МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студент гр. 9383 _____ Ноздрин В.Я.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Реализовать алгоритм Ахо-Корасик и оценить его сложность.

Задание 1.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст ($1 \le |T| \le 100000$).

Вторая - число n ($1 \le n \le 3000$), каждая следующая из nn строк содержит шаблон из набора $P=\{p1,\ldots,pn\}$ $1 \le |pi| \le 75$

Все строки содержат символы из алфавита {A,C,G,T,N}

Выход:

Все вхождения образцов из P в T.

Каждое вхождение образца в текст представить в виде двух чисел - i p

Где i - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p

(нумерация образцов начинается с 1).

Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

Задание 2.

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с джокером.

В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу P необходимо найти все вхождения P в текст T.

Например, образец ab??c?ab??c? с джокером ?? встречается дважды в тексте xabvccbababcax.

Символ джокер не входит в алфавит, символы которого используются в T. Каждый джокер соответствует одному символу, а не подстроке неопределённой длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы.

Все строки содержат символы из алфавита {A,C,G,T,N}

Вход:

Текст (T, $1 \leq |T| \leq 100000$)

Шаблон (P, $1 \leq |P| \leq 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания.

Описание алгоритма.

Для решения задачи был реализован алгоритм Ахо-Корасик для поиска вхождений строк-образцов из заданного словаря в тексте. Идея алгоритма в том, чтобы по словарю построить конечный автомат, которому затем передается строка-текст. Если автомат приходит в конечное состояние, найдено вхождение.

Оценка сложности жадного алгоритма.

Алгоритм имеет линейную сложность O(l+n+k), где l — длина текста, n — общая длина всех слов в словаре, умноженная на размер алфавита, а k — общая длина всех вхождений.

Описание функций и структур данных.

class AhoCorasick – класс-оболочка для алгоритма в конструкторе добавляются строки в словарь и строятся суффиксные ссылки. Далее вызывается поиск от строки.

struct Node – структура, на которой строится конечный автомат как дерево с ссылками.

Выводы.

Применен на практике алгоритм поиска вхождений строк заданного словаря в тексте. Исследован алгоритм Ахо-Корасик 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
}
```