

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 $P = \{ p_1, \dots, p_n \}$ ($1 < |p_i| < 75$).

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 `*zabucsbababcsax*`.

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:

- Создает 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.

- Обработывает ввод данных.

- Запускает соответствующий алгоритм поиска.

Анализ сложности алгоритма

Временная сложность:

- Построение бора (trie): $O(L)$, где L — суммарная длина всех паттернов. Каждый символ каждого паттерна обрабатывается за $O(1)$ в среднем (благодаря хэш-таблицам **map[rune]*Node**).
- Построение суффиксных и терминальных ссылок: $O(L \cdot k)$, где k — размер алфавита. Для каждой из $O(L)$ вершин происходит поиск суффиксных ссылок, который может потребовать до $O(k)$ шагов в худшем случае.
- Поиск в тексте: $O(N+t)$, где N — длина текста, t — общее количество вхождений паттернов. Каждый символ текста обрабатывается за $O(1)$ в среднем (благодаря суффиксным ссылкам), а проверка терминальных ссылок занимает $O(t)$.

Итоговая временная сложность: $O(L \cdot k + N + t)$.

Пространственная сложность:

- Хранение бора: $O(L \cdot k)$. Каждая вершина содержит **map[rune]*Node**, который занимает $O(k)$ памяти, а количество вершин пропорционально L .
- Хранение паттернов и метаданных: $O(L)$ для сохранения длин паттернов и индексов.

Итоговая пространственная сложность: $O(L \cdot k)$.

Тестирование:

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

Таблица 1. Тестирование решения задания 1

Input	Output
ACGTTACA	1
A?G?	
?	

Таблица 2. Тестирование решения задания 2

Input	Output
AAABBBCCC	1
3	4
AAA	
AB	
BBBC	

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.