Семестровая работа

Алгоритм Кнута-Морриса-Пратта

Выполнил:

Юсупов М.Р. 11-002

2021г

**Определение:**

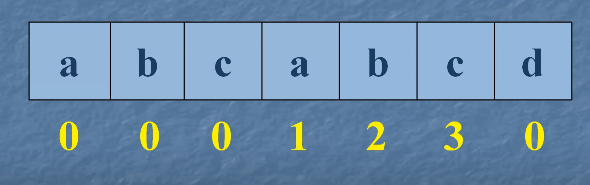
Алгоритм Кнута — Морриса — Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке. Алгоритм был разработан Д. Кнутом и В. Праттом и, независимо от них, Д. Моррисом. Результаты своей работы они опубликовали совместно в 1977 году.

**Постановка задачи:**

Даны образец (строка) T и строка S. Требуется определить индекс, начиная с которого образец T содержится в строке S. Если T не содержится в S — вернуть индекс, который не может быть интерпретирован как позиция в строке (например, отрицательное число). При необходимости отслеживать каждое вхождение образца в текст имеет смысл завести дополнительную функцию, вызываемую при каждом обнаружении образца.

**Принцип устройства. Особенности:**

Для решения данной задачи необходимо построить массив pi с помощью префикс-функции. Так, Префикс-функция для i-ого символа образа возвращает некоторое значение. Это значение будем хранить в pi[i]. Префикс-функция для i-ого символа образа возвращает значение, равное максимальной длине совпадающих префикса и суффикса подстроки в образе, которая заканчивается i-м символом. Тем самым в i-ой ячейке массива pi будет хранится максимальная длинна совпадающего префикса и суффикса, исключая при этом случай, когда суффикс и префикс имеет длину образа, так как это очевидно будет максимальная длина во всех случаях.



На данном примере можем видеть работу префикс-функции.

Главным преимуществом данного алгоритма в том, что он работает за линейное время O(n), где n – это длина строки + длина образа. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

**Оценка временной сложности:**

Алгоритм используют префикс-функцию, которая создает массив pi за линейное время от количество символов в строке, проходя по каждому символу за константное время и записывая в массив значение максимальной длины совпадения суффикса и префикса, то есть сложность префикс-функции T=O(S), где S- строка. Так, в алгоритме КМП на вход подается строка и образ, по которому нужно найти совпадения в строке, и для того, чтобы использовать префикс функцию мы склеиваем эти две строки, разделяя их символом, который не встречается в алфавите, например «@». Таким образом, строка, подающая в префикс-функцию равна S=P+T, где P-это строка, а T- это образ, следовательно сложность КМП T=O(P+T).

На данном графике можно заметить, что время работы линейно возрастает от увеличения размера входных данных.

**Вывод:**

Главным преимуществом данного алгоритма это легкая реализация и сложность по времени, которую она реализует. Недостатком же алгоритма можно считать, что в некоторых случаях, когда образец не содержит повторяющихся фрагментов, данный алгоритм не сильно увеличивает скорость, относительно ,например алгоритма Бойера-Мура. Данный алгоритм широко используется в приложения, где необходимо найти подстроку в строке, это применимо для приложений, хранящий большие текстовые документы.

**Литература:**

1.Кормен. Алгоритмы - построение и анализ.

2.Седжвик. Фундаментальные алгоритмы на C++

3.Препарата, Шеймос. Вычислительная геометрия: введение

**Код:**

#include <iostream>

#include <vector>

**using** **namespace** std**;**

vector**<**int**>** prefix\_function**(**string s**)** **{**

vector**<**int**>** pi**(**s**.**length**(),**0**);**//создаем массив префикс функции,заполняем нулями

int j **=** 0**;**//индекс отвечающий за префикс строки

int i **=** 1**;**//индекс отвечающий за суффикс строки

**while(**i**!=**s**.**length**())** **{**//пока индекс i не дошел до конца строки

**if(**s**[**i**]==**s**[**j**])** **{**//если символы на индексе префикса совпадают с суффиксом,то добавляем 1 в массив на i место

pi**[**i**]** **=** j**+**1**;**//так как увеличтся следующие значение массива может максимум на 1

i**++;**

j**++;**

**}**

**else** **if(**j**==**0**)** **{**//если же не совпадают,то проверяем стоит ли индекс префикса на начальной позиции

pi**[**i**]=**0**;**//если да,то ставим 0 в массив , то есть префикс и суффикс не совпадают

i**++;**

**}**

**else** **{**//если же индекс стоит не на начальном месте,смещаем его относительно массива pi

j**=**pi**[**j**-**1**];**

**}**

**}**

**return** pi**;**

**}**

int main**()**

**{**

string str**;** //строка,по которой ищем образ

string pattern**;**//образ(подстрока)

cin**>>**str**>>**pattern**;**

int pattern\_size **=** pattern**.**length**();**

**if(**pattern\_size**>**str**.**length**())** **{**//если размер подстроки больше самой строки выводим -1

cout**<<**"-1"**<<**endl**;**

**}**

**else** **{**

vector**<**int**>** pi **=** prefix\_function**(**pattern**+**'@'**+**str**);**//для префикс функции склеиваем str и pattern символом,не используемый в входных данных

bool flag **=** **true;**//проверка на то,есть ли образ в строке

**for(**int i**=**0**;**i**<**str**.**length**();**i**++)** **{**

**if(**pi**[**pattern\_size**+**1**+**i**]==**pattern\_size**)** **{**

flag **=** **false;**

cout**<<**"From "**<<**i**-**pattern\_size**+**1**<<**" index to "**<<**i**<<**endl**;**

**}**

**}**

**if(**flag**)** **{**//если образа нет,выводим -1.

cout**<<**"-1"**<<**endl**;**

**}**

**}**

**return** 0**;**

**}**

**Ссылка на репозиторий:** **https://github.com/MarselYsup/Itis-aicd**