|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №2.1  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9122-09.03.04прогин | | | |
|  |  |  | | А. В. Поляков | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2024 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

**Входные данные:**

Текстовый файл – InputFile.txt, каждая из строк которого содержит 3 поля в табличном виде (первое – Код направления, второе – ФИО, третье – название направления подготовки)(типы полей должны соответствовать типу данных в предметной области – все, что является числом, должно хранится в виде числа, а не строки, например код направления – это не строка, а структура из двух полей: char и int). Все данные во входном файле корректные (проверять на корректность не нужно).

Целое число n – количество строк входного файла для обработки (10 <= n <= 1000000).

Для каждого поля поиска:

Шаблон поиска;

Количество (сколько раз заданный шаблон должен встречаться в заданном поле поиска);

**Выходные данные:**

2 текстовых файла, строки которых содержат номер строки и данные входного файла, удовлетворяющие условиям поиска (DirectSearchOutput.txt – файл для алгоритма прямого поиска, WrightOutput.txt – файл для алгоритма поиска Райта), последняя строка – время, затраченное на поиск. Данные в выходном файле должны быть представлены в табличном виде.

**2 Описание алгоритмов поиска**

Райта:   
Функция поиска индекса первого вхождения сивола в массиве y с позиции fromIndex до позиции toIndex:

int findFirst(char[] y, int fromIndex, int toIndex, char symbol):

for (i = fromIndex .. toIndex)

if (y[i] == symbol)

return i

return -1  
  
Проверка, что все символы в y с позиции fromIndex и до toIndex и x с начала и до конца совпадают:

boolean restEquals(char[] y, int fromIndex, char[] x, int toIndex):

for (i = fromIndex .. toIndex)

if (y[i] != x[i - fromIndex])

return false

return true

Стадия препроцессинга (совпадает со стадией препроцессинга в алгоритме Бойера-Мура):

int[] preBmBc(char[] x, int m):

int[] result = int[ASIZE]

//Где ASIZE — размер алфавита

for (i = 0 .. ASIZE - 1)

result[i] = m;

for (i = 0 .. m - 2)

result[x[i]] = m - i - 1;

return result

Основная стадия алгоритма:

void RAITA(char[] x, int m, char[] y, int n):

int[] bmBc

char c, firstCh, middleCh, lastCh;

if (m == 0)

return

else if (m == 1)

//Проверка на случай поиска вхождения одного символа

int match = 0

while (match < n)

match = findFirst(y, match, n - 1, x[0])

if (match != -1)

print(match)

else

print("No matches")

return

//Вычисление массива плохих сиволов и объявление первого, последнего и среднего сиволов

bmBc = preBmBc (x, m)

firstCh = x[0];

middleCh = x[m/2];

lastCh = x[m - 1];

//Поиск

int j = 0

while (j <= n - m)

c = y[j + m - 1]

if (lastCh == c && middleCh == y[j + m / 2] && firstCh == y[j] && //Совпадение шаблона и окна из текста

restEquals(y, j + 1, x, j + m - 2))

print(j)

return

j += bmBc[c];

print("No matches")  
  
Прямой поиск:

Student[] foundStudents;

// Перебор всех студентов

for (i=0 … students.length())

// Поиск по ФИО

string studentFIO = student.fullName.FIO;

int fioOccurrencesFound = 0;

// Перебор символов ФИО студента

for (i = 0 .. studentFIO.size() - fio.size())

j = 0;

// Проверка на длину ФИО студента и искомого ФИО

if (studentFIO.size() < fio.size())

Выход

// Сравнение символов ФИО студента с искомым ФИО

while (j < fio.size() и studentFIO[i + j] == fio[j])

++j

// Если найдено полное совпадение

if (j == fio.size())

++fioOccurrencesFound

if (fioOccurrencesFound == fioOccurrences)

Выход

// Поиск по названию курса

string studentCourseName = student.name.name;

int courseNameOccurrencesFound = 0;

// Перебор символов названия курса

for (i = 0 .. studentCourseName.size() - courseName.size())

j = 0

// Проверка на длину названия курса и искомого названия курса

if (studentCourseName.size() < courseName.size())

Выход

// Сравнение символов названия курса студента с искомым названием курса

while (j < courseName.size() и studentCourseName[i + j] == courseName[j])

++j

// Если найдено полное совпадение

if (j == courseName.size()) {

++courseNameOccurrencesFound

if (courseNameOccurrencesFound == courseNameOccurrences)

Выход

// Если найдено нужное количество вхождений в ФИО и в названии курса, добавляем студента в список найденных

if (fioOccurrencesFound == fioOccurrences и courseNameOccurrencesFound == courseNameOccurrences)

foundStudents.push\_back(student)

return foundStudents

**3 Текст программы**

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include <ctime>

#include <vector>

#include <chrono>

using namespace std;

// Структура для хранения кода направления студента

struct Code {

    char letter;

    unsigned int numbers;

};

// Структура для хранения ФИО студента

struct FullName {

    string surname;

    string name;

    string patronymic;

    string FIO;

};

// Структура для хранения названия направления студента

struct CourseName {

    string name;

};

// Структура для хранения данных студента

struct Student {

    Code code;

    FullName fullName;

    CourseName name;

    int lineNumber;

};

// Функция для генерации случайного кода направления

Code generateCode() {

    Code code;

    code.letter = 'B';

    code.numbers = 1000 + rand() % 9000;

    return code;

}

// Перегрузка оператора вывода для Code

ostream& operator<<(ostream& os, const Code& code) {

    os << code.letter << code.numbers;

    return os;

}

// Перегрузка оператора вывода для CourseName

ostream& operator<<(ostream& os, const CourseName& name) {

    os << name.name;

    return os;

}

// Перегрузка оператора вывода для Student

ostream& operator<<(ostream& os, const Student& student) {

    os << student.code.letter << student.code.numbers << " " << student.fullName.FIO << " " << student.name.name;

    return os;

}

// Генерация рандомного имени студента

FullName generateFullName() {

    string names[] = {"Artem", "Ekaterina", "Maxim", "Alexander", "Dmitry", "Nikita", "Egor", "Mikhail", "Roman", "Vladimir", "Pavel", "Daniel", "Anton", "Alexey", "Sergei"};

    string surnames[] = {"Polyakov", "Krushinina", "Johnson", "Brown", "Taylor", "Wilson", "Harris", "Clark", "White", "Anderson", "Allen", "Davis", "Martin", "Hall", "Lee"};

    string patronymics[] = {"Victorovich", "Vladislavovna", "Ivanovich", "Petrovich", "Sidorovich", "Mikhailovich", "Fedorovich", "Egorovich", "Romanovich", "Vladimirovich", "Pavlovich", "Daniilovich", "Antonovich", "Alexandrovich", "Sergeevich"};

    FullName fullName;

    fullName.FIO = surnames[rand() % 15] + " " + names[rand() % 15] + " " + patronymics[rand() % 15];

    return fullName;

}

// Генерация рандомного названия курса

CourseName generateCourseName() {

    string names[] = {"progin", "economic", "menegment", "urfuck", "med", "arhitecture", "himbio", "fizvoz", "politech", "ped"};

    CourseName name;

    name.name = names[rand() % 10];

    return name;

}

// Генератор ключей для входного файла

void generateKyesFile() {

    ofstream inFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/Search/InputFile.txt");

    for (int i = 0; i < 1000000; ++i) {

        Student student;

        student.code = generateCode();

        student.fullName = generateFullName();

        student.name = generateCourseName();

        inFile << student.code << " " << student.fullName.FIO << " " << student.name << endl;

    }

    inFile.close();

    cout << "Ключи сгенерированы\n" << endl;

}

// Функция для чтения данных студентов из файла

vector<Student> readKeyFromFile() {

    ifstream inFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/Search/InputFile.txt");

    vector<Student> students;

    int lineNumber = 1;

    while (!inFile.eof()) {

        Student student;

        inFile >> student.code.letter >> student.code.numbers;

        inFile >> student.fullName.surname >> student.fullName.name >> student.fullName.patronymic;

        student.fullName.FIO = student.fullName.surname + " " + student.fullName.name + " " + student.fullName.patronymic;

        inFile >> student.name.name;

        if (!inFile.fail()) {

            student.lineNumber = lineNumber++;

            students.push\_back(student);

        }

    }

    inFile.close();

    return students;

}

void printStudentVector(vector<Student> students) {

    for (const auto& student : students) {

        cout << student.code << " " << student.fullName.FIO << " " << student.name.name << endl;

    }

}

vector<Student> directedSearch(const vector<Student>& students, const string& fio, int fioOccurrences, const string& courseName, int courseNameOccurrences) {

    vector<Student> foundStudents;

    for (const auto& student : students) {

        // Поиск по ФИО

        const string& studentFIO = student.fullName.FIO;

        int fioOccurrencesFound = 0;

        for (size\_t i = 0; i <= studentFIO.size() - fio.size(); ++i) {

            size\_t j = 0;

            if (studentFIO.size() < fio.size()) {

                break;

            }

            while (j < fio.size() && studentFIO[i + j] == fio[j]) {

                ++j;

            }

            if (j == fio.size()) {

                ++fioOccurrencesFound;

                if (fioOccurrencesFound == fioOccurrences) {

                    break;

                }

            }

        }

        // Поиск по названию курса

        const string& studentCourseName = student.name.name;

        int courseNameOccurrencesFound = 0;

        for (size\_t i = 0; i <= studentCourseName.size() - courseName.size(); ++i) {

            size\_t j = 0;

            if (studentCourseName.size() < courseName.size()) {

                break;;

            }

            while (j < courseName.size() && studentCourseName[i + j] == courseName[j]) {

                ++j;

            }

            if (j == courseName.size()) {

                ++courseNameOccurrencesFound;

                if (courseNameOccurrencesFound == courseNameOccurrences) {

                    break;

                }

            }

        }

        // Если найдено нужное количество вхождений в ФИО и в названии курса, добавляем студента в список найденных

        if (fioOccurrencesFound == fioOccurrences && courseNameOccurrencesFound == courseNameOccurrences) {

            foundStudents.push\_back(student);

        }

    }

    return foundStudents;

}

const int ASIZE = 256; // ASCII

// Функция для препроцессинга алгоритма Райта

vector<int> preWrightBc(const char\* x, int m) {

    vector<int> bmBc(256, m); // ASIZE = 256 для ASCII

    for (int i = 0; i < m - 1; ++i) {

        bmBc[static\_cast<int>(x[i])] = m - i - 1;

    }

    return bmBc;

}

int findFirst(const char\* y, int fromIndex, int toIndex, char symbol) {

    for (int i = fromIndex; i <= toIndex; ++i) {

        if (y[i] == symbol)

            return i;

    }

    return -1;

}

bool restEquals(const char\* y, int fromIndex, const char\* x, int toIndex) {

    for (int i = fromIndex, j = 0; i <= toIndex; ++i, ++j) {

        if (y[i] != x[j])

            return false;

    }

    return true;

}

// Функция для поиска с использованием алгоритма Райта

vector<Student> wrightSearch(const vector<Student>& students, const string& fio, int fioOccurrences, const string& courseName, int courseNameOccurrences) {

    vector<Student> foundStudents;

    // Препроцессинг для каждого паттерна

    vector<int> fioBc = preWrightBc(fio.c\_str(), fio.size());

    vector<int> courseNameBc = preWrightBc(courseName.c\_str(), courseName.size());

    // Применяем алгоритм Райта для каждого студента

    for (const auto& student : students) {

        // Поиск по ФИО

        int fioOccurrencesFound = 0;

        int j = 0;

        while (j <= student.fullName.FIO.size() - fio.size()) {

            // Проверяем, совпадает ли конечный символ строки с конечным символом подстроки и сами строки совпадают

            if (student.fullName.FIO[j + fio.size() - 1] == fio.back() && restEquals(student.fullName.FIO.c\_str() + j, 0, fio.c\_str(), fio.size() - 1)) {

                ++fioOccurrencesFound;

                if (fioOccurrencesFound == fioOccurrences) {

                    break;

                } else {

                    j += fio.size(); // Переход на следующую позицию после найденного вхождения

                }

            } else {

                j += fioBc[static\_cast<int>(student.fullName.FIO[j + fio.size() - 1])];

            }

        }

        // Поиск по названию курса

        int courseNameOccurrencesFound = 0;

        j = 0;

        if (student.name.name.size() < courseName.size()) {

            continue;

        }

        while (j <= student.name.name.size() - courseName.size()) {

            // Проверяем, совпадает ли конечный символ строки с конечным символом подстроки и сами строки совпадают

            if (student.name.name[j + courseName.size() - 1] == courseName.back() && restEquals(student.name.name.c\_str() + j, 0, courseName.c\_str(), courseName.size() - 1)) {

                ++courseNameOccurrencesFound;

                if (courseNameOccurrencesFound == courseNameOccurrences) {

                    break;

                } else {

                    j += courseName.size(); // Переход на следующую позицию после найденного вхождения

                }

            } else {

                j += courseNameBc[static\_cast<int>(student.name.name[j + courseName.size() - 1])];

            }

        }

        // Если найдено нужное количество вхождений в ФИО и в названии курса, добавляем студента в список найденных

        if (fioOccurrencesFound >= fioOccurrences && courseNameOccurrencesFound >= courseNameOccurrences) {

            foundStudents.push\_back(student);

        }

    }

    return foundStudents;

}

void printDirectedSearchVectorToFile(vector<Student> students, chrono::duration<double> directedSearchTime, ofstream& outFile) {

    for (int i = 0; i < size(students); ++i) {

        outFile << students[i] << " " << students[i].lineNumber << endl;

    }

    outFile << directedSearchTime.count() << " секунд" << endl;

    cout << "Найденные прямым поиском строки выведены в файл\n" << endl;

}

void printWrightSearchVectorToFile(vector<Student> students, chrono::duration<double> wrightSearchTime, ofstream& outFile) {

    for (int i = 0; i < size(students); ++i) {

        outFile << students[i] << " " << students[i].lineNumber << endl;

    }

    outFile << wrightSearchTime.count() << " секунд" << endl;

    cout << "Найденные поиском Райты строки выведены в файл\n" << endl;

}

int main() {

    srand(time(nullptr));

    //generateKyesFile();

    vector<Student> students = readKeyFromFile();

    string fio = "Artem";

    int fioOccurrences = 1;

    string courseName = "progin";

    int courseNameOccurrences = 1;

    // Прямой поиск

    ofstream directedFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/Search/DirectSearchOutput.txt");

    auto start = chrono::high\_resolution\_clock::now();

    vector<Student> forDirectedSearch = directedSearch(students, fio, fioOccurrences, courseName, courseNameOccurrences);

    auto end = chrono::high\_resolution\_clock::now();

    chrono::duration<double> directedSearchTime = end - start;

    printDirectedSearchVectorToFile(forDirectedSearch, directedSearchTime, directedFile);

    directedFile.close();

    // Райта серч

    ofstream wrightFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/Search/WrightOutput.txt");

    start = chrono::high\_resolution\_clock::now();

    vector<Student> forWrightSearch = wrightSearch(students, fio, fioOccurrences, courseName, courseNameOccurrences);

    end = chrono::high\_resolution\_clock::now();

    chrono::duration<double> wrightSearchTime = end - start;

    printWrightSearchVectorToFile(forWrightSearch, wrightSearchTime, wrightFile);

    wrightFile.close();

    return 0;

}

**4 Тесты**

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Прямой поиск** | **Алгоритм Райта** |
| Поиск ключа в наборе из 1000000 ключей | Вход: InputFile.txt, “Artem”, “progin”, n = 1000;  Выход: DirectSearchOutput.txt  Время: 0.0010679 секунд | Вход: InputFile.txt, “Artem”, “progin”, n = 1000;  Выход: WrightOutput.txt  Время: 0.0004463 секунд |
| Поиск ключа в наборе из 1000000 ключей | Вход: InputFile.txt, “Artem”, “progin”, n = 10000;  Выход: DirectSearchOutput.txt  Время: 0.0106129 секунд | Вход: InputFile.txt, “Artem”, “progin”, n = 1000;  Выход: WrightOutput.txt  Время: 0.0033201 секунд |
| Поиск ключа в наборе из 1000000 ключей | Вход: InputFile.txt, “Artem”, “progin”, n = 100000;  Выход: DirectSearchOutput.txt  Время: 0.0978301 секунд | Вход: InputFile.txt, “Artem”, “progin”, n = 1000;  Выход: WrightOutput.txt  Время: 0.0380776 секунд |

**5 Вывод**

Исходя из выше представленных тестов, можно сказать, что алгоритм прямого поиска работает существенно медленнее алгоритма Райта. Время его выполнения увеличивается почти линейно с увеличением числа ключей. Этот алгоритм показывает низкую производительность, однако не затрачивает ресурсы на предварительную подготовку к самому процессу поиска.

Что касается алгоритма Райта – он быстрее, время его работы также увеличивается с ростом числа ключей, но менее значительно по сравнению с прямым поиском. Несмотря на большую скорость работы на больших наборах, на маленьких наборах может быть проблема с дополнительной затратой времени на стадию препроцессинга.