Алгоритмы и структуры данных

Поиск подстрок

д.т.н., проф. Трифонов Петр Владимирович

Содержание лекции

Постановка задачи поиска подстрок

- Пусть даны текст, представленный в виде массива T длины n, и образец, представленный в виде массива длины $m \leq n$
- Элементами массивов являются символы некоторого конечного алфавита Σ
- Будем считать, что образец P входит в строку T со сдвигом $s, 0 \leq s \leq n - m$ (эквивалентно, входит с позиции $s + 1$), если $T[s + j] = P[j], 1 \leq j \leq m$. В этом случае также говорят, что s является допустимым сдвигом
- Задача поиска подстрок состоит в нахождении всех допустимых сдвигов для данных текста T и подстроки P

Обозначения

- $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$ — множество всех строк над алфавитом Σ, включая пустую строку
- $|x|$ — длина строки x
- xy — конкатенация строк x и y
- Строка w — префикс, или начало, строки x, если $\exists y \in \Sigma^* : x = wy$. Это свойство будем обозначать как $w \sqsubset x$
- Строка w — суффикс, или конец, строки x, если $\exists y \in \Sigma^* : x = yw$, что будем обозначать как $w \sqsupset x$
- Пример: $ab \sqsubset abcd$ и $bcd \sqsupset abcd$
- Из $x \sqsubset y$ не следует, что $y \sqsupset x$, и наоборот
- Пустая строка ϵ является префиксом и суффиксом любой строки.
- $\forall a \in \Sigma : x \sqsupset y \Leftrightarrow xa \sqsupset ya$ и $x \sqsubset y \Leftrightarrow ax \sqsubset ay$
- Префикс длины k строки S: $S_k = S[1..k], k \leq |S|$.

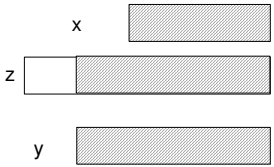
Лемма о двух суффиксах

Лемма

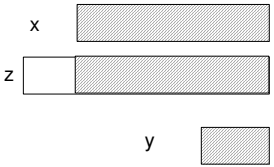
Пусть x, y, z — строки, для которых $x \sqsupset z$ и $y \sqsupset z$. Тогда $x \sqsupset y$, если $|x| \leq |y|$; $y \sqsupset x$, если $|x| \geq |y|$, и $x = y$, если $|x| = |y|$.

Доказательство.

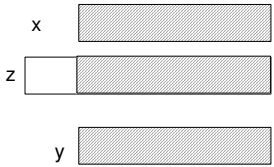
Совпадающие части строк заштрихованы



$|x| \leq |y|$



$|x| \geq |y|$



$|x| = |y|$

Простейший алгоритм поиска

- Поиск подстроки P в строке T сводится к поиску всех $s : 0 \leq |T| - |P| : P \sqsupset T_{s+m}$
- Будем считать, что операция сравнения подстрок выполняется путем посимвольного их сравнения, которое прекращается, как только обнаруживается расхождение.
- Сложность сравнения подстрок составляет $\Theta(t+1)$, где t — длина наибольшего общего префикса строк x и y.

FindSubstring(T, P)

```
1  n ← |T|
2  m ← |P|
3  for s ← 0 to n − m
4  do if P[1..m] = T[s + 1..s + m]
5        then print( "Подстрока входит со сдвигом s" )
```

Алгоритм Рабина-Карпа

- Пусть d — мощность рассматриваемого алфавита Σ
- Все строки можно рассматривать как числа в системе счисления по основанию d
- Пусть p — число, соответствующее образцу $P[1..m]$
- Для всех $s : 0 \leq s \leq n - m$ определим t_s как число, соответствующее подстроке $T[s + 1..s + m]$.
- Ссли s — допустимый сдвиг, то $t_s = p$
- Вычисление p: $p = P[m] + d(P[m - 1] + d(P[m - 2] + \ldots + dP[1]) \ldots)$ Аналогичным образом можно вычислить и t_0
- Прочие значения t_i можно вычислить как $t_{s+1} = d(t_s - d^{m-1}T[s + 1]) + T[s + m + 1]$.
- При непосредственной реализации этого метода возникают слишком большие числа
- Будем производить вычисления по модулю q, т.ч. dq помещается в разрядной сетке
- Если s $t_s \not\equiv p \bmod q$, то это s заведомо не является допустимым сдвигом. Но при $t_s \equiv p \bmod q$ нельзя исключать, что $t_s \neq p$. Требуется произвести непосредственное сравнение $T[s + 1..s + m]$ и P.

Алгоритм Рабина-Карпа

RabinKarpMatcher(T, P, d, q)

```
1  n ← |T|; m ← |P|; h ← d^{m−1} mod q
2  p ← 0; t_0 ← 0
3  for i ← 1 to m
4  do p ← (dp + P[i]) mod q
5     t_0 ← (dt_0 + T[i]) mod q
6  for s ← 0 to n − m
7  do if p = t_s
8        then if P[1..m] = T[s + 1..s + m]
9                  then print( "Образец входит со сдвигом s" )
10    if s < n − m
11       then t_{s+1} ← (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```

Сложность алгоритма Рабина-Карпа

- В худшем случае (как, например, при $T = a^n, P = a^m$) этот алгоритм требует $O((n - m + 1)m)$ операций
- Много вспомогательных операций, которые отсутствовали тривиальном алгоритме
- В большинстве практических случаев, когда допустимых сдвигов немного, сложность составляет $O(n + m)$ операций
  - Операция приведения по модулю q может рассматриваться как случайное отображение $\Sigma^* \to \mathbb{Z}_q$.
  - Вероятность того, что $t_s \neq p \wedge t_s \equiv p \bmod q$ равна примерно $1/q$, т.е. число операций составляет примерно $O(m + n - m) + O(m(v + (n - m)/q))$, где v — количество истинных вхождений подстроки P. Первое слагаемое соответствует вычислению t_i, второе — собственно сравнению подстрок
  - При $q \geq m$ и $v = O(1)$ средняя сложность алгоритма оказывается равной $O(n + m)$.
- Поиск несколько образцов: каждому из них может быть сопоставлено свое число p_i , а величины t_i достаточно вычислить однократно.

Конечный автомат

Конечным автоматом называется $M = (Q, q_0, A, \Sigma, \delta)$, где:

- Q — конечное множество состояний;
- $q_0 \in Q$ — начальное состояние;
- $A \subset Q$ — конечное множество допускающих состояний;
- Σ — конечный входной алфавит;
- $\delta : Q \times \Sigma \to Q$ — функция переходов.

Поиск подстрок с помощью КА

- Пусть первоначально автомат находится в состоянии q_0
- Находясь в состоянии q и читая очередной символ $a \in \Sigma$, автомат переходит в состояние $\delta(q, a)$
- Если автомат находится в состоянии $q \in A$, то говорят, что он допускает прочитанную часть строки
- В противном случае говорят, что он ее отвергает, что не исключает того, что после обработки оставшейся ее части она будет принята.

Суффикс-функция

- функция конечного состояния $\phi : \Sigma^* \to Q$. $\phi(w)$ есть состояние, в которое перейдет автомат, прочитав строку w. Автомат допускает строку w тогда и только тогда, когда $\phi(w) \in A$

$$\begin{aligned} \phi(\epsilon) &= q_0 \\ \phi(wa) &= \delta(\phi(w), a), \forall w \in \Sigma^*, a \in \Sigma. \end{aligned}$$

- Для построения автомата, распознающего заданную строку P, введем понятие суффикс-функции $\sigma(x)$, которая сопоставляет строке x длину максимального ее суффикса, являющегося префиксом P, т.е.

$$\sigma(x) = \max\{k : P_k \sqsupset x\}.$$

Т.к. $P_0 = \epsilon$ — суффикс любой строки, $\sigma(x)$ определена на всем множестве Σ^*.

Пример

Пусть $P = ab$. Тогда $\sigma(\epsilon) = 0, \sigma(ccaca) = 1, \sigma(ccab) = 2$.

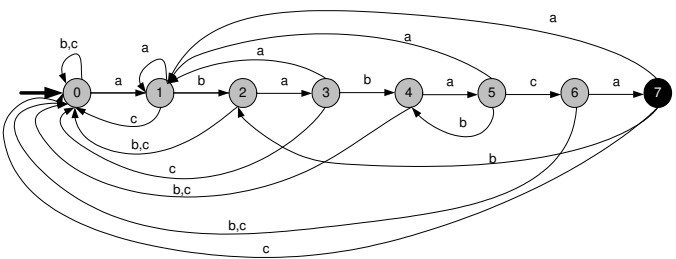
Построение автомата, распознающего строку

конечный автомат, соответствующий образцу $P[1..m]$:

- Множество состояний $Q = \{0, 1, \ldots, m\}$
- Начальное состояние $q_0 = 0$, единственное допускающее состояние m
- Функция переходов $\delta(q, a) = \sigma(P_q a)$.

Функция переходов обеспечивает $\phi(T_i) = \sigma(T_i)$, т.е. после обработки префикса длины i произвольной строки T автомат оказался в состоянии с таким номером $k = \phi(T_i)$, что $k = \sigma(T_i)$ равно длине наибольшего префикса P_k образца P, являющегося суффиксом T_i $\phi(T_i) = m$ означает, что последовательность $P = P_m$ входит в строку T со сдвигом $i - m$.

Пример: автомат, допускающий строки, оканчивающиеся на *ababaca*



	Переход по			
Состояние	a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

Результат применения к строке $T = abababacaba$

i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
$\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

$\delta(0, a) = \sigma(\epsilon a) = 1$, но $\delta(0, b) = \sigma(\epsilon b) = 0$. $\delta(5, a) = \sigma(P_5 a) = \sigma(ababaa) = 1$, т.к. $P_1 = a \sqsupset ababaa$, но $P_2 = ab \not\sqsupset ababaa$. $\delta(5, b) = \sigma(P_5 b) = \sigma(ababab) = 4$, т.к. $abab \sqsupset ababab$, но $ababa \not\sqsupset ababab$.

Поиск с помощью конечного автомата

FiniteAutomatonMatcher(T, δ, m)

```
1  n ← |T|
2  q ← 0
3  for i ← 1 to n
4  do q ← δ(q, T[i])
5      if q = m
6          then s ← i − m
7               print( "Образец входит со сдвигом s" )
```

- Предполагается, что таблица (функция) переходов конечного автомата, соответствующего искомой строке, уже построена
- Сложность $O(n)$

Корректность

Лемма

Для любой строки x и символа a выполняется $\sigma(xa) \leq \sigma(x) + 1$.

Доказательство.

Предположим, что $\sigma(xa) > \sigma(x) + 1$. Отбросим последний символ a от наибольшего суффикса xa, являющегося префиксом P. Это даст суффикс строки x, имеющий длину больше $\sigma(x)$ и являющийся префиксом P, что противоречит определению $\sigma(x)$. □

Лемма

Пусть для некоторой строки x $\sigma(x) = q$. Тогда для любого символа a $\sigma(xa) = \sigma(P_q a)$.

Доказательство.

В силу предыдущей леммы $\sigma(xa) \leq q + 1$. Поэтому значение $\sigma(xa)$ не изменится, если оставить от xa только последние $q + 1$ символов. Но последние q символов строки x совпадают с P_q, т.к. $\sigma(x) = q$. □

Корректность

Теорема

Пусть ϕ — функция конечного состояния автомата, построенного для поиска подстроки P. Тогда для произвольного текста T

$$\phi(T_i) = \sigma(T_i), i = 0, 1, \ldots, n.$$

Доказательство.

Для $i = 0$ утверждение очевидно. Предположим, что она верна для некоторого i и $q = \phi(T_i) = \sigma(T_i)$. Тогда $\phi(T_{i+1}) = \phi(T_i T[i+1]) = \delta(\phi(T_i), T[i+1]) = \delta(\sigma(T_i), T[i+1]) = \sigma(P_q T[i+1]) = \sigma(T_i T[i+1]) = \sigma(T_{i+1})$. □

Таким образом, автомат после прочтения i символов текста оказывается в состоянии q тогда и только тогда, когда P_q является самым длинным суффиксом строки T_i, являющимся одновременно префиксом строки P.

Построение функции переходов

ComputeTransitionFunction(P, Σ)

```
1  m ← |P|
2  for q ← 0 to m
3  do for ∀a ∈ Σ
4      do k ← min(m + 1, q + 2)
5         repeat
6                 k ← k − 1
7           until P_k ⊐ P_q a
8         δ(q, a) ← k
9  return δ
```

- перебор всех пары (q, a) аргументов функции и для каждого из них находит максимальное значение $k : P_k \sqsupset P_q a$.
- Два внешних цикла производят $m|\Sigma|$ итераций, внутренний цикл может выполняться не более $m + 1$ раз, и сравнение строк требует $O(m)$ операций. Сложность $O(m^3|\Sigma|)$

Алгоритм Кнута-Морриса-Пратта: идея

- Попытаемся избавиться от дорогостоящей процедуры построения функции переходов.
- Предположим, что для некоторого сдвига s оказалось, что первые q символов образца P совпадают со строкой T, но в следующем имеется расхождение. Эту позволяет чтобы исключить некоторые последующие заведомо недопустимые сдвиги

Пример

- поиск подстроки *ababaca* в строке *bacb*ababa*abcbab*
- При $s = 4$ наблюдается совпадение первых 5 символов образца с соответствующими символами строки
- Нет смысла рассматривать $s = 5$, т.к. в этом случае a, первый символ образца, будет сопоставляться с 6 символом строки, который, как уже известно, совпадает со 2 символом образца, равным b. Этого нельзя сказать про случай $s = 6$.

Алгоритм Кнута-Морриса-Пратта

- При обнаружении $P[1..q] = T[s+1..s+q]$ необходимо найти наименьшее значение сдвига $s' > s$, для которого

$$P[1..k] = T[s'+1..s'+k], s'+k = s+q. \quad (1)$$

  - $s'+k = s+q$ отражает тот факт, что решение о пропуске сдвигов с $s+1$ до s' принимается исключительно на основе уже проанализированных символов
  - В лучшем случае $s' = s+q$ можно будет отбросить сдвиги $s+1, \ldots, s+q-1$
  - При сравнении образца со строкой можно игнорировать первые k его символов, т.к. заведомо выполняется $P[1..k] = T[s+1..s+k] = T[s'+1..s'+k]$.
- В силу (1) получаем, что $T[s'+1..s'+k]$ — суффикс строки P_q. Поэтому число k в (1) является наибольшим числом $k < q$, таким что P_k является суффиксом P_q
- Префиксной функцией, для строки $P[1..m]$, называется функция $\pi : \{1, 2, \ldots, m\} \to \{0, 1, \ldots, m-1\}$, определенная как

$$\pi[q] = \max\{k | k < q \wedge P_k \sqsupset P_q\}.$$

Префиксная функция

$$\pi[q] = \max\{k | k < q \wedge P_k \sqsupset P_q\}.$$

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

Алгоритм Кнута-Морриса-Пратта: построение префиксной функции

```
ComputePrefixFunction(P)
 1  m ← |P|
 2  π[1] ← 0
 3  k ← 0
 4  for q ← 2 to m
 5  do while k > 0 ∧ P[k + 1] ≠ P[q]
 6        do k ← π[k]
 7        if P[k + 1] = P[q]
 8           then k ← k + 1
 9        π[q] ← k
10  return π
```

- k на каждой итерации FOR увеличивается не более чем на 1. Это происходит $O(m)$ раз
- k на каждой итерации WHILE уменьшается не менее чем на 1
- k никогда не становится отрицательным
- $\Rightarrow$ Общее число итераций в цикле WHILE $= O(m)$
- Сложность $O(m)$

Корректность алгоритма ComputePrefixFunction I

Пусть $\pi^*[q] = \{q, \pi[q], \pi^2[q], \ldots, \pi^t[q]\}$, где $\pi^i[q] = \pi[\pi^{i-1}[q]], \pi^0[q] = q$ и $\pi^t[q] = 0$. Такое число t всегда существует, т.к. $\forall j : \pi[j] < j$.

Лемма (Об итерациях префикс-функции)

Пусть P — строка длины m с префикс-функцией π. Тогда $\pi^*[q] = \{k | k < q, P_k \sqsupset P_q\}$.

Доказательство.

Покажем, что $i \in \pi^*[q] \Rightarrow P_i \sqsupset P_q$. Действительно, $P_{\pi[i]} \sqsupset P_i$, т.е. $\ldots P_{\pi[\pi[i]]} \sqsupset P_{\pi[i]} \sqsupset P_i$. Покажем и обратное, т.е. $\{k | P_k \sqsupset P_q\} \subset \pi^*[q]$. Предположим, что это не так. Пусть j — наибольшее число из $\{k | k < q, P_k \sqsupset P_q\} \setminus \pi^*[q]$. Ясно, что $j < q$. Т.к. $j \notin \pi^*[q]$, существует $j' \in \pi^*[q] :: j' > j > \pi[j']$. Строки P_j и $P_{j'}$ являются суффиксами P_q. Тогда $P_j \sqsupset P_{j'}$, причем P_j является префиксом строки P, являющимся суффиксом строки $P_{j'}$. Он имеет длину больше $\pi[j']$, что противоречит определению функции π. □

Корректность алгоритма ComputePrefixFunction II

Лемма

Пусть P — строка длины m, имеющая префикс-функцию π. Тогда $\pi[q] - 1 \in \pi^*[q-1]$ для всех $q = 1, 2, \ldots, m$, для которых $\pi[q] > 0$.

Доказательство.

Если $k = \pi[q] > 0$, то $P_k \sqsupset P_q$, откуда следует, что $P_{k-1} \sqsupset P_{q-1}$. В силу предыдущей леммы $k - 1 \in \pi^*[q-1]$. □

Для $q = 2, 3, \ldots, m$ определим множества $E_{q-1} = \{k | k \in \pi^*[q-1] \wedge P[k+1] = P[q]\}$, т.е. множества таких величин k, что P_k — суффикс P_{q-1} и за этими префиксами идут одни и те же символы $P[k+1]$, так что $P_{k+1} \sqsupset P_q$.

Корректность алгоритма ComputePrefixFunction III

Лемма

Пусть P — строка длины m с префикс-функцией π. Тогда для всех $q = 2, 3, \ldots, m$ имеем

$$\pi[q] = \begin{cases} 0, & \text{если } E_{q-1} = \emptyset \\ 1 + \max\{k \in E_{q-1}\}, & \text{если } E_{q-1} \neq \emptyset \end{cases}$$

Доказательство.

Если $r = \pi[q] \geq 1$, то $P[r] = P[q]$; кроме того, в силу предыдущей леммы $r - 1 \in \pi^*[q - 1]$, откуда $r - 1 \in E_{q-1}$. Следовательно, если $E_{q-1} = \emptyset$, то $\pi[q] = 0$. В противном случае $\pi[q] \leq 1 + \max\{k \in E_{q-1}\}$. С другой стороны, если $k \in E_{q-1}$, то $P_{k+1} \sqsupset P_q$, откуда $\pi[q] \geq k + 1$. □

Корректность алгоритма ComputePrefixFunction IV

```
ComputePrefixFunction(P)
 1  m ← |P|
 2  π[1] ← 0
 3  k ← 0
 4  for q ← 2 to m
 5  do while k > 0 ∧ P[k + 1] ≠ P[q]
 6        do k ← π[k]
 7        if P[k + 1] = P[q]
 8           then k ← k + 1
 9        π[q] ← k
10  return π
```

Корректность алгоритма ComputePrefixFunction V

Теорема

При входе в цикл FOR функции ComputePrefixFunction выполнено равенство $k = \pi[q-1]$.

Доказательство.

При $q = 2$ это равенство обеспечивается на этапе инициализации. Предположим, что утверждение теоремы верно для некоторого q. Рассмотрим результат выполнения очередной итерации этого цикла. В строках 5 и 6 ищется наибольший элемент множества E_{q-1}. Если это множество непусто (т.е. $P[k+1] = P[q]$ после выхода из цикла), то присваивание на строке 9 приведет в силу предыдущей леммы к $k = \pi[q]$. В противном случае окажется, что $\pi[q] = k = 0$, что также верно. □

Алгоритм Кнута-Морриса-Пратта: поиск

```
KMPMatch(T, P)
 1  n ← |T|
 2  m ← |P|
 3  π ← ComputePrefixFunction(P)
 4  q ← 0
 5  for i ← 1 to n
 6  do while q > 0 ∧ P[q + 1] ≠ T[i]
 7       do q ← π[q]
 8       if P[q + 1] = T[i]
 9         then q ← q + 1
10       if q = m
11         then print( "вхождение со сдвигом i − m" )
12              q ← π[q]
```

Сложность $O(n)$

Для обоснования корректности надо доказать, что для всех i

1. В момент исполнения строк 10–12 справедливо $q = \sigma(T_i)$.
2. Перед каждым исполнением тела цикла выполнено

$$q = \begin{cases} \sigma(T_{i-1}), & \text{если } \sigma(T_{i-1}) < m \\ \pi[m], & \text{если } \sigma(T_{i-1}) = m \end{cases}$$

Алгоритм Кнута-Морриса-Пратта: корректность KMPMatch I

- Суффикс-функция $\sigma(x)$ равна наибольшему числу k, такому что $P_k \sqsupset x$.
- При $i = 1$ второе утверждение выполняется
- Покажем, что для всех i из второго утверждения вытекает первое. Предположим, что это так при некотором i. Из леммы об итерациях ПФ вытекает, что в строках 6–7 алгоритма перебираются в убывающем порядке элементы множества $S = \{k < m | P_k \sqsupset P_q\}$. Перебор обрывается при нахождении наибольшего $k \in S$, для которого $P[k+1] = T[i]$, или при $k = 0$ и $P[1] \neq T[i]$
  - Согласно инд. предположению, в первом случае начальное значение q равно либо $\sigma(T_{i-1})$, либо $\pi[m]$, т.е. наибольшему $k : P_k \sqsupset P \sqsupset T_{i-1}$, т.е. после окончания цикла q равно наибольшему k, для которого $P_k \sqsupset T_{i-1}$. Отсюда вытекает, что $P_{k+1} \sqsupset T_i$, т.е. $\sigma(T_i) = k + 1$, и это значение присваивается переменной q после исполнения строки 9
  - Во втором случае после выхода из цикла на строках 6–7 имеем $\sigma(T_i) = 0$ и $q = 0$. Таким образом, первое утверждение выполнено.

Алгоритм Кнута-Морриса-Пратта: корректность KMPMatch II

- Если же утв. 1 справедливо для некоторого $i < n$, то второе утверждение верно для $i + 1$.
- На каждой итерации цикла проверка на строке 10 происходит при тех же условиях, что и на строке 5 в алгоритме *FiniteAutomatonMatcher*, что и гарантирует правильность алгоритма Кнута-Морриса-Пратта
- Алгоритм *KMPMatcher* выполняется за $O(n)$ операций
- Алгоритм *ComputePrefixFunction* требует $O(m)$ операций
- Общая сложность алгоритма КМП равна $O(m + n)$

Выводы

- Алгоритм Рабина-Карпа
- Поиск подстрок может быть выполнен с помощью конечного автомата
- Алгоримт Кнута-Морриса-Пратта позволяет выполнить поиск подстрок со сложностью $O(m + n)$