Лабораторная работа № 4 по курсу дискретного анализа: Поиск образца в строке

Выполнил студент группы М8О-209Б-23 *Кривошапкин Егор*.

Условие

Кратко описывается задача:

1. Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.
2. *Вариант алгоритма:* Поиск одного образца-маски: в образце может встречаться «джокер» (представляется символом ? — знак вопроса), равный любому другому символу. При реализации следует разбить образец на несколько, не содержащих «джокеров», найти все вхождения при помощи алгоритма Ахо-Корасик и проверить их относительное месторасположение.
3. *Вариант алфавита:* Числа от 0 до 2^32-1.
4. Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

Метод решения

Алгоритм Ахо–Корасик
Для поиска шаблона с джокерами (?) я использовал алгоритм Ахо–Корасик, который подходит для одновременного поиска нескольких подстрок (или, в данном случае, последовательностей чисел) в тексте. Поскольку шаблон может содержать как конкретные числа, так и джокеры, он предварительно разбивается на сегменты — подпоследовательности чисел без ?. Каждому сегменту сопоставляется смещение относительно начала шаблона.
Все сегменты добавляются в префиксное дерево (trie), после чего строятся ссылки неудач (failure links), позволяющие эффективно переходить к другому префиксу при несовпадении. Во время прохода по тексту автомат находит вхождения каждого сегмента и по сохранённым смещениям восстанавливает возможные позиции начала полного шаблона.
Чтобы засчитать вхождение, необходимо, чтобы в одной и той же позиции текста совпали все сегменты шаблона — при этом учитываются и относительные позиции, и длина всего шаблона.
Алгоритм работает за линейное время от длины текста и эффективно справляется с большими объёмами данных. Он позволяет обрабатывать произвольные 32-битные числа как алфавит, не привязываясь к строковому представлению.

Описание программы

Вся логика программы находится в файле main.cpp. В нем реализована структура префиксного дерева и функции для препроцессинга и выполнения алгоритма поиска

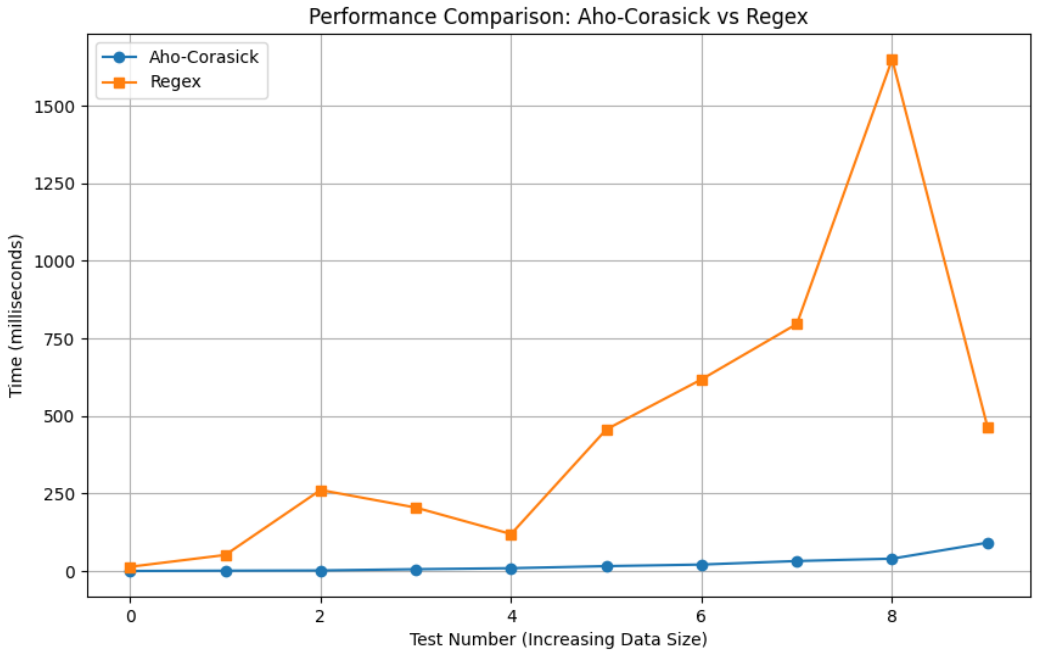
Ахо-Корасик.
В функции main реализована логика считывания ввода пользователя по условию задачи, построение дерева, поиск совпадений и вывод позиций начала шаблона в тексте.

Дневник отладки

1. WA на тесте 9. Добавлена обработка случаев, когда одинаковые сегменты встречаются в шаблоне несколько раз, теперь для каждого сегмента хранится не одно значение смещения, а вектор из всех вариантов смещения.

Тест производительности

Для теста производительности я сравнил скорость работы Ахо-Корасик и алгоритма на основе регулярных выражений.



По графику видно, что алгоритм Ахо-Корасик быстрее и стабильнее в задачах поиска шаблонов с подстановочными знаками благодаря своей линейной временной сложности и прямой работе с числовыми данными. В то же время, подход на основе регулярных выражений медленнее и менее предсказуем из-за необходимости преобразования данных и возможного увеличения временной сложности в некоторых случаях.

Выводы

В результате проделанной лабораторной работы я реализовал алгоритм Ахо-Корасик для поиска образца с джокерами в большом тексте. Алгоритм Ахо-Корасик обеспечивает линейную сложность O(n + m + z), где n — длина текста, m — суммарная длина образцов, z — число совпадений, благодаря построению префиксного дерева (Trie) и failure-ссылок, позволяющих эффективно обрабатывать несовпадения и находить все вхождения за один проход.

Преимущества алгоритма Ахо-Корасика:

- Линейная сложность поиска, независимо от количества джокеров и структуры текста.
- Эффективная обработка множественных подстрок и джокеров за один проход текста.
- Предсказуемое поведение и высокая производительность на больших данных.

Недостатки алгоритма Ахо-Корасика:

- Требуется дополнительная память O(m) для хранения Trie и failure-ссылок.
- Сложность реализации выше, чем у более простых алгоритмов, таких как Кнут-Моррис-Пратт. Для работы с экстремально большими текстами или множеством образцов Ахо-Корасик предпочтительнее, чем КМП, благодаря своей способности обрабатывать несколько подстрок одновременно, но для простых задач без джокеров может быть избыточным.