# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МОЭВМ**

# ОТЧЕТ

**по лабораторной работе №4**

# по дисциплине «Построение и анализ алгоритмов» Тема: Алгоритм Кнута-Морриса-Пратта

|  |  |  |
| --- | --- | --- |
| Студент гр. 9382 |  | Иерусалимов Н. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург 2021

# Цель работы.

Познакомиться с одним из часто используемых на практике алгоритма поиска подстроки в строке. Получить навыки решения задач на этот алгоритм.

# Задание.

**Первое задание:**

Реализуйте алгоритм КМП и с его помощью для заданных шаблона *P* (|𝑃| ≤ 15000) и текста 𝑇 (|𝑇| ≤ 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

# Второе задание:

Заданы две строки *A* (|𝐴| ≤ 5000000)и *B* (|𝐵| ≤ 5000000).

Определить, является ли А циклическим сдвигом В (это значит, что А и В имеют одинаковую длину и А состоит из суффикса В, склеенного с префиксом В).

Например, defabc является циклическим сдвигом abcdef. Вход:

Первая строка - *A* Вторая строка - *B* Выход:

Если *A* вляется 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 алгоритмов.

Алгоритм позволяет не рассматривать подстроки, начинающиеся с каждого символа в строке. Сначала мы считаем префикс-функцию для нашей подстроки. Значение префикс-функции означает длину наибольшего совпадения префикса и суффикса в подстроке шаблона, которая рассматривается. Всего значений будет N, где первое значение – 0, так как размер подстроки равен единице, соответственно максимальный размер префикса и суффикса этой подстроки равен единице. Если символ текста и символ шаблона равен, то рассматриваются следующие символы. Если этот символ был последним символом строки-шаблона, то было найдено вхождение шаблона в тексте, индекс вхождения записывается в результат. Индекс строки-шаблона в этом случае становится значением префикс-функции под предыдущим значением индекса строки-шаблона.

Если рассматриваемые символы не равны, и рассматриваемый символ строки-шаблона был начальным, то сдвигается индекс символа, который рассматривается в тексте на единицу. Если же символ был не начальным, то индекс символа в строке-шаблоне становится равен значению префикс-функции предыдущего индекса.

**Сложность:**

**O(n + k)** – эта сложность получается из логики что за O(k), где k это длина подстроки, строится массив префикс-функций. А за O(n), где n это длина текста, работает сам алгоритм, т.к. мы проходимся по тексту один раз, не возвращаясь уже к пройденным элементам.

По памяти сложность будет **O(k + n)** n – длина текста, k – длина подстроки

# Описание функций и структур данных.

std::vector<int> PrefixAndSuffix(std::string pattern) – функция, отвечающая за построение массива префикс-функций.

std∷string pattern –подстрока

Возвращает:

std::vector<int> - вектор префикс функции.

std::vector<int> Calculate(std::string pattern, std::string text) – функция основного алгоритма.

std∷string pattern –подстрока

std∷string text – строка где искать

Возвращает:

std∷vector<int> - вектор индексов, возвращаем индексы где были найдены подстроки

# Тестирование.

Тестирование жадного алгоритма.

Задание 1:

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Выходные данные |
| 1 | asdasddsaasd | 0,6 |
| 2 | vfdvcvdfcvssxcvsd | -1 |
| 3 | FAFAFAFAFAFAFAFAFAFAFfdasfwev | 0,2,4,6,8 |
| 4 | ILovePiaaSegmentationFaultILovePiaa | 17 |

# Задание 2:

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Выходные данные |
| 1 | defabcabcdef | 3 |
| 2 | acdefgacdefg | 0 |
| 3 | acddca | -1 |
| 4 | lovevelo | 2 |

# Выводы.

Был исследован часто используемый на практике алгоритм - поиск подстроки в строке. Также были получены навыки решения задач на этот алгоритм.

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Имя файла: main.cpp

#include <iostream>

#include <vector>

bool choiseInfo = 0;

int choiseTask = 1;

class KMP {

public:

void Display() {

if (choiseInfo) {

std::cout << info;

}

info = "";

}

std::string PrintAnimAnswer(int j, int i, std::string pattern, std::string txt) {

std::string res = "";

for (int k = 0; k < i; ++k) {

res += " ";

}

res += "i\n";

for (auto& k : txt) {

res += std::string(1, k) + " ";

}

res += '\n';

for (int k = 0; k < j; ++k) {

res += " ";

}

res += "j\n";

for (auto& k : pattern) {

res += std::string(1, k) + " ";

}

res += "\n\n";

return res;

}

std::vector<int> Calculate(std::string pattern, std::string txt) {

std::vector<int> pi;

std::vector<int> check;

pi = PrefixAndSuffix(pattern);//Подсчитываем префикс-функцию

int i = 0, j = 0;//Заводим два индекса для прохода по основной строке и по паттерну соответственно.

info += "Start index: \n" + PrintAnimAnswer(j, i, pattern, txt);

Display();

while (i < txt.length()) {

//если символы поставили то инкрементируем индексы и переходим на следю символ

if (pattern[j] == txt[i]) {

info += "j = i: increment both\n";

++j;

++i;

}

//Индекс совпадает с длиной подстроки, значит мы дошли до конца и было совпадение

//Записываем индекс начала найденной строки.

if (j == pattern.length()) {

info += "j = template length, match found, add index to result.\n"

"j move to the beginning of the pattern\n";

check.push\_back(i - j);

j = pi[j - 1];

}

else if (i < txt.length() && pattern[j] != txt[i]) {

info += "j != i: ";

//если не первый элемент в паттерне, то возвращаемся на элемент, следующий за префиксом

if (j != 0) {

info += "j != 0 increment j\n";

j = pi[j - 1];

}

//иначе рассматриваем следующий элемент в тексте

else {

info += "j = 0 increment i, look at the next elem\n";

i++;

}

}

info += "New index: \n";

info += PrintAnimAnswer(j, i, pattern, txt);

Display();

}

info += "All text is cheked, end algorithm\n";

Display();

return check;

}

void Task1(std::string pattern, std::string text) {

result = Calculate(pattern, text);

if (result.empty()) {

std::cout << "-1";

}

else {

for (int i = 0; i < result.size(); i++) {

std::cout << result[i];

if (i != result.size() - 1)

std::cout << ",";

}

}

}

void Task2(std::string pattern, std::string text) {

if (pattern.length() != text.length()) {

std::cout << "-1";

return;

}

result = Calculate(text, pattern + pattern);

if (result.empty()) {

std::cout << "-1";

}

else {

std::cout << result[0];

}

}

private:

std::string printPrefixFunc(int j, int i, std::string pattern, std::vector<int> pi) {

std::string res = "";

for (int k = 0; k < j; ++k) {

res += " ";

}

res += "j ";

for (int k = 0; k < i - j - 1; ++k) {

res += " ";

}

res += "i \n";

for (auto& k : pattern) {

res += std::string(1, k) + " ";

}

res += '\n';

for (auto& k : pi) {

res += std::string(1, k) + " ";

}

res += " - prefix func\n";

return res;

}

std::vector<int> PrefixAndSuffix(std::string pattern) {

std::vector<int> pi(pattern.length());//Сразу инициализируем длину префикс-функции

pi[0] = 0;//Первый эллемент в массиве 0

int j = 0;

int i = 1;

info += "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Calculate Prefix func\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nInitial state of the pattern\n" +

printPrefixFunc(j, i, pattern, pi);

Display();

while (i < pattern.length()) {

//если два текущих элемента в паттерне одинаковы

//то запоминаем, что при несовпадении на элементе i + 1, следует перейти на элемент с индексом j

if (pattern[i] == pattern[j]) {

info += "Elements with indices i and j are the same, remember j and advance both indices\n";

j++;

pi[i] = j;

i++;

}

else {

info += "Elements with indices i and j are different\n";

if (j != 0) {

info += "j is not zero, change j to the element with index j-1 in the prefix function array\n";

j = pi[j - 1];

}

else {

info += "j - zero, write zero to the element with index i and move i to the right\n";

pi[i] = 0;

i++;

}

}

info += printPrefixFunc(j, i, pattern, pi) + "\n";

Display();

}

info += "Index i has reached the end of the pattern string, we finish building the prefix function array.\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

Display();

return pi;

}

std::vector<int> result;

std::string info;

};

int main() {

std::string pattern, txt;

std::cout << "Display intermediate data?\n Yes - 1 : No - 0\n";

std::cin >> choiseInfo;

std::cout << "Select task number, 1 or 2\n";

std::cin >> choiseTask;

std::cout << "Enter data\n";

std::cin >> pattern >> txt;

KMP toDo;

if (choiseTask) {

toDo.Task1(pattern, txt);

}

else {

toDo.Task2(pattern, txt);

}

system("pause>nul");

return 0;

}