Computing q-gram Non-overlapping Frequencies on SLP Compressed Texts

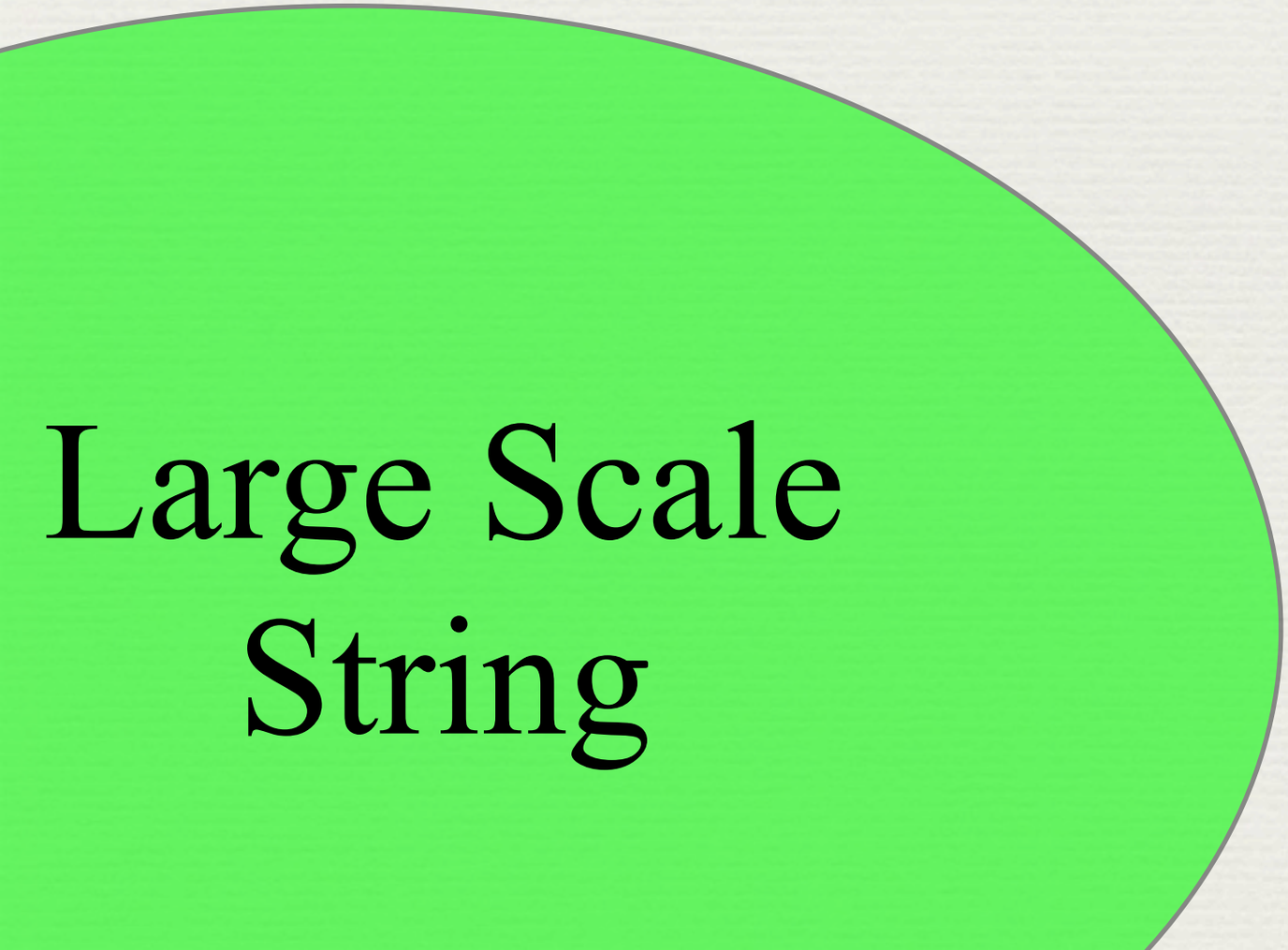
Kyushu University

◯Keisuke Goto, Hideo Bannai,

Shunsuke Inenaga, Masayuki Takeda

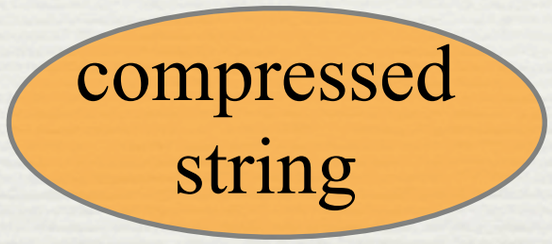
Background

- Large scale string data is usually stored in compressed form.



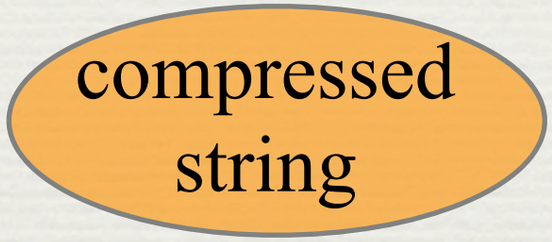
Background

- Large scale string data is usually stored in compressed form.



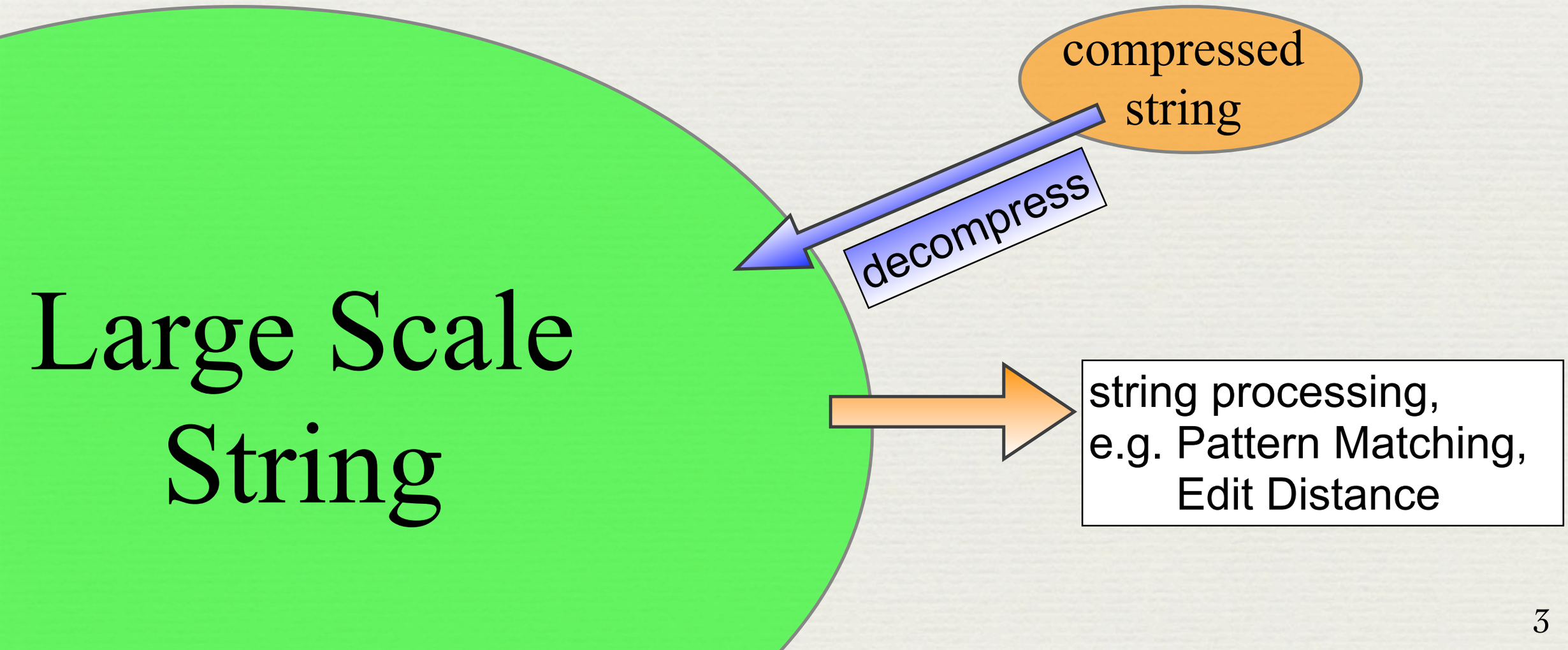
Background

- Large scale string data is usually stored in compressed form.
- In order to process such data, we usually need to decompress, which requires a lot of space and time.



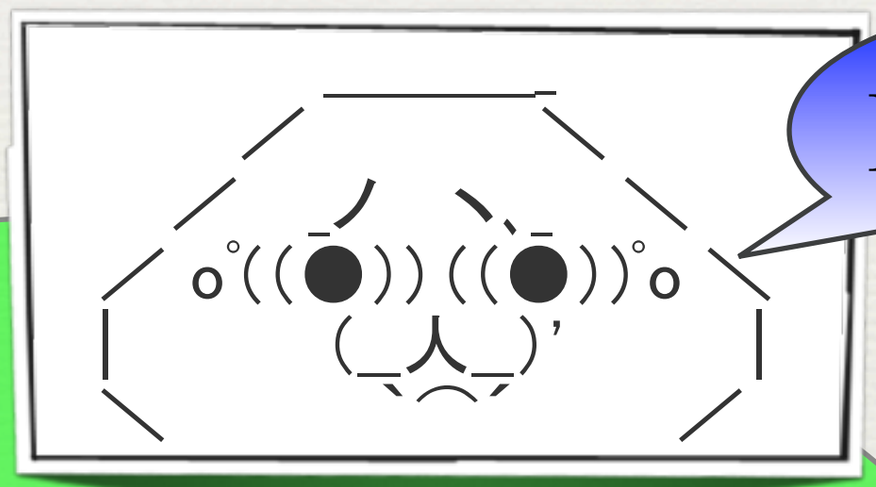
Background

- Large scale string data is usually stored in compressed form.
- In order to process such data, we usually need to decompress, which requires a lot of space and time.



Background

- Large scale string data is usually stored in compressed form.
- In order to process such data, we usually need to decompress, which requires a lot of space and time.



Large Scale String

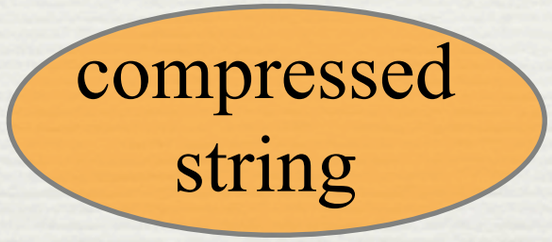
compressed string

decompress

string processing,
e.g. Pattern Matching,
Edit Distance

Background

- Large scale string data is usually stored in compressed form.
- In order to process such data, we usually need to decompress, which requires a lot of space and time.
- One solution is to process compressed strings **without explicit decompression.**



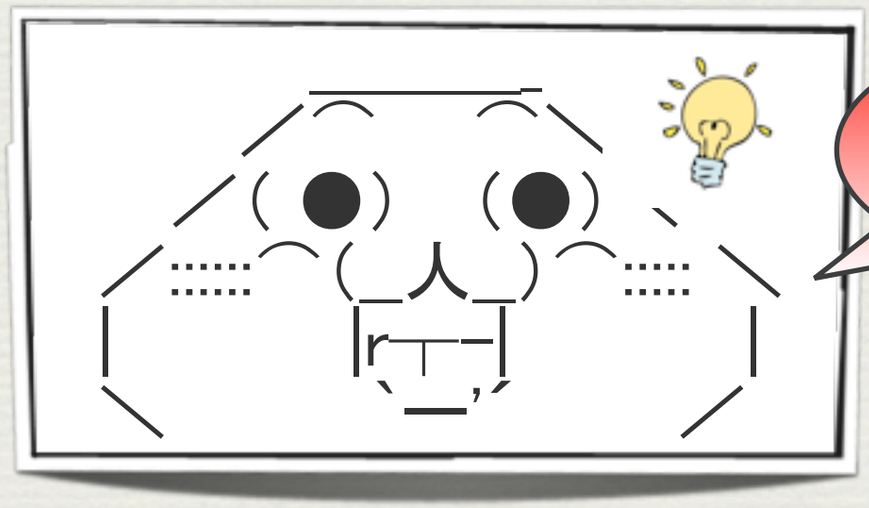
[1]

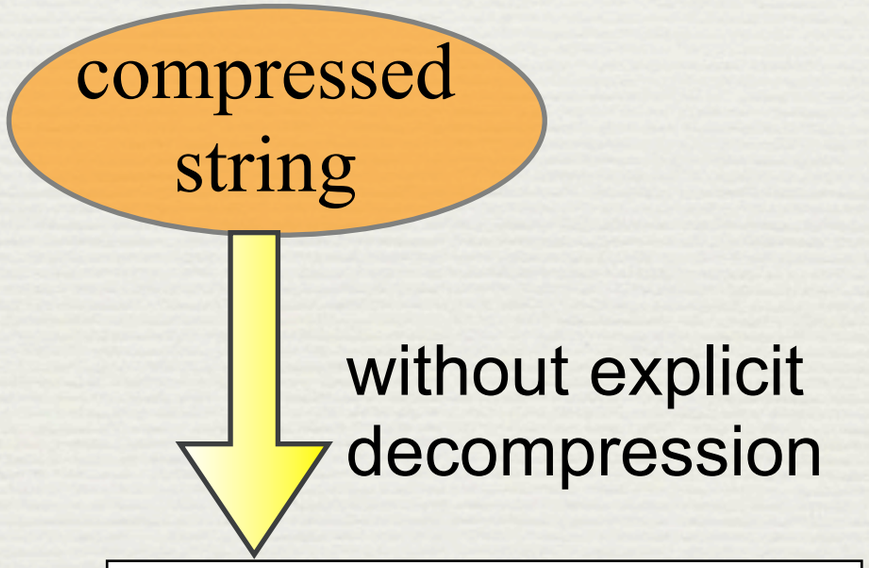
[2]

[1] **[Lifshits, Y. 2007]**
[2] **[Hermelin, D. 2009]**

Background

- Large scale string data is usually stored in compressed form.
- In order to process such data, we usually need to decompress, which requires a lot of space and time.
- One solution is to process compressed strings **without explicit decompression.**



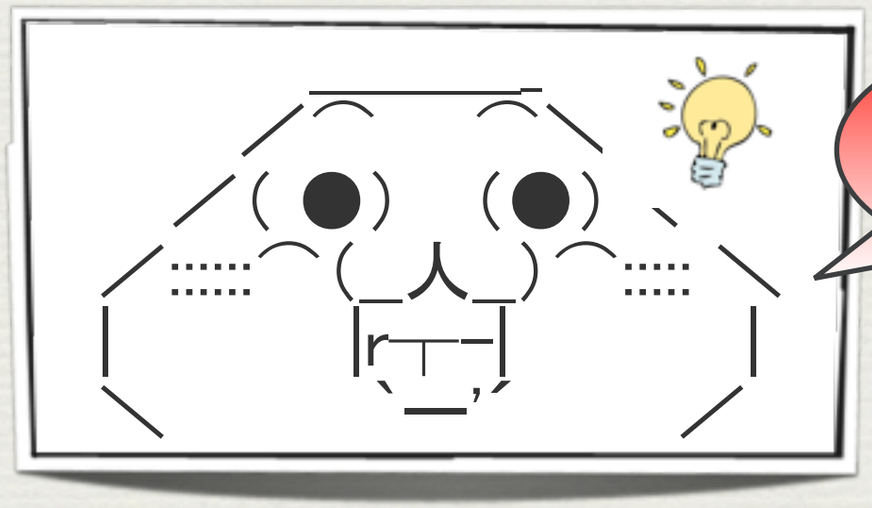


[1] **[Lifshits, Y. 2007]**

[2] **[Hermelin, D. 2009]**

Background

- Large scale string data is usually stored in compressed form.
- In order to process such data, we usually need to decompress, which requires a lot of space and time.
- One solution is to process compressed strings **without explicit decompression.**



compressed string

without explicit decompression

string processing,
e.g. Pattern Matching,[1]
Edit Distance[2]

this work: algorithm for computing
***q*-gram non-overlapping frequencies**
on compressed strings

[1] **[Lifshits, Y. 2007]**

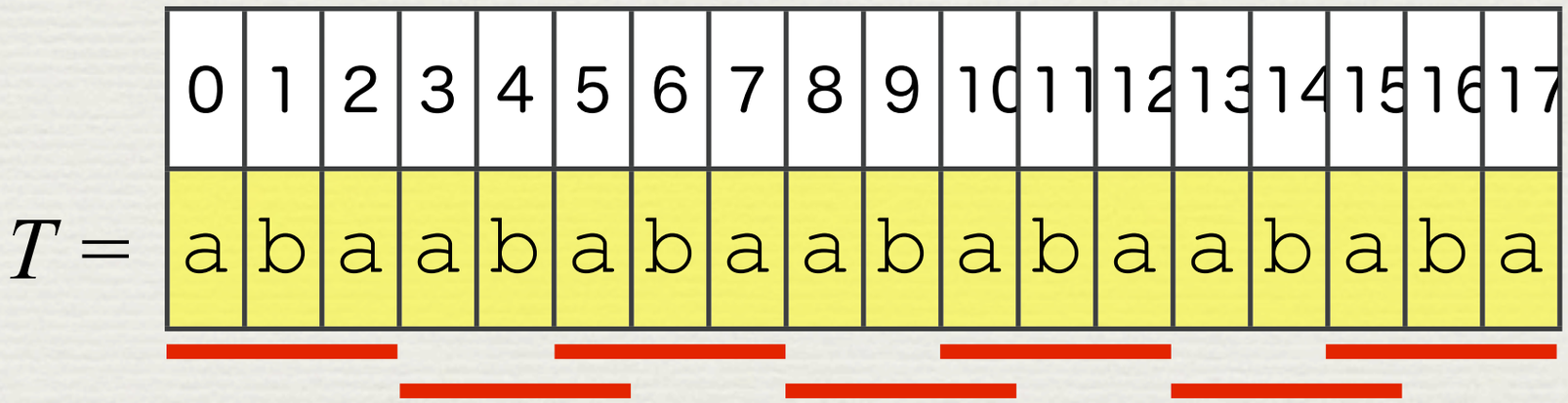
[2] **[Hermelin, D. 2009]**

Occurrence and Frequency

- Occurrence of pattern P in text T:

$$Occ(T, P)=\{\ i \mid\ T[i..(i+|P|-1)] = P,\ 0 \leqq i < |T| - |P|\ \}$$

- Frequency of P in T: $|Occ(T, P)|$



$Occ(T,$ aba$) = \{0, 3, 5, 8, 10, 13, 15\}$

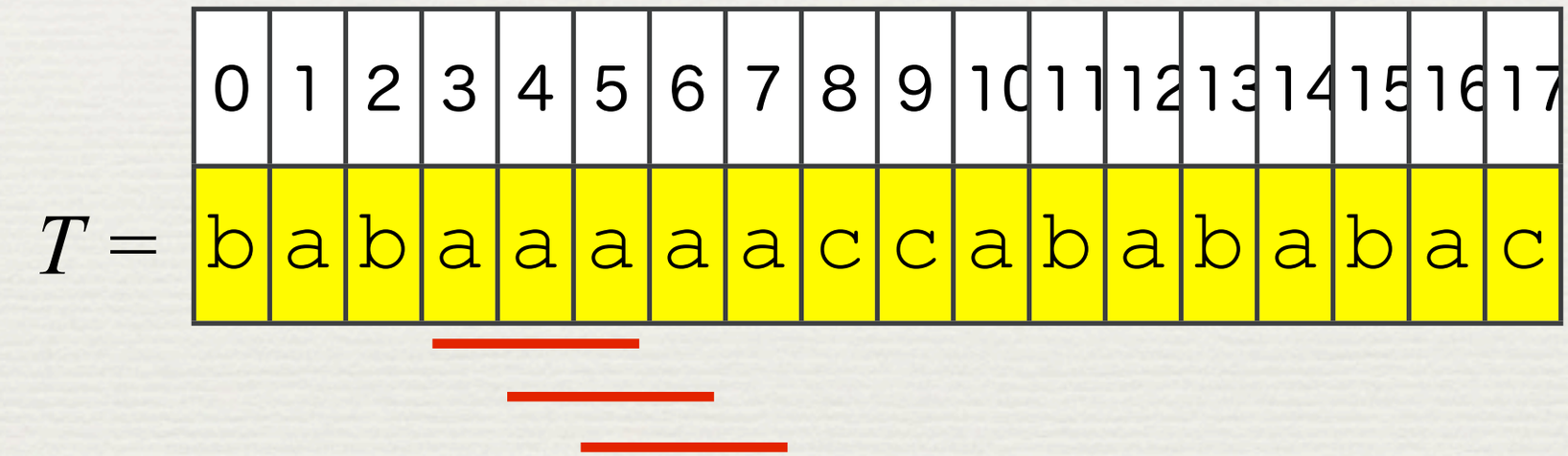
Frequency of “aba” = *7*

Non-overlapping Occurrence

Any two occurrences k_1, $k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be overlapping if $k_2 < k_1 + |P|$, and non-overlapping otherwise.

Non-overlapping Occurrence

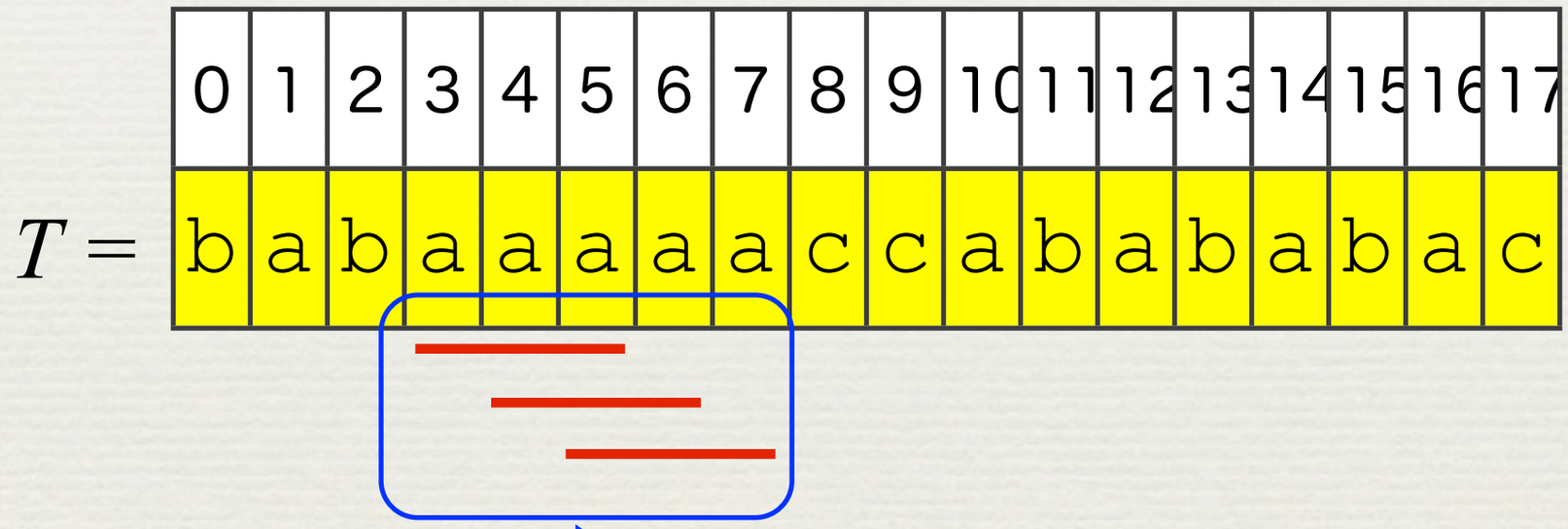
Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be overlapping if $k_2 < k_1 + |P|$, and non-overlapping otherwise.



$Occ(T, \text{aaa}) = \{3,4,5\}$

Non-overlapping Occurrence

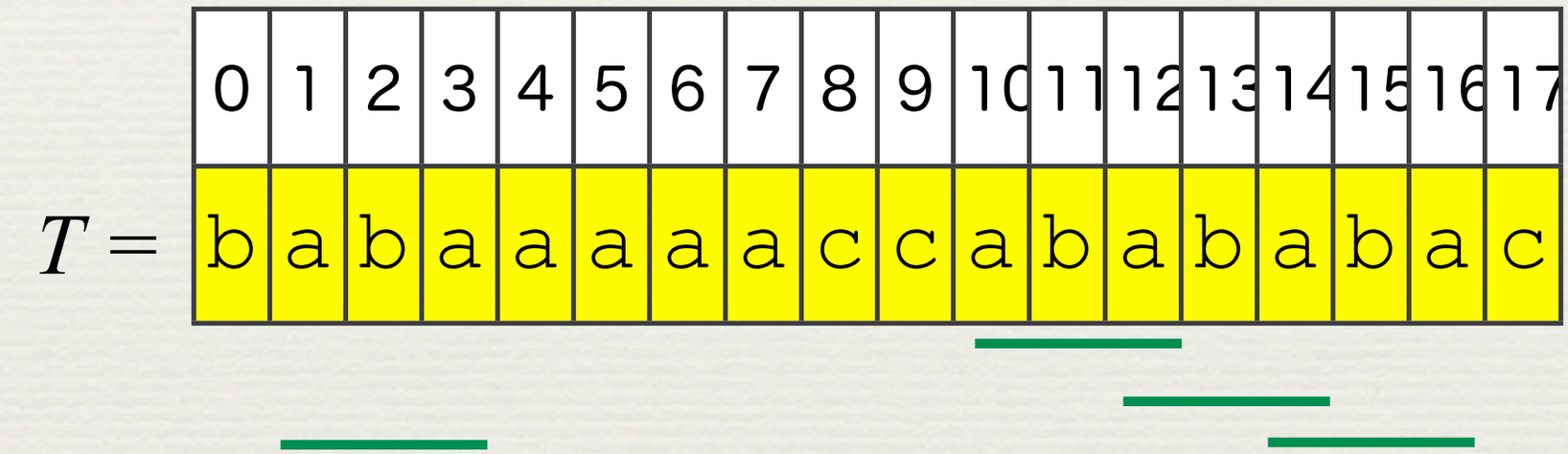
Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be overlapping if $k_2 < k_1 + |P|$, and non-overlapping otherwise.



$Occ(T, \text{aaa}) = \{3,4,5\}$

Non-overlapping Occurrence

Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be overlapping if $k_2 < k_1 + |P|$, and non-overlapping otherwise.



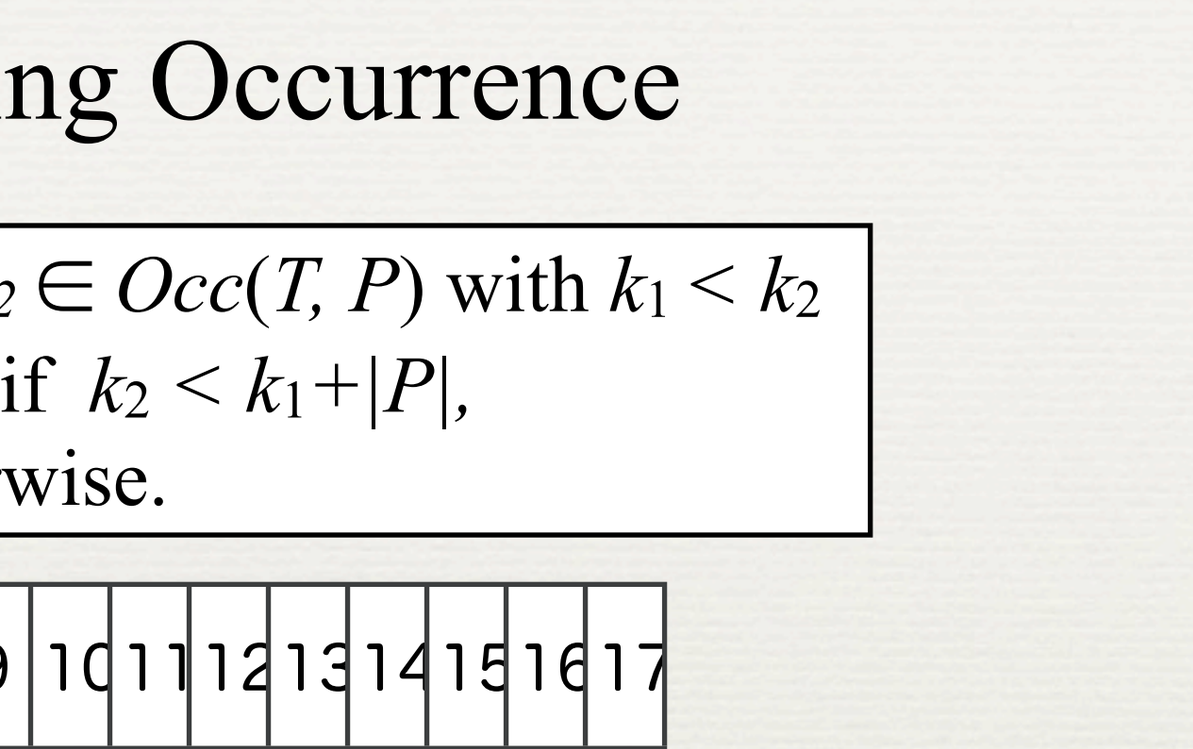
$Occ(T, \text{aba}) = \{1, 10, 12, 14\}$

Non-overlapping Occurrence

Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be overlapping if $k_2 < k_1 + |P|$, and non-overlapping otherwise.

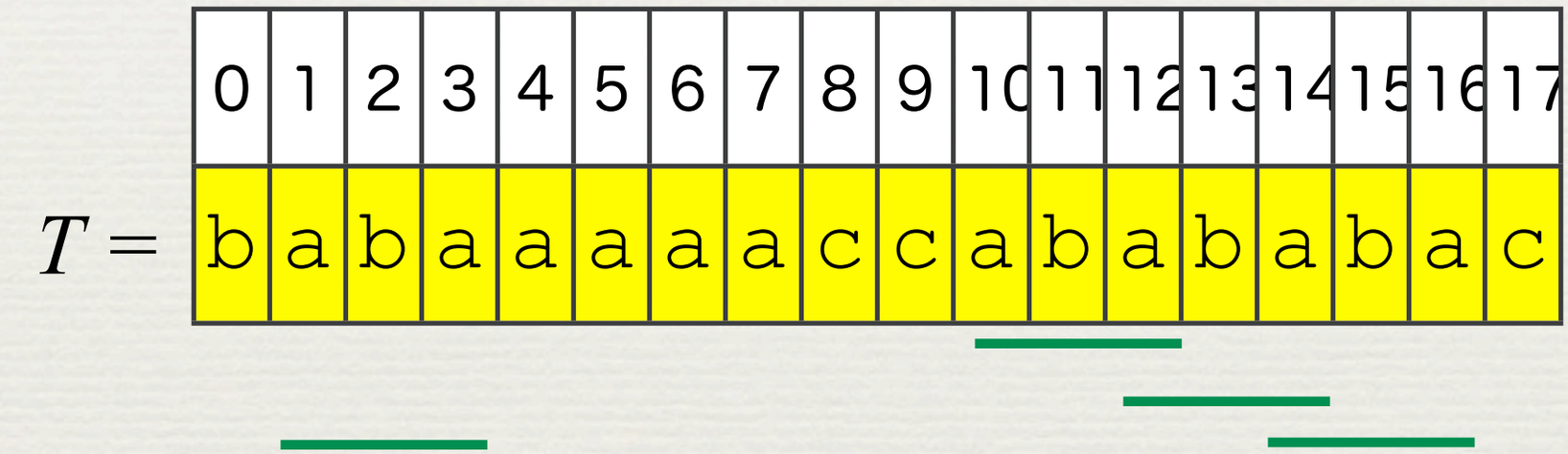
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$T =$	b	a	b	a	a	a	a	a	c	c	a	b	a	b	a	b	a	c

$Occ(T, \text{aba}) = \{1, 10, 12, 14\}$



Non-overlapping Occurrence

Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be overlapping if $k_2 < k_1 + |P|$, and non-overlapping otherwise.

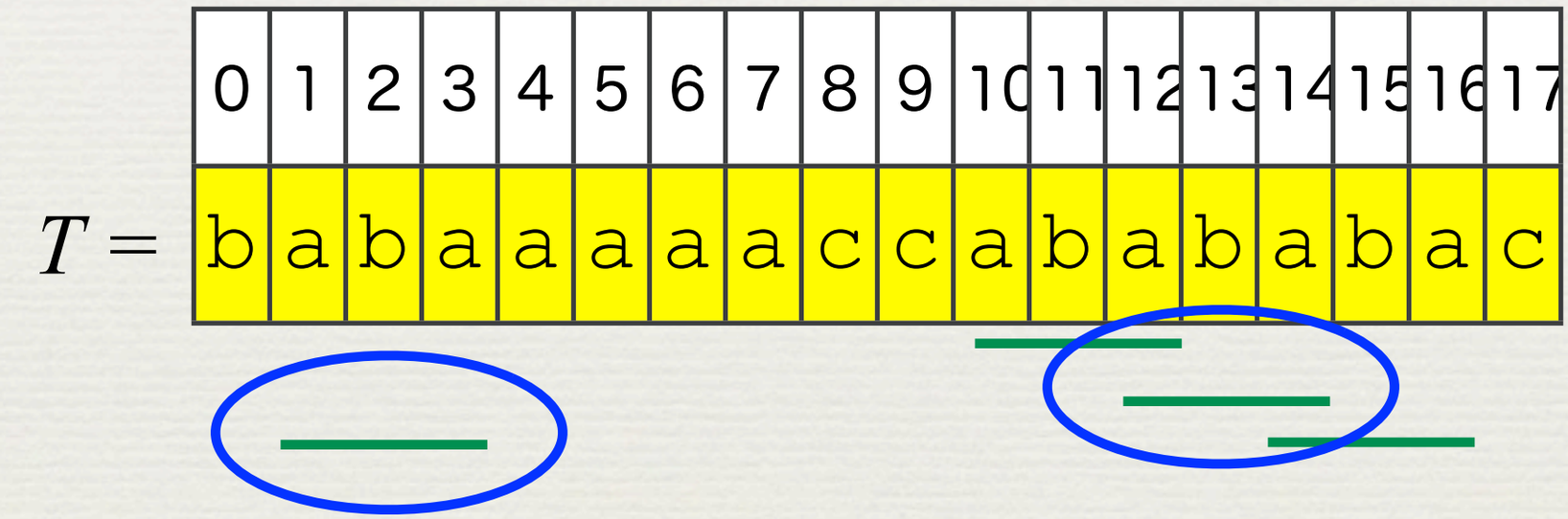


non-overlapping frequency

$nOcc(T, P)$: the size of largest subset of $Occ(T, P)$ where any two occrences in the set are non-overlapping

Non-overlapping Occurrence

Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be overlapping if $k_2 < k_1 + |P|$, and non-overlapping otherwise.

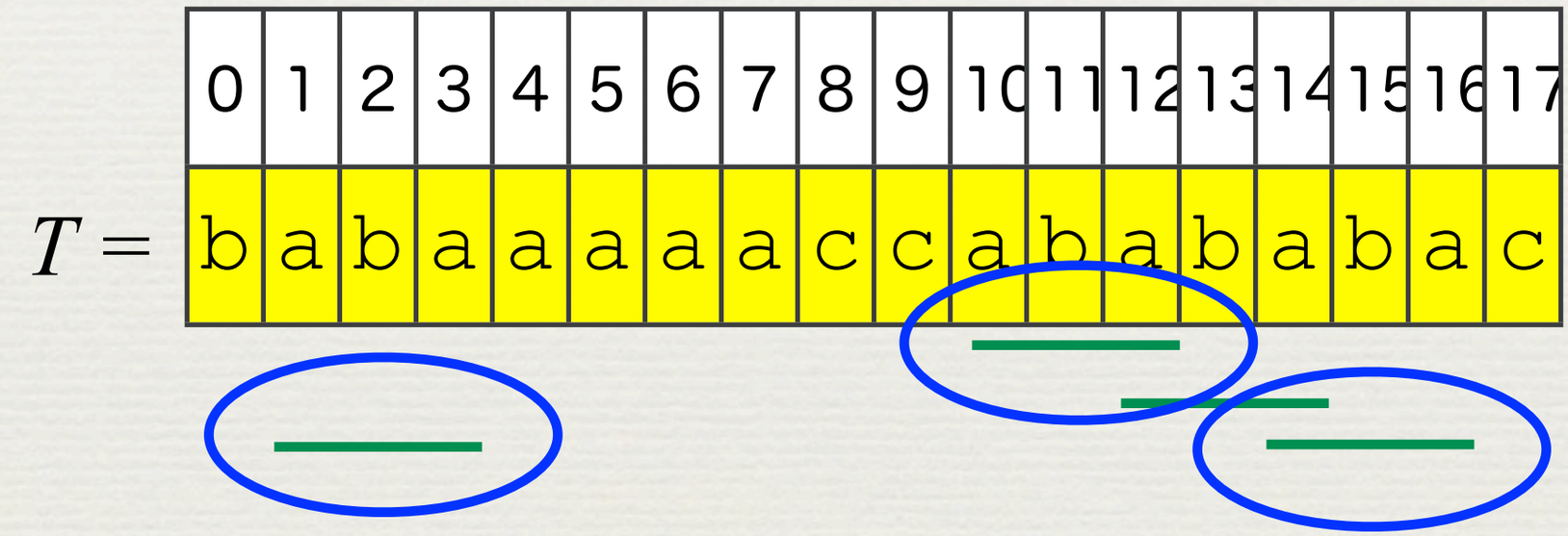


non-overlapping frequency

nOcc(*T*, *P*): the size of largest subset of *Occ*(*T*, *P*) where any two occrences in the set are non-overlapping

Non-overlapping Occurrence

Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be overlapping if $k_2 < k_1 + |P|$, and non-overlapping otherwise.



non-overlapping frequency

nOcc(*T*, *P*): the size of largest subset of *Occ*(*T*, *P*) where any two occrences in the set are non-overlapping

nOcc(*T*, aba) = 3

q-gram Non-overlapping Frequencies Problem

q-gram Non-overlapping Frequencies Problem

Input：string T,　positive integer q

Output：$nOcc(T, x)$ for $\forall\ x \in \Sigma^q$　s.t. $nOcc(T, x) > 0$

q-gram Non-overlapping Frequencies Problem

q-gram Non-overlapping Frequencies Problem

Input：string T,　positive integer q

Output：$nOcc(T, x)$ for $\forall\ x \in \Sigma^q$　s.t. $nOcc(T, x) > 0$

Example　$q = 3$

$T =$ `abaababaab`

3-gram	*nOcc*
`aab`	
`aba`	
`baa`	
`bab`	

q-gram Non-overlapping Frequencies Problem

q-gram Non-overlapping Frequencies Problem

Input：string T,　positive integer q

Output：$nOcc(T, x)$ for $\forall\ x \in \Sigma^q$　s.t. $nOcc(T, x) > 0$

Example　$q = 3$

$T =$ abaababaab

3-gram	*nOcc*
aab	2
aba	
baa	
bab	

q-gram Non-overlapping Frequencies Problem

q-gram Non-overlapping Frequencies Problem

Input：string T,　positive integer q

Output：$nOcc(T, x)$ for $\forall\ x \in \Sigma^q$　s.t. $nOcc(T, x) > 0$

Example　$q = 3$

$T =$ abaababaab

3-gram	*nOcc*
aab	2
aba	2
baa	
bab	

q-gram Non-overlapping Frequencies Problem

q-gram Non-overlapping Frequencies Problem

Input：string T,　positive integer q

Output：$nOcc(T, x)$ for $\forall\ x \in \Sigma^q$　s.t. $nOcc(T, x) > 0$

Example　$q = 3$

$T =$ `abaababaab`

3-gram	$nOcc$
`aab`	2
`aba`	2
`baa`	2
`bab`	1

We want to solve this problem when string T is given in compressed form.

Straight Line Program (SLP)

Definition (Straight Line Program)

Straight Line Program is a context free grammar in the Chomsky normal form that derives a single string

derivation rules :

$X_1 = \mathrm{expr}_1, X_2 = \mathrm{expr}_2, ..., X_n = \mathrm{expr}_n$

$\mathrm{expr}_i \in \Sigma$ or

$\mathrm{expr}_i = X_l \cdot X_r \ (l, r < i)$

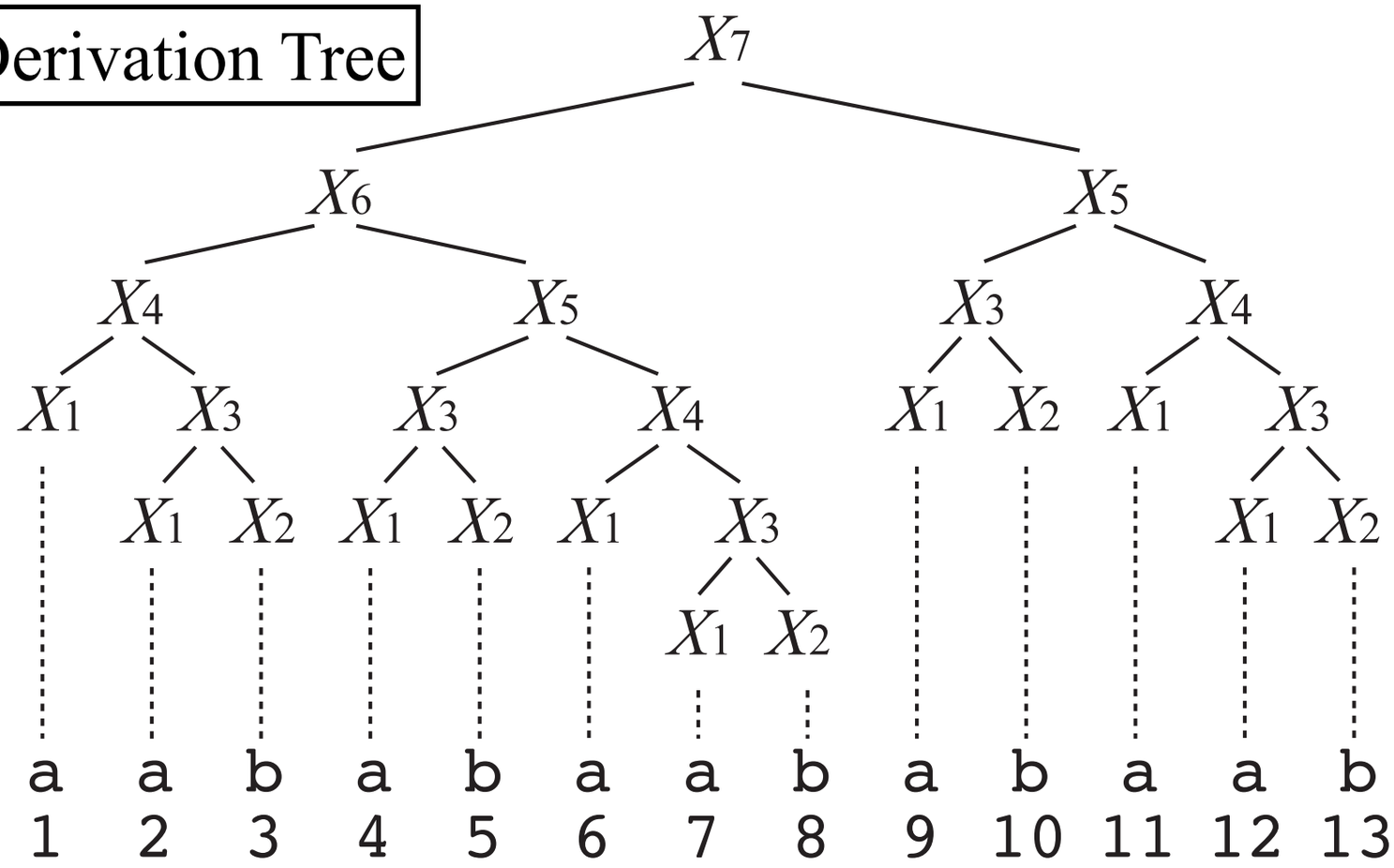
- SLP can represent the output of well-known compression schemes.
  - e.g. RE-PAIR SEQUITUR, LZ78, LZW
- LZ77, LZSS of size m can be quickly transformed to SLP of size $O(m \log |T|)$. **[Rytter, 2003]**

Example SLP

$$\mathcal{D} = \begin{aligned} X_1 &= \text{a} \\ X_2 &= \text{b} \\ X_3 &= X_1 X_2 \\ X_4 &= X_1 X_3 \\ X_5 &= X_3 X_4 \\ X_6 &= X_4 X_5 \\ X_7 &= X_6 X_5 \end{aligned}$$

$n = |\mathcal{D}| = 7$

Derivation Tree



$T =$

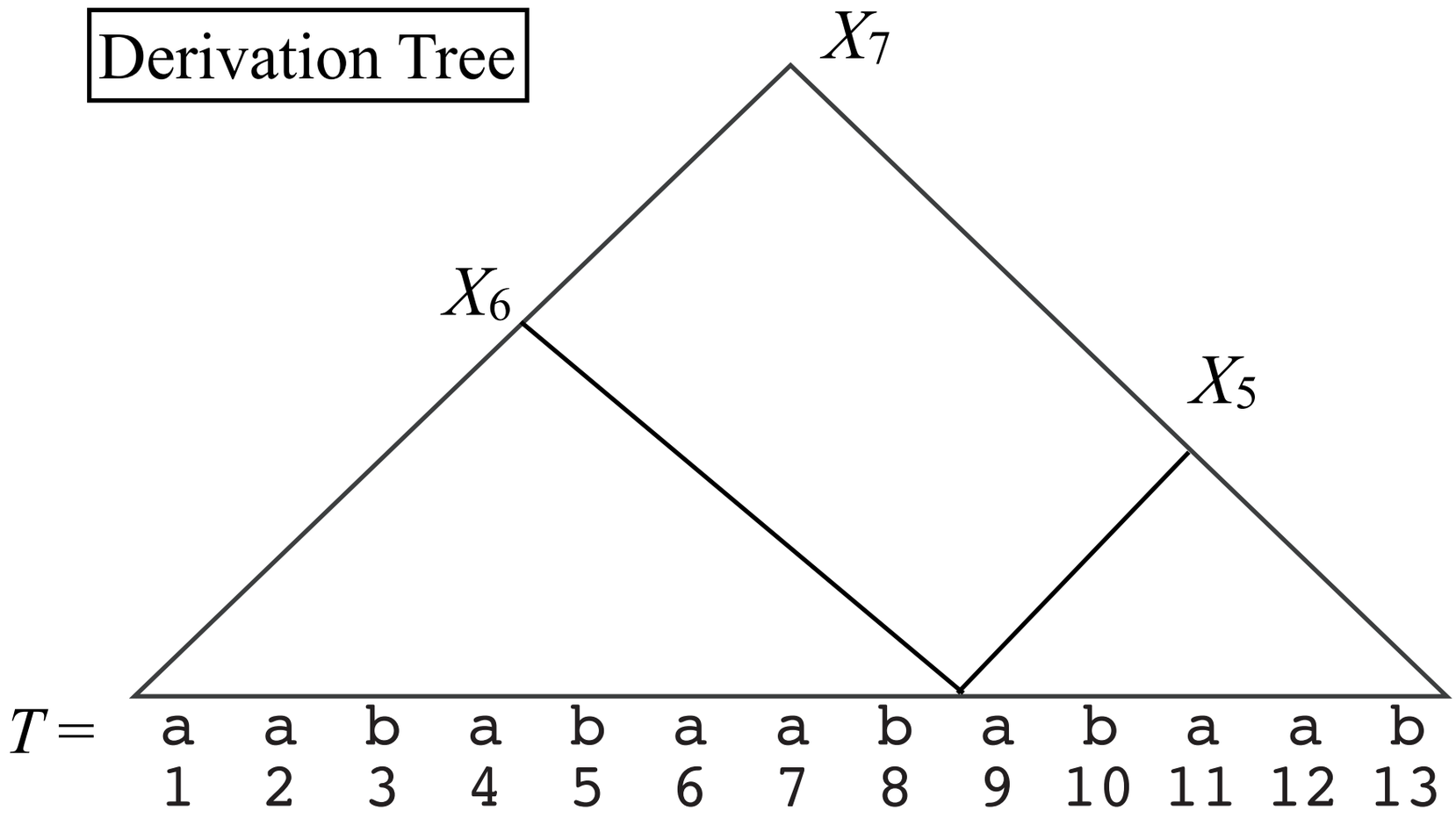
Length of decompressed string can be $\Theta(2^n)$

Example SLP

$$\mathcal{D} = \begin{aligned} X_1 &= \text{a} \\ X_2 &= \text{b} \\ X_3 &= X_1 X_2 \\ X_4 &= X_1 X_3 \\ X_5 &= X_3 X_4 \\ X_6 &= X_4 X_5 \\ X_7 &= X_6 X_5 \end{aligned}$$

$n = |\mathcal{D}| = 7$

Derivation Tree



Length of decompressed string can be $\Theta(2^n)$

Previous Work

	Uncompressed String	SLP (Previous Work)
q-gram Freq	$O(\|T\|)$ time and space	$O(qn)$ [2] time and space
2-gram non-overlapping Freq	$O(\|T\|)$ time and space	$O(n^4 \log n)$ [1] time $O(n^3)$ space
q-gram non-overlapping Freq	$O(\|T\|)$ time and space	-

[1] [Inenaga and Bannai, 2009]

[2] [Goto et al, 2011]

Previous Work

	Uncompressed String	SLP (Previous Work)	SLP (This Work)
q-gram Freq	$O(\|T\|)$ time and space	$O(qn)$ [2] time and space	-
2-gram non-overlapping Freq	$O(\|T\|)$ time and space	$O(n^4 \log n)$ [1] time $O(n^3)$ space	$O(n)$ time and space
q-gram non-overlapping Freq	$O(\|T\|)$ time and space	-	$O(q^2 n)$ time $O(qn)$ space

[1] [Inenaga and Bannai, 2009]

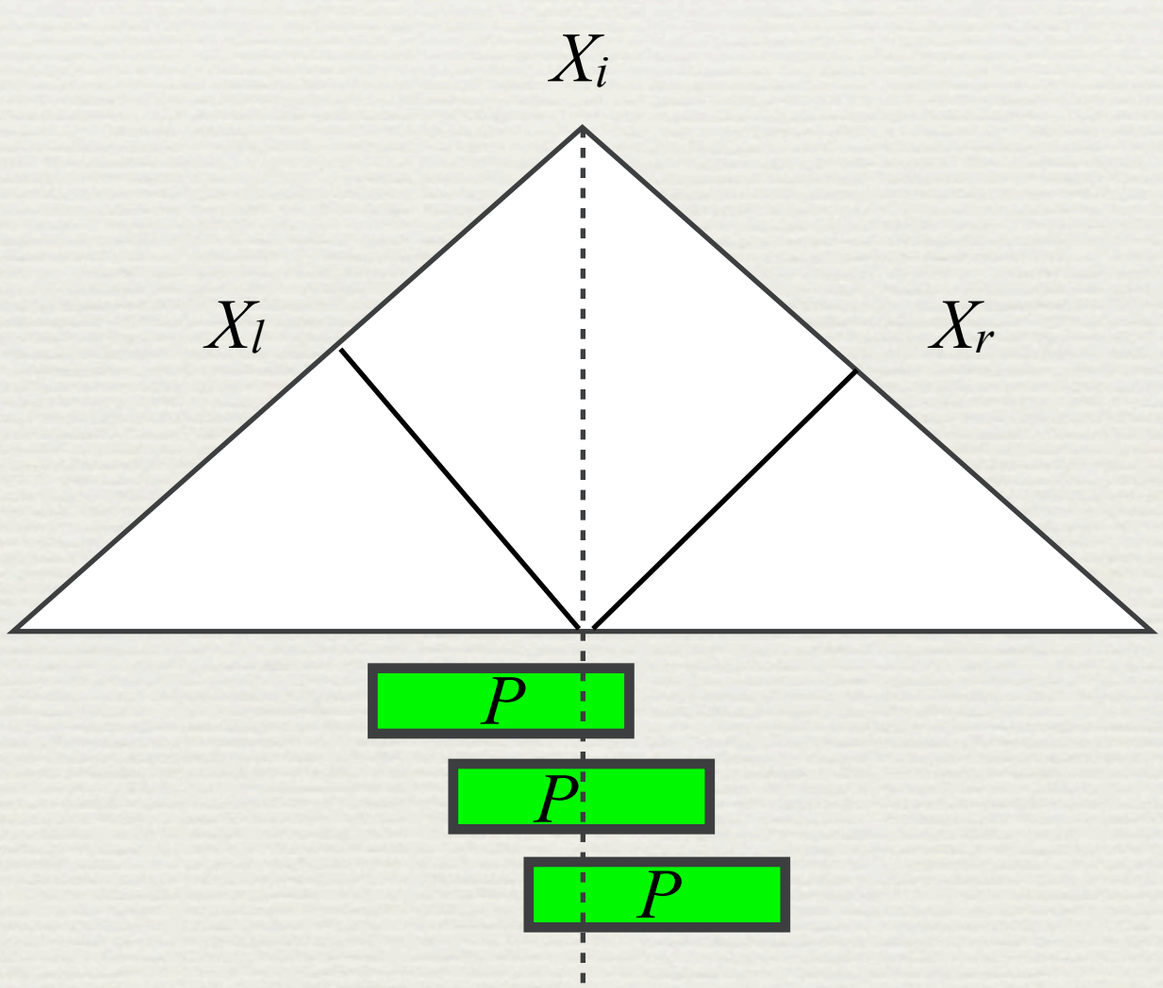
[2] [Goto et al, 2011]

$O(qn)$ algorithm for q-gram Frequencies Problem on SLP

[Goto et al., 2011]

*Occ**: Crossing Occurrences

- For $X_i = X_l\, X_r$, an occurrence of P *crosses* X_i, if P starts in X_l and ends in X_r.
- Denote the set of such occurrences by $Occ^*(X_i, P)$.

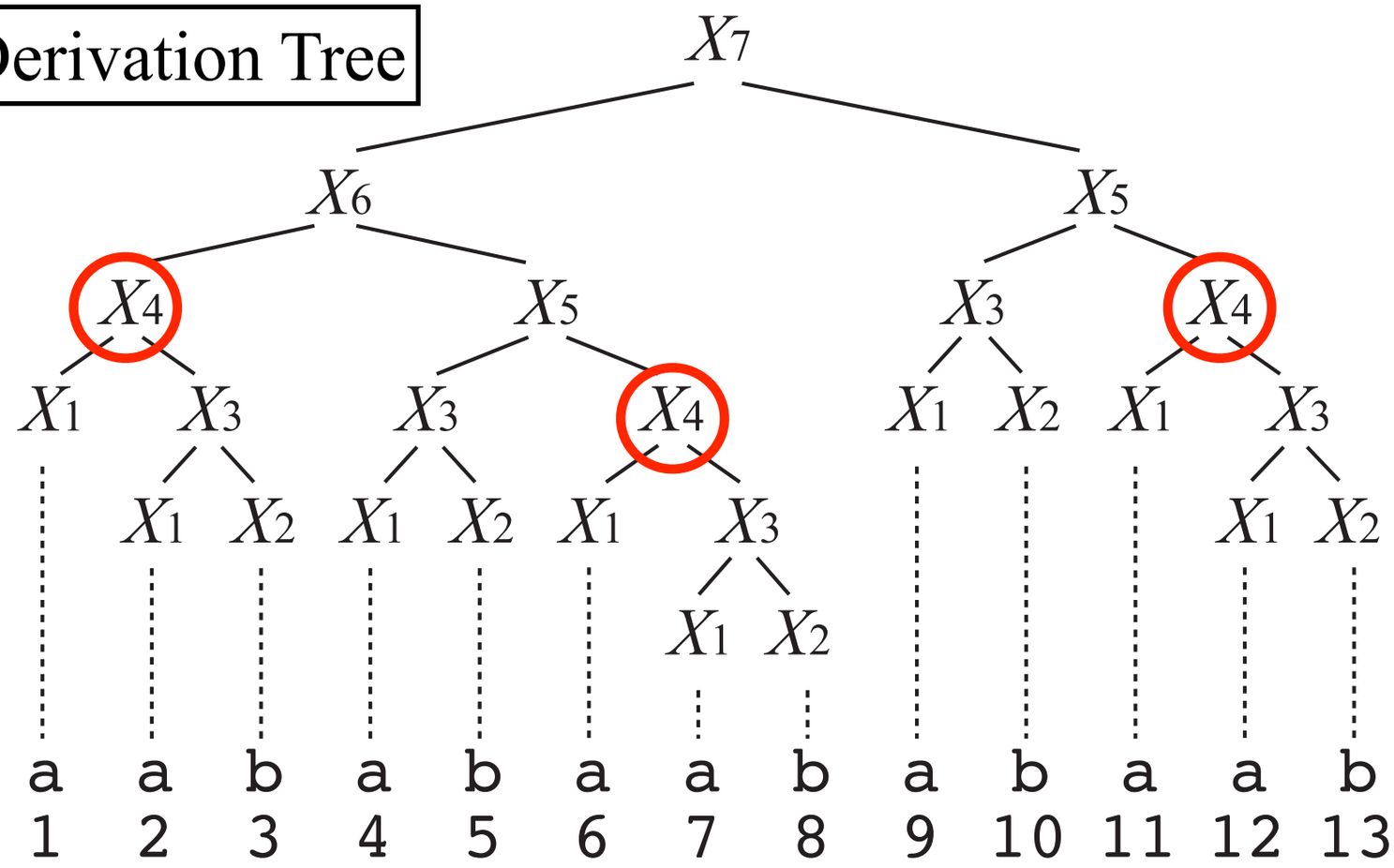


vOcc

$$\mathcal{D} = \begin{aligned} X_1 &= \mathrm{a} \\ X_2 &= \mathrm{b} \\ X_3 &= X_1 X_2 \\ X_4 &= X_1 X_3 \\ X_5 &= X_3 X_4 \\ X_6 &= X_4 X_5 \\ X_7 &= X_6 X_5 \end{aligned}$$

$n = |\mathcal{D}| = 7$

Derivation Tree



$vOcc(X_i)$: the number of occurrence X_i in the derivation tree of X_n

Example $vOcc(X_4) = 3$

Calculating Frequency by $Occ^*(X_i, P)$

Lemma 1

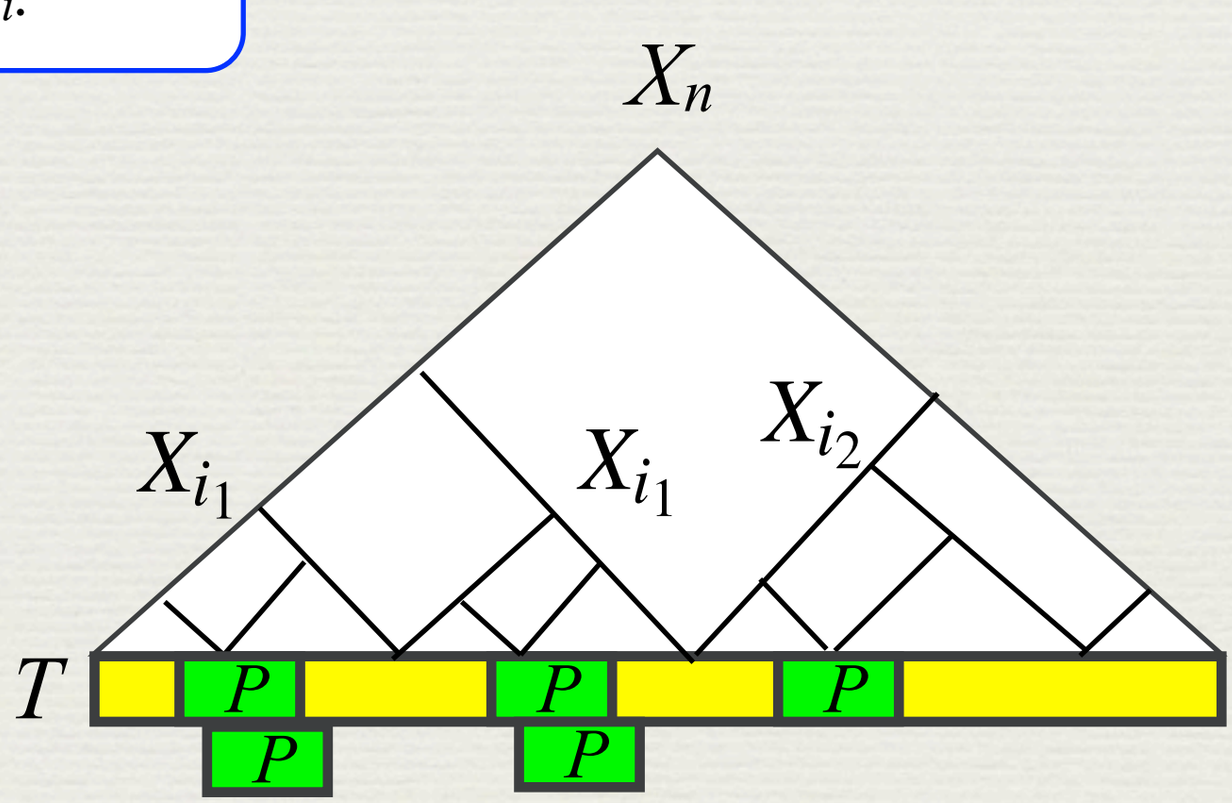
$$|Occ(T,P)| = \sum_{i=1}^{n} |Occ^*(X_i,P)| \cdot vOcc(X_i)$$

Calculating Frequency by $Occ^*(X_i, P)$

Lemma 1

$$|Occ(T,P)| = \sum_{i=1}^{n} |Occ^*(X_i,P)| \cdot vOcc(X_i)$$

For any occurrence of substring P in T,
there uniquely exists X_i such that P crosses X_i.



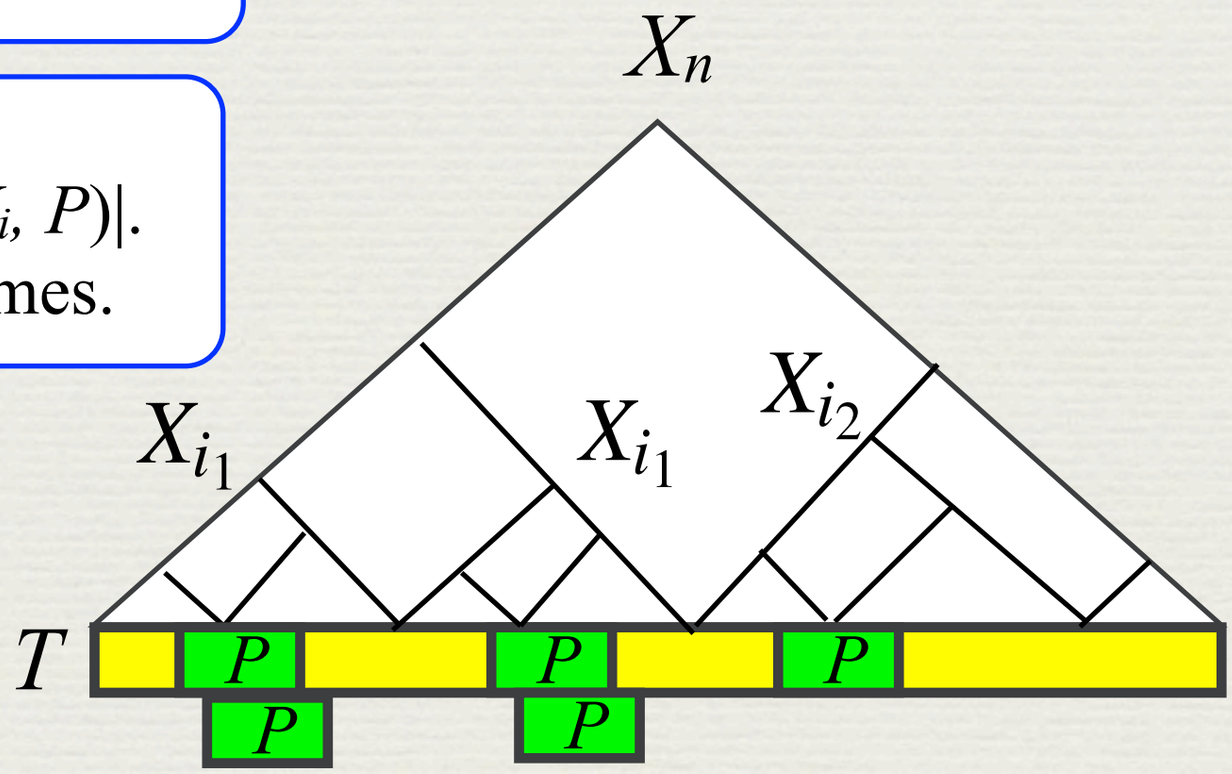
Calculating Frequency by $Occ^*(X_i, P)$

Lemma 1

$$|Occ(T,P)| = \sum_{i=1}^{n} |Occ^*(X_i,P)| \cdot vOcc(X_i)$$

For any occurrence of substring P in T,
there uniquely exists X_i such that P crosses X_i.

For each variable X_i,
- the number of times P crosses X_i is $|Occ^*(X_i, P)|$.
- X_i appears in the derivation tree $vOcc(X_i)$ times.



Calculating Frequency by $Occ^*(X_i, P)$

Lemma 1

$$|Occ(T,P)| = \sum_{i=1}^{n} |Occ^*(X_i,P)| \cdot vOcc(X_i)$$

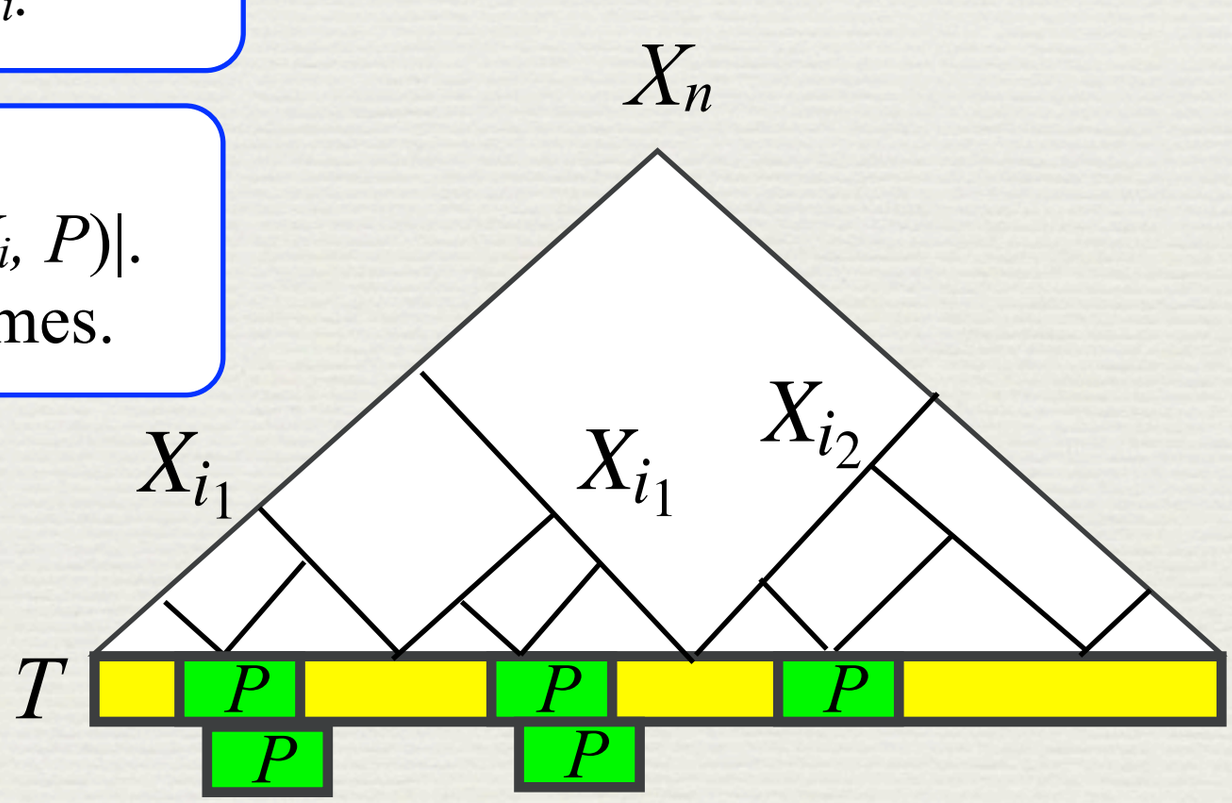
For any occurrence of substring P in T,
there uniquely exists X_i such that P crosses X_i.

For each variable X_i,
- the number of times P crosses X_i is $|Occ^*(X_i, P)|$.
- X_i appears in the derivation tree $vOcc(X_i)$ times.

$vOcc(X_{i_1}) = 2$, $vOcc(X_{i_2}) = 1$

$|Occ^*(X_{i_1}, P)| = 2$, $|Occ^*(X_{i_2}, P)| = 1$

Frequency of $P = 2 \cdot 2 + 1 = 5$



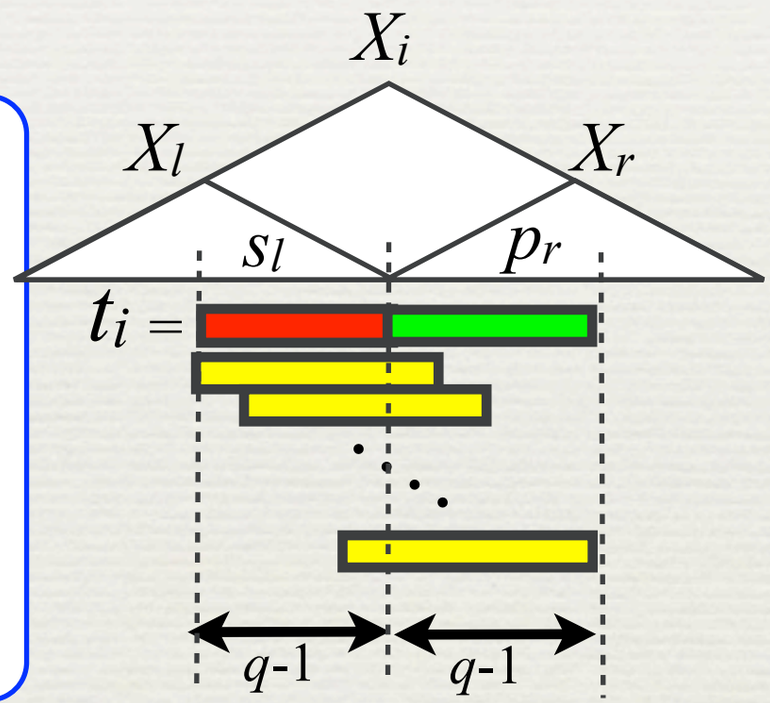
Computing the Number of Crossing Occurrences

For all $X_i = X_l\, X_r$, let $t_i = s_l\, p_r$ where s_l is the length-(q-1) suffix of X_l
p_r is the length-(q-1) prefix of X_r

the number of occurrences of q-gram in t_i
equals to that of crossing occurrences in X_i.

$$|Occ^*(X_i, t_i[\,j\,..\,j+q\text{-}1])| = |Occ(t_i, t_i[\,j\,..\,j+q\text{-}1])|$$

where $1 \leqq j \leqq |t_i|\text{-}q+1$



Lemma 2

$$\text{for } P \in \Sigma^q, \quad |Occ(T,P)| = \sum_{i=1}^{n} |Occ(t_i,P)| \cdot vOcc(X_i)$$

Algorithm for SLP q-gram Frequencies Problem

Theorem 1

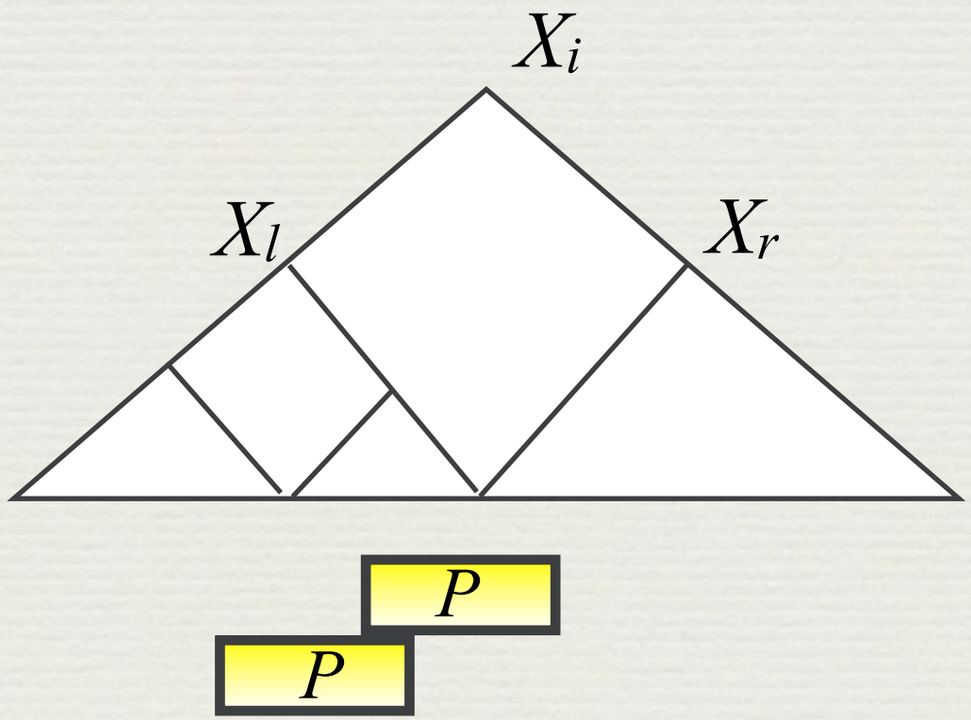
SLP q-gram Frequencies Problem
can be solved in $O(qn)$ time and space.

Idea:
Using the suffix array of the concatenation of all t_i's,
plus a linear-time construction algorithm of suffix arrays,
we can solve the SLP q-gram frequency problem in $O(qn)$ time.

Algorithm for q-gram Non-overlapping Frequencies Problem

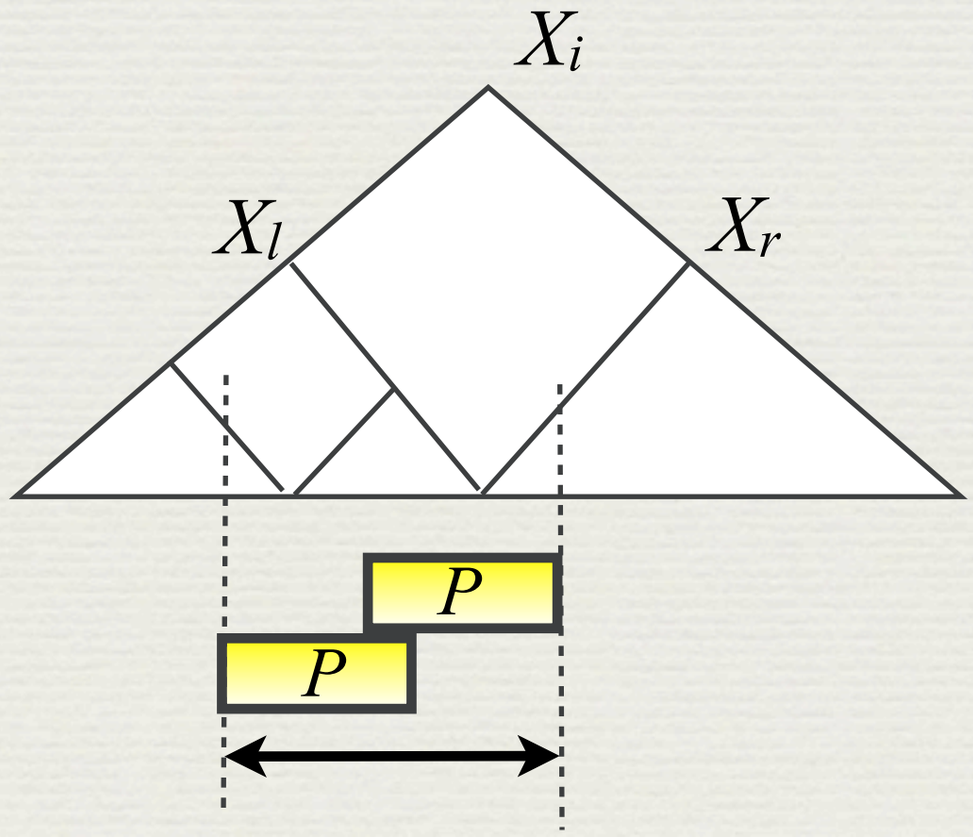
How to deal with crossing occurrences

- A crossing occurrence of P in X_i may overlap a crossing occurrence of P in another variable X_l.

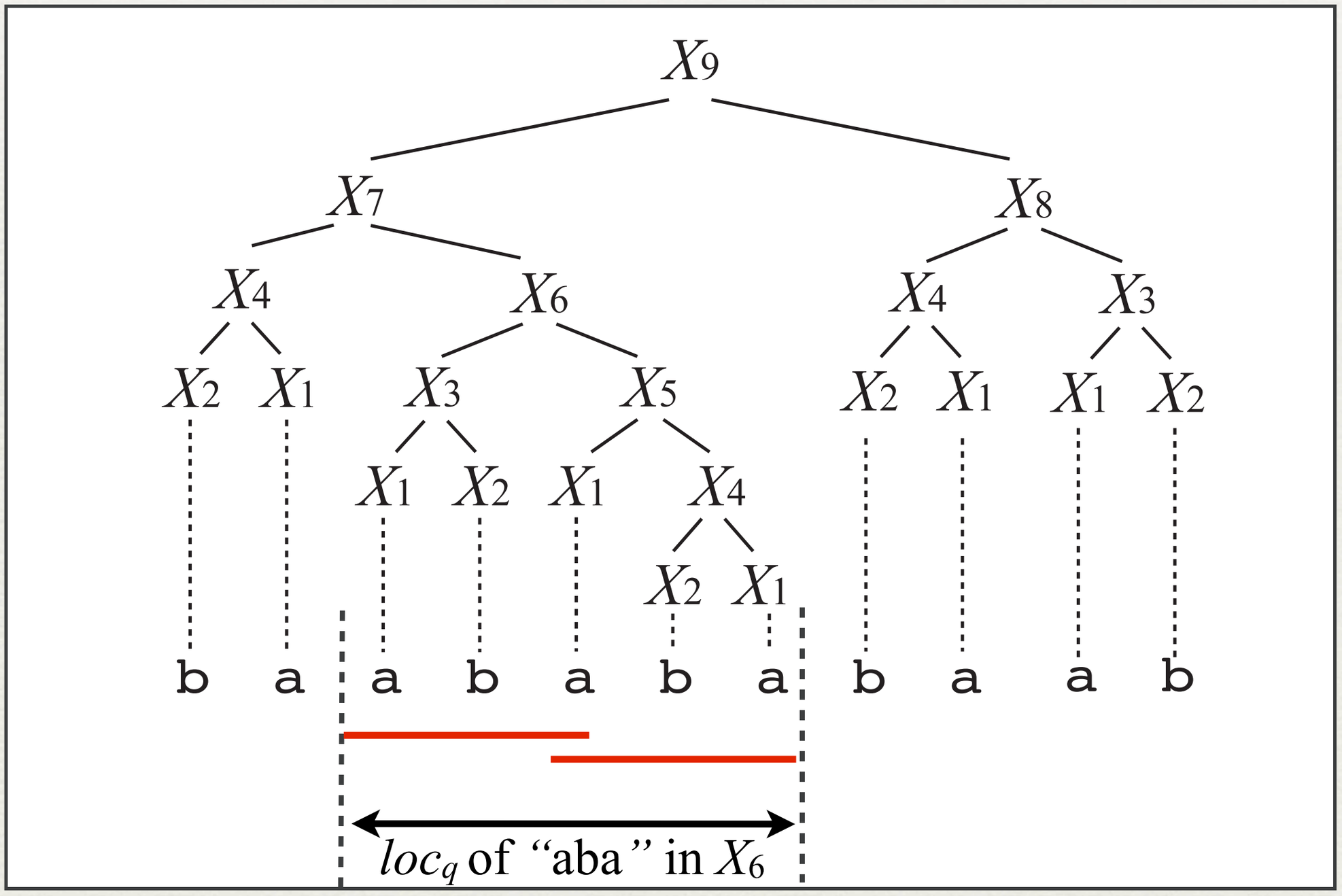


How to deal with crossing occurrences

- A crossing occurrence of P in X_i may overlap a crossing occurrence of P in another variable X_l.
- our strategy is that for string P which crosses X_i,
  - we compute the interval of maximum chain of overlapping occurrences of P.
  - we count $nOcc$ in its interval only if the interval cannot be extended either to the left or right.

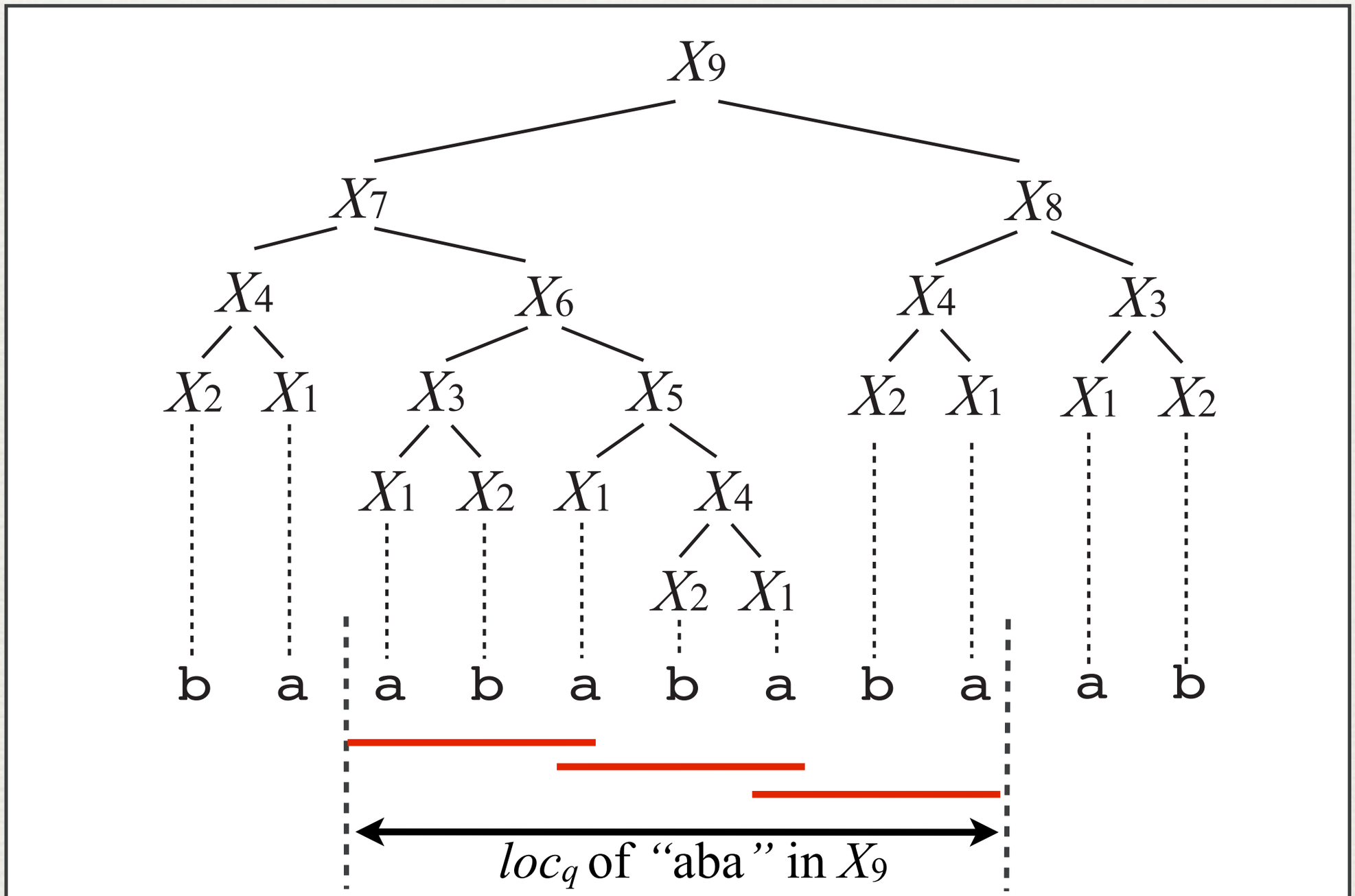


Longest Overlapping Cover on SLP

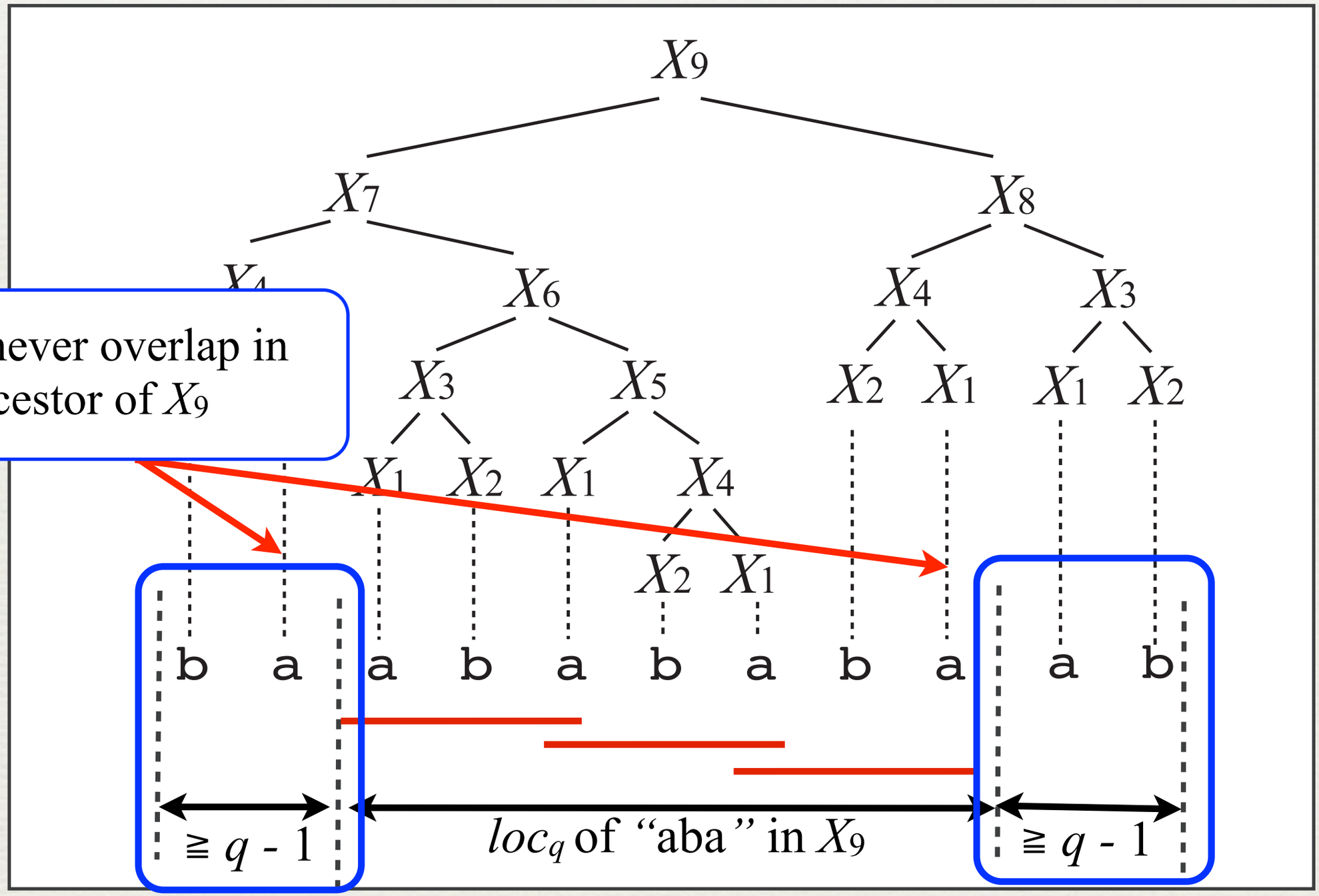


Longest Overlapping Cover on SLP

- Notice that loc_q in the variable X_i may or may not extend in X_i's ancestor.



Longest Overlapping Cover on SLP



Longest Overlapping Cover on SLP

- Let (b, e) is a pair of beginning and ending point of loc_q in the variable X_i, we obtain this lemma

Lemma 4

$$nOcc(T,P) = \sum_{i=1}^{n} nOcc(X_i[b:e],P) \cdot vOcc(X_i)$$

where (b, e) is loc_q of P in T, and $[b\text{-}q\text{+}1 : e\text{+}q\text{-}1]$ crosses X_i $(0 < b\text{-}q\text{+}1 \leqq |X_l| \leqq e\text{+}q\text{-}1 \leqq |X_i|)$

Lemma 5

For all variables $X_i = X_l X_r$, and integer $|X_l| - (2q\text{-}1) \leqq j \leqq |X_l|+q\text{-}1$, $loc_q(X_i, j) = (b, e)$ and $nOcc(X_i[b{:}e], X_i[\, j : j+q\text{-}1]$ can be computed in $O(q^2 n)$ time and $O(qn)$ space.

Theorem 2

Non-overlapping frequencies problem on SLP can be solved in $O(q^2 n)$ time and $O(qn)$ space.

Summary

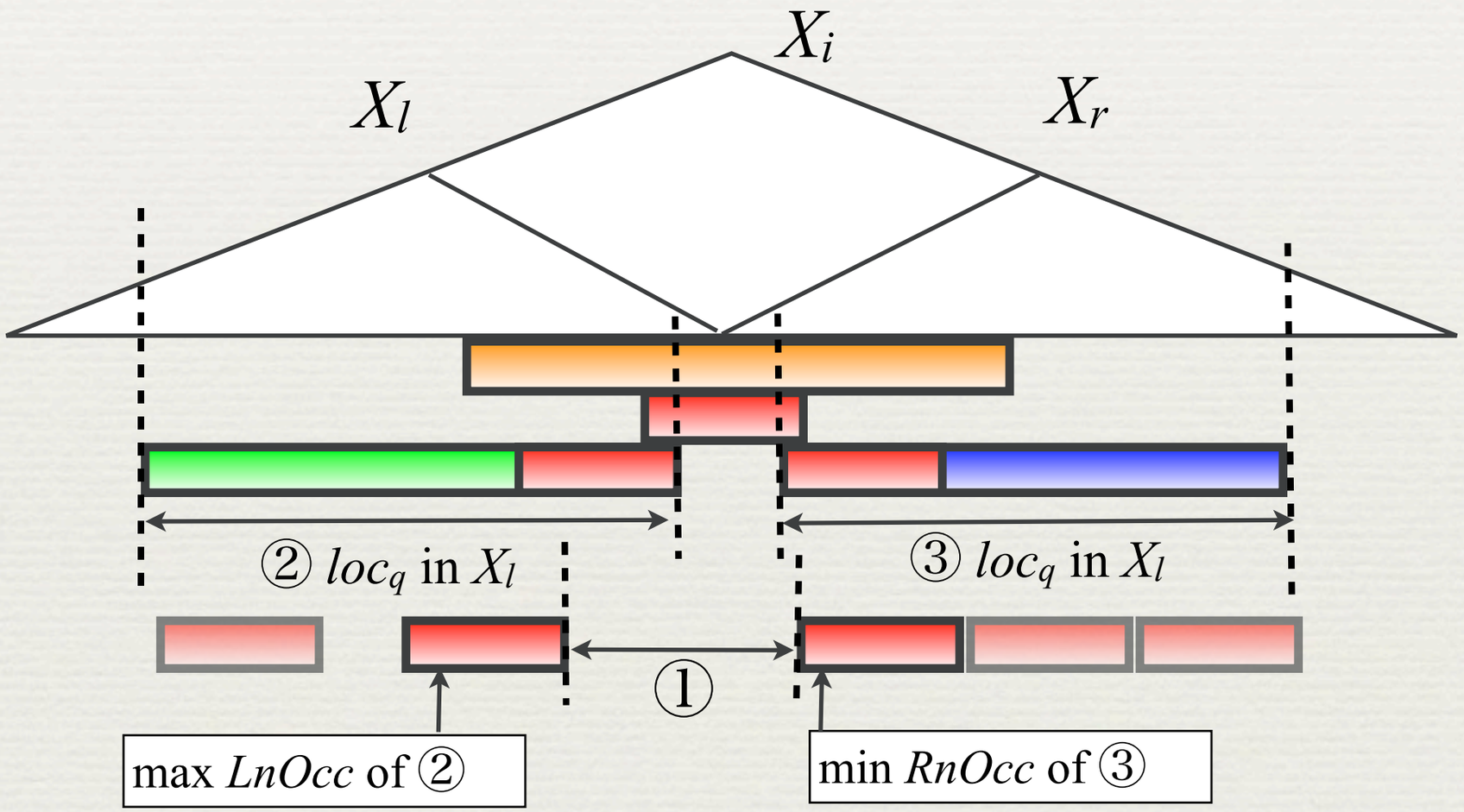
	Uncompressed String	SLP (Previous Work)	SLP (This Work)
q-gram Freq	$O(\|T\|)$ time and space	$O(qn)$ [2] time and space	-
2-gram non-overlapping Freq	$O(\|T\|)$ time and space	$O(n^4 \log n)$ [1] time $O(n^3)$ space	$O(n)$ time and space
q-gram non-overlapping Freq	$O(\|T\|)$ time and space	-	$O(q^2 n)$ time $O(qn)$ space

[1] [Inenaga and Bannai, 2009]

[2] [Goto et al, 2011]

Thank you for your attention

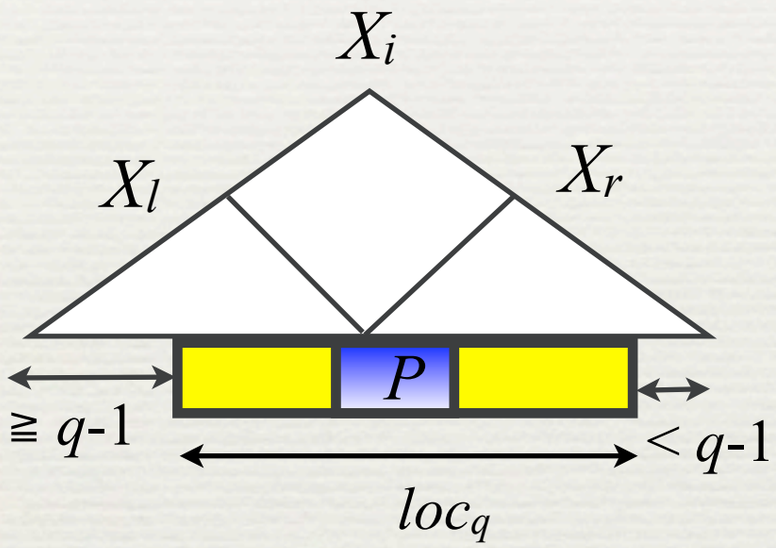
nOcc in loc_q



nOcc within *loc* = *nOcc* within ② + *nOcc* within ③
+ *nOcc* within ①

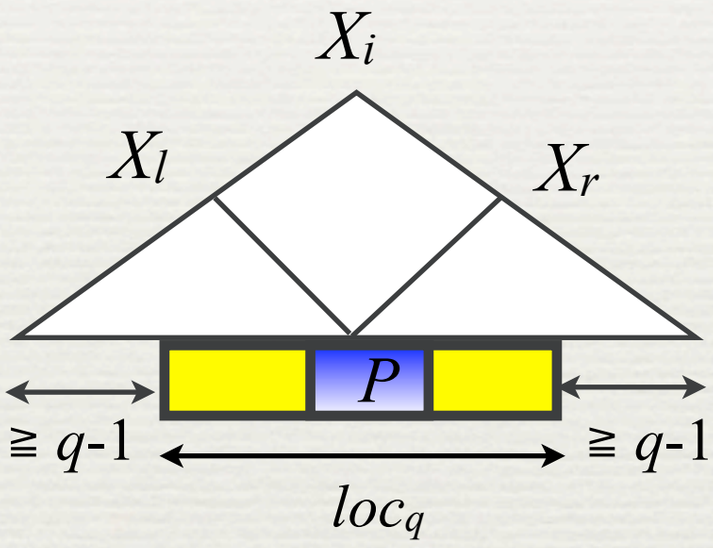
no q-gram頻度の重み付け

- すべての変数で対応する loc_q の範囲が確定した q-gram の重み付けを行う



なにもしない

重み 0

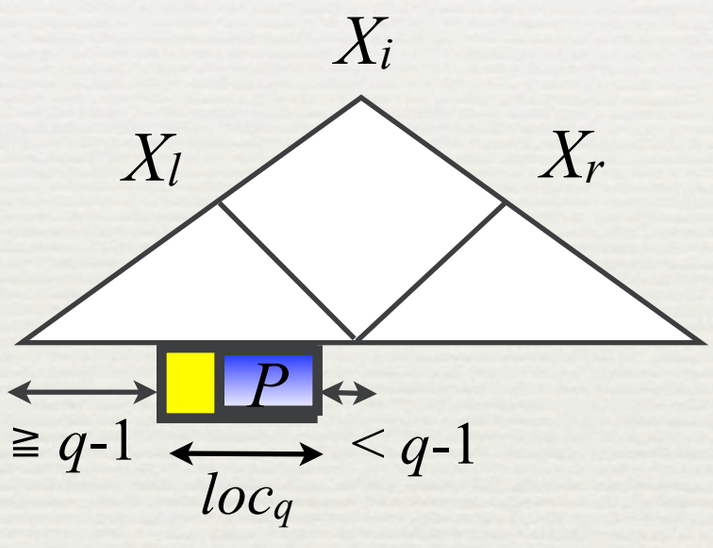


loc_qの範囲が確定

対応するq-gramがまたがる

重み

P : $vOcc(X_i)$ * loc_qのno頻度



loc_qの範囲が確定

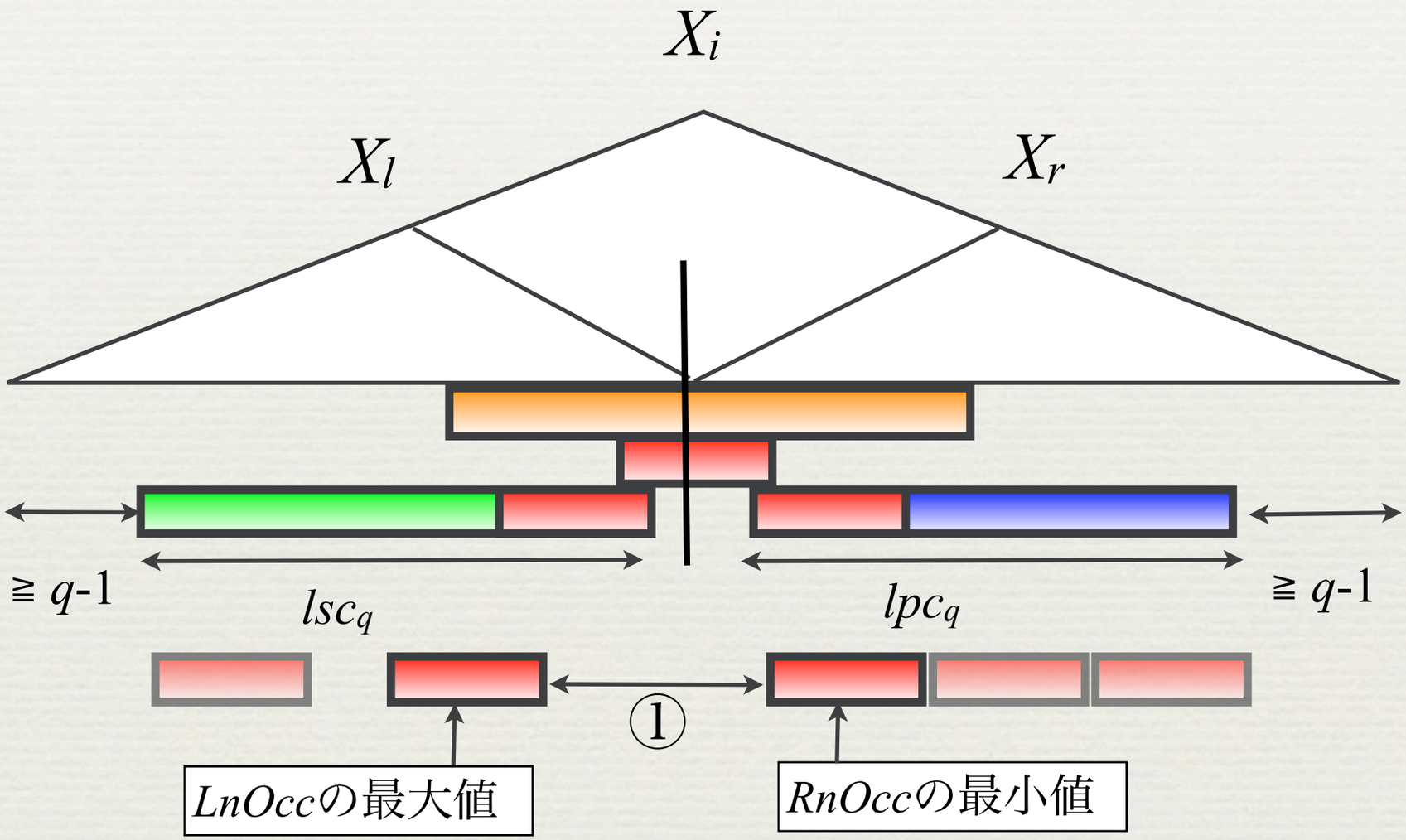
対応するq-gramがまたがらない

重み

P : $vOcc(X_i)$ * loc_qのno頻度

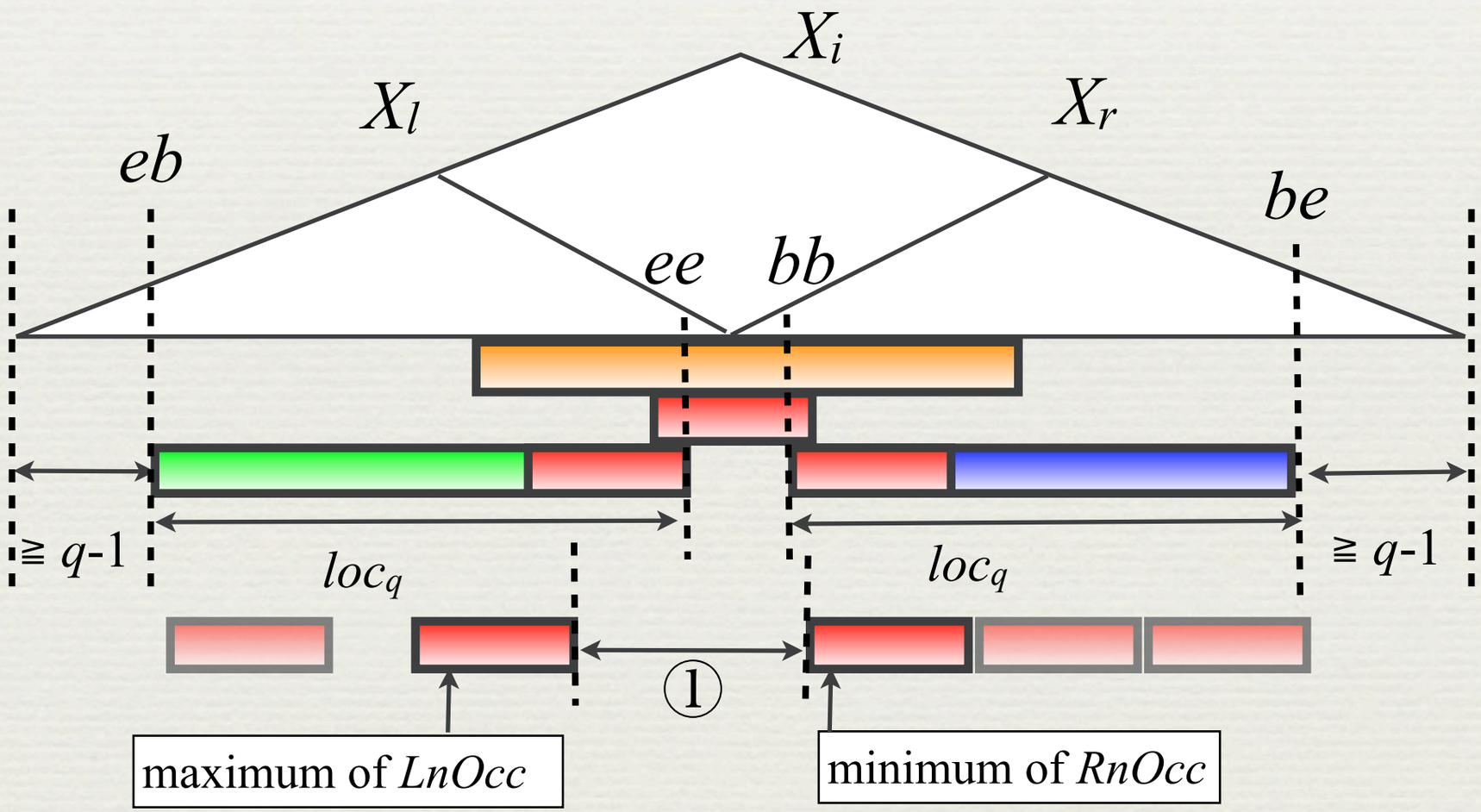
no q-gram頻度の重み付け

各変数に対しまたがるq-gramは
$O(q^2)$でloc_qのno頻度を求めることが可能



locのno頻度 = lscのno頻度 + lpcのno頻度 + ①の範囲のno頻度

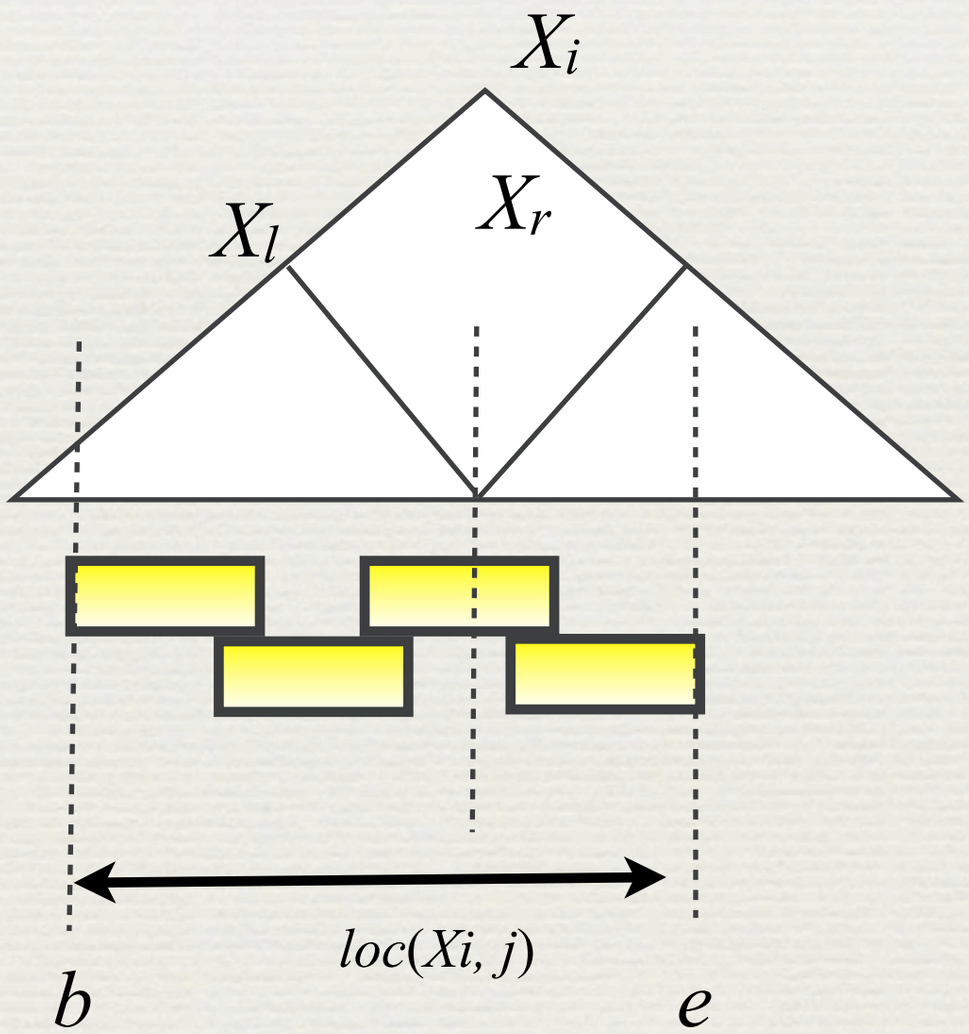
全ての変数についてloc_qのno頻度を求めることで
SLP no頻度問題を $O(q^2n)$ で解くことが出来る



nOcc within *loc*= *nOcc* within *loc* + *nOcc* within *loc*
+ *nOcc* within range of ①

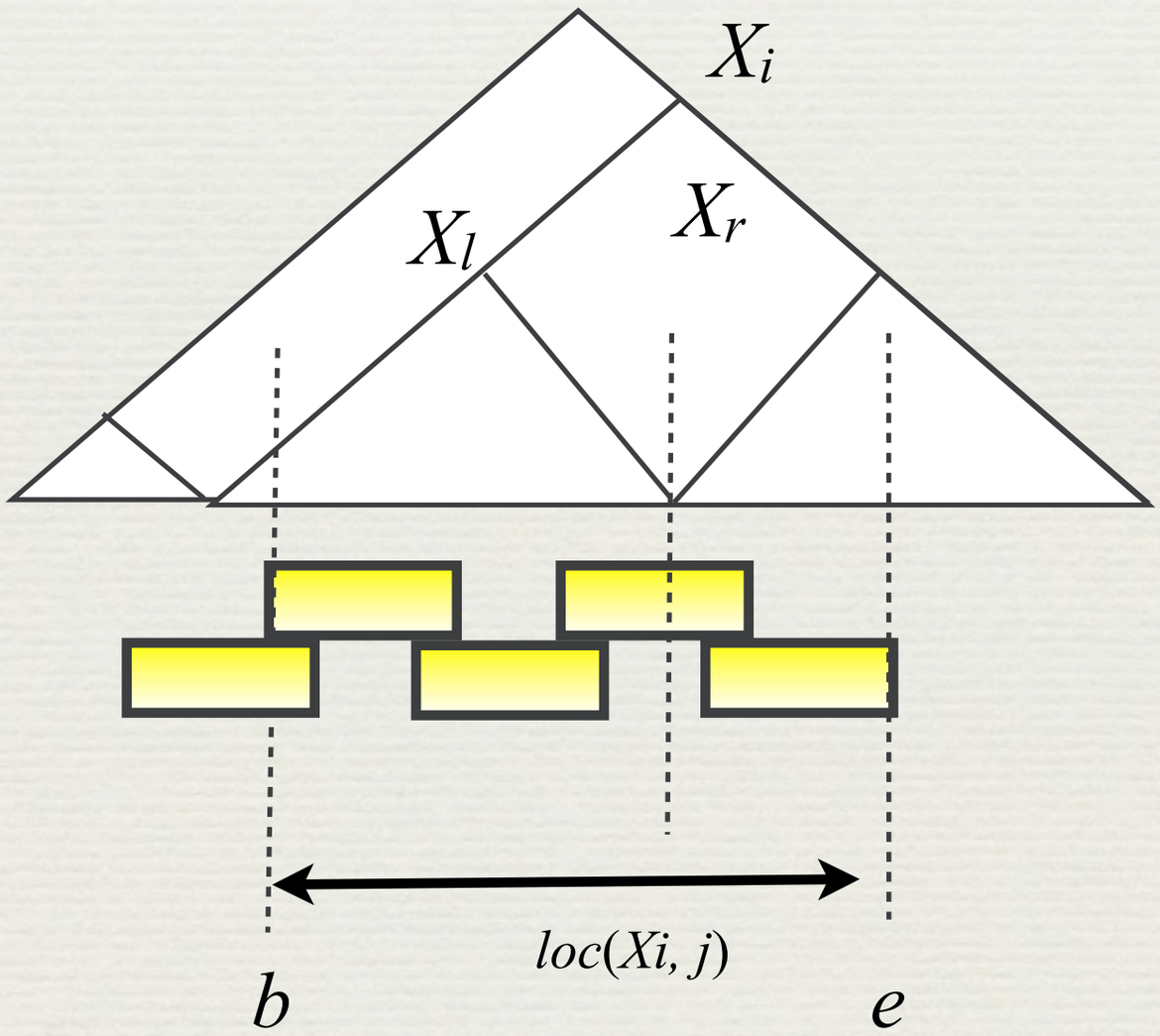
Crossing Occurrence

- For all variables X_i, we count non-overlapping frequencies of loc which crosses X_i rather than q-gram which crosses X_i



Crossing Occurrence

- For all variables X_i, we count non-overlapping frequencies of loc which crosses X_i rather than q-gram which crosses X_i

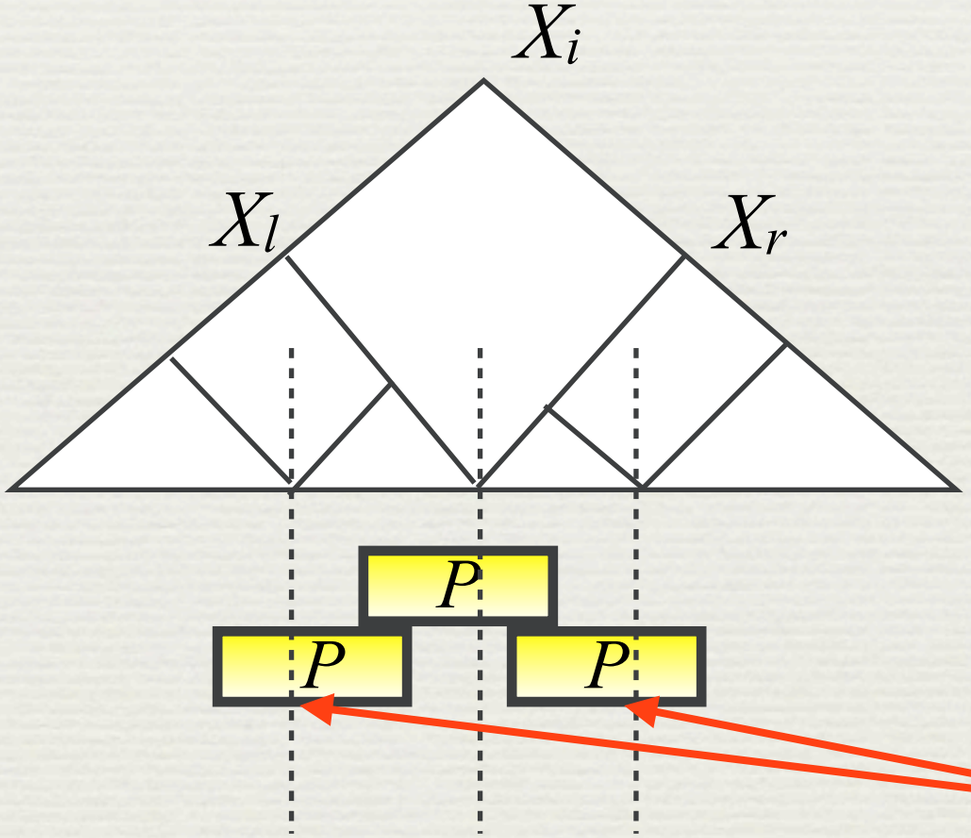


If $b < q$ or $|X_i|$-$q <$ e,
loc can be overlaped the q-gram crossing in X_i's ancestor

We count the crossing interval [b-q+1:e+q-1]

Difference of Algorithm for Overlapping Frequencies Problem

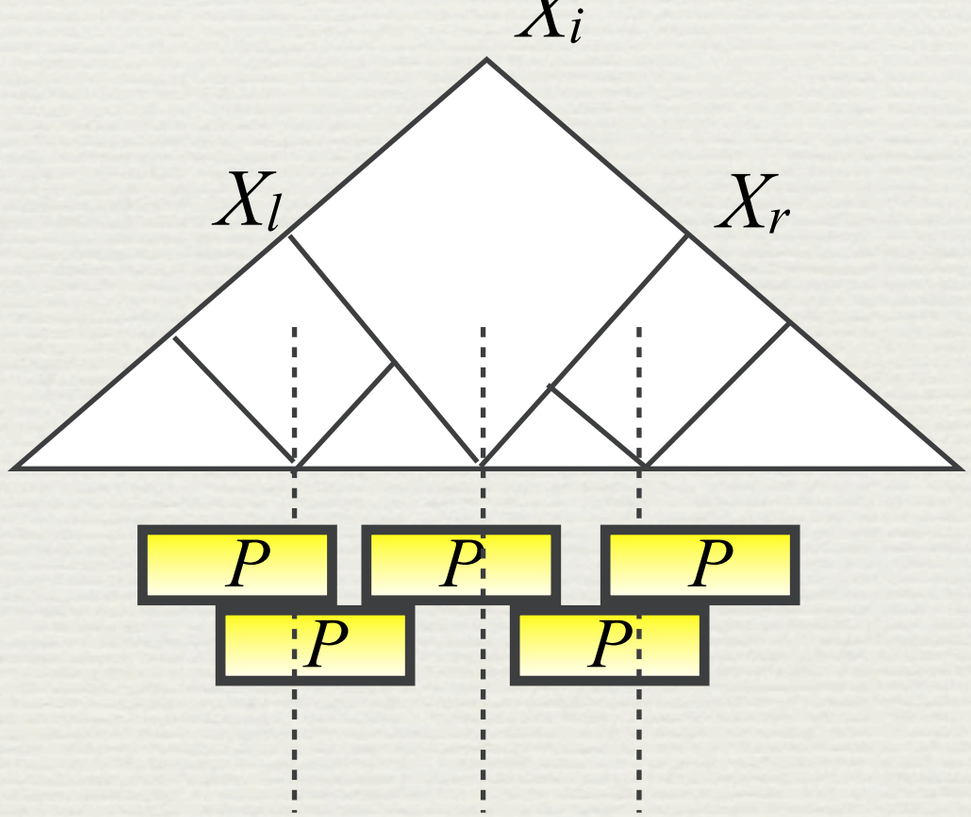
- overlapping algorithm doesn't care whether or not a q-gram which crosses in X_i is overlapping the same q-gram in X_l or X_r



non-overlapping frequency of P is 2, but algorithm count 3 time X_i, X_l, X_r

Difference of Algorithm for Overlapping Frequencies Problem

- Important Key is that non-overlapping frequency depend on the way of overlapping in the chain of overlapping



We want to know

- the interval of the maximum chain of

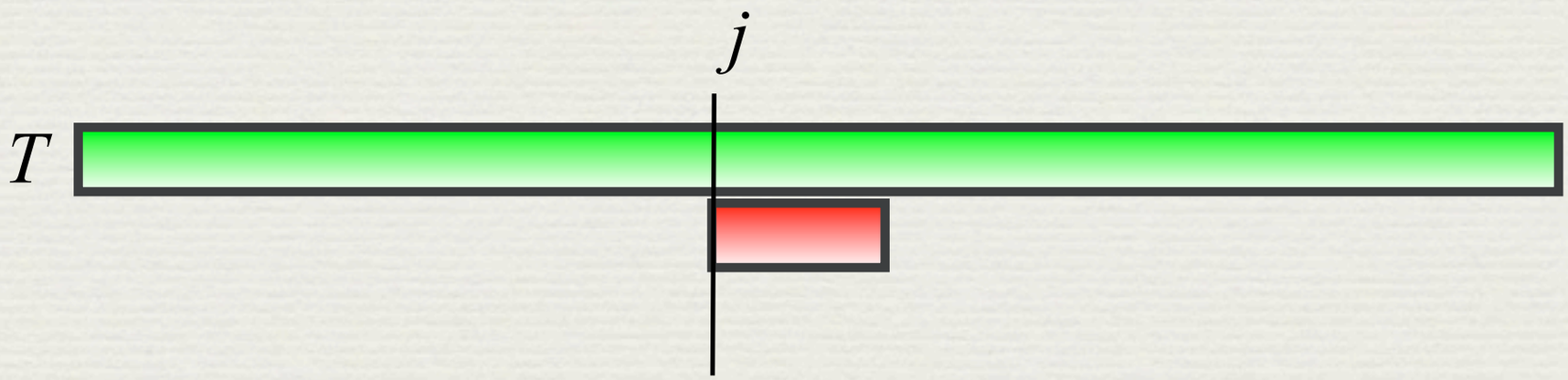
overlapping occurrences of q-gram

- non-overlapping frequency in the interval

Longest Overlapping Cover

longest overlapping cover

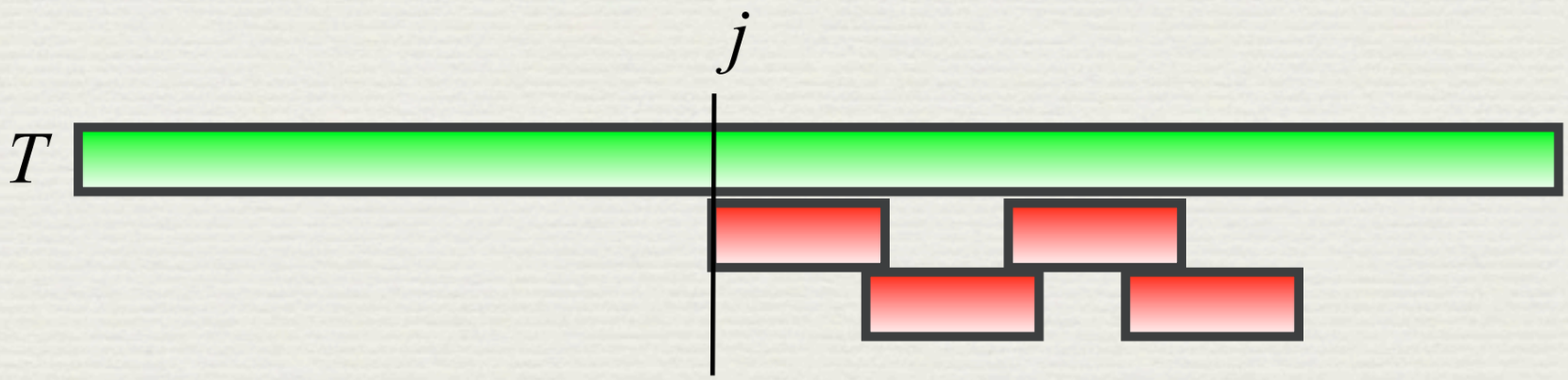
$\overleftrightarrow{loc_q}(T, j) = (b, e)$ is the beginning and ending positions of the maximum chain of overlapping occurrences of q-gram $T[j : j + q - 1]$



Longest Overlapping Cover

longest overlapping cover

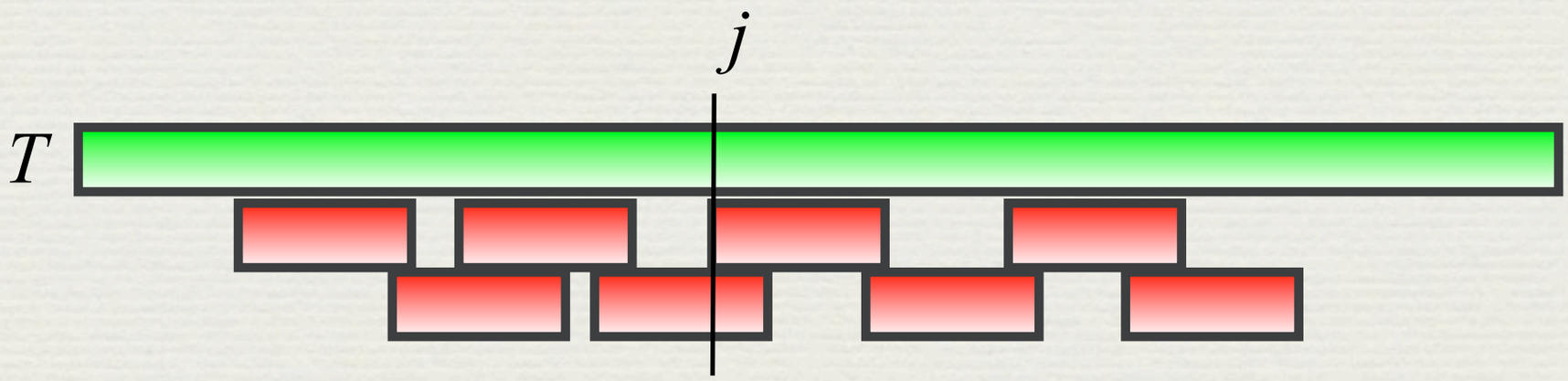
$\overleftrightarrow{loc_q}(T, j) = (b, e)$ is the beginning and ending positions of the maximum chain of overlapping occurrences of q-gram $T[j : j + q - 1]$



Longest Overlapping Cover

longest overlapping cover

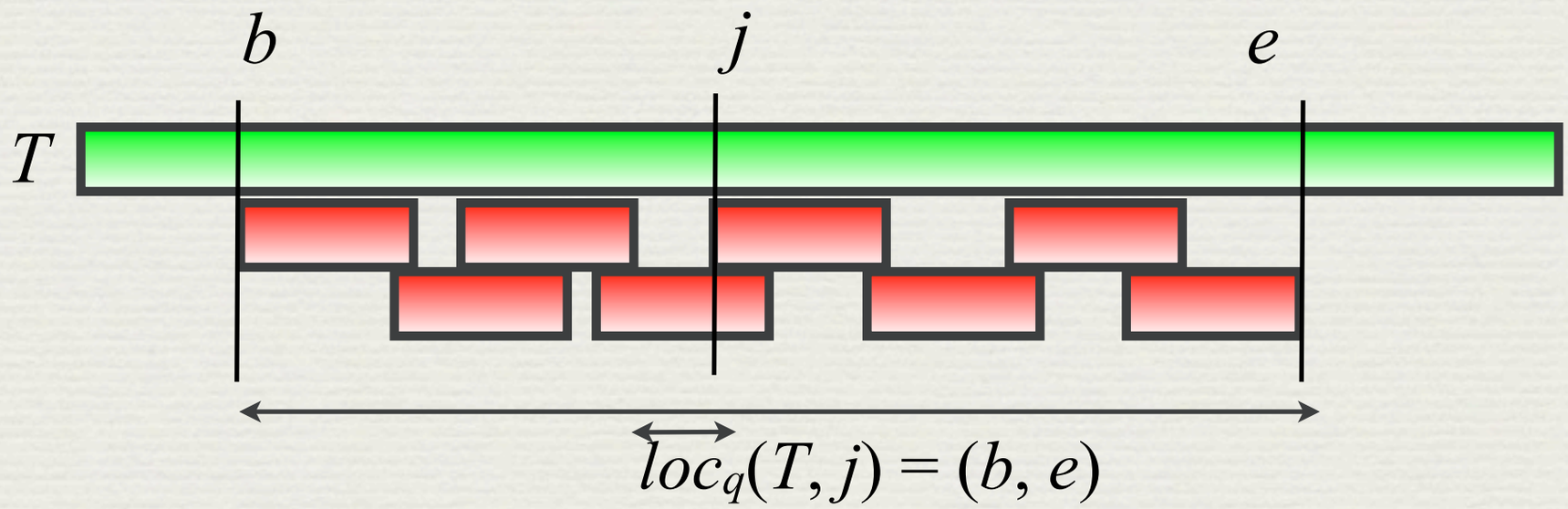
$\overleftrightarrow{loc_q}(T, j) = (b, e)$ is the beginning and ending positions of the maximum chain of overlapping occurrences of q-gram $T[j : j + q - 1]$



Longest Overlapping Cover

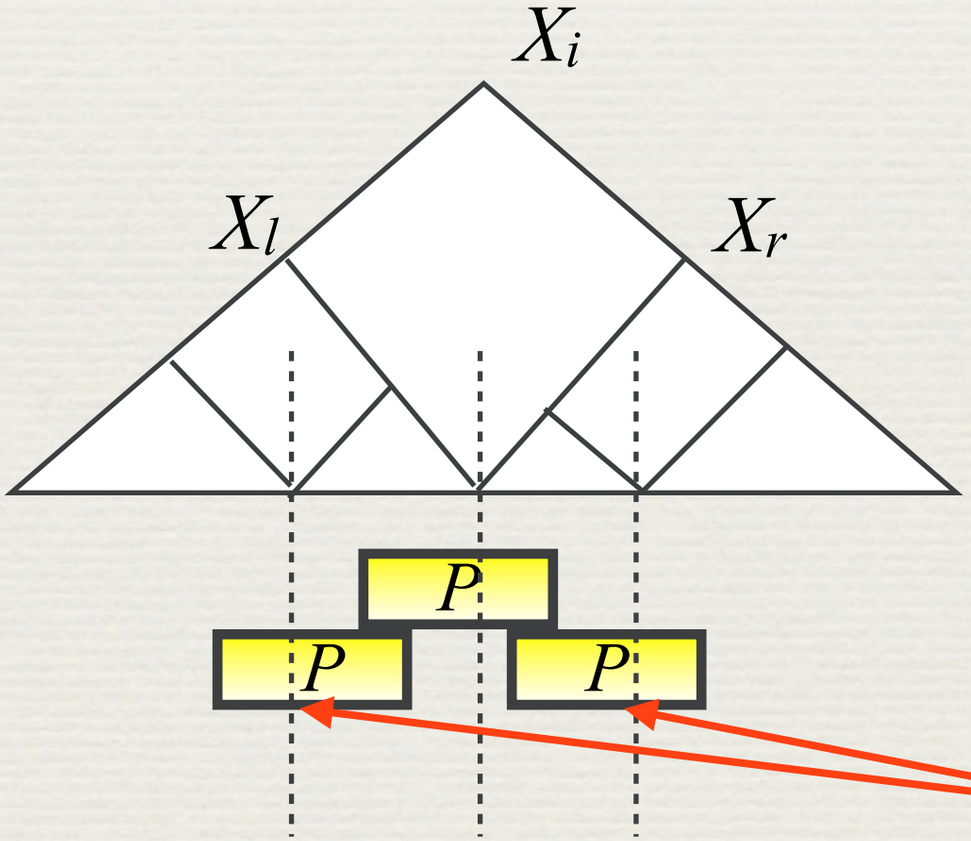
longest overlapping cover

$\overleftrightarrow{loc_q}(T, j) = (b, e)$ is the beginning and ending positions of the maximum chain of overlapping occurrences of q-gram $T[j : j + q - 1]$



Difference of Algorithm for Overlapping Frequencies Problem

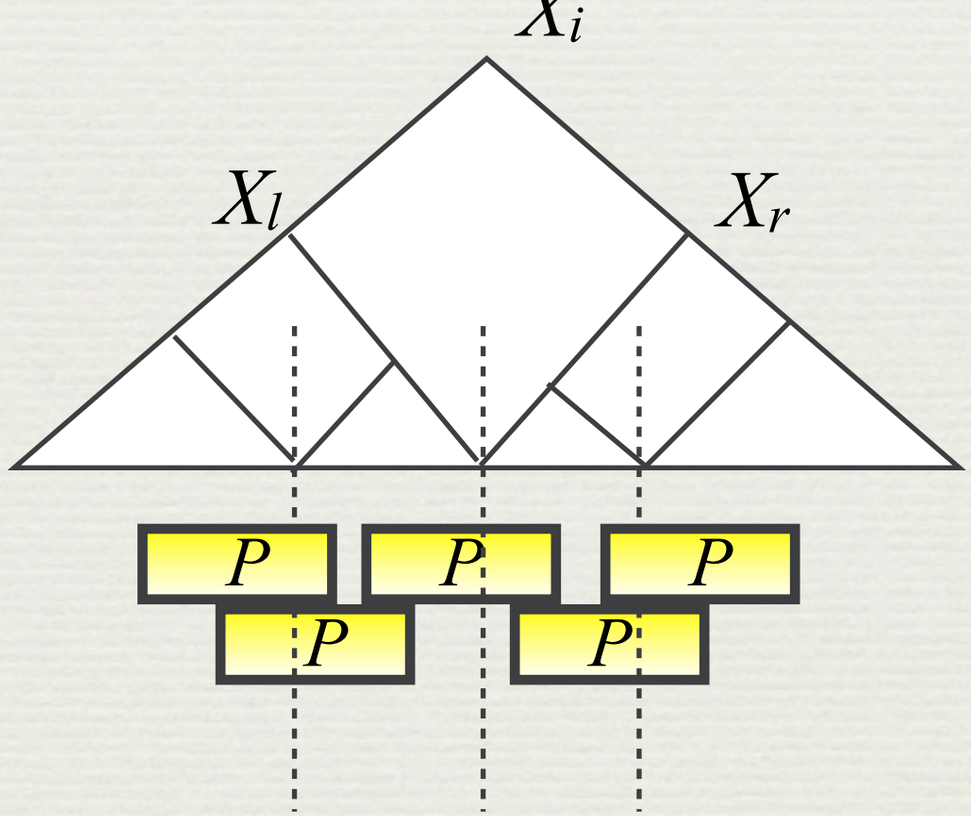
- overlapping algorithm doesn't care whether or not a q-gram which crosses in X_i is overlapping the same q-gram in X_l or X_r



non-overlapping frequency of P is 2, but algorithm count 3 time X_i, X_l, X_r

Difference of Algorithm for Overlapping Frequencies Problem

- Important Key is that non-overlapping frequency depend on the way of overlapping in the chain of overlapping



We want to know

- the interval of the maximum chain of

overlapping occurrences of q-gram

- non-overlapping frequency in the interval