МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное

образовательное учреждение

высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

**по дисциплине**

**“ Алгоритмы построение и анализ”**

Выполнил: студент гр. ФИб-3302-51-00 ё \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ПМиИ Разова Е. В. \_\_\_\_\_\_\_\_\_\_\_

Киров 2020

**Алгоритм Кнута-Морриса-Пратта**

Алгоритм был разработан Д. Кнутом и В. Праттом и, независимо от них, Д. Моррисом. Результаты своей работы они опубликовали совместно в 1977 году.

Алгоритм Кнута-Морриса-Пратта использует префикс-функцию для оптимизации сдвигов при наивном поиске.

Дана цепочка (текст) S и образец (подстрока) P.

Построим строку T=P#S, где # - любой символ, не входящий в алфавит P и T.

Найдем для строки T префикс-функцию p.

Заметим, что для любого i p[i]£|P| благодаря разделительному символу #.

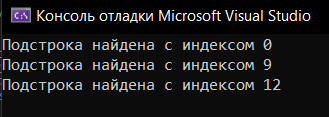
Если в какой-то позиции p[i]=|P|, где i>|P|, то в этой позиции начинается очередной вхождение образца в цепочку.

**Задание 1**: Дана цепочка S и образец P. Требуется, используя алгоритм Кнута-Морриса-Пратта,

А) найти первое вхождение P в S;

Б) найти последнее вхождение P в S;

С) найти все позиции, начиная с которых P входит в S;



#include <iostream>

#include <clocale>

#include <string>

using namespace std;

void prefix(char\* pat, int M, int\* lps);

// Печатает вхождения txt [] в pat []

void KMPSearch(char\* pat, char\* txt)

{

int M = strlen(pat);

int N = strlen(txt);

// создать lps [], и заполнить префикс функцией

int lps[1000];

prefix(pat, M, lps);

int i = 0; // индекс для txt []

int j = 0; // индекс для pat []

while (i < N) {

if (pat[j] == txt[i]) {

j++;

i++;

}

if (j == M) {

cout << "Подстрока найдена с индексом " << i - j << endl;

j = lps[j - 1];

}

// несоответствие после j совпадений

else if (i < N && pat[j] != txt[i]) {

if (j != 0)

j = lps[j - 1];

else

i = i + 1;

}

}

}

// префикс функция

void prefix(char\* pat, int M, int\* lps)

{

int len = 0;

lps[0] = 0;

int i = 1;

while (i < M) {

if (pat[i] == pat[len]) {

len++;

lps[i] = len;

i++;

}

else

{

if (len != 0) {

len = lps[len - 1];

}

else

{

lps[i] = 0;

i++;

}

}

}

}

int main()

{

setlocale(0, "");

char txt[] = "AABAACAADAABAABA";

char pat[] = "AABA";

KMPSearch(pat, txt);

return 0;

}

**Количество различных подстрок в строке**

**Задание 2**: Дана строка S длины n. Требуется посчитать количество её различных подстрок.

*Идея решения на основе использования префикс-функции:*

Будем решать эту задачу итеративно (i=0..S.length()-1). А именно, научимся, зная текущее количество различных подстрок, пересчитывать это количество при добавлении в конец одного символа.

Итак, пусть k – текущее количество различных подстрок строки S, и мы добавляем в конец символ c. Очевидно, в результате могли появиться некоторые новые подстроки, оканчивавшиеся на этом новом символе c. А именно, добавляются в качестве новых те подстроки, оканчивающиеся на символе c и не встречавшиеся ранее.

Возьмём строку T[i] = S[i] + c, где с= S[i], и инвертируем её (запишем символы в обратном порядке). Наша задача – посчитать, сколько у строки T таких префиксов, которые не встречаются в ней более нигде. Но если мы посчитаем для строки T префикс-функцию и найдём её максимальное значение pmax, то, очевидно, в строке T встречается (не в начале) её префикс длины pmax, но не большей длины. Понятно, префиксы меньшей длины уж точно встречаются в ней.

Итак, мы получили, что число новых подстрок, появляющихся при дописывании символа c, равно S.length() + 1 - pmax.

Таким образом, для каждого дописываемого символа мы за O(n) можем пересчитать количество различных подстрок строки. Следовательно, за O(n2) мы можем найти количество различных подстрок для любой заданной строки.

*Пример*:

S=abac

1. i=0, S – пустая строка, T=a, T’=a, p: [0], k= S.length() + 1 - pmax=0+1-0=1.

Действительно, существует только одна подстрока – a.

1. i=1, S=a, T=ab, T’=ba, p: [0, 0], k= S.length() + 1 - pmax=1+1-0=2

Действительно, существует три подстроки строки T {a, b, ab}, но новых только две - {b, ab}

1. i=2, S=ab, T=aba, T’=aba, p: [0, 0, 1], k= S.length() + 1 - pmax=2+1-1=2

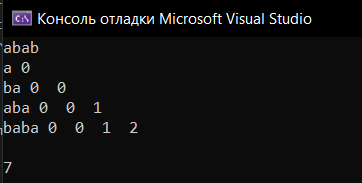
Действительно, существует пять подстрок строки T {a, b, ab, ba, aba}, но новых только две - {ba, aba}

1. i=2, S=aba, T=abaс, T’=сaba, p: [0, 0, 0, 0], k= S.length() + 1 - pmax=3+1-0=4

Действительно, существует девять подстрок строки T {a, b, ab, ba, aba, c, ac, bac, abac}, но новых только четыре - {c, ac, bac, abac}

Ответ: всего различных непустых подстрок существует 1+2+2+4=9 – это подстроки {a, ab, aba, abac, b, ba, bac, ac, c}.

Стоит заметить, что совершенно аналогично можно пересчитывать количество различных подстрок и при дописывании символа в начало, а также при удалении символа с конца или с начала.



#include <iostream>

#include <string>

#include <regex>

using namespace std;

int countnews = 0;

vector<int> prefix\_function(const string& s)

{

cout << s;

int len = s.length();

vector<int> p(len);

int k = 0; // счетчик

for (int i = 1; i < len; i++)

{

if (s[k] != s[i])

{

// повторная проверка при k = 0

if (s[0] != s[i]) k = 0;

else k = 1;

}

else

{

k++; // если символы совпадают -> увеличиваем значение счетчика

}

p[i] = k; // значение счетчик в вектор

}

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

cout << " " << p[i] << " ";

if (p[i]==0) countnews++;

}

cout << endl;

return p;

}

int main()

{

string s = "abab";

string tmps = "";

int answer = 0;

cout << s << endl;

for (int i = 0; i < 4; i++) {

tmps += s[i];

reverse(tmps.begin(), tmps.end());

prefix\_function(tmps); //вовзращает кол-во новых

reverse(tmps.begin(), tmps.end());

answer += countnews;

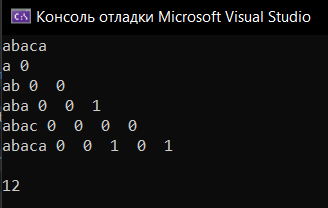
countnews = 0;

}

cout << endl << answer << endl;

}

**Задание 3**: Реализовать данные модификации. Выполнить сравнительный анализ.



#include <iostream>

#include <string>

#include <regex>

using namespace std;

int countnews = 0;

vector<int> prefix\_function(string s)

{

cout << s;

int len = s.length();

vector<int> p(len);

int k = 0; // счетчик

reverse(s.begin(), s.end());

for (int i = 1; i < len; i++)

{

if (s[k] != s[i])

{

// повторная проверка при k = 0

if (s[0] != s[i]) k = 0;

else k = 1;

}

else

{

k++; // если символы совпадают -> увеличиваем значение счетчика

}

p[i] = k; // значение счетчик в вектор

}

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

cout << " " << p[i] << " ";

if (p[i] == 0) countnews++;

}

cout << endl;

return p;

}

int main()

{

string s = "abaca";

string tmps = "";

int answer = 0;

cout << s << endl;

for (int i = 0; i < 5; i++) {

tmps += s[i];

//reverse(tmps.begin(), tmps.end());

prefix\_function(tmps); //вовзращает кол-во новых

//reverse(tmps.begin(), tmps.end());

answer += countnews;

countnews = 0;

}

cout << endl << answer << endl;

//модификация: избавились от переворачиваний, просто префикс функция ищется с конца

}

**Задание 4**: Дана строка S длины n. Требуется найти самое короткое её "сжатое" представление, т.е. найти такую строку T наименьшей длины, что S можно представить в виде конкатенации одной или нескольких копий T.

*Идея решения на основе использования префикс-функции:*

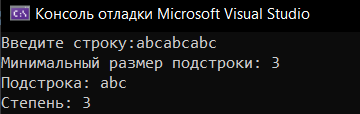
Очевидно, что проблема является в нахождении длины искомой строки T. Зная длину, ответом на задачу будет, например, префикс строки S этой длины.

Найдем по строке S префикс-функцию. Рассмотрим её последнее значение, т.е. p[n-1], и введём обозначение k = n - p[n-1]. Покажем, что если n делится на k, то это k и будет длиной ответа, иначе эффективного сжатия не существует, и ответ равен n.

Действительно, пусть n делится на k. Тогда строку можно представить в виде нескольких блоков длины k, причём, по определению префикс-функции, префикс длины n-k будет совпадать с её суффиксом. Но тогда последний блок должен будет совпадать с предпоследним, предпоследний - с предпредпоследним, и т.д. В итоге получится, что все блоки совпадают, и такое k действительно подходит под ответ.

Покажем, что этот ответ оптимален. Действительно, в противном случае, если бы нашлось меньшее k, то и префикс-функция на конце была бы больше, чем n-k, т.е. пришли к противоречию.

Пусть теперь n не делится на k. Покажем, что отсюда следует, что длина ответа равна n. Докажем от противного – предположим, что ответ существует, и имеет длину p (p делитель n). Заметим, что префикс-функция необходимо должна быть больше n-p, т.е. этот суффикс должен частично накрывать первый блок. Теперь рассмотрим второй блок строки; т.к. префикс совпадает с суффиксом, и префикс, и суффикс покрывают этот блок, и их смещение друг относительно друга, k не делит длину блока p (а иначе бы k делило n), то все символы блока совпадают. Но тогда строка состоит из одного и того же символа, отсюда k=1, и ответ должен существовать, т.е. так мы придём к противоречию.



#include <iostream>

#include <vector>

#include <string>

#include <clocale>

using namespace std;

vector<int> findsubstring(string s)

{

int n = (int)s.size();

vector<int> pi(n);

int tmpcount = 0;

for (int i = 1; i < n; i++)

{

int j = pi[i - 1];

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

j = pi[j - 1];

}

if (s[i] == s[j]) {

j++;

}

pi[i] = j;

}

return pi;

}

int main()

{

setlocale(0, "");

cout << "Введите строку:";

string s;

cin >> s;

vector <int> p = findsubstring(s);

cout << "Минимальный размер подстроки: ";

cout << s.size() - p.back() << endl;

cout << "Подстрока: ";

for (int i = 0; i < s.size() - p.back(); i++) {

cout << s[i];

}

cout << "\nСтепень: "<< (s.size() / (s.size() - p.back())) ;

return 0;

}