**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: Поиск набора подстрок в строке (Ахо-Корасик)

Вариант: 1

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Павлов А.Р. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2025

**Цель работы:**

Изучить принцип работы алгоритма Ахо-Корасик для нахождения подстрок в строке. Решить с его помощью задачи.

**Задание 1:**

Разработайте программу, решающую задачу точного поиска набора образцов.  
**Вход:**  
Первая строка содержит текст (T, 1≤∣T∣≤100000T).

Вторая - число *n* (1≤n≤3000), каждая следующая из *n* строк содержит шаблон из набора *P*={*p\_*1​,…,*p\_n*​} 1≤∣*p\_i*​∣≤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.

**Sample Input:**

NTAG

3

TAGT

TAG

T

**Sample Output:**

2 2

2 3

**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 Р в текст Т.  
Например, образец аb??с?аb??с? 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≤∣T∣≤100000)

Шаблон (*P*,1≤∣*P*∣≤40)

Символ джокера

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

Номера должны выводиться в порядке возрастания.

**Sample Input:**

ACTANCA

A$$A$

$

**Sample Output:**

1

**Реализация**

*Описание алгоритма Ахо-Корасик для точного поиска образцов:*

Алгоритм Ахо-Корасик предназначен для эффективного поиска всех вхождений множества заданных шаблонов P={p\_1,p\_2,…,p\_n} в текст T. Он использует структуру данных, называемую бором (trie), дополненную суффиксными и терминальными ссылками, что позволяет выполнять поиск за один проход по тексту независимо от числа шаблонов. В данной реализации алгоритм также подсчитывает количество вершин в автомате и определяет шаблоны, имеющие пересечения с другими вхождениями в тексте.

**Шаги алгоритма**

Создаётся корневое состояние (0) с suffLink = -1 (суффиксная ссылка) и terminalLink = −1 (терминальная ссылка). Определяется максимальное число состояний как сумма длин всех шаблонов плюс корень (∑∣pi∣+1).

Для каждого шаблона p\_i из P. Начиная с корня, для каждого символа c в p\_i добавляется новое состояние, если перехода по c ещё нет. Счётчик вершин state\_count увеличивается на 1 для каждого нового состояния. Конечное состояние шаблона отмечается его индексом i в массиве output.

Далее строится автомат. Используется очередь для обхода состояний в порядке ширины (BFS). Для каждого состояния из корня: suffLink = 0, terminalLink устанавливается как само состояние, если оно содержит шаблон, иначе -1.

Для остальных состояний. Суффиксная ссылка suffLink[next\_state] определяется поиском самого длинного суффикса текущей строки, доступного через suffLink[current\_state]. Терминальная ссылка terminalLink[next\_state] указывает на ближайший узел с шаблоном по цепочке suffLink, либо на −1.

Затем производится поиск в тексте. Проход по символам текста T с использованием текущего состояния current\_state. Если перехода по символу нет, алгоритм следует по suffLinkдо корня или подходящего состояния. При достижении состояния с непустым output или через terminalLink фиксируются все вхождения шаблонов с вычислением их позиций.

*Оценка сложности алгоритма:*

**Временная сложность**

Построение бора:

* + Для каждого символа всех шаблонов создаётся или используется состояние: , где — суммарная длина шаблонов

Построение автомата:

* + Обход BFS по всем состояниям (до ) с проверкой переходов для каждого символа алфавита.
  + для небольшого алфавита, так как считается константой.

Поиск:

* + Проход по тексту: переходов, каждый с по suffLink и terminalLink благодаря терминальным ссылкам.
  + Извлечение совпадений: , где — число вхождений.

*Итог:*

**Пространственная сложность**

Бор:

* + transitions: состояний, каждый с переходов (в худшем случае).
  + output: для хранения индексов шаблонов.

Ссылки:

* + suffLink, terminalLink: элементов.

Результаты:

* + results: пар.
  + overlap\_patterns: в худшем случае, где — число шаблонов.

*Итог:* для автомата плюс для результатов, итого .

**Тестирование**

Таблица 1. Тестирование.

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| ACGT  2  ACGTACGT  CGTA |  |
| AAAAA  2  A  AA | 1 1  1 2  2 1  2 2  3 1  3 2  4 1  4 2  5 1 |
| ACGTACGT  3  AC  CG  GT | 1 1  2 2  3 3  5 1  6 2  7 3 |
| ACGTACGT  3  A  AC  ACG | 1 1  1 2  1 3  5 1  5 2  5 3 |

*Описание Алгоритма Ахо-Корасик с джокерами:*

Алгоритм Ахо-Корасик в данной реализации адаптирован для поиска всех вхождений шаблона P с джокерами (специальный символ, обозначающий совпадение с любым символом) в тексте T. Он использует бор (trie) с суффиксными и конечными ссылками для поиска всех безджокерных подстрок шаблона P, а затем проверяет полное соответствие P в найденных позициях с учётом джокеров. Алгоритм также подсчитывает количество вершин в автомате и определяет шаблоны с пересекающимися вхождениями.

**Шаги алгоритма**

Создаётся корневое состояние (0) с suffLink = 0 (суффиксная ссылка) и termLink = −1 (конечная ссылка). Определяется максимальное число состояний как сумма длин всех шаблонов плюс корень (∑∣pi∣+1).

Для каждой подстроки Q\_i из P. Начиная с корня, создаются состояния для каждого символа c, если перехода нет. Счётчик state\_count увеличивается для новых состояний. Конечное состояние подстроки помечается её индексом в output.

Далее производится построение автомата. Для этого будет использоваться обход в ширину (BFS). Для детей корня: suffLink = 0, termLink — само состояние, если оно конец шаблона, иначе −1. Для остальных: suffLink определяется поиском через suffLink родителя, termLink — ближайший узел с шаблоном.

Далее проходим по T с использованием transitions и suffLink для переходов. Совпадения извлекаются через output и termLink.

Для каждого совпадения Q\_i на позиции j: C[j - l\_i] += 1, где l\_i — стартовая позиция Q\_i в P. Позиции i, где C[i] = k и T[i:i+|P|] соответствует P с джокерами, добавляются в результат.

Также по заданию из варианта требовалось осуществить сравнение с шаблоном, где джокер может быть любым символом кроме заданного. Это условие было добавлено в процесс сравнения потенциальных вариантов.

*Оценка сложности алгоритма:*

**Временная сложность:**

Разбиение P:

* + для разбиения по wildcard и вычисления start\_positions.

Построение бора:

* + для добавления подстрок, где .

Построение автомата:

* + для BFS с фиксированным алфавитом ( — константа).

Поиск:

* + для прохода по тексту, для извлечения совпадений ( — число вхождений).

Проверка джокеров:

* + для анализа C и проверки символов P в каждой позиции.

Пересечения:

* + для сравнения всех пар вхождений.

*Итого:*

**Пространственная сложность**

Бор:

* + transitions: состояний.
  + output, suffLink, termLink: .

Поиск:

* + C: .
  + results: .
  + overlap\_patterns: , где .

*Общая:* .

**Тестирование**

Таблица 2. Тестирование.

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| ACTATCA  A$$A$  $  N | 1 |
| ACTATCA  A$$A$  $  T |  |

**Вывод**

В ходе лабораторной работы были написаны программы с использованием алгоритма Ахо-Корасика. Также дополнительно было сделано: подсчёт вершин и определение пересечений.