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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «ПиАА»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент(ка) гр. 0000 _____ Ивченко А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с алгоритмом Кнута-Морриса-Пратта поиска всех вхождений образца в строке.

Формулировка задания.

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 - T

Вторая строка - P

Выход:

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

Вар. 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).

Функции и структуры данных.

`std::vector<int> prefix_function(const std::string& s)` – нахождение значений префикс функции для поисковой строки.

`int cyclicalKMP(std::string& str, std::string& sample, std::ostream& out)` — функция вычисляет циклический сдвиг образца относительно строки с помощью измененного алгоритма КМП.

`std::vector<int> simpleKMP(std::string& t, std::string& s, std::ostream& out)` – алгоритм КМП.

`std::vector<std::string> parallel_string(std::string& str, std::string& sample)` – разбиение строки на sample.size() равных элементов.

`std::vector<std::string> split_str` — список обрезков строки.

Описание алгоритма.

Программа разбивает поисковую строку на k равных элементов, к каждому из которых применяется алгоритм КМП. Алгоритм вычисляет префикс функцию для строки t + ‘#’ + s, на основе которой ищется совпадение pi[i] = sample.size(). Полученное значение pi[i] - это последний элемент вхождения образца в строку поиска.

Во избежание неоднозначности решения смежные обрезки поисковой строки "перекрываются" sample.size()-1 элементами. Найденные позиции вхождений обрабатываются, исходя из порядкового номера каждого обрезка строки.

В качестве промежуточной информации выводятся обрезки поисковой строки и позиции вхождений образца в них. Значения префикс функции для каждого обрезка записывается в файл.

Время работы алгоритма оценивается как O(P + T), где P – длина образца, T – длина поисковой строки.

Тестирование

Рисунок 1 - Входные данные

T = asbfafbasbfsabfasfannbasbfabbababfaabfafabfaba

S = aba

Рисунок 2 - результат работы программы

```
Input: console = 0; file = 1
0
asbfafbasbfsabfabfabfbasbfabbababfaabfafabfaba
aba
Output:
console(simpleKMP)&file(prefix-function) = 0
console(cyclical)&file(prefix-function) =  1
console(parallelKMP)&file(prefix-function) = 2
2
asbfafbasbfsabfab
Not any entries
abfabfbasbfabbaba
14
babfaabfafabfaba
13
29,43
```

Вывод.

В ходе лабораторной был реализован алгоритм поиска вхождений в строке с помощью метода, описанного в условии варианта №1; а также была оценена сложность алгоритма по времени.

Исходный код

```
#include <vector>
#include <string>
#include <iostream>
#include <fstream>
#include <math.h>

std::vector<int> prefix_function(const std::string& s) {

	std::vector<int> pi(s.length(), 0);

	for (int i = 1; i < s.length(); i++) {
		int j = pi[i - 1];

		while (j > 0 && s[i] != s[j]) {
			j = pi[j - 1];
		}

		if (s[i] == s[j]) {
			pi[i] = j + 1;
		}
		else {
			pi[i] = j;
		}

	}
	return pi;
}

int cyclicalKMP(std::string& str, std::string& sample, std::ostream& out) {

	if (str.size() != sample.size()) {
		std::cout << "Incorrect\n";
		return -1;
	}

	std::vector<int> solution;
	std::string search_str = sample + '.' + str + str;

	std::vector<int> pi = prefix_function(search_str);
	for (auto a : pi) out << a << ' ';

	for (int i = 0; i < str.length() * 2; i++) {
		if (pi[sample.length() + 1 + i] == sample.length())
			return i - sample.length() + 1;
	}
	return -1;
}

std::vector<int> simpleKMP(std::string& t, std::string& s, std::ostream& out) {

	std::string search = s + '.' + t;

	std::vector<int> pi = prefix_function(search);
	for (auto a : pi) out << a << ' ';

	std::vector<int> solution;

	for (int i = 0; i < t.length(); i++) {
```

```
			if (pi[s.length() + 1 + i] == s.length()) {
				solution.push_back(i - s.length() + 1);

			}
		}
		if (solution.size() == 0) { std::cout << "Not any entries"; }
		else return solution;

}

std::vector<std::string> parallel_string(std::string& str, std::string& sample) {

		std::vector<std::string> parts;

		int partSize = str.size() / sample.size();
		int partQ = str.size() / partSize;

		for (int i = 0; i < partQ; i++) {
			parts.push_back(str.substr(i * partSize , partSize + sample.size() - 1));
		}
		return parts;



}
int main() {


		std::string s, t;

		int input_ch, output_ch;

		std::cout << "Input: console = 0; file = 1\n";
		std::cin >> input_ch;

		if (input_ch == 1) {
			std::ifstream infile("input.txt");
			infile >> t >> s;
		}
		else if (input_ch == 0) std::cin >> t >> s;

		std::cout << "Output:\nconsole(simpleKMP)&file(prefix-function) = 0\
nconsole(cyclical)&file(prefix-function) =  1\nconsole(parallelKMP)&file(prefix-function) =
2" << std::endl;
		std::cin >> output_ch;

		std::ofstream file;
		file.open("prefix_function.txt");
		if (!file.is_open()) {
			std::cout << "Incorrect!\n";
			return 0;
		}

		if (output_ch == 0) {
			std::vector <int> result = simpleKMP(t, s, file);
			for (int i = 0; i < result.size(); i++) {
				std::cout << result[i];
				if (i != result.size() - 1)
					std::cout << ",";
			}
		}
		else if(output_ch == 1){
			std::cout << cyclicalKMP(t, s, file);
		}
```

```
        else if (output_ch == 2) {
                std::vector<std::string> split_str = parallel_string(t, s);
                std::vector<int> pos;
                int k = 0, q = 0;
                for (auto a : split_str) {

                        for (auto j : a)
                                std::cout << j;
                        std::cout << std::endl;

                        std::vector<int> part_solution = simpleKMP(a, s, file);
                        for (auto i : part_solution) {
                                pos.push_back(i + k * (t.size() / s.size()));
                                std::cout << i << ' ';
                        }
                        std::cout << std::endl;
                        k++;
                }
                for (int i = 0; i < pos.size(); i++) {
                        std::cout << pos[i];
                        if (i != pos.size() - 1)
                                std::cout << ",";

                }

        }
        file.close();
        return 0;

}
```