

# Конспект лекций по строковым алгоритмам

Глинских Георгий

19 июня 2025 г.



# Глава 1

## Поиск подстроки в строке

### 1.1 Вводные замечания

Строки – конечные последовательности символов над множеством, называемым алфавитом. Здесь и далее я буду полагать, что алфавит – множество чисел  $\{1, \dots, \sigma\}$  и что строки индексируются с  $1^1$ :

$$w = w(1)w(2) \dots w(n) = w(1, n).$$

Введем понятия периода и грани строки.

**Определение 1.1.1.** *Периодом строки назовем число  $p$ , такое, что*

$$\forall i : s(i) = s(i + p).$$

**Утверждение 1.1.1.** *Следующие определения равнозначны:*

1.  $p$  – период строки  $w$ .
2.  $s(p + 1, n)$  и суффикс, и префикс  $w$ .
3. Существует слова  $a, b$ :  $b$  префикс  $a$  и  $w = a^k b$ .

*Доказательство.*  $1 \Rightarrow 2$ . Посимвольно сравнивая,  $s(1, n - p) = s(p + 1, n)$ .

$2 \Rightarrow 3$ . Положим  $a = s(1, p)$ . Тогда видно, что  $s(p + 1, 2p) = s(1, p) = a$ , Аналогично

$$\forall l : s((l - 1)p + 1, lp) = a.$$

Остается взять  $b = s(kp + 1, n)$ , где  $k$  – наибольшее из чисел  $l$ .

$3 \Rightarrow 1$ . Очевидно.

□

**Определение 1.1.2.** *Гранью строки назовем суффикс, одновременно являющийся префиксом.*

---

<sup>1</sup>Так удобнее для математиков, но хуже для программистов.

**Утверждение 1.1.2.** Число  $0 < p \leq n$  – период  $s$  тогда  $s(1, n - p)$  – граница  $s$ .

*Доказательство.* Заметим, что  $s(1, n - p) = s(p + 1, n)$  по условию, и тогда достаточно воспользоваться пунктом 2 предыдущего утверждения.  $\square$

Важно мыслить объекты, которые обрабатываются алгоритмами, очень (очень!) большой размерности. Тогда будет появляться нужная интуиция.

## 1.2 Алгоритм Крошмора-Перрена

При помощи периода можно улучшить обычный алгоритм поиска подстроки в строке. Далее будем называть его Ord, его сложность  $T(\text{Ord}) = O(|\text{text}||\text{pat}|)$ .

Инвариант цикла в Ord – вхождения с началом в  $\text{text}(1, i)$  рассмотрены.

Пусть теперь известен  $p$  – минимальный период строки  $\text{pat}$ . Тогда этот алгоритм можно улучшить для нахождения серии вхождений (вхождения, расположенные достаточно близко друг к другу). Заметим, что вхождения находятся на расстоянии не меньше  $p$  (иначе противоречие с минимальностью периода), следовательно, при нахождении вхождения, можно не смотреть на следующие  $p$  возможных позиций. Получим алгоритм Per.

Инвариант цикла в Per – для нахождения вхождения остается рассмотреть  $|\text{pat}| - b + 1$  позиций. Иными словами,  $\text{pat}(1, b) = \text{text}(i, i + b)$ .

Теперь рассмотрим наконец алгоритм Крошмора-Перрена. Его будем обозначать TW: в англоязычной литературе его называют two-way алгоритмом. Он быстрый:  $T(\text{TW}) = O(|\text{text}| + |\text{pat}|)$ ,  $S(\text{TW}) = O(1)$ .

**Определение 1.2.1.** Локальным периодом разложения  $w = ab$  назовем число  $q$  такое, что

$$\forall i \in \{|a| - q, \dots, |a|\} : s(i) = s(i + q).$$

Заметим, что число  $p$  – период  $w$  будет локальным периодом для любого разложения.

**Определение 1.2.2.** Критическим назовем разложение  $w = ab$ , для которого минимальный локальный период совпадает с периодом строки  $w$ .

**Утверждение 1.2.1.** У любой строки есть критическое разложение  $w = ab$ :  $|a| = u < p$ , где  $p$  – период  $w$ .

*Доказательство.* Заметим сначала, что любой локальный период  $q$  для критического разложения  $(a, b)$  с условием  $q < |b|$  делится на  $p$ . Из критичности,  $q \geq p$ . Тогда  $(xy)^k x$  с  $|xy| = p, |x| < p$  будет префиксом  $b$ . Но тогда  $x$  будет суффиксом  $a$  и префиксом  $b$ . Пришли к противоречию: удалось найти локальный период  $|x| < p \leq q$ .  $\square$

Рассмотрим алгоритм TW1. Идейно он работает так: зафиксируем критическое разложение  $\mathbf{pat} = ab$  как в утверждении 1.2.1 и сначала пытаемся найти вхождение  $b$  слева направо, после чего проверим справа налево, будет ли это вхождение иметь вид  $ab = \mathbf{pat}$ .

В начале каждой итерации главного цикла соблюдается инвариант  $\mathbf{pat}(1, b) = \mathbf{text}(i, i + b)$  (аналогично алгоритму Per).

**Утверждение 1.2.2.** *Алгоритм TW1 находит все вхождения  $\mathbf{pat}$  в  $\mathbf{text}$  со сложностью*

$$T(\text{TW1}) = O(|\mathbf{text}|), \quad S(\text{TW1}) = O(1).$$

*Доказательство.* Допустим,  $i$  пробегает позиции  $\{i_k\}$ , и в позиции  $i_k < i' < i_{k+1}$  было вхождение, которое мы не выведем. Одно из двух: мы при сканировании перескочили этот индекс или мы не распознали вхождение по этому индексу.

Если мы перешли к  $i_{k+1}$  через строку 8, то мы не можем пропустить вхождение в силу критичности разложения и  $|a| < p$ . Формально,  $i' - i_k < d - u$ , тогда по утверждению 1.2.1,  $i - i_k = Ap$ . Следовательно,  $\mathbf{text}(i + c) \neq \mathbf{pat}(c) = \mathbf{pat}(c - Ap) = \mathbf{text}(i' - c - i')$ . Противоречие.

Если мы перешли к  $i_{k+1}$  через строку 15, то мы не можем пропустить в силу того, что  $p$  – наименьший локальный период.

Итак, алгоритм находит все вхождения.

Мы используем лишь несколько переменных, откуда  $S(\text{TW1}) = O(1)$ . Для оценки времени работы  $T(\text{TW1})$ , заметим, что в 6 и 12 строке мы не касаемся символа дважды.  $\square$

Далее необходимо найти разложение и период. Идейно надо сделать пользоваться алгоритмом TW2.

**Утверждение 1.2.3.** *Алгоритм TW3 находит все вхождения  $\mathbf{pat}$  в  $\mathbf{text}$  со сложностью*

$$T(\text{TW1}) = O(|\mathbf{text}|), \quad S(\text{TW1}) = O(1).$$

*Доказательство.* Ясно, что  $p' < p$ . Причем  $p'$  – период  $\mathbf{pat}$  тогда верно условие в строке 2. Остальное доказательство повторяет доказательство утверждения 1.2.2.  $\square$

**Утверждение 1.2.4.** *В алгоритме TW2*

$$q = \max\{|a|, |b|\} + 1 < p.$$

Рис. 1.1: Обычный алгоритм поиска подстроки в строке (Ord)

```

1  i = 1
2  пока i + |pat| <= |text|: # O(|text|)
3      если text(i, i + |pat|) = pat(1, |pat|): # O(|pat|)
4          верни i
5      i += 1

```

Рис. 1.2: Улучшенный алгоритм поиска подстроки в строке (Per)

```

1  i = 1, b = 1
2  пока i + |pat| <= |text|: # O(|text| / p)
3      c = max { c : pat(b, c) = text(i + b, i + c) } # O(|pat|)
4      если c < |pat|:
5          i += 1, b = 1
6      иначе: # c = |pat|
7          верни i
8      i += p, b = |pat| - p + 1

```

Рис. 1.3: Алгоритм TW1

```

1  # вход: text: str, pat: str, u: num, p: num
2  # ограничения: u < p, u критическая для pat, p - период pat
3  i = 1, b = 1
4  пока i + |pat| <= |text|:
5      c = max(u, b)
6      d = max { d : pat(c, d) = text(i + c, i + d) }
7      если d < |pat|:
8          i += d - u
9          b = 1
10     иначе: # d = |pat|
11         c = u - 1
12         d = min { d >= b : pat(d, c) = text(i + d, i + c) }
13         если d < b:
14             верни i
15         i += p
16         b = |pat| - p + 1

```

Рис. 1.4: Алгоритм TW2

```
1  # вход: крит. pat = ab, |a| < p. p' - период b.  
2  если a - суффикс b(1, p'):  
3      p = p'  
4      алгоритм TW1  
5  иначе:  
6      q = max { |a|, |b| } + 1  
7      алгоритм TW3.
```

Рис. 1.5: Алгоритм TW3

```
1  # вход: text: str, pat: str, u: num, q: num  
2  # ограничения: u < p, u критическая для pat  
3  i = 1  
4  пока i + |pat| <= |text|:  
5      c = u  
6      d = max { d : pat(c, d) = text(i + c, i + d) }  
7      если d < |pat|:  
8          i += d - u  
9      иначе: # d = |pat|  
10         c = u - 1  
11         d = min { d >= 0 : pat(d, c) = text(i + d, i + c) }  
12         если d = 0:  
13             верни i  
14         i += q
```

*Доказательство.* Известно, что  $|a| < p$ . Достаточно показать, что  $|b| < p$ . Это так в силу того, что иначе если  $p' | p$ , то  $p'$  – локальный период. Иначе найдется период еще меньше, чем  $p'$ .  $\square$

Займемся необходимой предобработкой. Обозначим через  $<$  лексикографический порядок символов, а через  $\leq$  – обратный к лексикографическому. Продолжим их на строки.

**Утверждение 1.2.5** (Волшебное разложение). *Пусть  $\text{pat} = ab = cd$  где  $b$  и  $d$  – лексикографически максимальные по  $<$  и  $\leq$  суффиксы. Если  $|b| < |d|$ , то  $\text{pat} = ab$  критичное. Иначе  $\text{pat} = cd$  критичное.*

*Доказательство.* Если строка из одинаковых символов, то очевидно. Иначе не умаляя общности, пусть  $|b| < |d|$ .

Заметим, что локальный период разложения  $\text{pat} = ab$  больше  $|a|$ . Ведь в противном случае можно прийти к противоречию с максимальной  $b$ .

Осталось показать, что  $q$  – минимальный период для разложения  $\text{pat} = ab$  будет также периодом  $\text{pat}$ . Используем такой факт: для любых строк  $x, y$ :  $x < y$  и  $x \leq y$ ,  $x$  будет префиксом  $y$ . Используем его для строк  $\text{pat}(|c| + q, \cdot)$  и  $\text{pat}(|c|, \cdot)$ .

Очевидно,  $\text{pat}(|c| + q, \cdot) \leq \text{pat}(|c|, \cdot)$  в силу выбора  $c$ . По порядку  $<$  первые символы совпадают, а для остальных порядков, как между  $\text{pat}(|a|, \cdot) = b$  и  $\text{pat}(|a| + q, \cdot)$ . Получили требуемое условие.  $\square$

Осталось найти максимальные суффиксы по заданным порядкам. Для этого используем модификацию алгоритма Дюваля (TW4). Идейно: читаем строку слева направо и для каждого префикса находим максимальный суффикс и его период.

**Утверждение 1.2.6** (Дюваль). *Пусть  $\text{pat}(i, k)$  – максимальный суффикс строки  $\text{pat}(1, k)$ .*

1. Если  $\text{pat}((k+1) - p) = \text{pat}(k+1)$ , то  $\text{pat}(i, k+1)$  новый суффикс, его период  $p$ .
2. Если  $\text{pat}((k+1) - p) > \text{pat}(k+1)$ , то  $\text{pat}(i, k+1)$  новый суффикс, его период  $k - i$  (тривиальный).
3. Если  $\text{pat}((k+1) - p) < \text{pat}(k+1)$ , то максимальный суффикс не может начинаться в позициях  $[0, i + \{d : d | p, d > r\}]$ .

*Доказательство.* Случай 1 очевиден. В остальных случаях от противного, и рассматриваем грани. Там можно вывести противоречие с максимальной суффикса на предыдущем шаге.  $\square$

Для оценки сложности заметим, что значение  $2i + j$  за  $k$  итераций увеличивается не меньше, чем на  $k$ .



Рис. 1.6: Алгоритм TW4 (Дюваля)

```
1  # вход: pat: str
2  p = <j - i>
3  i = 1, j = 2
4  пока j <= |pat|:
5      d = max { j + d <= |pat| : pat(i, i + d) = pat(j, j + d) }
6      если j + d > |pat|: выйди из цикла
7      если pat(i + d) > pat(j + d): j += d + 1
8      иначе: i += (d / p + 1), j = i + 1
9  верни pat(i, j - 1), p
```



## Глава 2

# Поиск нескольких подстрок в строке

### 2.1 Алгоритм Ахо-Корасик

Назовем множество строк  $D$ , которое будем называть словарем. Задача состоит в нахождении всех вхождений всех слов из  $D$  в строку `text`.

Задача решается в два этапа: составление вспомогательной структуры данных  $P$  и обработка строки `text`. Есть наивный алгоритм TR: в нем в качестве вспомогательной структуры берется бор, а вершины - структуры:

```
1  root : V = < корень бора >
2
3  v : V = {
4      .repr = < строка на пути root - v > # формально
5      .term = < .repr является словом словаря D >
6      .next = < (-): char -> V : .next(c).repr = .repr + c >
7  }
```

Сложность алгоритма TR будет зависеть от реализации  $(-).next$ . Если использовать отображение или отсортированный массив пар, то время доступа  $O(\log \sigma)$ . Если lookup-таблицу или массив, то  $O(1)$ . Для удобства будем представлять вершины числами  $\{1, 2, \dots\}$  и что  $next$  реализован lookup-таблицей.

Улучшим алгоритм TR: давайте добавим поле  $v.link$ , в котором будем указывать на вершину для которой  $u.repr$  будет наидлиннейшим суффиксом из  $v.repr$ .  $v.report$  будет указывать на вершину, которая при спуске по  $v.link$  будет удовлетворять  $.term = true$ . Тогда получаем алгоритм Ахо-Корасик АНК

**Утверждение 2.1.1.** Алгоритм АНК работает корректно и

$$T(\text{АНК}) = O(|\text{text}| \log \sigma + |D|)$$

*Доказательство.* Каждая итерация первого цикла увеличивает  $j = i - |v.repr| \leq i$ . Тогда он отработает за  $O(|\text{text}| \log \sigma)$ . Второй цикл каждый раз вызывается для разных слов, так что он будет вызван  $O(|D|)$  раз.

Корректность следует из того, что АНК – просто модификация алгоритма TR.  $\square$

Осталось научиться находить поле  $v.link$ . Для проверки максимального символа мы можем отрезать по одному символу  $v.repr$  и смотреть, в  $u.repr$  какой вершины мы перешли. Получим алгоритм АНК1.

**Утверждение 2.1.2.**  $T(\text{АНК1}) = O(\log \sigma \sum_{d \in D} |d|)$ .

*Доказательство.* Это верно из того, что на пути  $root - < v : v.repr = d >$  суммарное время работы для всех вершин будет  $O(|d|)$ :

$$\sum n_i \leq \sum |v_i.link.repr| - |v_d.link| + 1 = |v_d.link.repr| + |d|$$

$\square$

Если в словаре только одно слово, то получаем алгоритм Кнута-Морриса-Пратта. Тогда вместо дерева достаточно использовать массив значений. Вариант выше занимает  $S(\text{АНК}) = O(|V|)$  памяти, а автоматный вариант  $O(\sigma|V|)$  памяти.

Рис. 2.1: Алгоритм TR

```

1  v = root
2  для i = 1; i < |text| и v != nil; i += 1:
3      если v.term: нашли слово v.repr в позиции i
4      v = v.next(text(i))

```

Рис. 2.2: Алгоритм АНК

```

1  v = root
2  для i = 1; i < |text| и v != nil; i += 1:
3      пока v != root и v.next(text(i)) = nil: v = v.link
4      v = v.next(text(i))
5      если v = nil:
6          v = root
7      для u = v; u != root; u = u.report:
8          если u.term:
9              нашли слово u.repr в позиции i - |u.repr|

```

Рис. 2.3: Алгоритм АНК1

```

1  qu(1) = root.link = root.report = root
2  k = 2;
3  для i = 1; i < k; i += 1:
4      для v, c из { (v, c) qu(i).next(c) = v }:
5          qu[k++] = v
6          v.link = nil
7          для p = qu[i]; p.report != nil и v.link = nil; p = p.link:
8              v.link = p.link.next(c)
9          если v.link = nil:
10             v.link = root
11             v.report = v.link
12             пока !v.report.term:
13                 v.report = v.link.report

```



## Глава 3

# Строковые индексы

### 3.1 Суффиксный массив

По строке `text` необходимо построить структуру данных, с помощью которой можно находить вхождения строки `pat` в строку `text`.

**Определение 3.1.1.** *Суффиксный массив  $SA$  содержит перестановку чисел  $\{1, 2, \dots, |\text{text}|\}$ , причем  $\text{text}(SA(i), \cdot) < \text{text}(SA(j), \cdot)$  для  $i < j$ .*

#### 3.1.1 Алгоритм Карккайнена-Сандерса

Допустим, что алфавит представлен числами  $\{1, 2, \dots, |\text{text}|\}$ . Идейно можно бить строку на фрагменты длины  $2, 4, 8, \dots$  и сортировать их слиянием: тогда сложность  $T(n) = O(n) + T(\frac{n}{2}) = cn \sum_k \frac{1}{2^k} = O(n)$ .

Идейно: рекурсивно отсортируем все суффиксы: сначала на позициях не кратных трем, затем оставшиеся и выполним слияние.

Пусть на каком-то этапе отсортированными оказываются строки  $t_1, t_2$  (порядковой сортировкой). И в строке  $t_1$  все тройки меньше, чем в строке  $t_2$ . Пронумеруем тройки числами  $\{1, \frac{2}{3}|\text{text}|\}$ . Сформируем строку  $t_1\$t_2$  с бесконечно малым символом  $\$$ . Рассмотрим суффиксный массив  $SA_{12}$  для этой строки. Его занесем в  $ISA_{12}(SA(x)) = x$ .

Для сортировки оставшихся троек: если первые символы различны, то порядки определяются непосредственно, иначе – по построенным порядкам. Тогда, получается, достаточно отсортировать пары  $(\text{text}(3k), ISA_{12}(k))$ .

Для слияния так же переиспользуем порядок, зафиксированный в  $ISA_{12}$ .

### 3.2 Суффиксное дерево

**Определение 3.2.1.** *Массив  $LCP$ , содержит на позиции  $i$  наибольший общий префикс строк  $\text{text}(SA(i), \cdot)$  и  $\text{text}(SA(i+1), \cdot)$*

Если построить структуру, вычисляющую  $\min\{LCP(x) : i < x < j\}$  за  $O(1)$ , то можно построить наибольший общий префикс любой пары суффиксов за  $O(1)$ .  $LCP$  можно построить за  $O(|\text{text}|)$  используя  $SA, ISA$  по алгоритму Касаи и др.

**Определение 3.2.2.** *Суффиксный бор – бор, в котором пути отмечены всеми подстроками строки  $\text{text}$ . Терминальными вершинами считаются суффиксы.*

**Определение 3.2.3.** *Суффиксным деревом (сжатым суффиксным бором) назовем суффиксный бор без вершин с одним сыном. При удалении вершин соответствующие метки ребер склеиваются.*

Метками ребер оказываются подстроки. Будем хранить указатели на них. Тогда вершины являются структурами:

```

1  root : V = < корень бора >
2
3  v : V = {
4      .repr = < строка на пути root - v > # формально
5      .term = < .repr является суффиксом text >
6      .next = < по символу переходим в вершину,
7                на ребре до которой данный символ первый >
8      .par = < родитель >
9      .beg, .end = < text(.beg, .end) = str(.par, v) >
10 }

```

Из существования полей  $.beg, .end$ , очевидно, у суффиксного дерева  $|V| \leq 2|\text{text}|$ .

Суффиксное дерево можно построить по  $SA$  и  $LCP$ . Для этого надо вставлять суффиксы в порядке  $(\text{text}(SA(1), \cdot), \text{text}(SA(2), \cdot), \dots)$ . При вставке очередного суффикса  $\text{text}(SA(1), \cdot)$ , при вставке поднимаемся снизу этого пути, создавая вершину на нужном месте.

### 3.2.1 Упрощенный алгоритм Вейнера