Алгоритм Рабина-Карпа

Николай Пономарев

Проблема

- Поиск шаблона в строке достаточно частая проблема в информатике
- Самым простым решением является последовательное «прикладывание» шаблона к строке
- Сложность такого решения $O(nm)$
- Можно оптимизировать выходом из цикла при первом несовпадении, но гарантий на значительное ускорение нет

Чутоку теории

В информатике существуют так называемые хеш-функции, которые позволяют преобразовать произвольные данные в конечное множество значений

- У одинаковых входных данных одинаковый хеш
- Однако обратное неверно, т.к. происходят коллизии

Коллизия – совпадение результата хеш-функции для разных входных данных

- У «хороших» хеш-функций вероятность коллизий достаточно мала

Задумка

Алгоритм Рабина-Карпа был придуман Ричардом Карпом и Майклом Рабином в 1987 году.
Идея алгоритма заключается в подсчете хеша для подстроки и сравнении его с хешем образца, и только в случае равенства посимвольно сравнивать подстроку и образец

Реализация

```python
def rabin_karp(s: str, pattern: str) -> int:
    n = len(s)
    m = len(pattern)
    pattern_hash = hash(pattern)
    for i in range(n - m):
        substring = s[i : i + m]
        substring_hash = hash(substring)
        if pattern_hash == substring_hash:
            if pattern == substring:
                return i
    return -1
```

Сложность

- Хеширование и посимвольное сравнение имеют сложность $O(m)$
- Хеширование производится на каждом шаге цикла, итоговая сложность $O(nm)$

Но такая же сложность и у «наивного» алгоритма!
Для решения этой проблемы необходимо использовать скользящий (rolling) хеш.

Скользящий хеш

Скользящий хеш — это такая хеш-функция, для пересчета которой достаточно знать ее предыдущее значение и что изменилось в данных.

Пример

Самый простой хеш, который складывает все коды символов в строке $s = a_1 a_2 ... a_n$:

$$h(s) = \sum_{i=0}^{n} a_i,$$

является скользящим. После сдвига на 1 вправо его можно пересчитать за константное время:

$$h(s[2..n+1]) = h(s[1..n]) - a_1 + a_{n+1}$$

Полиномиальный хеш

Полиномиальный хеш является достаточно простым скользящим хешем, однако он обеспечивает достаточно низкую вероятность коллизий.

$$h(s) = \left(\sum_{i=1}^{n} s[i]x^{n-i}\right) \bmod q$$

На практике взятие остатка обычно выполняется после каждой операции:

```
def polynomial_hash(s: str, x: int, q: int) → int:
    hash = 0
    for a in s:
        hash = (hash * x) % q
        hash = (hash + a) % q
    return hash
```

Полиномиальный хеш. Сдвиг

За основу возьмем строку $s = a_1 a_2 a_3 a_4 \ldots a_n$
Посчитаем хеш для подстроки $s_1 = a_1 a_2 a_3$:

$$h(s_1) = \left[\left(\left[\left(\left[\left(a_1 x\right) \bmod q + a_2\right] \bmod q\right) x\right] \bmod q\right) + a_3\right] \bmod q$$

Теперь сдвинемся на 1 символ вправо и подсчитаем хеш для подстроки $s_2 = a_2 a_3 a_4$:

$$h(s_2) = \left[\left(h(s_1) + q - \left[\left(a_1 x^2\right) \bmod q\right]\right) x + a_4\right] \bmod q$$

Полиномиальный хеш. Выбор q и x

Вариант Рабина и Карпа:

- фиксированный $x = 2$
- случайное простое число $q \in [2, n^3]$, где n – длина строки

Вариант Дитзфелбингера и др.:

- случайное $x \in (0, q - 1]$
- фиксированное простое q (например простое число Мерсенна: $2^{31} - 1$ или $2^{61} - 1$)

Сложность с «хорошим» хешем

- Вычисление скользящего хеша занимает $O(n)$
- Сравнение строк при совпадении хеша – $O(m)$

Тогда сложность алгоритма $O(n + m)$

Применение

- Алгоритм легко модифицируется для поиска набора образцов в тексте сохраняя хорошее время работы
- Из-за этого часто применяется для проверки на плагиат

Материалы

- Русская Википедия
- Английская Википедия
- Rabin M. O., Karp R. M. Efficient randomized pattern-matching algorithms – статья Рабина и Карпа с описанием алгоритма
- Dietzfelbinger M., Gil J., Matias Y., Pippenger N. Polynomial hash functions are reliable – статья Дитзфелбингера и др. о полиномиальном хеше
- Rabin M. O. Fingerprinting by random polynomials – статья про хеш Рабина