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: Алгоритм Кнута-Морриса-Пратта

Студент гр. 9383 _____ Ноздрин В.Я.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Реализовать алгоритм Кнута-Морриса-Пратта и оценить его сложность.

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

Описание алгоритма.

Для решения задачи был реализован алгоритм Кнута-Морриса-Пратта для поиска вхождений строки-образца в тексте. Идея алгоритма заключается в том, чтобы вычислить префикс-функцию для образца и, пользуясь этими значениями, выполнить лишь один линейный проход по тексту для поиска всех вхождений образца в тексте.

Оценка сложности жадного алгоритма.

Префикс-функция вычилсяется линейно за O(m), где m — длина строки-образца. Поиск вхождений выполняется также за линейное время O(n), где n — длина текста. Тогда итоговая сложность будет O(n+m).

Описание функций и структур данных.

std::vector<size_t> prefixFunction – префикс-функция.

std::vector<size_t> kmp – метод, реализующий поиск вхождений.

void print() – метод, выводящий на экран результат поиска.

Выводы.

Применен на практике алгоритм поиска вхождений строки-шаблона в теест. Исследован алгоритм Кнута-Морриса-Пратта на предмет сложности.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <iostream>
#include <vector>

#define KMP 1

void print(std::vector<size_t> numbers) {
  for (int i = 0; i < numbers.size(); i++) {
    if (i == 0)
      std::cout << numbers[i];
    else
      std::cout << "," << numbers[i];
  }
  std::cout << "\n";
}

std::vector<size_t> prefixFunction(const std::string& str) {
  std::vector<size_t> prefixValues(str.length());
  for (int i = 1; i < str.size(); i++) {
    size_t counter = prefixValues[i-1];
    while (counter > 0 && str[counter] != str[i])
      counter = prefixValues[counter-1];
    if (str[i] == str[counter])
      counter++;
    prefixValues[i]=counter;
  }
  return prefixValues;
}

std::vector<size_t> kmp(std::string& sample, std::string& text) {
  std::vector<size_t> result;
  std::vector<size_t> samplePrefixValues = prefixFunction(sample);
  size_t sampleInd = 0;
  for (size_t textInd = 0; textInd < text.size(); textInd++) {
    if (text[textInd] == sample[sampleInd]) {
      sampleInd++;
      if (sampleInd == sample.size()) {
        result.push_back(textInd-sample.size()+1);
      }
      continue;
    }
    if (sampleInd == 0) continue;
    sampleInd = samplePrefixValues[sampleInd-1];
    textInd--;
  }
  if (result.empty())
    result.push_back(-1);
  return result;
}

int isCyclicShift(std::string& sample, std::string& text) {
  if (sample.length()!=text.length())
    return -1;
```

```
  if (sample == text)
    return 0;
  std::vector<size_t> result;
  std::vector<size_t> samplePrefixValues = prefixFunction(sample);
  size_t sampleInd = 0;
  for (size_t textInd = 0; textInd < 2*text.size(); textInd++) {
    if (text[textInd%text.size()] == sample[sampleInd]) {
      sampleInd++;
      if (sampleInd == sample.size()) {
        return textInd-sample.size()+1;
      }
      continue;
    }
    if (sampleInd == 0) continue;
    sampleInd = samplePrefixValues[sampleInd-1];
    textInd--;
  }
  return -1;
}


int main() {
  std::string sample, text;
  std::cin >> sample >> text;
#if KMP
  auto result = kmp(sample, text);
  print(result);
#else
  std:: cout << isCyclicShift(sample, text);
  return 0;
#endif
}
```