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. 3342 _____ Мохамед М.Х.

Преподаватель _____ Виноградова Е.В.

Санкт-Петербург
2025

Цель работы

Изучить работу алгоритма Кнута-Морриса-Пратта, с его помощью решить задачу поиска вхождений заданного шаблона в текст и определение того, является ли одна строка циклическим сдвигом другой.

Задание 1

Реализуйте алгоритм КМП и с его помощью для заданных шаблона $P(|P| \leq 15000)$ и текста $T(|T| \leq 5.000.000)$ найдите все вхождения P в T.

Входные данные:

Первая строка – P

Вторая строка – T

Выходные данные:

Индексы начал вхождений P в T, разделенные запятой, если P не входит в T, то вывести -1.

Sample Input:

ab

abab

Sample Output:

0,2

Задание 2

Заданы две строки $A(|A| \leq 5.000.000)$ и $B(|B| \leq 5.000.000)$.

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.

Входные данные:

Первая строка – A

Вторая строка – B

Выходные данные:

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

Выполнение работы:

Описание алгоритма

- **Алгоритм Кнута–Морриса–Пратта (КМП)** — это эффективный метод поиска всех вхождений подстроки (паттерна) в текст. Он позволяет избежать повторного сравнения символов, используя заранее построенную вспомогательную информацию о паттерне.

Префикс-функция

- В основе алгоритма лежит **префикс-функция**. Для каждой позиции строки-паттерна она вычисляет длину наибольшего префикса, который одновременно является суффиксом для подстроки, заканчивающейся в этой позиции. Это позволяет при несовпадении символов не возвращаться в начало паттерна, а продолжать сравнение с наиболее подходящей позиции.
- Реализация функции:
- `std::vector<int> prefix_function(const std::string& str);`

- Возвращает массив префиксов для строки `str`.

Поиск подстроки (KMP)

- Алгоритм поиска по КМП использует два указателя: один по тексту, другой по паттерну. При совпадении символов оба указателя сдвигаются. Когда найдено полное совпадение паттерна, сохраняется позиция вхождения. При несовпадении алгоритм использует данные префикс-функции, чтобы сдвинуться в паттерне, не начиная сравнение с нуля.
- Реализация функции:
- `std::vector<int> kmp(const std::string& patt, const std::string& temp, bool stop_at_first);`

- `patt` — искомый паттерн
- `temp` — текст, в котором производится поиск
- `stop_at_first` — если `true`, поиск останавливается после первого совпадения
- Возвращает вектор позиций начала вхождений или {-1}, если совпадений нет

Проверка циклического сдвига

- Для проверки, является ли одна строка циклическим сдвигом другой, можно воспользоваться тем, что если строка `B` является сдвигом строки `A`, то `B` содержится как подстрока в строке `A + A`.
- Реализация функции:
- `int check_cycle(const std::string& patt, const std::string& temp);`

- Возвращает индекс начала вхождения или `-1`, если строки не являются сдвигами
- Использует функцию `kmp` для поиска `temp` в `patt + patt`

- # Оценка сложности

- **Временная сложность**:
  - Построение префикс-функции: **O(m)**, где `m` — длина паттерна
  - Поиск по тексту: **O(n)**, где `n` — длина текста
  - Проверка циклического сдвига: **O(2n)**, так как используется `patt + patt`
- **Пространственная сложность**:
  - **O(m)** — для хранения массива префикс-функции
  - **O(k)** — для хранения позиций совпадений (максимум `n - m + 1`)
-

- # Функции программы

Функция	Назначение
`prefix_function`	Строит префикс-функцию для строки
`kmp`	Выполняет поиск всех вхождений паттерна в текст
`check_cycle`	Проверяет, является ли строка циклическим сдвигом другой
`read_strings`	Считывает две строки из потока ввода
`print_vector`	Печатает вектор целых чисел (позиции совпадений)
`main`	Обрабатывает аргументы командной строки и запускает нужную задачу

Тестирование

Обе программы были протестированы на различных входных данных. Соответственно составлены две таблицы:

Таблица 1. Тестирование задачи поиска шаблона.

Входные данные	Выходные данные
ab abab	0,2
abc abc	0
abcd abc	-1
aa aaaaa	0,1,2,3
abac ababacabac	2,6
a b	-1

Таблица 2. Тестирование задачи поиска циклического сдвига.

Входные данные	Выходные данные
defabc abcdef	3
abc abcd	-1
abc def	-1
a a	0

abcabcabc bcabcabca	7
aaaaabaa baaaaaaa	5
abcdef cdefab	2

Выводы

Во время выполнения лабораторной работы, была изучена работа алгоритма Кнута-Морриса-Пратта. Решены задачи поиска вхождений заданного шаблона в текст и определение того, является ли одна строка циклическим сдвигом другой.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Имя файла: kmp.cpp

```cpp
#include <iostream>
#include <vector>
#include <string>

// Computes the prefix function (also known as "failure function") used in KMP
std::vector<int> prefix_function(const std::string& str) {
    int n = str.length();
    if(n == 0) return std::vector<int> (0);

    std::vector<int> prefix_arr(n);
    int j;

    for (int i = 1; i < n; ++i) {
        j = prefix_arr[i - 1];
        // Go back in the prefix array until a match is found or j becomes 0
        while (j > 0 && str[i] != str[j]) j = prefix_arr[j - 1];
        if (str[i] == str[j]) j++;  // If match found, increment j
        prefix_arr[i] = j;  // Store the result
    }

    return prefix_arr;
}

// Implements the Knuth-Morris-Pratt (KMP) algorithm for pattern matching
std::vector<int> kmp(const std::string& patt, const std::string& temp, bool
stop_at_first) {
    std::vector<int> answer;
    int patt_len = patt.size();
    int temp_len = temp.size();

    if(patt_len == 0 || temp_len == 0) {
        answer.push_back(-1);  // If any string is empty, return -1
        return answer;
    }

    // Compute prefix function for pattern with separator
    std::vector<int> p = prefix_function(patt + "#");
    int j = 0;

    for(int i = 0; i < temp_len; ++i) {
        while(j > 0 && patt[j] != temp[i]) j = p[j-1];  // Follow prefix function
        if(patt[j] == temp[i]) j++;  // Characters match
        if(j == patt_len) {
            answer.push_back(i - patt_len + 1);  // Match found at position
            if(stop_at_first) break;  // Stop if only the first match is needed
```

```
        }
    }

    if(answer.empty()) answer.push_back(-1);  // If no match found
    return answer;
}

// Checks whether "temp" is a cyclic shift of "patt"
int check_cycle(const std::string& patt, const std::string& temp) {
    int patt_len = patt.size();
    int temp_len = temp.size();

    if(patt_len == 0 || temp_len == 0) return -1;  // Empty input
    if(patt_len != temp_len) return -1;  // Lengths must match for a cyclic shift

    // Double the original string and search for the other
    std::vector<int> res = kmp(temp, patt + patt, true);
    return res[0];  // Return position of match (or -1)
}

// Reads two strings from the input
bool read_strings(std::string& patt, std::string& temp, std::istream& in) {
    in >> patt;
    in >> temp;

    if(patt.size() == 0 || temp.size() == 0) return false;
    return true;
}

// Prints a vector of integers separated by commas
void print_vector(const std::vector<int>& vec) {
    for(int i = 0; i < vec.size(); ++i) {
        if(i == vec.size() - 1) {
            std::cout << vec[i] << "\n";
        }
        else {
            std::cout << vec[i] << ",";
        }
    }
}

// Main program entry point
int main(int argc, char** argv) {
    int task = 1;

    // Parse command-line arguments
    for(int i = 0; i < argc; ++i) {
        if(std::string(argv[i]) == "-kmp") {
            task = 1;
            break;
```

```
        }
        else if(std::string(argv[i]) == "-cycle") {
            task = 2;
            break;
        }
        else if(std::string(argv[i]) == "-h" || std::string(argv[i]) == "--help") {
            task = 0;
            break;
        }
    }

    // If no task specified
    if(task == -1) {
        std::cout << "You have to specialize the task: either choose KMP or Cycle
Check.\nTo get more info use key -h or --help.\n";
        return 0;
    }
    else if(task == 0) {
        // Show help message
        std::cout << "Use -kmp to start Knuth-Morris-Pratt Algorithm\n";
        std::cout << "Use -cycle to check if a string is a cycle shift of another
one\n";
        return 0;
    }

    std::string pattern, temp;

    // Read input strings
    if(!read_strings(pattern, temp, std::cin)) {
        std::cout << "You've entered empty string\n";
        return 0;
    }

    // Execute the chosen task
    if(task == 1) {
        std::vector<int> res = kmp(pattern, temp, false);
        print_vector(res);
        return 0;
    }
    else {
        std::cout << check_cycle(pattern, temp) << '\n';
        return 0;
    }

    return 0;
}
```

Имя файла: shift.cpp

```cpp
#include <iostream>
#include <vector>
#include <string>

// Computes the prefix function used in the KMP algorithm
std::vector<int> prefix_function(const std::string& str) {
    int n = str.length();
    if (n == 0) return std::vector<int>(0); // If the string is empty

    std::vector<int> prefix_arr(n);
    int j;

    // Build the prefix array
    for (int i = 1; i < n; ++i) {
        j = prefix_arr[i - 1];
        // Backtrack in prefix array while characters don't match
        while (j > 0 && str[i] != str[j]) j = prefix_arr[j - 1];
        // If characters match, increase j
        if (str[i] == str[j]) j++;
        // Assign the prefix value
        prefix_arr[i] = j;
    }

    return prefix_arr;
}

// KMP (Knuth-Morris-Pratt) string matching algorithm
std::vector<int> kmp(const std::string& patt, const std::string& temp, bool
stop_at_first) {
    std::vector<int> answer;
    int patt_len = patt.size();
    int temp_len = temp.size();

    // If either string is empty, return -1
    if (patt_len == 0 || temp_len == 0) {
        answer.push_back(-1);
        return answer;
    }

    // Build prefix function for pattern
    std::vector<int> p = prefix_function(patt + "#");
    int j = 0;

    for (int i = 0; i < temp_len; ++i) {
        // While characters mismatch, backtrack using prefix function
        while (j > 0 && patt[j] != temp[i]) j = p[j - 1];
        if (patt[j] == temp[i]) j++; // If characters match, move forward
```

```
        if (j == patt_len) {
            // Found full match
            answer.push_back(i - patt_len + 1); // Store start index of match
            if (stop_at_first) break; // Stop if only one match is needed
        }
    }

    // If no matches found, return -1
    if (answer.empty()) answer.push_back(-1);

    return answer;
}

// Checks if one string is a cyclic shift of another
int check_cycle(const std::string& patt, const std::string& temp) {
    int patt_len = patt.size();
    int temp_len = temp.size();

    // If lengths differ or any string is empty, return -1
    if (patt_len == 0 || temp_len == 0) return -1;
    if (patt_len != temp_len) return -1;

    // Check if temp exists in patt + patt using KMP
    std::vector<int> res = kmp(temp, patt + patt, true);

    return res[0]; // Return the match index or -1
}

// Reads two input strings from input stream
bool read_strings(std::string& patt, std::string& temp, std::istream& in) {
    in >> patt;
    in >> temp;

    // Return false if any string is empty
    if (patt.empty() || temp.empty()) return false;
    return true;
}

// Prints the content of a vector separated by commas
void print_vector(const std::vector<int>& vec) {
    for (int i = 0; i < vec.size(); ++i) {
        if (i == vec.size() - 1)
            std::cout << vec[i] << "\n";
        else
            std::cout << vec[i] << ",";
    }
}

int main(int argc, char** argv) {
    int task = 2; // Default to cycle check
```

```
    // Parse command-line arguments to set task type
    for (int i = 0; i < argc; ++i) {
        if (std::string(argv[i]) == "-kmp") {
            task = 1;
            break;
        } else if (std::string(argv[i]) == "-cycle") {
            task = 2;
            break;
        } else if (std::string(argv[i]) == "-h" || std::string(argv[i]) == "--
help") {
            task = 0;
            break;
        }
    }

    // If no valid task specified
    if (task == -1) {
        std::cout << "You have to specialize the task: either choose KMP or Cycle
Check.\nTo get more info use key -h or --help.\n";
        return 0;
    } else if (task == 0) {
        // Display help information
        std::cout << "Use -kmp to start Knuth-Morris-Pratt Algorithm\n";
        std::cout << "Use -cycle to check if a string is a cycle shift of another
one\n";
        return 0;
    }

    std::string pattern, temp;

    // Read input strings
    if (!read_strings(pattern, temp, std::cin)) {
        std::cout << "You've entered empty string\n";
        return 0;
    }

    // Execute selected task
    if (task == 1) {
        // Perform KMP search
        std::vector<int> res = kmp(pattern, temp, false);
        print_vector(res);
        return 0;
    } else {
        // Perform cycle shift check
        std::cout << check_cycle(pattern, temp) << '\n';
        return 0;
    }

    return 0;
```

```
}
```