Строки – 2: наивный поиск и КМП

Denis Bakin

Подстроки: определения

- **Строка** — последовательность символов. Индексация с 0, $|s|$ — длина строки.
- **Подстрока** — непрерывный фрагмент: $s[i..j]$ (обычно i и j включительно).
- **Префикс** — подстрока, начинающаяся в 0.
- **Суффикс** — подстрока, заканчивающаяся в $|s| - 1$.

Задача: найти подстроку в тексте

Дано:

- текст t
- образец (pattern) s

Нужно найти все позиции i, где $t[i...(i + |s| - 1)] = s$.

Наивный алгоритм поиска подстроки

Идея:

- перебираем все позиции начала i в тексте
- сравниваем s с фрагментом $t[i...(i + |s| - 1)]$ посимвольно

Наивный алгоритм: псевдокод

```
for (int i = 0; i + s.size() <= t.size(); ++i) {
    bool ok = true;
    for (int j = 0; j < s.size(); ++j) {
        if (t[i + j] != s[j]) {
            ok = false;
            break;
        }
    }
    if (ok) {
        // найдено вхождение на позиции i
    }
}
```

Наивный алгоритм: сложность

- В худшем случае для каждой позиции i сравниваем до $|s|$ символов.
- Всего позиций $\approx |t|$.

$$\mathcal{O}(|t| \cdot |s|)$$

Хотим быстрее — **линейно** по $|t|$.

Префикс-функция: определение

Пусть дана строка s.

$\pi[i]$ — длина наибольшего префикса s, который является суффиксом строки $s[0..i]$.

- То есть максимальная длина k, такая что:
  - $s[0..k-1] = s[i-k+1..i]$
- Всегда: $0 \le \pi[i] \le i$.

Префикс-функция: пример

Строка: `aabaataabaa`

i	0	1	2	3	4	5	6	7	8	9	10
$s[i]$	a	a	b	a	a	t	a	a	b	a	a
$\pi[i]$											

Префикс-функция: пример

Строка: `aabaataabaa`

i	0	1	2	3	4	5	6	7	8	9	10
$s[i]$	a	a	b	a	a	t	a	a	b	a	a
$\pi[i]$	0	1	0	1	2	0	1	2	3	4	5

- “ожидаемый символ”

Интуиция: «ожидаемый символ»

Если мы знаем $\pi[i-1] = k$, то:

- у нас есть совпадение префикса длины k
- ожидаемый символ, который продолжит совпадение — это $s[k]$

Интуиция: «ожидаемый символ»

Если мы знаем $\pi[i-1] = k$, то:

- у нас есть совпадение префикса длины k
- ожидаемый символ, который продолжит совпадение — это $s[k]$

Если $s[i] \neq s[k]$, то мы пытаемся уменьшить k, но не до нуля сразу, а «прыгаем» по уже посчитанным значениям префикс-функции.

Префикс-функция: алгоритм за $\mathcal{O}(n)$

```
s = "..."
p = [0, 0, ..., 0]
for i: 1 -> n
    k = p[i - 1]
    while s[k] != s[i] and k > 0: # пока фактический символ (s[i]) не совпал с ожидаемым
        k = p[k - 1]
    if s[k] == s[i]: # если мы все же нашли совпадение, то тогда считаем
        p[i] = k + 1 # длина найденного префикс + 1
    # а иначе значение префикс-функции -- ноль
```

- `while` работает амортизированно за $\mathcal{O}(n)$

Как π помогает искать подстроку?

Идея алгоритма КМП (Кнута-Морриса-Пратта):

- введем новую строку: $u = s + \# + t$

Как π помогает искать подстроку?

Идея алгоритма КМП (Кнута-Морриса-Пратта):

- введем новую строку: $u = s + \# + t$
- # — символ, который **не встречается** ни в s, ни в t

Как π помогает искать подстроку?

Идея алгоритма КМП (Кнута-Морриса-Пратта):

- введем новую строку: $u = s + \# + t$
- # — символ, который **не встречается** ни в s, ни в t
- считаем префикс-функцию для u

Как π помогает искать подстроку?

Идея алгоритма КМП (Кнута-Морриса-Пратта):

- введем новую строку: $u = s + \# + t$
- # — символ, который **не встречается** ни в s, ни в t
- считаем префикс-функцию для u
- если в какой-то позиции i получилось $\pi[i] = |s|$, значит, здесь заканчивается очередное вхождение s в t

КМП: индекс начала вхождения

Пусть:

- $u = s + \# + t$
- $|s| = m$

Если $\pi[i] = m$, то:

- вхождение заканчивается в t в позиции $(i - (m + 1))$
- начало вхождения в t:

$$(i - (m + 1)) - (m - 1) = i - 2m$$

То есть **начало** $= i - 2|s|$.

Сложность КМП

- Построение u: $\mathcal{O}(|s| + |t|)$
- Префикс-функция: $\mathcal{O}(|s| + |t|)$
- Сканирование результатов: $\mathcal{O}(|s| + |t|)$

$$\mathcal{O}(|t| + |s|)$$

Итоги

- Наивный поиск: $\mathcal{O}(|t| \cdot |s|)$
- Префикс-функция считает совпадения «префикс = суффикс» для всех префиксов
- КМП на основе π находит все вхождения за $\mathcal{O}(|t| + |s|)$