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 Кнута-Морриса-Пратта

Студент гр. 8303 _____ Данилов А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Научиться использовать алгоритм Кнута-Морриса-Пратта поиска индексов вхождений подстроки в строку и алгоритм определения циклического сдвига путём разработки программы.

Индивидуализация.

Вариант 1

Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки поиска).

Задание.

- Задание 1.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) найдите все вхождения P в T.

Вход:

Первая строка - P

Вторая строка - T

Выход:

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

Sample Input:

ab

abab

Sample Output:

0,2

- Задание 2

Заданы две строки A ($|A| \leqslant 5000000$) и B ($|B| \leqslant 5000000$).

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

Вход:

Первая строка – A

Вторая строка – B

Выход:

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

Sample Input:

defabc

abcdef

Sample Output:

3

Описание алгоритмов.

КМП

Рассмотрим сравнение строк на позиции i, где образец S[0, m-1] сопоставляется с частью текста T[i, i+m-1]. Предположим, что первое несовпадение произошло между T[i+j] и S[j] , где 1 < j < m. Тогда T[i, i+j-1]=S[0, j-1]=P и a = T[i+j] ≠ S[j] = b.

При сдвиге вполне можно ожидать, что префикс (начальные символы) образца S сойдется с каким-нибудь суффиксом (конечные символы) текста P. Длина наиболее длинного префикса, являющегося одновременно суффиксом, есть значение префикс-функции от строки S для индекса j.

Это приводит нас к следующему алгоритму: пусть pi[j] — значение префикс-функции от строки S[0, m-1] для индекса j. Тогда после сдвига мы можем возобновить сравнения с места T[i+j] и S[pi[j]] без потери возможного местонахождения образца.

Сложности алгоритма.

Сложность алгоритма по операциям:

O (m+n), где n – количество символов в исходной строке, а m – количество символов в искомой строке.

Сложность алгоритма по памяти:

O (m+n), где m – количество символов в образце (искомой строке), n — количество символов в исходной строке.

Циклический сдвиг

В данном алгоритме можно обойтись без удваивания строки. В самом начале происходит проверка на соответствие длин строк. Если соответствия не было обнаружено, то выводится -1. Создаются два счётчика для первой и второй строки. Далее сравниваются символы первой и второй строки, если символы совпадают переход к следующим, счётчики увеличиваются, если совпадения не обнаружено, счётчик для второй строки уменьшается. В том случае, если счётчик второй строки равен её длине, то сдвиг найден, а если счётчик первой строки равен её длине, то происходит его обнуление, таким образом строка зацикливается.

Сложность алгоритма

Сложность по операциям:

O (n+n), n — длина строки

Сложность по памяти:

O(n+n), где n — размер первой и второй строки

Префикс функция

Префикс-функция от строки и позиции в ней — длина наибольшего собственного префикса подстроки, который одновременно является суффиксом этой подстроки. То есть, в начале подстроки длины нужно найти такой префикс максимальной длины , который был бы суффиксом данной подстроки .

Разделение исходного текста

Получаем исходный текст (строку) и число частей, на которое нужно разделить строку. Анализируя длину строки и число частей получаем длину каждой части строки. Далее сохраняем части строки в массив. Отдельно проверяем не раздели ли мы строку на месте образца. Для этого получаем

подстроку с разрезом строки по середине и обрабатываем данную подстроку отдельно, ее длина 2*n-2, где n - длина образца.

Пример:

abacaba | cabaaba

обрабатываемая подстрока baca.

Описание функций

1. vector<int> getPrefix (string s)

   string s — строка

   Возвращает массив значений префикс - функции для указанной строки.

2. void split(string t, string p, int k, vector<string> &s, vector<int> &currentAnswer, set<int> &allAnswers, vector<int> &pi)

   string t — исходный текст

   string p — строка, вхождения которой ищем

   int k - число частей, на которое нужно разделить текст

   vectror <string> s - все части текста

   vector <int> currentAnswer - вектор ответов для данной части текста

   set <int> allAnswers - отсортированный массив всех ответов

   vector<int> pi — результаты префикс — функции

   Функция разделяет исходную строку t на k частей и проверяет места разрыва на случай существовании в нем искомой строки.

3. vector<int> KMP(string t, string p, vector<int> &pi)

   string t — строка, в которой ищем вхождения

   string p — строка, вхождения которой ищем

   vector<int> pi — результаты префикс — функции

   Функция, которая ищет индексы вхождения строки p в строку t

Тестирование.

KMP

№	Входные данные	Выходные данные
1	abbab abbbaaababababbabaabbabba	10,16

2	queue ququqeuqqquqeq	-1
3	laaaaaaaaaaaaaaaaaaaaaaaaaaaaa a	-1
4	a laaaaaaaaaaaaaaaaaaaaaaaa	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
5	slice slice slice slice slice slice like a samurai cut it up real nice, i'm a samurai go karate on the mic, i'm a samurai samurai, i-i-i'm a samurai	0,6,12,18,24
6	hi he said hi, but meant not hi	8,26

```
Введите образец (искомую подстроку)
hi
Введите текст
he said hi, but meant not hi
Введите число от 1 до 8
3
PREFIX FUNCTION
h i
0 0
i ≠ h index: 1 0; pi[1] ⇒ 0
TOTAL FUNCTION
h i
0 0
-----------------------------------------
Строка будет разделена на 3 частей
Итоговое разделение текста
h e   s a i d   h i | ,   b u t   m e a n | t   n o t   h i |
---------------------------------------
Подстрака с центром на месте разреза - i,
Индексы в исходном тексте: 9 10
Индексы:                   0 1
Символы подстроки:         i ,
Префикс-функция для образца  hi
0 0
Несовпадение: i≠h index: 0 0
Несовпадение: ,≠h index: 1 0
---------------------------------------
Подстрака с центром на месте разреза - nt
Индексы в исходном тексте: 19 20
Индексы:                   0  1
Символы подстроки:         n  t
Префикс-функция для образца  hi
0 0
Несовпадение: n≠h index: 0 0
Несовпадение: t≠h index: 1 0
------------------------
Часть исходного текста     he said hi
Индексы в исходном тексте: 0 1 2 3 4 5 6 7 8 9
Индексы:                   0 1 2 3 4 5 6 7 8 9
Символы подстроки:         h e   s a i d   h i
Префикс-функция для образца  hi
0 0
```

```
Совпадение:   h=h index: 0 0
Несовпадение: e≠i index: 1 1
Несовпадение: e≠h index: 1 0
Несовпадение:  ≠h index: 2 0
Несовпадение: s≠h index: 3 0
Несовпадение: a≠h index: 4 0
Несовпадение: i≠h index: 5 0
Несовпадение: d≠h index: 6 0
Несовпадение:  ≠h index: 7 0
Совпадение:   h=h index: 8 0
Совпадение:   i=i index: 9 1
Найдена подстрака
------------------------
------------------------
Часть исходного текста     , but mean
Индексы в исходном тексте: 10 11 12 13 14 15 16 17 18 19
Индексы:                   0  1  2  3  4  5  6  7  8  9
Символы подстроки:         ,     b  u  t     m  e  a  n
Префикс-функция для образца  hi
0 0
Несовпадение: ,≠h index: 0 0
Несовпадение:  ≠h index: 1 0
Несовпадение: b≠h index: 2 0
Несовпадение: u≠h index: 3 0
Несовпадение: t≠h index: 4 0
Несовпадение:  ≠h index: 5 0
Несовпадение: m≠h index: 6 0
Несовпадение: e≠h index: 7 0
Несовпадение: a≠h index: 8 0
```

```
Несовпадение: n≠h index: 9 0
-----------------------
Часть исходного текста     t not hi
Индексы в исходном тексте: 20 21 22 23 24 25 26 27
Индексы:                   0  1  2  3  4  5  6  7
Символы подстроки:         t     n  o  t     h  i
Префикс-функция для образца  hi
0 0
Несовпадение: t≠h index: 0 0
Несовпадение:  ≠h index: 1 0
Несовпадение: n≠h index: 2 0
```

```
Введите строки 1 и 2
magicthisis
thisismagic
PREFIX FUNCTION
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
a ≠ m index: 1 0; pi[1] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
g ≠ m index: 2 0; pi[2] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
i ≠ m index: 3 0; pi[3] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
c ≠ m index: 4 0; pi[4] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
t ≠ m index: 5 0; pi[5] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
h ≠ m index: 6 0; pi[6] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
i ≠ m index: 7 0; pi[7] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
s ≠ m index: 8 0; pi[8] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
```

Cycle Shift

	Выходные данные
	3
	0
	-1
	-1
	6

```
i ≠ m index: 9 0; pi[9] ⇒ 0
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
```

```
Cycle shift begin
t h i s i s m a g i c
m a g i c t h i s i s
0 0 0 0 0 0 0 0 0 0 0
it_a - указатель на текущий символ в строке 1
it_b - указатель на текущий символ в строке 2
Несовпадение: t≠m index: 0 0
Увеличиваем it_a
Несовпадение: h≠m index: 1 0
Увеличиваем it_a
Несовпадение: i≠m index: 2 0
Увеличиваем it_a
Несовпадение: s≠m index: 3 0
Увеличиваем it_a
Несовпадение: i≠m index: 4 0
Увеличиваем it_a
Несовпадение: s≠m index: 5 0
Увеличиваем it_a
Совпадение:   m=m index: 6 0
Совпадение:   a=a index: 7 1
Совпадение:   g=g index: 8 2
Совпадение:   i=i index: 9 3
Совпадение:   c=c index: 10 4
Совпадение:   t=t index: 0 5
Совпадение:   h=h index: 1 6
Совпадение:   i=i index: 2 7
Совпадение:   s=s index: 3 8
Совпадение:   i=i index: 4 9
Совпадение:   s=s index: 5 10
Цикл: 6
```

Вывод

Были получены умения по использованию алгоритм Кнута-Морриса-Пратта поиска индексов вхождений подстроки в строку и алгоритма определения

циклического сдвига. Написана программа, реализующая алгоритм Кнута-Морриса-Пратта и выводящая индексы вхождений подстроки в строку, также написана программа, выводящая индекс начала циклического сдвига в исходной строке. Помимо основного алгоритма, так же был реализован механизм распараллеливания строки, для запуска алгоритма сразу в нескольких потоках.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

```
#include <iostream>
```

```
#include <vector>
#include <string>
#include <iterator>
#include <thread>
#include <algorithm>
#include <set>

//#define PRINT

using namespace std;
vector<int> KMP(string t, string p, vector<int> &pi);
vector<int> getPrefix (string s) {
	int n = (int) s.length();
	vector<int> pi(n, 0);
	int i = 1, j = 0;
	#ifdef PRINT
	cout << "PREFIX FUNCTION" << endl;
//	cout << s << endl;
	#endif
	while(i < n){
	#ifdef PRINT
		for (auto x:s)
			cout << x << ' ';
		cout << endl;
		for(int h = 0; h < n; h++)
			cout << pi[h] << ' ';
		cout << endl;
	#endif
		if(s[i] == s[j]){
			#ifdef PRINT
			cout << s[i] << " == " << s[j] << " index: " << i << ' ' << j
<< "; pi[" << i << "] => " << j+1 << endl;
			#endif
			pi[i]= j+1;
			i++;
			j++;
		}
		else {
			if(j == 0){
				#ifdef PRINT
```

```
                        cout << s[i] << " != " << s[j] << " index: " << i << ' '
<< j << "; pi[" << i << "] => " << 0 << endl;
                        #endif
                        pi[i] = 0;
                        i++;
                  }
                  else {
                        #ifdef PRINT
                        cout << s[i] << " != " << s[j] << " index: " << i << ' '
<< j << "; j => " << pi[j-1] << endl;
                        #endif
                        j = pi[j-1];
                  }
            }
      }
#ifdef PRINT
      cout << "TOTAL FUNCTION\n";
      for (auto x:s)
            cout << x << ' ';
      cout << endl;
      for(int h = 0; h < n; h++)
            cout << pi[h] << ' ';
      cout << endl;
#endif
      return pi;
}

void split(string t, string p, int k, vector<string> &s, vector<int>
&currentAnswer, set<int> &allAnswers, vector<int> &pi){
//    t - text
//    p - pattern
//    k = число частй
//    s - все части texta
//  currentAnswer  - вектор ответов для данной части
//  allAnswers -  сет всех ответов
//  pi - функция

      int len_parts, flag = 0;
      int k1;
      //-------------------------------------------------
      //определяем длинну каждой части
```

```
	if(t.length() % k){
		len_parts = int(t.length()/k)+1; //длинна части строки
		flag = 1;
		k1 = k-1;
	}
	else {
		k1 = k;
		len_parts = t.length()/k;
	}
	//-------------------------------------------------
	int begin = 0;
	string part = "";
	//цикл для получения массива подстрок из текста
	while(k1 > 0){
		part = "";
		part.append(t, begin, len_parts);
		s.push_back(part);
		begin += len_parts;
		k1--;
	}
	if(flag){
		part = "";
		part.append(t, begin, (t.length()-(len_parts*(k-1))));
		s.push_back(part);
	}
#ifdef PRINT
	cout << "\nИтоговое разделение текста\n";
	for(auto x: s){
		for (auto y : x)
			cout << y << ' ';
		cout << "| ";
	}
	cout << endl;
#endif
	//цикл для получения и проверки подстрок на стыках на каждом стыке
проверяется 2 стрки
	k1 = 1;
	while(k1 < k){
		part = "";
		part.append(t, (len_parts*k1)-p.length()+1, 2*p.length()-2);
		int top = (len_parts*k1)-p.length()+1;
```

```
            #ifdef PRINT
            cout << "---------------------------------------" << endl;
            cout << "Подстрака с центром на месте разреза - " << part << endl;
            cout << "Индексы в исходном тексте: ";
            for(int i = 0; i < part.size(); i++){
                  cout << i+top << ' ';
            }
            cout << endl;
            cout << "Индексы:                    ";
            for(int i = 0; i < part.size(); i++){
                  if(i+top > 9)
                        cout << i << "  ";
                  else {
                        cout << i << " ";
                  }
            }
            cout << endl;
            cout << "Символы подстроки:         ";
            for(int i = 0; i < part.size(); i++){
                  if(i+top > 9)
                        cout << part[i] << "  ";
                  else {
                        cout << part[i] << " ";
                  }
            }
            cout << endl;
            #endif
            currentAnswer = KMP(part, p, pi);
            if(currentAnswer.size() > 0){
                  for(int i = 0; i < currentAnswer.size(); i++){
                        currentAnswer[i] += top; //определяем номер символа
начала подстроки в исходном тексте
                        allAnswers.insert(currentAnswer[i]);
                  }
            }
            k1++;
      }
}

vector<int> KMP(string t, string p, vector<int> &pi){
      vector<int> ans;
```

```
#ifdef PRINT
	cout << "Префикс-функция для образца  " << p << endl;
	for(int i = 0; i < pi.size(); i++)
		cout << pi[i] << ' ';
	cout << endl;
#endif
	int n = t.length();
	int m = p.length();
	int k = 0, l = 0;
	while(k < n){
		if(t[k] == p[l]){
#ifdef PRINT
			cout << "Совпадение:   " << t[k] << "==" << p[l] <<" index:
"<< k << " " << l << endl;
#endif
			k++;
			l++;
			if(l == m){
				ans.push_back(k-l);
#ifdef PRINT
				cout << "Найдена подстрака\n----------------------" <<
endl;
#endif
			}
		}
		else {
			if(l == 0){
#ifdef PRINT
				cout << "Несовпадение: " << t[k] << "!=" << p[l] <<"
index: "<< k << " " << l << endl;
#endif
				k++;
			}
			else {
#ifdef PRINT
				cout << "Несовпадение: " << t[k] << "!=" << p[l] <<"
index: "<< k << " " << l << endl;
#endif
				l = pi[l-1];
			}
		}
```

```
	}
	return ans;
}

void KMP(){
	string p,t;
	#ifdef PRINT
	cout << "Введите образец (искомую подстроку)" << endl;
	#endif
	getline(cin, p);
	#ifdef PRINT
	cout << "Введите текст" << endl;
	#endif
	getline(cin, t);
	if(t.length() < p.length()){
		cout << "Образец не может быть больше текста!" << endl;
		cout << -1 << endl;
		return;
	}
	int max_threads = sizeof(thread); // определяем максимально возможное
число потоков
	//-------------------------------------------------
	// определяем на сколько частей можно раделить строку
	double alpha = (double)t.length()/(double)p.length();
	max_threads = min(max_threads, int(alpha)-1);
	if(max_threads <= 0)
		max_threads = 1;
	int k ;//= max_threads;
	if(max_threads == 1){
		cout << "Длина исходного текста недостаточна для деления строки" <<
endl;
		k = 1;
	}
	else{
		cout << "Введите число от 1 до "<< max_threads << endl;
		cin >> k;
		while(k < 1 or k > max_threads){
			cout << "Введите число от 1 до "<< max_threads << endl;
			cin >> k;

		}
```

```
	}
	//-------------------------------------------------
	vector<int> pi = getPrefix(p);
	vector<int> ans, ans_current;
	vector<string> str;
	set<int> ans_all;

	#ifdef PRINT
	if(k > 1){
		cout << "-----------------------------------------" << endl;
		cout << "Строка будет разделена на " << k << " частей" << endl;
	}
	#endif
	if(k == 1){
		ans = KMP(t, p, pi);
		for(int j = 0; j < ans.size(); j++)
			ans_all.insert(ans[j]);
	}
	else {
		//-------------------------------------------------
		// определяем длинну каждой части
		int len_parts;
		if(t.length() % k){
			len_parts = int(t.length()/k)+1; //длинна части строки
		}
		else {
			len_parts = t.length()/k;
		}
		#ifdef PRINT
		cout << "Максимальная длинна части исходного текста - " << len_parts
<< endl;
		cout << "-----------------------------------------" << endl;
		cout << endl;
		#endif
		split(t, p, k, str, ans_current, ans_all, pi);
		//-------------------------------------------------
		//заполняем исходный массив ответов
		for(int i = 0; i< str.size(); i++){
			#ifdef PRINT
			cout << "----------------------\nЧасть исходного текста
" << str[i] << endl;
```

```
                cout << "Индексы в исходном тексте: ";
                for(int j = 0; j < str[i].size(); j++){
                        cout << j+len_parts*i << ' ';
                }
                cout << endl;
                cout << "Индексы:                    ";
                for(int j = 0; j < str[i].size(); j++){
                        if(j+len_parts*i > 9)
                                cout << j << "  ";
                        else {
                                cout << j << " ";
                        }
                }
                cout << endl;
                cout << "Символы подстроки:         ";
                for(int j = 0; j < str[i].size(); j++){
                        if(j+len_parts*i > 9)
                                cout << str[i][j] << "  ";
                        else {
                                cout << str[i][j] << " ";
                        }
                }
                cout << endl;
                #endif
                ans_current = KMP(str[i], p, pi);
                if(ans_current.size() > 0){
                        for(int j = 0; j < ans_current.size(); j++)
                                ans_current[j] += (len_parts*i); // определяем
номер символа начала образца в исходном тексте
                        for(int j = 0; j < ans_current.size(); j++)
                                ans_all.insert(ans_current[j]);
                        //ans.insert(ans.end(), ans_current.begin(),
ans_current.end());
                }
        }
    }
    // Ввывод ответа
    if(!ans_all.empty()){
        int end = *ans_all.rbegin();
        ans_all.erase(end);
```

```
            copy(ans_all.begin(), ans_all.end(), ostream_iterator<int>(cout,
","));
            cout << end << endl;
      }
      else {
            cout << -1 << endl;
      }
}

void cycleShift(){
      string a,b;
      #ifdef PRINT
      cout << "Введите строки 1 и 2" << endl;
      #endif
      cin >> b >> a;
      vector<int> pi = getPrefix(b);
      #ifdef PRINT
      cout << "\nCycle shift begin\n";
      for (auto x:a)
            cout << x << ' ';
      cout << endl;
      for (auto x:b)
            cout << x << ' ';
      cout << endl;
      for(auto x: pi)
            cout << x << ' ';
      cout << endl;

      #endif

      if(b.length() != a.length())
      {
            cout << "Разная длинна строк!" << endl;
            cout << "-1" << endl;
            return;
      }
      if(a == b){
            cout << "Строки совпадают" << endl;
            cout << 0 << endl;
            return;
      }
```

```
	#ifdef PRINT
	cout << "it_a - указатель на текущий символ в строке 1" << endl;
	cout << "it_b - указатель на текущий символ в строке 2" << endl;
	#endif
	int it_a = 0, it_b = 0;
	int cikle = 0;
	int al = a.length();
	while(true){
		if(a[it_a] == b[it_b]){
			#ifdef PRINT
			cout << "Совпадение:   " << a[it_a] << "==" << b[it_b] <<"
index: "<< it_a << " " << it_b << endl;
			#endif
			it_a++;
			it_b++;
		}
		if(it_a == al){
			it_a = 0;
			cikle++;
		}
		if(it_b == al){
			#ifdef PRINT
			cout << "Цикл: ";
			#endif
			cout << it_a << endl;
			return;
		}
		else{
			if(a[it_a] != b[it_b]){
				#ifdef PRINT
				cout << "Несовпадение: " << a[it_a] << "!=" << b[it_b]
<<" index: "<< it_a << " " << it_b << endl;
				#endif
				if(it_b == 0){
					it_a++;
				#ifdef PRINT
					cout << "Увеличиваем it_a" << endl;
				#endif
				}
				else {
					it_b = pi[it_b-1];
```

```
                        #ifdef PRINT
                              cout << "Уменьшаем it_b" << endl;
                        #endif
                        }
                  }
            }
            if(cikle > 1){
                  cout << -1 << endl;
                  return;
            }

      }
}

int main()
{
      KMP();
//    cycleShift();
      return 0;
}
```