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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 8304 Алтухов А.Д.

Преподаватель Размочаева Н. В.

Санкт-Петербург

2020

Цель работы.

Построить алгоритм Кнута-Морриса-Пратта для определения всех вхождений подстроки в строку, определить его сложность.

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).

Основные теоретические положения.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P и текста T, 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.

Описание алгоритма.

Если искомая подстрока намного меньше строки, то строка делится на несколько частей, в зависимости от подстроки. Для каждой из них запускается стандартный алгоритм КМП, который вычисляет префикс-функцию и на основе полученных значений становятся возможным сдвиги сразу на несколько символов вперед при поиске, так как суффикс обрабатываемой части строки для поиска может быть равен префиксу искомой подстроки, что и отображает префикс-функция. Помимо прочего, возвращаются данные о том, сколько конечных символов строки равно начальным символам подстроки. Исходя из этих данных сравниваются начала следующих частей разбиения с конечной частью подстроки и результаты дополняются.

Время работы алгоритма ограничено $O(m + n)$, где m – это длина образца, а n – длина текста.

Требуемая память: $O(2m + n)$, хранятся данные префикс-функции, образец и текст.

Задание по определению циклического сдвига выполняется путем применения алгоритма КМП на двойную строку.

Описание основных структур данных и функций.

std::vector<int> notStandartKMP(std::string& str, std::string& example, std::vector<int>& suffixes) – работает как обычный алгоритм КМП, но собирает информацию о том, сколько последних символов совпало с началом example.

std::vector<int> parallelCMP(std::string& str, std::string& example) – разделяет исходную строку на части, обрабатывает результаты.

std::vector<int> prefix(std::string& example) – префикс-функция.

int cyclicKMP(std::string& str, std::string& example) – функция для определения циклического сдвига.

Тестирование.

Таблица 1 – Результаты тестирования.

Ввод	**Вывод**
ab abab	0,2
Aabbccabcssqqwweerrttyyuuiioabcoppaassddfabchhj jkkllzzabcxxccvvbbnnmmqwertasdabcfgzxcvb abc	6,28,41,54,77

Вывод.

В ходе работы был построен алгоритм Кнута-Морриса-Пратта для поиска вхождений подстроки методом, соответствующим варианту.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

```
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
#include <locale>
#include <fstream>

std::ostream* out;
std::istream* in;

std::vector<int> prefix(std::string& example) {
	std::vector<int> pi(example.size());
	int k = 0;
	pi[0] = 0;
	for (int i = 1; i < example.size(); ++i) {
		while (k > 0 && example[k] != example[i]) {
			k = pi[k];
		}
		if ( example[k] == example[i]) {
			k += 1;
		}
		pi[i] = k;
	}
	return pi;
}

std::vector<int> standartKMP(std::string& str, std::string& example)
{//обычный кмп без параллелизации

	auto pi = prefix(example);
	std::vector<int> result;

	int k = 0;
	for (int i = 0; i < str.size(); i++) {
		if (k > 0 && str[i] != example[k]) {
			*out << "Несовпадение в " << i << ", следующее k " << pi[k
- 1] << "\n";
			k = pi[k - 1];
		}
		if (str[i] == example[k]) {
			*out << "Совпадение в " << i << "\n";
			k++;
		}
		if (k == example.size()) {
			*out << "Новый результат! Индекс " << i - example.size() +
1 << "\n";
```

```
                result.push_back(i - example.size() + 1);
                k = pi[k - 1];
            }
        }
        return result;

}

std::vector<int> notStandartKMP(std::string& str, std::string& example,
std::vector<int>& suffixes) {//дополнительно заполняет массив суффиксов,
который отражает, сколько последних символов совпало с началом example

        auto pi = prefix(example);
        std::vector<int> result;
        *out << "Обработка " << str << "\n";
        int k = 0;
        for (int i = 0; i < str.size(); i++) {
            if (k > 0 && str[i] != example[k]) {
                *out << "Несовпадение в " << i << ", следующее k " << pi[k
- 1] << "\n";
                k = pi[k - 1];
            }
            if (str[i] == example[k]) {
                *out << "Совпадение в " << i << "\n";
                k++;
            }
            if (k == example.size()) {
                *out << "Новый результат! Индекс " << i - example.size() +
1 << "\n";
                result.push_back(i - example.size() + 1);
                k = pi[k - 1];
            }
        }
        *out << "Совпало последние " << k << " символов\n";
        suffixes.push_back(k);
        return result;

}

std::vector<int> parallelCMP(std::string& str, std::string& example) {

        int maxSize = 10;
        if ((str.size() / maxSize) < example.size()) {
            return standartKMP(str, example);
        }

        //разделение строки на части

        std::vector<std::string> parts;
        int maxParts = str.size() / example.size()/maxSize;
        maxSize = str.size()/maxParts;
```

```
	for (int i = 0; i <= maxParts; i++) {
		parts.push_back(str.substr(i * maxSize, maxSize));
	}
	*out << "Разделение строки: \n";
	for (int i = 0; i < parts.size(); i++) {
		*out << parts[i] << "\n";
	}

	std::vector<int> suffixes;
	std::vector<std::vector<int>> results(parts.size());

	//поиск по каждой части

	for (int i = 0; i < parts.size(); i++) {
		results[i] = notStandartKMP(parts[i], example, suffixes);
	}

	//поиск дополнительных результатов в случае объединения частей
	for (int i = 0; i < results.size()-1; i++) {

		if (suffixes[i] > 0) {
			*out << "Попытка найти новый результат в объединении частей
" << i << " и " << i + 1 << "...";

			if (parts[i + 1].substr(0, example.size() - suffixes[i]) ==
example.substr(suffixes[i])) {
				*out << " Успешно!\n";
				results[i].push_back(parts[i].size() - suffixes[i]);
			}
			else {
				*out << " Безуспешно!\n";
			}
		}
	}

	//правка найденных  индексов
	std::vector<int> oneBigResult;
	for (int i = 0; i < results.size(); i++) {
		for (int j = 0; j < results[i].size(); j++) {
			oneBigResult.push_back(results[i][j] + i * maxSize);
		}
	}
	return oneBigResult;
}

int cyclicKMP(std::string& str, std::string& example) {
	if (str.size() != example.size()) {
		return -1;
	}
```

```
	auto pi = prefix(example);
	std::string doubleStr = str + str;
	int k = 0;
	for (int i = 0; i < doubleStr.size(); i++) {
		if (k > 0 && doubleStr[i] != example[k]) {
			k = pi[k - 1];
		}
		if (doubleStr[i] == example[k]) {
			k++;
		}
		if (k == example.size()) {
			return i - str.size() + 1;
		}
	}
	return -1;
}

void printKMPResults(std::vector<int>& result) {
	if (result.empty()) {
		*out << -1;
		return;
	}
	for (int i = 0; i < result.size(); i++) {
		*out << result[i];
		if (i != result.size() - 1) {
			*out << ",";
		}
	}
}

int main() {

	setlocale(LC_ALL, "Russian");

	int mode;
	int inputMode, outputMode;
	std::cout << "Режим работы (0 - поиск всех вхождений в строку, 1 -
определить циклический сдвиг): ";
	std::cin >> mode;
	std::cout << "Ввод из... (0 - из консоли, 1 - из файла): ";
	std::cin >> inputMode;
	std::cout << "Вывод из... (0 - из консоли, 1 - из файла): ";
	std::cin >> outputMode;


	std::ifstream inFile("input.txt");
	std::ofstream outFile("output.txt");
	in = inputMode == 0 ? &std::cin : &inFile;
	out = outputMode == 0 ? &std::cout : &outFile;
```

```
	std::string a, b;
	*in >> a;
	*in >> b;


	//std::cout << cyclicKMP(a, b);
	//auto res = standartKMP(a, b);
	//printKMPResults(res);
	if (mode == 1) {
		*out << cyclicKMP(a, b);
	}
	else {
		auto res = parallelCMP(a, b);
		printKMPResults(res);
	}
	inFile.close();
	outFile.close();

	return 0;
}
```