**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: Кнут-Моррис-Пратт

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Коняева М. В. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2025

**Цель работы.**

Изучить и реализовать алгоритм Кнута-Морриса-Пратта.

**Задание.**

**4.1**: 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

**4.2**: 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 алгоритма.**

Алгоритм Кнута-Морриса-Пратта (KMP) предназначен для поиска подстроки (шаблона) в строке. Он состоит из двух этапов.

Функция computeLPS вычисляет массив lps (Longest Prefix Suffix), который используется в алгоритме. Этот массив хранит для каждой позиции i в шаблоне pattern длину наибольшего собственного префикса подстроки pattern[0..i], который одновременно является суффиксом. Сначала создается вектор lps размером m, где m — длина шаблона, и он заполняется нулями. Далее вводятся две переменные: i, начинающаяся с 1, и j, изначально равная 0. Переменная i отвечает за текущий индекс в pattern, а j отслеживает длину совпадающего префикса-суффикса.

Алгоритм нахождения работает следующим образом: пока i меньше m, если pattern[i] совпадает с pattern[j], увеличивается j, значение lps[i] устанавливается равным j, и i увеличивается.

Если же pattern[i] не совпадает с pattern[j], то возможны два случая. Если j > 0, то j уменьшается до lps[j - 1], чтобы попытаться найти более короткий префикс-суффикс, который мог бы продолжить совпадение. В противном случае, если j == 0, устанавливается lps[i] = 0, и i просто увеличивается, так как совпадение отсутствует.

Этот процесс продолжается, пока i не достигнет конца строки pattern, после чего функция возвращает массив lps, который затем используется в основном алгоритме поиска Кнута-Морриса-Пратта.

Функция searchKMP, которая выполняет основной алгоритм, начинается с вычисления массива lps с помощью функции computeLPS, которая была описана выше.

После этого создаются переменные: вектор result для хранения индексов всех вхождений шаблона в строку, а также переменные i и j, которые служат индексами для строки и шаблона соответственно.

Алгоритм затем проходит по строке с помощью цикла. Если символы в строке и шаблоне совпадают, оба индекса увеличиваются. Когда j достигает длины шаблона, это означает, что найдено полное вхождение шаблона в строку, и его индекс сохраняется в вектор result. Далее значение j обновляется с помощью массива lps, что позволяет продолжить поиск, не начиная его с самого начала шаблона.

Если символы не совпадают, то с помощью массива lps индекс j сдвигается, и продолжается поиск с другой позиции, избегая повторных сравнений. Если j равно нулю, то просто увеличивается индекс i, чтобы проверить следующий символ строки.

Когда весь текст проверен, функция возвращает вектор result, который содержит индексы всех вхождений шаблона в строку. Этот алгоритм значительно ускоряет поиск, так как он использует информацию о предыдущих совпадениях и избегает повторных проверок символов.

Алгоритм Кнута-Морриса-Пратта (searchKMP) имеет линейную сложность как по времени, так и по памяти. Его общая сложность составляет O(n+m), где n— длина строки text, а m — длина шаблона pattern. Для вычисления массива lps (Longest Prefix Suffix) для шаблона алгоритм проходит по всем его символам, начиная с индекса 1 и заканчивая на m−1. Каждый символ обрабатывается один раз, и за каждый шаг обновляется массив lps. Таким образом, время работы функции computeLPS составляет O(m). Основной цикл поиска по строке text также выполняется за время O(n). В этом цикле каждый символ строки проверяется один раз. Если символы строки и шаблона совпадают, индексы увеличиваются, и если весь шаблон найден, индекс шаблона обновляется с использованием массива lps. Если символы не совпадают, индекс шаблона сдвигается с использованием информации из массива lps. Это позволяет избежать лишних сравнений и сдвигов.

Таким образом, сложность основного цикла также составляет O(n), где n — длина строки. Общая временная сложность алгоритма составляет O(n+m), где n — длина строки, а m — длина шаблона.

Время работы алгоритма searchКМП состоит из двух частей: во-первых, вычисление префиксной функции для паттерна требует O(m) времени, где m — длина паттерна, а во-вторых, сам поиск вхождений с использованием этой функции требует O(n) времени, где n — длина текста. Таким образом, общее время работы алгоритма составляет O(m+n), что является линейным временем относительно размеров паттерна и текста. Память для работы алгоритма используется следующим образом: требуется массив для хранения префиксной функции (O(m)). Этот алгоритм позволяет существенно ускорить процесс поиска подстроки, используя информацию о совпадениях в начале и конце строки, минимизируя количество повторных сравнений.

Для нахождения циклических сдвигов строки необходимо немного адаптировать алгоритм Кнута-Морриса-Пратта. В отличие от обычного поиска подстроки, в данном случае строка str2 удваивается, что дает возможность искать все возможные циклические сдвиги строки. Если шаблон найден, возвращается индекс первого вхождения. Если нет, возвращается -1.

**Описание функций и структур данных.**

1. std::vector<int> computeLPS (const std::string &pattern): принимает pattern - строку-паттерн и возвращает std::vector<int> - вектор целых чисел, который представляет массив наибольших префиксов, совпадающих с суффиксами для каждой подстроки.
2. std::vector<int> searchKMP(const std::string &text, const std::string &pattern): принимает две строки const std::string &text — текст и const std::string &pattern — паттерн; возвращает вектор индексов, на которых начинается каждое вхождение паттерна в тексте. Если вхождений нет, возвращается пустой вектор. Использует массив LPS, вычисленный в предыдущей функции, для поиска всех вхождений паттерна в текст.

Для циклического сдвига:

1. int searchKMP(const std::string& text, const std::string& pattern): принимает две строки const std::string& text — текст и const std::string& pattern **—** паттерн. Выполняет поиск первого вхождения паттерна в текст с помощью алгоритма Кнута-Морриса-Пратта. Она сначала вычисляет массив наибольших префиксов, совпадающих с суффиксами для паттерна с помощью функции computeLPS. Затем с помощью этого массива она пытается найти первое вхождение паттерна в тексте. Если вхождение найдено, функция возвращает индекс начала вхождения паттерна в тексте. Если паттерн не найден, возвращается -1.
2. int check(const std::string& str1, const std::string& str2): принимает две строки — str1 и str2. Она проверяет, что строки имеют одинаковую длину и не пустые. Затем она создает строку, которая является конкатенацией строки str2 самой с собой (str2 + str2), и передает эту строку и строку str1 в функцию searchKMP. Это позволяет проверить, является ли строка str1 циклическим сдвигом строки str2. Если циклический сдвиг найден, возвращается индекс первого вхождения, иначе — -1.

**Тестирование.**

Результаты тестирования приведены в Приложении Б.

**Выводы.**

В результате выполнения работы реализована программа на языке python, решающая поставленные задачи с использованием алгоритма Кнута-Морриса Пратта. Программа была протестирована.

**Приложение А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Имя файла: lb4\_comments2.cpp

#include <iostream>

#include <vector>

// Функция для вычисления массива LPS (Longest Prefix Suffix)

std::vector<int> computeLPS(const std::string &pattern) {

int m = pattern.size(); // Длина шаблона

std::vector<int> lps(m, 0); // Вектор для хранения значений LPS

int i = 1, j = 0; // i - текущий индекс в шаблоне, j - длина предыдущего наибольшего префикса

std::cout << "\n=== Вычисление LPS массива для шаблона: " << pattern << " ===\n";

std::cout << "lps[0" << "] = 0\n";

while (i < m) {

if (pattern[i] == pattern[j]) { // Если символы совпадают

j++; // Увеличиваем длину префикса

lps[i] = j; // Записываем значение в LPS-массив

std::cout << "Совпадение: pattern[" << i << "] == pattern[" << j-1 << "] (" << pattern[i] << ") -> lps[" << i << "] = " << j << "\n";

i++; // Переходим к следующему символу

} else {

if (j > 0) {

std::cout << "Несовпадение: pattern[" << i << "] != pattern[" << j << "] (" << pattern[i] << " != " << pattern[j] << ") -> j = lps[" << j-1 << "] = " << lps[j-1] << "\n";

j = lps[j - 1]; // Перемещаем j по LPS-массиву

} else {

lps[i] = 0; // Если нет совпадений, записываем 0

std::cout << "Несовпадение: pattern[" << i << "] != pattern[" << j << "] (" << pattern[i] << " != " << pattern[j] << ") -> lps[" << i << "] = 0\n";

i++; // Двигаем i дальше

}

}

}

std::cout << "LPS массив: ";

for (int val : lps) std::cout << val << " ";

std::cout << "\n";

return lps;

}

// Функция поиска подстроки в строке с помощью алгоритма Кнута-Морриса-Пратта

std::vector<int> searchKMP(const std::string &text, const std::string &pattern) {

std::vector<int> result;

std::vector<int> lps = computeLPS(pattern);

int n = text.size(), m = pattern.size();

int i = 0, j = 0;

std::cout << "\n=== Начало поиска (searchKMP) ===\n";

while (i < n) {

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

j++;

i++;

std::cout << "i = " << i << ", j = " << j

<< ", text[0:" << i << "] = \"" << text.substr(0, i)

<< "\", pattern[0:" << j << "] = \"" << pattern.substr(0, j) << "\"\n";

if (j == m) {

std::cout << "Найдено совпадение в позиции: " << (i - m) << "\n";

result.push\_back(i - m);

j = lps[j - 1];

}

} else {

if (j > 0) {

std::cout << "Несовпадение: text[" << i << "] = " << text[i]

<< " и pattern[" << j << "] = " << pattern[j] << "\n";

std::cout << "Откат к lps[j" << "-1] = lps["<< j << "-1] = "<< "Теперь j = " << lps[j - 1] << "\n";

j = lps[j - 1];

} else {

i++;

}

}

}

if (result.empty()) {

std::cout << "Совпадений не найдено.\n";

}

return result;

}

int main() {

std::string pattern, text;

std::cout << "Введите шаблон: ";

std::cin >> pattern;

std::cout << "Введите текст: ";

std::cin >> text;

std::vector<int> result = searchKMP(text, pattern); // Выполняем поиск searchKMP

std::cout << "\n=== Результат поиска: ===\n";

if (result.empty()) {

std::cout << "Шаблон не найден (-1)\n";

} else {

std::cout << "Шаблон найден на позициях: ";

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

if (i > 0) std::cout << ", ";

std::cout << result[i];

}

std::cout << "\n";

}

return 0;

}

Имя файла: task2\_comments.cpp  
#include <iostream>

#include <vector>

// Функция для вычисления массива LPS (Longest Prefix Suffix)

std::vector<int> computeLPS(const std::string &pattern) {

int m = pattern.size(); // Длина шаблона

std::vector<int> lps(m, 0); // Вектор для хранения значений LPS

int i = 1, j = 0; // i - текущий индекс в шаблоне, j - длина предыдущего наибольшего префикса

std::cout << "\n=== Вычисление LPS массива для шаблона: " << pattern << " ===\n";

std::cout << "lps[0] = 0\n";

while (i < m) {

if (pattern[i] == pattern[j]) { // Если символы совпадают

j++; // Увеличиваем длину префикса

lps[i] = j; // Записываем значение в LPS-массив

std::cout << "Совпадение: pattern[" << i << "] == pattern[" << j-1 << "] (" << pattern[i] << ") -> lps[" << i << "] = " << j << "\n";

i++; // Переходим к следующему символу

} else {

if (j > 0) {

std::cout << "Несовпадение: pattern[" << i << "] != pattern[" << j << "] (" << pattern[i] << " != " << pattern[j] << ") -> j = lps[" << j-1 << "] = " << lps[j-1] << "\n";

j = lps[j - 1]; // Перемещаем j по LPS-массиву

} else {

lps[i] = 0; // Если нет совпадений, записываем 0

std::cout << "Несовпадение: pattern[" << i << "] != pattern[" << j << "] (" << pattern[i] << " != " << pattern[j] << ") -> lps[" << i << "] = 0\n";

i++; // Двигаем i дальше

}

}

}

std::cout << "LPS массив: ";

for (int val : lps) std::cout << val << " ";

std::cout << "\n";

return lps;

}

// Функция для поиска подстроки (pattern) в строке (text) с использованием алгоритма searchKMP

int searchKMP(const std::string& text, const std::string& pattern) {

std::vector<int> lps = computeLPS(pattern); // Получаем массив LPS для паттерна

int n = text.size(), m = pattern.size(); // Длины строк

int i = 0, j = 0; // Индексы для прохода по тексту и паттерну

std::cout << "\n=== Начало поиска (searchKMP) ===\n";

// Проходим по тексту

while (i < n) {

// Если символы совпадают, увеличиваем индексы i и j

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

i++, j++;

std::cout << "i = " << i << ", j = " << j

<< ", text[0:" << i << "] = \"" << text.substr(0, i)

<< "\", pattern[0:" << j << "] = \"" << pattern.substr(0, j) << "\"\n";

if (j == m) return i - m; // Если все символы совпали, возвращаем индекс начала совпадения

} else {

if (j > 0) {

std::cout << "Несовпадение: text[" << i << "] = " << text[i]

<< " и pattern[" << j << "] = " << pattern[j] << "\n";

std::cout << "Откат к lps[j" << "-1] = lps["<< j << "-1] = "<< "Теперь j = " << lps[j - 1] << "\n";

j = lps[j - 1];

} else {

i++;

}

}

}

return -1; // Если совпадения не найдено, возвращаем -1

}

// Функция для проверки, является ли строка str1 вращением строки str2

int check(const std::string& str1, const std::string& str2) {

// Если строки имеют разные длины или одна из строк пуста, сразу возвращаем -1

if (str1.size() != str2.size() || str1.empty()) {

std::cout << "\n=== Проверка на вращение ===\n";

std::cout << "Строки не могут быть вращением: различие в длине или одна из строк пуста.\n";

return -1;

}

// Строка str2 удваивается, чтобы покрыть все возможные вращения

std::string str = str2 + str2;

std::cout << "\n=== Проверка на вращение ===\n";

std::cout << "Удвоенная строка str2: " << str << "\n";

// Используем алгоритм searchKMP для поиска str1 в удвоенной строке str2

int result = searchKMP(str, str1);

// Выводим результат проверки вращения

if (result == -1) {

std::cout << "Шаблон (str1) не найден в удвоенной строке (str2 + str2). Это означает, что str1 не является вращением str2.\n";

} else {

std::cout << "Шаблон найден на позиции: " << result << " в удвоенной строке. Это означает, что str1 является вращением str2.\n";

}

return result;

}

int main() {

std::string str1, str2;

std::cin >> str1 >> str2; // Ввод двух строк

std::cout << "Проверка на вращение: ";

std::cout << "str1 = " << str1 << ", str2 = " << str2 << "\n";

int result = check(str2, str1); // Проверка, является ли str1 вращением str2

std::cout << "\n=== Результат проверки вращения: ===\n";

// Выводим результат

if (result == -1) {

std::cout << "Шаблон не найден (-1)\n";

} else {

std::cout << "Шаблон найден на позиции: " << result << "\n";

}

return 0;

}

**ПРИЛОЖЕНИЕ Б**

**ТЕСТИРОВАНИЕ**

Таблица 1 – Результаты тестирования 1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарий |
| 1 | dfdf  dfdf dfgd dsxc sz | 0 | Верно |
| 2 | df  s | -1 | Верно |
| 3 | dfsd  dfhshbd | -1 | Верно |
| 4 | sdsdsdsd  dsdsdsdsdsds | 1,3 | Верно |
| 5 | f  dfdfdfdfdfdf | 1,3,5,7,9,11 | Верно |

Таблица 2 - Результаты тестирования 2

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарий |
| 1 | ds  dfhh | -1 | Верно |
| 2 | dfgh  ghdf | 2 | Верно |
| 3 | dfgd  cvbb | -1 | Верно |
| 4 | asdfghjkl  asdfghjkl | 0 | Верно |
| 5 | asdfghjkl  lasdfghjk | 8 | Верно |

Вывод в режиме откладки:

Задание 1:  
Введите шаблон: aba

Введите текст: ababa

=== Вычисление LPS массива для шаблона: aba ===

lps[0] = 0

Несовпадение: pattern[1] != pattern[0] (b != a) -> lps[1] = 0

Совпадение: pattern[2] == pattern[0] (a) -> lps[2] = 1

LPS массив: 0 0 1

=== Начало поиска (searchKMP) ===

i = 1, j = 1, text[0:1] = "a", pattern[0:1] = "a"

i = 2, j = 2, text[0:2] = "ab", pattern[0:2] = "ab"

i = 3, j = 3, text[0:3] = "aba", pattern[0:3] = "aba"

Найдено совпадение в позиции: 0

i = 4, j = 2, text[0:4] = "abab", pattern[0:2] = "ab"

i = 5, j = 3, text[0:5] = "ababa", pattern[0:3] = "aba"

Найдено совпадение в позиции: 2

=== Результат поиска: ===

Шаблон найден на позициях: 0, 2

Задание 2:  
abcdef

defabc

Проверка на вращение: str1 = abcdef, str2 = defabc

=== Проверка на вращение ===

Удвоенная строка str2: abcdefabcdef

=== Вычисление LPS массива для шаблона: defabc ===

lps[0] = 0

Несовпадение: pattern[1] != pattern[0] (e != d) -> lps[1] = 0

Несовпадение: pattern[2] != pattern[0] (f != d) -> lps[2] = 0

Несовпадение: pattern[3] != pattern[0] (a != d) -> lps[3] = 0

Несовпадение: pattern[4] != pattern[0] (b != d) -> lps[4] = 0

Несовпадение: pattern[5] != pattern[0] (c != d) -> lps[5] = 0

LPS массив: 0 0 0 0 0 0

=== Начало поиска (searchKMP) ===

i = 4, j = 1, text[0:4] = "abcd", pattern[0:1] = "d"

i = 5, j = 2, text[0:5] = "abcde", pattern[0:2] = "de"

i = 6, j = 3, text[0:6] = "abcdef", pattern[0:3] = "def"

i = 7, j = 4, text[0:7] = "abcdefa", pattern[0:4] = "defa"

i = 8, j = 5, text[0:8] = "abcdefab", pattern[0:5] = "defab"

i = 9, j = 6, text[0:9] = "abcdefabc", pattern[0:6] = "defabc"

Шаблон найден на позиции: 3 в удвоенной строке. Это означает, что str1 является вращением str2.

=== Результат проверки вращения: ===

Шаблон найден на позиции: 3