**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Інститут **ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №11

На тему: «Алгоритм Кнута-Прата-Морріса»

З дисципліни: «Алгоритми та структури даних»

**Лектор** : доцент каф.ПЗ

Коротєєва Т.О.

**Виконала:** ст.гр.ПЗ-23

Кохман О.В.

**Прийняв:** асистент каф.ПЗ

Франко А.В.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022 р.

\_ \_ \_ \_ \_ \_ \_ \_ .

Львів – 2022

**Тема:** алгоритм Кнута-Прата-Морріса.

**Мета:** навчитися використовувати алгоритм Кнута-Прата-Морріса.

**Теоретичні відомості**

**Алгоритм Кнута-Прата-Морріса** — один із алгоритмів пошуку рядка, що шукає входження слова W у рядку S, використовуючи просте спостереження, що коли відбувається невідповідність, то слово містить у собі достатньо інформації для того, щоб визначити, де наступне входження може початися, таким чином пропускаючи кількаразову перевірку попередньо порівняних символів.

Алгоритм, що винайшли Дональд Кнут та Вон Пратт, а також незалежно від них Джеймс Морріс, опубліковано у спільній статті у 1977 році.

Часова асимптотична складність алгоритму становить O(N+M), де N — довжина слова W, M — довжина рядка S.

Алгоритм повинен знайти початковий індекс m рядка W[] в рядку S[].

Найпростіший алгоритм пробігає по всьому рядку S[m], де m — індекс. Якщо індекс m досягне кінця рядка, то W[] не знайдено і алгоритм поверне результат «fail». На кожній позиції перевіряється рівність елемента на позиції m з S[] й елемента на першій позиції з W[], тобто S[m] =? W[0]. Якщо вони рівні, то алгоритм перевіряє наступні відповідні елементи в рядках за індексом i. Алгоритм перевіряє всі вирази S[m+i] =? W[i]. Якщо всі елементи з W знайдені, то алгоритм поверне позицію m.

Зазвичай, пробна перевірка швидко відкидає можливість збігу. Якщо рядки складаються з рівномірно розподілених елементів, то шанс, що перші елементи дорівнюють один одному, буде 1 до 26. Отже, в більшості випадків пробна перевірка відкидатиме початкові елементи. Шанс, що перші два елементи будуть рівними, дорівнює 1 до 262 (тобто, 1 до 676). Тобто, якщо елементи рівномірно розподілені, очікувана складність пошуку в рядку S[] довжини *k* буде порядку *k* порівнянь або *O*(*k*). Якщо S[] має 1.000.000.000 елементів і W[] має 1000 елементів, то пошук рядка займе приблизно 1.000.000.000 порівнянь.

Проте очікувана продуктивність не гарантована. Якщо рядки не випадкові, то на кожному кроці m може знадобитися багато порівнянь. У найгіршому випадку два рядки збігаються майже за всіма літерами. Якщо рядок S[] має 1.000.000.000 елементів, що рівні *A* і рядок W[] складається з 999 елементів *A* і останній елемент *B*. Тоді найпростіший алгоритм на кожному кроці виконуватиме 1000 перевірок, а всіх перевірок буде 1 трильйон. Отже, якщо довжина W[] — *n*, то в найгіршому випадку складність становитиме *O*(*k*⋅*n*).

Алгоритм КМП має кращий показник швидкодії в найгіршому випадку. КМП витрачає небагато часу (за порядком розміру W[], *O*(*n*)) на попереднє обчислення таблиці, і потім використовує таблицю для швидкого пошуку рядка за час *O*(*k*).

З іншого боку, на відміну від попередньо розглянутого простого алгоритму, алгоритм КМП використовує відомості про попередні порівняння. У прикладі, що наведений вище, коли KMП зустрічає незбіг на 1000-ному елементі (i = 999), тобто S[m+999] ≠ W[999], КМП знатиме, що 999 позицій вже перевірено. КМП містить ці знання у попередньо обчисленій таблиці і додаткових змінних. Коли KMП знаходить незбіг, з таблиці визначається, наскільки збільшиться змінна m.

Послідовний опис алгоритму Кнута-Прата-Морріса

**Алгоритм КМП**

КМП 1. Встановити *і=0*.

КМП 2. *j=0, d=1*.

КМП 3. Поки *j<m, i<n*

 Перевірка: якщо *S[i]=P[j]*, то *d++, i++.j++* поки *d != m*.

КМП 4. Інакше встановити зсув взірця на *d-D[d]* позицій по тексту . Перейти на крок КМП 2.

КМП 5. Кінець.

**Індивідуальне завдання**

9. Задано два тексти. В першому тексті знайти найдовше слово і знайти його входження в другий текст відповідним алгоритмом пошуку.

**Код програми**

**Назва файлу: MyForm.cpp**

#include "MyForm.h"

using namespace Main;

int main() {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MyForm());

return 0;

}

**Назва файлу: MyForm.h**

#pragma once

#include <string>

#include "standardString.h"

using namespace std;

namespace Main {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::RichTextBox^ richTextBox1;

protected:

private: System::Windows::Forms::RichTextBox^ richTextBox2;

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::RichTextBox^ richTextBox3;

private: System::Windows::Forms::Button^ button2;

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->richTextBox1 = (gcnew System::Windows::Forms::RichTextBox());

this->richTextBox2 = (gcnew System::Windows::Forms::RichTextBox());

this->button1 = (gcnew System::Windows::Forms::Button());

this->richTextBox3 = (gcnew System::Windows::Forms::RichTextBox());

this->button2 = (gcnew System::Windows::Forms::Button());

this->SuspendLayout();

//

// richTextBox1

//

this->richTextBox1->Location = System::Drawing::Point(46, 73);

this->richTextBox1->Name = L"richTextBox1";

this->richTextBox1->Size = System::Drawing::Size(299, 64);

this->richTextBox1->TabIndex = 0;

this->richTextBox1->Text = L"";

//

// richTextBox2

//

this->richTextBox2->Location = System::Drawing::Point(394, 73);

this->richTextBox2->Name = L"richTextBox2";

this->richTextBox2->Size = System::Drawing::Size(299, 64);

this->richTextBox2->TabIndex = 1;

this->richTextBox2->Text = L"";

//

// button1

//

this->button1->Location = System::Drawing::Point(307, 30);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(113, 37);

this->button1->TabIndex = 2;

this->button1->Text = L"generate";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// richTextBox3

//

this->richTextBox3->Location = System::Drawing::Point(160, 209);

this->richTextBox3->Name = L"richTextBox3";

this->richTextBox3->Size = System::Drawing::Size(394, 195);

this->richTextBox3->TabIndex = 3;

this->richTextBox3->Text = L"";

//

// button2

//

this->button2->Location = System::Drawing::Point(307, 159);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(113, 33);

this->button2->TabIndex = 4;

this->button2->Text = L"find";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2\_Click);

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(727, 495);

this->Controls->Add(this->button2);

this->Controls->Add(this->richTextBox3);

this->Controls->Add(this->button1);

this->Controls->Add(this->richTextBox2);

this->Controls->Add(this->richTextBox1);

this->Name = L"MyForm";

this->Text = L"MyForm";

this->ResumeLayout(false);

}

#pragma endregion

String^ text1 = "And involving the antiemotion in these anticipations could have been a smart idea";

String^ text2 = "And involving the antiemotion anticipation Panticipations in these could have been a smart idea";

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

richTextBox1->Text = text1;

richTextBox2->Text = text2;

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

string one = toStandardString(text1);

string two = toStandardString(text2);

string pattern = search(one);

richTextBox3->Text += "Word to search - " + (gcnew String(pattern.c\_str())) + "\n";

int result = KMP(two, pattern, richTextBox3);

if (result < 0) {

richTextBox3->Text += "There is no instance of such string";

}

else {

richTextBox3->Text += "Index: " + result;

}

}

private: string search(string text) {

int size = text.length();

char\* char\_array = new char[size + 1];

char\* char\_array2 = new char[size + 1];

strcpy(char\_array, text.c\_str());

strcpy(char\_array2, text.c\_str());

char\* token;

int arraySize = 0;

token = strtok(char\_array, " ");

while (token != NULL) {

token = strtok(NULL, " ");

arraySize++;

}

string\* array = new string[arraySize];

char\* token2;

token2 = strtok(char\_array2, " ");

int i = 0;

while (token2 != NULL) {

array[i] = token2;

token2 = strtok(NULL, " ");

i++;

}

int max = array[0].length();

int index = 0;

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

if (max < array[i].length()) {

max = array[i].length();

index = i;

}

}

return array[index];

}

private: int KMP(string text, string pattern, System::Windows::Forms::RichTextBox^ richTextBox) {

int textLength = text.length();

int patternLength = pattern.length();

if (patternLength == 0) {

return -1;

}

if (textLength < patternLength) {

return -1;

}

int\* next = new int[patternLength + 1];

for (int i = 0; i < patternLength + 1; i++) {

next[i] = 0;

}

for (int i = 1; i < patternLength; i++) {

int j = next[i + 1];

while (j > 0 && pattern[j] != pattern[i]) {

j = next[j];

}

if (j > 0 || pattern[j] == pattern[i]) {

next[i + 1] = j + 1;

}

}

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

if (text[i] == pattern[j]) {

richTextBox->Text += (gcnew String(string(1,(text[i])).c\_str())) + " == " + (gcnew String(string(1,(pattern[j])).c\_str())) + "\ti = " + i + "\tj = "+ j + "\n";

if (++j == patternLength) {

return (i - j + 1);

}

}

else if (j > 0) {

richTextBox->Text += (gcnew String(string(1, (text[i])).c\_str())) + " != " + (gcnew String(string(1, (pattern[j])).c\_str())) + "\n";

j = next[j];

i--;

richTextBox->Text += "i = " + i + " j = " + j + "\n";

}

}

}

};

}

**Назва файлу: standardString.h**

#include <string>

static std::string toStandardString(System::String^ string) {

using System::Runtime::InteropServices::Marshal;

System::IntPtr pointer = Marshal::StringToHGlobalAnsi(string);

char\* charPointer = reinterpret\_cast<char\*>(pointer.ToPointer());

std::string returnString(charPointer, string->Length);

Marshal::FreeHGlobal(pointer);

return returnString;

}

**Протокол роботи**

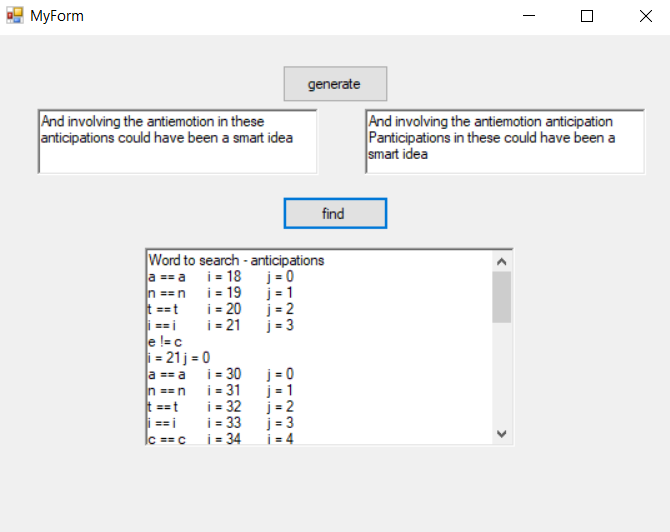
****

Рис. 1 Результат роботи програми.

**Висновок**

На цій лабораторній роботі я дізналась про алгоритм Кнута-Прата-Морріса та реалізувала пошук слова в тексті за допомогою цього алгоритму та продемонструвала результат роботи програми на формі у Visual Studio 2022. Також дізналась складність цього алгоритму, що дорівнює O(n + m), де n – довжина тексту, а m – довжина паттерну.