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, $1 \leq |T| \leq 100000$T).

Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P=\{p\_1,\ldots,p\_n\}$ $1 \leq |p\_i| \leq 75$.

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 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.

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: $O(\sum | pi |)$, где $| \sum | pi |$ — суммарная длина шаблонов

Построение автомата:

- Обход BFS по всем состояниям (до $\sum | pi | + 1$) с проверкой переходов для каждого символа алфавита.
- $O(\sum | pi |)$ для небольшого алфавита, так как $| \Sigma |$ считается константой.

Поиск:

- Проход по тексту: $O(| T |)$ переходов, каждый с $O(1)$ по suffLink и terminalLink благодаря терминальным ссылкам.

Итог: $O(|T|)$

Пространственная сложность

Бор:

- transitions: $O(\sum | pi |)$ состояний, каждый с $O(| \Sigma |)$ переходов (в худшем случае).
- output: $O(\sum | pi |)$ для хранения индексов шаблонов.

Ссылки:

- suffLink, terminalLink: $O(\sum | pi |)$ элементов.

Результаты:

- results: $O(z)$ пар.
- overlap_patterns: $O(n)$ в худшем случае, где n — число шаблонов.

Итог: $O(T + z)$

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 бездженкерных 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 ($\sum|p_i|+1$).

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:

- $O(|P|)$ для разбиения по wildcard и вычисления start_positions.

Построение бора:

- $O(\sum |Qi|)$ для добавления подстрок, где $\sum |Qi| \leq |P|$.

Построение автомата:

- $O(\sum |Qi|)$ для BFS с фиксированным алфавитом ($|\Sigma|$ — константа).

Поиск:

- $O(|T|)$ для прохода по тексту, $O(z)$ для извлечения совпадений (z — число вхождений).

Проверка джокеров:

- $O(|T| \cdot |P|)$ для анализа C и проверки символов P в каждой позиции.

Пересечения:

- $O(z^2)$ для сравнения всех пар вхождений.

Итого: $O(|T| \cdot |P| + z^2)$

Пространственная сложность

Бор:

- transitions: $O(\sum |Qi|)$ состояний.
- output, suffLink, termLink: $O(\sum |Qi|)$.

Поиск:

- C: $O(|T|)$.
- results: $O(z)$.
- overlap_patterns: $O(k)$, где $k \leq |P|$.

Общая: $O(|T| + |P| + z)$, $z \leq |T|$.

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.