МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритм Кутта-Морриса-Пратта

Студент гр. 7383 _____ Лосев М.Л.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2018

Постановка задачи.

Цель работы: реализация и исследование алгоритма Кутта-Морриса-Пратта, получение опыта и знания по его использованию.

Формулировка задачи: Реализовать алгоритм Кнута-Морриса-Пратта и с его помощью для заданных шаблона P ($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) найти все вхождения P в T. По памяти сложность не должна превышать O(|P|).

Входные данные:

Первая строка – **P**

Вторая строка – **T**

Выходные данные:

Индексы начал вхождений **P** в **T**, разделенных запятой, если **P** не входит в **T**, то вывести **-1**.

Описание алгоритма:

Сначала для шаблона вычисляется значение префикс функции и записывается в вектор. Для **i**-го символа строки s префикс функция **p** вычисляется по правилам: **j = p[i - 1]**, пока **s[j]** не равно **s[i]** и **j > 0**, то **j = p[j - 1]**, если **j** не равно 0, то **p[i] = j + 1**, иначе **p[i] = 0**. Далее для каждого символа текста вычисляется значение префикс функции, используя полученные значения для шаблона и значение, полученное для предыдущего символа. Если для **i**-го символа текста значение префикс функции равно **|P|**, то на **i**-ом символе заканчивается подстрока в тексте, соответствующая шаблону, и в ответ записывается индекс **i - |P| + 1**.

Реализация

Был использован следующие функции:

`vector<int> prefix_function(const string& s)` – вычисляет префикс-функцию строки s

`void print_answ(const set<int> &s)` – выводит ответ

`set<int> KMP(string& T, string& P)` – реализует алгоритм в один поток

`void one_threaded_KMP(mutex &cout_mutex, mutex &solution_mutex, set<int>& solution, string& T, string& P, vector<int>& prefixes, int start, int end)` – один из потоков КМП

`set<int> multithreaded_KMP(string& T, string& P, int k)` – реализует алгоритм КМП в k потоков

`int cyclic_shift(string& A, string& B)` – проверят, является ли строка A циклическим сдвигом строки B и возвращает индекс, с которого одна строка начинается в другой.

Исследование

Пусть **m** – длина шаблона, **n** – длина текста. Таблица префикс-функций шаблона вычисляется (амортизационно) за **O(m)** сравнений перед началом поиска и занимает столько же памяти. А поскольку текст будет пройден ровно один раз, суммарное время работы алгоритма будет равно **O(m + n)**.

Тестирование.

Тестирование проводилось в ОС Ubuntu 18.04 компилятором GCC. Кроме того, программа прошла тесты на Stepic. Результаты тестирования показали, что программа работает корректно.

Вывод.

При выполнении работы был реализован и изучен Кутта-Морриса-Пратта. Его вычислительная сложность в худшем случае есть **O(m + n),** а затраты памяти **O(m).**

ПРИЛОЖЕНИЕ А

Код программы

```
#include <iostream>
#include <string>
#include <vector>
#include <set>
#include <thread>    // parallel
#include <mutex>
#include <functional>      // std::ref
#include <cstdio>
#include <cctype>

using namespace std;

vector<int> prefix_function(const string& s) {
      int len = s.length();
      vector<int> p(len);

      p[0] = 0;

      int k = 0;
      for (int i = 1; i < len; ++i) {
            while ((k > 0) && (s[k] != s[i]))
                  k = p[k - 1];
            if (s[k] == s[i])
                  ++k;
            p[i] = k;
      }
      return p;
}

void print(const vector<int> &v) {
      if (v.size() == 0) {
            cout << "-1" << endl;
            return;
      }
      for (int i = 0; i < v.size() - 1; i++)
            cout << v[i] << ",";
      cout << v[v.size() - 1] << endl;
}

void print_answ(const set<int> &s) {
      if (s.size() == 0) {
            cout << "-1" << endl;
            return;
      }
      auto end = s.end();
      end--;
      auto it = s.begin();
```

```
	for (auto it = s.begin(); it != end; it++)
		cout << *it << ",";
	cout << *end << endl;
}



set<int> KMP(string& T, string& P) {

	set<int> solution;
	if (P.length() >= T.length()) {
		solution.insert(-1);
		return solution;
	}
	vector<int> prefixes = prefix_function(P);

	int j = 0;
	for (int i = 0; i < T.length(); i++) {
		while (j == P.length() || (j > 0 && P[j] != T[i])) {
			j = prefixes[j - 1];
			if (P.length() - j > T.length() - i)
				break;
		}

		if (T[i] == P[j])
			j++;

		if (j == P.length())
			solution.insert(i - j + 1);
	}

	return solution;
}

void one_threaded_KMP(mutex &cout_mutex, mutex &solution_mutex, set<int>&
solution, string& T, string& P, vector<int>& prefixes, int start, int
end)
{
	cout_mutex.lock();
	cout << "A thread " << this_thread::get_id() << " started ";
	cout << "with start = " << start << "; end = " << end << ";" <<
endl;
	cout_mutex.unlock();

	set<int> partly;

	int j = 0;
	for (int i = start; i < end + P.length() - 1; i++) {
		while (j == P.length() || (j > 0 && P[j] != T[i])) {
			j = prefixes[j - 1];
			if (P.length() - j > end + P.length() - 1 - i)
```

```
                    break;
          }

          if (T[i] == P[j])
               j++;

          if (j == P.length()) {
                    solution_mutex.lock();
                    solution.insert(i - j + 1);
                    solution_mutex.unlock();
                    partly.insert(i - j + 1);
          }
     }

     cout_mutex.lock();
     cout << "The threads " << this_thread::get_id() << " result: ";
     print_answ(partly);
     cout_mutex.unlock();

     cout_mutex.lock();
     cout << "A thread " << this_thread::get_id() << " is now ready to
be joined" << endl;
     cout_mutex.unlock();
}

set<int> multithreaded_KMP(string& T, string& P, int k) {
     set<int> solution;
     if (P.length() >= T.length()) {
          solution.insert(-1);
          return solution;
     }

     vector<int> prefixes = prefix_function(P); // prefix-function


     auto part_len = (T.length() / k);
     mutex solution_mutex;
     mutex cout_mutex;
     thread thr[k];

     int start = 0;
     int end = part_len;
     for (int i = 0; i < k - 1; i++) {     // create treads
          thr[i] = thread(one_threaded_KMP, ref(cout_mutex),
ref(solution_mutex), ref(solution), ref(T), ref(P), ref(prefixes), start,
end);
          start = end;
          end += part_len;
     }
```

```
	thr[k - 1] = thread(one_threaded_KMP, ref(cout_mutex),
ref(solution_mutex), ref(solution), ref(T), ref(P), ref(prefixes), start,
T.length());

	for (int i = 0; i < k; i++) {		// join threads
		auto current_thread_id = thr[i].get_id();
		thr[i].join();
		cout_mutex.lock();
		cout << "The thread " << current_thread_id << " is now
joined" << endl;
		cout_mutex.unlock();
	}

	return solution;
}

int cyclic_shift(string& A, string& B) {
	if (A == B) return 0;
	if (A.length() != B.length()) return -1;
	//vector<int> p = prefix_function(A + "$" + B);
	int A_prefix = prefix_function(A + "$" + B)[A.length() +
B.length()];		// prefix function of the last char
	//p = prefix_function(B + "$" + A);
	int A_postfix = prefix_function(B + "$" + A)[A.length() +
B.length()];

	if (A_postfix + A_prefix != A.size() || A.size() != B.size())
		return -1;

	return A_prefix;
}

void test_KMP(string& p, string& t, int k) {
	auto answ = multithreaded_KMP(t, p, k);

	cout << "1-thread KMP: ";
	print_answ(KMP(t, p));
	cout << k << "-thread KMP: ";
	print_answ(answ);
}

void test_cyclic_shift(string& a, string& b) {
	int index;
	index = cyclic_shift(a, b);
	cout << index << endl;
	if (index != -1) {
		cout << a << " = " << a.substr(0, index) << " + " <<
a.substr(index, a.length()) << endl;
		cout << b << " = " << a.substr(index, a.length()) << " + " <<
a.substr(0, index) << endl;
	}
```

```
}

int main()
{
	string a, b;
	cin >> a >> b;

	//test_KMP(a, b, 3);  // the last argument is number of threads
	test_cyclic_shift(a, b);


	return 0;
}
```