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: 7

Студент гр. 3388 _____ Трунов Б.Г.

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 $P=\{p\_1,\ldots,p\_n\}$ $1\leq|p\_i|\leq75$.

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'а для точного поиска образцов:

Алгоритм Ахо-Корасик предназначен для точного поиска множества подстрок (образцов) в тексте за линейное время. Он объединяет идеи *trie*-дерева, *failure*-ссылок (аналог префикс-функции в алгоритме Кнута-Морриса-Пратта) и терминальных ссылок, что позволяет обрабатывать текст за один проход. Алгоритм особенно эффективен, когда требуется найти множество образцов одновременно.

Шаги алгоритма

- Построение бора шаблонов
    - Создается корневой узел автомата.
    - Для каждого шаблона добавляется цепочка переходов по символам:
        - Начинаем с корня
        - Для каждого символа в шаблоне:
            - Если переход по символу отсутствует, создается новый узел.
        - В конце шаблона узел помечается номером шаблона(добавляется в *outputs*).
- Расчет *fail*-ссылок(алгоритм *BFS*)
    - Для корневых узлов(прямых потомков корня):
        - *fail* устанавливается в корень
        - Добавляются в очередь для обработки
    - Для остальных узлов:
        - *fail* вычисляется через *fail*-ссылку родителя:
            - Поднимаемся по *fail*-ссылкам, пока не найдем узел с переходом по текущему символу.
            - Если переход найден, *fail* ребенка указывает на него. Иначе на корень.
        - *terminal* устанавливается на ближайший терминальный узел через цепочку *fail*-ссылок.

- Поиск вхождений в тексте
    - Инициализация текущего узла корнем.
    - Для каждого символа текста:
        - Переход по автомату с учетом *fail*-ссылок:
            - Если переход невозможен, поднимаемся по *fail* до корня.
        - Проверка текущего узла и всех узлов в цепочке *terminal*:
            - Сбор всех найденных шаблонов из *outputs*.
- Вывод результатов
    - Возвращаемые вхождения сортируются по позиции в тексте.
    - Вывод в формате: позиция начала шаблона и его номер.

.

Описание функций и структур:

- Класс *AhoNode* – базовая единица автомата Ахо-Корасик.
    - Поля:
        - *trans : Dict[str, int]* – словарь перехода по символам.
        - *fail : int* – *fail*-ссылка. Указывает на узел, к которому нужно перейти при несовпадении символа.
        - *terminal : int* – ссылка на ближайший терминальный узел в цепочке *fail*-ссылок. Оптимизирует поиск вхождений.
        - *outputs : List[int]* – список номеров шаблонов, которые заканчиваются в этом узле.
- *build_automaton(patterns: List[str]) -> List[AhoNode]* – строит автомат на основе списка шаблонов.
    - Первый этап – построение бора.
    - Второй этап - расчет *fail*-ссылок.
- *search(text: str, nodes: List[AhoNode], patterns: List[str]) -> List[Tuple[int, int]]* – ищет все вхождения шаблонов в тексте.
- *visualize_automaton(nodes: List[AhoNode], filename: str = 'automaton')* – генерирует графическое представление автомата через *Graphviz.*
    - Синие стрелки – переходы по символам.
    - Красные пунктирные стрелки – fail-ссылки.
    - Узлы отображают: переходы(trans), fail, outputs.
- *read_input() -> Tuple[str, List[str]]* – читает входные данные из *stdin*. Возвращает: кортеж (текст, список_шаблонов).
- *process_data(text: str, patterns: List[str]) -> List[Tuple[int, int]]* – основная логика:
    - Строит автомат через *build_automaton.*
    - Визуализирует через *visualize_automaton.*
    - Запускает поиск через *search.*
- *print_results(result: List[Tuple[int, int]])* – вывод результатов поиска.

Оценка сложности алгоритма:

Временная сложность

Построение бора:

- *O(M)*, где *M* – суммарная длина всех шаблонов. Каждый символ шаблона обрабатывается ровно один раз.

Построение автомата:

- *O(M)*, где *M* – суммарная длина всех шаблонов. Обработка всех узлов через *BFS*. Для каждого узла:
  - Проверка переходов (*trans*), которые существуют (не по всему алфавиту).
  - Вычисление *fail* и *terminal* через уже обработанные узлы.

Поиск:

- Временная сложность: *O(N + K)*, где:
  - *N* — длина текста,
  - *K* — общее количество найденных вхождений. Каждый символ текста обрабатывается за константное время (амортизированно), благодаря оптимизации с *terminal*.

Итог: *O(M+N+K)*

Пространственная сложность

- Хранение узлов автомата:
  - В худшем случае (шаблоны не имеют общих префиксов): Количество узлов M.
  - В лучшем случае (шаблоны полностью пересекаются по префиксам):
  - Количество узлов равно длине самого длинного шаблона.
  - Количество узлов ≤ *M*, где *M* — суммарная длина всех шаблонов.
  - Каждый узел хранит переходы в виде словаря, размер словаря не больше чем размер алфавита(|A|).
- *K* – количество вхождений шаблонов в тексте.

Итого: *O(M*|A|+K)*.

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-Корасика с джокерами:

Алгоритм Ахо-Корасика 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

- Разбиение шаблона на сегменты
    - **Цель:** Выделить части шаблона между *wildcard*-символами.
    - **Действия:**
        - Шаблон сканируется посимвольно.
        - Последовательности символов между *wildcard*-символами сохраняются как сегменты.
        - Для каждого сегмента запоминается его позиция в исходном шаблоне.
- Построение автомата Ахо-Корасик
- Поиск сегментов в тексте
    - Цель: Найти все стартовые позиции сегментов в тексте.
    - Действия:
        - Текст обрабатывается посимвольно.
        - Для каждого символа осуществляется переход по автомату с учётом fail-ссылок.
        - При обнаружении терминального узла (сегмента) сохраняются позиции начала этого сегмента в тексте.
- Агрегация результатов
    - **Цель:** Определить позиции, где все сегменты шаблона совпадают с учётом их расположения.

- **Действия:**
    - Для каждой возможной стартовой позиции шаблона в тексте подсчитывается, сколько сегментов начинаются в правильных местах.
    - Если количество совпадений равно числу сегментов, позиция считается валидной.
- Фильтрация и вывод результатов
    - **Цель:** Убрать дубликаты и отсортировать позиции.

Описание функций и структур:

prepare_patterns(...) - Подготавливает сегменты для поиска.

split_pattern(pattern: str, wildcard: str) -> List[Tuple[str, int]] - Разбивает шаблон с wildcard на сегменты.

find_occurrences(...) -> List[int] - Ищет все стартовые позиции сегментов в тексте.

process_results(...) -> List[int] - Фильтрует позиции, где все сегменты совпадают.

Оценка сложности алгоритма:

- Временная сложность:
  - Построение автомата: *O(M)*, где *M* — суммарная длина всех сегментов шаблона (после удаления *wildcard*-символов). Каждый символ каждого сегмента обрабатывается один раз при построении *trie* и *fail*-ссылок.
- Поиск сегментов в тексте : *O(N + K)*, где:
  - *N* — длина текста,
  - *K* — общее количество найденных вхождений сегментов. Каждый символ текста обрабатывается за амортизированное константное время благодаря оптимизации с *terminal*.
- Обработка результатов: *O(N + C)*, где:
  - *N* — длина текста.
  - *C* — количество предварительно найденных стартовых позиций. *B* худшем случае (например, если все символы текста — стартовые позиции) сложность достигает *O(N)*.

Итоговая временная сложность: *O(M + N + K)*.

- Пространственная сложность:
  - Автомат (*nodes*): *O(M)*. Хранение узлов с переходами (*trans*), *fail*-ссылками и *outputs*.
  - Счётчик стартовых позиций (*process_results*): *O(N)*, где *N* — длина текста. Массив *counter* размером *max_start + 1*.
  - Хранение результатов поиска: *O(K)* для списка *occurrences*.

Итоговая пространственная сложность: *O(M + N + K).*

Тестирование

Таблица 2. Тестирование.

Входные данные	Выходные данные
AAAAA A*A *	1 2 3
ACGTACGT *CG* *	1 5
ACTANCA A$$A$ $	1
NTAG T*G *	2

Вывод

В ходе лабораторной работы были написаны программы с использованием алгоритма Ахо-Корасика. Также дополнительно было сделано: визуализация автомата.