортировку массивов слиянием – простое слияние, итеративная реализация.

Выполнение работы.

Программа на вход ожидает строку для анализа сразу после флага “*-s*”, иначе ожидается путь до файла со строкой для обработки. Возможно совместное использование:

./lab4 some/path/1 -s “some_string_1” -s “some_string_2” some/path/2

При таком вызове программа обработает строку из файла *some/path/1*, затем строку *some_string_1*, после строку *some_string_2*, далее строку из файла *some/path/2*.

Считанная строка разбивается на элементы, затем эти элементы добавляются в std::vector. Далее итераторы на начало вектора и на его конец передаются в шаблонную функцию MergeSort, где происходит сортировка элементов.

Краткий алгоритм:

Выделяется вспомогательный вектор additional.

Сначала все элементы вектора представляются, как n массивов (в каждом таком массиве находится лишь один элемент на данный момент), где n – длина вектора (Рисунок 1). Массивы только представляются, на самом деле всё действия будут происходить лишь внутри двух векторов: переданный нам и вспомогательный. Далее для большей понятности будет разбираться следующая строка:

«5 4 3 2 1 7»

Рисунок 1 – Представление вектора на первом шаге

Далее происходит слияние к двух упорядоченных массивов в один. Слияние происходит следующим методом. На каждом шаге берется меньший из двух первых элементов подмассивов и записывается в результирующий. Далее увеличиваются счетчик результирующего массива и подмассива, откуда был взят элемент. Если вдруг один из подмассивов закончился, то оставшиеся элементы второго подмассива просто копируются в результирующий (так как все элементы этого подмассива упорядочены). В роли результирующего выступает вспомогательный вектор additional.

Затем additional копируется в основной вектор. Далее все действия повторяются.

На рисунке 2 представлен результат после первого слияния.

Рисунок 2 – Представление вектора на втором шаге

На рисунке 3 представлен результат после второго слияния.

Рисунок 3 – Представление вектора на втором шаге

Далее происходит последнее слияние, на рисунке 4 представлен результат сортировки.

Рисунок 4 – Представление отсортированного вектора

Программа выводит изначальную строку, шаги слияния массивов, результат работы сортировки, результат работы std::sort для этой же строки.

Исходный код находится в приложении А.

Тестирование.

Программу можно собрать через *Makefile* командой *make*, после этого создается исполняемый файл *lab4*. Существует несколько вариантов провести тестирование:

1. Вызвать *lab4*, указав путь до файла с тестом, либо передать флаг "*-s*", а затем строку, которую требуется проанализировать, в кавычках;
2. Запустить python-скрипт – *testing.py*, в котором можно изменять параметры для тестирования, например, количество тестов, их расположение, расположение эталонных ответов, расположение ответов, полученных от программы.

Далее будет представлена таблица тестирования с несколькими тестами.

Таблица 1. Примеры входных и выходных данных

№	Входные данные	Выходные данные
1	- * + a b c - d + e * f g	* * + + - - a b c d e f g
2	gkjkzlk klzjdi jw88 8888 l 23hj j23 hsdjkfkajl kjh ajksdf alksdj iwjnvkj skjei	23hj 8888 ajksdf alksdj gkjkzlk hsdjkfkajl iwjnvkj j23 jw88 kjh klzjdi l skjei
3	2 2 - 0 + 1 2 3 - + -	+ + - - - 0 1 2 2 2 3
4	welcome hello bye thank you friend man women	bye friend hello man thank welcome women you
5	-123891273 1141 -124 128387 1 0 - 128371239 9123871283	-123891273 -124 -128371239 0 1 1141 128387 9123871283
6	g f d e c b a	a b c d e f g
7	12 912931 020 19292 39391293	020 12 19233 19292 39391293

	921939231 19233	912931 921939231
8	t 2 9 k * * *	* * * 2 9 k t
9	fa qwe lfa nsn alsdk kewn jas	alsdk fa jas kewn lfa nsn qwe
10	i3 2k la ka kans, jas .kasjd	.kasjd 2k i3 jas ka kans, la

Выводы.

В ходе выполнения лабораторной работы была реализована шаблонная функция MergeSort, выполняющая сортировку переданных ей данных. Данная сортировка является устойчивой – сохраняет порядок равных элементов. Однако она работает с одинаковой скоростью как на почти отсортированных векторах, так и векторах, из полностью хаотичных элементов. Также требуется дополнительная память равна размеру исходного вектора.

ПРИЛОЖЕНИЕ А

main.cpp

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

#include "../lib/merge_sort.h"

void FillVector(std::vector<std::string>& vec, std::stringstream&
sstr) {
  std::string str;
  while (sstr >> str) {
    vec.push_back(str);
  }
}

void print(std::vector<std::string> vec) {
  for (auto it : vec) {
    std::cout << it << ' ';
  }
  std::cout << '\n';
}

void Sort(std::vector<std::string>& vec) {
  std::cout << "In:\n    ";
  print(vec);
  MergeSort<std::vector<std::string>::iterator,
std::string>(vec.begin(),

vec.end());
  std::cout << "Out:\n    ";
  print(vec);
}

int main(int argc, char** argv) {
  if (argc == 1) {
    std::cout
        << "Too small arguments.\n"
        << "A expression is expected after \"-s\" flag, "
        << "otherwise a file path is expected.\n\n"
        << "example: ./lab4 -s \"2 3 0 18 3 299 11 94\"
Tests/test/test1.txt\n";
```

```
} else {
  bool is_string = false;

  for (int i = 1; i < argc; ++i) {
    std::string arg = argv[i];
    std::stringstream sstr;
    std::vector<std::string> vec;
    std::vector<std::string> vec_copy;
    if (is_string) {
      is_string = false;
      sstr << arg;
      FillVector(vec, sstr);
      vec_copy.resize(vec.size());
      std::copy(vec.begin(), vec.end(), vec_copy.begin());
      std::sort(vec_copy.begin(), vec_copy.end());
      Sort(vec);
      std::cout << "std::sort:\n    ";
      print(vec_copy);

    } else if (arg == "-s") {
      is_string = true;

    } else {
      std::ifstream file_in(arg);

      if (file_in.is_open()) {
        std::string str;
        std::getline(file_in, str);
        sstr << str;
        FillVector(vec, sstr);
        vec_copy.resize(vec.size());
        std::copy(vec.begin(), vec.end(), vec_copy.begin());
        std::sort(vec_copy.begin(), vec_copy.end());
        Sort(vec);
        std::cout << "std::sort:\n    ";
        print(vec_copy);

        file_in.close();

      } else {
        std::cout << "Couldn't open the file.\n";
      }  // else
    }    // else
  }      // for
}        // else
return 0;
```

```
}  // main
```

merge_sort.h

```
#ifndef MERGE_SORT_H_
#define MERGE_SORT_H_

#include <iostream>
#include <memory>
#include <vector>

template <typename RandomIt>
void PrintStep(RandomIt first, RandomIt last, size_t
step_quantity,
               size_t width) {
  std::cout << "Step №" << step_quantity << ":\n    (";
  for (size_t i = 0; first != last; ++first, ++i) {
    if (i && !(i % width)) {
      std::cout << ") ";
    }
    if (i && !(i % width)) {
      std::cout << '(';
    }
    std::cout << *first;
    if (((i + 1) % width) && !(first + 1 == last)) {
      std::cout << ' ';
    }
  }
  std::cout << ")\n";
}

template <typename RandomIt>
void Copy(RandomIt additional, RandomIt main, size_t len) {
  for (size_t i = 0; i < len; ++i, ++additional, ++main) {
    *main = *additional;
  }
}

template <typename RandomIt>
void Merge(RandomIt first, RandomIt second, RandomIt end, RandomIt
first_add) {
  RandomIt curr_first = first;
  RandomIt curr_second = second;
  RandomIt curr_add = first_add;
  for (size_t i = 0, len = end - first; i < len; ++i, ++curr_add)
{
```

```
    if (curr_first != second &&
        (curr_second == end || !(*curr_second < *curr_first))) {
      *curr_add = *curr_first;
      ++curr_first;
    } else {
      *curr_add = *curr_second;
      ++curr_second;
    }
  }
}

template <typename RandomIt, typename RandomType>
void MergeSort(RandomIt first, RandomIt last) {
  size_t len = last - first;
  if (last - first < 2) {
    return;
  }
  std::vector<RandomType> additional;
  additional.resize(len);

  size_t step_quantity = 1;
  for (size_t width = 1; width < len; width *= 2) {
    PrintStep<RandomIt>(first, last, step_quantity++, width);
    for (size_t i = 0; i < len; i += 2 * width) {
      Merge<RandomIt>(first + i, first + std::min(i + width, len),
                      first + std::min(i + 2 * width, len),
                      additional.begin() + i);
    }
    Copy<RandomIt>(additional.begin(), first, len);
  }
}

#endif  // MERGE_SORT_H_
```

Makefile

```
CXX = g++
TARGET = lab4
CXXFLAGS = -g -c -std=c++17
CXXOBJFLAGS = -g -std=c++17
LIBDIR = source/lib
SRCDIR = source/src
SRCS = $(wildcard $(SRCDIR)/*.cc)
OBJS = $(SRCS:.cpp=.o)

all: $(TARGET)
```

```
$(TARGET): $(OBJS)
     $(CXX) $(CXXOBJFLAGS) $(OBJS) -o $(TARGET)

%.o: $(SRCDIR)/%.cpp $(LIBDIR)/*.h
     $(CXX) $(CXXFLAGS) $<

clean:
     rm -rf $(SRCDIR)/*.o $(TARGET)
```