МИНОБ*Р*НАУКИ *Р*ОССИИ
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. 8383 _____ Костарев К.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с принципами работы алгоритма Кнута-Морриса-Пратта нахождения подстроки в строке и реализовать с оптимизацией; научиться применять алгоритм для решения задач.

Задача.

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.

   Sample Input:

   ```
   ab
   abab
   ```

   Sample Output:

   ```
   0,2
   ```

2. 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.

   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
```

Вариант № 2.

Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длина образца. Это возможно, если не учитывать память, в которой хранится строка поиска.

Основные теоретические сведения.

Префикс-функцией строки S длины N называется вектор размера N из чисел, где каждое i-ое число определяется как максимальная длина префикса подстроки в строке S от первого до i-ого символа, которая посимвольно равна суффиксу этой же подстроки. Для первого символа строки значение полагается равным 0.

Описание алгоритма.

Допустим, P – строка (образец), которую необходимо найти в строке T (тексте). Рассмотрим сравнение строк на позиции i, где образец P[0, m-1] сопоставляется с частью текста T[i, i+m-1]. Предположим, что первое несовпадение произошло между T[i+j] и P[j], где 1<j<m. Тогда T[i, i+j-1] = P[0, j-1] и a = T[i+j] ≠ P[j] = b.

При сдвиге вполне можно ожидать, что префикс (начальные символы) образца P сойдется с каким-нибудь суффиксом (конечные символы) текста T. Длина наиболее длинного префикса, являющегося одновременно суффиксом, есть значение префикс-функции от строки P для индекса j.

Таким образом алгоритм заключается в следующем: пусть p[j] — значение префикс-функции от строки P[0,m-1] для индекса j. Тогда после сдвига мы

можем возобновить сравнения с места T[i+j] и P[p[j]] без потери возможного местонахождения образца.

Так как каждый символ в тексте T обрабатывается отдельно и поочерёдно, то нет необходимости хранить весь текст в памяти: достаточно сразу обрабатывать каждый символ текста при считывании, а хранить только образец и префикс-функцию образца. Таким образом получается оптимизация по памяти.

Задача 2 на определение того, является ли строка A циклическим сдвигом строки B, является примером использования алгоритма: достаточно дублировать строку A и произвести поиск строки B в удвоенной строке A (удваивая строку A, соединяются ее суффикс и префикс, а следовательно, если строки являются циклическими сдвигами друг друга, должна получиться подстрока B). Как только будет найден первая совпавшая подстрока, алгоритм прекращает работу (т.к. необходимо найти минимальный индекс).

Описание структур данных и функций для первой задачи.

1. string p – образец, который необходимо найти в тексте.
2. vector<int> answer – вектор индексов вхождений строки p в тексте.
3. vector<int> prefixes – префикс-функция образца;
4. char c – символ текста, которые поочередно считываются и обрабатываются;
5. void prefixFunc(string p, vector<int> &prefixes) – функция нахождения префикс-функции для образца.

   Аргументы: p – образец, prefixes – ссылка на массив значений префикс-функции.
6. vector<int> KMPFunc() – реализация алгоритма, возвращает вектор answer.

Описание структур данных и функций для второй задачи.

1. string a, b – соответственно строка A, для которой надо проверить, является ли она циклическим сдвигом строки B;
2. vector<int> prefixes – префикс-функция строки B;

3. void prefixFunc(string b, vector<int> &prefixes) – функция нахождения префикс-функции для строки B.

   Аргументы: b – строка, prefixes – ссылка на массив значений префикс-функции.
4. int<int> KMPFunc() – реализация алгоритма, возвращает индекс первого вхождения строки B в строке A, или -1, если вхождений нет.

Сложность алгоритма по времени.

До начала поиска образца длины M в тексте длины N вычисляется префикс-функция образца, где для каждого символа она вычисляется за одну итерацию, и потом уже происходит поиск подстроки в тексте, который происходит тоже посимвольно на каждой итерации. Таким образом, сложность алгоритма по времени равна

$$O(M+N).$$

Сложность алгоритма по памяти.

Для первой задачи реализация алгоритма требует хранения в памяти только лишь строки образца, следовательно в этом случае сложность по памяти составляет

$$O(M).$$

Во второй задаче необходимо хранение обеих строк A и B, так как для реализации алгоритма требуется нахождение префикс-функции для строки B и дублирование строки A, поэтому в этом случае сложность равна

$$O(|A|+|B|).$$

Тестирование программы решения первой задачи.

При ручном вводе данных с консоли, программа заканчивает считывание при нажатии Enter. Демонстрация работы программы приведена на рис. 1. Входные данные для демонстрации:

```
aba
```

abbaba

```
aba
abbaba
Вычисление префикс-ф p для образца P = aba
 p[0] = 0
 p[1]:
  Предыдущее значение p[0]=0
 p[1] = 0
 p[2]:
  Предыдущее значение p[1]=0
  P[0] = P[2] = a, прибавляем значение на единицу
 p[2] = 1
Префикс-ф для образца равна [0,0,1]
Начинаем поиск, посимвольно обрабатывая строку, indexP = 0
 T[0] = a:
  P[0] = T[0] = a, увеличиваем indexP на 1
 T[1] = b:
  P[1] = T[1] = b, увеличиваем indexP на 1
 T[2] = b:
  Возвращаемся навзад, пока indexP не станет равным 0 или P[indexP] станет равным T[2]
  indexP = p[indexP-1] = p[1] = 0
 T[3] = a:
  P[0] = T[3] = a, увеличиваем indexP на 1
 T[4] = b:
  P[1] = T[4] = b, увеличиваем indexP на 1
 T[5] = a:
  P[2] = T[5] = a, увеличиваем indexP на 1
  indexP = длине образца = 3, подстрока найдена
  Индекс первого вхождения = indexT - длина образца + 1 = 5-3+1
  Пушаем в массив индексов, indexP теперь равен предыдущму значению p[2-1] = 1
3
```

Рисунок 1 – Демонстрация работы первой программы

Тестирование программы приведено в табл. 1. В ходе тестирования все выходные данные оказались корректными.

Таблица 1 – Тестирование первой программы

Входные данные	Выходные данные
ab ba	-1
a a	0
aba abababbaba	0,2,7
abbcbbca bbcbbcabbcbbccaabbcbbabbcbbca	21
abcbabcbabc abcbabcbabcabcbcbaabcabccbaabcbcabcba	0

Тестирование программы решения второй задачи.

При ручном вводе данных с консоли, программа заканчивает считывание при нажатии Enter. Демонстрация работы программы приведена на рис. 2. Входные данные для демонстрации:

```
bcdae
daebc
```

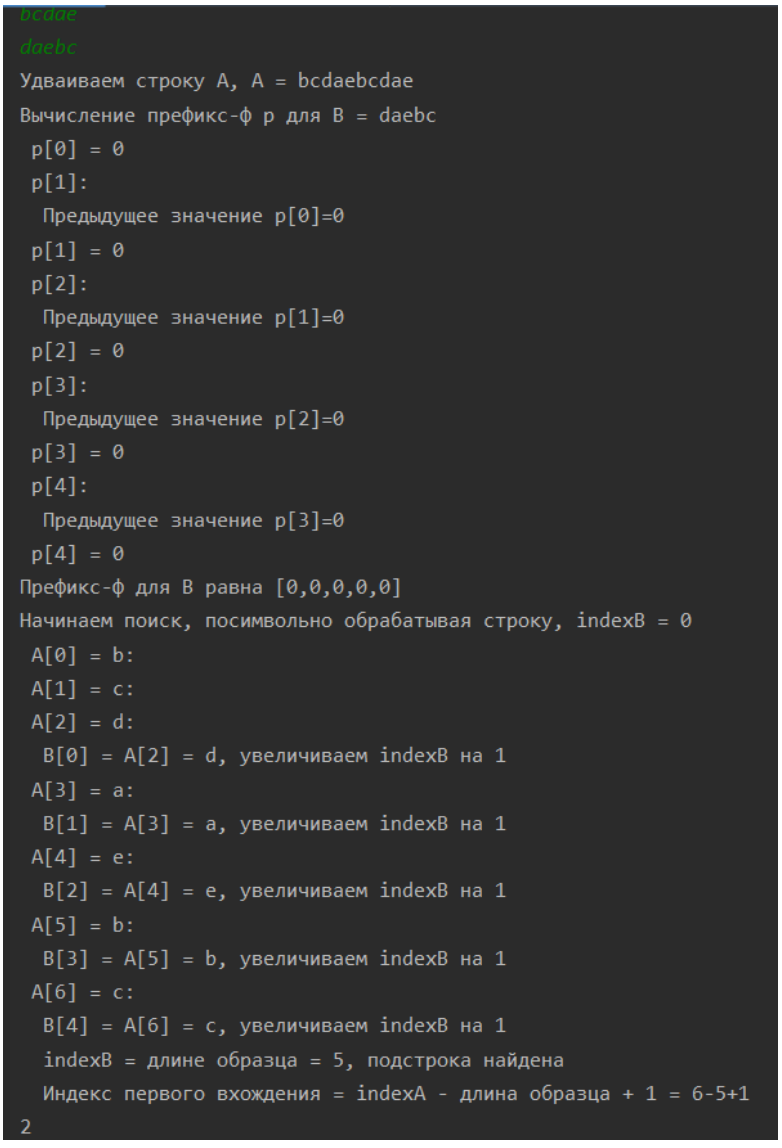

Рисунок 2 – Демонстрация работы второй программы

Тестирование программы приведено в табл. 2. В ходе тестирования все выходные данные оказались корректными.

Таблица 2 – Тестирование второй программы

Входные данные	Выходные данные
ab ba	1
a a	0
abcbca bcbcaa	1
aaa aaaa	-1
ababababa aababababab	8
abcabcabc cbacbacba	-1

Выводы.

В данной лабораторной работе был изучен алгоритм Кнута-Морриса-Пратта нахождения подстроки в строке. Алгоритм был реализован с оптимизацией по памяти, так как строка текста не хранится в памяти, а обрабатывается посимвольно при считывании, таким образом, сложность по памяти линейна и зависит только от длины образца. Также была решена задача определения циклического сдвига, но так как в данном случае необходимо удвоение одной из строк и при посимвольном считывании это невозможно, то сложность алгоритма уже возрастает и зависит от суммы длин обеих строк.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ РЕШЕНИЯ ПЕРВОЙ ЗАДАЧИ

```
#include <iostream>
#include <string>
#include <vector>
//#include <windows.h>

void printArray(std::vector<int>& arr){
    for (unsigned int i = 0; i < arr.size() - 1; i++) {
        std::cout << arr[i] << ',';
    }
    std::cout << arr[arr.size() - 1];
}

void prefixFunc(std::string &p, std::vector<int>& prefixes){ //вычисление
префикс-ф для образца
    std::cout << "Вычисление префикс-ф p для образца P = " << p <<
std::endl;
    std::cout << " p[0] = 0" << std::endl;
    for (unsigned int index = 1; index < p.size(); index++){    //проход
по каждому символу строки кроме первого
        std::cout << " p[" << index << "]: " << std::endl;
        int previous = prefixes[index-1]; //изначально равен значению
предыдущего символа
        std::cout << "  Предыдущее значение p[" << index-1 << "]=" <<
previous << std::endl;
        while (previous > 0 && p[previous] != p[index]){   //если
предыдущее положительно и не равно исходному символу...
            std::cout << "  Оно больше нуля и P[" << previous << "] !=
P[" << index << "] = " << p[index] << ", берем p[" << previous-1 << "]"
<< std::endl;
            previous = prefixes[previous-1];    //...берем еще назад
        }
        if (p[previous] == p[index]){ //если символы образца равны, то
прибавляем значение на единицу
            std::cout << "  P[" << previous << "] = P[" << index << "] =
" << p[index] << ", прибавляем значение на единицу" << std::endl;
            previous++;
        }
        std::cout << " p[" << index << "] = " << previous << std::endl;
        prefixes.push_back(previous);   //пушаем в вектор
    }
}

std::vector<int> KMPFunc(std::string &p){     //алгоритм
    std::vector<int> answer;    //индексы вхождений образца в тексте
    std::vector<int> prefixes;  //префикс-ф образца
    prefixes.push_back(0);  //пушаем значение для первого символа образца
    prefixFunc(p, prefixes);    //находим остальные значения
    std::cout << "Префикс-ф для образца равна [";
    printArray(prefixes);
    std::cout << "]" << std::endl;
    int indexT = 0;     //индекс символа в тексте
    int indexP = 0;     //индекс символа в образце и значение префикс-ф
    std::cout << "Начинаем поиск, посимвольно обрабатывая строку, indexP
= 0" << std::endl;
```

```
    char c;     //символ текста
    while (std::cin.get(c)){    //считывание символов текста
        if (c == '\n' || c == EOF) break;
        std::cout << " T[" << indexT << "] = " << c << ":" << std::endl;
        while (indexP > 0 && p[indexP] != c) { //аналогичный алгоритм
нахождения префикс-ф
            std::cout << "  Возвращаемся навзад, пока indexP не станет
равным 0 или P[indexP] станет равным T[" << indexT << "]" << std::endl;
            std::cout << "  indexP = p[indexP-1] = p[" << indexP-1 << "]
= " << prefixes[indexP-1] << std::endl;
            indexP = prefixes[indexP-1];
        }
        if (p[indexP] == c){
            std::cout << "  P[" << indexP << "] = T[" << indexT << "] = "
<< c << ", увеличиваем indexP на 1" << std::endl;
            indexP++;
        }
        if (indexP == p.size()){    //если значение префикс-ф равно длине
образца, то подстрока найдена
            std::cout << "  indexP = длине образца = " << p.size() << ",
подстрока найдена" << std::endl;
            std::cout << "  Индекс первого вхождения = indexT - длина
образца + 1 = " << indexT << "-" << p.size() << "+1" << std::endl;
            answer.push_back(indexT - p.size() + 1);    //индекс первого
вхождения
            indexP = prefixes[indexP-1];    //последний символ, для
которого выполняется идентичность подстроки и обзаца
            std::cout << "  Пушаем в массив индексов, indexP теперь равен
предыдущму значению p[" << indexP+1 << "-1] = " << indexP << std::endl;
        }
        indexT++;
    }
    return answer;
}

int main() {
    setlocale(LC_ALL, "RUS");
    //SetConsoleOutputCP(CP_UTF8);
    std::string p;  //образец
    std::cin >> p;
    std::cin.get();     //пропускаем enter
    std::vector<int> answer = KMPFunc(p);    //массив индексов вхождений
    if (answer.empty()){    //массив пустой значит вхождений нет
        std::cout << "-1";
    } else{ //печать в требуемом виде
        printArray(answer);
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ В

КОД ПРОГРАММЫ РЕШЕНИЯ ВТОРОЙ ЗАДАЧИ

```
#include <iostream>
#include <string>
#include <vector>
//#include <windows.h>

void printArray(std::vector<int>& arr){
    for (unsigned int i = 0; i < arr.size() - 1; i++) {
        std::cout << arr[i] << ',';
    }
    std::cout << arr[arr.size() - 1];
}

void prefixFunc(std::string b, std::vector<int>& prefixes){ //вычисление
префикс-ф для строки B
    std::cout << "Вычисление префикс-ф p для B = " << b << std::endl;
    std::cout << " p[0] = 0" << std::endl;
    for (unsigned int index = 1; index < b.size(); index++){    //проход
по каждому символу строки кроме первого
        std::cout << " p[" << index << "]: " << std::endl;
        int previous = prefixes[index-1]; //изначально равен значению
предыдущего символа
        std::cout << "  Предыдущее значение p[" << index-1 << "]=" <<
previous << std::endl;
        while (previous > 0 && b[previous] != b[index]){ //если
предыдущее положительно и не равно исходному символу...
            std::cout << "  Оно больше нуля и B[" << previous << "] !=
B[" << index << "] = " << b[index] << ", берем p[" << previous-1 << "]"
<< std::endl;
            previous = prefixes[previous-1];    //...берем еще назад
        }
        if (b[previous] == b[index]){ //если символы образца равны, то
прибавляем значение на единицу
            std::cout << "  B[" << previous << "] = B[" << index << "] =
" << b[index] << ", прибавляем значение на единицу" << std::endl;
            previous++;
        }
        std::cout << " p[" << index << "] = " << previous << std::endl;
        prefixes.push_back(previous);   //пушаем в вектор
    }
}

int KMPFunc(std::string &a, std::string &b){
    std::vector<int> prefixes;  //префикс-ф строки B
    if (a.size() != b.size()){   //если строки с разными длинами то они
не цикл сдвиг
        std::cout << "Строки не равны по длине" << std::endl;
        return -1;
    }
    a += a;     //удваиваем строку A
    std::cout << "Удваиваем строку A, A = " << a << std::endl;
    prefixes.push_back(0);  //значение для первого символа строки B
    prefixFunc(b, prefixes);    //вычисление остальных знач
    std::cout << "Префикс-ф для B равна [";
    printArray(prefixes);
```

```
    std::cout << "]" << std::endl;
    int indexB = 0;
    std::cout << "Начинаем поиск, посимвольно обрабатывая строку, indexB
= 0" << std::endl;
    for (int indexA = 0; indexA < a.size(); indexA++){  //проход по
удвоенной строке A
        std::cout << " A[" << indexA << "] = " << a[indexA] << ":" <<
std::endl;
        while (indexB > 0 && b[indexB] != a[indexA]){  //аналогично как
для нахождения значений префикс-ф
            std::cout << "  Возвращаемся навзад, пока indexB не станет
равным 0 или A[indexA] станет равным B[" << indexB << "]" << std::endl;
            std::cout << "  indexA = p[indexA-1] = p[" << indexA-1 << "]
= " << prefixes[indexA-1] << std::endl;
            indexB = prefixes[indexB-1];
        }
        if (b[indexB] == a[indexA]){
            std::cout << "  B[" << indexB << "] = A[" << indexA << "] = "
<< a[indexA] << ", увеличиваем indexB на 1" << std::endl;
            indexB++;
        }
        if (indexB == b.size()){    //подстрока найдена
            std::cout << "  indexB = длине образца = " << b.size() << ",
подстрока найдена" << std::endl;
            std::cout << "  Индекс первого вхождения = indexA - длина
образца + 1 = " << indexA << "-" << b.size() << "+1" << std::endl;
            return indexA - b.size() + 1;   //первое вхождение
        }
    }
    return -1;  //если вхождения не найдены то печатаем -1
}

int main() {
    setlocale(LC_ALL, "RUS");
    //SetConsoleOutputCP(CP_UTF8);
    std::string a;  //A
    std::string b;  //B
    std::cin >> a;
    std::cin >> b;
    std::cout << KMPFunc(a, b);
    return 0;
}
```