**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

Образовательная программа «Информационные системы и технологии»

Дисциплина «Теория и технология программирования»

Отчет к лабораторной работе № 7

«Методы поиска»

Выполнил:

студент группы 3530902/20001

Сафонов Андрей Игоревич

Принял:

Доцент к.т.н Хлопин. С. В.

Санкт-Петербург, 2023 год

**Задание лабораторной № 7:**

Составить программу для поиска в массиве данных последовательным и бинарным (по возможности) методами. В случае невозможности (из условий индивидуального задания) поиска бинарным методом, провести поиск бинарным методом одной буквы (цифры). Придумать тестовые примеры, для которых были бы эффективными каждый из методов.

Произвести поиск элемента (последовательности элементов) по нечетным строкам матрицы, заданной случайным образом. Вывести количество шагов для последовательного и бинарного поиска.

**Текст программы:**

Файл Laba\_7.cpp:

#include <iostream>

#include <vector>

#include "WriteFunction.h"

#include "ReadFunction.h"

#include "Search.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

Search<int> search;

search.Start();

}

Файл ReadFunction.h:

#pragma once

#include <string>

#include <vector>

#include <iostream>

std::string ReadLine() {

std::string s;

std::getline(std::cin, s);

return s;

}

std::vector<std::string> SplitIntoWords(const std::string& text) {

std::vector<std::string> words;

std::string word;

for (const char c : text) {

if (c == ' ') {

if (!word.empty()) {

words.push\_back(word);

word.clear();

}

}

else {

word += c;

}

}

if (!word.empty()) {

words.push\_back(word);

}

return words;

}

bool CheckIsNum(std::string str) {

if (str[0] == '+' || str[0] == '-')

str = str.substr(1, str.length());

bool have\_dots = false;

for (char ch : str) {

if (ch == '.' || ch == ',') {

have\_dots = true;

}

else if (have\_dots && ch != '0' || ch > '9' || ch < '0') {

return false;

}

}

return true;

}

int GetPositiveNum(std::string message, std::string error\_message) {

while (true) {

std::cout << message;

std::string str = ReadLine();

std::vector<std::string> words = SplitIntoWords(str);

if (words.size() == 1 && CheckIsNum(words[0]) && std::stoi(words[0]) != 0

&& std::stoi(words[0]) == std::abs(std::stoi(words[0])))

{

return std::stoi(str);

}

std::cout << error\_message << std::endl;

}

}

Файл Search.h:

#pragma once

#include <random>

#include <set>

#include <string>

#include <algorithm>

#include <map>

using namespace std;

struct Coord {

int column;

int row;

};

std::ostream& operator<< (std::ostream& out, const Coord& coord) {

out << "["s << coord.row << "]["s << coord.column << "]"s;

return out;

}

template<typename Integer>

class Search {

public:

Search() {

CreateMas();

}

void Start() {

GetQuery();

LineSearch();

BinarySearch();

}

private:

vector<vector<Integer>> origin\_;

Coord origin\_size\_;

set<Integer> split\_query\_;

void CreateMas() {

GetSize();

FillRandom();

cout << origin\_;

};

void GetSize() noexcept {

do {

origin\_size\_ = { GetPositiveNum("введите n:", "Некорректное значение n"), GetPositiveNum("введите m:", "Некорректное значение m") };

if (origin\_size\_.row \* origin\_size\_.column <= 100000000) {

break;

}

cout << "Слишком большой массив, максимальный размер массива 100000000 элементов"s << std::endl;

} while (true);

}

void FillRandom() {

srand(static\_cast<unsigned int> (time(NULL)));

for (size\_t i = 0; i < origin\_size\_.row; ++i) {

vector<Integer> buffer;

for (size\_t j = 0; j < origin\_size\_.column; ++j) {

Integer number = rand() % 200;

buffer.push\_back(number);

}

origin\_.push\_back(buffer);

}

}

void GetQuery() {

using namespace std::literals;

std::cout << "Введите последовательность элементов: "s;

std::string line = ReadLine();

for (const std::string& elem : SplitIntoWords(line)) {

if (!CheckIsNum(elem)) {

std::cout << elem << " - не число" << std::endl;

}

else if (split\_query\_.contains(stoi(elem))) {

std::cout << elem << " уже есть в запросе" << std::endl;

}

else {

split\_query\_.insert(stoi(elem));

}

}

std::cout << std::endl;

}

void Inicialize(std::map<Integer, std::vector<Coord>>& v) {

for (Integer elem : split\_query\_) {

v[elem];

}

}

void LineSearch() {

std::map<Integer, std::vector<Coord>> elem\_to\_coord;

Inicialize(elem\_to\_coord);

for (int row = 0; row < origin\_size\_.row; row += 2) {

for (int column = 0; column < origin\_size\_.column; column++) {

for (const Integer& query\_elem : split\_query\_) {

Integer elem = origin\_[row][column];

if (query\_elem == elem) {

elem\_to\_coord[query\_elem].push\_back({ column, row });

}

}

}

}

PrintLineSearchResult(elem\_to\_coord);

}

void PrintLineSearchResult(const std::map<Integer, std::vector<Coord>>& data) {

for (auto [elem, v\_coords] : data) {

std::cout << elem << ": "s;

if (!v\_coords.empty()) {

std::cout << v\_coords;

}

else {

std::cout << "No"s;

}

std::cout << std::endl;

}

std::cout << std::endl;

}

void BinarySearch() {

Binary\_Search binary\_search(origin\_, origin\_size\_);

binary\_search.Search(split\_query\_);

}

class Binary\_Search {

public:

friend class Search;

Binary\_Search(const std::vector<std::vector<Integer>>& elems, Coord size) {

ParseElemsForBinarySearch(elems, size);

std::sort(for\_search\_.begin(), for\_search\_.end());

}

void Search(std::set<Integer> split\_query) {

for (const Integer& elem : split\_query) {

SearchElem(elem);

}

PrintBinarySearchResult();

}

private:

struct Binary\_Info {

int amount = 0;

int first\_occurrence = 0;

int last\_occurrence = 0;

int begin\_index = 0;

int end\_index = 0;

};

std::vector<Integer> for\_search\_;

std::map<int, Binary\_Info > elem\_to\_step\_;

void ParseElemsForBinarySearch(std::vector<std::vector<Integer>> elems, Coord size) {

for (int row = 0; row < size.row; row += 2) {

for (int column = 0; column < size.column; column++) {

for\_search\_.push\_back(elems[row][column]);

}

}

}

struct Binary\_Border {

Binary\_Border(int size) {

high = size - 1;

middle = high / 2;

}

void Up() {

low = middle + 1;

middle = (low + high) / 2;

}

void Down() {

high = middle - 1;

middle = (low + high + 1) / 2;

}

bool Stop() {

return low <= high;

}

int low = 0;

int high = 0;

int middle = 0;

};

void SearchIdentical(Integer elem, int index, Binary\_Info& info) {

++info.amount;

for (int i = index - 1; i >= 0 && for\_search\_[i] == elem; i--) {

++info.amount;

}

for (int i = index + 1; i < for\_search\_.size() && for\_search\_[i] == elem; i++) {

++info.amount;

}

}

void SearchElem(Integer elem) {

Binary\_Info info;

Binary\_Border border(for\_search\_.size());

while (border.Stop()) {

++info.first\_occurrence;

if (for\_search\_[border.middle] == elem) {

SearchIdentical(elem, border.middle, info);

elem\_to\_step\_[elem] = info;

return;

}

else if (for\_search\_[border.middle] > elem) {

border.Down();

}

else {

border.Up();

}

}

elem\_to\_step\_[elem] = info;

}

void PrintBinarySearchResult() {

for (const auto& [elem, elem\_data] : elem\_to\_step\_) {

std::cout << elem << ": "s;

if (elem\_data.amount == 0) {

std::cout << "элемента нет в массиве, было выполнено " << elem\_data.first\_occurrence << " сравнений.";

}

else {

std::cout << "Первое вхождение было найдено за "s << elem\_data.first\_occurrence << " сравнений. Найдено "

<< elem\_data.amount << " элемент.";

}

std::cout << std::endl;

}

}

};

};

Файл WriteFucntion.h:

#pragma once

#include <iostream>

#include <vector>

#include <sstream>

#include <iomanip>

#include <stdio.h>

#include <unordered\_map>

#include <locale.h>

#include <algorithm>

#include <set>

#define MIN\_BORDER 2

#define MIN\_ELEM\_WIDTH 3

#define MAX\_SIZE\_OUTPUT\_MASSIV 20

class PrintContainer {

public:

PrintContainer() = default;

PrintContainer(size\_t n)

:elem\_width\_((MIN\_ELEM\_WIDTH > n ? MIN\_ELEM\_WIDTH : n) + MIN\_BORDER)

{

}

template<typename Elem>

std::ostream& Print2d(std::ostream& out, const std::vector<std::vector<Elem>>& container) {

using namespace std::literals;

for (auto line : container) {

Print(out, line);

out << std::endl;

}

return out;

}

template<typename Container>

std::ostream& Print(std::ostream& out, const Container& container) {

using namespace std::literals;

for (const auto& elem : container) {

PrintElem(out, elem);

}

return out;

}

template<typename Elem>

std::ostream& PrintElem(std::ostream& out, const Elem& elem) {

out << std::setw(elem\_width\_) << elem;

return out;

}

private:

int elem\_width\_ = MIN\_ELEM\_WIDTH + MIN\_BORDER;

};

template<typename Elem>

std::ostream& operator<< (std::ostream& out, const std::vector<std::vector<Elem>>& container) {

if (container.size() > MAX\_SIZE\_OUTPUT\_MASSIV || !container.empty() && container[0].size() > MAX\_SIZE\_OUTPUT\_MASSIV) {

using namespace std::literals;

out << " massiv..."s << std::endl;

return out;

}

long max\_elem = 0;

for (size\_t i = 0; i < container.size(); i++) {

long max\_in\_line = \*std::max\_element(container[i].begin(), container[i].end());

if (max\_in\_line > max\_elem) {

max\_elem = max\_in\_line;

}

}

PrintContainer print(std::to\_string(max\_elem).length());

print.Print2d(out, container);

return out;

}

template<typename Elem>

std::ostream& operator<< (std::ostream& out, const std::set<Elem>& container) {

for (auto elem : container) {

out << elem << " ";

}

out << std::endl;

return out;

}

template<typename Elem>

std::ostream& operator<< (std::ostream& out, const std::vector<Elem>& container) {

for (auto elem : container) {

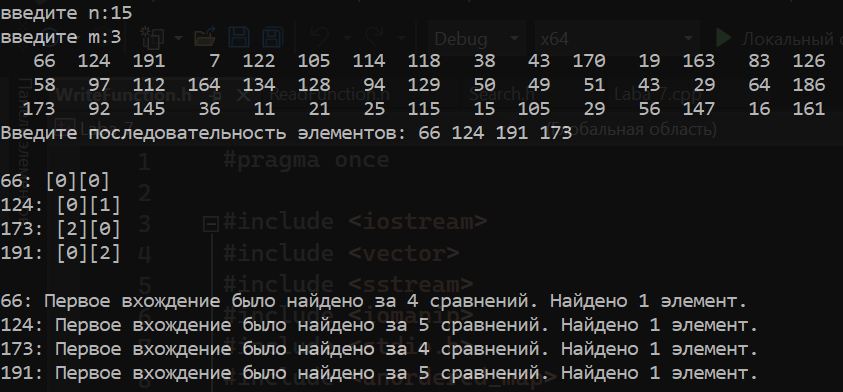
out << elem << " ";

}

return out;

}

**Пример работы:**



**Выводы:** в ходе лабораторной работы были изучены 2 метода алгоритма сортировки: последовательный и бинарный. Применив их на практике, были выявлены все тонкости работы между ними. При больших наборов данных и частых запросах лучше использовать бинарный алгоритм поиска, так как он работает за время log2(n), в то время как последовательный работает за время n.