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. 8383 _____ Мололкин К.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы

Изучить работу алгоритма Кнута-Морриса-Пратта для нахождения подстроки в строке и решить поставленную задачу с помощью данного алгоритма.

Постановка задачи.

Вар. 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.

1) Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (|P|≤15000) и текста TT (|T ≤ 5000000) найдите все вхождения PP в TT.

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|≤5000000) и B (|B|≤5000000).

Определить, является ли AA 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

Описание алгоритма КМП

1. На вход алгоритм получает строку шаблон и строку текст.
2. В начале алгоритма выполняется вычисление префикс-функции для строки шаблона:
   2.1. Функция проходится по всем символам строки от 1 до n-1
   2.2. Для подсчета текущего значения функции для i-го элемента строки, заводим переменную j, обозначающую длину текущего рассматриваемого образца. Изначально j = значение функции для i – 1 символа.
   2.3. Затем в цикле пока j > 0 и i-ый символ входной строки не равен j-ому, к j приравниваем значение функции для j-1-го элемента.
   2.4. Если символы совпали, то инкрементируем j.
   2.5. Префикс функция i-го элемента равна j.
   2.6. Функция возвращает массив значений для каждого символа в строке.
3. Затем создаются две индексные переменные, одна для шаблона другая для текста.
4. Далее в цикле пока переменная индекса текста не равна длине текста:
   4.1. Если символы шаблона и текста по текущим значениям соответствующих индексных переменных совпадают, то увеличиваем индексные переменные.

4.2. Если после увеличения индекс шаблона равен его длине, то нашли вхождение, следовательно выводим индекс первого элемента подстроки вхождения.

4.3. Если символы не совпали, то к индексу шаблона приравниваем значение префикс-функции для предшествующего элемента шаблона.

Описание функций

void prefixFunction(const std::string& str, std::vector<int>& result) – префикс-функция:

Принимает на вход две переменные, std::string& str – ссылка на строку для расчёта префикс-функции, std::vector<int>& result – ссылка на вектор содержащий результат работы функции. Затем в цикле проходится от 1-го до n-1, где n – длина строки и будем считать значение префикс-функции для каждого элемента и записывать в результирующий вектор. Для подсчета текущего значения result[i], заводим переменную j = result[i-1], затем в цикле пока j > 0 и str[i] != str[j], изменяем значение j = result[j-1], при выходе из цикла, если str[i] == str[j], инкрементируем j. Затем записываем значение префикс функции для текущего элемента строки в результат.

void KMP(const std::string& pattern, const std::string& text) – функция осуществляющая реализацию алгоритма Кнута-Морисса-Пратта:

На вход принимает: std::string& pattern – ссылка на строку-шаблон для поиска, std::string& text ссылка на строку-текст в котором будет произведен поиск. Сначала функция создает вектор prefixFuncRes с длиной равной pattern.length(), а затем вызывает prefixFunction, передавая вектор и строку-шаблон. Далее создаются 3 переменные int textIndex = 0 – используется для итерации по тексту, int patternIndex = 0 – используется для итерации по шаблону, bool isAnyEntry – используется для того, чтобы определить есть ли вообще вхождения, а также вектор result, для хранения индексов вхождений шаблона. Далее пока textIndex != text.length(), если

pattern[patternIndex] == text[textIndex], увеличиваем обе индексные переменные, если patternIndex == pattern.size(), то меняем значение на isAnyEntry на true и добавляем в результирующий вектор textIndex - pattern.length(), если символы не совпадают и значения индекса шаблона равно нулю, то инекреминтируем переменную индекса текста, если индекс шаблона не ноль, то приравниваем его к значению prefixFuncRes[patternIndex-1]. Если не было найдено ни одного вхождения, то выводится “-1”. В противном случае выводим значения индексов входа шаблона в текст через запятую.

Сложность алгоритма по времени

Функция вычисления префикс-функции имеет сложность равную O(n), так как линейно проходится по переданной строке. Алгоритм поиска вхождения подстроки в строке так же выполняется линейно, имея сложность O(m), где m – длина входного текста. Следовательно сложность алгоритма по времени равна O(n+m).

Сложность алгоритма по памяти

Алгоритм хранит значения префикс-функции для строки шаблона. Следовательно сложность по памяти равна O(n).

Описание алгоритма поиска циклического сдвига

1. На вход алгоритм получает две строки.
2. Если длины строк не равны, то одна строка не может быть циклическим сдвигом другой.
3. Затем сравниваем если строки равны, то выводим 0 и заканчиваем алгоритм
4. Если два первых пункта не выполняются, то создается строка, которая является склейкой первой строки и двух вторых, вызываем префикс-функцию для полученной строки.

5. Если в результате возвращаемым префикс-функцией есть значение равное длине изначальной строки, и если индекс этого значения в результирующем векторе больше size*2 + 1, где size – длина входной строки, то следовательно вторая строка является циклическим сдвигом первой.

Описание функция алгоритма

void shift(std::string& str1, std::string& str2) – функция реализует поиск циклического сдвига.

На вход принимает две переменные: std::string& str1 – первая строка, std::string& str2 – втора строка. Затем сравнивает длины строк, если они разные то воводим -1, если нет, то сравниваем строки, если они равны, то выводим 0, если не равны то вызываем префикс-функция для строки состоящей из второй строки и двух первых. Затем в цикле проходимся по вектору значений префикс-функции для каждого элемента новой строки, если значение префикс функции совпадает с размером исходной строки и i – size*2 + 1 > 0, то значит нашли сдвиг выводим i - size*2 + 1, где i индекс элемента в векторе, а size – размер исходной строки. Если прошли весь вектор и не нашли сдвиг, то выводим -1.

void prefixFunction(std::string& str, std::vector<int>& result) – совпадает с описанием аналогичной функции для предыдущего алгоритма.

Сложность алгоритма по времени

Алгоритм имеет сложность O(n), где n – исходной строки, так как в худшем случае алгоритм 2 раза пройдется по склеенной строке.

Сложность алгоритма по памяти

По памяти алгоритм занимает O(n), так как хранится одна склеенная строка размера 3n, а также вектор содержащий 3n элементов.

Тестирование

Алгоритм КМР

```
ab
abab
Start prefix function for string: ab
result[0] = 0
Input string length: 2
Check symbol: b index in input string = 1
Current j = 0
Prefix function for symbol b index in input string = 1 is 0
End prefix function

Start KMP
Symbol on index 0 in pattern is equal to symbol on index 0 in text
Increase indexes
Symbol on index 1 in pattern is equal to symbol on index 1 in text
Increase indexes
Entry was found, the occurrence index 0
Symbols not equals
Change pattern index to 0
Symbol on index 0 in pattern is equal to symbol on index 2 in text
Increase indexes
Symbol on index 1 in pattern is equal to symbol on index 3 in text
Increase indexes
Entry was found, the occurrence index 2
End KMP print result:
0,2
```

	Input	Output
1	aa ajhjhsjhfaaflaakdfjaa	9, 13, 19
2	b abcdabcdbbdddb	1,5,8,9,13
3	aaaaaaaaaaa vvvvvvvvvvvv	-1
4	aaaa aaaaaaaa	0,1,2,3,4,5

Алгоритм поиска сдвига

```
abc
cab
Start searching shift
Strings not equals
Create new string: cababcabc
Call prefix function for new string: cababcabc

Start prefix function for string: cababcabc
result[0] = 0
Input string length: 9
Check symbol: a, index in input string = 1
Current j = 0
Prefix function for symbol a, index in input string = 1 is 0
Check symbol: b, index in input string = 2
Current j = 0
Prefix function for symbol b, index in input string = 2 is 0
Check symbol: a, index in input string = 3
Current j = 0
Prefix function for symbol a, index in input string = 3 is 0
Check symbol: b, index in input string = 4
Current j = 0
Prefix function for symbol b, index in input string = 4 is 0
Check symbol: c, index in input string = 5
Current j = 0
Sumbols at position j in input string and current checking symbol are equal, increase j, j = 1
Prefix function for symbol c, index in input string = 5 is 1
Check symbol: a, index in input string = 6
Current j = 1
Sumbols at position j in input string and current checking symbol are equal, increase j, j = 2
Prefix function for symbol a, index in input string = 6 is 2
Check symbol: b, index in input string = 7
Current j = 2
Sumbols at position j in input string and current checking symbol are equal, increase j, j = 3
Prefix function for symbol b, index in input string = 7 is 3
Check symbol: c, index in input string = 8
Current j = 3
Sumbols at position j in input string and current checking symbol not equal and j > 0, change j value to result[j - 1]: j =  0
Sumbols at position j in input string and current checking symbol are equal, increase j, j = 1
Prefix function for symbol c, index in input string = 8 is 1
End prefix function

Prefix function result:
0 0 0 0 0 1 2 3 1
Search in result vector: 3 with positive result index
Check value at index: 0 in prefix function result
Check value at index: 1 in prefix function result
Check value at index: 2 in prefix function result
Check value at index: 3 in prefix function result
Check value at index: 4 in prefix function result
Check value at index: 5 in prefix function result
```

	Input	Output
1	defabc abcdef	3
2	abc cab	2
3	aaaaaaaaaaa vvvvvvvvvvvv	-1
4	aaaaaa aaaaaa	0
5	vvvvvv aaaaaa	-1
6	bbabbb bbabba	-1

Вывод

В результате выполнения был изучены алгоритм Кнута-Морриса-Пратта, префикс-функция, написаны программа по поиску всех вхождений подстроки в строку, а также программа по определению циклического сдвига.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ KMP.cpp

```
#include <iostream>
#include <vector>

void prefixFunction(const std::string& str, std::vector<int>&
result) { //префикс функция
    std::cout << "Start prefix function for string: "<< str <<
std::endl;
    std::cout << "result[0] = 0" << std::endl;
    int n = str.length();
//получаем длину входной строки
    std::cout << "Input string length: " << n << std::endl;
    for (int i = 1; i < n; ++i){
        std::cout << "Check symbol: " << str[i] << " index in
input string = " << i << std::endl;
        int j = result[i - 1];
//приравниваем текущий j к значению результата префикс ффункции
для предыдущего символа
        std::cout << "Current j = " << j << std::endl;
        while(j > 0 && str[i] != str[j]) {
//если символы не совпадают и j больше 0 переходим к предыдущему
результирующем векторе
            j = result[j - 1];
            std::cout << "Sumbols at position j in input string
and current checking symbol not equal, change j value to result[j
- 1]: j =  " << j << std::endl;
        }
        if(str[i] == str[j]) { //если нашли совпадени увеличиваем
j
            ++j;
            std::cout << "Sumbols at position j in input string
and current checking symbol are equal, increase j, j = " << j <<
std::endl;
        }
        result[i] = j;
// записываем результат для тек. символа
        std::cout << "Prefix function for symbol " << str[i] << "
index in input string = " << i << " is " << result[i] <<
std::endl;
    }
    std::cout << "End prefix function \n" << std::endl;
}

void KMP(const std::string& pattern, const std::string& text) {
    std::vector<int> prefixFuncRes(pattern.size());
    std::vector<int> result;
    prefixFunction(pattern, prefixFuncRes);
//вычисляем префикс функцию для шаблона
    int textIndex = 0;
    int patternIndex = 0;
```

```
    bool isAnyEntry = false;
    std::cout << "Start KMP" << std::endl;
    while(textIndex != text.length()){
//запускаем цикл пока не пройдем весь текст
        if(pattern[patternIndex] == text[textIndex]) {
//если нашли совпадение символов шаблона и текста
            std::cout << "Symbol on index " << patternIndex << "
in pattern is equal to symbol on index " << textIndex << " in
text" << std::endl;
            std::cout << "Increase indexes" << std::endl;
            textIndex++;
            patternIndex++;
            if(patternIndex == pattern.size()) {
//если прошли весь шаблон значит нашли вхождение
                isAnyEntry = true;
                result.push_back(textIndex - pattern.length());
//индекс вхождения записываем в рез. вектор
                std::cout << "Entry was found, the occurrence
index " << textIndex - pattern.length() << std::endl;
            }
        }
        else {
            std::cout << "Symbols not equals" << std::endl;
            if(patternIndex == 0){
//символы не совпали и индекс шаблона 0
                std::cout << "Pattern index is 0, increase text
index" << std::endl;
                textIndex++;
            }
            else {
//индекс шаблона не 0
                patternIndex = prefixFuncRes[patternIndex - 1];
//переходим к предидущему символу в строке значений префикс
функции
                std::cout << "Change pattern index to " <<
patternIndex << std::endl;
            }
        }
    }
    std::cout << "End KMP print result: " << std::endl;
    if(!isAnyEntry) std::cout << -1;
//если не нашил вхождений
    else {
// вывод результата
        std::cout << result[0];
        for(int i = 1; i < result.size(); i++){
            std::cout << "," << result[i];
        }
    }
}

int main() {
    std::string pattern;
```

```
    std::string text;
    std::cin >> pattern >> text;
    KMP(pattern, text);
    std::cout << std::endl;
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

КОД ПРОГРАММЫ shift.cpp

```
#include <iostream>
#include <vector>

void prefixFunction(const std::string& str, std::vector<int>&
result) { //префикс функция
    std::cout << "Start prefix function for string: "<< str <<
std::endl;
    std::cout << "result[0] = 0" << std::endl;
    int n = str.length();
//получаем длину входной строки
    std::cout << "Input string length: " << n << std::endl;
    for (int i = 1; i < n; ++i){
        std::cout << "Check symbol: " << str[i] << ", index in
input string = " << i << std::endl;
        int j = result[i - 1];
//приравниваем текущий j к значению результата префикс ффункции
для предыдущего символа
        std::cout << "Current j = " << j << std::endl;
        while(j > 0 && str[i] != str[j]) {
//если символы не совпадают и j больше 0 переходим к предыдущему
результирующем векторе
            j = result[j - 1];
            std::cout << "Sumbols at position j in input string
and current checking symbol not equal and j > 0, change j value to
result[j - 1]: j =  " << j << std::endl;
        }
        if(str[i] == str[j]) { //если нашли совпадени увеличиваем
j
            ++j;
            std::cout << "Sumbols at position j in input string
and current checking symbol are equal, increase j, j = " << j <<
std::endl;
        }
        result[i] = j;
// записываем результат для тек. символа
        std::cout << "Prefix function for symbol " << str[i] << ",
index in input string = " << i << " is " << result[i] <<
std::endl;
    }
    std::cout << "End prefix function \n" << std::endl;
}

void shift(std::string& str1, std::string& str2){
    std::cout << "Start searching shift" << std::endl;
    if(str1.length() != str2.length()) {
//если длина строк разная
        std::cout << "Strings lengths not equals" << std::endl;
        std::cout << -1;
    }
```

```
    else if (str1 == str2) {
//если строки полностью совпадают
        std::cout << "Strings are equals" << std::endl;
        std::cout << "Result: ";
        std::cout << 0;
    }
    else {
//если длина одинаковая но строки не совпадают
        std::cout << "Strings not equals" << std::endl;
        int size = str1.size();
        str2 += str1;
//создаем новую строку состоящую из второй и двух первых
        str2 += str1;
        str1.clear();
//очищаем первую строку
        std::cout << "Create new string: " << str2 << std::endl;
        std::vector<int> result(str2.size());
        std::cout << "Call prefix function for new string: "<<
str2 << "\n" << std::endl;
        prefixFunction(str2, result);
//вызываем префикс функцию для новой строки
        std::cout << "Prefix function result: " << std::endl;
        for(int i = 0; i < result.size(); i ++){
            std::cout << result[i] << " ";
        }
        std::cout << std::endl;
        str2.clear();
//очищаем ее так как работать будем только с результатом
        bool isShift = false;
//флаг для выхода из цикла
        std::cout << "Search in result vector: " << size << " with
positive result index"<< std::endl;
        for(int i = 0; i < result.size(); i ++){
            std::cout << "Check value at index: " << i << " in
prefix function result" << std::endl;
            if(result[i] == size) {
//нашли сдвиг
                if(i - size*2 + 1 >= 0){
                    std::cout << "Shift was found, result index:
";
                    isShift = true;
                    std::cout << i - size*2 + 1;
//печааем индекс
                    break;
                }
                else std::cout << size << " was found in vector,
but result not positive" << std::endl;
            }
        }
        if(!isShift) {
//если строка 2 не является циклическим сдвигом
            std::cout << "Shift wasn't found, result -1" <<
std::endl;
```

```
            }
        }
    }

    int main() {
        std::string str1;
        std::string str2;
        std::cin >> str1 >> str2;
        shift(str1, str2);
        std::cout << std::endl;
        return 0;
    }
```