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: «Поиск подстроки в строке»

Студент гр. 3343 _____ Бондаренко Ф. А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы.

Изучить и реализовать алгоритм Кнута-Морриса-Пратта для поиска указанного шаблона в тексте. Написать программу, которая применяет алгоритм КМП на практике: 1) для поиска шаблона в тексте; 2) для определения, являются ли строки циклическим сдвигом друг друга.

Задание.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P| \leq 15000$) и текста T ($|T| \leq 5000000$) 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

Заданы две строки A ($|A| \leq 5000000$) и B ($|B| \leq 5000000$).

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.

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

Выполнение работы.

Описание реализованных функций для алгоритма КМП:

- s*td::vector<int> prefix_function(const std::string& text, bool record)* — префикс-функция — вычисляет значение префикс-функции (нахождение максимального по длине собственного префикса, совпадающего с суффиксом) для каждого среза *text[0:i+1]* (от первого символа до *i*-го включительно). Флаг *record* необходим для фиксации поэтапной работы алгоритма. Функция возвращает вектор полученных значений.
- *std::vector<int> find(const std::string& text, const std::string& pattern, bool record)* — функция поиска — основана на алгоритме КМП: вычисляет вектор значений префикс-функции для шаблона (*pattern*), который используется алгоритме: при несовпадении текущих символов в тексте и шаблоне откат указателя в шаблоне будет осуществляться к предыдущему значению префикс-функции, что существенно уменьшает время работы алгоритма. Т.е. если *prefix[j-1] = k*, то продолжение поиска совпадений будет осуществляться не с первого элемента в шаблоне, а с k-го. Флаг *record* необходим для фиксации поэтапной работы алгоритма. Функция возвращает массив вхождений шаблона в текст.

- *int is_cycle_shift(const std::string& a, const std::string& b)* — функция проверки: является ли *a* циклическим сдвигом *b*. Работа функции основана на применении алгоритма КМП: если *a* — циклический сдвиг *b*, то при удвоении строки *b*, строка *a* должна содержаться в ней в качестве подстроки. Функция возвращает целое число — индекс начала *b* в *a*.

Описание алгоритма префикс-функции:

Алгоритм поиска префикс-функции основан на двух свойствах:

Пусть P(text, i) = k, где: P() — префикс-функция, text — обрабатываемый текст, i — максимальный индекс в text (срез), тогда:

- Если text[i + 1] == text[k + 1], то: P(text, i + 1) = k + 1.
- text[1..P(text, k)] — префикс-суффикс строки text[1..i].

Тогда итоговый алгоритм будет таким:

1) Инициализация:
   - Создаем последовательность P[n] (вектор) длины n (длина строки). Первый элемент последовательности — 0.
   - Создаем две переменные для хранения индексов: j — индекс первого элемента строки, i — следующего за j.
2) Сравниваем символы на индексах i и j:
   - Если T[j] == T[i], тогда: P[i] = j + 1 и увеличиваем i и j на единицу.
   - Если T[j] != T[i], тогда:
     - Если j == 0, то P[i] = 0 и увеличиваем i на единицу.
     - Если j != 0, то j = P[j - 1].
3) Если i < n (длина строки), то переходим к п. (2), иначе: заканчиваем алгоритм.

Описание алгоритма Кнута-Морриса-Пратта:

Работает на основе префикс-функции: если текущие символы шаблона и текста не совпали, то мы не обнуляем шаблон полностью (т.е. не ставим указатель в шаблоне на первый элемент), а берем указатель в шаблоне из предыдущего значения префикс-функции.

Алгоритм поиска КМП:

1) Инициализация:

- Вычисляем префикс функцию для шаблона
- Объявляем две дополнительные переменные (i, j) для хранения индексов позиций в строке и шаблоне соответственно. Значение для j всегда — индекс первого элемента в подстроке, для i — индекс любого (а какой части строки ищем шаблон) элемента строки.

2) Сравниваем символы на индексах i и j:

- Если text[i] == pattern[j]:
  - Если j == m - 1 (длина шаблона), то заканчиваем алгоритм (успех).
  - Если j != m - 1, то увеличиваем i и j на единицу и переходим к п. (2).
- Если text[i] != pattern[j]:
  - Если i >= n - 1 (длина текста), то заканчиваем алгоритм (неудача).
  - Если j == 0, то увеличиваем i на единицу и переходим к п. (2).
  - Если j > 0, то j = prefix[j-1] и переходим к п. (2).

Оценка сложности алгоритма.

Обычного алгоритма КМП:

- По времени: $O(n+m)$, где n — длина текста, m — длина шаблона: за O(m) строится префикс-функция для шаблона, за O(n) осуществляется проход по тексту.

- По памяти: $O(m)$, где m — длина шаблона: необходимо хранить вектор значений префикс функции для шаблона.

Алгоритма КМП для циклического сдвига:

- По времени: $O(3n)=O(n)$, где n — длина текста и его сдвига: за O(n) строится префикс-функция для сдвига, за O(2n) проходится удвоенный текст.
- По памяти: $O(n)$, где n — длина сдвига: необходимо хранить вектор значений префикс функции для сдвига.

Тестирование.

Входные данные	Выходные данные	Комментарий
abra abracadabra	0,7	Успех для поиска
a aaaaaaaaaaaaaaaa	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16	Успех для поиска
aba Helloworld	-1	Успех для поиска
abra cadabraabra	-1	Успех для сдвига
abcd dabc	3	Успех для сдвига

Вывод.

Изучен алгоритм Кнута-Морриса-Пратта для нахождения шаблона в тексте. Алгоритм был применен на практике: написаны функция для поиска шаблона в тексте и нахождении циклического сдвига.

ПРИЛОЖЕНИЕ А

Файл *main.cpp*:

```
#include <functional>
#include <iostream>
#include <map>
#include <string>
#include <vector>

#include "../include/kmp.hpp"

void findMode() {
  std::string pattern, text;
  std::cin >> pattern >> text;

  std::vector<int> matches = kmp::find(text, pattern, false);
  for (size_t i = 0; i < matches.size(); i++) {
    std::cout << matches[i];
    if (i < matches.size() - 1) {
      std::cout << ',';
    }
  }

  std::cout << std::endl;
}

void cycleMode() {
  std::string a, b;
  std::cin >> a >> b;

  int pos = kmp::is_cycle_shift(a, b);

  std::cout << pos << std::endl;
}

void stepMode() {
  std::string pattern, text;
  std::cin >> pattern >> text;

  std::vector<int> matches = kmp::find(text, pattern, true);
}

int main(int argc, char* argv[]) {
  if (argc < 2) {
    return 1;
  }

  std::string mode = argv[1];
    std::map<std::string,  std::function<void()>>  modes  =  {{"find",
findMode},
                                                             {"cycle",
cycleMode},
                                                              {"step",
stepMode}};

  auto it = modes.find(mode);
  if (it != modes.end()) {
    it->second();
```

```
  }

  return 0;
}
```

Файл *kmp.hpp*:

```
#ifndef KMP_HPP_
#define KMP_HPP_

#include <string>
#include <utility>
#include <vector>

namespace kmp {

std::vector<int>   prefix_function(const   std::string&   text,   bool
record);

std::vector<int>  find(const  std::string&  text,  const  std::string&
pattern,
                       bool record);

int is_cycle_shift(const std::string& a, const std::string& b);

}  // namespace kmp

#endif  // KMP_HPP_
```

Файл *kmp.cpp*:

```
#include "../include/kmp.hpp"

#include <iostream>

namespace kmp {

namespace details {

const std::string red = "\033[31m";
const std::string reset = "\033[0m";

}  // namespace details

std::vector<int> prefix_function(const std::string& text, bool record)
{
  int n = text.length();
  std::vector<int> prefix(n, 0);

  if (record) {
    std::cout << std::endl;
    std::cout << "==========================" << std::endl;
    std::cout << "Свойства префикс-функции: " << std::endl;
     std::cout << "Пусть P(text, i) = k, где: P() - префикс-функция,
text - "
                  "обрабатываемый текст, i - максимальный индекс в text
(срез)"
              << std::endl;

    std::cout << "Тогда: " << std::endl;
    std::cout
```

```
        << "\t1) Если text[i + 1] == text[k + 1], то: P(text, i + 1) =
k + 1"
        << std::endl;
     std::cout << "\t2) text[1..P(text, k)] - префикс-суффикс строки
text[1..i]"
              << std::endl;

    std::cout << std::endl;

       std::cout << "В данном алгоритме введены переменные: " <<
std::endl;
     std::cout << "\t1) j - конец максимального префикса для данной
подстроки, "
                  "что совпадает с суффиксом"
              << std::endl;
     std::cout << "\t2) i - конец максимального суффикса для данной
подстроки, "
                  "что совпадает с префиксом"
              << std::endl;
    std::cout << "==========================" << std::endl;
    std::cout << std::endl;
  }

  for (int i = 1; i < n; i++) {
    int j = prefix[i - 1];
    if (record) {
      std::cout << std::endl;
      std::cout << "--------------------------" << std::endl;
           std::cout << "Поиск максимального префикс-суффикса для
подстроки: "
                << "[" << text.substr(0, i + 1) << "]" << std::endl;

      std::cout << "Значение [j]: " << "[" << j << "]" << std::endl;
      std::cout << "Значение [i]: " << "[" << i << "]" << std::endl;
    }

    while ((j > 0) && (text[i] != text[j])) {
      j = prefix[j - 1];
      if (record) {
        std::cout << "Пользуемся свойством (2): " << std::endl;
          std::cout << "Новое значение [j]: " << "[" << j << "]" <<
std::endl;
      }
    }

    if (text[i] == text[j]) {
      j++;
      if (record) {
        std::cout << "Пользуемся свойством (1): " << std::endl;
          std::cout << "Новое значение [j]: " << "[" << j << "]" <<
std::endl;
      }
    }

    prefix[i] = j;
    if (record) {
       std::cout << "Максимальный префикс-суффикс (длина): " << j <<
std::endl;
      std::cout << "Максимальный префикс-суффикс (значение): "
```

```
                << "[" << text.substr(0, j) << "]" << std::endl;
      std::cout << "-------------------------" << std::endl;
      std::cout << std::endl;
    }
  }

  if (record) {
    std::cout << std::endl;
    std::cout << "=========================" << std::endl;
    std::cout << "Итоговый вектор префикс функции: " << std::endl;
    std::cout << "prefix = [";
    for (size_t i = 0; i < prefix.size(); i++) {
      std::cout << prefix[i];
      if (i < prefix.size() - 1) {
        std::cout << ", ";
      }
    }
    std::cout << "]" << std::endl;
    std::cout << "=========================" << std::endl;
    std::cout << std::endl;
  }

  return prefix;
}

std::vector<int>  find(const  std::string&  text,  const  std::string&
pattern,
                       bool record) {
  std::vector<int> matches;

  int n = text.length();
  int m = pattern.length();
  if (m > n) {
    return {-1};
  }

  if (record) {
    std::cout << std::endl;
    std::cout << "=========================" << std::endl;
      std::cout  << "В данном алгоритме КМП введены переменные: " <<
std::endl;
       std::cout  <<  "\t1)  j  -  индекс текущего элемента в шаблоне
(pattern)"
              << std::endl;
    std::cout << "\t2) i - индекс текущего элемента в тексте (text)"
              << std::endl;
    std::cout << "=========================" << std::endl;
    std::cout << std::endl;
  }

  std::vector<int> prefix = prefix_function(pattern, record);

  int j = 0;
  for (int i = 0; i < n; i++) {
    if (record) {
      std::cout << std::endl;
      std::cout << "-------------------------" << std::endl;
      std::cout << "Значение [j]: " << "[" << j << "]" << std::endl;
      std::cout << "Значение [i]: " << "[" << i << "]" << std::endl;
```

```
        std::cout << "Текст [text]: " << "[" << text.substr(0, i) <<
details::red
                  << text[i] << details::reset << text.substr(i + 1) <<
"]"
                  << std::endl;

      std::cout << "Шаблон [pattern]: " << "[";
      std::cout << pattern.substr(0, j) << details::red << pattern[j]
                << details::reset << pattern.substr(j + 1);
      std::cout << "]" << std::endl;
    }

    while ((j > 0) && (text[i] != pattern[j])) {
      j = prefix[j - 1];
      if (record) {
        std::cout << std::endl;
          std::cout << "Символы text[i] и pattern[j] не совпали: " <<
std::endl;
            std::cout << "text[i] = " << "[" << text[i] << "]" <<
std::endl;
          std::cout << "pattern[j] = " << "[" << pattern[j] << "]" <<
std::endl;
           std::cout << "Новое значение [j]: " << "[" << j << "]" <<
std::endl;
      }
    }

    if (text[i] == pattern[j]) {
      j++;
      if (record) {
        std::cout << std::endl;
        std::cout << "Найдено совпадение символов: " << std::endl;
          std::cout << "Новое значение [j]: " << "[" << j << "]" <<
std::endl;
      }
    }

    if (j >= m) {
      matches.push_back(i - j + 1);
      if (record) {
        std::cout << std::endl;
         std::cout << "Найдена подстрока: " << "[" << text.substr(i - j
+ 1, j)
                   << "]" << std::endl;
        std::cout << "Начало подстроки в тексте [i - j + 1]: " << "["
                   << i - j + 1 << "]" << std::endl;
         std::cout << "Конец подстроки в тексте [i]: " << "[" << i <<
"]"
                   << std::endl;
      }
      j = prefix[j - 1];
    }

    if (record) {
      std::cout << "-------------------------" << std::endl;
      std::cout << std::endl;
    }
  }
```

```
  if (matches.size() == 0) {
    matches.push_back(-1);
  }

  if (record) {
    std::cout << std::endl;
    std::cout << "=========================" << std::endl;
    std::cout << "Итоговый вектор совпадений: " << std::endl;
    std::cout << "matches = [";
    for (size_t i = 0; i < matches.size(); i++) {
      std::cout << matches[i];
      if (i < matches.size() - 1) {
        std::cout << ", ";
      }
    }
    std::cout << "]" << std::endl;
    std::cout << "=========================" << std::endl;
    std::cout << std::endl;
  }

  return matches;
}

int is_cycle_shift(const std::string& a, const std::string& b) {
  if (a.length() != b.length()) {
    return -1;
  }

  std::vector<int> matched = find(a + a, b, false);

  return matched[0];
}

}  // namespace kmp
```

Файл *Makefile*

```
CCX = g++
CXXFLAGS = -std=c++20 -Wall -Wextra -pedantic

SRC_DIR = src
INCLUDE_DIR = include
OBJ_DIR = obj

SRC_FILES = $(wildcard $(SRC_DIR)/*.cpp)
OBJ_FILES = $(patsubst $(SRC_DIR)/%.cpp,$(OBJ_DIR)/%.o,$(SRC_FILES))

EXEC = $(OBJ_DIR)/kmp

$(shell mkdir -p $(OBJ_DIR))

all: $(EXEC)

$(EXEC) : $(OBJ_FILES)
	$(CXX) $(CXXFLAGS) -I$(INCLUDE_DIR) -o $@ $^

$(OBJ_DIR)/%.o : $(SRC_DIR)/%.cpp
	$(CXX) $(CXXFLAGS) -I$(INCLUDE_DIR) -c $< -o $@
```

```
clean:
	rm -rf $(OBJ_DIR)/*.o $(EXEC)

find: $(EXEC)
	./$(EXEC) find

cycle: $(EXEC)
	./$(EXEC) cycle

step: $(EXEC)
	./$(EXEC) step

.PHONY: all clean find cycle step
```