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

Студентгр. 8304 _____ Нам Ё Себ

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Изучение алгоритма Кнута-Морриса-Пратта поиска образца в строке.

Задание

• Задание 1

Реализуйте алгоритм КМП и с его помощью для заданных образца P (|P | ⩽ 15000) и текста T (|T | ⩽ 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| ⩽ 5000000) и B (|B| ⩽ 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 является 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

Постановка задачи.

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).

Описание алгоритма.

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 $a = T[i+j] \neq S[j] = b$.

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.

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.

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.

Анализ алгоритма.

Сложность алгоритма КМП :

O(n + m), n – длина подстроки, m– длина строки.

Сложность алгоритма поиска циклического сдвига:

O (n+n) = O(n).

Описание функций.

1) vector<int> prefix_function (string s)

Возвращает значение префикс функции для строки.

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

Функция нахождения образца в тексте алгоритмом Кнута-Морриса-Пратта.

string t — исходный текст

string p - образец

vector<int> &pi — ссылка на вектор значений префикс-функции.

3) void split(string t, string p, int k, vector<string> &str, vector<int> &ans_current, vector<int> &ans, vector<int> &pi)

Функция разделения исходного текста на части.

string t — исходный текст

string p — образец

int k — число частей исходного текста

vector<int> &str — ссылка на вектор хранящий части строк исходного текста

vector<int> &ans_current — ссылка на вектор ответов для текущей части исходного текста

vector<int> &ans — ссылка на вектор ответов для всего текста

vector<int> &pi — ссылка на вектор значений префикс-функции.

Тестирование.

№	Входные данные	Выходные данные
1	abacabacabaaba aba	0,4,8,11
2	Dance for me, dance for me, dance for me, oh, oh, oh I've never seen anybody do the things you do before They say move for me, move for me, move for me, ay, ay, ay And when you're done I'll make you do it all again for	6,20,34,100,119,132,145
3	qwertyuilkjhgfdszxcvbn ,mn	-1
4	abababsssababasssababa ababa	0,8,16
5	fffffffffff f	0,1,2,3,4,5,6,7,8,9
6	aaaaaaaaaaaaaaaaaaaa	0,1,2,3,4,5,6,7,8,9,10,11,

Таблица 1 – Пример вывода для простых входных данных

```
Номер задачи - 1
Название - KMP или kmp
        Определить, является ли стока 1 циклическим сдвигом строки 2:
Номер задачи - 2
Название - Rotation или rotation

Введите номер задачи или название алгоритма
1
Введите текст
abasdqwer
Введите образец (искомую подстроку)
as
Введите число от 1 до 3
2
-----------------------------------
as
0 0
s != a index: 1 0; pi[1] => 0
0 0
-------------------------------------------
-------------------------------------------
Строка будет разделена на 2 частей
Максимальная длинна части исходного текста - 5
-------------------------------------------

---------------------------------------
Подстрака с центром на месте разреза - dq
Индексы в исходном тексте: 4 5
Индексы:                   0 1
Символы подстроки:         d q
Префикс-функция для образца  as
0 0
Несовпадение: d!=a index: 0 0
Несовпадение: q!=a index: 1 0
-----------------------
Часть исходного текста     abasd
Индексы в исходном тексте: 0 1 2 3 4
Индексы:                   0 1 2 3 4
Символы подстроки:         a b a s d
Префикс-функция для образца  as
0 0
Совпадение:   a==a index: 0 0
Несовпадение: b!=s index: 1 1
Несовпадение: b!=a index: 1 0
Совпадение:   a==a index: 2 0
Совпадение:   s==s index: 3 1
Найдена подстрока
-----------------------
Несовпадение: d!=  index: 4 2
Несовпадение: d!=a index: 4 0
-----------------------
Часть исходного текста     qwer
Индексы в исходном тексте: 5 6 7 8
Индексы:                   0 1 2 3
Символы подстроки:         q w e r
Префикс-функция для образца  as
0 0
Несовпадение: q!=a index: 0 0
Несовпадение: w!=a index: 1 0
Несовпадение: e!=a index: 2 0
Несовпадение: r!=a index: 3 0
2
```

Рисунок 1 – Тестирование алгоритма КМП.

№	Входные данные	Выходные данные
1	defabc abcdef	3
2	defabc abcder	-1
3	foobar foobar	0
4	abaa aaba	3

Таблица 2 – Пример вывода для простых входных данных

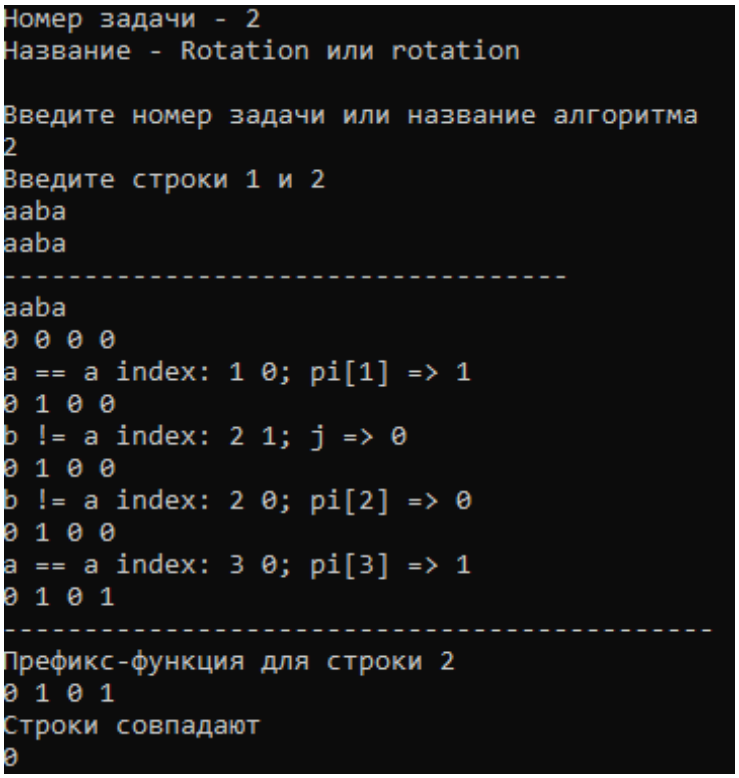

Рисунок 2 – Тестирование циклического сдвига.

Выводы.

В ходе выполнения лабораторной работы был изучен и реализован алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке, результатом которого является набор индексов вхождения подстроки. Для работы алгоритма также реализована префикс-функция. Помимо основного алгоритма, так же реализован механизм распараллеливания строки, для запуска алгоритма сразу в нескольких местах.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

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


std::vector<int> prefix_function(std::string s) {
int n = (int)s.length();
std::vector<int> pi(n, 0);
int i = 1, j = 0;

std::cout << "-----------------------------------" << std::endl;
std::cout << s << std::endl;

while (i < n)
{
	for (int h = 0; h < n; h++)
		std::cout << pi[h] << ' ';
	std::cout << std::endl;

	if (s[i] == s[j])
	{
		std::cout << s[i] << " == " << s[j] << " index: " << i << ' '
<< j << "; pi[" << i << "] => " << j + 1 << std::endl;
		pi[i] = j + 1;
		i++;
		j++;
	}
	else
	{
		if (j == 0)
		{
			std::cout << s[i] << " != " << s[j] << " index: " << i <<
' ' << j << "; pi[" << i << "] => " << 0 << std::endl;
			pi[i] = 0;
			i++;
		}
		else
		{
			std::cout << s[i] << " != " << s[j] << " index: " << i <<
' ' << j << "; j => " << pi[j - 1] << std::endl;
			j = pi[j - 1];
		}
	}
}
for (int h = 0; h < n; h++)
	std::cout << pi[h] << ' ';
std::cout << std::endl;
std::cout << "-----------------------------------------" << std::endl;
```

```
	return pi;
	}

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

	std::cout << "Префикс-функция для образца  " << p << std::endl;
	for (int i = 0; i < pi.size(); i++)
		std::cout << pi[i] << ' ';
	std::cout << std::endl;

	int n = t.length();
	int m = p.length();
	int k = 0, l = 0;

	while (k < n)
	{
		if (t[k] == p[l])
		{
			std::cout << "Совпадение:    " << t[k] << "==" << p[l] << " in-
dex: " << k << " " << l << std::endl;
			k++;
			l++;
			if (l == m) { ans.push_back(k - l); std::cout << "Найдена
подстрока\n-----------------------" << std::endl; }
		}
		else
		{
			if (l == 0)
			{
				std::cout << "Несовпадение: " << t[k] << "!=" << p[l] <<
" index: " << k << " " << l << std::endl;
				k++;
			}
			else
			{
				std::cout << "Несовпадение: " << t[k] << "!=" << p[l] <<
" index: " << k << " " << l << std::endl;
				l = pi[l - 1];
			}
		}
	}

	return ans;
	}

	void split(std::string t, std::string p, int k, std::vector<std::string>&
str, std::vector<int>& ans_current, std::set<int>& ans_all, std::vector<int>& pi)
{
	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;
      std::string part = "";
      //цикл для получения массива подстрок из текста
      while (k1 > 0)
      {
            part = "";
            part.append(t, begin, len_parts);
            str.push_back(part);
            begin += len_parts;
            k1--;
      }
      if (flag)
      {
            part = "";
            part.append(t, begin, (t.length() - (len_parts * (k - 1))));
            str.push_back(part);
      }

      //цикл для получения и проверки подстрок на стыках на каждом стыке проверя-
ется 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;

            std::cout << "---------------------------------------" << std::endl;
            std::cout << "Подстрака с центром на месте разреза - " << part <<
std::endl;
            std::cout << "Индексы в исходном тексте: ";
            for (int i = 0; i < part.size(); i++)
                  std::cout << i + top << ' ';
            std::cout << std::endl;

            std::cout << "Индексы:                    ";
            for (int i = 0; i < part.size(); i++)
            {
                  if (i + top > 9)
                        std::cout << i << "  ";
                  else
                        std::cout << i << " ";
            }
            std::cout << std::endl;
```

```
            std::cout << "Символы подстроки:         ";
            for (int i = 0; i < part.size(); i++)
            {
                  if (i + top > 9)
                        std::cout << part[i] << "  ";
                  else
                        std::cout << part[i] << " ";
            }
            std::cout << std::endl;

            ans_current = KMP(part, p, pi);
            if (ans_current.size() > 0)
            {
                  for (int i = 0; i < ans_current.size(); i++)
                  {
                        ans_current[i] += top; //определяем номер символа начала
подстроки в исходном тексте
                        ans_all.insert(ans_current[i]);
                  }
            }

            k1++;
      }
      }


      int main(){
      setlocale(LC_ALL, "Russian");

      std::cout << "\tСправка\nЧтобы запустить программу введите номер задачи или
ее название.\n"
            "\tНайдите все вхождения обзарца в тексте:\nНомер задачи -
1\nНазвание - KMP или kmp\n"
            "\tОпределить, является ли стока 1 циклическим сдвигом строки
2:\nНомер задачи - 2\nНазвание - Rotation или rotation" << std::endl;
      std::cout << std::endl;

      std::string task;
      std::cout << "Введите номер задачи или название алгоритма" << std::endl;
      std::getline(std::cin, task);

      if (task == "KMP" or task == "kmp" or task == "1")
      {
            std::string p, t;
            std::cout << "Введите текст" << std::endl;
            std::getline(std::cin, t);
            std::cout << "Введите образец (искомую подстроку)" << std::endl;
            std::getline(std::cin, p);

            if (t.length() < p.length())
            {
                  std::cout << "Образец не может быть больше текста!" <<
std::endl;
                  std::cout << -1 << std::endl;
```

```
            return 0;
        }

        int max_threads = sizeof(std::thread); // определяем максимально воз-
можное число потоков
        //-----------------------------------------------------
        // определяем на сколько частей можно раделить строку
        double alpha = (double)t.length() / (double)p.length();
        max_threads = std::min(max_threads, int(alpha) - 1);

        if (max_threads <= 0)
            max_threads = 1;
        int k;//= max_threads;

        if (max_threads == 1) {
            std::cout << "Длина исходного текста недостаточна для деления
строки" << std::endl;
            k = 1;
        }
        else
        {
            std::cout << "Введите число от 1 до " << max_threads <<
std::endl;
            std::cin >> k;

            while (k < 1 or k > max_threads)
            {
                std::cout << "Введите число от 1 до " << max_threads <<
std::endl;
                std::cin >> k;

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

        if (k > 1)
        {
            std::cout << "-----------------------------------------" <<
std::endl;
            std::cout << "Строка будет разделена на " << k << " частей" <<
std::endl;
        }

        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;

                std::cout << "Максимальная длинна части исходного текста - " <<
len_parts << std::endl;
                std::cout << "-----------------------------------------" <<
std::endl;
                std::cout << std::endl;

                split(t, p, k, str, ans_current, ans_all, pi);
                //-----------------------------------------------
                //заполняем исходный массив ответов
                for (int i = 0; i < str.size(); i++)
                {
                      std::cout << "----------------------\nЧасть исходного
текста     " << str[i] << std::endl;
                      std::cout << "Индексы в исходном тексте: ";
                      for (int j = 0; j < str[i].size(); j++)
                            std::cout << j + len_parts * i << ' ';
                      std::cout << std::endl;

                      std::cout << "Индексы:                    ";
                      for (int j = 0; j < str[i].size(); j++) {
                            if (j + len_parts * i > 9)
                                  std::cout << j << "  ";
                            else
                                  std::cout << j << " ";
                      }
                      std::cout << std::endl;

                      std::cout << "Символы подстроки:          ";
                      for (int j = 0; j < str[i].size(); j++) {
                            if (j + len_parts * i > 9)
                                  std::cout << str[i][j] << "  ";
                            else
                                  std::cout << str[i][j] << " ";
                      }
                      std::cout << std::endl;

                      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); // опреде-
ляем номер символа начала образца в исходном тексте
                            for (int j = 0; j < ans_current.size(); j++)
                                  ans_all.insert(ans_current[j]);
                      }
                }
          }
```

```
            // Ввывод ответа
            if (!ans_all.empty())
            {
                  int end = *ans_all.rbegin();
                  ans_all.erase(end);
                  std::cout << end << std::endl;
            }
            else
                  std::cout << -1 << std::endl;
      }
      else
      {
            if (task == "Rotation" or task == "rotation" or task == "2")
            {
                  std::string a, b;
                  std::cout << "Введите строки 1 и 2" << std::endl;
                  std::cin >> a >> b;
                  std::vector<int> pi = prefix_function(b);
                  std::cout << "Префикс-функция для строки 2" << std::endl;

                  for (int i = 0; i < pi.size(); i++)
                        std::cout << pi[i] << ' ';
                  std::cout << std::endl;

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

                  std::cout << "it_a - указатель на текущий символ в строке 1" <<
std::endl;
                  std::cout << "it_b - указатель на текущий символ в строке 2" <<
std::endl;

                  int it_a = 0, it_b = 0;
                  int cikle = 0;
                  int al = a.length();

                  while (true)
                  {
                        if (a[it_a] == b[it_b])
                        {
                              std::cout << "Совпадение:   " << a[it_a] << "==" <<
b[it_b] << " index: " << it_a << " " << it_b << std::endl;
                              it_a++;
                              it_b++;
                        }
                        if (it_a == al)
```

```
                        {
                                it_a = 0;
                                cikle++;
                        }
                        if (it_b == al)
                        {
                                std::cout << "Цикл: ";
                                std::cout << it_a << std::endl;
                                std::cout << "Ответом является текущей it_a + 1,
т.к. мы прошли всю строку и it_a указывает на ее конец" << std::endl;
                                return 0;
                        }
                        else
                        {
                                if (a[it_a] != b[it_b])
                                {
                                        std::cout << "Несовпадение: " << a[it_a] <<
"!=" << b[it_b] << " index: " << it_a << " " << it_b << std::endl;
                                        if (it_b == 0)
                                        {
                                                it_a++;
                                                std::cout << "Увеличиваем it_a" <<
std::endl;
                                        }
                                        else
                                        {
                                                it_b = pi[it_b - 1];
                                                std::cout << "Уменьшаем it_b" <<
std::endl;
                                        }
                                }
                        }
                        if (cikle > 1)
                        {
                                std::cout << -1 << std::endl;
                                return 0;
                        }

                }
        }
    }
    return 0;
    }
```