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. 3388 _____ Шубин П.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Задание

Вариант 4. Реализовать режим поиска, при котором все найденные образцы не пересекаются в строке поиска (т.е. некоторые вхождения не будут найдены; решение задачи неоднозначно).

Задача 1:

Вход:
Первая строка содержит текст T (1< |T|< 100000).

Вторая строка содержит число n (1< n< 3000). Каждая следующая из n строк содержит шаблон из набора P = { p_1, ldots, 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 по номеру позиции, затем по номеру шаблона.

Задача 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?c) с джокером ? встречается дважды в тексте *zabuccbababcax*.

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.

Выполнение работы

Для реализации задания использован алгоритм Ахо-Корасик. Алгоритм выполняет поиск подстрок в тексте с использованием конечного автомата, построенного на боре. Реализация поддерживает три режима работы:

- Поиск по нескольким шаблонам.
- Обработка шаблонов с wildcard-символом.
- Фильтрация неперекрывающихся вхождений.

Структуры данных

Node - Узел бора, содержащий:

- children map[rune]*Node — переходы к дочерним узлам по символам.
- suffixLink *Node — суффиксная ссылка для эффективного перехода при несовпадении.
- terminalLink *Node — терминальная ссылка для быстрого перехода к ближайшему терминальному узлу.
- patternIndices []int — индексы шаблонов, завершающихся в этом узле.
- patternLength int — длина соответствующего шаблона.
- id int — уникальный идентификатор узла.

AhoCorasick - Управляет построением и работой автомата:

- root *Node — корневой узел бора.
- patterns []string — список шаблонов для поиска.
- patternLengths []int — длины шаблонов (для фильтрации пересечений).
- nodeID int — счётчик идентификаторов узлов.

- verbose bool — флаг вывода отладочной информации.

Pair - Структура для хранения результатов поиска:

- Pos int — позиция вхождения в тексте.
- PatternIndex int — индекс шаблона (начиная с 1).

Методы и функции

Инициализация

NewAhoCorasick(patterns []string, verbose bool) - Конструктор, инициализирующий автомат:

- Создаёт корневой узел.
- Сохраняет длины шаблонов для фильтрации.
- Вызывает методы построения бора и ссылок.

buildTrie() - Строит бор из списка шаблонов:

- Добавляет узлы для каждого символа шаблонов.
- Фиксирует терминальные узлы с индексами шаблонов.

buildSuffixAndTerminalLinks() - Вычисляет суффиксные и терминальные ссылки:

- Использует BFS для обхода узлов.
- Суффиксные ссылки строятся по аналогии с алгоритмом Кнута-Морриса-Пратта.
- Терминальные ссылки указывают на ближайший терминальный узел.

Поиск

Search(text string) []Pair - Ищет все вхождения шаблонов в текст:

- Перемещается по автомату, используя суффиксные ссылки при несовпадениях.

- Собирает результаты при проходе через терминальные ссылки.

- Возвращает отсортированные по позициям результаты.

splitPattern(pattern string, wildcard rune) (substrings []string, positions []int) -Разбивает шаблон с джокерами на подстроки:

- Удаляет wildcard-символы.

- Возвращает подстроки и их позиции в исходном шаблоне.

filterNonOverlapping(pairs []Pair, patternLengths []int) []Pair - Фильтрует непересекающиеся вхождения:

- Преобразует позиции в интервалы [start, end].

- Жадным алгоритмом выбирает интервалы без перекрытий.

Вспомогательные функции

PrintAutomaton() - Выводит структуру автомата в виде:

- Идентификаторы узлов.

- Суффиксные и терминальные ссылки.

- Переходы по символам.

- Терминальные шаблоны в узлах.

main() - Реализует интерфейс командной строки:

- Предлагает выбор режима работы.

- Обрабатывает ввод данных.

- Запускает соответствующий алгоритм поиска.

Анализ сложности алгоритма

Временная сложность

Построение бора (Trie)

Сложность: O(L), где L — суммарная длина всех шаблонов.

Обоснование: Каждый символ каждого шаблона обрабатывается ровно один раз. В коде это реализовано в цикле по шаблонам и их символам в методе buildTrie().

Построение суффиксных и терминальных ссылок

Сложность: O(L·Σ), где Σ — размер алфавита.

Обоснование: Для каждого узла выполняется поиск суффиксной ссылки через переходы. В худшем случае для каждого узла требуется проверка всех символов алфавита. Это реализовано в цикле BFS в методе buildSuffixAndTerminalLinks().

Поиск вхождений в текст

Сложность: O(M+Z), где M — длина текста, Z — общее количество вхождений.

Обоснование: Каждый символ текста обрабатывается один раз (O(M)), а проверка терминальных ссылок для каждого вхождения выполняется за константное время (O(Z)). Это реализовано в методе Search().

Дополнительные операции

Фильтрация непересекающихся вхождений:

Сложность: O(ZlogZ).

Обоснование: Сортировка интервалов занимает O(ZlogZ), затем линейный проход (O(Z)). Это реализовано в функции filterNonOverlapping().

Разделение шаблона с джокерами:

Сложность: O(K), где K — длина шаблона.

Обоснование: Линейный проход по символам шаблона. Это реализовано в функции splitPattern().

Итоговая временная сложность:

Основной алгоритм (Ахо-Корасик): O(L·Σ+M+Z).

Режим с джокерами: O(L·Σ+M+Z+K).

Режим непересекающихся вхождений: O(L·Σ+M+ZlogZ).

Сложность по памяти

Бор: O(L), где L — суммарная длина шаблонов.

Обоснование: Каждый узел хранится в памяти ровно один раз.

Хранение вхождений: O(Z), где Z — количество найденных вхождений.

Обоснование: Результаты поиска сохраняются в массиве.

Дополнительные структуры для режима с джокерами: O(M), где M — длина текста.

Обоснование: Массив C для подсчёта совпадений занимает O(M).

Итоговая сложность по памяти:

O(L+Z+M)

Тестирование:

Input	Output
ababa 2 aba ba	1 1 2 2 3 1 4 2

Таблица 1. Тестирование решения задания 1

Input	Output
ACGTTACA A?G? ?	1

Таблица 2. Тестирование решения задания 2

Input	Output
AAABBBCCC 3 AAA AB BBBC	1 4

Таблица 3. Тестирование решения задания 3

Выводы:

В ходе работы был разработан и протестирован алгоритм для поиска вхождений шаблона с джокером, без джокера, с режимом непересекающихся вхождений. Алгоритм использует автомат Ахо-Корасик для эффективного поиска подстрок. Тестирование показало, что реализация алгоритма верна.