НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



**ЗВІТ**

**Про виконання лабораторної роботи № 10**

**з дисципліни «Алгоритми і структури даних»**

**Лектор:**

Доцент кафедри ПЗ

Коротєєва Т.О.

**Виконав:**

студ. групи ПЗ-23

Михалевич П.-І.В.

**Прийняла:**

Доцент кафедри ПЗ

Мельник Н.Б.

«\_\_\_» \_\_\_\_\_\_ 2021 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2021

**Тема роботи**: Алгоритм прямого пошуку в рядку.

**Мета роботи**: Навчитися застосовувати алгоритм прямого пошуку при пошуку cтрічки в тексті.

**Теоретичні відомості**

Часто доводиться стикатися зі пошуком рядка в тексті. Нехай є деякий текст Т і слово (або стрічка) W. Необхідно знайти всі входження цього слова в зазначеному тексті. Ця дія типово для будь-яких систем обробки текстів. (Елементи масивів Т і W - символи деякого кінцевого алфавіту - наприклад, {0, 1}, або {a, ..., z}, або {а, ..., я}.)

Найбільш типовим додатком такого завдання є документальний пошук: заданий фонд документів, що складаються з послідовності бібліографічних посилань, кожна посилання супроводжується «дескриптором», що вказує тему відповідного посилання. Треба знайти деякі ключові слова, що зустрічаються серед дескрипторів. Міг би мати місце, наприклад, запит «Програмування» і «C#». Такий запит можна трактувати наступним чином: чи існують статті, які мають дескрипторами «Програмування» і «C#».

Одним із таких алгоритмів є алгоритм прямого пошуку. Кількість порівнянь дорівнює N \* M. Цей алгоритм працює достатньо ефективно, якщо припустити, що незбіг пари символів відбувається достатньо часто. Можна припустити, що для текстів, складених із 128 символів, незбіг виявлятиметься після однієї або двох перевірок. Проте, у гіршому випадку, продуктивність може виявлятися набагато нижчою.

**Постановка завдання**

10. Задано два тексти. В першому тексті знайти слово яке складається з найменшої кількості голосних літер замінити його на слово записане у зворотньому порядку і знайти його входження в другий текст відповідним алгоритмом пошуку.

**Покроковий опис алгоритму лінійного пошуку:**

R1. Встановити *і=1.. n-m, j=1..m*.

R2. Якщо  *S[i] = P[j]* , то зафіксувати перше співпадіння *k=i*, та перевірити співпадіння всього масиву *P* у масиві *S.*При першому неспівпадінні відмінити значення *k* та продовжити пошук.

R3. Кінець.

**Код програми**

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

connect(ui->pushButton\_clear,SIGNAL(clicked()),this,SLOT(MySlot()));

connect(ui->pushButton\_start,SIGNAL(clicked()),this,SLOT(MySlot()));

ui->textEdit->setText("These modules was hard");

ui->textEdit\_text\_2->setText("I saw him earlier");

}

MainWindow::~***MainWindow***()

{

delete ui;

}

// Removes unnecessary spaces.

std::string **RemoveSpaces**(std::string myString){

for (size\_t i = 1; i <= myString.size(); ++i) {

// if(i-1 == 0 && myString[i - 1] == ' '){

// myString.erase(myString.begin() + (i - 1));

// myString.shrink\_to\_fit();

// i--;

// continue;

// }

if (myString[i - 1] == ' ' && (myString[i] == ' ' || myString[i] == '\0')) {

myString.erase(myString.begin() + (i - 1));

myString.shrink\_to\_fit();

i--;

}

}

return myString;

}

void MainWindow::**MySlot**()

{

QPushButton\* btn = (QPushButton \*) sender();

if(btn->text() == "Start"){

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

if(ui->textEdit->toPlainText() == "" || ui->textEdit\_text\_2->toPlainText() == ""){

QMessageBox::information(nullptr,"No input data","Enter text 1 and text 2!");

return;

}

QString line = ui->textEdit->toPlainText();

QString lowerline = line.toLower();

QStringList list = QString::fromStdString(RemoveSpaces(lowerline.toStdString())).split(' ');

int minIdx = 0;

int minCount = 100;

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

int vovelCount = 0;

for (int j = 0;j < list[i].count(); j++) {

if(list[i][j] == 'a' || list[i][j] == 'e' || list[i][j] == 'i' || list[i][j] == 'u' || list[i][j] == 'o'){

vovelCount++;

}

}

if(vovelCount < minCount){

minCount = vovelCount;

minIdx = i;

}

}

ui->lineEdit\_word->setText(list[minIdx]);

ui->lineEdit\_word\_idx->setText(QString::number(minIdx+1));

QString reverseWord;

std::string str;

std::string minstr = list[minIdx].toStdString();

str.resize(minstr.size());

int idx = 0;

for(int j = minstr.size()-1;j >= 0; j--){

str[idx++] = minstr[j];

}

reverseWord = QString::fromStdString(str);

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

auto elapsed = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin);

ui->lineEdit\_task\_time->setText(QString::number(elapsed.count()));

QString reverseWordSpaces = " " + reverseWord + " ";

std::string strreverseWordSpaces = reverseWordSpaces.toStdString();

QString line\_2 = " " + ui->textEdit\_text\_2->toPlainText() + " ";

QString lowerline\_2 = line\_2.toLower();

std::string strlowerline\_2 = RemoveSpaces(lowerline\_2.toStdString());

int count = 0;

int Idx = 0;

begin = std::chrono::high\_resolution\_clock::now();

int i = 0;

for (int j = 0; j < strreverseWordSpaces.size() && i < strlowerline\_2.size() ; j++) {

if(strlowerline\_2[i] == ' '){

Idx++;

}

if(strreverseWordSpaces[j] == strlowerline\_2[i]){

count++;

}else{

j = -1;

count = 0;

}

if(count == strreverseWordSpaces.size()){

break;

}

i++;

}

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

elapsed = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin);

ui->lineEdit\_find\_time->setText(QString::number(elapsed.count()));

ui->lineEdit\_rword->setText(reverseWord);

ui->lineEdit\_rword\_idx->setText(QString::number(Idx - 1));

}else if(btn->text() == "Clear"){

ui->textEdit->clear();

ui->textEdit\_text\_2->clear();

ui->lineEdit\_rword->clear();

ui->lineEdit\_rword\_idx->clear();

ui->lineEdit\_word->clear();

ui->lineEdit\_word\_idx->clear();

ui->lineEdit\_find\_time->clear();

ui->lineEdit\_task\_time->clear();

}

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

void **MySlot**();

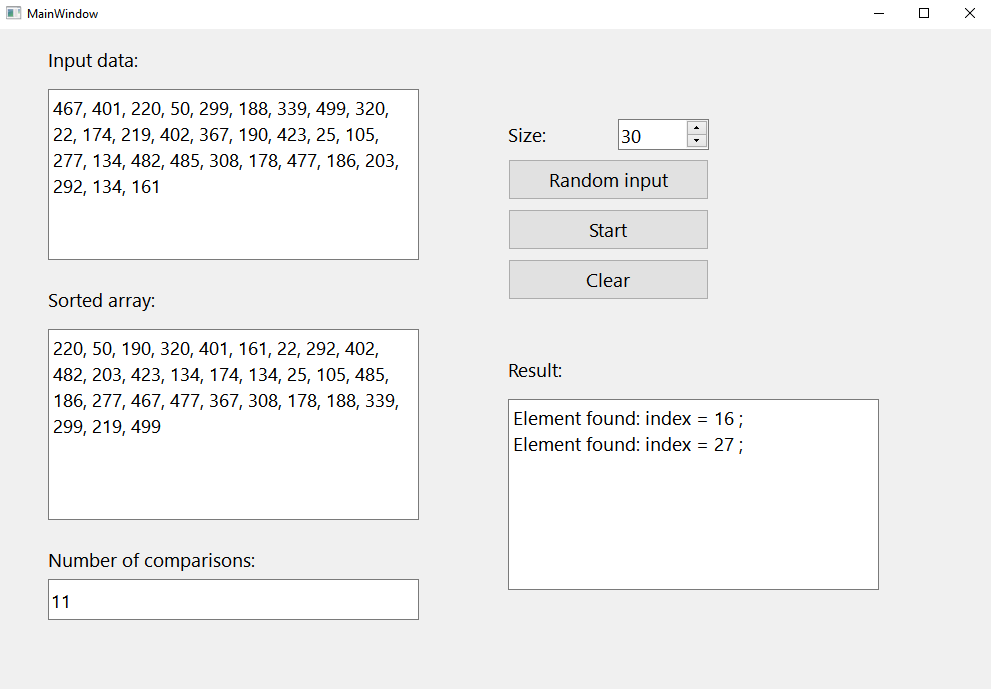
private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**Результат програми**

****

### Висновок

На цій лабораторній роботі я ознайомився з алгоритмом прямого пошуку. Складність цього алгоритму в якому N – розмір текста,а M- розмір стрічки яку шукаємо, O (N \* M). Недоліками алгоритму являєтся висока складність - O (N \* M), в гіршому випадку - Θ ((N-M + 1) \* M), та після розбіжності перегляд завжди починається з першого символу зразка і тому може включати символи, які раніше вже були видимими.