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. 9383 _____ Камзолов Н.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Познакомиться на практике с алгоритмом Кнута-Морриса-Пратта, реализовать данный алгоритм на языке программирования C++.

Задание.

1 задача:

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.

Входные данные:

Первая строка – P

Вторая строка – T

Выходные данные:

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

2 задача:

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

Входные данные:

Первая строка – A

Вторая строка – B

Выходные данные:

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

Основные теоретические положения.

Префикс строки – какое-то количество начальных символов строки, но не равное всей строке.

Суффикс строки - какое-то количество конечных символов строки, но не равное всей строке.

Префикс-функция - длина наиболее длинного префикса, являющегося одновременно суффиксом, есть значение префикс-функции от строки для индекса j.

Алгоритм Кнута-Морриса-Пратта – эффективный алгоритм, осуществляющий поиск подстроки в строке.

Описание алгоритма.

- ***Нахождение значения префикс-функции для всех элементов строки:***
  1. Элемент с индексом 0 всегда имеет значение префикс-функции равное 0.
  2. Далее для работы алгоритма нам потребуются два указателя i = 1 и j = 0. i будет отвечать за текущий индекс суффикса в строке, а j за текущий индекс префикса.
  3. В том случае, если значения по индексу i и j не равны и j = 0, то кладем по индексу i в значения префикс-функции 0. Если же j != 0, то приравниваем j к значению префикс-функции по индексу j-1.
  4. В том случае, если значения по индексу i и j совпадают, кладем по индексу i в значения префикс-функции j+1 и сдвигаем оба значения j и i на единицу в сторону конца строки.
  5. Алгоритм завершает свою работу в том случае, если символ i достиг конца строки.
- ***Алгоритм Кнута-Морриса-Пратта:***
  1. Считаем префикс-функцию подстроки, которую будем подставлять в строку.
  2. Далее нам снова понадобится 2 указателя l = 0 и k = 0, где k будет отвечать за текущий индекс элемента, с которым происходит сравнение, строки, в которой осуществляется поиск, а k – будет отвечать за текущий элемент подстроки, которую мы хотим найти в строке.

3. В том случае, если символы по текущим индексам l и k совпадают, мы прибавляем к обоим индексам по единице и продолжаем сравнение. В том случае, если индекс l достиг конца строки, мы говорим о том, что одно из вхождений подстроки в строку найдено.
4. Если же символы по текущим индексам l и k не совпадают и l != 0, то в l кладется значение префикс-функции по индексу l-1. Если при том же условии l = 0, то k увеличивается на единицу и алгоритм продолжает свою работу.
5. Алгоритм завершает свою работу в том случае, если индекс k достигает конца строки.

Для решения второй задачи, можем прибавить к строке B ее дупликат. И далее осуществлять поиск подстроки A в строке BB.

Временная сложность алгоритма Кнута-Морриса-Пратта O(m+n), где n – длина подстроки, а m – длина строки. Так как сложность построения префикс-функции O(n), а сложность дальнейших сравнений O(m).

Емкостная сложность алгоритма – O(n), где n – длина подстроки, так как нам нужен массив для хранения префикс-функции.

Описание функций и структур данных.

void readStrings() – функция для считывания строки и подстроки из входного потока.

void printAnswer() – функция для вывода ответа на задачу в выходной поток(рознится в зависимости от задачи).

std::vector<int> prefixFunc() – функция для нахождения массива, который хранит значение префикс-функции для каждого элемента, искомой подстроки.

std::vector<int> KMP() – функция, реализующая алгоритм Кнута-Морриса-Пратта.

Было написано тестирование для некоторых функций программы:

1. Тестирование функции readStrings() – была протестирована корректная обработка входных данных программы.

2. Тестирование функции prefixFunc() – было протестировано корректное построение префикс-функции для различных строк.
3. Тестирование функции KMP() – была протестирована работа алгоритма Кнута-Морриса-Прата на нескольких входных данных.

Тестирование(Задание 1).

Входные данные	Выходные данные
ab abab	0, 2
ghf wfjghfwdjghf	3, 9
aaa bbbbbbbbbb	-1
a aaaaaaaaa	0,1,2,3,4,5,6,7,8,9

Тестирование(Задание 2).

Входные данные	Выходные данные
defabc abcdef	3
abctabct tabctabc	3
asdasd asdas	-1
aaaa aaaa	0

Выводы.

На практике были применены теоретические знания об алгоритме Кнута-Морриса-Пратта. Была успешно написана программа, реализующая данный алгоритм. Данный алгоритм работает гораздо быстрее наивного алгоритма поиска подстроки в строке, а также может быть использован для проверки того, являются ли строки циклическим сдвигом друг друга.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main_1.cpp:

```
#include "KMP/KMP.h"

void printAnswer(std::vector<int>& answer) {
    if (answer.size() == 0) {
        std::cout << -1;
        return;
    }
    for(size_t i = 0; i < answer.size(); i++) {
        if(i < answer.size() - 1) std::cout << answer[i] << ',';
        else std::cout << answer[i] << '\n';
    }
}


int main() {
    std::string substring, text;
    readStrings(substring, text, std::cin);
    std::vector<int> answer = KMP(substring, text);
    printAnswer(answer);
}
```

Файл main_2.cpp:

```
#include "KMP/KMP.h"

void printAnswer(std::vector<int>& answer) {
    if (answer.empty()) {
        std::cout << -1;
        return;
    }
    std::cout << answer[0];
}


int main() {
    std::string A, B;
    readStrings(A, B, std::cin);
    if(A.size() != B.size()) {
        std::cout << -1;
        return 0;
    }

    if(A == B) {
        std::cout << 0;
        return 0;
    }
    A = A + A;
    std::vector<int> answer = KMP(B, A);
    printAnswer(answer);
}
```

Файл KMP.h:

```
#ifndef KMP_H
#define KMP_H

#include <iostream>
#include <string.h>
#include <fstream>
#include <vector>
#include <cstdlib>

void readStrings(std::string& str1, std::string& str2, std::istream& in);
std::vector<int> prefixFunc(std::string& text);
std::vector<int> KMP(std::string& substring, std::string& text);

#endif
```

Файл KMP.cpp:

```
#include "KMP.h"

void readStrings(std::string& str1, std::string& str2, std::istream& in)
{
    in >> str1 >> str2;
}


std::vector<int> prefixFunc(std::string& text) {
    std::vector<int> prefix(text.size());
    int j = 0;
    for(size_t i = 1; i < text.size(); i++) {
        if(text[j] == text[i]) {
            prefix[i] = j + 1;
            j++;
            continue;
        }
        if(j == 0) {
            prefix[i] = 0;
            continue;
        }
        j = prefix[j - 1];
        i--;
    }

    return prefix;
}

std::vector<int> KMP(std::string& pattern, std::string& text) {
    std::vector<int> answer;
    std::vector<int> prefix = prefixFunc(pattern);
    int l = 0;
    for (size_t k = 0; k < text.size(); k++) {
        if(text[k] == pattern[l]) {
            l++;
            if(l == (int)pattern.size()) {
```

```
                answer.push_back(k + 1 - pattern.size());
            }
            continue;
        }
        if(l == 0) continue;
        l = prefix[l-1];
        k--;
    }
    return answer;
}
```