**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практической работе №4**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Алгоритм Кнута-Морриса-Пратта**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Лящевская А.П. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2019

**Цель работы.**

Исследовать и реализовывать задачу поиска вхождения подстроки в строке, используя алгоритм Кнута – Морриса – Пратта.

**Постановка задачи.**

**Задание №1.**

Реализуйте алгоритм КМП и с его помощью для заданных шаблона PP (|P|≤15000|P|≤15000) и текста TT (|T|≤5000000|T|≤5000000) найдите все вхождения PP в TT.

Вход:

Первая строка - PP

Вторая строка – TT

Выход:

индексы начал вхождений PP в  TT, разделенных запятой, если PP не входит в TT, то вывести −1−1

**Sample Input:**

ab

abab

**Sample Output:**

0,2

**Задание №2.**

Заданы две строки AA (|A|≤5000000|A|≤5000000) и BB (|B|≤5000000|B|≤5000000).

Определить, является ли АА циклическим сдвигом ВВ (это значит, что АА и ВВ имеют одинаковую длину и АА состоит из суффикса ВВ, склеенного с префиксом ВВ). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка - AA

Вторая строка - BB

Выход:

Если AA вляется циклическим сдвигом BB, индекс начала строки BB в AA, иначе вывести −1−1. Если возможно несколько сдвигов вывести первый индекс.

**Sample Input:**

defabc

abcdef

**Sample Output:**

3

**Индивидуальное задание.**

Вариант 2. Оптимизация по памяти: программа должна требовать *O(m)* памяти, где *m* - длина образца. Это возможно, если не учитывать память, в которой хранится строка поиска.

**Описание алгоритма.**

Описание алгоритма №1.

На вход программе подается шаблон и текст, требуется найти все вхождения шаблона в заданный текст. В *main()* считывается строка шаблон, а затем для нее вызывается основная функция программы *void KMP()*. В функции реализован алгоритм КМП: происходит посимвольное считывание строки и сравнение её с шаблоном, используя значения префикс – функции. На вход функция получает строку шаблон *pattern,* в результате работы функции заполняется вектор *pf* для строки шаблона.

Если сравнение успешное, т.е. программа нашла вхождение шаблона в текст, то через запятую выписывается позиция шаблона в тексте, если вхождений не было обнаружено, то программа выведет -1.

Для более понятной работы алгоритма рассмотрим пример работы программы для строк «abaabb» и «aba». Расчет значения префикс – функции представлен на рис. 1.

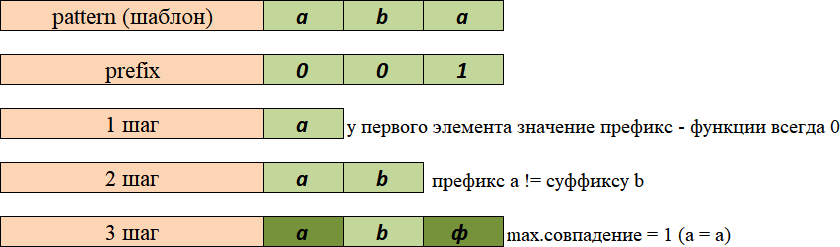


Рисунок 1 – Расчет значения префикс-функции

Дальнейшие рассуждения предоставлены на рис.2.

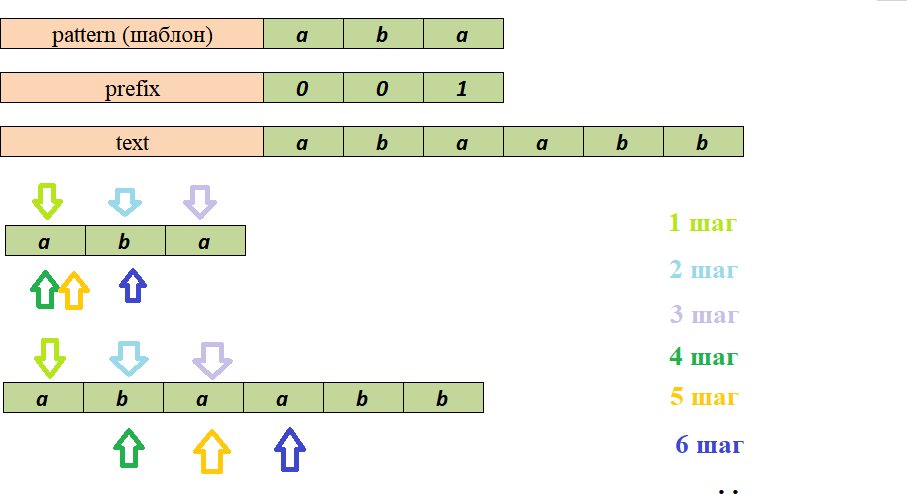


Рисунок 2- Ход работы алгоритма КМП

Пока символы одинаковые оба указателя двигаем на единицу вперёд, если шаблон закончился, то возвращаемся по префикс – функции к a и опять сравниваем значения. Алгоритм работает до конца строки.

Описание алгоритма №2.

На вход программе подается две строки *A* и *B*, требуется найти является ли *A* циклическим сдвигом *B*. В *main()* происходит считывание строк, затем сравнение их длин, если они разные, то программа возвращает -1 и заканчивает работу, иначе вызывается основная функция программы *int KMP()*. Сначала в функции происходит вычисление префикс-функции, аналогично ее вычислению в первой программе (см. описание алгоритма №1). Затем пока мы не дошли до конца строки шаблона, в данном случае строки *A*, последовательно сравниваем символы строк.

1. *Если символы равны =.*

Переходим на следующие символы в каждой строке. Для того чтобы найти очередные индексы символов для сравнения, выполняем операцию “остаток от деления”, т.е. учитываем циклический сдвиг. Процесс окончится, если *А* циклический сдвиг *В* или символы строк не совпадут.

1. *Если символы не равны !=.*

Переходим на следующие символы в шаблоне, используя значения префикс-функции, вычисленное ранее.

Если прошли весь шаблон, и *А* является циклическим сдвигом *В*, то возвращаем индекс начала строки *B* в строке *А*, иначе выводится -1*.*

**Описание структур данных.**

Таблица 1 – Описание структур данных алгоритма №1

|  |  |
| --- | --- |
| Название | Назначение |
| *vector<int> pf* | Вектор для хранения префикс-функции |
| *string pattern* | Строковый класс для хранения шаблона |
| *string S* | Строковый класс для хранения строки |

Таблица 2 – Описание структур данных алгоритма №2

|  |  |
| --- | --- |
| Название | Назначение |
| *vector<int> pf* | Вектор для хранения префикс-функции |
| *string A* | Первая строка |
| *string B* | Вторая строка |

**Описание функций.**

Таблица 3 – Описание функций алгоритма №1

|  |  |  |
| --- | --- | --- |
| Сигнатура | Параметры | Описание |
| *void KMP(string &pattern, string &S)* | *string &pattern*-строка шаблон *string &S*-строка поиска | Принимает на вход строку шаблон pattern и сроку поиска S. Функция реализовывает алгоритм КМП. Посимвольно сравнивает ее с шаблоном, используя полученное значение префикс-функции. |

Таблица 4 – Описание функций алгоритма №2

|  |  |  |
| --- | --- | --- |
| Сигнатура | Параметры | Описание |
| *int KMP(string &S, string &pattern)* | *string &S-*первая строка*, string &pattern-*вторая строка | Функция принимает две строки и проверяет является ли первая *S* циклическим сдвигом второй *pattern.* Возвращает -1 в случае неудачи или индекс начала строки *pattern* встроке *text.* |

**Сложность алгоритма.**

В начале метод КМП задает значения префикс функции для шаблона. Он проходит по всем символам один раз, следовательно, временная сложность O(|N|), где N – количество символов шаблона. Затем мы проходимся по всем символам текста, вычисляя значения префикс функции, следовательно, временная сложность O(|М|), где М – количество символов текста. Значит общая временная сложность алгоритма O(|M| + |N|).

Память в данном алгоритме выделяется только под значения префикс функции шаблона, поэтому по памяти сложность алгоритма можно оценить, как O(|N|), где N – количество символов шаблона.

**Тестирование.**

Таблица 5 – Тестирование алгоритма №1

|  |  |  |
| --- | --- | --- |
| № теста | Тест | Результат |
| 1 | ab  abab | 0,2 |
| 2 | we  wertwengkwe | 0,4,9 |
| 3 | lpkps  knfkndlngvflspe | -1 |
| 4 | an  annaalenann | 0,8 |

Таблица 6 – Тестирование алгоритма №2

|  |  |  |
| --- | --- | --- |
| № теста | Тест | Результат |
| 1 | aslf  dfdg | -1 |
| 2 | defabc  abcdef | 3 |
| 3 | anna  nnaa | 1 |
| 4 | wer  erw | 1 |

**Вывод.**

В ходе выполнения лабораторной работы были закреплены теоретические знания о строках, методах работы с ними, а также изучен алгоритм поиска подстроки в строке – алгоритм Кнута-Морриса-Пратта.

**ПРИЛОЖЕНИЕ А**

**Исходный код программы lr4\_1.cpp**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

bool KMP(const string& pattern, const string& S) //Возвращает логическую переменную означающую была ли найдена подстрока

{

vector<int> pf (pattern.length()); //Вектор для значений префикс-функции шаблона

cout << "Значения префикс функции шаблона: " << pattern << endl;

pf[0] = 0;

cout << "1) Для элемента " << pattern[0] << " = 0" << endl << endl;

for (int k = 0, i = 1; i < pattern.length(); ++i) //Установим значения для pf

{

cout << i+1 <<") Для элемента " << pattern[i] << ":" << endl;

while ((k > 0) && (pattern[i] != pattern[k])) //Найдем одинаковый элемент для текущего используя значения pf

{

cout << "Значение предыдущей п-функции для элемента " << pattern[k-1] << " = " << pf[k-1] << endl;

k = pf[k-1];

}

if (pattern[i] == pattern[k]) //Если такой нашелся, то увеличиваем переменную префик-функции

k++;

pf[i] = k; //Установим значение для элемента префикс-функции в значение ее переменной

cout << "Итоговое значение п-функции для элемента " << pattern[i] << " = " << k << endl << endl;

}

bool first = true; //Несет инфу о нахождении подстроки в строке

cout << "Значения префикс функции строки поиска: " << S << endl;

for(int i = 0, k = 0; i < S.length(); ++i) //Найдем подстроку в строке

{

cout << i+1 <<") Для элемента " << S[i] << ":" << endl;

while ((k > 0) && (pattern[k] != S[i])) //Пока значение префикс функции больше нуля или мы не нашли одинаковых элементов

{

cout << "Значение предыдущей п-функции для элемента " << pattern[k-1] << " = " << pf[k-1] << endl;

k = pf[k-1]; //Переходим к рассмотрению подпрефикса для текующего префикса

}

if (pattern[k] == S[i]) //Если нашли одинаковые элементы, то увеличиваем переменную длины вхождения pattern в S

k++;

cout << "Итоговое значение п-функции для элемента " << S[i] << " = " << k << endl << endl;

if (k == pattern.length()) //Если нашли полную строку pattern в S, то возращаемся

{

cout << "Найдена подстрока в позиции : " << i - pattern.length() + 1 << endl << endl;

first = false;

}

}

return first; //Не нашли pattern в S

}

int main()

{

setlocale(LC\_ALL, "rus");

string pattern;

string S;

cin >> pattern >> S;

if(KMP(pattern, S))

cout << -1;

return 0;

}

**ПРИЛОЖЕНИЕ Б**

**Исходный код программы lr4\_2.cpp**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

int KMP(const string& S, const string& pattern) //Определяет является ли вторая строка циклическим сдвигом первой

{

if(S.length() != pattern.length()) //Если длины не равны, то не является

return -1;

vector<int> pf (pattern.length()); //Вектор для значений префикс-функции шаблона

cout << "Значения префикс функции шаблона: " << pattern << endl;

pf[0] = 0;

cout << "1) Для элемента " << pattern[0] << " = 0" << endl << endl;

for (int k = 0, i = 1; i < pattern.length(); ++i) //Установим значения для pf

{

cout << i+1 <<") Для элемента " << pattern[i] << ":" << endl;

while ((k > 0) && (pattern[i] != pattern[k])) //Найдем одинаковый элемент для текущего используя значения pf

{

cout << "Значение предыдущей п-функции для элемента " << pattern[k-1] << " = " << pf[k-1] << endl;

k = pf[k-1];

}

if (pattern[i] == pattern[k]) //Если такой нашелся, то увеличиваем переменную префик-функции

k++;

pf[i] = k; //Установим значение для элемента префикс-функции в значение ее переменной

cout << "Итоговое значение п-функции для элемента " << pattern[i] << " = " << k << endl << endl;

}

cout << "Значения префикс функции строки поиска: " << S << endl;

for (int k = 0, i = 0; i < 2\*S.length(); ++i) //Найдем циклический сдвиг в строке S+S

{

cout << i+1 <<") Для элемента " << S[i%S.length()] << ":" << endl;

while ((k > 0) && (pattern[k] != S[i%S.length()])) //Пока значение префикс функции больше нуля или мы не нашли одинаковых элементов

{

cout << "Значение предыдущей п-функции для элемента " << pattern[k-1] << " = " << pf[k-1] << endl;

k = pf[k-1]; //Переходим к рассмотрению подпрефикса для текующего префикса

}

if (pattern[k] == S[i%S.length()]) //Если нашли одинаковые элементы, то увеличиваем переменную длины вхождения pattern в S

k++;

cout << "Итоговое значение п-функции для элемента " << S[i%S.length()] << " = " << k << endl << endl;

if (k == pattern.length()) //Если нашли полную строку pattern в S, то возращаемся

return (i + 1)%pattern.length();

}

return -1; //Не нашли pattern в S

}

int main()

{

setlocale(LC\_ALL, "rus");

string A;

string B;

cin >> A >> B;

int result;

if((result = KMP(A, B)) == -1)

cout << "Циклический сдвиг не найден." << endl;

else

cout << "Циклический сдвиг найден со смещением " << result << endl;

return 0;

}