Лекция 4: Критическое разбиение и поиск по образцу

А. М. Шур

Институт математики и компьютерных наук (матмех) УрФУ

20 марта 2015 г.

Задача поиска в тексте по образцу

Задача поиска в тексте по образцу

Даны два слова: «текст» T длины n и «образец» x длины m (как правило, $n \gg m$). Требуется найти все позиции подслова x в T.

Основной вариант задачи выглядит так: образец известен заранее, а текст – нет.

Задача поиска в тексте по образцу

Даны два слова: «текст» T длины n и «образец» x длины m (как правило, $n \gg m$). Требуется найти все позиции подслова x в T.

Основной вариант задачи выглядит так: образец известен заранее, а текст – нет.

- Известно множество алгоритмов, которые выполняют препроцессинг образца, сохраняют его результаты и потом ищут образец, сравнивая его с текстом в режиме онлайн за время $O(n)$ (алгоритмы Кнута-Морриса-Пратта, Ахо-Корасик, Бойера-Мура, Апостолико-Джианкарло и другие)

Однако всем этим алгоритмам нужна дополнительная память размером $\Omega(m)$ (в предположении, что запись числа m входит в машинное слово и может считаться константой), а иногда и больше — $\Omega(|\Sigma|m)$, где Σ — алфавит образца.

Задача поиска в тексте по образцу

Даны два слова: «текст» T длины n и «образец» x длины m (как правило, $n \gg m$). Требуется найти все позиции подслова x в T.

Основной вариант задачи выглядит так: образец известен заранее, а текст – нет.

- Известно множество алгоритмов, которые выполняют препроцессинг образца, сохраняют его результаты и потом ищут образец, сравнивая его с текстом в режиме онлайн за время $O(n)$ (алгоритмы Кнута-Морриса-Пратта, Ахо-Корасик, Бойера-Мура, Апостолико-Джианкарло и другие)

Однако всем этим алгоритмам нужна дополнительная память размером $\Omega(m)$ (в предположении, что запись числа m входит в машинное слово и может считаться константой), а иногда и больше — $\Omega(|\Sigma| m)$, где Σ — алфавит образца.

- Алгоритмы, основанные на конечных автоматах, выполняют ровно n операций сравнения и требуют $\Omega(|\Sigma| m)$ памяти
- Алгоритмы, требующие $\Omega(m)$ памяти, выполняют в худшем случае около $2n$ операций сравнения

Задача поиска в тексте по образцу

Даны два слова: «текст» T длины n и «образец» x длины m (как правило, $n \gg m$). Требуется найти все позиции подслова x в T.

Основной вариант задачи выглядит так: образец известен заранее, а текст – нет.

- Известно множество алгоритмов, которые выполняют препроцессинг образца, сохраняют его результаты и потом ищут образец, сравнивая его с текстом в режиме онлайн за время $O(n)$ (алгоритмы Кнута-Морриса-Пратта, Ахо-Корасик, Бойера-Мура, Апостолико-Джианкарло и другие)

Однако всем этим алгоритмам нужна дополнительная память размером $\Omega(m)$ (в предположении, что запись числа m входит в машинное слово и может считаться константой), а иногда и больше — $\Omega(|\Sigma| m)$, где Σ — алфавит образца.

- Алгоритмы, основанные на конечных автоматах, выполняют ровно n операций сравнения и требуют $\Omega(|\Sigma| m)$ памяти
- Алгоритмы, требующие $\Omega(m)$ памяти, выполняют в худшем случае около $2n$ операций сравнения

Что делать, если объём доступной памяти не m, а константа?

Задача поиска «девичья память»

Задача поиска «девичья память»

Пусть текст T длины n и образец x длины m даны в виде read-only массивов и имеется константное число регистров, в которых можно хранить числа, не превосходящие n. Требуется найти все позиции подслова x в T.

Задача поиска «девичья память»

Пусть текст T длины n и образец x длины m даны в виде read-only массивов и имеется константное число регистров, в которых можно хранить числа, не превосходящие n. Требуется найти все позиции подслова x в T.

Теорема (Крошмор, Перрен, 1991)

Задачу «девичья память» можно решить, использовав не более $2n$ сравнений с текстом.

Наивный алгоритм

Чтобы объяснить алгоритм Крошмора-Перрена, нужен небольшой экскурс в алгоритмы поиска.

Наивный алгоритм

Чтобы объяснить алгоритм Крошмора-Перрена, нужен небольшой экскурс в алгоритмы поиска.

Наивный алгоритм поиска

- состоит из $n-m+1$ фаз; в i-й фазе первая позиция образца приложена к i-й позиции текста
  - физически, конечно, ничего никуда не прикладывается, есть пара указателей — на текущие позиции в образце и в тексте, но для объяснений такой образ удобен
- в каждой фазе происходит посимвольное сравнение образца с текущим блоком текста слева направо до первого несовпадения либо до конца образца, если несовпадений нет; во втором случае найдено вхождение образца и номер текущей фазы добавляется к ответу
- в худшем случае (например, при поиске образца $a^{m-1}b$ в тексте a^n) алгоритм совершит $m(n-m+1)$ сравнений, что недопустимо много

Наивный алгоритм

Чтобы объяснить алгоритм Крошмора-Перрена, нужен небольшой экскурс в алгоритмы поиска.

Наивный алгоритм поиска

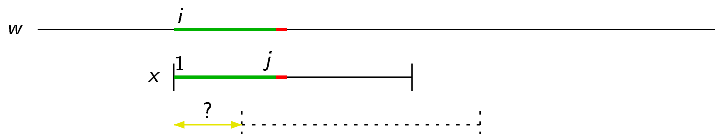
- состоит из $n-m+1$ фаз; в i-й фазе первая позиция образца приложена к i-й позиции текста
  - физически, конечно, ничего никуда не прикладывается, есть пара указателей — на текущие позиции в образце и в тексте, но для объяснений такой образ удобен
- в каждой фазе происходит посимвольное сравнение образца с текущим блоком текста слева направо до первого несовпадения либо до конца образца, если несовпадений нет; во втором случае найдено вхождение образца и номер текущей фазы добавляется к ответу
- в худшем случае (например, при поиске образца $a^{m-1}b$ в тексте a^n) алгоритм совершит $m(n-m+1)$ сравнений, что недопустимо много

Пути улучшения наивного алгоритма:

- уменьшить число фаз, пропуская те, которые «заведомо» не приведут к успеху
  - иными словами, мы хотим сдвигать образец более чем на один символ вправо
- пропускать сравнения, про которые уже известно, что они успешны

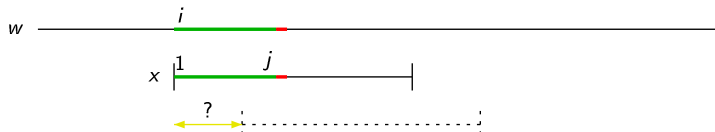
Алгоритм Морриса-Пратта

Момент окончания i-й фазы (пунктиром – следующая фаза):



Алгоритм Морриса-Пратта

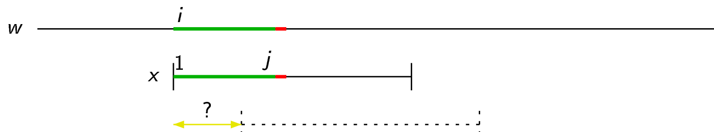
Момент окончания i-й фазы (пунктиром – следующая фаза):



Определение величины сдвига:

- в тексте, начиная с i-й позиции, записано слово $x[1..j]$; при небольших сдвигах его префикс (в образце) наложится на его суффикс в тексте

Момент окончания i-й фазы (пунктиром – следующая фаза):

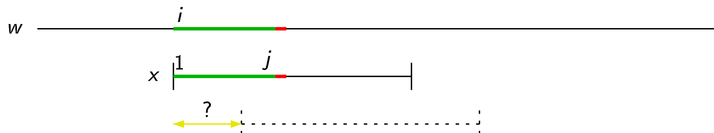


Определение величины сдвига:

- в тексте, начиная с i-й позиции, записано слово $x[1..j]$; при небольших сдвигах его префикс (в образце) наложится на его суффикс в тексте
- префикс $x[1..j-p]$ и суффикс $x[p+1..j]$ слова $x[1..j]$ совпадают тогда и только тогда, когда p – период $x[1..j]$

Алгоритм Морриса-Пратта

Момент окончания i-й фазы (пунктиром – следующая фаза):

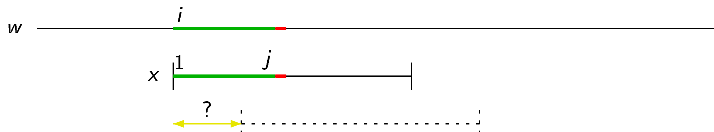


Определение величины сдвига:

- в тексте, начиная с i-й позиции, записано слово $x[1..j]$; при небольших сдвигах его префикс (в образце) наложится на его суффикс в тексте
- префикс $x[1..j-p]$ и суффикс $x[p+1..j]$ слова $x[1..j]$ совпадают тогда и только тогда, когда p – период $x[1..j]$
- если мы выполним сдвиг на минимальный период p_j слова $x[1..j]$, то пропустим только заведомо неуспешные фазы; именно этот сдвиг выполняет алгоритм Морриса-Пратта (при $j = 0$ выполняется сдвиг на 1 символ)

Алгоритм Морриса-Пратта

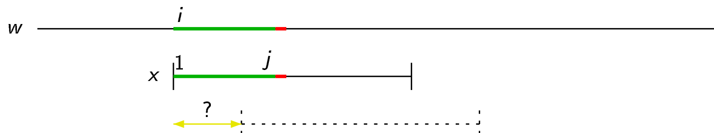
Момент окончания i-й фазы (пунктиром – следующая фаза):



Определение величины сдвига:

- в тексте, начиная с i-й позиции, записано слово $x[1..j]$; при небольших сдвигах его префикс (в образце) наложится на его суффикс в тексте
- префикс $x[1..j{-}p]$ и суффикс $x[p{+}1..j]$ слова $x[1..j]$ совпадают тогда и только тогда, когда p – период $x[1..j]$
- если мы выполним сдвиг на минимальный период p_j слова $x[1..j]$, то пропустим только заведомо неуспешные фазы; именно этот сдвиг выполняет алгоритм Морриса-Пратта (при $j = 0$ выполняется сдвиг на 1 символ)
- после сдвига сравнения начинаются с $(j{-}p_j{+}1)$-й позиции образца и $(i{+}j)$-й позиции текста, поскольку первые $(j{-}p_j)$ сравнений заведомо успешны

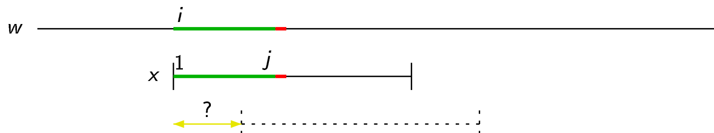
Момент окончания i-й фазы (пунктиром – следующая фаза):



Определение величины сдвига:

- в тексте, начиная с i-й позиции, записано слово $x[1..j]$; при небольших сдвигах его префикс (в образце) наложится на его суффикс в тексте
- префикс $x[1..j-p]$ и суффикс $x[p+1..j]$ слова $x[1..j]$ совпадают тогда и только тогда, когда p – период $x[1..j]$
- если мы выполним сдвиг на минимальный период p_j слова $x[1..j]$, то пропустим только заведомо неуспешные фазы; именно этот сдвиг выполняет алгоритм Морриса-Пратта (при $j = 0$ выполняется сдвиг на 1 символ)
- после сдвига сравнения начинаются с $(j-p_j+1)$-й позиции образца и $(i+j)$-й позиции текста, поскольку первые $(j-p_j)$ сравнений заведомо успешны
- ★ алгоритм Морриса-Пратта хранит массив значений p_j для $j = 1, \ldots, m$, полученный при препроцессинге образца

Момент окончания i-й фазы (пунктиром – следующая фаза):



Определение величины сдвига:

- в тексте, начиная с i-й позиции, записано слово $x[1..j]$; при небольших сдвигах его префикс (в образце) наложится на его суффикс в тексте
- префикс $x[1..j-p]$ и суффикс $x[p+1..j]$ слова $x[1..j]$ совпадают тогда и только тогда, когда p – период $x[1..j]$
- если мы выполним сдвиг на минимальный период p_j слова $x[1..j]$, то пропустим только заведомо неуспешные фазы; именно этот сдвиг выполняет алгоритм Морриса-Пратта (при $j = 0$ выполняется сдвиг на 1 символ)
- после сдвига сравнения начинаются с $(j-p_j+1)$-й позиции образца и $(i+j)$-й позиции текста, поскольку первые $(j-p_j)$ сравнений заведомо успешны

★ алгоритм Морриса-Пратта хранит массив значений p_j для $j = 1, \dots, m$, полученный при препроцессинге образца

★ алгоритм Морриса-Пратта выполняет не более 1 успешного сравнения с каждым символом текста и не более 1 неуспешного сравнения в каждой фазе, всего меньше $2n - m + 1$ сравнений

Можно ли избавиться от необходимости хранить массив периодов p_j и вычислять их на лету? В частном случае – да.

Автомаксимальные слова

Можно ли избавиться от необходимости хранить массив периодов p_j и вычислять их на лету? В частном случае – да.

Для слова w над упорядоченным алфавитом обозначим его лексикографически максимальный суффикс через $\mathrm{maxsuf}(w)$.

Определение

Слово w называется автомаксимальным, если $\mathrm{maxsuf}(w) = w$.

Можно ли избавиться от необходимости хранить массив периодов p_j и вычислять их на лету? В частном случае – да.

Для слова w над упорядоченным алфавитом обозначим его лексикографически максимальный суффикс через $\mathrm{maxsuf}(w)$.

Определение

Слово w называется автомаксимальным, если $\mathrm{maxsuf}(w) = w$.

Замечание

Любой префикс автомаксимального слова — автомаксимальное слово.

Пусть $u < w$ и $k = |u| < |w| = m$. Тогда для любого j $u[1..k-j] < w[1..m-j]$. Это очевидно, если u — префикс w. Если же $u[1..i] = w[1..i]$ и $u[i+1] < w[i+1]$, то либо $i+1 \leqslant k-j$, и это условие продолжает выполняться, либо $k-j < i$ и $u[1..k-j]$ является префиксом $w[1..m-j]$. □

Автомаксимальные слова - 2

Лемма о периоде автомаксимального слова

Минимальный период автомаксимального слова (и заодно всех его префиксов) вычисляется следующей простой функцией.

```
period:=1
for i:=2 to m do
    if x[i]<>x[i-period] then period:=i;
return period
```

Лемма о периоде автомаксимального слова

Минимальный период автомаксимального слова (и заодно всех его префиксов) вычисляется следующей простой функцией.

```
period:=1
for i:=2 to m do
    if x[i]<>x[i-period] then period:=i;
return period
```

Пусть при добавлении i-й буквы период нарушился, а y — длиннейший префикс в $x[1..i-1]$, являющийся суффиксом этого слова (т.е. $i-1-|y|$ – минимальный период $x[1..i-1]$). Тогда $a = x[i] < x[i-period] = b$, т.к. суффикс ya автомаксимального слова $x[1..i]$ меньше самого слова. Если $\mathrm{per}(x[1..i]) < i$, то некий префикс za являлся бы суффиксом, как на рисунке ($|z| < |y|$ по выбору y; возможно, $|z| = \lambda$). Но тогда суффикс слова x, начинающийся с zb, больше самого x, начинающегося с za, противоречие. □

$x[1..i] =$ [diagram: y with z, a; z, b; ... y with z, a]

Следующий алгоритм производит поиск по автомаксимальному образцу:

```
i:=1; j:=0; period:=1;
while i<=n-m+1 do
    while j<m and x[j+1]=T[j+i] do
       j:=j+1;
       if j>period and x[j]<>x[j-period] then period:=j;
    if j=m then return MATCH-AT i;
    i:=i+period;
    j:=j-period;
    if j<period then recompute(period);
return NO-MATCH
```

Поиск по автомаксимальному образцу

Следующий алгоритм производит поиск по автомаксимальному образцу:

```
i:=1; j:=0; period:=1;
while i<=n-m+1 do
    while j<m and x[j+1]=T[j+i] do
       j:=j+1;
       if j>period and x[j]<>x[j-period] then period:=j;
    if j=m then return MATCH-AT i;
    i:=i+period;
    j:=j-period;
    if j<period then recompute(period);
return NO-MATCH
```

Сдвиг текста относительно образца происходит как в алгоритме Морриса-Пратта, только периоды не хранятся, а вычисляются на ходу.

- Поскольку сдвиги такие же, как у Морриса-Пратта, то и число сравнений образца с текстом такое же ($< 2n$); к сожалению, присутствуют еще сравнения внутри образца

Поиск автомаксимального суффикса

На основе алгоритма вычисления периода автомаксимального слова можно написать алгоритм, вычисляющий maxsuf для любого слова x за линейное время с использованием константной памяти:

Поиск автомаксимального суффикса

На основе алгоритма вычисления периода автомаксимального слова можно написать алгоритм, вычисляющий maxsuf для любого слова x за линейное время с использованием константной памяти:

- вычисляем период для x как для автомаксимального слова (однобуквенные слова автомаксимальны, значит, некоторый префикс x автомаксимален)
- если $x[i] = x[i{-}period]$, продолжаем вычисление (автомаксимальность не нарушена)
- если $x[i] < x[i{-}period]$, продолжаем вычисление (автомаксимальность не нарушена)

Поиск автомаксимального суффикса

На основе алгоритма вычисления периода автомаксимального слова можно написать алгоритм, вычисляющий maxsuf для любого слова x за линейное время с использованием константной памяти:

- вычисляем период для x как для автомаксимального слова (однобуквенные слова автомаксимальны, значит, некоторый префикс x автомаксимален)
- если $x[i] = x[i - period]$, продолжаем вычисление (автомаксимальность не нарушена)
- если $x[i] < x[i - period]$, продолжаем вычисление (автомаксимальность не нарушена)
- если $x[i] > [i - period]$, найден суффикс $x[period+1..m]$, больший x; переходим к обработке этого суффикса
  - ★ более точно, для любого натурального c такого, что $c \cdot period < i$, суффикс $x[c \cdot period+1..m]$ больше x, т.е. можно перейти к обработке суффикса $x[\ell..m]$, где $\ell = \lfloor i/period \rfloor \cdot period$

Поиск автомаксимального суффикса

На основе алгоритма вычисления периода автомаксимального слова можно написать алгоритм, вычисляющий maxsuf для любого слова x за линейное время с использованием константной памяти:

- вычисляем период для x как для автомаксимального слова (однобуквенные слова автомаксимальны, значит, некоторый префикс x автомаксимален)
- если $x[i] = x[i{-}period]$, продолжаем вычисление (автомаксимальность не нарушена)
- если $x[i] < x[i{-}period]$, продолжаем вычисление (автомаксимальность не нарушена)
- если $x[i] > [i{-}period]$, найден суффикс $x[period{+}1..m]$, больший x; переходим к обработке этого суффикса
  - ★ более точно, для любого натурального c такого, что $c \cdot period < i$, суффикс $x[c \cdot period{+}1..m]$ больше x, т.е. можно перейти к обработке суффикса $x[\ell..m]$, где $\ell = \lfloor i/period \rfloor \cdot period$
- если дошли до конца x, то текущее обрабатываемое слово автомаксимально, а значит, является максимальным суффиксом в x.

Поиск автомаксимального суффикса

На основе алгоритма вычисления периода автомаксимального слова можно написать алгоритм, вычисляющий maxsuf для любого слова x за линейное время с использованием константной памяти:

- вычисляем период для x как для автомаксимального слова (однобуквенные слова автомаксимальны, значит, некоторый префикс x автомаксимален)
- если $x[i] = x[i{-}period]$, продолжаем вычисление (автомаксимальность не нарушена)
- если $x[i] < x[i{-}period]$, продолжаем вычисление (автомаксимальность не нарушена)
- если $x[i] > [i{-}period]$, найден суффикс $x[period{+}1..m]$, больший x; переходим к обработке этого суффикса
  - ★ более точно, для любого натурального c такого, что $c \cdot period < i$, суффикс $x[c \cdot period{+}1..m]$ больше x, т.е. можно перейти к обработке суффикса $x[\ell..m]$, где $\ell = \lfloor i/period \rfloor \cdot period$
- если дошли до конца x, то текущее обрабатываемое слово автомаксимально, а значит, является максимальным суффиксом в x.

Благодаря ★, если слово обрабатывалось в течение ℓ шагов до перехода к суффиксу, то суффикс по крайней мере на $\ell/2$ шагов короче. Это гарантирует, что суммарное число шагов не превосходит $2m$.

Поиск слова по частям

Идея поиска по образцу x: записать $x = uv$, где $v = \mathrm{maxsuf}(x)$, искать v как автомаксимальное слово и каждый раз, когда вхождение v в текст найдено, проверять, есть ли примыкающее к нему слева вхождение u. Для эффективной реализации этой идеи нужны дополнительные соображения.

Поиск слова по частям

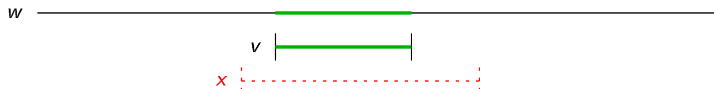
Идея поиска по образцу x: записать $x = uv$, где $v = \mathrm{maxsuf}(x)$, искать v как автомаксимальное слово и каждый раз, когда вхождение v в текст найдено, проверять, есть ли примыкающее к нему слева вхождение u. Для эффективной реализации этой идеи нужны дополнительные соображения.

Мы полагаем $r = |v|$, $\ell = m - r = |u|$.

Лемма о вхождениях u

Если $T[i..i{+}r{-}1] = v$, то $T[j..j{+}\ell{-}1] \neq u$ для всех $j = i{-}\ell{+}1, \ldots, i$.

Лемма утверждает, что x не может входить в T так, как это показано пунктиром:



Суффикс штрихового x, начинающийся с «зеленого» v, больше $v = \mathrm{maxsuf}(x)$. □

Поиск слова по частям

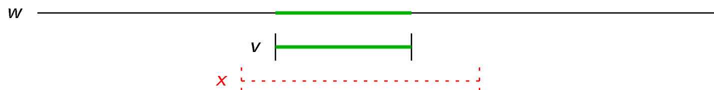
Идея поиска по образцу x: записать $x = uv$, где $v = \text{maxsuf}(x)$, искать v как автомаксимальное слово и каждый раз, когда вхождение v в текст найдено, проверять, есть ли примыкающее к нему слева вхождение u. Для эффективной реализации этой идеи нужны дополнительные соображения.

Мы полагаем $r = |v|$, $\ell = m - r = |u|$.

Лемма о вхождениях u

Если $T[i..i+r-1] = v$, то $T[j..j+\ell-1] \neq u$ для всех $j = i-\ell+1, \ldots, i$.

Лемма утверждает, что x не может входить в T так, как это показано пунктиром:



Суффикс штрихового x, начинающийся с «зеленого» v, больше $v = \text{maxsuf}(x)$. □

- После найденного вхождения v надо проверить, помещается ли слева от него u в соответствии с леммой о вхождениях; если нет – сравнение с u пропускаем
- Все фрагменты текста, сравниваемые с u, не пересекаются (и даже не примыкают друг к другу), т.е. сравнений с u производится менее n
- Запоминание предыдущего вхождения v требует константной памяти

Критическое разбиение и лемма о сдвиге

Пусть разбиение $x = uv$ является критическим (напомним – см. доказательство теоремы о критическом разбиении в лекции 3 – что надо вычислить максимальные суффиксы x для двух лексикографических порядков, порожденных обратными порядками на алфавите, и взять за v более короткий из них). Отметим, что $\ell > 0$.

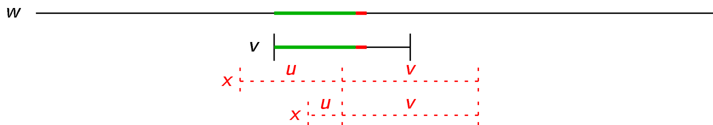
Критическое разбиение и лемма о сдвиге

Пусть разбиение $x = uv$ является критическим (напомним – см. доказательство теоремы о критическом разбиении в лекции 3 – что надо вычислить максимальные суффиксы x для двух лексикографических порядков, порожденных обратными порядками на алфавите, и взять за v более короткий из них). Отметим, что $\ell > 0$.

Лемма о сдвиге

Если при сравнении v с образцом в текущей фазе произошло несовпадение на шаге $j{+}1$, можно безопасно сдвинуть образец на $j{+}1$ позиций.

Лемма утверждает, что оба варианта «пунктирных» вхождений x в T невозможны:



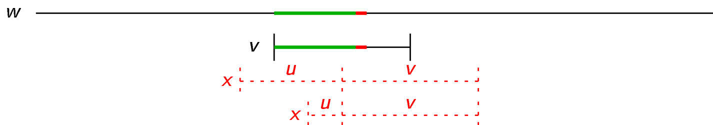
Верхний вариант расположения x противоречит лемме о суффиксе из лекции 3: никакой непустой суффикс u не может совпадать с префиксом v.

Пусть разбиение $x = uv$ является критическим (напомним – см. доказательство теоремы о критическом разбиении в лекции 3 – что надо вычислить максимальные суффиксы x для двух лексикографических порядков, порожденных обратными порядками на алфавите, и взять за v более короткий из них). Отметим, что $\ell > 0$.

Лемма о сдвиге

Если при сравнении v с образцом в текущей фазе произошло несовпадение на шаге $j+1$, можно безопасно сдвинуть образец на $j+1$ позиций.

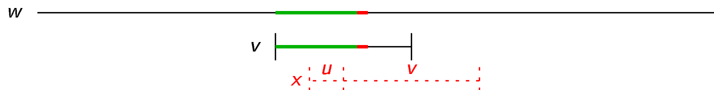
Лемма утверждает, что оба варианта «пунктирных» вхождений x в T невозможны:



Верхний вариант расположения x противоречит лемме о суффиксе из лекции 3: никакой непустой суффикс u не может совпадать с префиксом v.

Нижний вариант – на следующем слайде $\Longrightarrow$

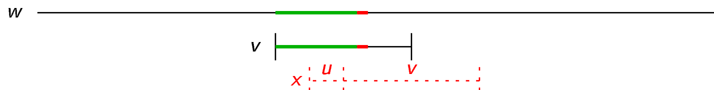
Покажем, что вхождение x в T так, как на рисунке, невозможно:



Если такое вхождение есть, то u — подслово в v; пусть $v = zuy$. Тогда $|zu|$ – локальный период x в точке $|u|$, а значит – период x (см. док-во теоремы о критическом разбиении).

Критическое разбиение и лемма о сдвиге (окончание)

Покажем, что вхождение x в T так, как на рисунке, невозможно:

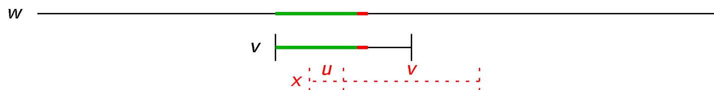


Если такое вхождение есть, то u — подслово в v; пусть $v = zuy$. Тогда $|zu|$ – локальный период x в точке $|u|$, а значит – период x (см. док-во теоремы о критическом разбиении).

- Если мы сдвигаем v так, что слева от текущей позиции v может находиться u, то сдвиг произошел на период x (а значит, на период v)

Критическое разбиение и лемма о сдвиге (окончание)

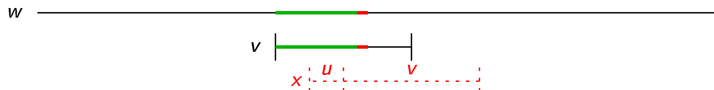
Покажем, что вхождение x в T так, как на рисунке, невозможно:



Если такое вхождение есть, то u — подслово в v; пусть $v = zuy$. Тогда $|zu|$ – локальный период x в точке $|u|$, а значит – период x (см. док-во теоремы о критическом разбиении).

- Если мы сдвигаем v так, что слева от текущей позиции v может находиться u, то сдвиг произошел на период x (а значит, на период v)
- Тогда символ текста, который дал несовпадение в предыдущей фазе, в новой фазе будет сравниваться с таким же символом, т.е. снова даст несовпадение

Покажем, что вхождение x в T так, как на рисунке, невозможно:



Если такое вхождение есть, то u — подслово в v; пусть $v = zuy$. Тогда $|zu|$ – локальный период x в точке $|u|$, а значит – период x (см. док-во теоремы о критическом разбиении).

- Если мы сдвигаем v так, что слева от текущей позиции v может находиться u, то сдвиг произошел на период x (а значит, на период v)
- Тогда символ текста, который дал несовпадение в предыдущей фазе, в новой фазе будет сравниваться с таким же символом, т.е. снова даст несовпадение
- Итак, при сдвиге, как на картинке, либо u, либо v гарантированно не совпадут с соответствующим блоком текста

Лемма о сдвигах доказана.

Алгоритм Крошмора-Перрена

Согласно лемме о сдвигах мы можем, сравнивая v с текстом в текущей фазе

- При несовпадении сдвигаться за последнюю просмотренную позицию текста
- При нахождении v, сдвигаться на период v (его можно запомнить) и сравнивать, начиная с первой непросмотренной позиции текста

Таким образом, число сравнений v с текстом равно n. Поскольку число сравнений u с текстом меньше n, мы доказали теорему Крошмора-Перрена.

Алгоритм Крошмора-Перрена

Согласно лемме о сдвигах мы можем, сравнивая v с текстом в текущей фазе

- При несовпадении сдвигаться за последнюю просмотренную позицию текста
- При нахождении v, сдвигаться на период v (его можно запомнить) и сравнивать, начиная с первой непросмотренной позиции текста

Таким образом, число сравнений v с текстом равно n. Поскольку число сравнений u с текстом меньше n, мы доказали теорему Крошмора-Перрена.

Дадим компактное неформальное описание алгоритма Крошмора-Перрена:

- Вычислить критическое разбиение $x = uv$ и (параллельно) период $\mathrm{per}(v)$ автомаксимального слова v
- Искать v в тексте слева направо, запоминая каждый раз последнее вхождение (и только его)
- Если фаза завершилась несовпадением j-го символа, сдвинуть v на j, в новой фазе сравнивать, начиная с первого символа
- Если в i-й фазе найдено вхождение v,
  - сравнить предшествующие вхождению ℓ символов текста с u, при совпадении вернуть вхождение x в позиции $i-\ell$
  - сдвинуть v на $\mathrm{per}(v)$, в новой фазе сравнивать, начиная с $(|v|-\mathrm{per}(v)+1)$-го символа

Более эффективная эксплуатация девичьей памяти

Алгоритм Крошмора-Перрена является линейным онлайновым алгоритмом (т.е. способен дать ответ для любого префикса текста в момент, когда этот префикс становится известен), но он не удовлетворяет более сильному условию real-time, означающему, что на обработку каждого входного символа должно тратиться константное число операций. Действительно, фаза, в которой регистрируется вхождение шаблона, требует $|u|$ сравнений на поиск u в тексте.

- Real-time алгоритмы поиска нужны для обработки непрерывно поступающих данных, например, для отслеживания сигнатур известных атак и вредоносных программ в сетевом трафике

Более эффективная эксплуатация девичьей памяти

Алгоритм Крошмора-Перрена является линейным онлайновым алгоритмом (т.е. способен дать ответ для любого префикса текста в момент, когда этот префикс становится известен), но он не удовлетворяет более сильному условию real-time, означающему, что на обработку каждого входного символа должно тратиться константное число операций. Действительно, фаза, в которой регистрируется вхождение шаблона, требует $|u|$ сравнений на поиск u в тексте.

- Real-time алгоритмы поиска нужны для обработки непрерывно поступающих данных, например, для отслеживания сигнатур известных атак и вредоносных программ в сетевом трафике

В 2011 году Бреслауэром, Гросси и Миньоси была предложена real-time версия алгоритма Крошмора-Перрена. Она опирается на две основные идеи:

- можно «синхронно» искать v вперед, а u – назад от текущей позиции, организовав это так, чтобы условия, на которых базируется оценка алгоритма Крошмора-Перрена, сохранялась
  - в случае, когда $|u| \leqslant |v|$, этой идеи достаточно для онлайн алгоритма
- если $|u| > |v|$, то для критического разбиения $u = yz$ слова u условие $|y| \leqslant |z|$ обязательно выполнится (упражнение); значит, можно параллельно запустить две копии «синхронной» версии алгоритма Крошмора-Перрена – одну для разбиения x, а другую – для разбиения u, и в совокупности они найдут все вхождения x