**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №14 на тему

«Методы поиска подстроки в строке»

Вариант №11

Выполнил студент группы РИС-20-1б

Шумилов Лев Сергеевич

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с особыми методами поиска подстроки в строке:

1. Метод Бойера-Мура;
2. Метод Кнута-Морриса-Пратта;

**Постановка задачи**

Задача – реализовать на языке C++ программы реализующие данные методы поиска подстроки в строке. Вывести на экран все вхождения, если количество совпадающих ключей больше одного.

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code
3. Система контроля версий Git

Для проведения операций ввода-вывода, необходимо подключить заголовочный файл iostream через директиву #include. Для работы со строками подключается заголовочный файл string.

Для удобства, сообщения о просьбе ввода данных с клавиатуры были выведены в отдельные кодовые слова с помощью директивы #define. Для удобного использования операторов ввода-вывода (чтобы каждый раз не прописывать std::) используется директива using.

#include <iostream>

#include <string>

#define INPUT\_MESSAGE "Enter the string, which will be used for searching:"

#define NEW\_INPUT\_MESSAGE "Enter the substring, which is necessary to find:"

using namespace std;

Для удобного чтения строки из консоли, реализована функция ReadLine, которая возвращает тип string и позволяет вводить пробелы.

string ReadLine()

{

string line;

getline(cin, line);

return line;

}

Основная функция main будет выполнять следующие действия:

* Выводить сообщения о вводе данных (ввод строки и подстроки);
* Принимать на ввод с клавиатуры строку и подстроку;
* Вызывать функцию поиска подстроки в строке;

int main()

{

cout << INPUT\_MESSAGE << endl;

string mainString = ReadLine();

cout << NEW\_INPUT\_MESSAGE << endl;

string substring = ReadLine();

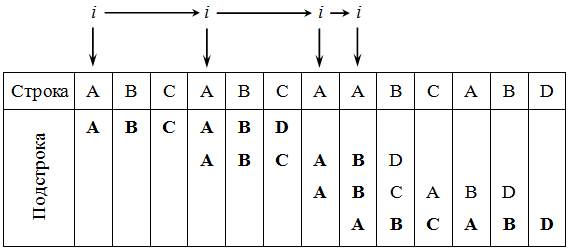
<функция поиска подстроки в строке>(mainString, substring);

return 0;

}

**Алгоритм Кнута-Морриса-Пратта:**

Это эффективный алгоритм поиска подстроки в строке, который с помощью таблицы π быстро находит подстроку. Данный алгоритм является самым быстрым, так как время выполнения линейно зависит от объёма входных данных.



Функция объявляется следующим образом: в качестве аргументов принимается строка, в которой происходит поиск и подстрока, которую необходимо найти в данной строке, а возвращается индекс первого вхождения подстроки в строке (если подстрока не найдена, то возвращается -1).

int KnuthMorrisPrattSearch (string baseString, string substring)

Для удобства работы объявляются следующие переменные:

* Длина строки;
* Длина подстроки;
* Индекс первого вхождения;
* Порядковый номер текущего вхождения (для вывода на консоль);
* Флаг, показывающий был ли найден элемент;
* Динамический массив π;

int

baseStrLength = baseString.size(),

subStrLength = substring.size(),

firstEntryIndex = -1,

entryOrderNumber = 0;

bool isElementFound = false;

int\* pi = new int[subStrLength];

Инициализация массива π:

В первую очередь, необходимо заполнить массив сдвигов образа – массив π. Для этого была реализована отдельная функция, которая вызывает в цикле задаёт значение каждого элемента массива при помощи функции, которая находит максимальную длину равных суффиксов и префиксов соответствующих частей строки.

int\* OffsetPi(string substring)

{

int subStrLength = substring.size(),

\*pi = new int[subStrLength];

string workPart = "";

for (int i = 0; i < subStrLength; i++)

{

workPart += substring[i];

pi[i] = MaxEqualityLength(workPart);

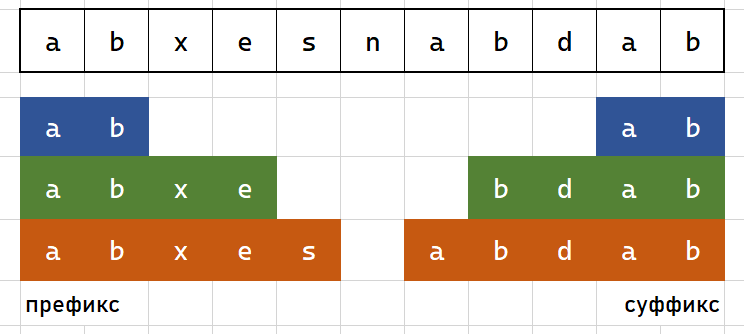
}

return pi;

}

Префикс – это часть образа, находящаяся в левой части строки (с самого первого и по n-й элемент, не включая последний).

Суффикс – это часть образа, который находится в правой части строки (с самого последнего и по n-й элемент, не включай первый).



Функция, которая возвращает значение самых длинных равных суффиксов работает по следующему принципу:

1. Через цикл for накапливаются суффикс и префикс;
2. Суффикс и префикс сравниваются;
3. Если они равны, то записывается полученная длина;

int MaxEqualityLength(string stringPart)

{

int

pi = 0,

equalityLength = 0,

strLength = stringPart.size();

string

prefix = "",

suffix = "";

for (int i = 0; i < strLength - 1; i++)

{

pi++;

prefix += stringPart[i];

suffix = stringPart[strLength - i - 1] + suffix;

if (prefix == suffix)

equalityLength = pi;

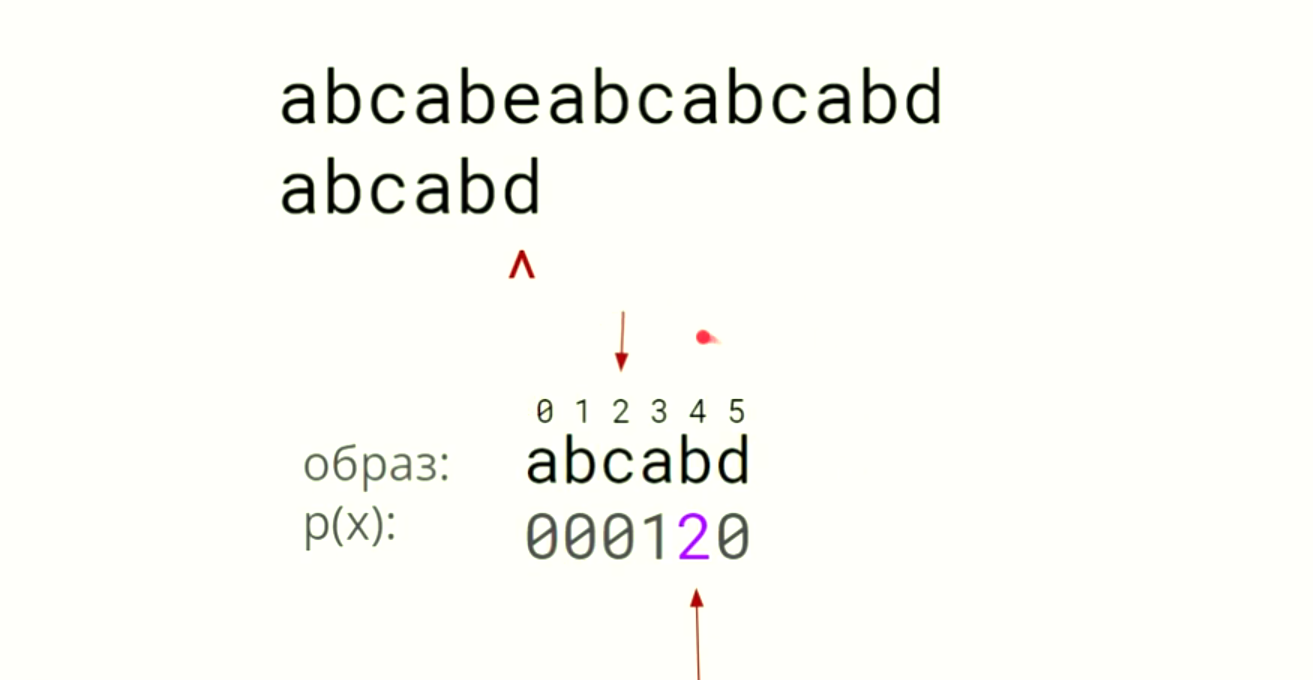
}

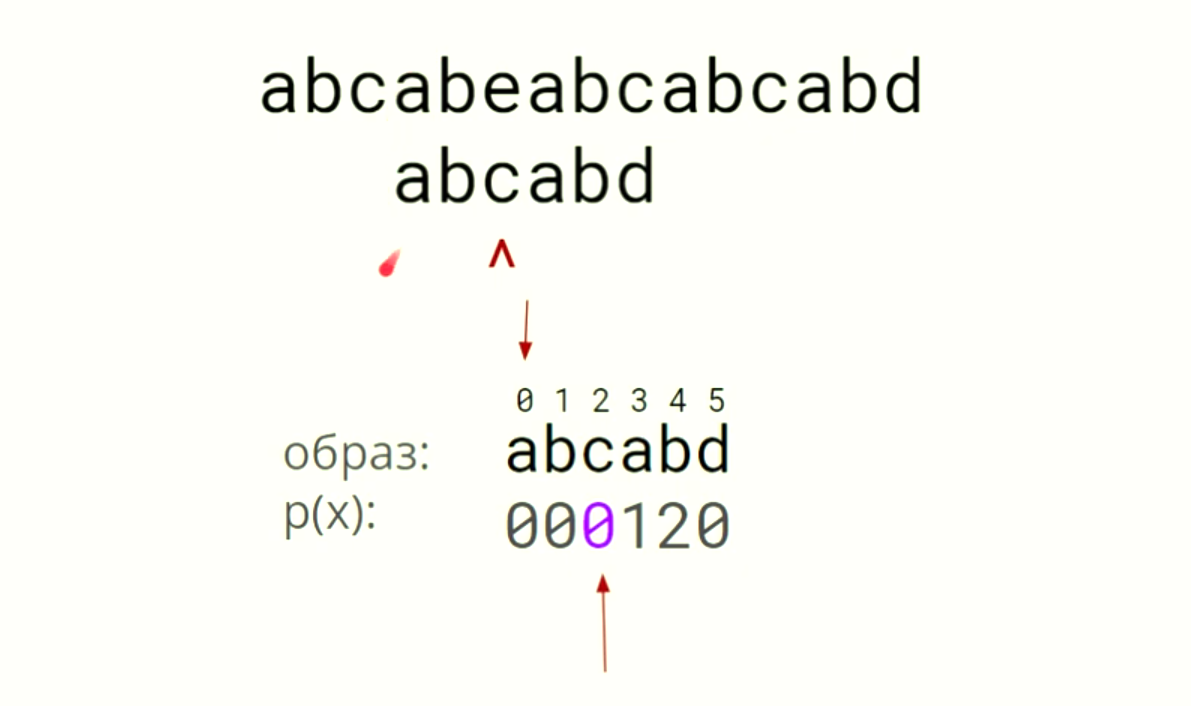
return equalityLength;

}

Toлько после того, как была сформирована таблица π, можно осуществлять поиск образа (подстроки) в строке. Поиск осуществляется при помощи сравнения элементов образа с ключом:

* Если символы совпадают, проверка продолжается – идёт переход к следующему символу.
* Если символы не совпали, то происходит переход, в котором образ «смещается» таком образом, что его позиция относительно рабочей точки будет равно значению элемента таблицы π с индексом предыдущего совпавшего элемента.





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

{

while (j > 0 && substring[j] != baseString[i])

j = pi[j - 1];

if (substring[j] == baseString[i])

j++;

if (j == subStrLength)

{

//вывод найденных элементов

}

}

После того, как элемент найден, нужно определиться является ли элемент первым вхождением: для этого идёт проверка флага на наличие найденного элемента: false – значит первое вхождение, true – не первое. Если вхождение первое, то записываем в соответствующую переменную индекс этого вхождения, затем идёт вывод индекса вхождения на экран.

if (!isElementFound)

{

isElementFound = true;

firstEntryIndex = i - j + 1;

}

entryOrderNumber++;

cout << "Entry #" << entryOrderNumber << " is found: index = "

<< i - j + 1 << endl;

В конце выполнения программы идёт проверка на то, была ли найдена подстрока: если подстрока не найдена, то выводится соответствующее сообщение на экран, затем очищается динамический массив и возвращается индекс первого вхождения.

if (!isElementFound)

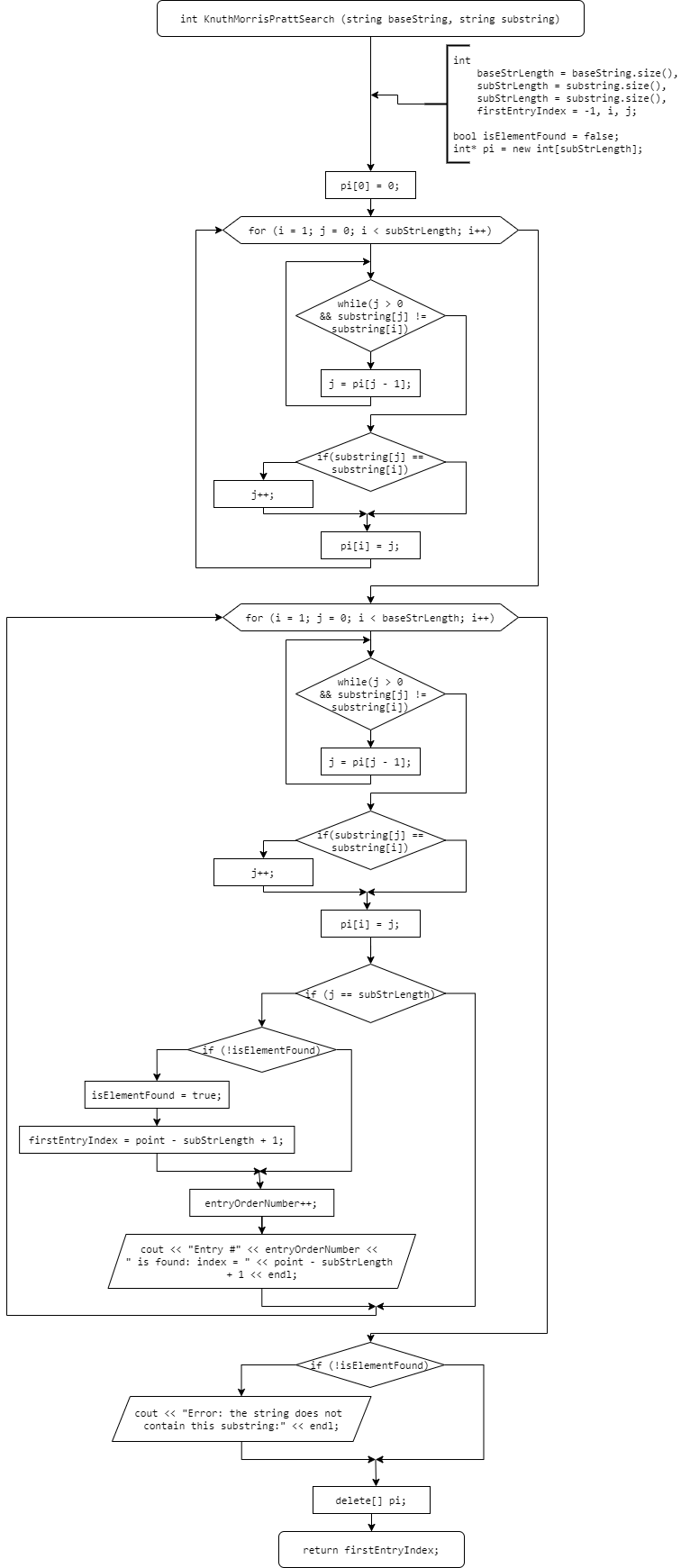
cout << "Error: the string does not contain this substring:"

<< endl;

delete[] pi;

return firstEntryIndex;

**Блок-схема** алгоритма Кнута-Морриса-Пратта:



**Алгоритм Бойера-Мура:**

Это быстрый алгоритм поиска подстроки в строке общего назначения, использующая предварительные вычисления над подстрокой, которую необходимо найти. Результат вычисления – таблица сдвигов, по которой происходит сдвиг рабочего индекса в строке.

Функция объявляется точно так же, как функция с алгоритмом Кнута-Морриса-Пратта: в качестве аргументов принимается строка и подстрока. Для удобства используются те же переменные, за исключением динамического массива (используется статический на 256 элементов – в таблице ASCII ровно 256 знаков).

int MooreSearch(string baseString, string substring)

Также используются дополнительная переменная - позиция, в которой происходит работа со строкой;

В начале выполнения функции идёт проверка на длину строки и подстроки: если подстрока короче строки и строка ненулевая, то поиск подстроки возможен.

if (baseStrLength != 0 && subStrLength != 0 && baseStrLength >= subStrLength)

{

int

point,

offset[256],

entryOrderNumber = 0;

//Выполнение поиска

}

Заполнение таблицы сдвигов:

for (int i = 0; i < 256; i++)

offset[i] = subStrLength;

for (int i = subStrLength - 2; i >= 0; i--)

if (offset[(int)((char)(substring[i]))] == subStrLength)

offset[(int)((char)(substring[i]))] = subStrLength - i - 1;

point = subStrLength - 1;

Поиск последнего символа подстроки:

while (point < baseStrLength)

{

if (substring[subStrLength - 1] != baseString[point])

point += offset[int((unsigned char)baseString[point])];

else

{

//накапливание подстроки

}

}

Накапливание подстроки:

for (int i = subStrLength - 1; i >= 0; i--)

{

if (substring[i] != baseString[point - subStrLength + i + 1])

{

point += offset[int((char)baseString[point])];

break;

}

else if (i == 0)

{

if (!isElementFound)

{

isElementFound = true;

firstEntryIndex = point - subStrLength + 1;

}

entryOrderNumber++;

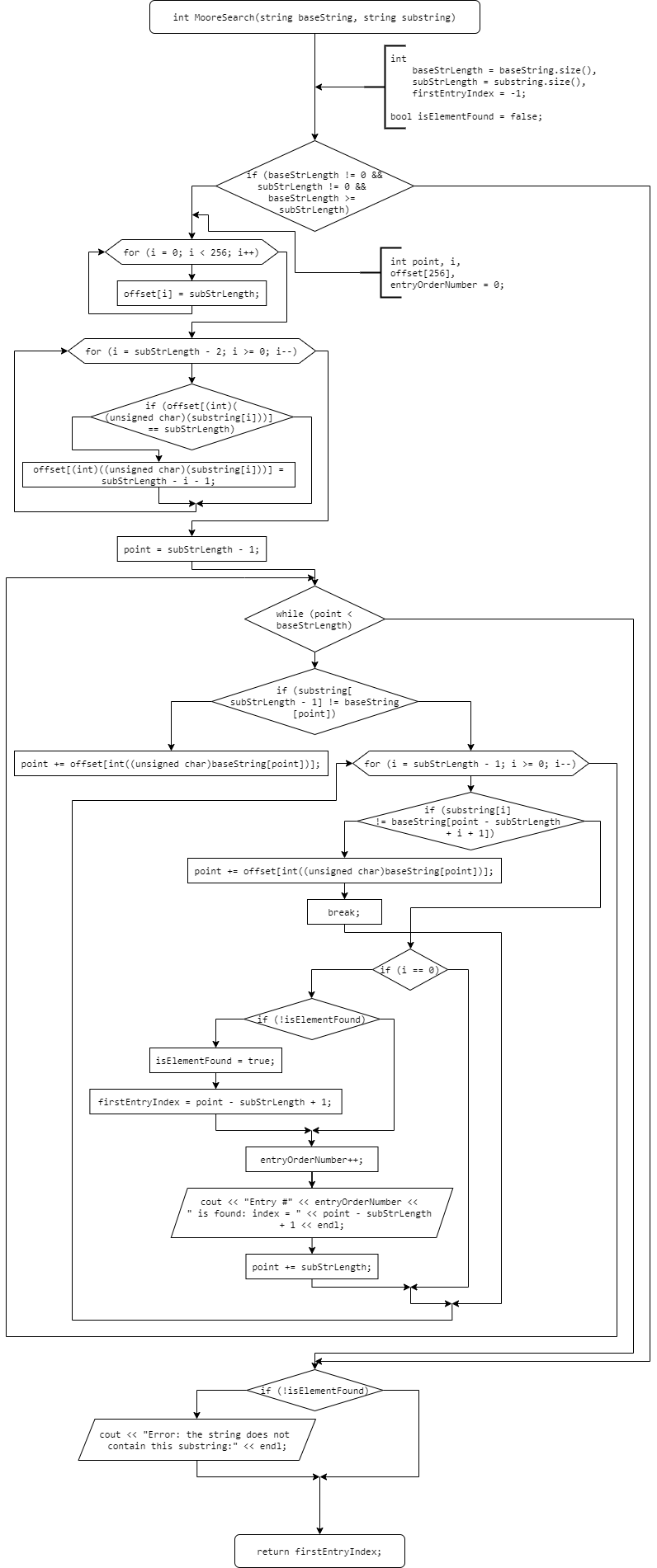
cout << "Entry #" << entryOrderNumber << " is found: index = " << point - subStrLength + 1 << endl;

point++;

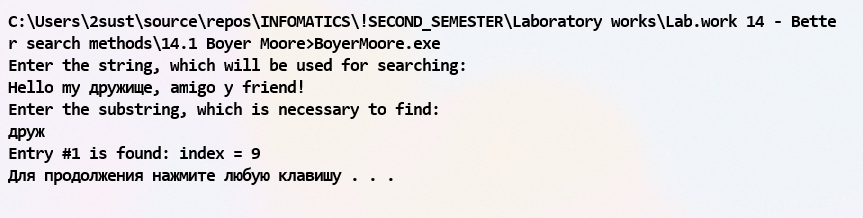
}

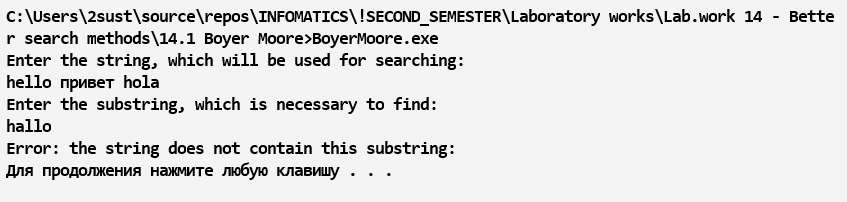
}

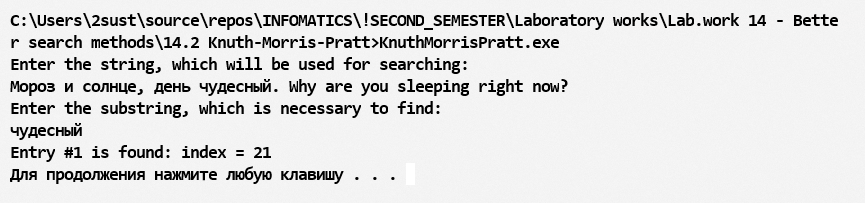
**Блок-схема** алгоритма Бойера-Мура:

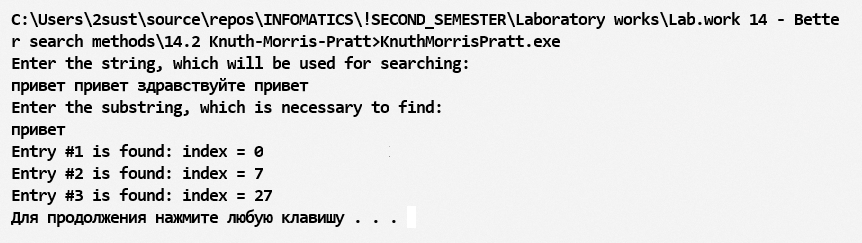


**Скриншоты** выполненных программ:









**Полный исходный код функций на языке программирования C++:**

Алгоритм Кнута-Морриса-Пратта:

#include <iostream>

#include <string>

#define INPUT\_MESSAGE "Enter the string, which will be used for searching:"

#define NEW\_INPUT\_MESSAGE "Enter the substring, which is necessary to find:"

using namespace std;

string ReadLine()

{

string line;

getline(cin, line);

return line;

}

bool CouldConsist(string baseString, string substring)

{

int

baseStrLength = baseString.size(),

subStrLength = substring.size();

return baseStrLength != 0 && subStrLength != 0 && baseStrLength >= subStrLength;

}

int MaxEqualityLength(string stringPart)

{

int

pi = 0,

equalityLength = 0,

strLength = stringPart.size();

string

prefix = "",

suffix = "";

for (int i = 0; i < strLength - 1; i++)

{

pi++;

prefix += stringPart[i];

suffix = stringPart[strLength - i - 1] + suffix;

if (prefix == suffix)

equalityLength = pi;

}

return equalityLength;

}

int\* OffsetPi(string substring)

{

int subStrLength = substring.size(),

\*pi = new int[subStrLength];

string workPart = "";

for (int i = 0; i < subStrLength; i++)

{

workPart += substring[i];

pi[i] = MaxEqualityLength(workPart);

}

return pi;

}

int KnuthMorrisPrattSearch(string baseString, string substring)

{

int

baseStrLength = baseString.size(),

subStrLength = substring.size(),

firstEntryIndex = -1,

entryOrderNumber = 0;

bool isElementFound = false;

if (CouldConsist(baseString, substring))

{

int\* pi = OffsetPi(substring);

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

{

while (j > 0 && substring[j] != baseString[i])

j = pi[j - 1];

if (substring[j] == baseString[i])

j++;

if (j == subStrLength)

{

if (!isElementFound)

{

isElementFound = true;

firstEntryIndex = i - j + 1;

}

entryOrderNumber++;

cout << "Entry #" << entryOrderNumber << " is found: index = " << i - j + 1 << endl;

}

}

delete[] pi;

}

if (!isElementFound)

cout << "Error: the string does not contain this substring:" << endl;

return firstEntryIndex;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << INPUT\_MESSAGE << endl;

string mainString = ReadLine();

cout << NEW\_INPUT\_MESSAGE << endl;

string substring = ReadLine();

KnuthMorrisPrattSearch(mainString, substring);

system("pause");

return 0;

}

Алгоритм Бойера-Мура:

#include <iostream>

#include <string>

#include "Windows.h"

#define INPUT\_MESSAGE "Enter the string, which will be used for searching:"

#define NEW\_INPUT\_MESSAGE "Enter the substring, which is necessary to find:"

using namespace std;

string ReadLine()

{

string line;

getline(cin, line);

return line;

}

int MooreSearch(string baseString, string substring)

{

int

baseStrLength = baseString.size(),

subStrLength = substring.size(),

firstEntryIndex = -1;

bool isElementFound = false;

if (baseStrLength != 0 && subStrLength != 0 && baseStrLength >= subStrLength)

{

int

point,

offset[256],

entryOrderNumber = 0;

for (int i = 0; i < 256; i++)

offset[i] = subStrLength;

for (int i = subStrLength - 2; i >= 0; i--)

if (offset[(int)((unsigned char)(substring[i]))] == subStrLength)

offset[(int)((unsigned char)(substring[i]))] = subStrLength - i - 1;

point = subStrLength - 1;

while (point < baseStrLength)

{

if (substring[subStrLength - 1] != baseString[point])

point += offset[int((unsigned char)baseString[point])];

else

{

for (int i = subStrLength - 1; i >= 0; i--)

{

if (substring[i] != baseString[point - subStrLength + i + 1])

{

point += offset[int((unsigned char)baseString[point])];

break;

}

else if (i == 0)

{

if (!isElementFound)

{

isElementFound = true;

firstEntryIndex = point - subStrLength + 1;

}

entryOrderNumber++;

cout << "Entry #" << entryOrderNumber << " is found: index = " << point - subStrLength + 1 << endl;

point += subStrLength;

}

}

}

}

}

if (!isElementFound)

cout << "Error: the string does not contain this substring:" << endl;

return firstEntryIndex;

}

int main()

{

cout << INPUT\_MESSAGE << endl;

string mainString = ReadLine();

cout << NEW\_INPUT\_MESSAGE << endl;

string substring = ReadLine();

MooreSearch(mainString, substring);

return 0;

}