**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. 9382 _____ Кузьмин Д. И.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2021

**Цель работы.**

Изучить алгоритм Кнута-Морриса-Пратта для поиска подстроки. Освоить навыки разработки программ, реализующих этот алгоритм.

**Основные теоретические положения.**

Алгоритм Кнута-Морриса-Пратта — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

**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 $-1$

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

Определить, является ли A циклическим сдвигом BB (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.

**Описание функций и структур данных.**

1) В качестве строк использовался класс std::string

2) std::vector<int> countPrefixFunction(std::string str) – вычисление префикс-функции для строки, str – строка для которой вычисляется префикс-функция; возвращаемое значение – вектор из длин максимальных префиксов, являющихся также суффиксами для подстрок str, индексы соответствуют длинам подстрок

3) std::vector<int> findPatternKMP(std::string text, std::string pattern) – поиск подстроки с помощью алгоритма Кнута-Морриса-Пратта. Text – текст, в котором ищется подстрока, pattern – подстрока. Возвращаемое значение – вектор из индексов вхождений подстроки в тексте, либо -1, если подстрока в текст в не входит.

4) int circularShift(std::string shifted, std::string original) – проверка, является ли строка циклическим сдвигом другой. Shifted – сдвинутая строка, original – исходная строка. Возвращаемое значение – индекс вхождения префикса исходной строки в сдвинутую и -1, если строка сдвигом не является.

**Описание алгоритма (вычисление префикс-функции)**

1) Создается вектор длины n, заполняемый нулями. В нем будут хранится максимальные префикс-суффиксы для различных префиксов строки.

2) Далее следует проверка первой подстроки – ей будет являться подстрока длины 2, тогда сраниваются ее первый и последний символ и если они совпадают, то значение максимального префикса-суффикса для данной подстроки становится равным 1.

3) Далее идет проверка следующей подстроки. Ее максимальный префикс-суффикс может получиться из дополнения уже имеющегося префикса-суффикса. Проверяется следующий за предыдущим максимальным префиксом-суффиксом символ и если он совпадает с последним символом текущей подстроки, то макс. префикс можно увеличить еще на один. Если не совпадает, то рассматриваемый префикс-суффикс для дополнения меняется. Им становится максимальный

префикс-суффикс рассмотренного префикса-суффикса. Повторяются аналогичные действия пока не будет найдено совпадение символов (следующего за рассматриваемым префиксом и последним в текущей подстроке), тогда в вектор максимальных префиксов-суффиксов кладется длина рассматриваемого префикса, увеличенного на 1. Или когда рассматриваемый префикс не будет пустым. Если пуст, то при несовпадении первого символа с последним символом данной подстроки длине максимального префикса-суффикса текущей подстроки присваивается 0.

4) В результате получится вектор, состоящий из максимальных префиксов-суффиксов для префиксов исходной строки.

Сложность по времени можно оценить $O(n)$, по памяти – $O(n)$

**Описание алгоритма (поиск подстроки)**

1) На первом шаге вычисляется префикс-функция подстроки. Заводится переменная, показывающая количество совпавших символов на текущем шаге

2) Далее идет по циклу проверка символов (следующий за совпавшей части и текущий символ в цикле). Каждый совпавший символ увеличивает длину совпадающей части на 1.

3) В случае несовпадения символов длина совпадающей части становится равной максимальному префикс-суффиксу этой части.

4) Идет повтор 2 и 3го пункта пока длина совпадающей части не станет равна длине подстроки (это значит подстрока нашлась) или когда не закончится цикл, рассматривающий символы(это значит строка не нашлась).

Сложность по времени можно оценить $O(n)$, по памяти – $O(n)$

**Описание алгоритма (определение циклического сдвига)**

1) Строка 1 будет являться циклическим сдвигом строки 2, если строка 2 будет содержаться в удвоенной строке 1.

2) Запускается алгоритм КМП поиска строки 2 в строке, состоящей из двух строк 1 и если она находится, значит строка 1 – цикличский сдвиг строки 2.

Сложность по времени можно оценить $O(n)$, по памяти – $O(n)$

Исходный код см. в приложении А.

**Тестирование**

Результаты тестирования представлены в табл. 1. и табл. 2.

Таблица 1 — результаты тестирования алгоритма поиска подстроки.

| № п/п | Входные данные | Выходные данные | Комментарий |
|---|---|---|---|
| 1 | ab<br>abab | 0,2 | abab<br>abab |
| 2 | ab<br>abhkabab | 0,4,6 | abhkabab<br>abhkabab<br>abhkabab |
| 3 | bb<br>bbbb | 0,1,2 | bbbb<br>bbbb<br>bbbb |
| 4 | aa<br>ababa | -1 | Такой подстроки в ababa нет |
| 5 | aaa<br>aaa | 0 | Строки совпадают |

Таблица 2 — результаты тестирования алгоритма определения циклического сдвига

| № п/п | Входные данные | Выходные данные | Комментарий |
|---|---|---|---|
| 1 | defabc<br>abcdef | 3 | Строка defabc получится если abcdef сдвинуть на 3 символа |
| 2 | aaaaab<br>abaaaa | 4 | Строка aaaaab получится если abaaaa сдвинуть на 4 символа вправо |
| 3 | bbb<br>bbb | 0 | Одинаковые строки являются частным случаем циклического сдвига |

| | | | |
|---|---|---|---|
| 4 | aab<br>ba | -1 | Длина строк не совпадает |
| 5 | abcde<br>acbde | -1 | Строку abcde нельзя получить из циклического сдвига acbde |

**Выводы.**

Был изучен принцип алгоритма Кнута-Морриса-Пратта. Получены навыки разработки программ, реализующих этот алгоритм.

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <string>
#include <vector>
#define INFO

//вычисление префикс-функции
std::vector<int> countPrefixFunction(std::string str) {

	int m = str.size();
//вектор из значений максимальных префикс-суффиксов
#ifdef INFO
	std::cout << "Вычисление префикс-функции для "<< str<<"\n";
#endif
	std::vector<int> p(str.size());
	p[0] = 0;

	//итерация по строке
	for (int i = 1; i < m; i++) {

		//длина макс. префикс-суффикса текущего префикс-суффикса

		int len = p[i - 1];
#ifdef INFO
		std::cout << "Текущая подстрока " << str.substr(0, i + 1) <<
"\n";
		if (len > 0)
			std::cout << "Префикс предыдущей подстроки, который
возможно дополнить: " << str.substr(0, len) << "\n";
		else
			std::cout << "Префикс предыдущей подстроки является
нулевым\n";

#endif

		//цикл, который проверит все символы-кандидаты на составление
префикса и дойдет до нуля, если среди них нет подходящих
		while (1){

#ifdef INFO
			std::string pre = "_";
			if (len > 0)
				pre = str.substr(0, len);
			std::cout << "Пара-кандидат на дополнение префикса " <<
pre << " : " << str[len] << ", " << str[i] << "\n";
#endif
			//условие выхода из цикла
			if (len == 0 || str[len] == str[i]) break;
```

```
                //берется следующий возможный префикс-суффикс
                len = p[len - 1];

#ifdef INFO
                pre = "_";
                if (len > 0)
                     pre = str.substr(0, len);
                std::cout << "Невозможно дополнить текущий префикс,
переход к префиксу " << pre << "\n";
#endif
           }
           //увеличение длины префикса
           if (str[len] == str[i])
                len += 1;

           p[i] = len;
#ifdef INFO
           std::cout << "Получен префикс длины " << len << " для
подстроки длины " << i + 1 << "\n\n";
#endif

     }
#ifdef INFO
     std::cout << "Полученная префикс-функция: ";
     for (auto& it : p)
          std::cout << it << " ";
     std::cout << "\n";
#endif
     return p;

}
//поиск подстроки
std::vector<int> findPatternKMP(std::string text, std::string pattern){

     int n = text.size();
     int m = pattern.size();
#ifdef INFO
     std::cout << "Поиск подстроки по алгоритму Кнута-Морриса-Пратта\n";
     std::cout << "Текст - " << text << "\nСтрока: " << pattern <<
"\n\n";
#endif
     std::vector<int> indexes;
     std::vector<int> pattern_prefixes = countPrefixFunction(pattern);
     int match = 0;
#ifdef INFO
     std::cout << "Переход к поиску подстроки\n\n";
#endif

     for (int i = 0; i < n; i++) {
#ifdef INFO
           std::cout << "Длина текущего совпадения: " << match << "\n";
```

```
            std::cout << "Символ текста: " << text[i] << "\nСимвол искомой
строки: " << pattern[match % m] << "\n\n";
#endif
            //обновление совпадающей части
            while (match > 0 && pattern[match] != text[i]){
#ifdef INFO
                  std::string pre = "_";
                  int newMatch = pattern_prefixes[match - 1];
                  if (newMatch > 0)
                        pre = pattern.substr(0, newMatch);
                  std::cout << "Обновление совпадающей части " <<
pattern.substr(0, match) << " -> " << pre <<"\n";
#endif
                  match = pattern_prefixes[match - 1];
            }


            //если символы на текущей позиции в строках совпадают
            //длина совпадения увеличивается
            if (pattern[match] == text[i]) {
#ifdef INFO
                  std::cout << "Символы совпадают. Увеличение длины
совпадающей части\n\n";
#endif
                  match++;
            }
            //если не совпадает, то она становится равна длине префикса
            else if (match != 0) {
#ifdef INFO
                  std::cout << "Найден несовпадающий символ. Изменение
длины совпадающей части\n\n";
#endif
                  match = pattern_prefixes[match - 1];

            }

            //если в какой-то момент длина совпадения станет равной длине
строки
            if (match == m) {
#ifdef INFO
                  std::cout << "Найдено совпадение\nИндекс начала искомой
строки в тексте: " << i - match + 1 << "\n\n";
#endif
                  indexes.push_back(i - match + 1);
            }

#ifdef INFO
            if (match == 0)
            std::cout << "Найден несовпадающий символ. Длина совпадения
остается = 0\n\n";
#endif
```

```
	}
#ifdef INFO
	std::cout << "Алгоритм завершен\n";
#endif
	return indexes;
}

//вывод индексов вхождений подстроки
void printIndexes(std::vector<int> indexes){

	if (!indexes.empty()) {
		int _len = indexes.size() - 1;
		for (int i = 0; i < _len; i++)
			std::cout << indexes[i] << ", ";
		std::cout << indexes[_len];
		return;
	}
	std::cout << "-1";
}

//проверка циклического сдвига
int circularShift(std::string shifted, std::string original) {

	if (shifted.size() != original.size())
		return -1;
#ifdef INFO
	std::cout << "Проверка циклического сдвига. \n\n";
#endif
	std::vector<int> indexes = findPatternKMP(shifted + shifted,
original);
	if (!indexes.empty())
		return indexes[0];

	return -1;

}
//вывод результата проверки циклического сдвига
void doCircularShiftCheck() {
	std::string s1;
	std::string s2;
	std::cin >> s2 >> s1;
	int index = circularShift(s2, s1);
#ifdef INFO
	if (index != -1) {
		std::cout << "Строка " << s2 << " является циклическим сдвигом
строки " << s1 << "\n";
		std::cout << "Индекс начала вхождения: ";
	}
	else {
		std::cout << "Строка " << s2 << " не является является
циклическим сдвигом строки " << s1 << "\n";
```

```
            std::cout << "Релуьтат: ";
      }
#endif
      std::cout << index<<"\n";
}
//вывод результата алгоритма КМП
void doKMPSearch() {
      std::string s1;
      std::string s2;
      std::cin >> s2 >> s1;
      std::vector<int> indexes = findPatternKMP(s1, s2);
#ifdef INFO
      if (!indexes.empty()) {

            std::cout << "Индексы вхождений строки " << s2 << " в "
<<s1<<": ";
      }
      else {
            std::cout << "Строка " << s2 << " не входит в" << s1 << "\n";
            std::cout << "Результат: ";
      }
#endif
      printIndexes(indexes);
      std::cout<<"\n";
}
int main() {

      setlocale(LC_ALL, "rus");
      doKMPSearch();
      //doCircularShiftCheck();
      return 0;
}
```