# Лабораторная работа «Алгоритм Кнута-Морриса-Пратта»

# Выполнил Мишутин Андрей 381506-1

**Введение**

**Задача:**

1. Написать программу, реализующую наивный алгоритм поиска в тексте и алгоритм Кнута-Морриса-Пратта подстроки в тексте.
2. Написать программу, реализующую эти алгоритмы, для проведения экспериментов, в которых можно выбирать:

* Количество букв в алфавите
* Способ независимого друг от друга задания слов X и Y из числа следующих:
* Непосредственный ввод слова
* Образования слова данной длины , составленного из выбранных букв алфавита, встречающихся равновероятно
* Образование слова вида (B1B2B3..Bs)^k на основе ввода числа k и слова B1B2..Bs

Выходом данной программы должно быть время работы Т1, тривиального алгоритма и время работы T2 , алгоритма Кнута-Морриса-Пратта.

**Цель:** реализация, сравнение и анализ алгоритмов поиска в тексте, основываясь на результатах экспериментов.

**Постановка задачи:**

Для слов Y = Y1 Y2 … Yn и X = X1 X2 … Xm в алфавите A найти все вхождения слова Y в X. В дальнейшем под решением этой задачи мы будем понимать такую функцию f: {1, 2, … ,m}®{0, 1, 2, … , n}, что f[j] = n, если Y1 Y2 … Yn =X j-(n-1) X j-(n-2) … X j (на j-ой букве слова X заканчивается очередное вхождение в него слова Y), и f[j]<n в противном случае. Для избавления псевдокодов алгоритмов от мешающих их пониманию деталей, мы будем в дальнейшем считать, что 1≤n≤m.

**Описание алгоритмов**

**Наивный алгоритм поиска фрагмента в тексте**

Суть данного алгоритма иллюстрируется рамкой длины n, движущейся по слову X. Если фрагмент слова X, находящийся в рамке, совпадает со словом Y, это означает, что очередное вхождение слова Y в X найдено.

|  |
| --- |
| procedure SFT\_TRIVIAL(X; m; Y; n; var f); |
| begin |
| for i:= 1 to n-1 do f[i]:= 0; |
| for i:= n to m do begin |
| s:= 1; |
| while (Y[s] =X[i-(n-s)]) &(s<n) do s:= s+1; |
| if (s=n)& (Y[n] =X[i]) then f[i]:= n else f[i]:= 0; |
| end; |
| end; |

Временная сложность наивного алгоритма есть O(n×m).

**Алгоритм Кнута-Морриса-Пратта**

В общем виде алгоритм Кнута-Морриса-Пратта строит функцию откатов

fY: {1, 2, … ,n}®{0, 1, 2, … , n-1} для поданного на его вход слова Y = Y1 Y2 … Yn. Таким образом, fY[i] является длиной наибольшего собственного суффикса слова Y = Y1 Y2 … Yn, который является также и его префиксом.

|  |
| --- |
| procedure KMP(Y; n; var fY); |
| begin |
| i:= 1; fY[1]:= 0; |
| while i<n do begin |
| j:= fY[i]; |
| while (Y[j+1] ¹ Y[i+1]) &(j>0) do j:= fY[j]; |
| if Y[j+1] = Y[i+1] then f[i+1]:= j+1 else f[i+1]:= 0; |
| i:= i+1; |
| end; |
| end; |

Тогда исходную задачу поиска всех вхождений слова Y = Y1 Y2 … Yn в слово (текст) X = X1 X2 … Xm решит построение функции откатов fZ для слова Z=Y#X, где # является произвольной буквой, не принадлежащей алфавиту A. В этом случае искомую функцию f можно найти, положив f[i]=fZ[i+n+1], i = 1, …, m. Сохраняя неизменной суть данного построения функции f, удается организовать его так, чтобы не было необходимости вводить в алфавит новую букву. Таким образом, алгоритм Кнута-Морриса-Пратта поиска всех вхождений слова Y=Y1 Y2 … Yn в X=X1 X2 … Xm можно представить в следующем виде.

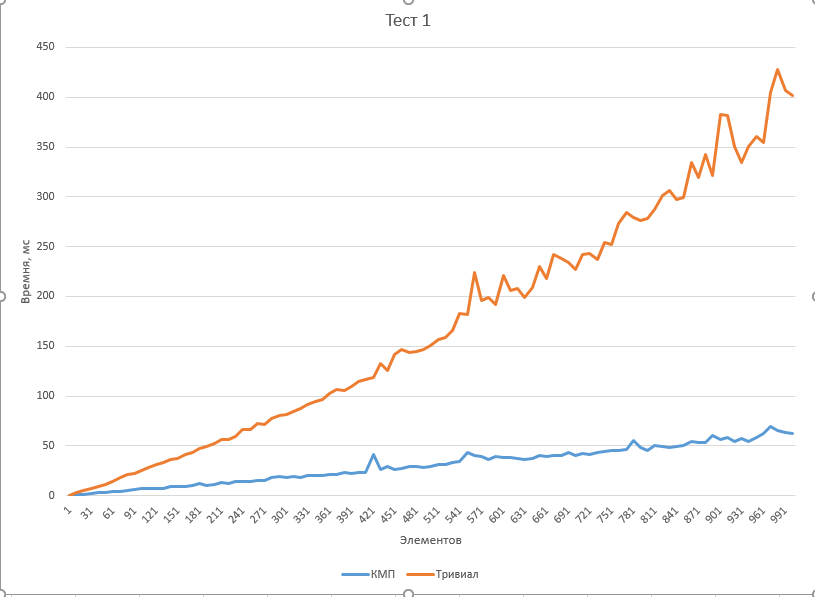
|  |
| --- |
| procedure SFT\_KMP(X; m; Y; n; var f); |
| begin |
| KMP(Y, n, fY); |
| i:= 1; if Y[1]=X[1] then f[1]:= 1 else f[1]:= 0; |
| while i<m do begin |
| j:= f[i]; |
| if j=n then j:= fY[j]; |
| while (Y[j+1] ¹ X[i+1]) & (j>0) do j:= fY[j]; |
| if Y[j+1] = X[i+1] then f[i+1]:= j+1 else f[i+1]:= 0; |
| i:= i+1; |
| end; |
| end; |

Временная сложность алгоритма Кнута-Морриса-Пратта есть O(n+m).

**Эксперименты**

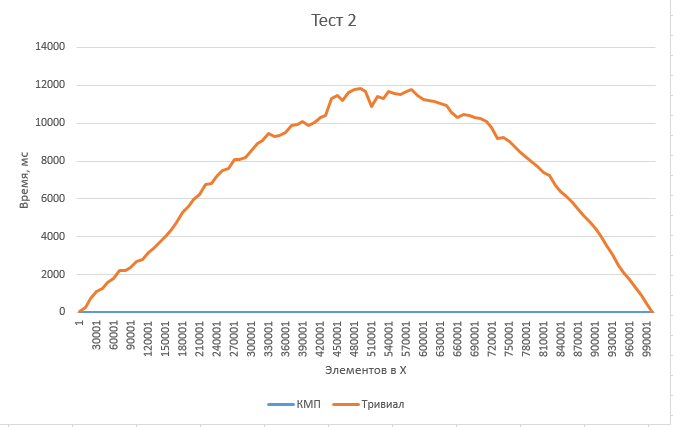
**Всего было проедено 3 серии экспериментов.**

1. Y=(ab)k и X=(ab)1000\*k, k = 1, … ,1001 с шагом 10 (нарисовать графики функций T1(k) и T2(k)).



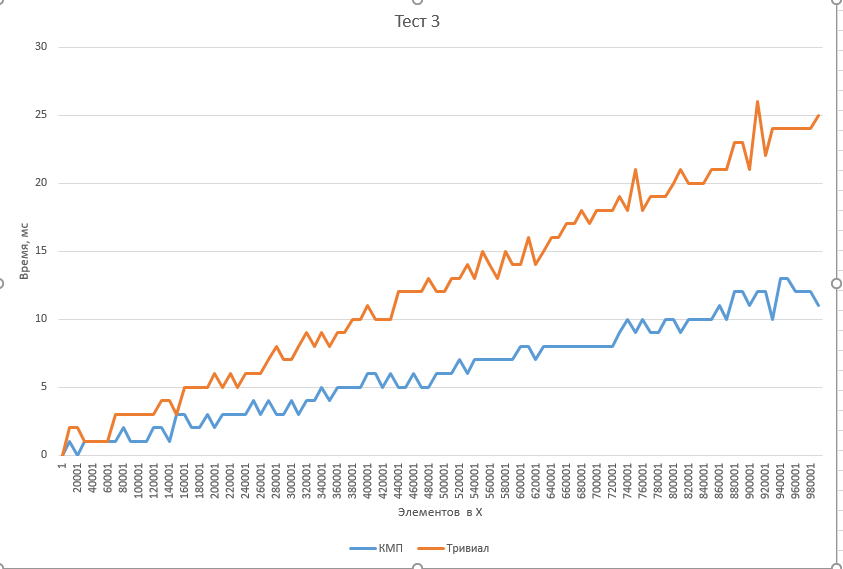
В данном эксперименте длина слова и длина строки изменялись. Длина строки была значительно больше длины слова. Строка состоит из повторяющихся элементов, следовательно, в алгоритме поиска Кнута-Морриса-Пратта может быть эффективно применена функция вычисления префикса слова. Это обуславливает более высокую эффективность работы этого алгоритма.

1. A={a,b}, Y=(a)m, слово X состоит из 10^6+1 букв алфавита A, встречающихся равновероятно, m = 1, … ,10^6+1 с шагом 10^4 (нарисовать графики функций T1(m) и T2(m)).



В данном примере длина строки не изменялась, менялась только длина строки. Т.к. слово Y состоит из последовательности буквы «а», а строка то КМП алгоритм работает значительно эффективнее, потому что префикс функция позволяет осуществлять сдвиг по строке, при котором пропускаем часть строки, которую наивный алгоритм поиска будет всегда проходить. Уменьшение времени работы наивного алгоритма поиска обусловлено уменьшением количества комбинаций различных подстрок с увеличением длины слова Y.

1. Y=aaaaa и X=(aaaaab)h, h = 1, … ,10^6+1 с шагом 10^4 (нарисовать графики функций T1(h) и T2(h)).



В данном эксперименте длина слова Y постоянна. Для нее мы можем посчитать максимальное значение префикс функции – 2. Значит, максимальное смещение по строке – 2 элемента. Это обуславливает не слишком высокий прирост по скорости работы в сравнении с наивным алгоритмом поиска.

**Вывод**

**Достоинства КМП алгоритма поиска:**

* Высокая эффективность при работе со словами, для которых можем получить высокие значения префикс функции
* В среднем случае работает эффективнее наивного алгоритма по иска

Однако, на небольших размерах слова Y и строки Х, а также для слов, для которых префикс функция имеет невысокие значения, время выполнения наивного алгоритма и КМП примерно одинаково. Значит можно применять как один алгоритм, так и другой.

**Приложение**

#include <iostream>

#include <fstream>

#include <string>

#include <list>

#include <cmath>

#include <iterator>

#include <vector>

#include <ctime>

using namespace std;

vector<char> A = { 'a', 'b' };

const string filename1 = "Text.txt";

int menu() {

system("cls");

int key = -1;

cout << "Type 0 to exit" << endl;

cout << "Type 1 to choose nunm of letters in alphabet" << endl;

cout << "Type 2 to type the needle to find" << endl;

cout << "Type 3 to generate the needle automatically" << endl;

cout << "Type 4 to choose text doc to search in, default is 'TextGen.txt' " << endl;

cout << "Type 5 to perform SFT\_TRIVIAL and SFT\_KMP in text, default text is 'TextGen.txt'" << endl;

cout << "Type 6 to show work time of algorithms" << endl;

cout << "Type 7 to show the result of alghs work" << endl;

cout << "Type 8 to type the haystack" << endl;

cout << "Type 9 to generate the haystack automatically" << endl;

cout << "Type 10 to proceed tests" << endl;

cin >> key;

return key;

}

//number of strings in the file

int strNum(string \_filename) {

int res = 0;

string s;

ifstream fin(\_filename, ios\_base::in);

if (fin.is\_open()) while (getline(fin, s)) res++;

else return 0;

fin.close();

return res;

}

//naive text search

void SFT\_Trivial(string needle, string haystack, list< pair<int, int> > &list, int m) {

for (int j = 0; j < haystack.length() - needle.length() + 1; j++)

if (haystack.substr(j, needle.length()) == needle) list.push\_back(make\_pair(m, j));

}

//function for printing the elements in a list

void showlist(list <pair <int, int> > \_list) {

list < pair <int, int> > ::iterator it;

for (it = \_list.begin(); it != \_list.end(); ++it)

cout << '\t' << "string : " << (\*it).first << " pos : " << (\*it).second << endl;

cout << endl;

}

//prefix function

vector<int> kmp(string needle) {

vector<int> v(needle.length());

v[0] = 0;

int j;

for (int i = 1; i < needle.length(); i++) {

j = v[i - 1];

while ((j > 0) && (needle[i] != needle[j])) j = v[j - 1];

if (needle[i] == needle[j]) j++;

v[i] = j;

}

return v;

}

//kmp search alg

void SFT\_KMP(string needle, string haystack, list < pair<int, int> > &list, int m) {

unsigned int stime = clock();

vector<int> pi(needle.length());

int k = 0;

pi = kmp(needle);

for (int i = 0; i < haystack.length(); ++i)

{

while ((k > 0) && (needle[k] != haystack[i]))

k = pi[k - 1];

if (needle[k] == haystack[i])

k++;

if (k == needle.length())

list.push\_back(make\_pair(m, i - needle.length() + 1));

}

}

//gets symbol from alphabet

char getSymb(int ind) {

return A[ind];

}

//tests

void tests() {

string haystack = "";

string needle = "";

ofstream kmpfout, trivfout;

double t1, t2, tmp;

string kmpflname = "TestResKMP1.txt";

string trivflname = "TestResTriv1.txt";

list<pair<int, int>> kmpRes;

list<pair<int, int>> trivRes;

trivfout.open(trivflname, ios\_base::out && ios\_base::trunc);

kmpfout.open(kmpflname, ios\_base::out && ios\_base::trunc);

//1st test

cout << "[[Performing test 1]]" << endl;

for (int k = 1; k <= 1001; k += 10) {

for (int i = 0; i <= k; i++) needle += "ab";

for (int i = 0; i <= 1000 \* k; i++) haystack += "ab";

//trivial output

tmp = clock();

SFT\_Trivial(needle, haystack, trivRes, 1);

t1 = clock() - tmp;

trivfout << k << " " << t1 << endl;

//kmp output

tmp = clock();

SFT\_KMP(needle, haystack, kmpRes, 1);

t2 = clock() - tmp;

kmpfout << k << " " << t2 << endl;

//refreshing storage

needle = "";

haystack = "";

trivRes.clear();

kmpRes.clear();

}

//closing fstreams

kmpfout.clear();

kmpfout.close();

trivfout.clear();

trivfout.close();

//2nd test

kmpflname = "TestResKMP2.txt";

trivflname = "TestResTriv2.txt";

trivfout.open(trivflname, ios\_base::out && ios\_base::trunc);

kmpfout.open(kmpflname, ios\_base::out && ios\_base::trunc);

//fill haystack

cout << "[[Performing test 2]]" << endl;

for (int i = 0; i <= pow(10, 6) + 1; i++) haystack += getSymb(rand() % 2);

for (int m = 1; m <= pow(10, 6) + 1; m += pow(10, 4)) {

for (int i = needle.length(); i <= m; i++) needle += "a";

//trivial output

tmp = clock();

SFT\_Trivial(needle, haystack, trivRes, 1);

t1 = clock() - tmp;

trivfout << m << " " << t1 << endl;

//kmp output

tmp = clock();

SFT\_KMP(needle, haystack, kmpRes, 1);

t2 = clock() - tmp;

kmpfout << m << " " << t2 << endl;

//refreshing storage

trivRes.clear();

kmpRes.clear();

}

//closing fstreams

kmpfout.clear();

kmpfout.close();

trivfout.clear();

trivfout.close();

//3d test

kmpflname = "TestResKMP3.txt";

trivflname = "TestResTriv3.txt";

trivfout.open(trivflname, ios\_base::out && ios\_base::trunc);

kmpfout.open(kmpflname, ios\_base::out && ios\_base::trunc);

needle = "aaaaa";

haystack = "";

cout << "[[Performing test 3]]" << endl;

for (int h = 1; h <= pow(10, 6); h += pow(10, 4)) {

for (int i = haystack.length(); i <= h; i++) haystack += "aaaaab";

//trivial output

tmp = clock();

SFT\_Trivial(needle, haystack, trivRes, 1);

t1 = clock() - tmp;

trivfout << h << " " << t1 << endl;

//kmp output

tmp = clock();

SFT\_KMP(needle, haystack, kmpRes, 1);

t2 = clock() - tmp;

kmpfout << h << " " << t2 << endl;

//refreshing storage

trivRes.clear();

kmpRes.clear();

}

//closing fstreams

kmpfout.clear();

kmpfout.close();

trivfout.clear();

trivfout.close();

}

int main(int argc, char \*\*argv) {

list< pair<int, int> > naive, kmp;

string haystack, needle; //haystack - source text, needle - string to find

string filename = "TextGen.txt";

int counter = 0;

ifstream fin;

ofstream fout;

int alph = 26;

int k = -1;

int len = 0;

int flg = 0;

double T1, tmp, T2;

srand(time(0));

while (k != 0) {

k = menu();

switch (k) {

case 1:

cout << "Type the number of letters in alphabet" << endl;

cin >> alph;

system("pause");

break;

case 2:

cout << "Type needle to find" << endl;

cin >> needle;

system("pause");

break;

case 3:

cout << "Type needle length" << endl;

cin >> len;

needle = "";

if (0 < alph < 27) for (int i = 0; i < len; i++) needle += (char)(rand() % alph + 97);

else for (int i = 0; i < len; i++) needle += (char)(rand() % alph);

cout << "Needle: " << needle << endl;

system("pause");

break;

case 4:

cout << "Type filename" << endl;

cin >> filename;

system("pause");

break;

case 5:

fin.open(filename, ios\_base::in);

tmp = clock();

//performing trivial

counter = 0;

naive.clear();

while (getline(fin, haystack)) {

SFT\_Trivial(needle, haystack, naive, counter);

counter++;

}

T1 = clock() - tmp;

//performing KMP

fin.clear(); //clears the bits with errors

fin.seekg(0); //returns pointer back to the beginning of the file

counter = 0;

kmp.clear();

tmp = clock();

while (getline(fin, haystack)) {

SFT\_KMP(needle, haystack, kmp, counter);

counter++;

}

T2 = clock() - tmp;

fin.clear();

fin.close();

break;

case 6:

cout << "SFT\_TRIVIAL worktime: " << T1 / 1000.0 << endl;

cout << "SFT\_KMP worktime: " << T2 / 1000.0 << endl;

system("pause");

break;

case 7:

cout << "SFT\_TRIVIAL result:" << endl;

showlist(naive);

cout << "SFT\_KMP result:" << endl;

showlist(kmp);

system("pause");

break;

case 8:

cout << "Type haystack to find" << endl;

cin >> haystack;

fout.open(filename, ios\_base::out & ios\_base::trunc);

fout << haystack;

fout.clear();

fout.close();

system("pause");

break;

case 9:

fout.open(filename, ios\_base::out & ios\_base::trunc);

cout << "Type 1 to gen haystack with rand letters, 2 to build haystack, with needle base" << endl;

cin >> flg;

if (flg != 2) {

cout << "Type length of haystack" << endl;

int hlength;

cin >> hlength;

haystack = "";

if (0 < alph < 27) for (int i = 0; i < hlength; i++) haystack += (char)(rand() % alph + 97);

else for (int i = 0; i < hlength; i++) haystack += (char)(rand() % alph);

}

else {

cout << "Type k to build hay" << endl;

int k;

cin >> k;

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

haystack += needle;

}

}

fout << haystack;

fout.clear();

fout.close();

system("pause");

break;

case 10:

tests();

break;

case 0: break;

default: break;

}

}

return 0;

}