Поиск образца в строке

Дискретный анализ 2017/18

28 октября 2017 г.

Литература

- Дэн Гасфилд, «Строки деревья и последовательности в алгоритмах: Информатика и вычислительная биология», 2003. Глава 1, «Точное совпадение», глава 2, «Точное совпадение: классические методы», глава 3 «Более глубокий взгляд», стр. 19–94.
- Билл Смит, «Методы и алгоритмы вычислений на строках», 2006.

Введение
Основные определения
Постановка задачи
Очевидный метод
Предварительная обработка образца
Z-блоки
Построение Z-блоков за линейное время
Поиск с использованием Z-блоков
Алгоритм Кнута-Морриса-Пратта
Основной алгоритм
Алгоритм реального времени
Алгоритм Бойера-Мура
Основная идея
Правило хорошего суффикса
Реализация
Алгоритм Апостолико-Джанкарло
Имитация алгоритма Бойера-Мура
Линейность

Раздел

Введение

Основные определения

Постановка задачи

Очевидный метод

Предварительная обработка образца

Z-блоки

Построение Z-блоков за линейное время

Поиск с использованием Z-блоков

Алгоритм Кнута-Морриса-Пратта

Основной алгоритм

Алгоритм реального времени

Алгоритм Бойера-Мура

Основная идея

Правило хорошего суффикса

Реализация

Алгоритм Апостолико-Джанкарло

Имитация алгоритма Бойера-Мура

Линейность

Строка

1. Строка S — упорядоченный список символов, записанный подряд слева направо.
2. $S(i)$ — i-й символ строки S.
3. $S[i..j]$ — сплошная подстрока, начинающаяся в позиции с i и заканчивающаяся в позиции j. $S[i..j]$ пуста, если $i > j$.
4. $|S|$ — количество символов в строке S.
5. $S[1..i]$ — префикс строки S.
6. $S[i..|S|]$ — суффикс строки S.
7. Собственные суффикс и префикс строки S не пусты и не совпадают со строкой.

Строки и последовательности

- Строка и последовательность — синонимы.
- Подстрока и подпоследовательность — разные термины: подстрока является подпоследовательностью, обратное не обязательно. В подстроке символы исходной строки должны идти подряд, в подпоследовательности должен только соблюдаться порядок.
- Для $ababac$: aba — подстрока, abc — подпоследовательность.

Соглашения

- S — любая строка.
- P — образец.
- T — текст.
- Строчные греческие буквы (α, β, γ, δ и т.д.) — переменные строки.
- Строчные латинские буквы (x, y, z и т.д.) — отдельные переменные символы.
- Σ — алфавит.

Раздел

- Введение
  - Основные определения
  - Постановка задачи
  - Очевидный метод
- Предварительная обработка образца
  - Z-блоки
  - Построение Z-блоков за линейное время
  - Поиск с использованием Z-блоков
- Алгоритм Кнута-Морриса-Пратта
  - Основной алгоритм
  - Алгоритм реального времени
- Алгоритм Бойера-Мура
  - Основная идея
  - Правило хорошего суффикса
  - Реализация
- Алгоритм Апостолико-Джанкарло
  - Имитация алгоритма Бойера-Мура
  - Линейность

Точное совпадение

Заданы:

1. Строка P — образец или паттерн.
2. Строка T — текст.

Необходимо отыскать все вхождения образца P в текст T.
Для $P = aba$ и $T = bbabaxababay$ образец входит в текст начиная с позиций 3, 7, 9.

Раздел

Введение
- Основные определения
- Постановка задачи
- Очевидный метод

Предварительная обработка образца
- Z-блоки
- Построение Z-блоков за линейное время
- Поиск с использованием Z-блоков

Алгоритм Кнута-Морриса-Пратта
- Основной алгоритм
- Алгоритм реального времени

Алгоритм Бойера-Мура
- Основная идея
- Правило хорошего суффикса
- Реализация

Алгоритм Апостолико-Джанкарло
- Имитация алгоритма Бойера-Мура
- Линейность

Алгоритм

Naive-Pattern-Search(P, T)

$n \leftarrow |P|$
$m \leftarrow |T|$
for $i \leftarrow 1$ **to** $m - n + 1$
    $j \leftarrow 1$
    while $j \leq n$ and $P(j) = T(i + j - 1)$
        $j \leftarrow j + 1$
    if $j = n + 1$
        // Вхождение образца в позиции i.

Время работы

1. Худший случай: P и T состоят из одного и того же повторяющегося символа.
2. Тогда выполняется $n \times (m - n + 1)$ сравнений.
3. $P = aaa$ и $T = aaaaaaaaaa$, то $n = 3$ и $m = 10$, выполняется 24 сравнения.

Идеи по улучшению

- Не сравнивать два раза одни и те же символы из текста, использовать информацию о результатах предыдущих сравнений с образцом.
- Предварительно обрабатывать образец, выявлять в нём структуру, закономерности.
- Сдвигать образец более чем на один символ.
- Возможны линейный и сублинейный алгоритмы поиска образца.

Раздел

Введение
- Основные определения
- Постановка задачи
- Очевидный метод

Предварительная обработка образца
- Z-блоки
- Построение Z-блоков за линейное время
- Поиск с использованием Z-блоков

Алгоритм Кнута-Морриса-Пратта
- Основной алгоритм
- Алгоритм реального времени

Алгоритм Бойера-Мура
- Основная идея
- Правило хорошего суффикса
- Реализация

Алгоритм Апостолико-Джанкарло
- Имитация алгоритма Бойера-Мура
- Линейность

$Z_i(S)$

Для данной строки S и позиции $i > 1$ определяется величина $Z_i(S)$ как длина наибольшего префикса $S[i..|S|]$, совпадающего с префиксом S.

Если $S = aabcaabxaaz$, то

- $Z_5(S) = 3$ $(aabc \dots aabx \dots)$.
- $Z_6(S) = 1$ $(aa \dots ab \dots)$.
- $Z_7(S) = Z_8(S) = 0$.
- $Z_9(S) = 2$ $(aab \dots aaz)$.

Z-блок

Для любой позиции $i > 1$

1. в которой $Z_i > 0$, Z-блок это интервал, начинающийся в i и кончающийся в позиции $i + Z_i - 1$.
2. r_i — крайний правый конец Z-блоков, начинающихся не позднее i. Или

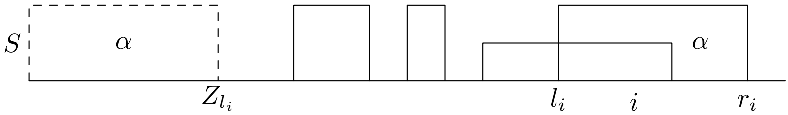
$$r_i = \max_{1<j\leq i}\{j + Z_j - 1\}$$

3. l_i — значение, на котором достигается этот максимум (т.е. Z_{l_i} заканчивается в r_i). Или

$$l_i = \arg\max_{1<j\leq i}\{j + Z_j - 1\}$$

Z-блоки

Позиция i находится внутри двух Z-блоков, выбирается с максимальной правой границей, r_i.



Раздел

- Введение
  - Основные определения
  - Постановка задачи
  - Очевидный метод
- Предварительная обработка образца
  - Z-блоки
  - Построение Z-блоков за линейное время
  - Поиск с использованием Z-блоков
- Алгоритм Кнута-Морриса-Пратта
  - Основной алгоритм
  - Алгоритм реального времени
- Алгоритм Бойера-Мура
  - Основная идея
  - Правило хорошего суффикса
  - Реализация
- Алгоритм Апостолико-Джанкарло
  - Имитация алгоритма Бойера-Мура
  - Линейность

Основные идеи

1. Z_2, r_2 и l_2 строятся непосредственным сравнением строк S и $S[2..|S|]$.
2. Допустим, Z_i получено для всех $1 < i \leq k-1$ и нужно найти Z_k.
3. Известно, что $S[k..r_{k-1}]$ совпадает с некоторой строкой из префикса S, начиная с $k' = k - l_{k-1} + 1$.
4. Тогда, исходя из значения $Z_{k'}$, можно пропустить прямое сравнение символов до r_{k-1}.
5. Тем самым, ни один символ обрабатываемой строки никогда не будет сравниваться дважды.

Подстрока β

Обрабатывается k-й символ, α начинается с l и кончается в r, длина блока Z_l символов, та же строка α располагается в префиксе строки S. Тогда в префиксе позиции k соответствует позиция $k' = k - l + 1$, где находится подстрока β:

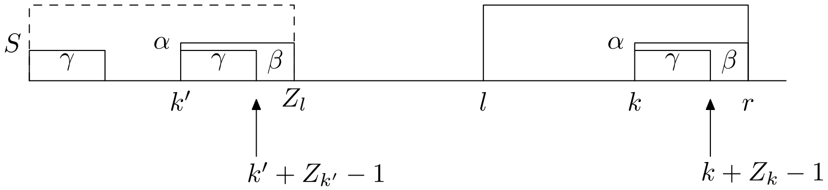
Алгоритм

$\textsc{Build-ZBlocks}(S)$

$r \leftarrow 0$
for $1 < k \leq |S|$
    if $k > r$ // Случай 1
        $Z_k = \textsc{Common-Prefix-Length}(S, S[k..|S|])$
        if $Z_k > 0$
            $r \leftarrow k + Z_k - 1$, $l \leftarrow k$
    else $k' \leftarrow k - l + 1$
        if $Z_{k'} < |\beta|$ // β — строка в $S[k..r]$ и $S[k'..Z_l]$
            $Z_k = Z_{k'}$ // Случай 2а
        else $q \leftarrow r + \textsc{Common-Prefix-Length}$ // Случай 2б
            $(S[|\beta| + 1..|S|], S[r + 1..|S|])$
            $Z_k \leftarrow q - k + 1$, $r \leftarrow q$, $l \leftarrow k$

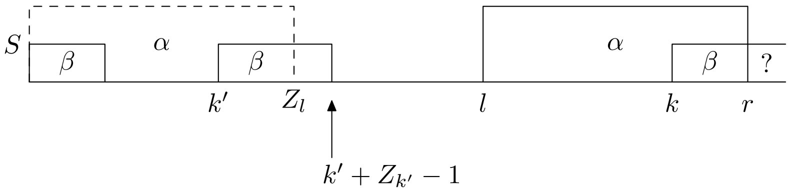
$Z_{k'} < |\beta|$

Z-блок, начинающийся в k', короче β. Отсюда Z_k присваивается значение $Z_{k'}$:



$Z_{k'} \geq |\beta|$

Z-блок, начинающийся в k', длиннее β. Следовательно, Z_k как минимум равен $r - k + 1$, дальше нужно непосредственно сравнить подстроки $S[r+1..|S|]$ и $S[Z_l..|S|]$ для вычисления точного значения Z_k:



Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k											
r	0										
l											

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1									
r	0	2									
l		2									

$k = 2$: Z_2 получен прямым сравнением.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0								
r	0	2	2								
l		2	2								

$k = 3$: Так как r меньше 3, то опять выполняется непосредственное сравнение префиксов начиная с 3-й позиции.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0	0							
r	0	2	2	2							
l		2	2	2							

$k = 4$: $r < 4$, вычисляем Z_4 прямым сравнением.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0	0	3						
r	0	2	2	2	7						
l		2	2	2	5						

$k = 5$: $r < 5$, сравниваем префиксы. На этот раз $Z_5 = 3$, заполняем значения $l = k = 5$ и $r = k + Z_5 - 1 = 7$.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0	0	3	1					
r	0	2	2	2	7	7					
l		2	2	2	5	5					

$k = 6$: $r > 6$, $k' = 2$ и $Z_2 = 1$, т.е. правая граница Z-блока заканчивается раньше, чем Z_5 символов от начала S, следовательно $Z_6 = Z_2 = 1$ без каких либо сравнений.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0	0	3	1	0				
r	0	2	2	2	7	7	7				
l		2	2	2	5	5	5				

$k = 7$: Аналогично, $Z_7 = Z_3 = 0$ без выполнения сравнений символов.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0	0	3	1	0	0			
r	0	2	2	2	7	7	7	7			
l		2	2	2	5	5	5	5			

$k = 8$: $r < 8$, выполняем непосредственное сравнение.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0	0	3	1	0	0	2		
r	0	2	2	2	7	7	7	7	10		
l		2	2	2	5	5	5	5	9		

$k = 9$: Непосредственным сравнением находим $Z_9 = 2$, заполняем $r = 10$ и $l = 9$.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0	0	3	1	0	0	2	1	
r	0	2	2	2	7	7	7	7	10	10	
l		2	2	2	5	5	5	5	9	10	

$k = 10$: Т.к. $Z_2 = 1$, то Z_{10} как минимум равен 1 и Z-блок заканчивается на границе проверенных данных, нужно попытаться сравнить символы 11-й и 2-й. Несмотря на то, что они не равны, модифицируются переменные r и l.

Пример выполнения алгоритма

k	1	2	3	4	5	6	7	8	9	10	11
S	a	a	b	c	a	a	b	x	a	a	z
Z_k		1	0	0	3	1	0	0	2	1	0
r	0	2	2	2	7	7	7	7	10	10	10
l		2	2	2	5	5	5	5	9	10	10

$k = 11$: Непосредственным сравнением получаем $Z_{11} = 0$.

Корректность вычисления Z_k

Теорема

При использовании одной итерации алгоритма Build-ZBlocks *значения* Z_k *корректно вычисляются и переменные* r *и* l *корректно обновляются.*

Следствие

Повторное применение алгоритма для каждой позиции $k > 2$ *корректно находит все значения* Z_k.

Корректность вычисления Z_k: доказательство

Доказательство.

1. Случай 1, $k > r$: Z_k вычисляется непосредственно. Замена r корректна, т.к. нет ни одного Z-блока, кончающегося в k или после неё.
2. Случай 2а, $Z_{k'} < |\beta|$: Если $Z_k > Z_{k'}$, то $S(k' + Z_{k'}) = S(k + Z_k) = S(1 + Z_{k'})$, что противоречит корректности вычисления $Z_{k'}$. $k + Z_k - 1 < r$, т.е. r и l не меняются.
3. Случай 2б, $Z_{k'} \geq |\beta|$: β является префиксом S, продолжение вычисляется непосредственно. $k + Z_k - 1 \geq r$, следовательно меняются r и l.

□

Линейность алгоритма построения Z-блоков

Теорема

Все значения $Z_k(S)$ вычисляются алгоритмом Build-ZBlocks *за время $O(|S|)$.*

Доказательство.

1. Число итераций: $|S|$.
2. При выполнении сравнений несовпадение завершает итерацию, т.е. количество несовпадений $|S|$.
3. $\forall k : r_k \geq r_{k-1}$. Если было $q > 0$ совпадений, то $r_k \geq r_{k-1} + q$. Кроме того, $r_k \leq |S|$ и сравнения выполняются только когда $k > r$, следовательно число совпадений не превосходит $|S|$.

$\square$

Раздел

Введение
- Основные определения
- Постановка задачи
- Очевидный метод

Предварительная обработка образца
- Z-блоки
- Построение Z-блоков за линейное время
- Поиск с использованием Z-блоков

Алгоритм Кнута-Морриса-Пратта
- Основной алгоритм
- Алгоритм реального времени

Алгоритм Бойера-Мура
- Основная идея
- Правило хорошего суффикса
- Реализация

Алгоритм Апостолико-Джанкарло
- Имитация алгоритма Бойера-Мура
- Линейность

Простейший алгоритм линейного поиска

- $S = P\$T$, где $\$$ — символ, отсутствующий в P и T.
- $|P| = n$, $|T| = m$, $n \leq m$, $\Rightarrow |S| = n + m + 1 = O(m)$.
- Вычисляем $Z_i(S)$ для $2 \leq i \leq n + m + 1$, $Z_i \leq n$.
- Для всех значений $i > n + 1$, для которых $Z_i(S) = n$, есть вхождение P в T в позиции $i - (n + 1)$.
- И наоборот: если есть вхождение P в T в, то $Z_{n+1+j} = n$.
- $O(n + m) = O(m)$.

Свойства алгоритма

- Время выполнения $O(m)$.
- Требуется дополнительная память размера $O(n)$ (Z-блоки для образца).
- Алфавитно-независимый метод: нужна только операция сравнения символов, не нужно даже знать размерность алфавита.

Зачем продолжать?

- Поиск набора образцов в тексте (алгоритм Ахо-Корасика).
- Алгоритмы реального времени.
- Сублинейное время работы.
- Время работы, пропроциональное длине образца (суффиксные деревья).

Раздел

Введение
- Основные определения
- Постановка задачи
- Очевидный метод

Предварительная обработка образца
- Z-блоки
- Построение Z-блоков за линейное время
- Поиск с использованием Z-блоков

Алгоритм Кнута-Морриса-Пратта
- Основной алгоритм
- Алгоритм реального времени

Алгоритм Бойера-Мура
- Основная идея
- Правило хорошего суффикса
- Реализация

Алгоритм Апостолико-Джанкарло
- Имитация алгоритма Бойера-Мура
- Линейность

Общая идея

- Образец прикладывается к тексту как в очевидном («наивном») алгоритме.
- Выполняются сдвиги образца более, чем на один символ.
- Для выполнения сдвигов исследуются суффиксы и префиксы подстрок образца.
- Сравнение продолжается с той же позиции в тексте, на котором закончилась предыдущая итерация.
- Для $P = abcxabcde$ и несовпадении в 8-й позиции можно сдвинуть P на 4 места вне зависимости от текста T.

Пример

```
....abcxabc?....
    abcxabcde
```

Пример

```
....abcxabc?....
    abcxabcde
```

Пример

```
....abcxabc?....
      abcxabcde
```

Пример

```
....abcxabc?....
      abcxabcde
```

Пример

```
....abcxabc?....
       abcxabcde
```

sp_i и sp'_i

- $sp_i(P)$ — длина наибольшего собственного суффикса $P[1..i]$, совпадающим с префиксом P.
- $sp'_i(P)$ — то же самое с дополнительным условием $P(i+1) \neq P(sp'_i + 1)$.
- $sp'_i(P) \leq sp_i(P)$

Примеры sp_i и sp'_i

i	1	2	3	4	5	6	7	8	9	10	11
P	a	b	c	a	e	a	b	c	a	b	d
sp_i	0	0	0	1	0	1	2	3	4	2	0
sp'_i	0	0	0	1	0	0	0	0	4	2	0

i	1	2	3	4	5	6	7	8	9	10	11	12
P	b	b	c	c	a	e	b	b	c	a	b	d
sp_i	0	1	0	0	0	0	1	2	3	0	1	0
sp'_i	0	1	0	0	0	0	0	1	3	0	1	0

Правило сдвига Кнута-Морриса-Пратта

- $P(i+1) \neq T(k)$: сдвиг выполняется таким образом, что $P[1..sp'_i] \rightarrow T[k - sp'_i..k-1]$, т.е. образец смещается вправо на $i - sp'_i$ позиций.
- При обнаружении вхождения: сдвиг на $n - sp'_n$ мест.
- В следующем сравнении будут участвовать символы $T(k)$ и $P(sp'_i + 1)$.

Корректность

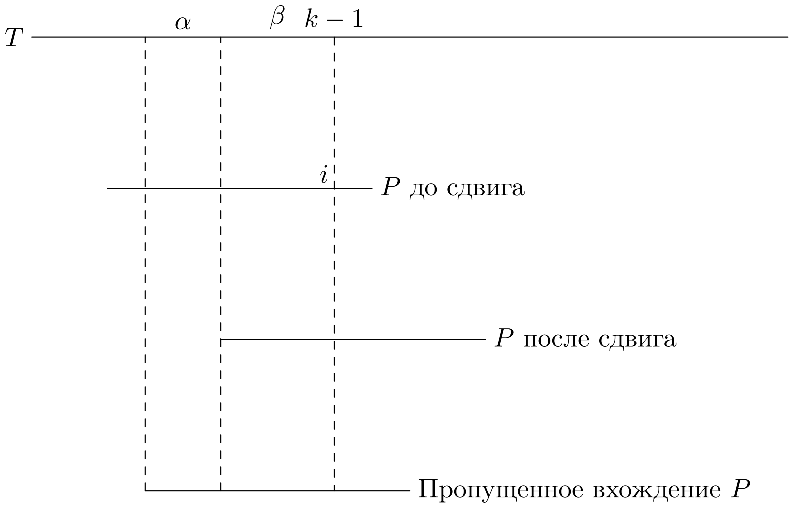
Теорема

После несовпадения в позиции $i+1$ образца P и сдвига на $i - sp'_i$ мест вправо левые sp'_i символов P гарантировано совпадут со своими парами в T.

Теорема

Для любого выравнивания P с T если $P[1..i] = T[k-i..k-1]$ и $P(i+1) \neq T(k)$, то P может быть сдвинуто на $i - sp'_i$ позиций вправо без пропуска вхождений P в T.

Отсутствие пропусков



Линейность

Теорема

В методе Кнута-Морриса-Пратта число сравнений символов не превосходит $2m$.

Доказательство.

- Фазы сравнения/сдвига: некоторое количество сравнений и сдвиг образца.
- В каждой фазе не более одного сравнения с символом из T, который уже сравнивался.
- Количество сравнений $m + s$, где s — число сдвигов; т.к. $s < m$, то общее число сравнений меньше $2m$.

$\square$

Связь sp'_i с Z-блоками

1. j отображается в i, если $i = j + Z_j(P) - 1$.
2. $sp'_i = Z_j = i - j + 1$, где j — наименьший индекс отображаемый в i. Если такого j нет, то $sp'_i = 0$.
3. $sp_i = i - j + 1$, где j — наименьший индекс $1 < j \leq i$, отображаемый в i или дальше. Если такого j нет, то $sp_i = 0$.

Примеры sp_i и sp'_i с Z-блоками

i	1	2	3	4	5	6	7	8	9	10	11
P	a	b	c	a	e	a	b	c	a	b	d
sp_i	0	0	0	1	0	1	2	3	4	2	0
sp'_i	0	0	0	1	0	0	0	0	4	2	0
Z_i		0	0	1	0	4	0	0	2	0	0

i	1	2	3	4	5	6	7	8	9	10	11	12
P	b	b	c	c	a	e	b	b	c	a	b	d
sp_i	0	1	0	0	0	0	1	2	3	0	1	0
sp'_i	0	1	0	0	0	0	0	1	3	0	1	0
Z_i		1	0	0	0	0	3	1	0	0	1	0

Вычисление sp'_i и sp_i

for $i \leftarrow 1$ **to** n
    $sp'_i \leftarrow 0$
for $j \leftarrow n$ **downto** 2
    $i \leftarrow j + Z_j(P) - 1$
    $sp'_i \leftarrow Z_j(P)$
$sp_n \leftarrow sp'_n$
for $i \leftarrow n-1$ **downto** 2
    $sp_i \leftarrow \max\{sp_{i+1} - 1, sp'_i\}$

Функция неудачи

- $F'(i) = sp'_{i-1} + 1$.
- $F(i) = sp_{i-1} + 1$.
- Функция неудачи показывает, какой символ образца нужно сравнивать с текущим символов текста при несовпадении в $P(i)$.

Алгоритм Кнута-Морриса-Пратта

// Обработать P и найти $F'(k)$ для $1 \le k \le n+1$.

$c \leftarrow 1$, $p \leftarrow 1$

while $c + n - p \le m$

    while $P(p) = T(c)$ and $p \le n$

        $p \leftarrow p + 1$, $c \leftarrow c + 1$

    if $p = n + 1$

        // Вхождение P в T в позиции $c - n$.

    if $p = 1$

        $c \leftarrow c + 1$

    $p \leftarrow F'(p)$

Выводы

- Сдвиги выполняются больше чем на один символ.
- Текст обрабатывается последовательно, нет возвратов.
- Линейный (но не сублинейный) алгоритм.
- Нет зависимости от алфавита.

Раздел

- Введение
  - Основные определения
  - Постановка задачи
  - Очевидный метод
- Предварительная обработка образца
  - Z-блоки
  - Построение Z-блоков за линейное время
  - Поиск с использованием Z-блоков
- **Алгоритм Кнута-Морриса-Пратта**
  - Основной алгоритм
  - **Алгоритм реального времени**
- Алгоритм Бойера-Мура
  - Основная идея
  - Правило хорошего суффикса
  - Реализация
- Алгоритм Апостолико-Джанкарло
  - Имитация алгоритма Бойера-Мура
  - Линейность

Алгоритмы реального времени

- Константное время между проверками любых позиций текста.
- Более строгое понятие, чем линейный алгоритм.
- Предыдущий алгоритм линейный, но не реального времени: неизвестно сколько времени потребуется на выполнение сдвига у определённой позиции, может быть 1 такт, а может $|P|$ тактов.
- Для преобразования нужно рассчитать sp_i' для каждого символа алфавита, с которым происходит сравнение.

$sp'_{(i,x)}$

- $x \in \Sigma$.
- $sp'_{(i,x)}$ — длина наибольшего собственного суффикса $P[1..i]$, совпадающий с префиксом P, при условии что $P(sp'_i + 1) = x$.
- Можно сформулировать правило сдвига, при котором или гарантируется совпадение $T(k) = P(sp'_{i,x} + 1)$, или гарантируется отсутствие префикса, продолжающегося символом $T(k)$.
- Тогда никакой символ T не проверяется больше одного раза.

Вычисление $sp'_{(i,x)}$

Если $P(i+1) \neq x$, то $sp'_{(i,x)} = i - j + 1$, где j — наименьшая позиция, такая что j отображается в i и $P(Z_j + 1) = x$. Если такого j нет, то $sp'_{(i,x)} = 0$.

```
for i ← 1 to n
    for x ∈ Σ
        sp'_(i,x) ← 0
for j ← n downto 2
    i ← j + Z_j(P) − 1, x ← P(Z_j + 1), sp'_(i,x) ← Z_j
```

Раздел

- Введение
  - Основные определения
  - Постановка задачи
  - Очевидный метод
- Предварительная обработка образца
  - Z-блоки
  - Построение Z-блоков за линейное время
  - Поиск с использованием Z-блоков
- Алгоритм Кнута-Морриса-Пратта
  - Основной алгоритм
  - Алгоритм реального времени
- Алгоритм Бойера-Мура
  - Основная идея
  - Правило хорошего суффикса
  - Реализация
- Алгоритм Апостолико-Джанкарло
  - Имитация алгоритма Бойера-Мура
  - Линейность

Основная идея

- Образец последовательно прикладывается к тексту, сравнение идёт справа налево.
- Правило сдвига по плохому символу.
- Правило сдвига по хорошему суффиксу.
- Обычно проверяется не больше, чем $m + n$ символов, линейное время в худшем случае (сильное правило хорошего суффикса) и ожидаемое сублинейное время.

Функция $R(x)$

- Для каждого символа алфавита x пусть $R(x)$ — позиция крайнего правого вхождения x в P. Если x в P не входит, $R(x) = 0$.
- Время подготовки таблицы для $R(x)$: $O(n)$.
- Например: $abad$, $R(b) = 2$, $R(a) = 3$. Соответственно, если при первом сравнении образца в тексте был обнаружен символ b, то образец можно сдвинуть на $|P| - R(b) = 4 - 2 = 2$ символа.

Правило сдвига по плохому символу

- Правый символ P приложен к позиции q в тексте.
- $P(n) = T(q)$, $P(n-1) = T(q-1)$, ...
- $P(n-i) \neq T(k)$; $k = q - i$.
- Тогда P можно сдвинуть вправо на $\max\{1, i - R(T(k))\}$ мест.
- Если крайнее правое вхождение в P символа $T(k)$ занимает позицию $j < i$, то P сдвигается так, чтобы символ j поравнялся с символом k в T. В противном случае, P сдвигается на одну позицию.
- Правило эффективно при несовпадениях, близких к правому концу P.

Расширенное правило плохого символа

- Несовпадение в позиции i образца P.
- x — несовпадающий символ в T.
- Нужно совместить с этим x ближайшее вхождение x в P слева от i.
- Для $abad$ $R(a, 4) = 3$ и $R(a, 2) = 1$.

Реализация расширенного правила

- Таблица размером $n \times |\Sigma|$.
- Для каждого символа алфавита x хранить список позиций в P, расположенных по убыванию. Например, для $P = abacbabc$ и символа a получится список $\langle 6, 3, 1 \rangle$.

Свойства правила

- Высокая эффективность в практических условиях, для текстов с большими размерами алфавитов (английский текст).
- Для маленьких алфавитов — хуже.
- Не гарантирует линейности.

Раздел

Введение
- Основные определения
- Постановка задачи
- Очевидный метод

Предварительная обработка образца
- Z-блоки
- Построение Z-блоков за линейное время
- Поиск с использованием Z-блоков

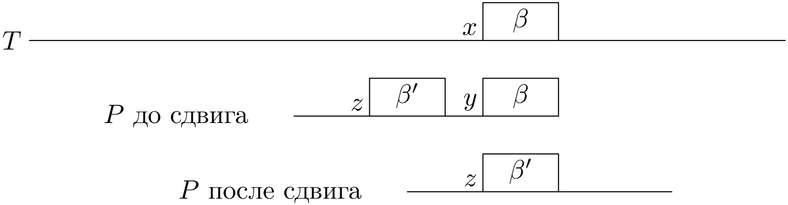
Алгоритм Кнута-Морриса-Пратта
- Основной алгоритм
- Алгоритм реального времени

Алгоритм Бойера-Мура
- Основная идея
- Правило хорошего суффикса
- Реализация

Алгоритм Апостолико-Джанкарло
- Имитация алгоритма Бойера-Мура
- Линейность

Сдвиг по хорошему суффиксу

Символ x из T не совпал с y из P. По сильному правилу хорошего суффикса обеспечивается сдвиг таким образом, чтобы β' приложилось к β, где $z \neq y$.



Сильное правило хорошего суффикса

- P приложена к T. β — суффикс P, совпадает с подстрокой T, символ y не совпадает с x из T.
- Если существует β', самая правая копия β в P такая, что предшествующий ей символ $z \neq y$, то нужно обеспечить такой сдвиг образца, чтобы β' приложилась к β.
- В противном случае нужно сдвинуть образец на наименьший сдвиг, при котором собственный префикс сдвинутого образца совпадает с суффиксом вхождения P в T.
- Если и такой сдвиг невозможен, то выполняется свдиг P на n позиций.

Пример

```
1234567890123456789
prstabstubabvqxrst
  qcabdabdab
  1234567890
```

$P(8) \neq T(10)$.

Пример

```
12345678901234567
prstabstubabvqxrst
  qcabdabdab
  1234567890
```

$\beta = ab$, $\beta' = P[3..4]$.

Пример

```
123456789012345678
prstabstubabvqxrst
  qcabdabdab
  1234567890
```

$P[3..4] = T[11..12]$.

```
123456789012345678
prstabstubabvqxrst
          qcabdabdab
          1234567890
```

Корректность правила хорошего суффикса

Теорема

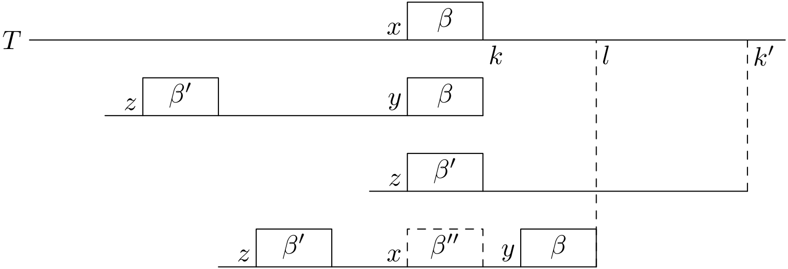
Использование правила хорошего суффикса никогда не сдвинет P за его вхождение в T.

Доказательство.

- До сдвига правый конец P стоял у позиции k в T и сдвиг произошёл к позиции k'.
- Предположим, что в позиции $k < l < k'$ заканчивалось пропущенное вхождение P.
- Тогда в P должна быть либо более близкая копия β или более длинный префикс совпадал бы с суффиксом β.

□

Пропущенное вхождение в l



Слабое правило хорошего суффикса

- Было сформулировано в оригинальном алгоритме.
- Аналогично сильному правилу, но не требует разных символов перед β и β'.
- Не даёт возможность получить линейную наихудшую оценку.

Раздел

Введение
- Основные определения
- Постановка задачи
- Очевидный метод

Предварительная обработка образца
- Z-блоки
- Построение Z-блоков за линейное время
- Поиск с использованием Z-блоков

Алгоритм Кнута-Морриса-Пратта
- Основной алгоритм
- Алгоритм реального времени

Алгоритм Бойера-Мура
- Основная идея
- Правило хорошего суффикса
- Реализация

Алгоритм Апостолико-Джанкарло
- Имитация алгоритма Бойера-Мура
- Линейность

$L(i)$, $L'(i)$, $N_i(P)$

Для каждого i:

- $L(i) < n$ — наибольшая позиция, такая что $P[i..n]$ совпадает с суффиксом $P[1..L(i)]$.
- $L'(i) < n$ — наибольшая позиция, такая что $P[i..n]$ совпадает с суффиксом $P[1..L'(i)]$, а символ, предшествующий ему, не равен $P(i-1)$.
- $N_i(P)$ — длина наибольшего суффикса $P[1..i]$, который является также суффиксом P.
- Если P^r — реверс P, то $N_i(P) = Z_{n-i+1}(P^r)$, т.е. $N_i(P)$ вычисляется при помощи $\textsc{Build-ZBlocks}$.

Вычисление $L(i)$ и $L'(i)$

$$
\begin{aligned}
L(i) &= \max\left\{1 \le j < n \,\middle|\, N_j(P) \ge |P[i..n]| = n - i + 1\right\} \\
L'(i) &= \max\left\{1 \le j < n \,\middle|\, N_j(P) = n - i + 1\right\}
\end{aligned}
$$

for $i \leftarrow 1$ **to** n
    $L'(i) \leftarrow 0$
for $j \leftarrow 1$ **to** $n - 1$
    $i \leftarrow n - N_j(P) + 1$
    $L'(i) \leftarrow j$
$L(2) \leftarrow L'(2)$
for $i \leftarrow 3$ **to** n
    $L(i) \leftarrow \max\{L(i-1), L'(i)\}$

i	1	2	3	4	5	6	7	8	9
P	c	a	b	d	a	b	d	a	b
L	0	0	0	0	6	6	6	6	6
L'	0	0	0	0	6	0	0	3	0
N	0	0	2	0	0	5	0	0	

$l'(i)$

- $l'(i)$ — длина наибольшего суффикса $P[i..n]$, который является префиксом P, если такой существует. Если его не существует, то $l'(i) = 0$.
- $l'(i) = \max\left\{1 \le j \le |P[i..n]| = n - i + 1 \middle| N_j(P) = j\right\}$

```
l'(n + 1) ← 0
for i ← n downto 1
        j ← n − i + 1
        if N_j(P) = j
                l'(i) ← j
        else l'(i) ← l'(i + 1)
```

i	1	2	3	4	5	6	7	8	9
P	a	b	a	b	c	a	b	a	b
l'	4	4	4	4	4	4	2	2	0
N	0	2	0	4	0	0	2	0	

Алгоритм Бойера-Мура

```
BM-Pattern-Search(P, T)
k ← n
while k ≤ m
        i ← n, h ← k
        while i > 0 and P(i) = T(h)
                i ← i − 1, h ← h − 1
        if i = 0 // Вхождение P в T, оканчивающиеся в k
                k ← k + n − l′(2)
        elseif i = n
                k ← k + 1
        else
                if L′(i + 1) > 0
                        s ← n − L′(i + 1)
                else s ← n − l′(i + 1)
                k ← k + max{1, s, i + 1 − R(x)}
```

Свойства алгоритма

- Для слабого правила хорошего суффикса если образец не встречается в тексте, то время работы в худшем случае $O(nm)$. Однако в среднем — сублинейное время.
- Сильное правило хорошего суффикса: $O(m)$, не больше $4m$ сравнений.
- Алфавито-независимый алгоритм (с точностью до организации $R(x)$).
- Доказательство линейности очень сложное, вместо него будет разобран алгоритм Апостолико-Джанкарло с простой оценкой количества сравнений $2m$ в худшем случае.

Раздел

Введение
- Основные определения
- Постановка задачи
- Очевидный метод

Предварительная обработка образца
- Z-блоки
- Построение Z-блоков за линейное время
- Поиск с использованием Z-блоков

Алгоритм Кнута-Морриса-Пратта
- Основной алгоритм
- Алгоритм реального времени

Алгоритм Бойера-Мура
- Основная идея
- Правило хорошего суффикса
- Реализация

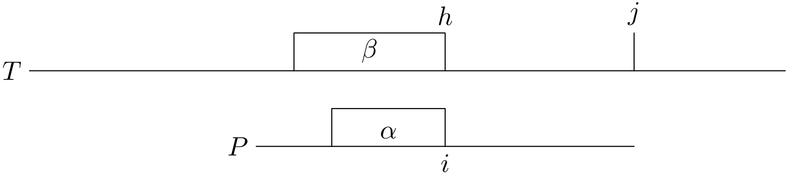
Алгоритм Апостолико-Джанкарло
- Имитация алгоритма Бойера-Мура
- Линейность

Основная идея

1. Имитация алгоритма Бойера-Мура, фазы от 1 до $q \leq m$.
2. Для каждой позиции текста T запоминаются значения $M(j) \leq l$, где l — количество совпавших символов с суффиксом P.
3. В случае, если сравниваются $P(i)$ и $T(h)$ то по ненулевым значениям N_i или $M(h)$ можно предсказать результат этого и последующих сравнений.

Сравнение $P(i)$ с $T(h)$

$|\alpha| = N_i$, $|\beta| = M(h) > N_i$: строки совпадают на N_i символов, но дальше различаются.



Одна фаза

Начало фазы: $h = j$, $i = n$

1. $M(h)$ не определено или $M(h) = N_i = 0$:
   - $T(h) = P(i)$, $i = 1$: вхождение образца, $M(j) \leftarrow n$, сдвиг.
   - $T(h) = P(i)$, $i > 1$: $h \leftarrow h - 1$, $i \leftarrow i - 1$, повтор.
   - $T(h) \neq P(i)$: $M(j) \leftarrow j - h$, сдвиг.
2. $M(h) < N_i$: $i \leftarrow i - M(h)$, $h \leftarrow h - M(h)$, повтор.
3. $M(h) \geq N_i$ и $N_i = i > 0$: вхождение образца, $M(j) \leftarrow j - h$ $(M(j) \leq n)$, сдвиг.
4. $M(h) > N_i$ и $N_i < i$: $P(i - N_i) \neq T(h - N_i)$, $M(j) \leftarrow j - h$ $(M(j) \leq j - h + N_i)$, сдвиг.
5. $M(h) = N_i$ и $0 < N_i < i$: $i \leftarrow i - M(h)$, $h \leftarrow h - M(h)$, повтор.

Корректность алгоритма Апостолико-Джанкарло

Теорема

Используя M *и* N *алгоритм Апостолико-Джанкарло правильно находит все вхождения* P *в* T*.*

Доказательство.

Алгоритм полностью имитирует алгоритм Бойера-Мура, за исключением пропуска некоторых сравнений. $\square$

Раздел

Введение
- Основные определения
- Постановка задачи
- Очевидный метод

Предварительная обработка образца
- Z-блоки
- Построение Z-блоков за линейное время
- Поиск с использованием Z-блоков

Алгоритм Кнута-Морриса-Пратта
- Основной алгоритм
- Алгоритм реального времени

Алгоритм Бойера-Мура
- Основная идея
- Правило хорошего суффикса
- Реализация

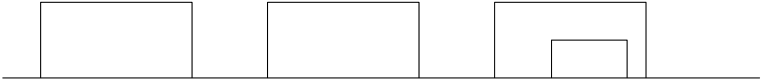
Алгоритм Апостолико-Джанкарло
- Имитация алгоритма Бойера-Мура
- Линейность

Вспомогательные определения

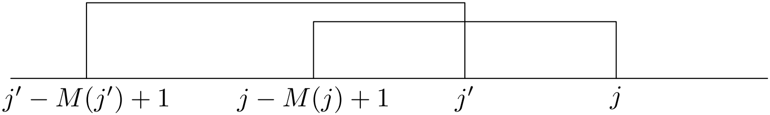
- Если $M(j) \neq 0$, то $[j - M(j) + 1..j]$ — покрытый интервал для j.
- Если для $j' < j$ и j определены покрытые интервалы, то они пересекаются, если $j' - M(j') + 1 < j - M(j) + 1 \leq j'$.

Интервалы

Непересекающиеся интервалы:



Пересекающиеся:



Непересечение интервалов

Теорема

Никакие два покрытых интервала, вычисленных алгоритмом, не пересекаются. Более того, если алгоритм проверяет позицию h *из* T *в покрытом интервале, то* h *— правый конец этого интервала.*

Непересечение интервалов: доказательство

Доказательство.

- Индукция по количеству интервалов.
- Перемещение h внутрь интервала возможно только в случаях 2 или 5, но тогда будет нарушено индукционное предположение.
- Новый интервал $[h+1..j]$ создаётся в случаях 1 (h не принадлежит ни одному интервалу), 3 и 4 (h — правый конец интервала), следовательно он не пересекает существующих интервалов.

$\square$

Линейность алгоритма

Теорема

Алгоритм Апостолико-Джанкарло выполняет не более $2m$ сравнений символов и не более $O(m)$ дополнительных действий.

Доказательство.

- Количество несовпадений $\leq m$ (завершение фазы и сдвиг.)
- Символы сравниваются в случае 1, результаты совпадений заносятся в $M(j)$, т.е. эти символы попадают в интервал.
- Символы внутри покрытых интервалов не сравниваются, т.е. m совпадений и $2m$ сравнений.
- Количество обращений к M имеет порядок $O(M)$, т.к. в случаях 3 и 4 происходит сдвиг, а в случаях 2 и 5 создастся новый интервал.

$\square$

Выводы

- Алгоритм полностью имитирует алгоритм Бойера-Мура.
- За счёт сохранения накопленной в процессе сравнивания образца с текстом пропускаются некоторые сравнения.
- Простая линейная оценка.