**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 KMP, 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 (KMP) 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)O(n)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. Общая сложность по памяти составляет также O(n+m).

Время работы алгоритма КМП 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 два массива, один для хранения префиксной функции (O(m)) и второй для хранения результатов поиска (O(n) в худшем случае). Второй массив используется только для первого задания Таким образом, общее потребление памяти составляет O(m+n). 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 строку-паттерн и возвращает вектор целых чисел, который представляет массив наибольших префиксов, совпадающих с суффиксами для каждой подстроки.
2. std::vector<int> KMP(const std::string &text, const std::string &pattern): принимает две строки — текст и паттерн — и 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 KMP(const std::string& text, const std::string& pattern): принимает две строки — текст и паттерн — и 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 KMP. Это позволяет проверить, является ли строка str1 циклическим сдвигом строки str2. Если циклический сдвиг найден, возвращается индекс первого вхождения, иначе — -1.

**Способ хранения решений.**

В обоих заданиях исходные текст и паттерн хранятся в строковых переменных, а результат работы префикс-функции сохраняется в векторе. В языке C++ как строки, так и векторы представляют собой динамически выделенные массивы. Код программы приведен в Приложении А.

**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: lab4\_1.cpp

#include <iostream>

#include <vector>

#include <string>

// Функция для вычисления префикс-функции

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

int n = pattern.size();

std::vector<int> p(n, 0); // Массив для хранения значений префикс-функции

int j = 0; // Длина текущего наибольшего префикса, который является суффиксом

std::cout << "\n=== Вычисление префикс-функции ===\n";

for (int i = 1; i < n; i++) { // начинаем с 1, так как p[0] всегда 0

// Пока есть несоответствие, откатываемся с помощью префикс-функции

while (j > 0 && pattern[i] != pattern[j]) {

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

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

j = p[j - 1]; // Откатываемся к предыдущему совпадению

std::cout << "Назад к p[" << j << "]\n";

}

// Если символы совпадают, увеличиваем j

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

j++;

}

p[i] = j; // Записываем значение префикс-функции для i-го символа

// Вывод текущего состояния массива префикс-функции

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

<< ", pattern[0:" << i + 1 << "] = \"" << pattern.substr(0, i + 1)

<< "\", p[" << i << "] = " << p[i] << "\n";

}

return p;

}

// Функция поиска подстроки с использованием алгоритма Кнута-Морриса-Пратта

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

int n = text.size();

int m = pattern.size();

std::vector<int> positions; // Вектор для хранения позиций вхождений

int j = 0;

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

// Проходим по строке text, при этом ищем совпадения с pattern

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

// Пока символы не совпадают, откатываемся по префикс-функции

while (j > 0 && text[i] != pattern[j]) {

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

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

j = p[j - 1]; // Возвращаемся к предыдущему возможному совпадению

std::cout << "Откат к p[" << j << "]\n";

}

// Если символы совпали, увеличиваем j

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

j++;

}

// Выводим текущее состояние поиска

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

<< ", text[0:" << i + 1 << "] = \"" << text.substr(0, i + 1)

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

// Если мы нашли полное совпадение, записываем позицию

if (j == m) {

int pos = i - m + 1;

positions.push\_back(pos);

std::cout << "Найдено вхождение в позиции: " << pos << "\n";

j = p[j - 1]; // Переход к следующему возможному совпадению

}

}

// Если совпадений не найдено, возвращаем -1

if (positions.empty()) {

positions.push\_back(-1);

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

}

return positions;

}

int main() {

std::string templ, text;

std::cout << "Введите строку template (образец): ";

std::cin >> templ;

std::cout << "Введите строку text (текст): ";

std::cin >> text;

// 1. Вычисляем префикс-функцию для `templ`

std::vector<int> p = computePrefixFunction(templ);

// 2. Выполняем поиск

std::vector<int> result = kmpSearch(text, templ, p);

// 3. Выводим результат

std::cout << "\n=== Итоговый результат ===\n";

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: lab4\_2.cpp

#include <iostream>

#include <vector>

#include <string>

// Функция для вычисления префикс-функции

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

int n = pattern.size();

std::vector<int> p(n, 0); // Массив для хранения значений префикс-функции

int j = 0; // Длина текущего наибольшего префикса, который является суффиксом

std::cout << "\n=== Вычисление префикс-функции ===\n";

for (int i = 1; i < n; i++) { // начинаем с 1, так как p[0] всегда 0

// Пока есть несоответствие, откатываемся с помощью префикс-функции

while (j > 0 && pattern[i] != pattern[j]) {

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

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

j = p[j - 1]; // Откатываемся к предыдущему совпадению

std::cout << "Назад к p[" << j << "]\n";

}

// Если символы совпадают, увеличиваем j

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

j++;

}

p[i] = j; // Записываем значение префикс-функции для i-го символа

// Вывод текущего состояния массива префикс-функции

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

<< ", pattern[0:" << i + 1 << "] = \"" << pattern.substr(0, i + 1)

<< "\", p[" << i << "] = " << p[i] << "\n";

}

return p;

}

// Функция Кнута-Морриса-Пратта для поиска циклического сдвига

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

int n = text.size();

int j = 0; // Индекс для текущего совпадения с шаблоном

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

// Проходим по строке text \* 2(доступ по индексам с операцией %), при этом ищем совпадения с pattern

for (int i = 0; i < n \* 2; i++) {

// Пока символы не совпадают, откатываемся по префикс-функции

while (j > 0 && text[i % n] != pattern[j]) {

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

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

j = p[j - 1]; // Возвращаемся к предыдущему возможному совпадению

std::cout << "Откат к p[" << j << "]\n";

}

// Если символы совпали, увеличиваем j

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

j++;

}

// Выводим текущее состояние поиска

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

<< ", text[0:" << i + 1 << "] = \"" << text.substr(0, i + 1)

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

// Когда j достигает длины шаблона, мы нашли вхождение

if (j == n) {

std::cout << "Найдено совпадение в позиции: " << (i - n + 1) % n << "\n";

return (i - n + 1) % n; // Возвращаем индекс сдвига

}

}

// Если совпадений не найдено, возвращаем -1

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

return -1;

}

int main() {

std::string text, shift;

std::cout << "Введите строку text (текст): ";

std::cin >> text;

std::cout << "Введите строку shift (сдвиг): ";

std::cin >> shift;

// Проверяем, равны ли длины строк (если нет — сдвига быть не может)

if (text.size() != shift.size()) {

std::cout << "-1\n";

return 0;

}

// 1. Вычисляем префикс-функцию для строки `s`

std::vector<int> p = computePrefixFunction(shift);

// 2. Выполняем поиск сдвига (ищем shift в text + text, чтобы учесть циклические сдвиги)

int result = kmpSearch(text, shift, p);

// 3. Выводим результат

std::cout << "\n=== Итоговый результат ===\n";

std::cout << result << "\n";

return 0;

}