Поиск подстроки в строке

Постановка задачи

Строкой будем называть упорядоченный набор символов некоторого алфавита.

На вход поступают пары (S, P), где S - строка (string), P - образец (pattern).

Необходимо найти все или какую-то пару (i, j) : $S[i...j] == P$.

Тривиальный алгоритм

```
def Search(S, P):
  for i from 0 to |S| - |P|:      # O(S - P)
    if S[i...i + |P| - 1] == P:   # O(P)
      yield i
```

Время работы $O(P(S-P))$.

На практике работает неплохо, так как, как правило, $|P| << |S|$ и неравенство подстрок можно определить по первым символам (не обязательно смотреть на все символы P).

Но в худшем случае ($P \approx S/2$) время $\Theta(S^2)$.

Префикс-функция

Префикс-функция

Опр. *Префикс строки* S - подстрока вида $S[0 \dots i]$. *Собственный префикс* - префикс не совпадающий со всей строкой S.

Опр. *Суффикс строки* S - подстрока вида $S[i \dots n-1]$. *Собственный суффикс* - суффикс не совпадающий со всей строкой S.

Далее полагаем, что все префиксы и суффиксы собственные (если не указано иное).

Префикс-функция

Опр. *Префикс-функция строки S в позиции i* :
$\pi(S, i) =$ длина наибольшего префикса строки S, который совпадает с суффиксом префикса S длины $i + 1$.

Префикс-функция

Опр. *Префикс-функция строки S в позиции i* :
$\pi(S, i) =$ длина наибольшего префикса строки S, который совпадает с суффиксом префикса S длины $i + 1$.

То есть

$\pi(S, 0) = 0$
$\pi(S, i) = \max\{0 \le k \le i : S[0...k-1] == S[i-k+1...i]\}$

Далее для краткости будем опускать аргумент S, полагая, что он фиксирован.

Пример

Наивное построение префикс-функции

```
def PrefixFunction(S):
  p = [0,...,0]
  for i from 1 to |S| - 1:  # O(S)
    for k from i downto 1:  # O(S)
      if S[0...k-1] == S[i-k+1...i]:  # O(S)
        p[i] = k
        break
  return p
```

$O(S^3)$

Эффективное построение префикс-функции

Будем последовательно строить значения $\pi(0)$, $\pi(1)$, $\pi(2)$, ...

- Как связаны значения $\pi(i)$ и $\pi(i-1)$?

Эффективное построение префикс-функции

Будем последовательно строить значения $\pi(0)$, $\pi(1)$, $\pi(2)$, ...

- $\pi(i) \leq \pi(i-1) + 1$
- В каком случае $\pi(i) = \pi(i-1) + 1$?

Эффективное построение префикс-функции

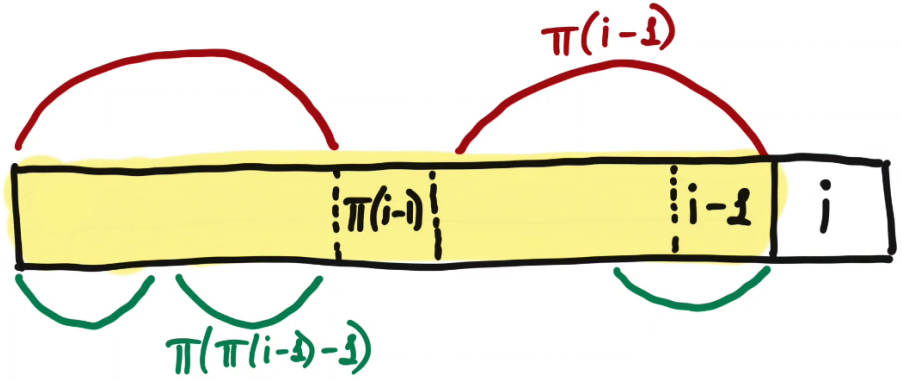
Будем последовательно строить значения $\pi(0)$, $\pi(1)$, $\pi(2)$, ...

- $\pi(i) \leq \pi(i-1) + 1$
- Если $S[i] == S[\pi(i-1)]$, то $\pi(i) = \pi(i-1) + 1$.
- Иначе нужно проверить следующий по величине подходящий префикс. Чему равна его длина?

Эффективное построение префикс-функции

Будем последовательно строить значения $\pi(0)$, $\pi(1)$, $\pi(2)$, ...

- $\pi(i) \leq \pi(i-1) + 1$
- Если $S[i] == S[\pi(i-1)]$, то $\pi(i) = \pi(i-1) + 1$.
- Иначе нужно проверить следующий по величине подходящий префикс. Его длина $k_1 = \pi(\pi(i-1) - 1)$.
- Если не подходит и он ($S[i] \neq S[k_1]$), то переходим к $k_2 = \pi(k_1 - 1)$, и так далее.



Построение префикс-функции: TLDR

- Перебираем потенциально подходящие размеры префикса строки S в порядке убывания, пока не найдем совпадение по следующему символу:
  $k_0 = \pi(i-1)$
  $k_1 = \pi(k_0 - 1)$
  $k_2 = \pi(k_1 - 1)$
  ...
- Если нашли $k' : S[i] == S[k']$, то $\pi(i) = k' + 1$.
- Иначе $\pi(i) = 0$.

Эффективное построение префикс-функции

```
def PrefixFunction(S):
  p = [0,...,0]
  for i from 1 to |S| - 1:
    k = p[i - 1]
    while S[i] != S[k] and k > 0:
      k = p[k - 1]
    if S[i] == S[k]:
      p[i] = k + 1
  return p
```

Цикл в цикле $\Rightarrow O(S^2)$. Или нет?..

Эффективное построение префикс-функции

```
def PrefixFunction(S):
  p = [0,...,0]
  for i from 1 to |S| - 1:
    k = p[i - 1]
    while S[i] != S[k] and k > 0:
      k = p[k - 1]  # 2
    if S[i] == S[k]:
      p[i] = k + 1  # 1
  return p
```

- На каждой итерации очередное значение $\pi(i)$ либо увеличивается на 1 (#1), либо уменьшается (#2).
- Общее количество уменьшений (while) $\leq$ Общее число увеличений на 1 (for).
- Общее число итераций for $= S - 1$, а значит число итераций while $\leq S - 1$
- Общее время построения $\Theta(S)$.

Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта

Дана строка S и образец P. Найти все вхождения P в S.

1. Пусть $S' = P + \# + S$
2. Строим префикс-функцию для строки S'.
3. Ответом будут все i: $\pi(S', i + |P| + 1) == |P|$

$T(S, P) = \Theta(S + P)$
$M(S, P) = \Theta(S + P)$

Алгоритм Кнута-Морриса-Пратта: улучшение

Дана строка S и образец P. Найти все вхождения P в S.

Так как $\pi(S', i) \leq |P|$, достаточно хранить $\pi(P, \cdot)$ и текущее значение $\pi(S', i)$.

1. Пусть $S' = P + \# + S$ (можно не хранить явно!).
2. Строим префикс-функцию для строки P.
3. $\forall i$ если $\pi(S', i + |P| + 1) == |P|$, то добавляем i в ответ.

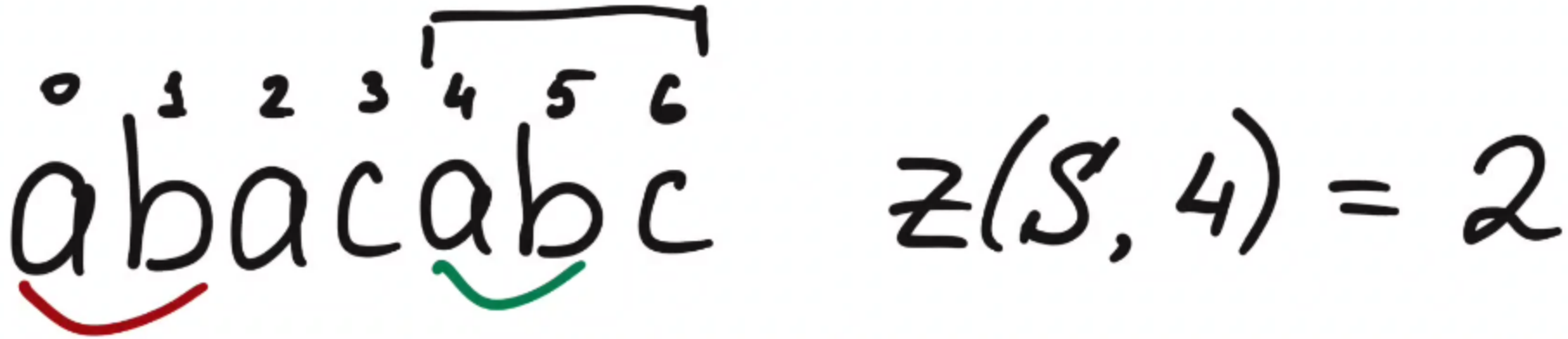
$T(S, P) = \Theta(S + P)$
$M(S, P) = \Theta(P)$

z-функция

z-функция

Опр. *z-функция строки S в позиции i* :
$z(S, i) =$ длина наибольшего префикса суффикса длины $|S| - i$ строки S, который совпадает с префиксом S.

$z(S, 0) := 0$ (иногда $|S|$)
$z(S, i) := \max\{k \geq 0 : S[0...k-1] == S[i...i+k-1]\}$



Пример

Наивное построение z-функции

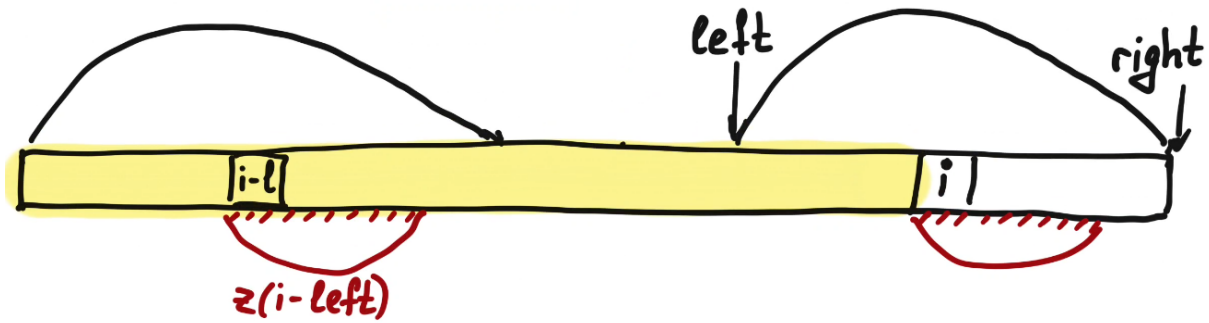
```
def ZFunction(S):
  z = [0,...,0]
  for i from 1 to |S| - 1:     # O(S)
    for k from 1 to |S| - i:   # O(S)
      if S[k - 1] != S[i + k - 1]:
        z[i] = k - 1
        break
  return z
```

$O(S^2)$

Эффективное построение z-функции

Будем последовательно строить значения $z(0)$, $z(1)$, $z(2)$, ...

- Назовем z-блоком пару $(i, j) : z(i) = j - i$.
- Пусть $right$ - самая правая граница j среди всех построенных z-блоков.
- Также $left$ - левая граница соответствующая $right$.
- Если $i < right$, то $z(i) \geq \min(z[i - left], right - i)$.
  Далее уточняем перебором.
- Иначе ($i \geq right$) придется считать $z[i]$ честно.



Эффективное построение z-функции

```
def ZFunction(S):
  z = [0,...,0]
  left = right = 0
  for i from 1 to |S| - 1:
    if i < right:
      z[i] = min(z[i - left], right - i)
    while i + z[i] < |S| and S[z[i]] == S[i + z[i]]:
      ++z[i];
    if (right < i + z[i]):
      left = i
      right = i + z[i]
  return z;
```

Цикл в цикле $\Rightarrow O(S^2)$. Или нет?..

Эффективное построение z-функции

```
def ZFunction(S):
  z = [0,...,0]
  left = right = 0
  for i from 1 to |S| - 1:
    if i < right:
      z[i] = min(z[i - left], right - i)
    while i + z[i] < |S| and S[z[i]] == S[i + z[i]]:
      ++z[i];
    if (right < i + z[i]):
      left = i
      right = i + z[i]
  return z;
```

На каждой итерации либо находим ответ за $O(1)$, либо вычисляем значение перебором. Во втором случае двигается $right$. А сдвинуться $right$ может не более чем S раз. $\Rightarrow \Theta(S)$

Алгоритм Кнута-Морриса-Пратта

По аналогии в алгоритме КМП можно использовать z-функцию вместо префикс-функции.

z-π-S преобразования

Задача

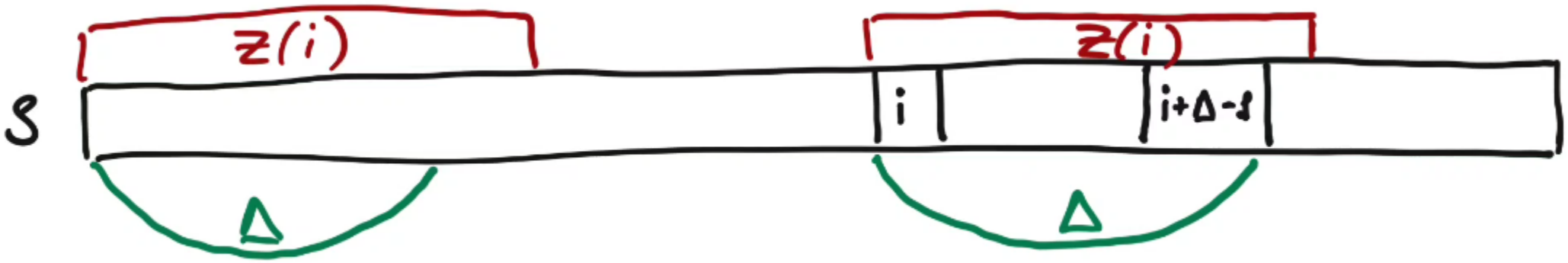
Ранее рассмотрели построение префикс- и z-функций по строке S.

Теперь зададимся вопросом, как перейти от одной функции к другой и как восстановить строку по заданным префикс- и z-функциям.

$z \to \pi$ преобразование

Дана z-функция, построить префикс-функцию.

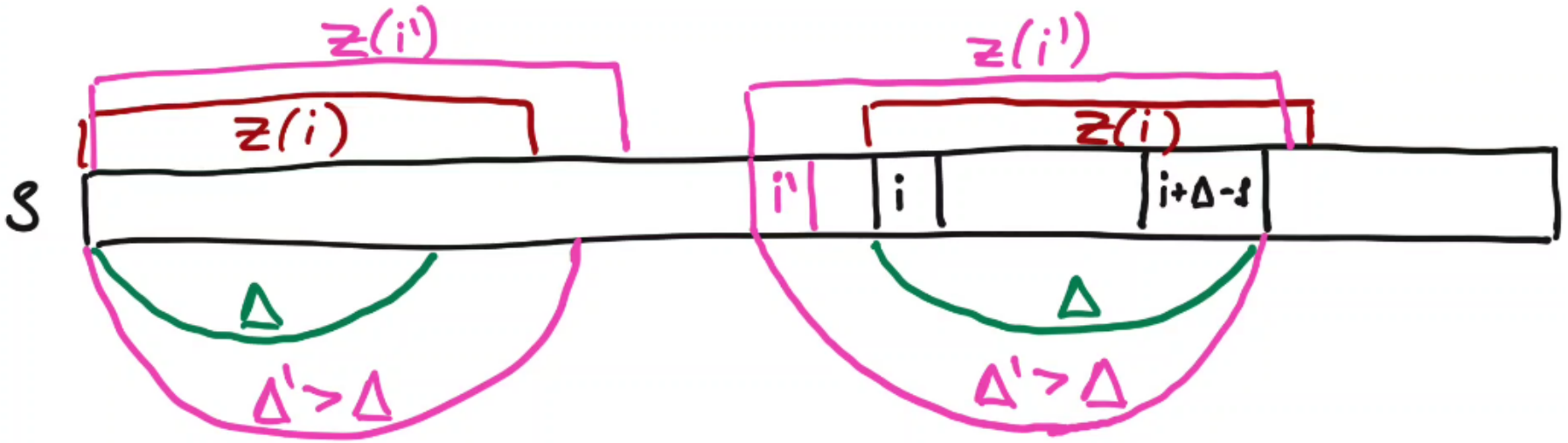
- Если $z(i) > 0$, то можем обновить значения $1 \le \Delta \le z(i), \pi(i + \Delta - 1) = \Delta$



$z \longrightarrow \pi$ преобразование

Дана z-функция z, построить префикс-функцию π.

- Если $z(i) > 0$, то можем обновить значения $1 \leq \Delta \leq z(i)$, $\pi(i + \Delta - 1) = \Delta$ (по убыванию Δ)
- Но! Если на каком-то шаге уже $\pi(i + \Delta - 1) > 0$, то процесс обновления нужно остановить, так как, он был выставлен при рассмотрении $z(i') : i' < i$ и значение $\pi(i + \Delta - 1)$ не увеличится.



$z \longrightarrow \pi$ преобразование

```
def z2p(z):
  p = [0,...,0]
  for i from 0 to |z| - 1:
    for delta from z[i] - 1 to 0:
      if p[i + delta] > 0:
        break
      p[i + delta] = delta + 1
  return p
```

Так как каждое значение π выставляется ровно один раз, сложность равна $O(|z|)$

$\pi \to S$ преобразование

Дана префикс-функция π, построить строку S.

Ясно, что таких S много. Легко реализовать тривиальный алгоритм построения какой-то из возможных строк (при достаточно большом алфавите).

Поставим задачу поиска лексикографически наименьшей строки, удовлетворяющей π.

$\pi \to S$ преобразование

Будем последовательно строить значения $S[i]$.

- Что делать, если $\pi(i) > 0$?

$\pi \to S$ преобразование

Будем последовательно строить значения $S[i]$.

- Если $\pi(i) > 0$, то выбора нет: $S[i] = S[\pi(i) - 1]$.
- Если $\pi(i) = 0$, найти лексикографически наименьший допустимый символ.
  А какие символы допустимы?

$\pi \to S$ преобразование

Будем последовательно строить значения $S[i]$.

- Если $\pi(i) > 0$, то выбора нет: $S[i] = S[\pi(i) - 1]$.
- Если $\pi(i) = 0$, найти лексикографически наименьший допустимый символ.
  Допустимый символ - тот, который не продолжает ни один из π-блоков.
  То есть недопустимые: $S[\pi(i-1)], S[\pi(\pi(i-1)-1)]$, ...

$\pi \to S$ преобразование

```
def p2S(p):
  S = ""
  S[0] = 'a'
  for i from 1 to |p| - 1:
    if p[i] > 0:
      S[i] = S[p(i) - 1]
    else:
      ban = {}
      k = p(i-1)
      while k > 0:
        ban.add(s[k])
        k = p(k - 1)
      for symbol from 'b' to 'z':  # почему начинаем с 'b'?
        if symbol not in ban:
          S[i] = symbol
          break
  return S
```

Как и построение префикс-функции работает за $O(|\pi|)$.