Алгоритмы поиска подстрок

Борисов Иван Максимович

2025.11.29

1 Алгоритм Рабина-Карпа

1.1 Принцип работы

Алгоритм Рабина-Карпа использует хеширование для эффективного поиска подстрок. Основная идея:

1. Вычисляется хеш-значение искомой подстроки `target`
2. Вычисляется хеш-значение для каждого окна длины m в тексте `text`
3. При совпадении хешей выполняется проверка на точное совпадение подстрок (для избежания коллизий)
4. Для эффективного пересчета хеша используется скользящее хеширование

1.2 Оценка сложности

Пусть n - длина текста, m - длина таргета.

1. Предварительное вычисление хешей: $O(m)$
2. Основной цикл по тексту: $O(n - m + 1) = O(n)$ операций
3. Каждая операция пересчета хеша: $O(1)$ (скользящее хеширование)
4. В худшем случае при многих коллизиях: $O(nm)$ы
5. В среднем случае: $O(n + m)$

Таким образом:

$$\boxed{T(n) = O(n + m) \text{ (в среднем)}, \quad O(nm) \text{ (в худшем)}}$$

1.3 Объяснение Rolling Hash

Используется полиномиальный хеш:

$$H(s) = (s_0 \cdot b^{m-1} + s_1 \cdot b^{m-2} + \cdots + s_{m-1} \cdot b^0) \mod p,$$

где b - основание системы счисления, p - простое число, s_i - код символа в позиции i.

1.3.1 Пересчет хеша

Для окна $[i, i+m-1]$:

$$H_i = (t_i \cdot b^{m-1} + t_{i+1} \cdot b^{m-2} + \cdots + t_{i+m-1} \cdot b^0) \mod p$$

Для окна $[i+1, i+m]$:

$$H_{i+1} = (t_{i+1} \cdot b^{m-1} + t_{i+2} \cdot b^{m-2} + \cdots + t_{i+m} \cdot b^0) \mod p$$

Тогда:

$$\begin{aligned} H_{i+1} &= [H_i - t_i \cdot b^{m-1}] \cdot b + t_{i+m} \mod p \\ &= (H_i \cdot b - t_i \cdot b^m + t_{i+m}) \mod p \end{aligned}$$

В коде:

```
text_hash = (base * (text_hash - ord(text[i]) * h)
             + ord(text[i + m])) % prime
```

Итого: пересчет хеша за $O(1)$ вместо $O(m)$ - главный выигрыш алгоритма.

2 Алгоритм Кнута-Морриса-Пратта

2.1 Принцип работы

Алгоритм KMP использует специальную префикс-функцию для избежания возвратов в тексте:

1. Строится таблица префикс-функции `kmp_table` для таргета
2. Префикс-функция определяет максимальную длину собственного суффикса, совпадающего с префиксом
3. При несовпадении символов используется таблица для определения следующей позиции сравнения
4. Текст просматривается только один раз без возвратов

2.2 Оценка сложности

Пусть n - длина текста, m - длина таргета.

1. Построение таблицы префикс-функции: $O(m)$
2. Поиск подстроки в тексте: $O(n)$
3. Каждая операция сравнения и обновления индексов: $O(1)$

Таким образом:

$$\boxed{T(n) = O(n + m)}$$

2.3 "Доказательство"линейной сложности

Для построения таблицы префикс-функции:
$T_{\text{table}}(m) = O(m)$ (каждый символ обрабатывается конст. число раз)
Для поиска в тексте:
$T_{\text{search}}(n) = O(n)$ (индекс i движется только вперед)
Общая сложность:
$T(n, m) = T_{\text{table}}(m) + T_{\text{search}}(n) = O(m) + O(n) = O(n + m)$