Speeding up q-gram mining on grammar based compressed text

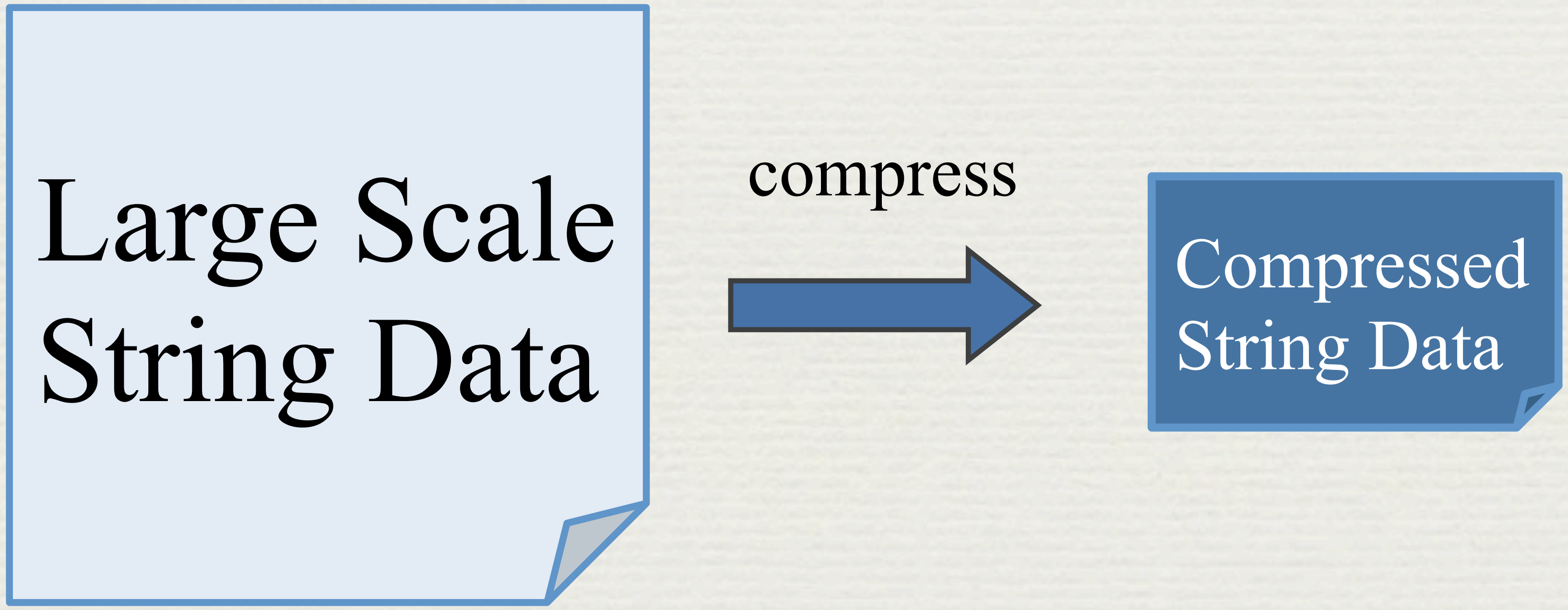
Kyushu University

○Keisuke Goto, Hideo Bannai,

Shunsuke Inenaga, Masayuki Takeda

Background: Processing large scale string data

- Data compression allows large scale string data to be stored compactly



Background: Processing large scale string data

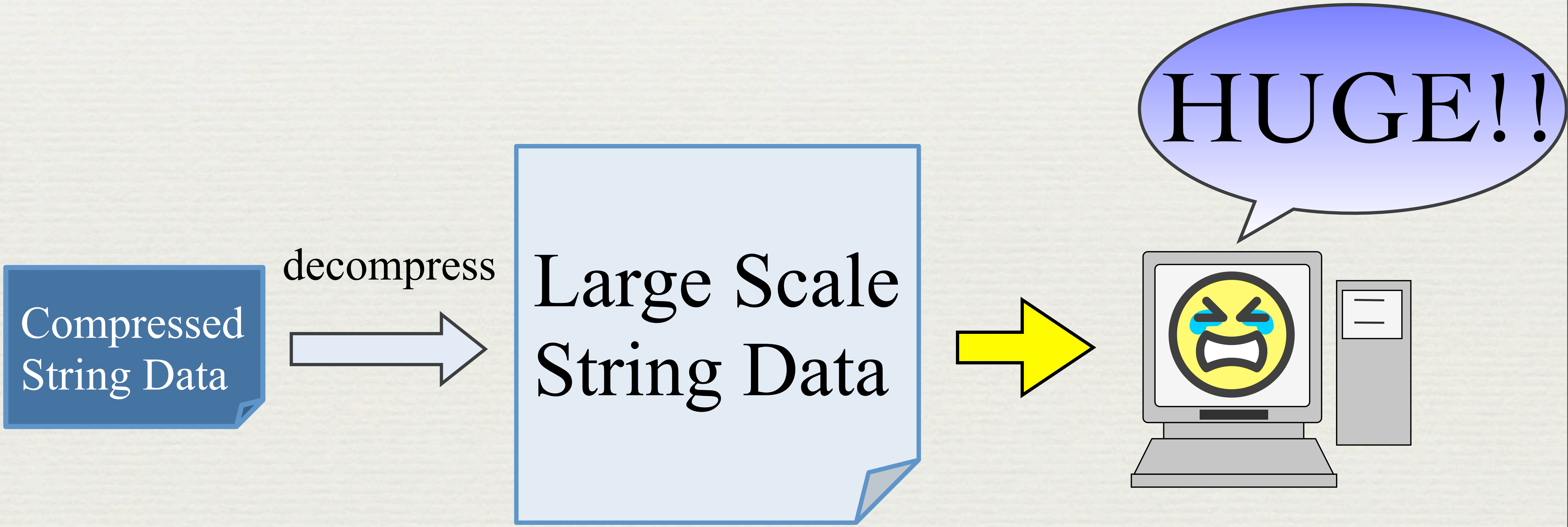
- In order to process such data, we usually decompress them, which requires a lot of space and time.

Background: Processing large scale string data

- In order to process such data, we usually decompress them, which requires a lot of space and time.

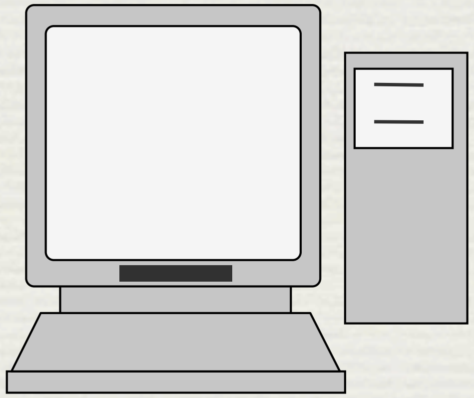
Background: Processing large scale string data

- In order to process such data, we usually decompress them, which requires a lot of space and time.



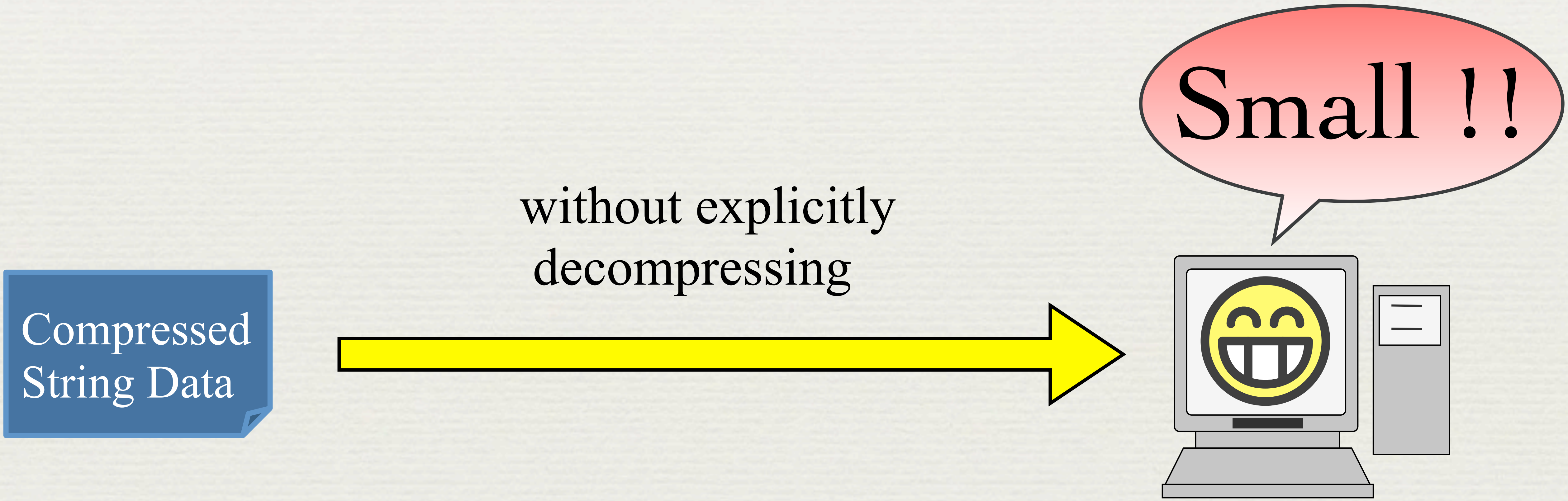
Background: Processing large scale string data

- One solution is to process compressed strings **without explicit decompression.**



Background: Processing large scale string data

- One solution is to process compressed strings **without explicit decompression.**



Grammar-Based Compressed String Processing

Problem	Previous Work
Equality Test	[Plandowski ’94]; Lifshits ’07]; [Schmidt-Schauss+ ’09];
Pattern Match	[Karpinski+ '97], [Miyazaki+ '97], [Inenaga+ '04], [Lifshits ’06], [Gawrychowski '11]
Approximate Pattern Match	[Matsumoto+ ’00]; [Navarro+ ’01]
Subsequence Match	[Cegielski+ ’00]; [Tiskin ’09]; [Yamamoto+ ’11]
Longest Common Subsequence / Edit Distance	[Tiskin ’07, ’08]; [Hermelin + ’09, ’11]
Pattern Discovery	[Inenaga+ ’09]; [Matsubara+ ’09]
q-gram Frequencies	[Goto+ ’11]; [Goto+ ’12]

Grammar-Based Compressed String Processing

Problem	Previous Work
Equality Test	[Plandowski ’94]; Lifshits ’07]; [Schmidt-Schauss+ ’09];
Pattern Match	[Karpinski+ '97], [Miyazaki+ '97], [Inenaga+ '04], [Lifshits ’06], [Gawrychowski '11]
Approximate Pattern Match	[Matsumoto+ ’00]; [Navarro+ ’01]
Subsequence Match	[Cegielski+ ’00]; [Tiskin ’09]; [Yamamoto+ ’11]
Longest Common Subsequence / Edit Distance	[Tiskin ’07, ’08]; [Hermelin + ’09, ’11]
Pattern Discovery	[Inenaga+ ’09]; [Matsubara+ ’09]
q-gram Frequencies	[Goto+ ’11]; [Goto+ ’12]

Main contribution

	Uncompressed String	SLP (SPIRE 2011)	SLP (This work)
q-gram Freq	$O(\|T\|) = O(2^n)$ time and space	$O(qn)$ time and space	$O(\min\{qn, \|T\|\text{-}dup(q, D)\})$ time and space

T : uncompressed string, n : the size of SLP

$dup(q, D)$: a quantity that represents the amount of redundancy that the SLP D captures with respect to q-grams

The algorithm is asymptotically always at least as fast and better in many cases compared to working on the uncompressed string

q-gram frequencies problem

Definition

Input：string T, positive integer q

Output：$\{(P, Freq(T, P)) \mid P \in \Sigma^q, Freq(T, P) > 0\}$

where $Freq(T, P)$ is # occurrences of P in T

q-gram frequencies problem

Definition

Input : string T, positive integer q

Output : $\{(P, Freq(T, P)) \mid P \in \Sigma^q, Freq(T, P) > 0\}$

where $Freq(T, P)$ is # occurrences of P in T

Example $q = 3$

$T =$ `abaababaab`

q-gram frequencies problem

Definition

Input : string T, positive integer q

Output : $\{(P, Freq(T, P)) \mid P \in \Sigma^q, Freq(T, P) > 0\}$

where $Freq(T, P)$ is # occurrences of P in T

Example $q = 3$

```
T = abaababaab
    aba
     baa
      aab
       aba
        bab
         aba
          baa
           aab
```

q-gram frequencies problem

Definition

Input : string T, positive integer q

Output : $\{(P, Freq(T, P)) \mid P \in \Sigma^q, Freq(T, P) > 0\}$

where $Freq(T, P)$ is # occurrences of P in T

Example $q = 3$

```
T = abaababaab
    aba
     baa
      aab
       aba
        bab
         aba
          baa
           aab
```

$Freq(T, \text{“aba”}) = 3$

q-gram frequencies problem

Definition

Input : string T, positive integer q

Output : $\{(P, Freq(T, P)) \mid P \in \Sigma^q, Freq(T, P) > 0\}$

where $Freq(T, P)$ is # occurrences of P in T

Example $q = 3$

```
T = abaababaab
    aba
     baa
      aab
       aba
        bab
         aba
          baa
           aab
```

$Freq(T, \text{“aba”}) = 3$

$Freq(T, \text{“baa”}) = 2$

q-gram frequencies problem

Definition

Input : string T, positive integer q

Output : $\{(P, Freq(T, P)) \mid P \in \Sigma^q, Freq(T, P) > 0\}$

where $Freq(T, P)$ is # occurrences of P in T

Example $q = 3$

```
T = abaababaab
    aba
     baa
      aab
       aba
        bab
         aba
          baa
           aab
```

$Freq(T, \text{"aba"}) = 3$

$Freq(T, \text{"baa"}) = 2$

$Freq(T, \text{"aab"}) = 2$

q-gram frequencies problem

Definition

Input : string T, positive integer q

Output : $\{(P, Freq(T, P)) \mid P \in \Sigma^q, Freq(T, P) > 0\}$

where $Freq(T, P)$ is # occurrences of P in T

Example $q = 3$

```
T = abaababaab
    aba
     baa
      aab
       aba
        bab
         aba
          baa
           aab
```

$Freq(T, \text{"aba"}) = 3$

$Freq(T, \text{"baa"}) = 2$

$Freq(T, \text{"aab"}) = 2$

$Freq(T, \text{"bab"}) = 1$

Straight Line Program (SLP)

Definition

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

$X_1 = expr_1, X_2 = expr_2, \ldots., X_n = expr_n$

$expr_i \in \Sigma$ or

$expr_i = X_l \cdot X_r \ (l, r < i)$

SLP can represent the output of well-known compression algorithms

- e.g. RE-PAIR, SEQUITUR, LZ78, LZW, LZ77, LZSS

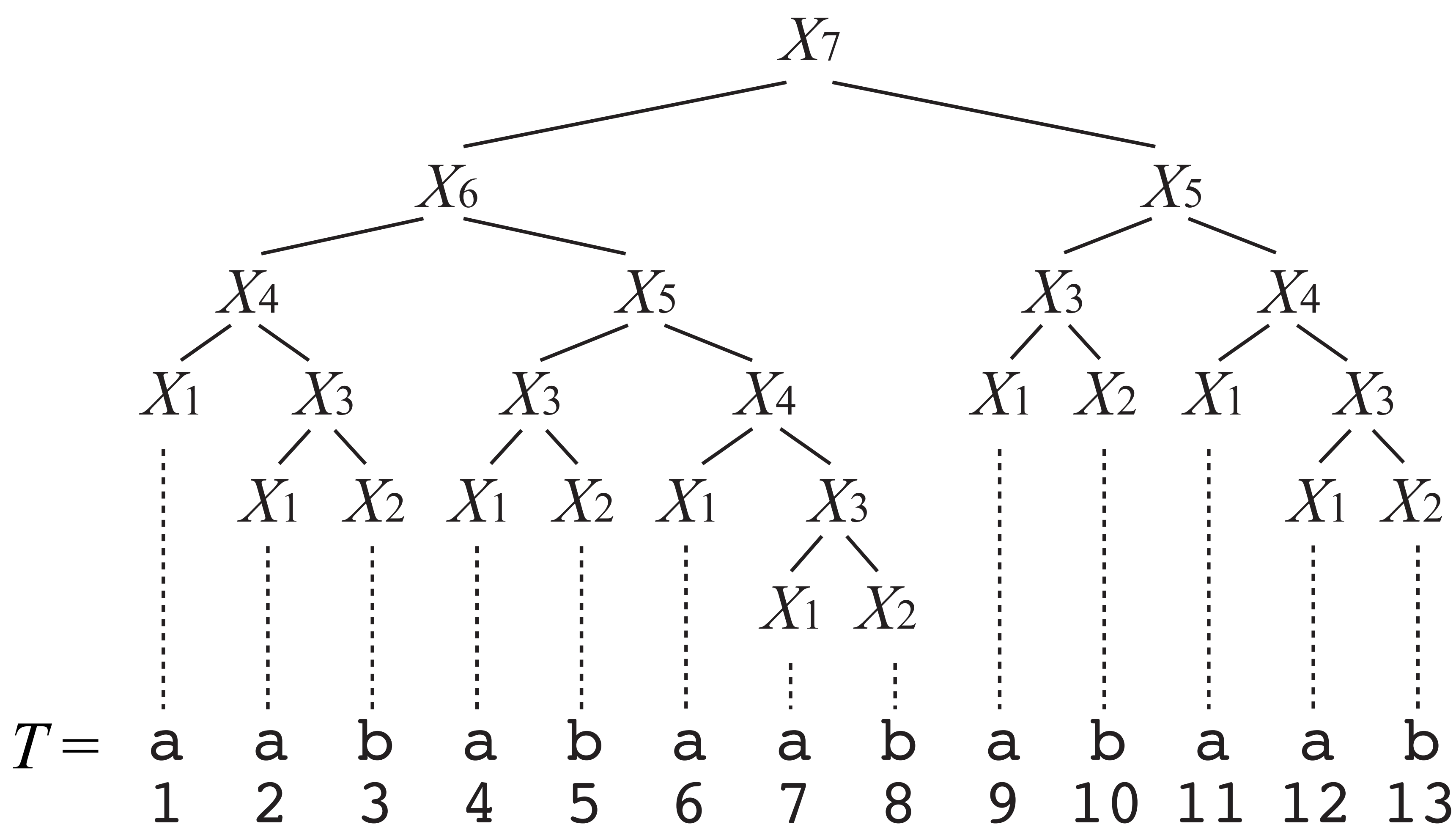
Example of SLP

SLP: D

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

$n = |D| = 7$

Derivation Tree of D



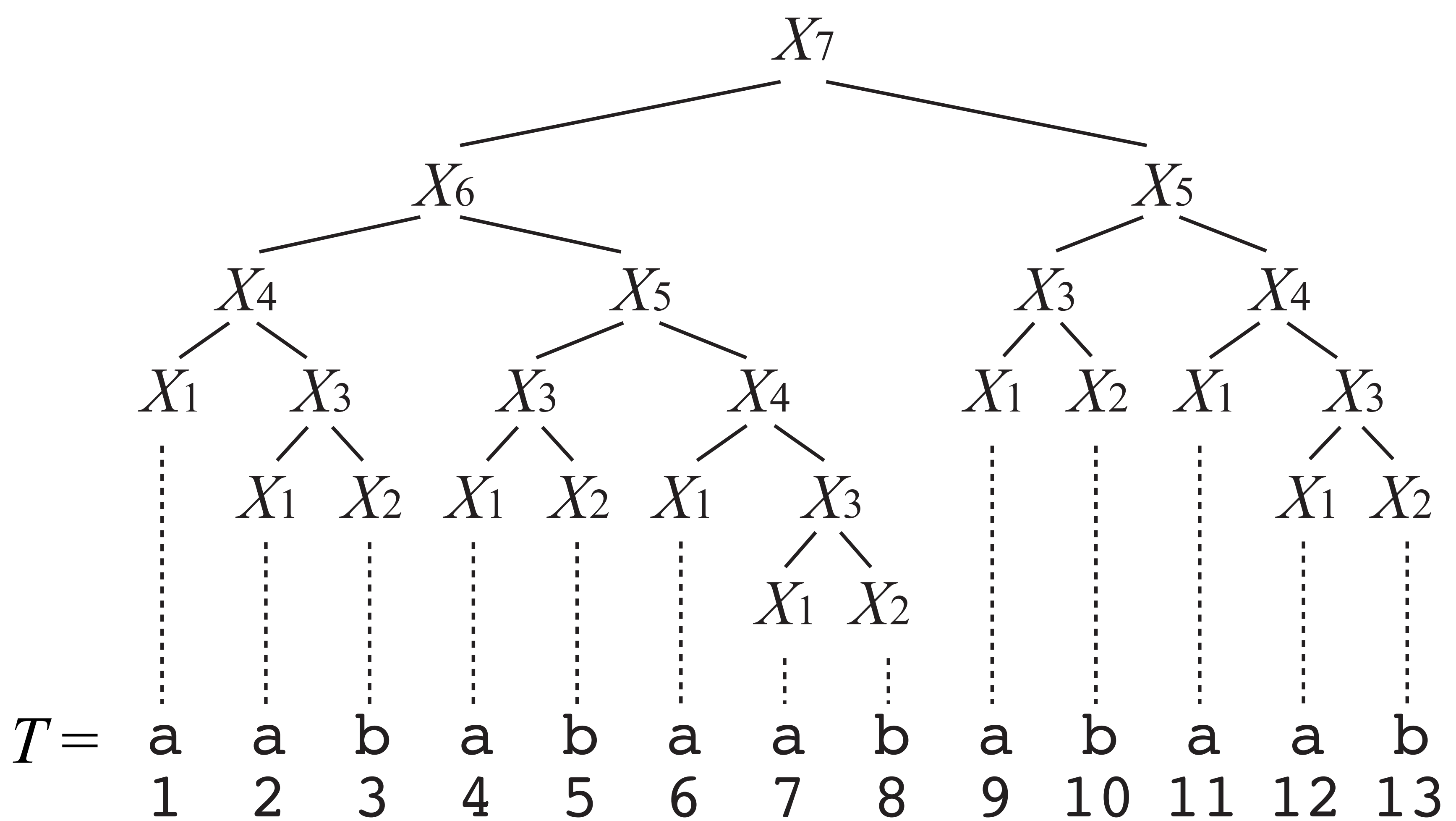
Example of SLP

SLP: D

$$X_1 = \mathrm{a}$$
$$X_2 = \mathrm{b}$$
$$X_3 = X_1 X_2$$
$$X_4 = X_1 X_3$$
$$X_5 = X_3 X_4$$
$$X_6 = X_4 X_5$$
$$X_7 = X_6 X_5$$

$n = |D| = 7$

Derivation Tree of D



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

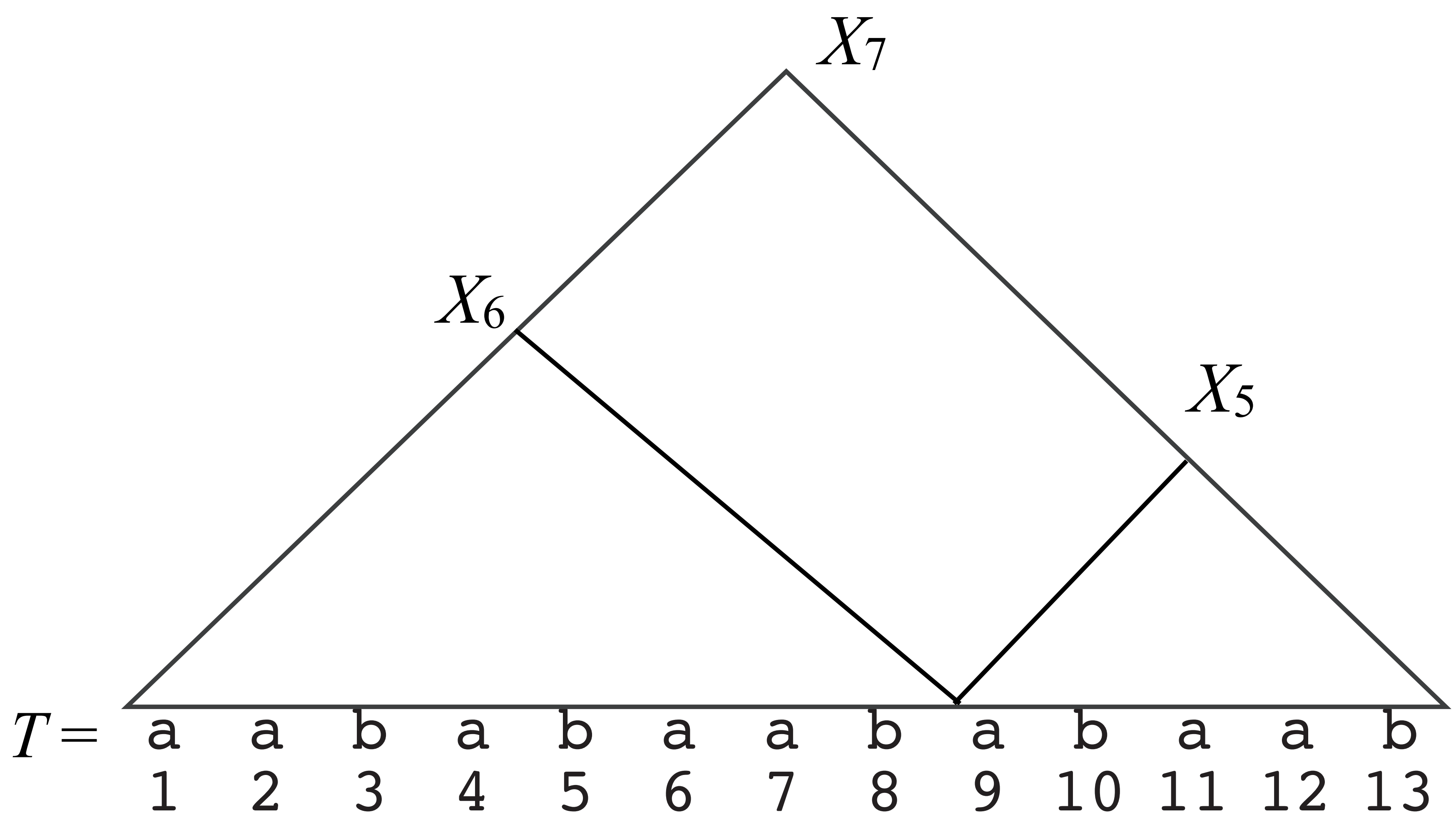
Example of SLP

SLP: D

$$X_1 = \mathrm{a}$$
$$X_2 = \mathrm{b}$$
$$X_3 = X_1 X_2$$
$$X_4 = X_1 X_3$$
$$X_5 = X_3 X_4$$
$$X_6 = X_4 X_5$$
$$X_7 = X_6 X_5$$

$n = | D | = 7$

Derivation Tree of D



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

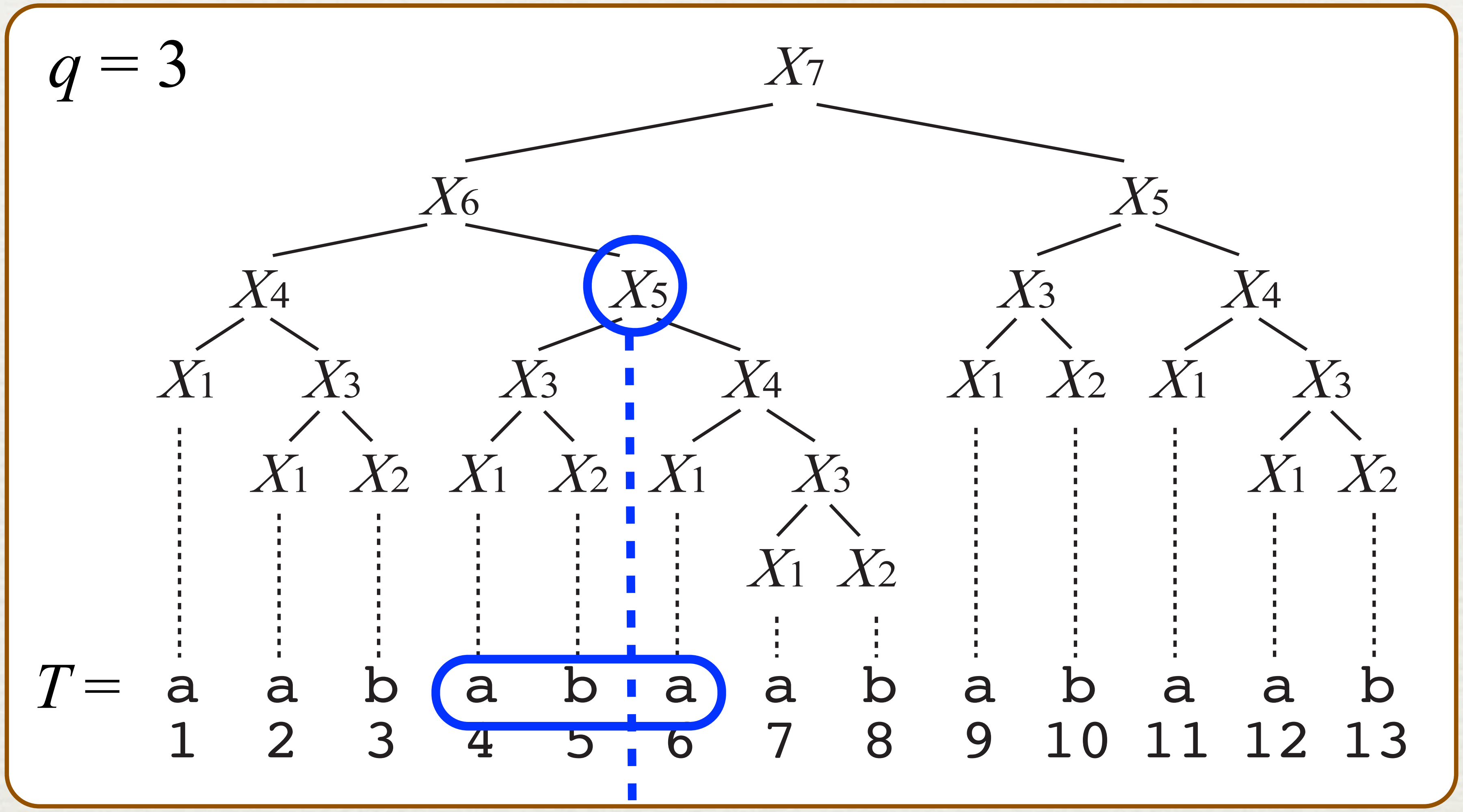
$O(qn)$ algorithm for q-gram frequencies problem on SLP

[Goto et al., SPIRE 2011]

Important Observation: stabbing

Definition

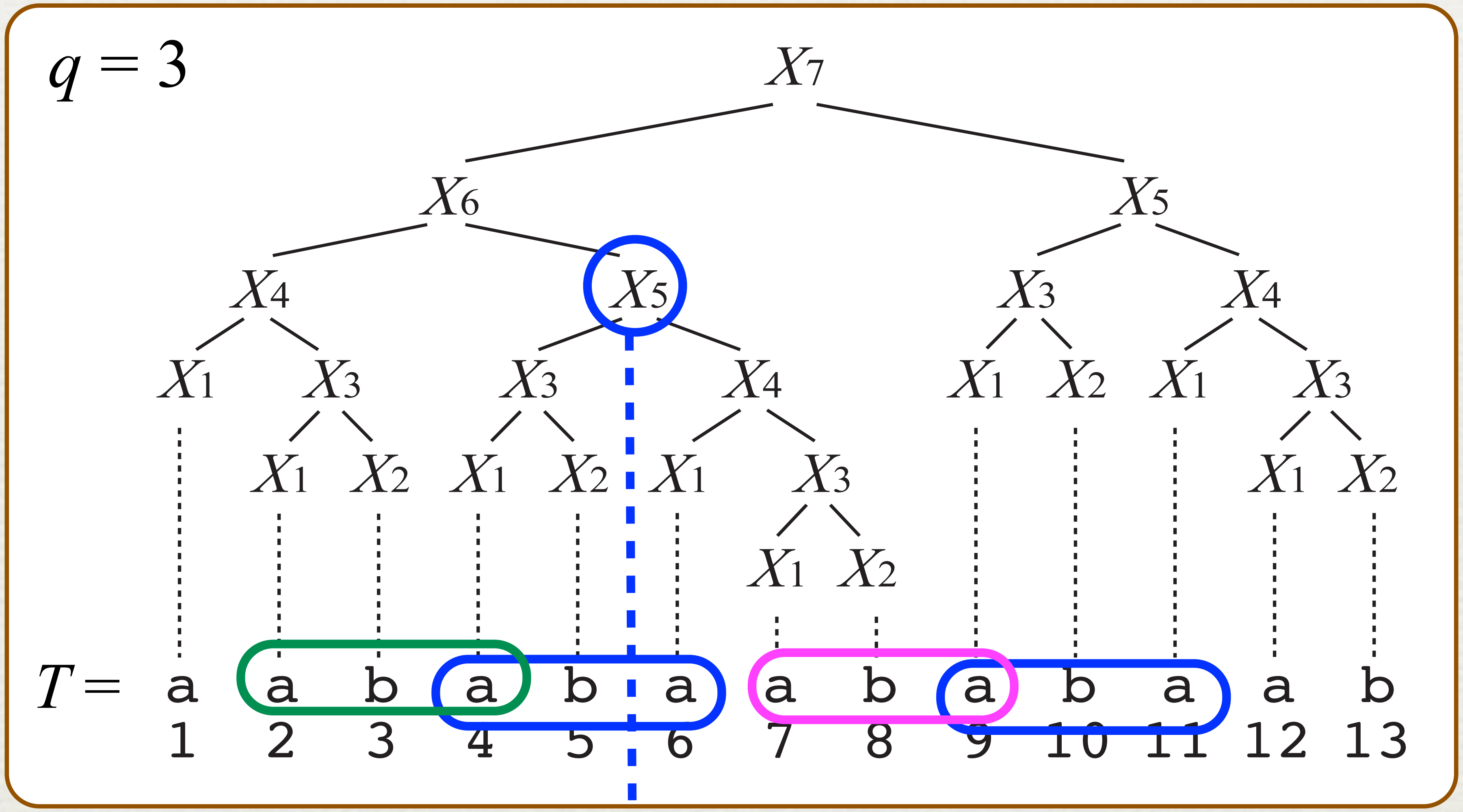
For $X_i = X_l X_r$, X_i stabs an occurrence of P $\Leftrightarrow$ P starts in X_l and ends in X_r



Important Observation: stabbing

Definition

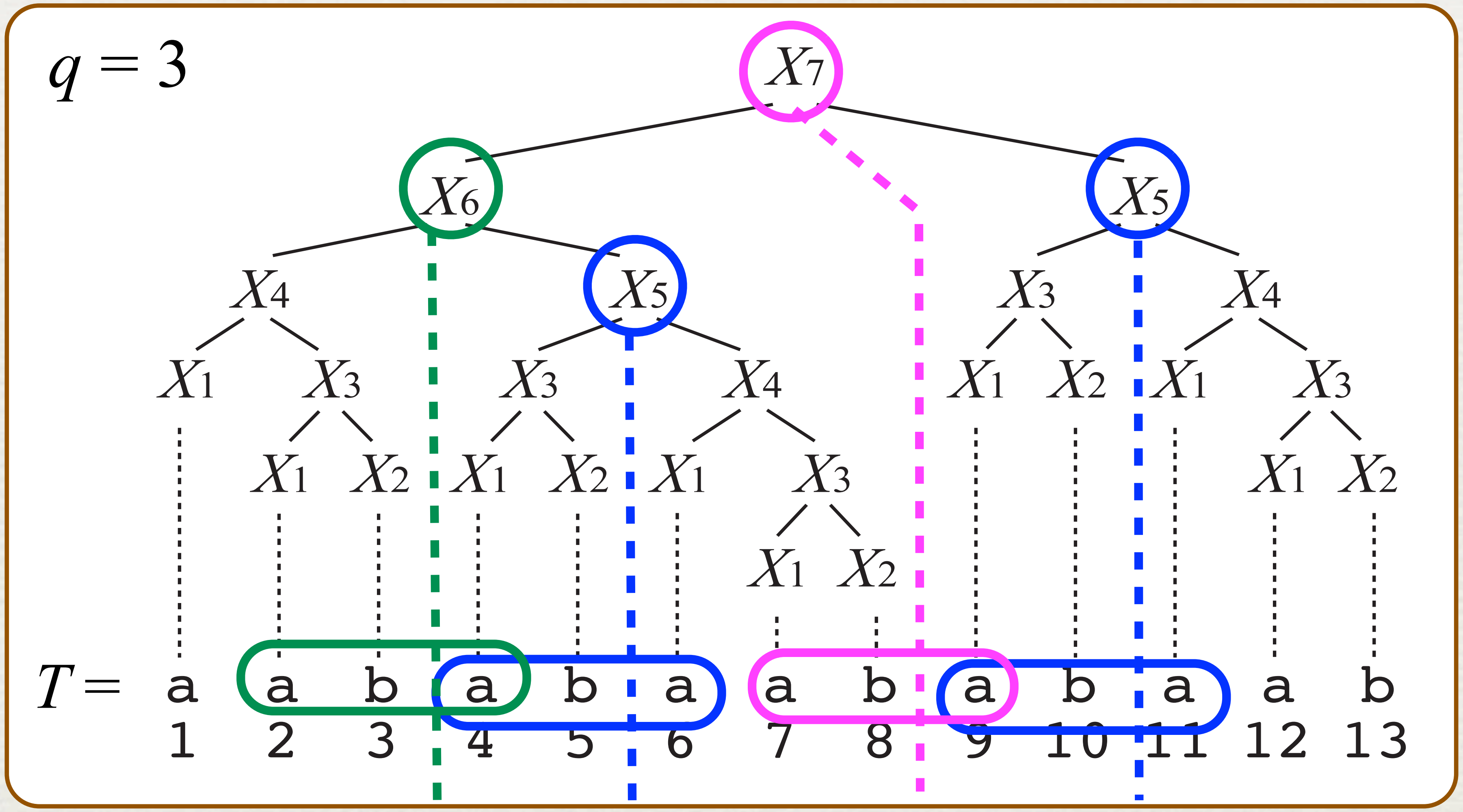
For $X_i = X_l X_r$, X_i stabs an occurrence of $P \Leftrightarrow P$ starts in X_l and ends in X_r



Important Observation: stabbing

Definition

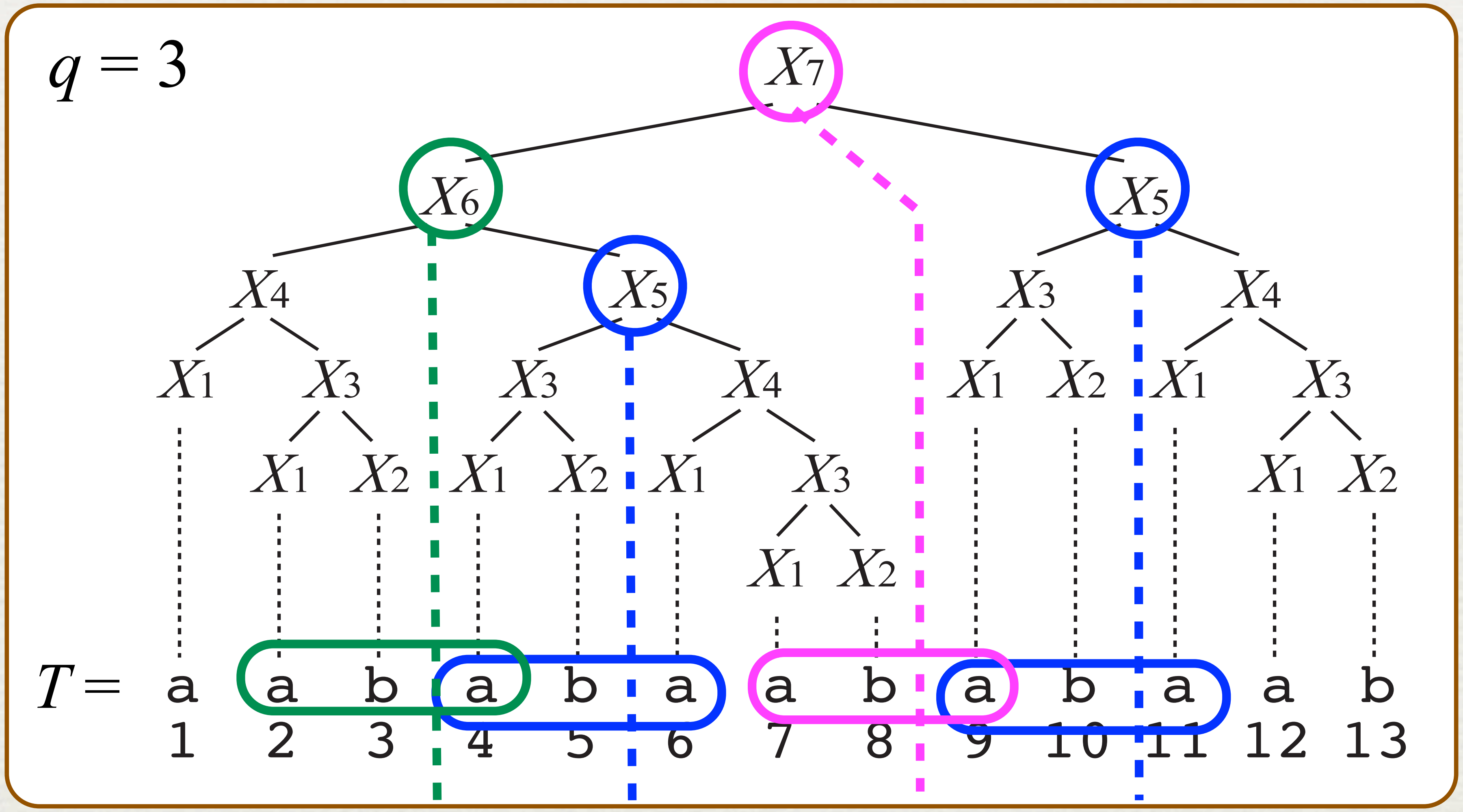
For $X_i = X_l X_r$, X_i stabs an occurrence of P $\Leftrightarrow$ P starts in X_l and ends in X_r



Important Observation: stabbing

Definition

For $X_i = X_l X_r$, X_i stabs an occurrence of P $\Leftrightarrow$ P starts in X_l and ends in X_r



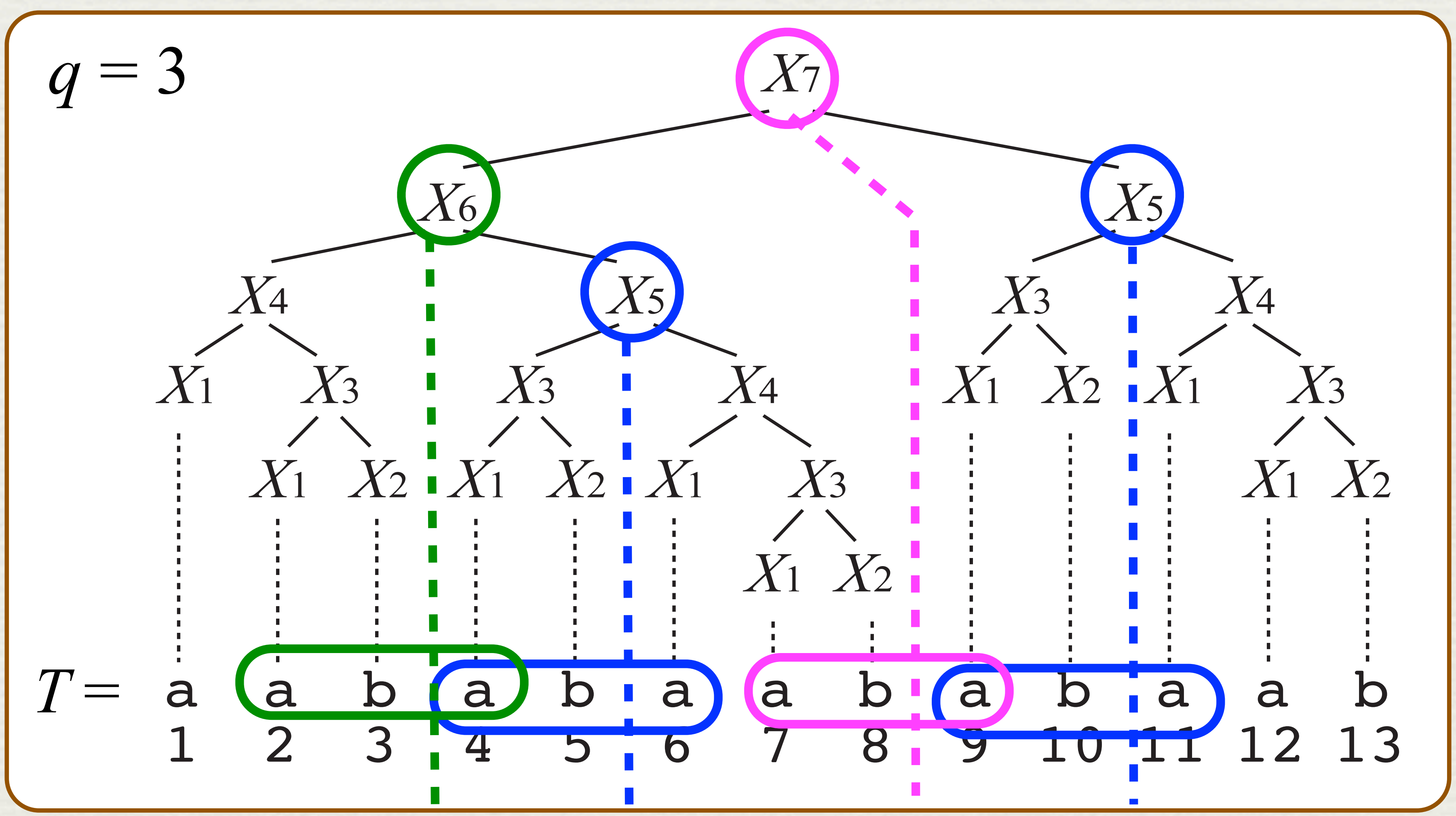
Observation

For each occurrence of q-gram P, there exists a unique variable which stabs the occurrence of P

Important idea: counting stabbed occurrences

We can compute $Freq(T, P)$ by counting the number of occurrences of P stabbed by X_i, and summing them up for all X_i

$$Freq(T, P) = \underbrace{2 \cdot 1}_{X_5} + \underbrace{1}_{X_6} + \underbrace{1}_{X_7}$$

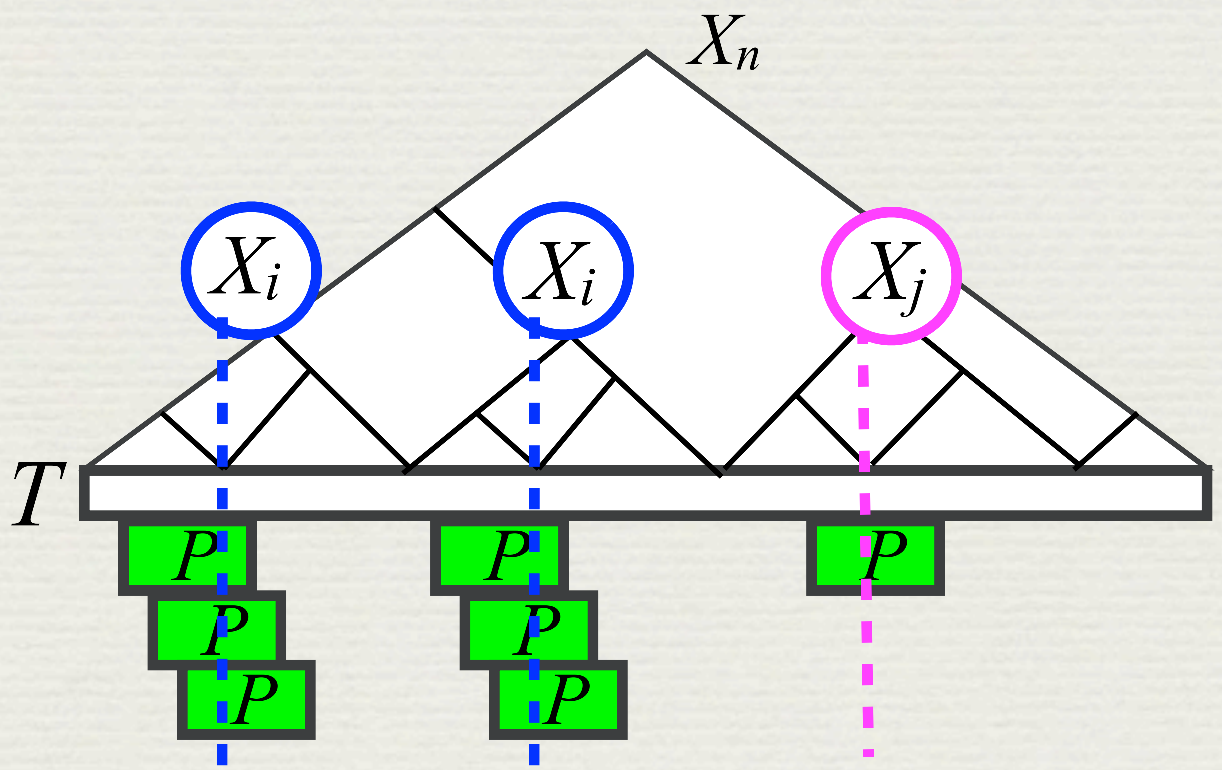


More formal description

Definition

For each variable X_i,

- $Freq^{串}(X_i, P)$: # occurrences of P stabbed by X_i in the string derived from X_i.
- $vOcc(X_i)$: # nodes labeled by X_i in the derivation tree of the last variable X_n.



$O(qn)$ Algorithm in 2011

More formal description

Definition

For each variable X_i,

- $\mathit{Freq}^{串}(X_i, P)$: # occurrences of P stabbed by X_i in the string derived from X_i.
- $\mathit{vOcc}(X_i)$: # nodes labeled by X_i in the derivation tree of the last variable X_n.

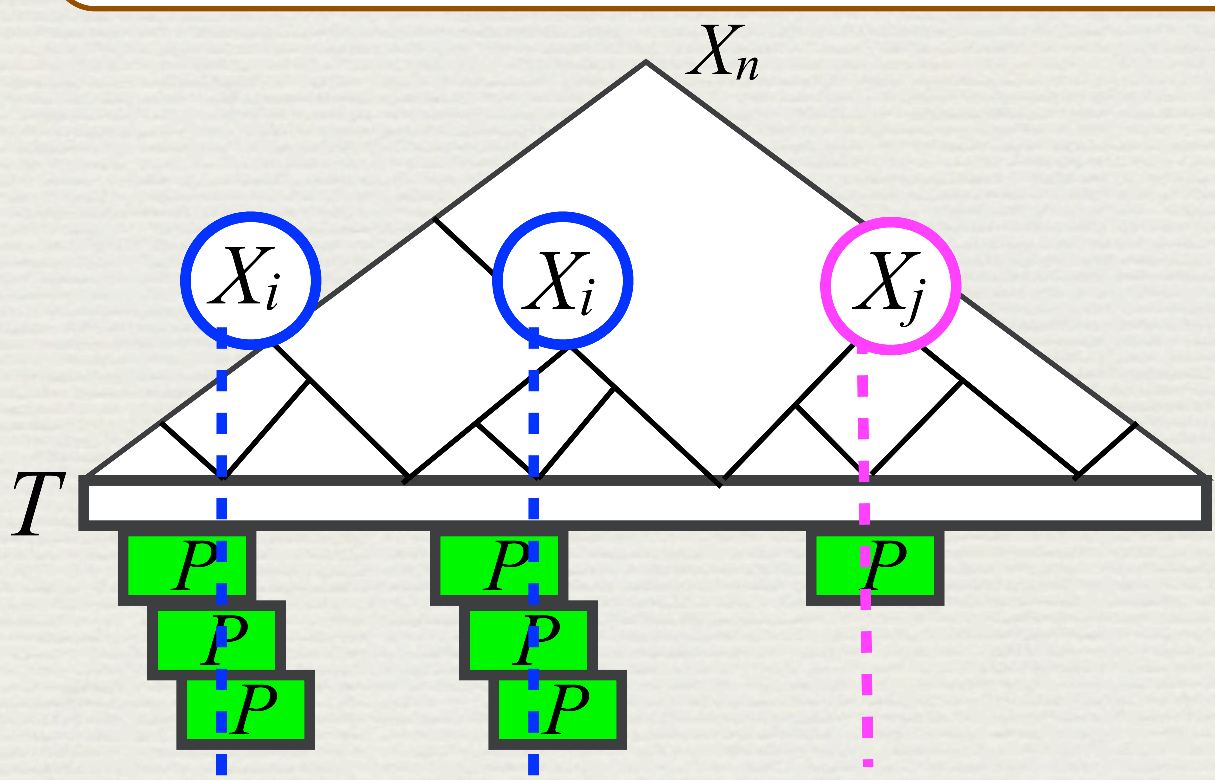
串(Kushi):

Japanese skewer,
used to stab foods

BBQ

Yakitori

Oden



More formal description

Definition

For each variable X_i,

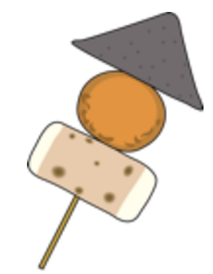
- $Freq^{串}(X_i, P)$: # occurrences of P stabbed by X_i in the string derived from X_i.
- $vOcc(X_i)$: # nodes labeled by X_i in the derivation tree of the last variable X_n.

串(Kushi):

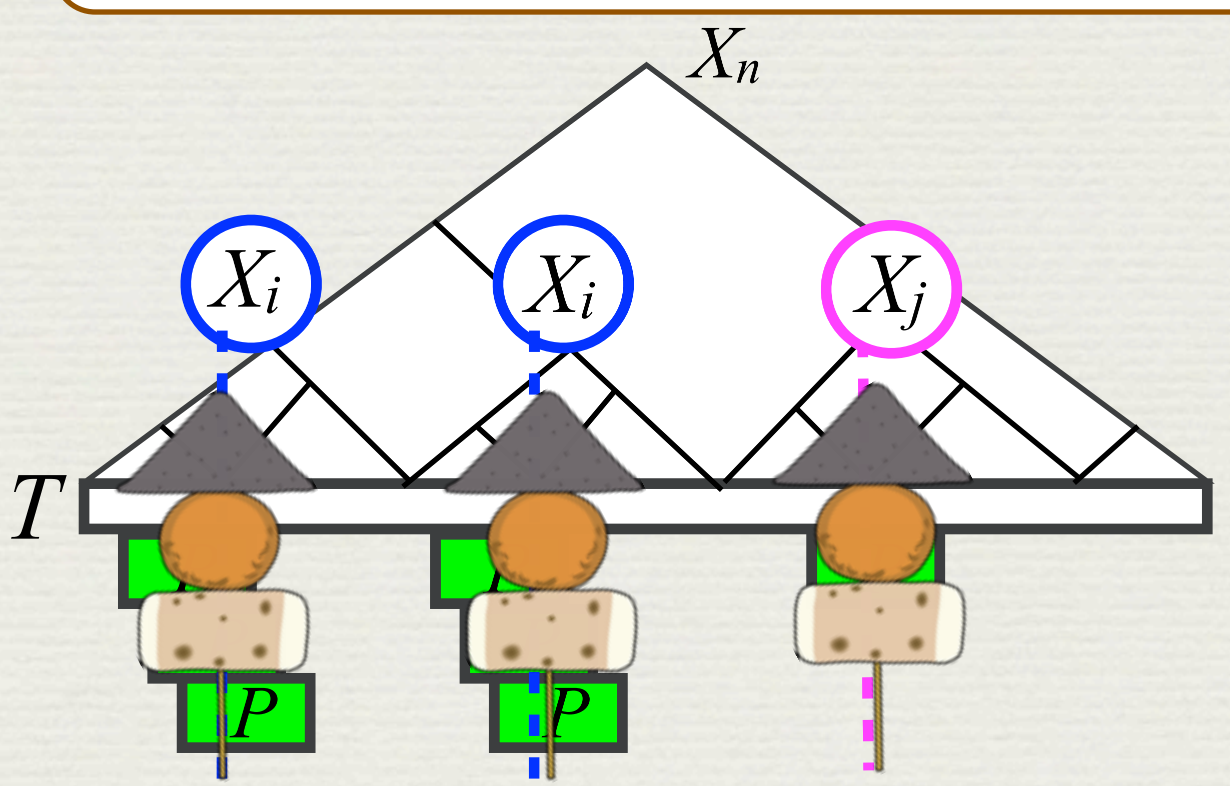
Japanese skewer, used to stab foods

BBQ

Yakitori



Oden

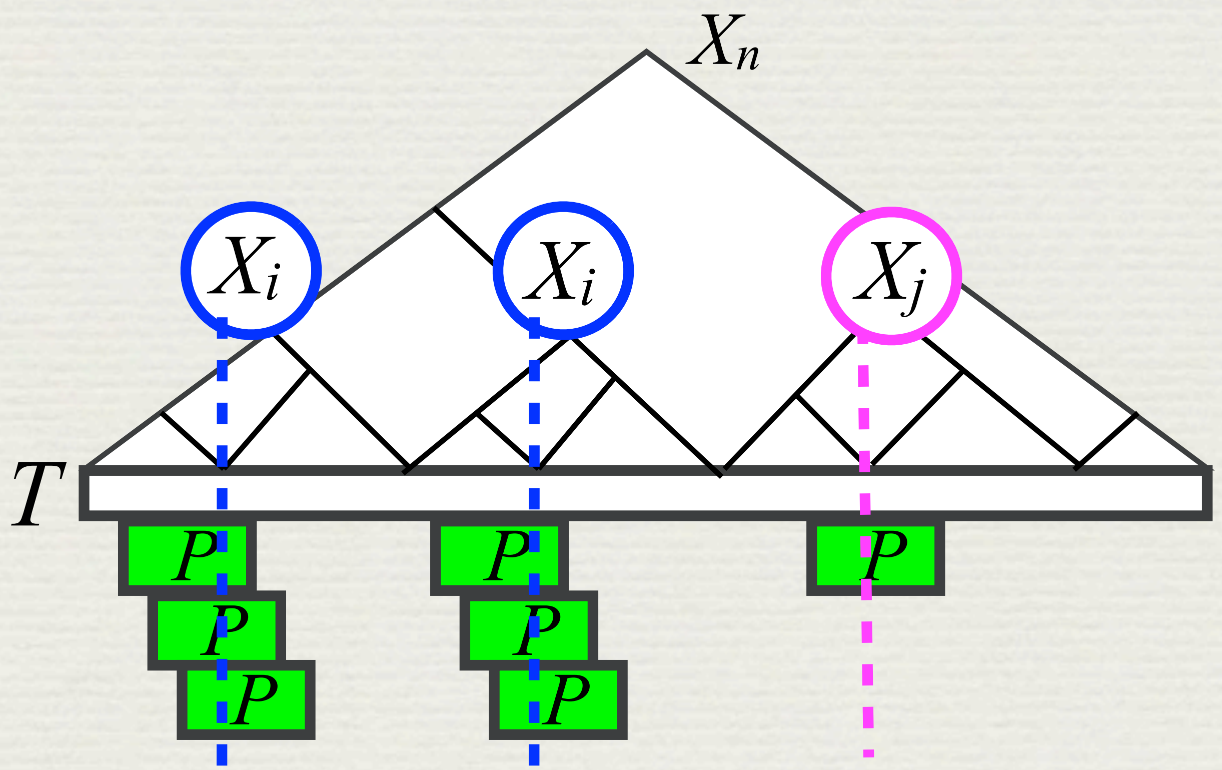


More formal description

Definition

For each variable X_i,

- $Freq^{串}(X_i, P)$: # occurrences of P stabbed by X_i in the string derived from X_i.
- $vOcc(X_i)$: # nodes labeled by X_i in the derivation tree of the last variable X_n.



$Freq^{串}(X_i, P) = 3$, $Freq^{串}(X_j, P) = 1$

$vOcc(X_i) = 2$, $vOcc(X_j) = 1$

Frequency of $P = \underline{3 \cdot 2} + \underline{1 \cdot 1} = 7$

More formal description

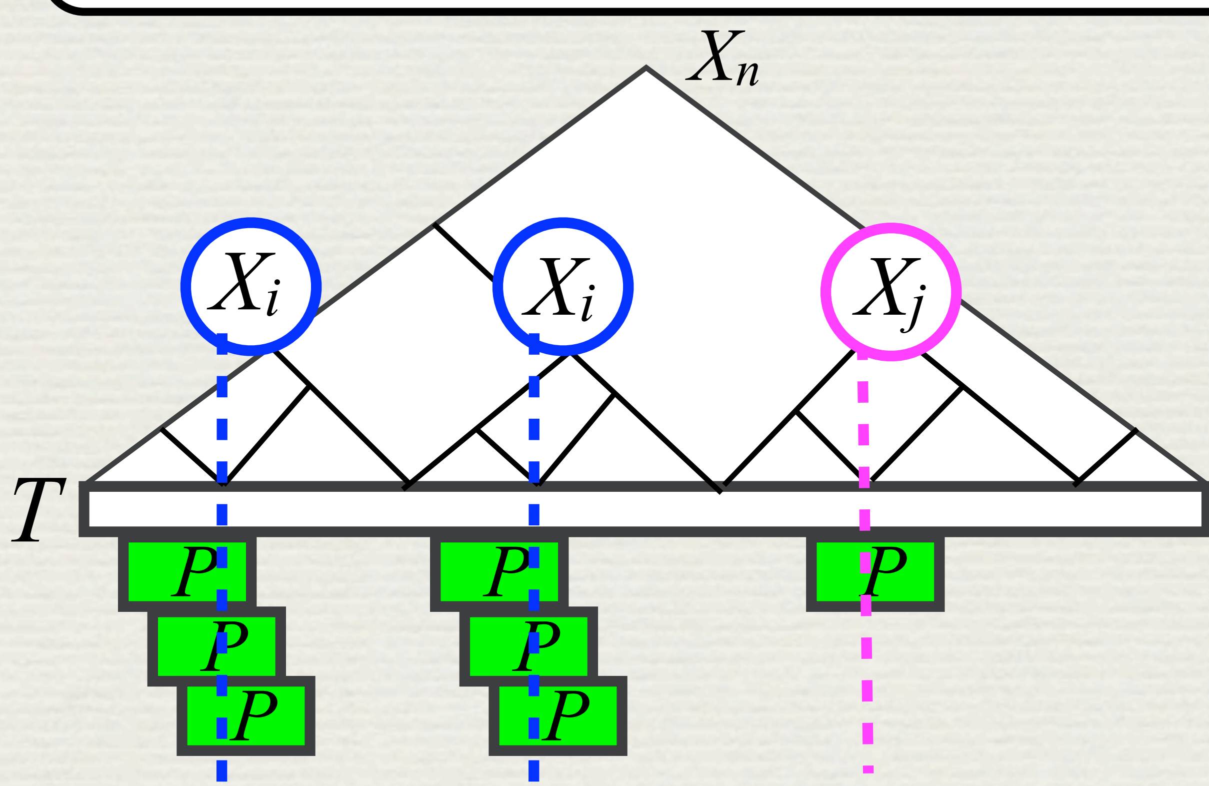
Definition

For each variable X_i,

- $Freq^{串}(X_i, P)$: # occurrences of P stabbed by X_i in the string derived from X_i.
- $vOcc(X_i)$: # nodes labeled by X_i in the derivation tree of the last variable X_n.

Lemma

$$Freq(T,P) = \sum_{i=1}^{n} Freq^{串}(X_i,P) \cdot vOcc(X_i)$$

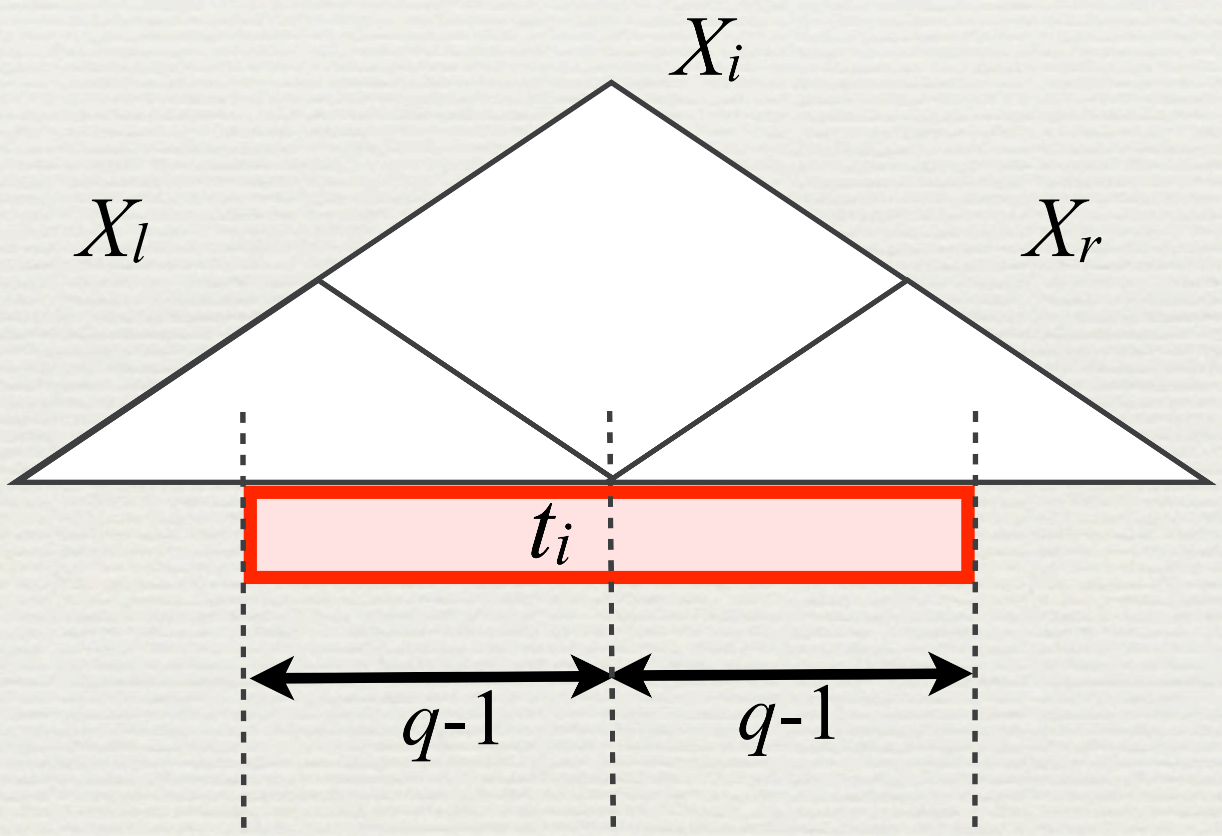


$Freq^{串}(X_i, P) = 3$, $Freq^{串}(X_j, P) = 1$

$vOcc(X_i) = 2$, $vOcc(X_j) = 1$

Frequency of $P = 3 \cdot 2 + 1 \cdot 1 = 7$

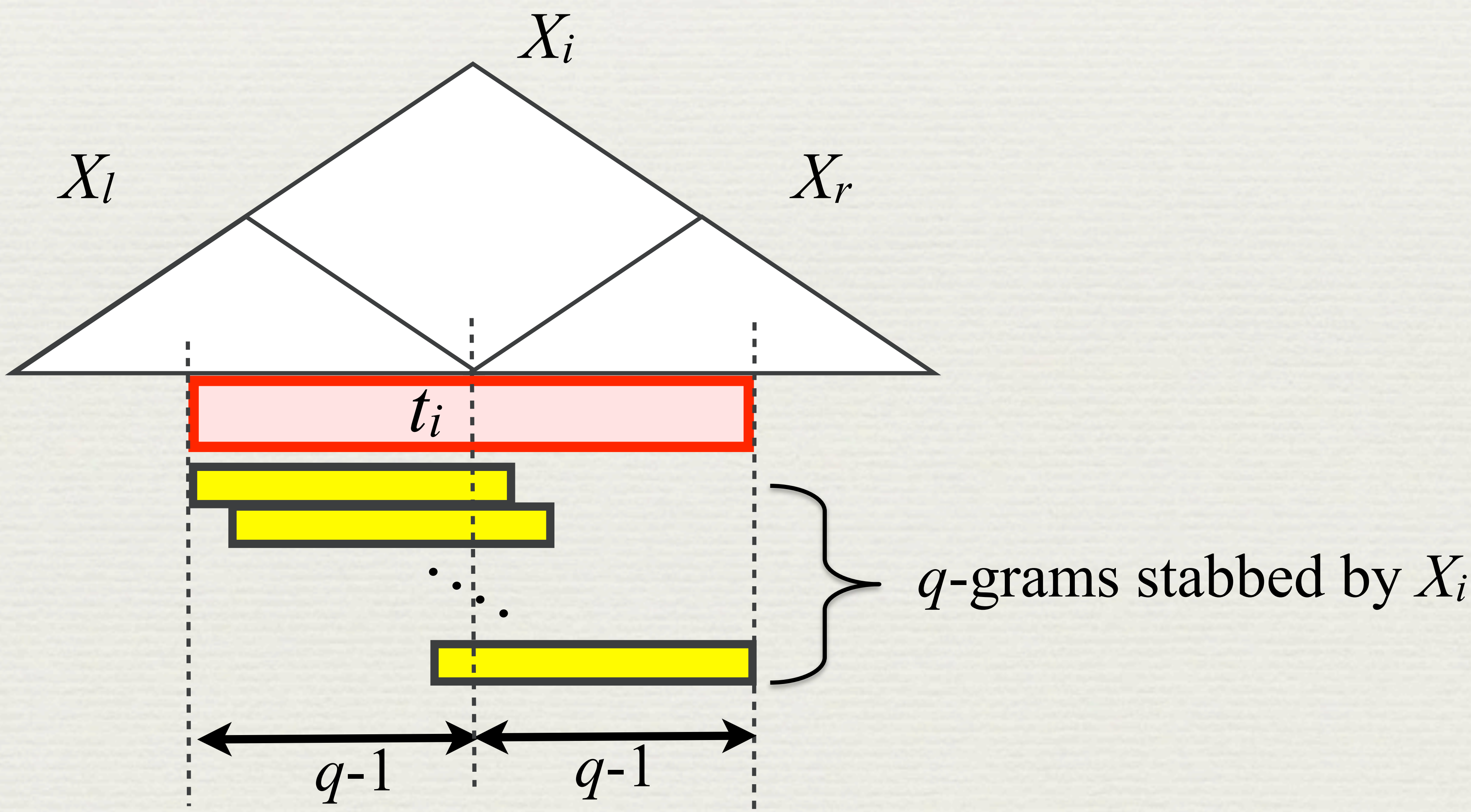
Computing $Freq^{串}(X_i, P)$



$O(qn)$ Algorithm in 2011

Computing $Freq^{串}(X_i, P)$

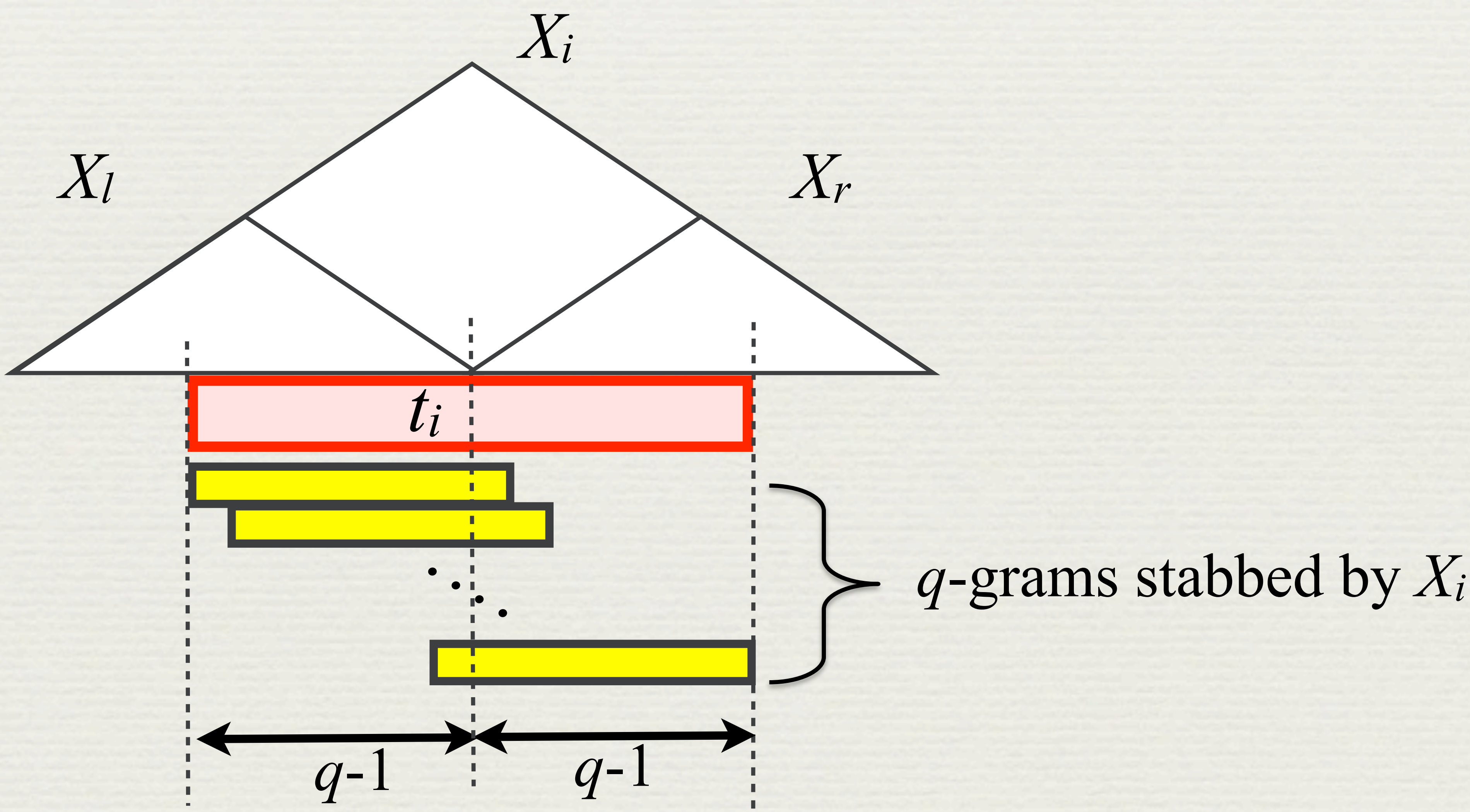
X_i stabs $P \Leftrightarrow$ P starts in X_l and ends in X_r



Computing $Freq^{串}(X_i, P)$

Observation

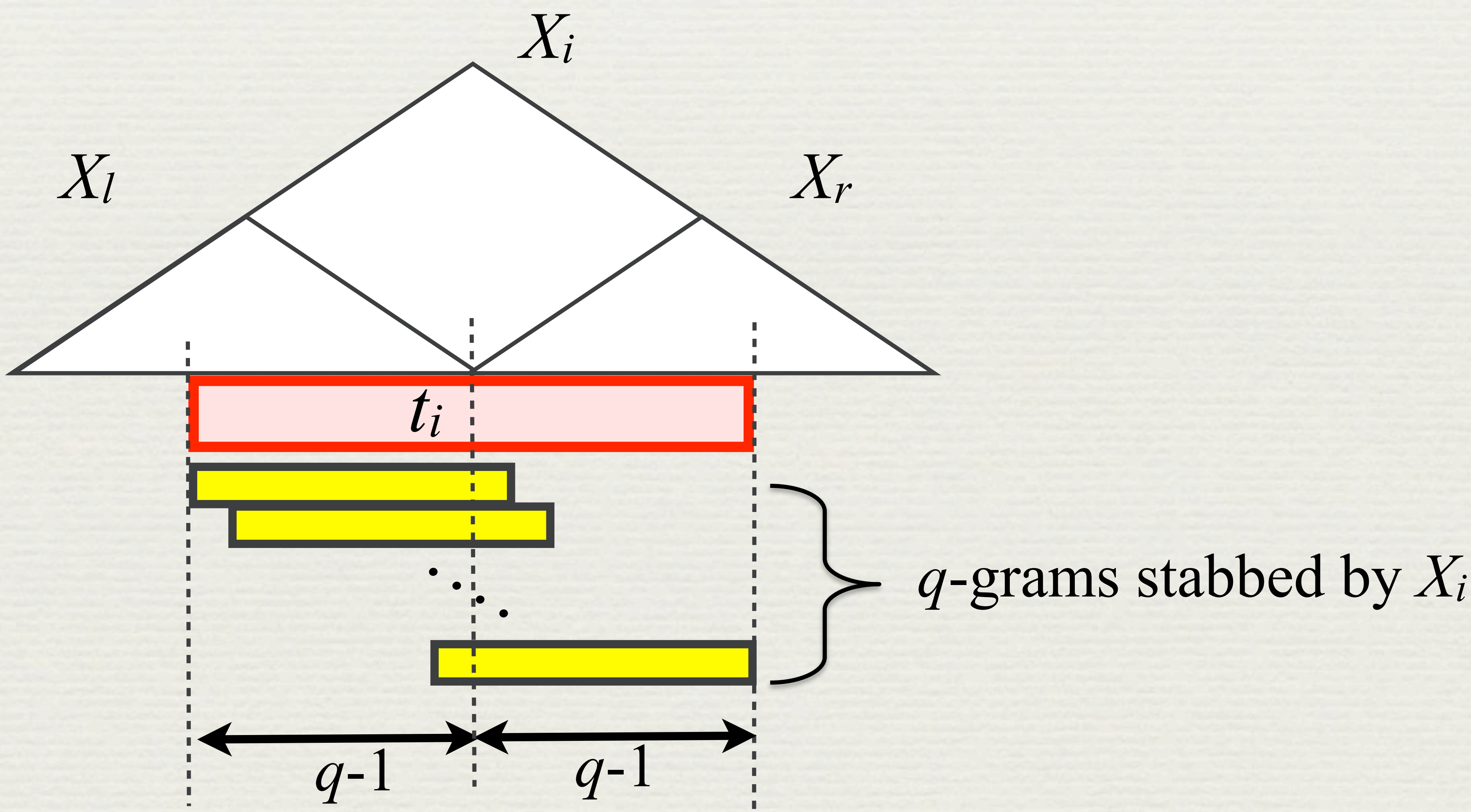
For any $P \in \Sigma^q$, $Freq^{串}(X_i, P) = Freq(t_i, P)$



Computing $Freq^{串}(X_i, P)$ by $Freq(t_i, P)$

Lemma

$$Freq(T,P) = \sum_{i=1}^{n} Freq(t_i,P) \cdot vOcc(X_i)$$



Computing frequencies by $Freq(t_i, P)$ and $vOcc(X_i)$

$O(n)$ time and space in total

Lemma

$$Freq(T,P) = \sum_{i=1}^{n} Freq(\boxed{t_i}, P) \cdot \boxed{vOcc(X_i)}$$

$O(qn)$ time and space in total

Computing frequencies by $Freq(t_i, P)$ and $vOcc(X_i)$

$O(n)$ time and space in total

Lemma

$$Freq(T,P) = \sum_{i=1}^{n} Freq(\boxed{t_i}, P) \cdot \boxed{vOcc(X_i)}$$

$O(qn)$ time and space in total

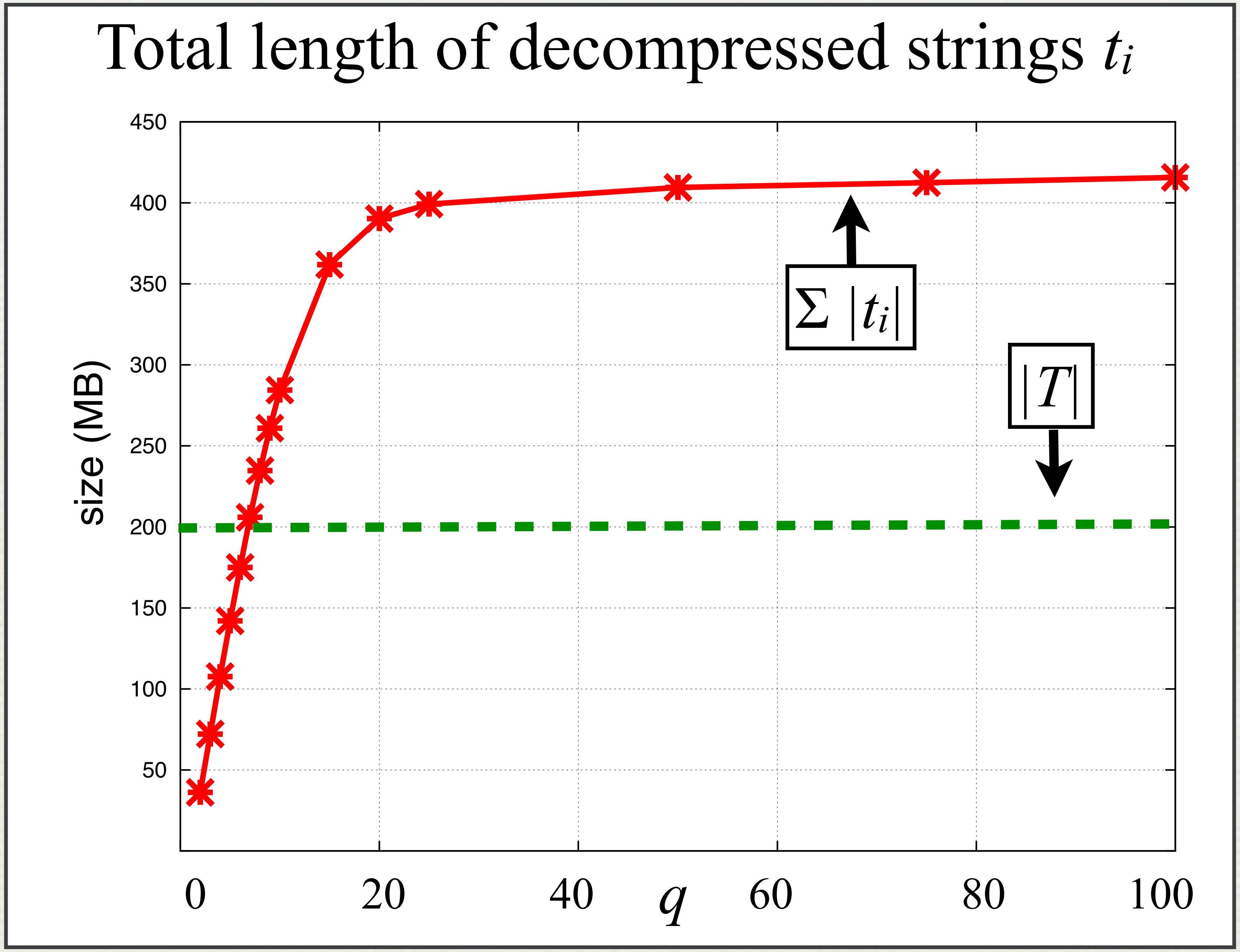
Theorem

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

Sketch of proof:

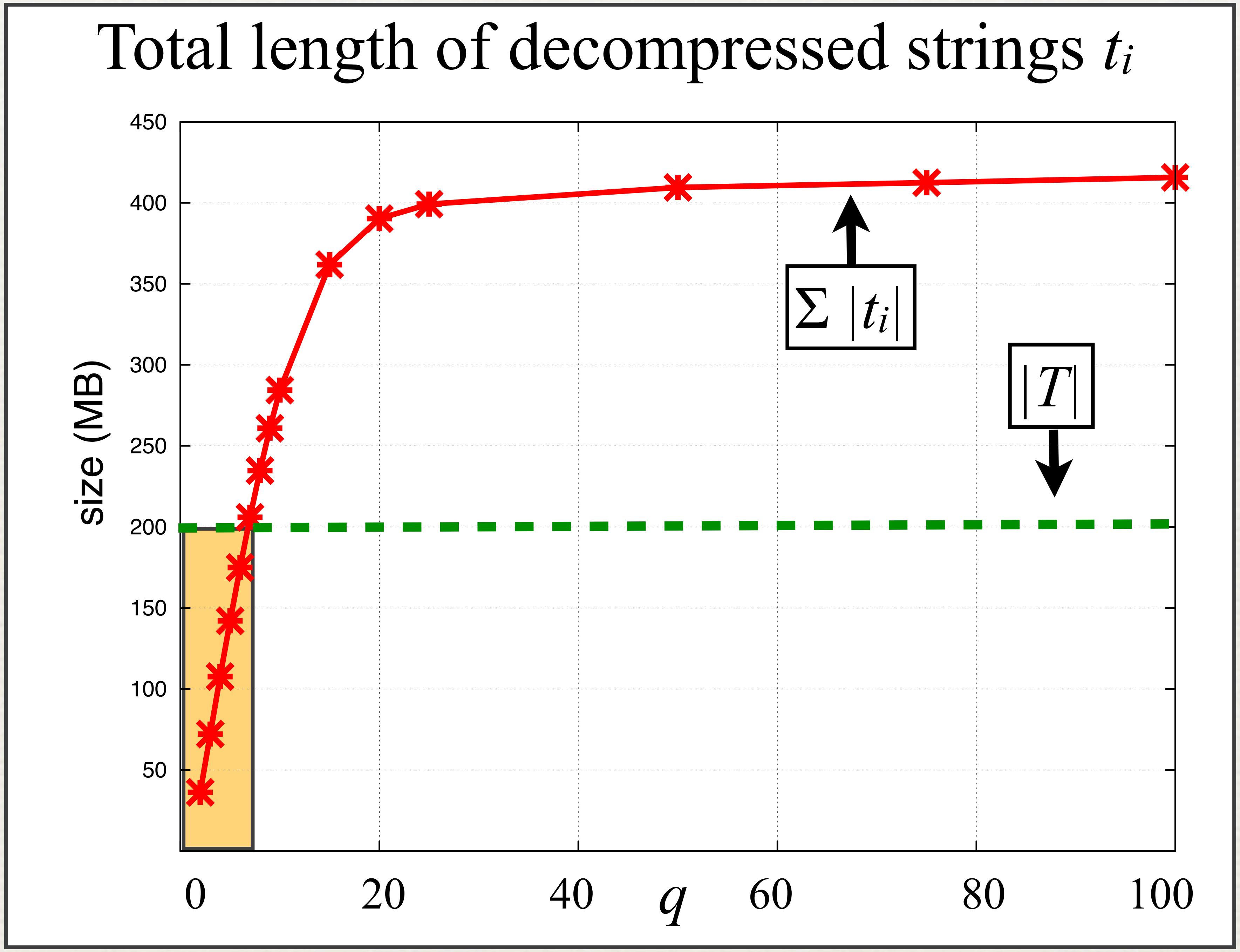
Using the suffix array of the concatenation of all t_i's, we can compute all q-gram frequencies in $O(qn)$ time and space.

Efficiency & Inefficiency of $O(qn)$ algorithm



ENGLISH data of 200MB from pizza & chili corpus

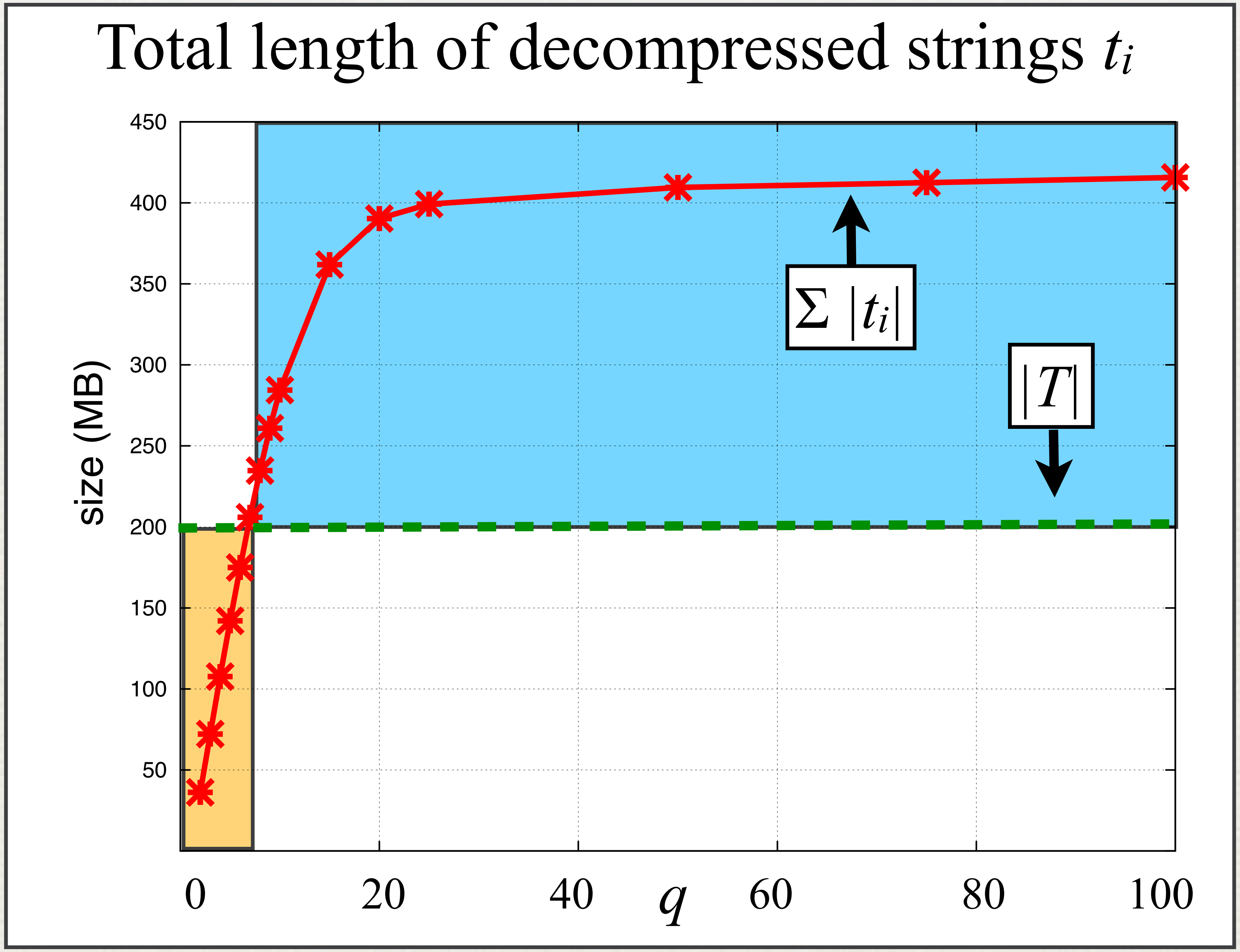
Efficiency & Inefficiency of $O(qn)$ algorithm



- when q is **small**, the algorithm runs **faster**

ENGLISH data of 200MB from pizza & chili corpus

Efficiency & Inefficiency of $O(qn)$ algorithm



- when q is **large**, the algorithm runs **slower**
- when q is **small**, the algorithm runs **faster**

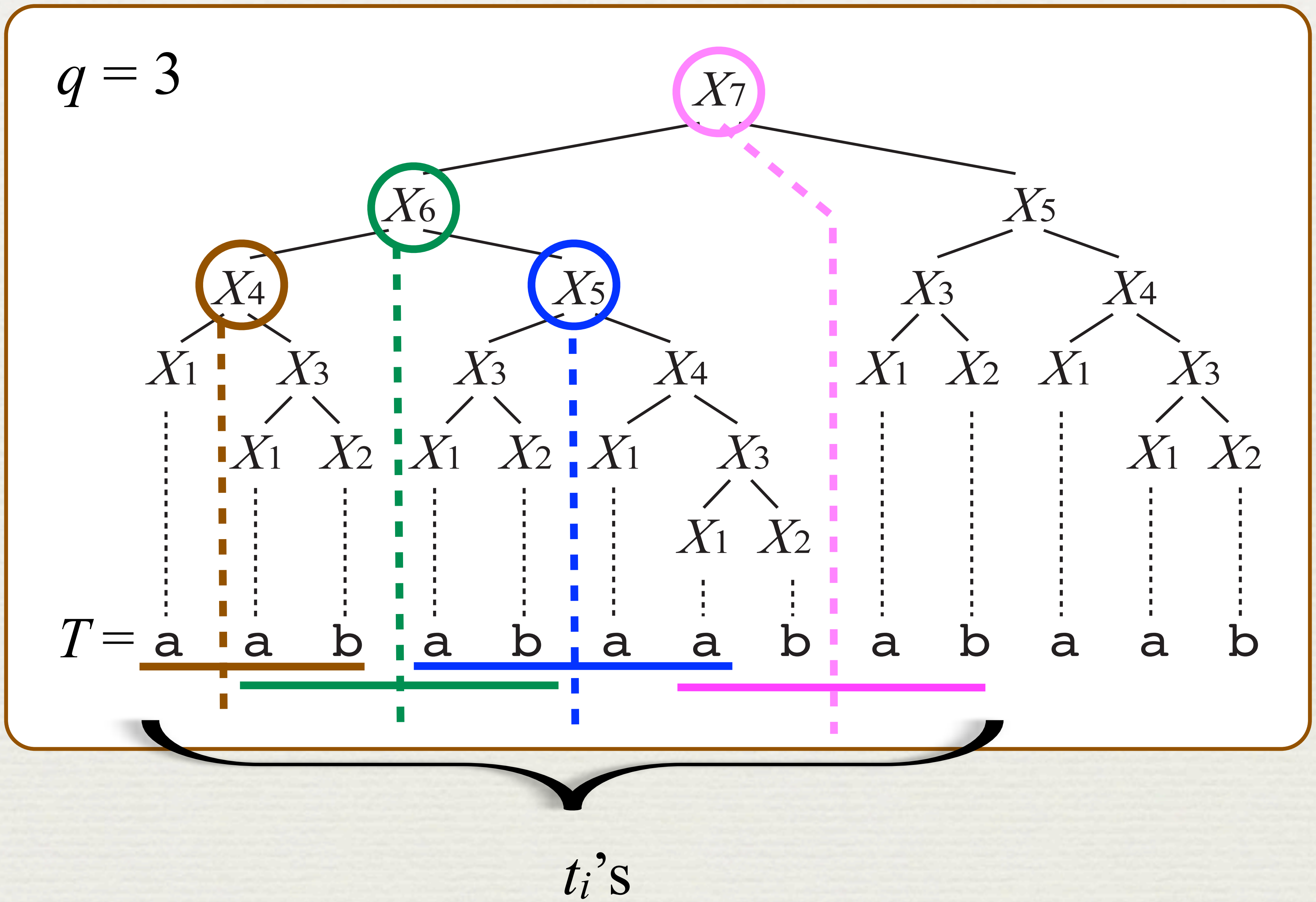
ENGLISH data of 200MB from pizza & chili corpus

New algorithm

New Algorithm

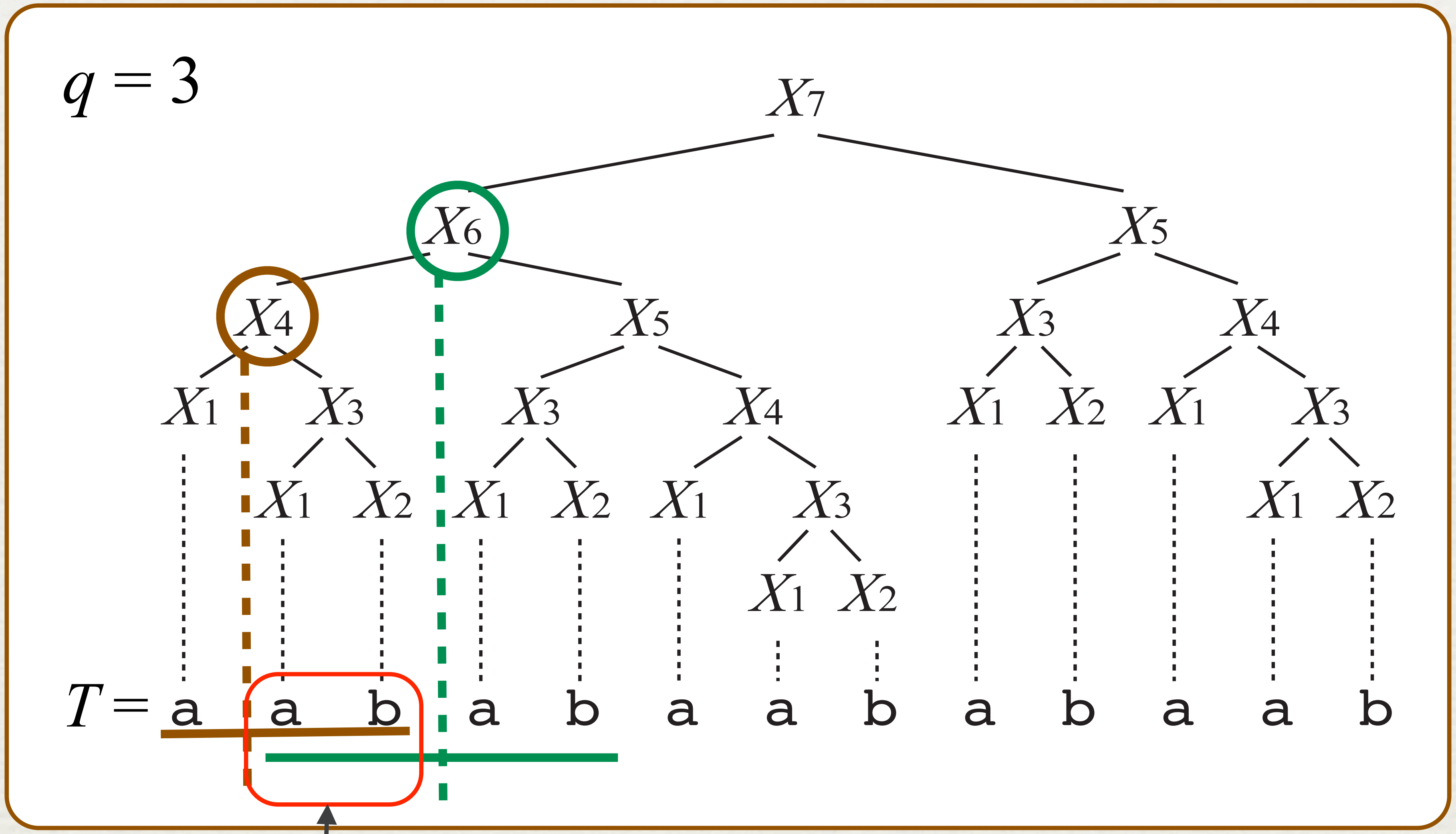
Inefficiency of $O(qn)$ algorithm

- Total length of decompressed strings t_i can be larger than $|T|$



Inefficiency of $O(qn)$ algorithm

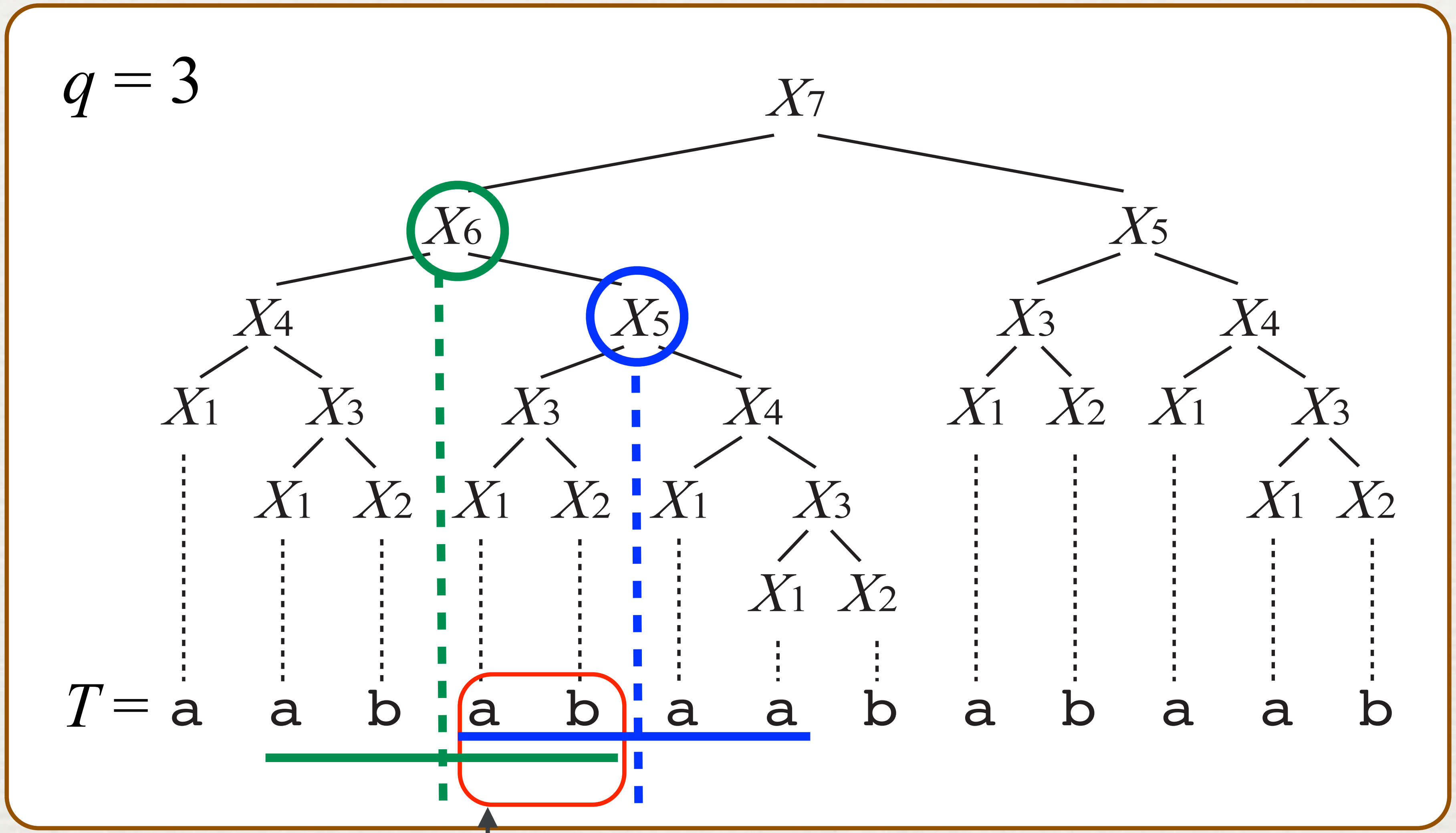
- There are overlaps between partially decompressed strings t_i



t_4 and t_6 overlap with "ab"

Inefficiency of $O(qn)$ algorithm

- There are overlaps between partially decompressed strings t_i

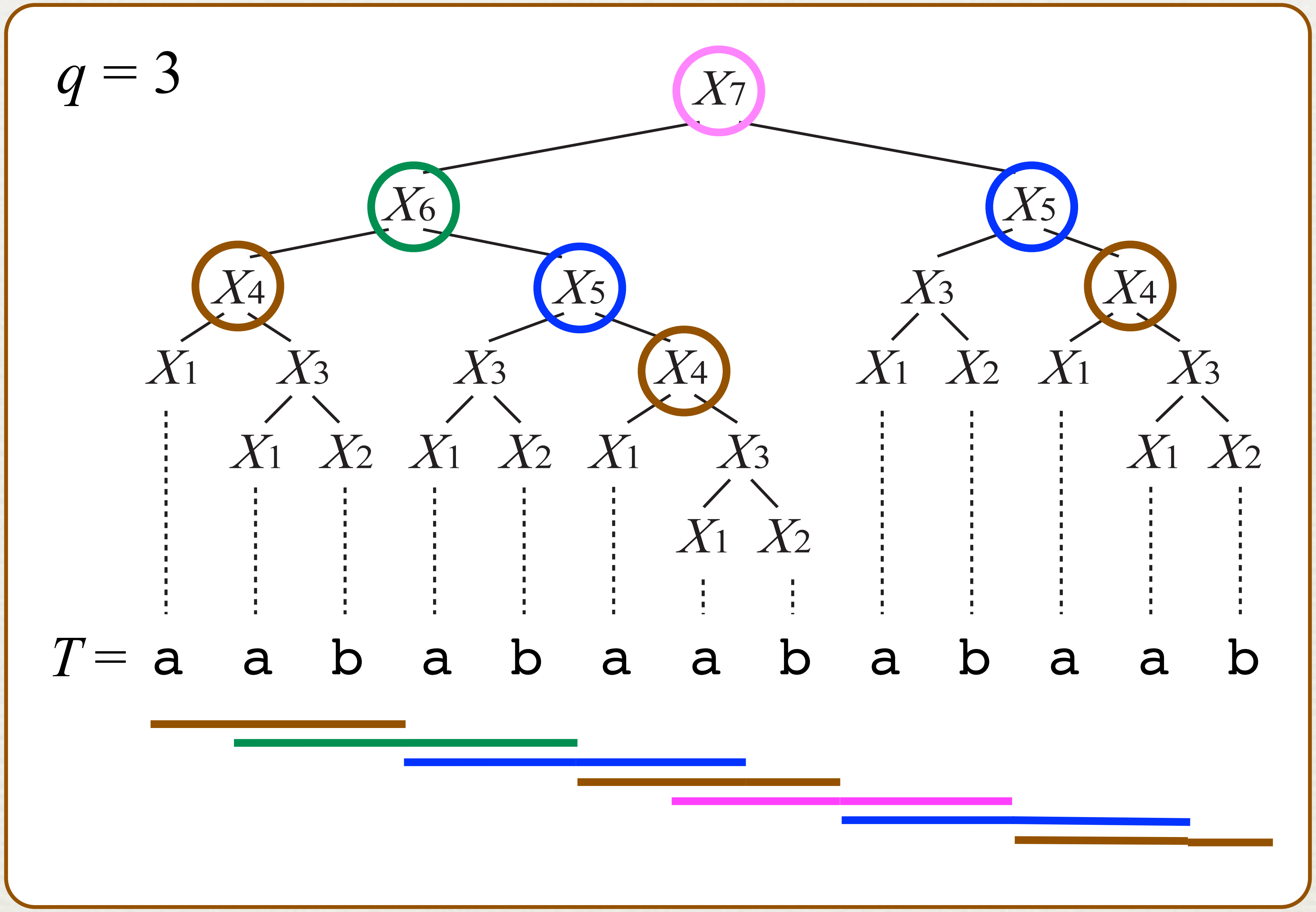


t_6 and t_5 overlap with "ab"

New Algorithm

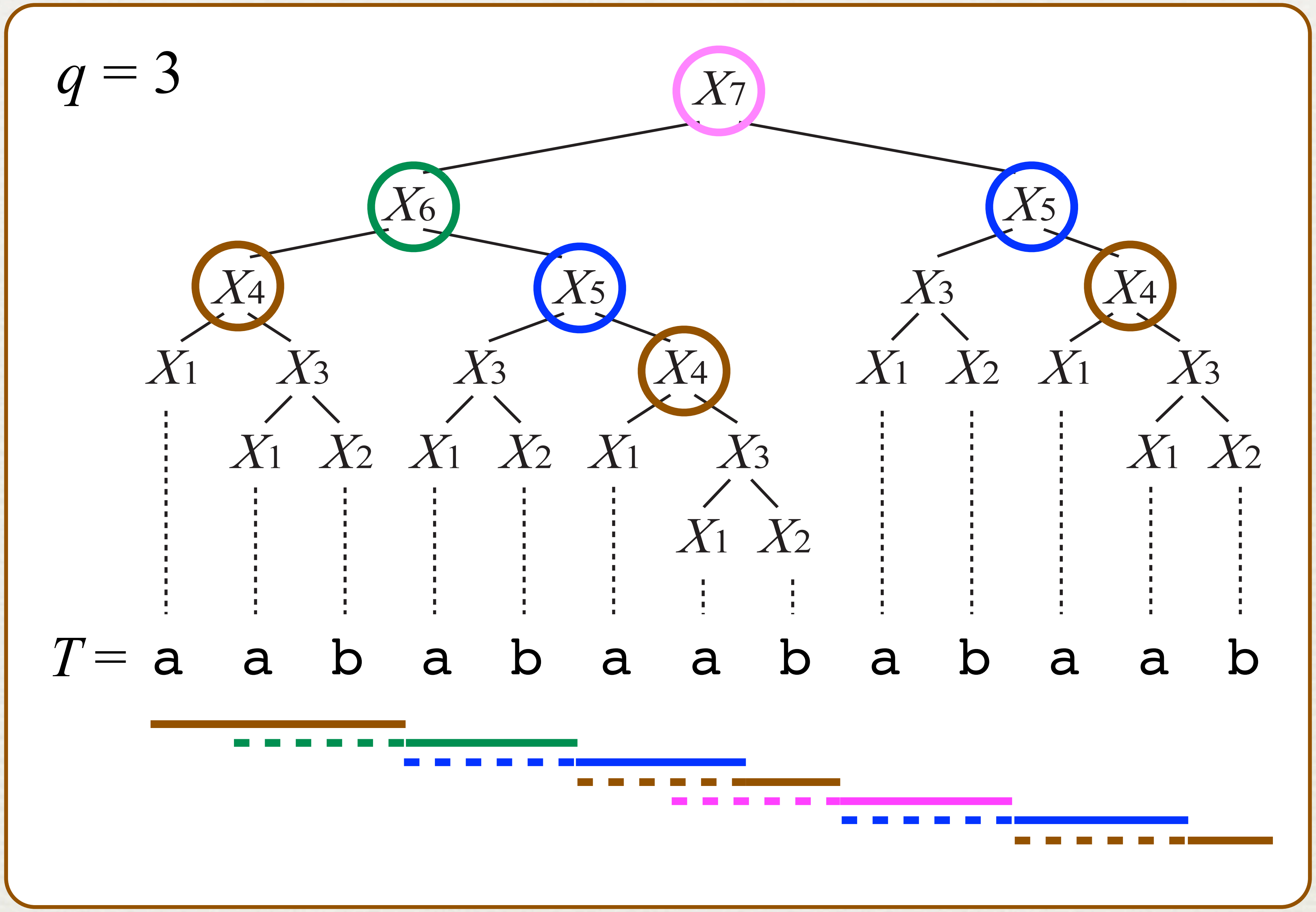
Identifying the redundancies

- Consider all partially decompressed strings t_i in derivation tree



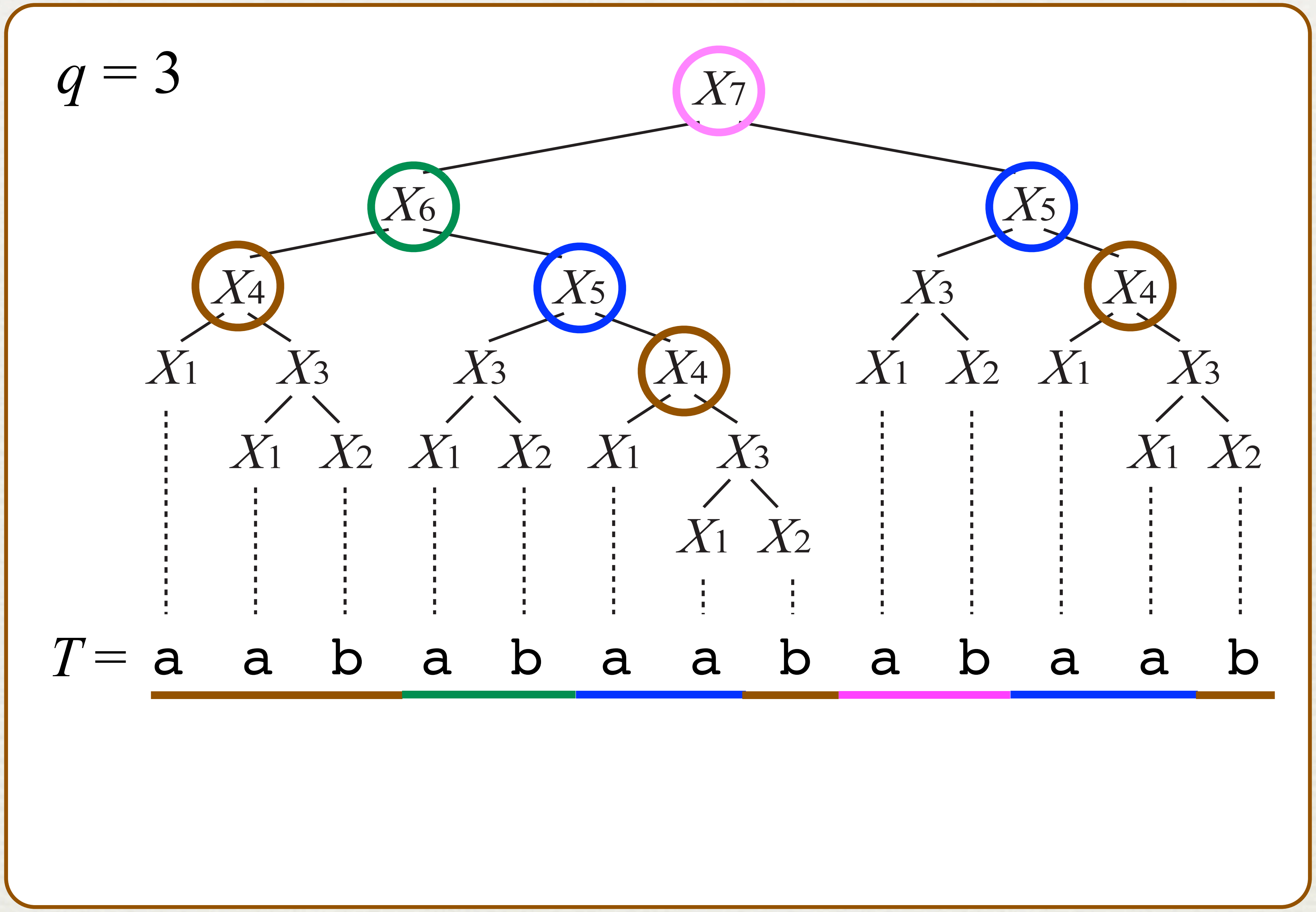
Removing overlaps of neighboring t_i's

- Eliminate length-(q-1) prefix of all t_i's except for leftmost one



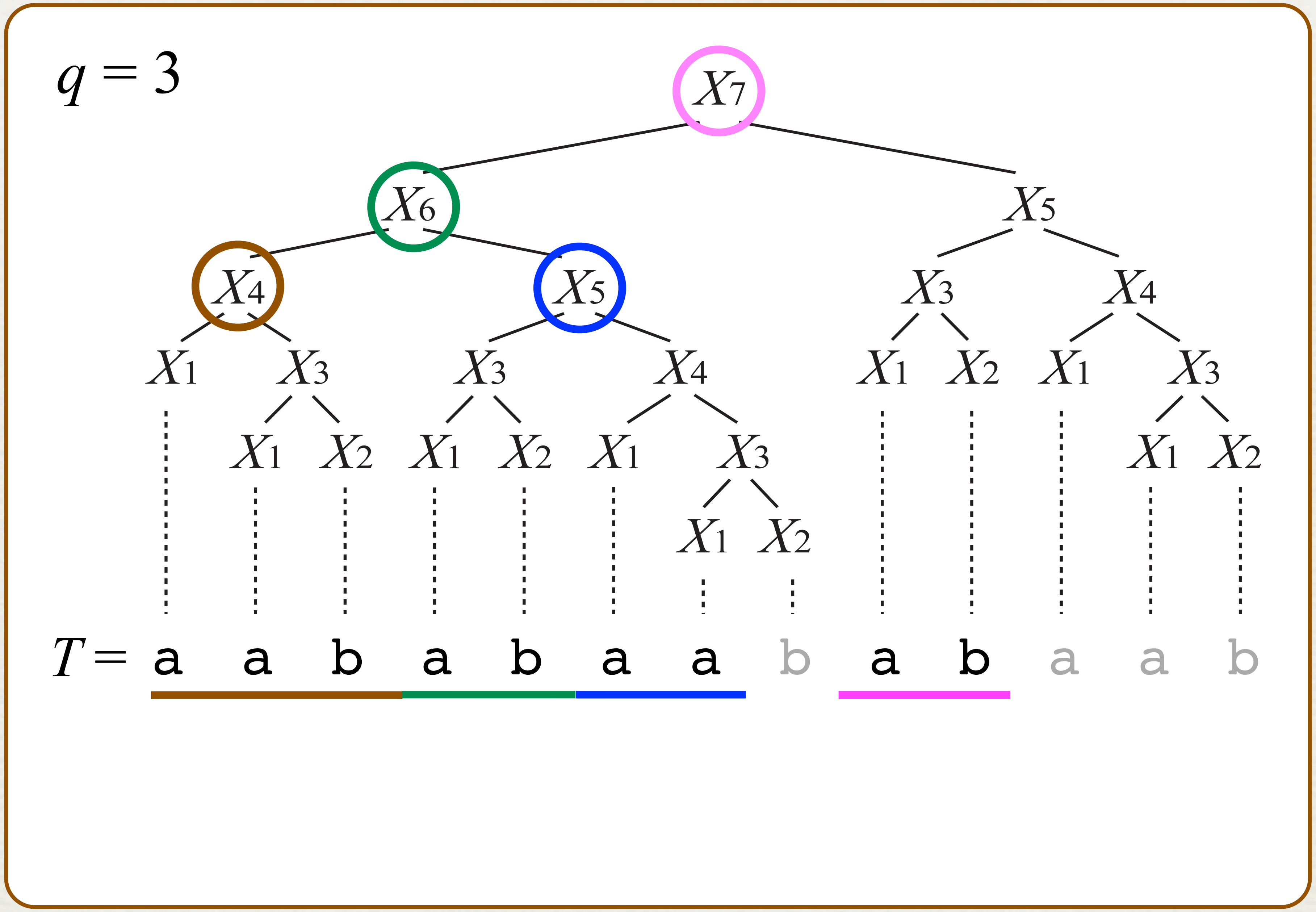
Removing overlaps of neighboring t_i's

- Concatenation of remaining strings equals to T



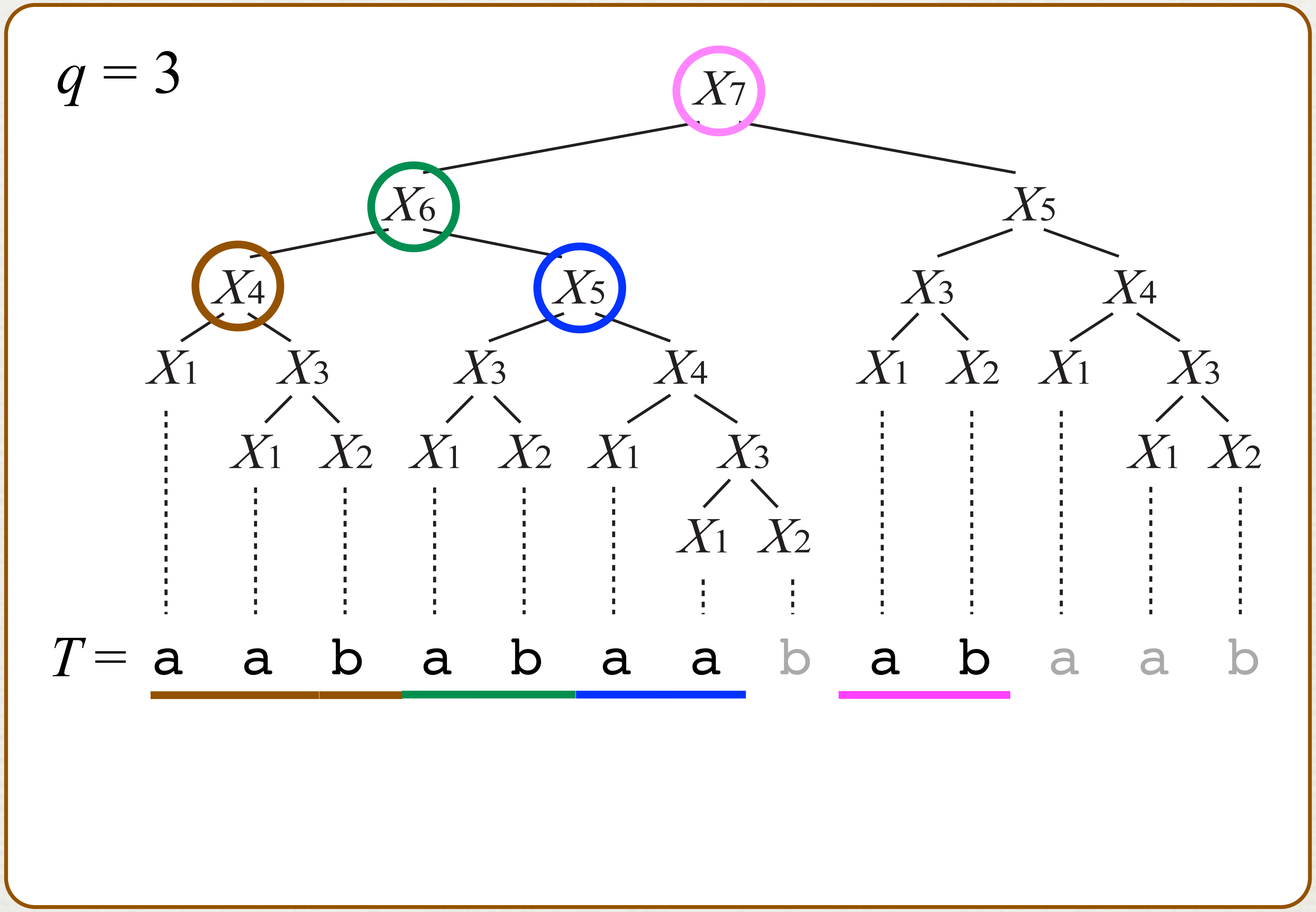
Removing duplicate t_i's

- For all partially eliminated t_i, remove all but first occurrence



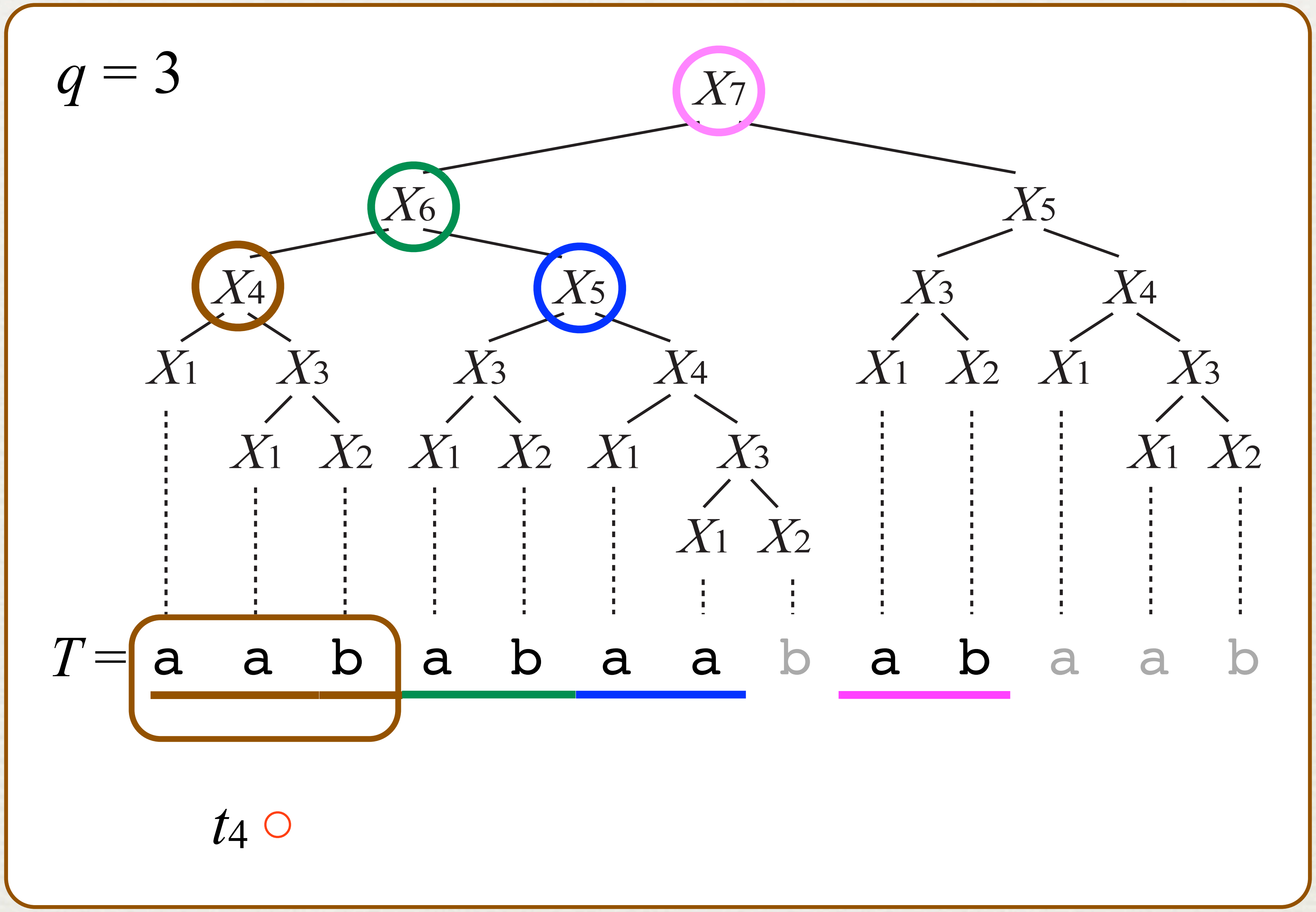
What we have left

- Compact representation of all t_i's



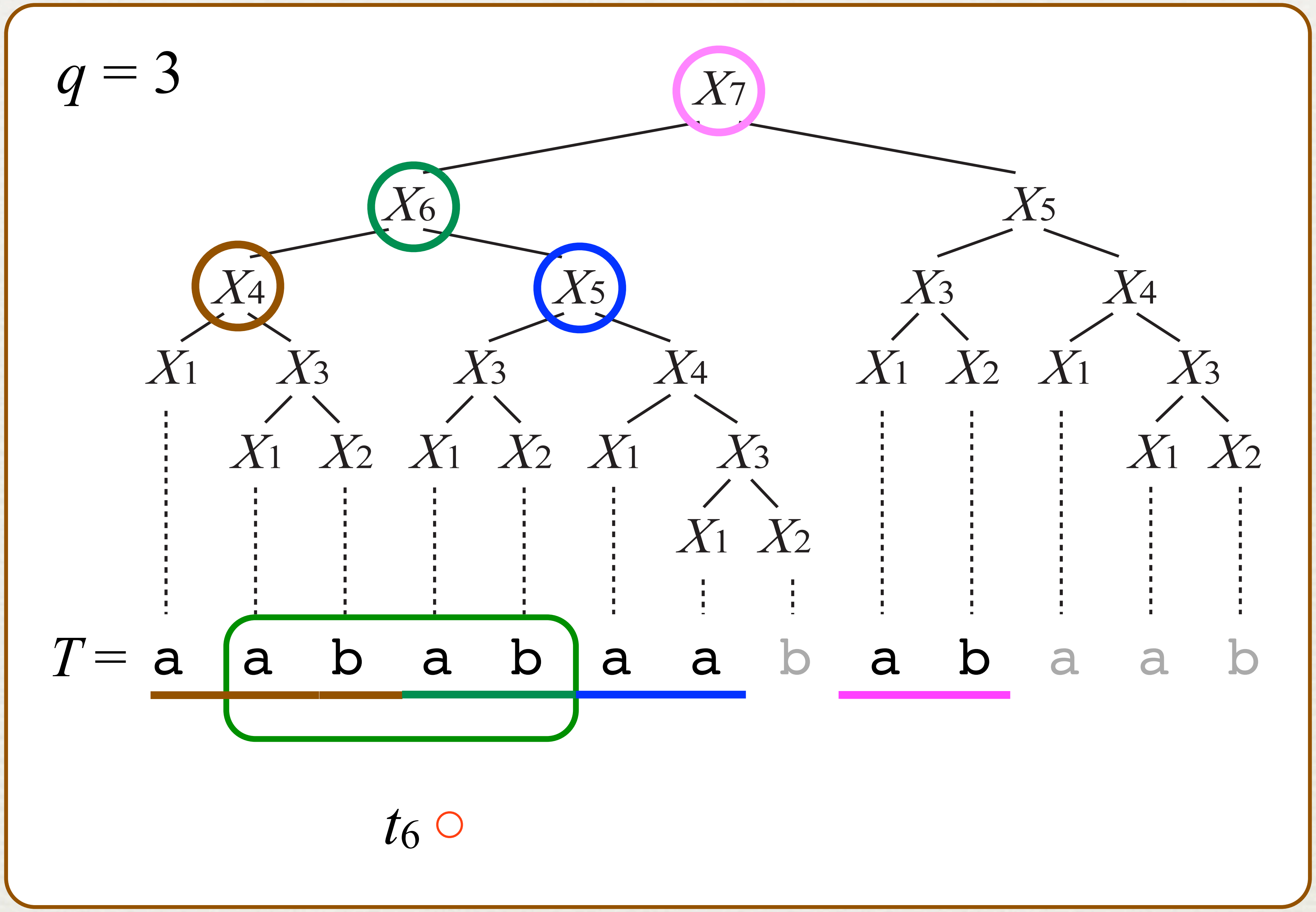
What we have left

- Compact representation of all t_i's



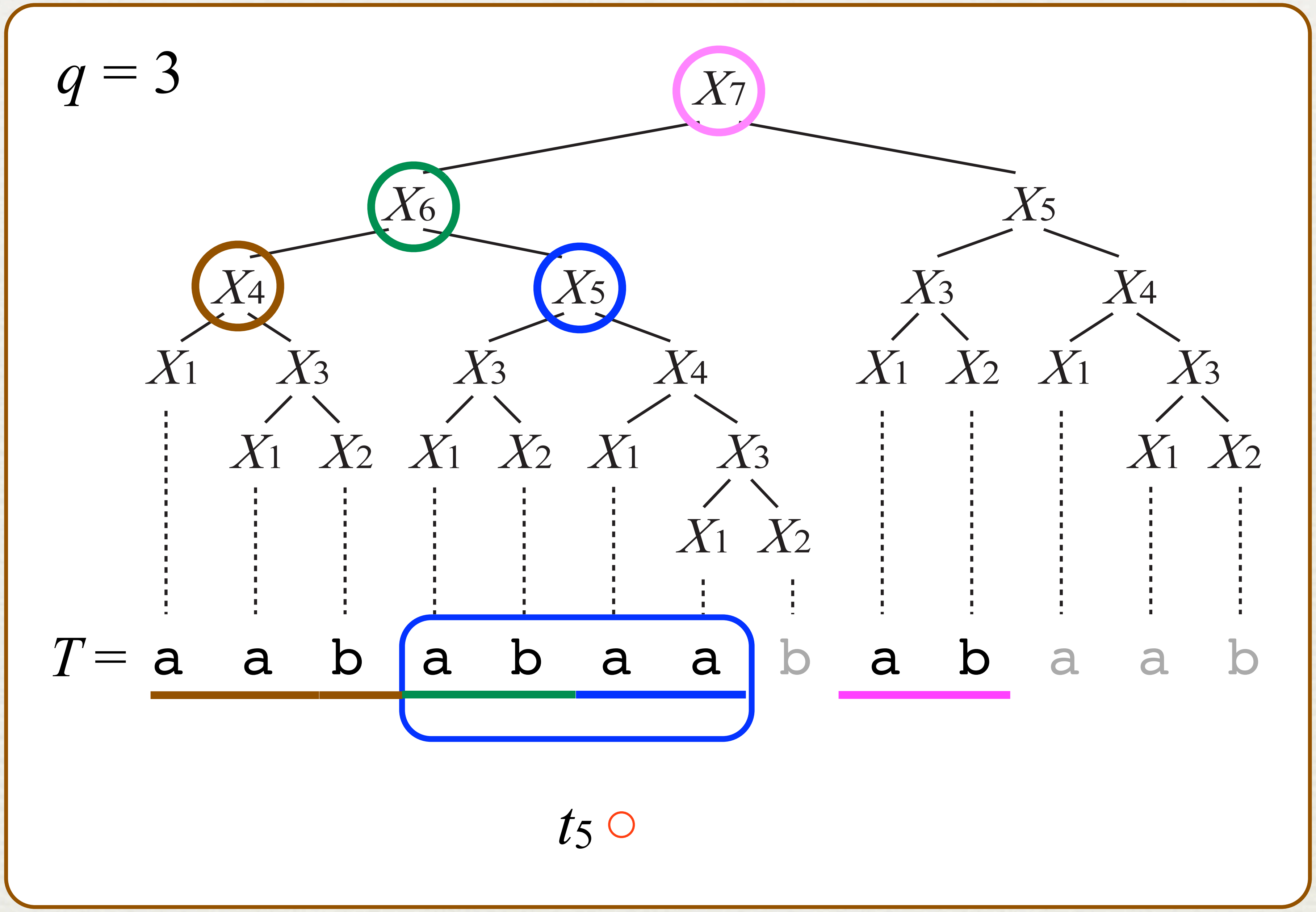
What we have left

- Compact representation of all t_i's



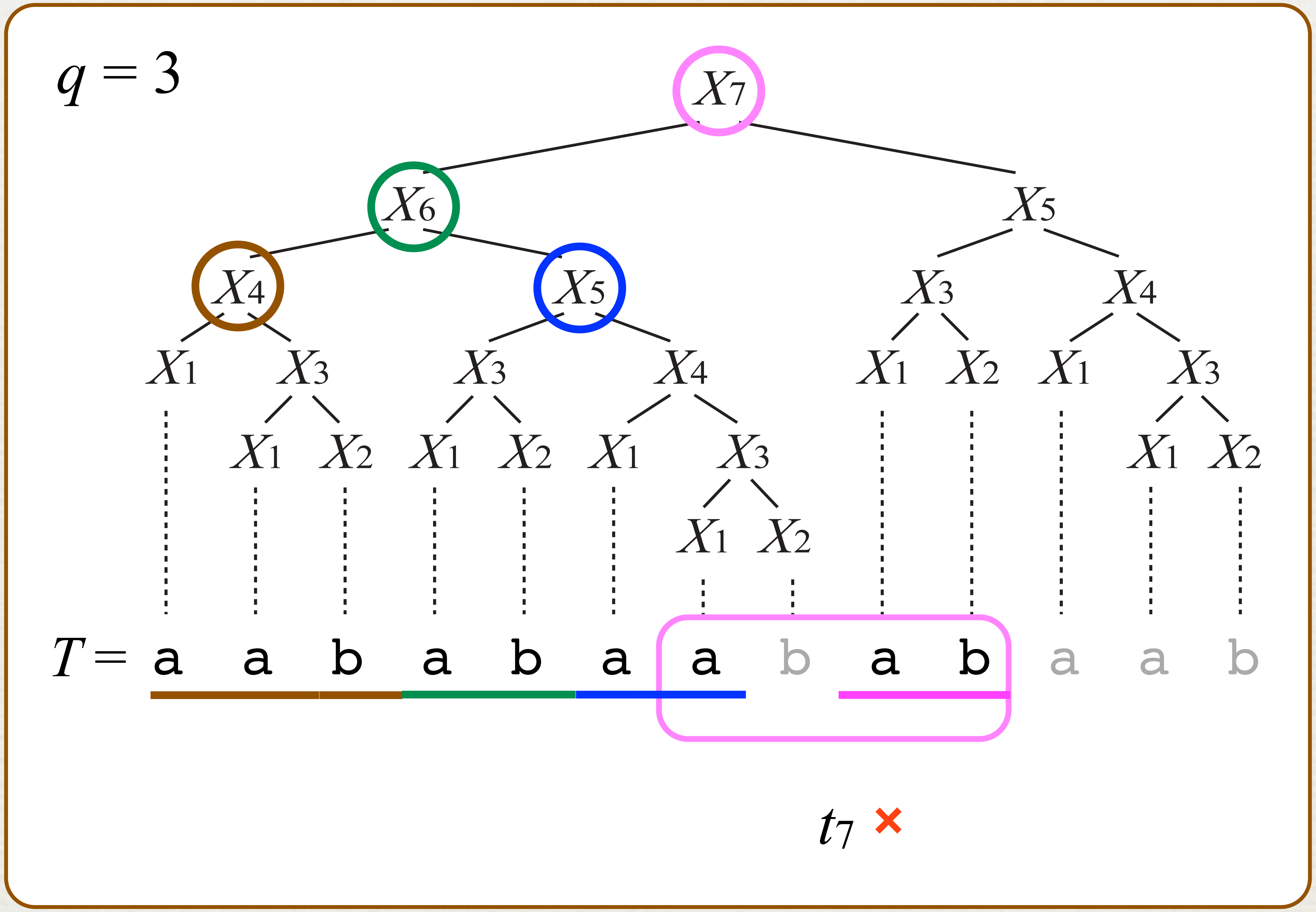
What we have left

- Compact representation of all t_i's



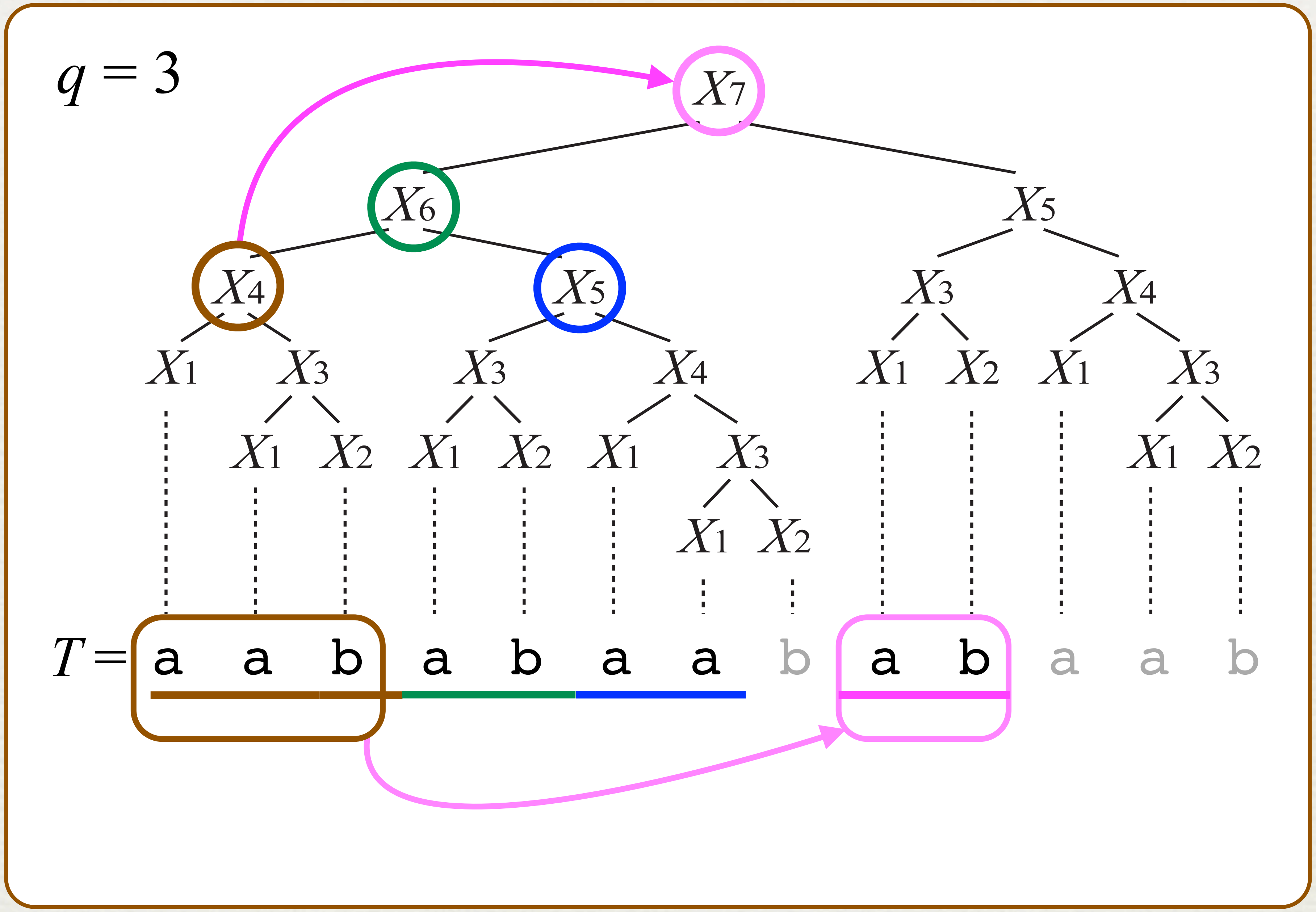
What we have left

- Compact representation of all t_i's



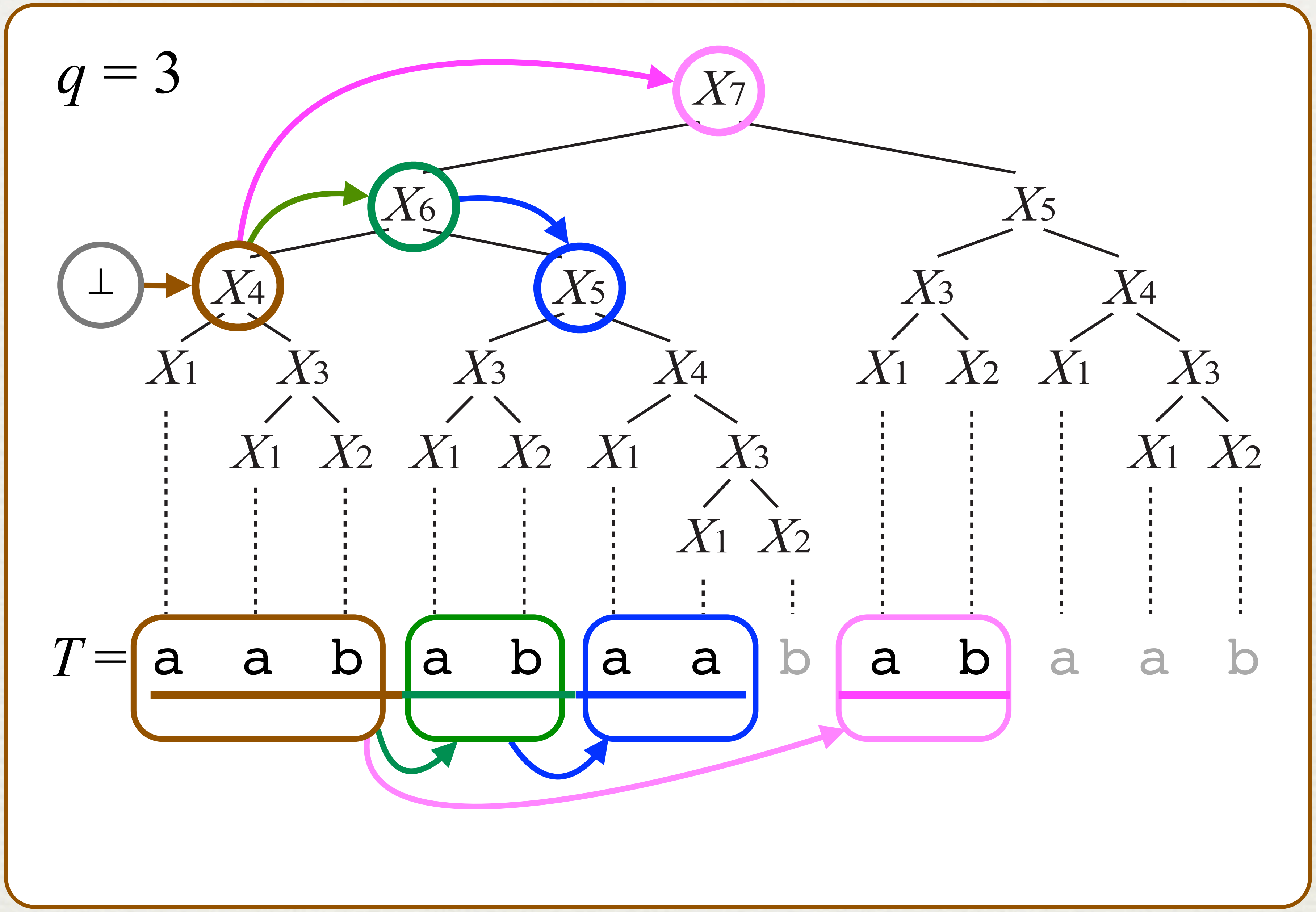
Neighbor tree

- Edge from X_i to X_j $\Leftrightarrow$ t_i and t_j are neighboring



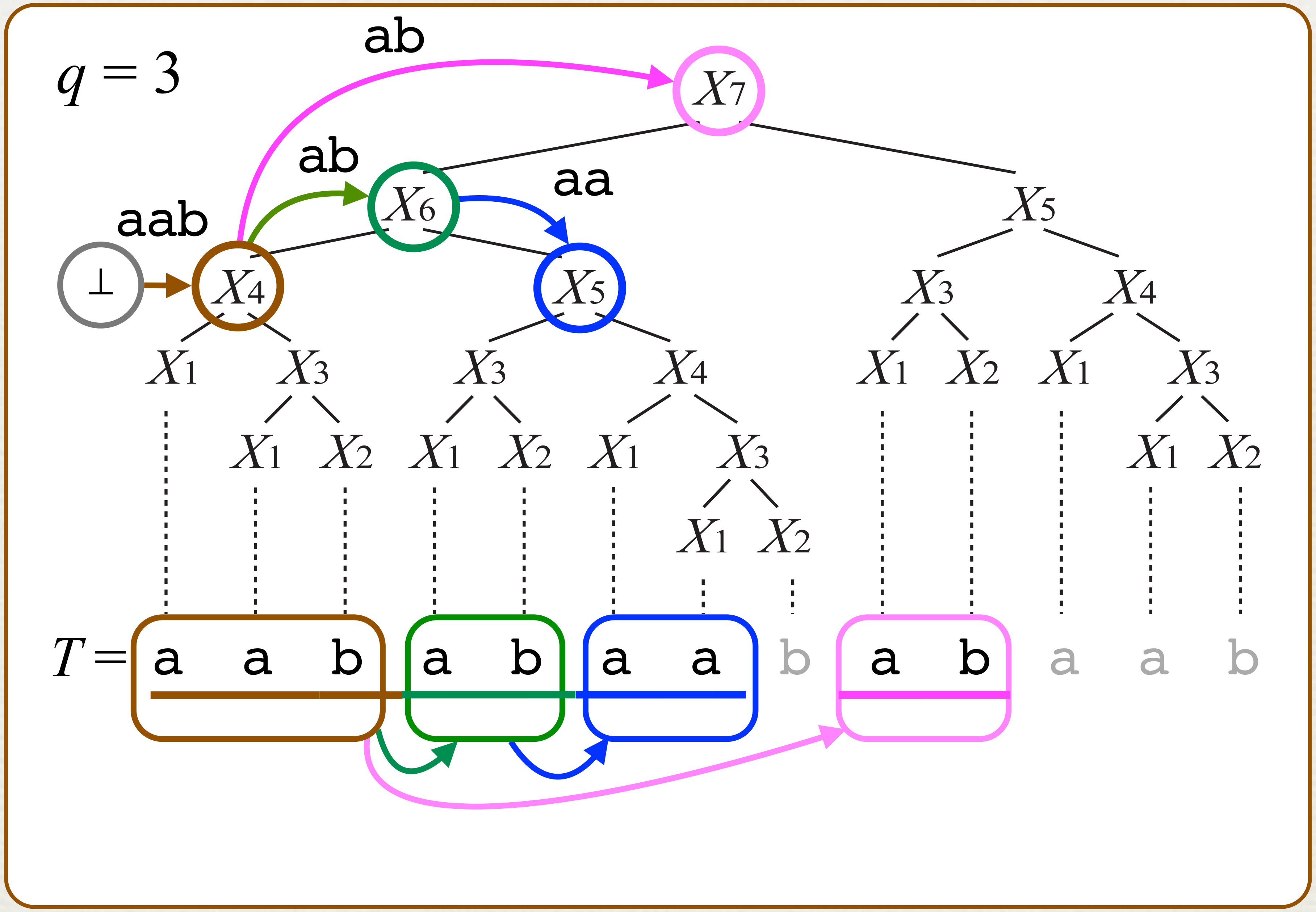
Neighbor tree

- Edge from X_i to X_j $\Leftrightarrow$ t_i and t_j are neighboring



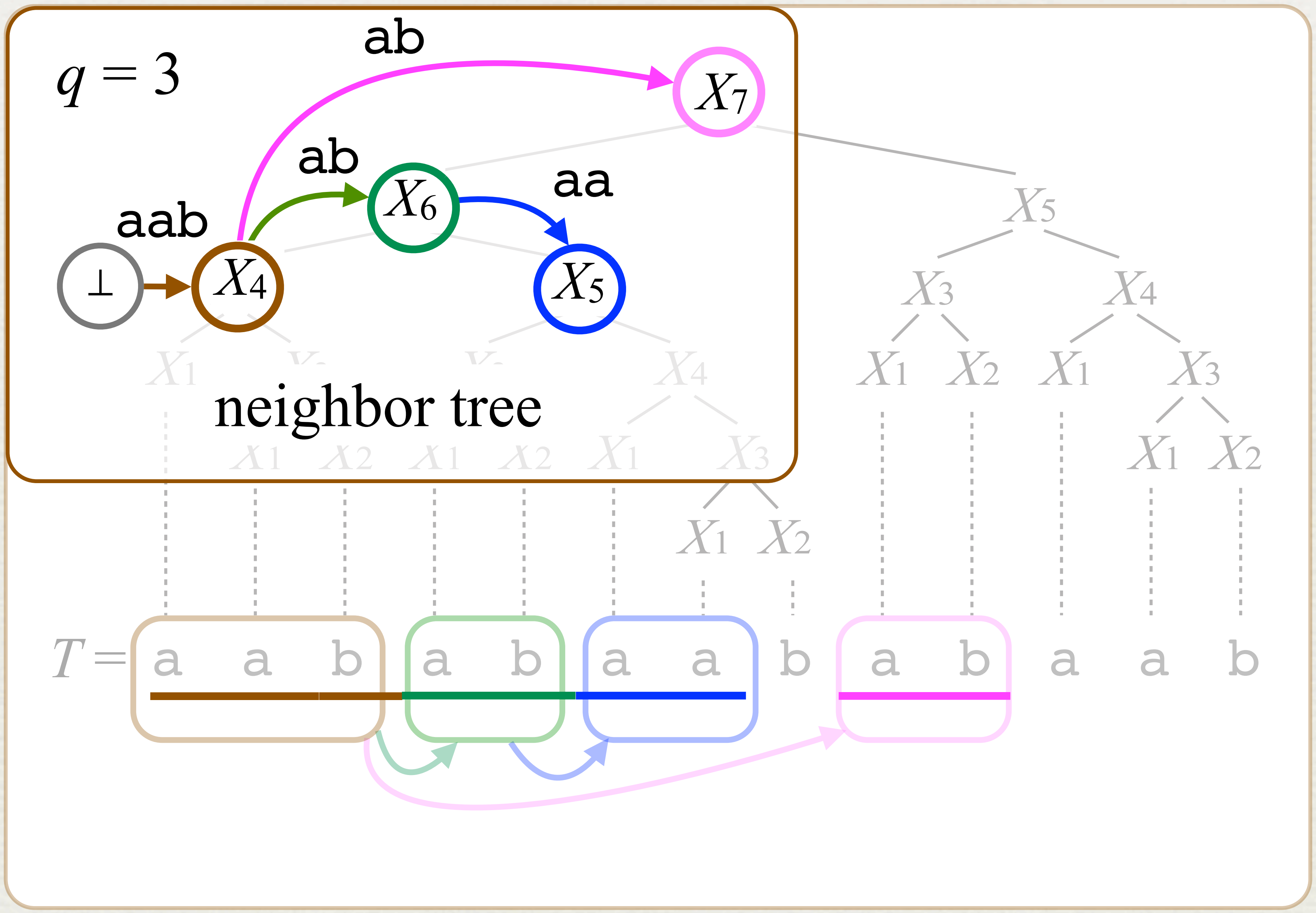
Neighbor tree

- Edge from X_i to X_j $\Leftrightarrow$ t_i and t_j are neighboring



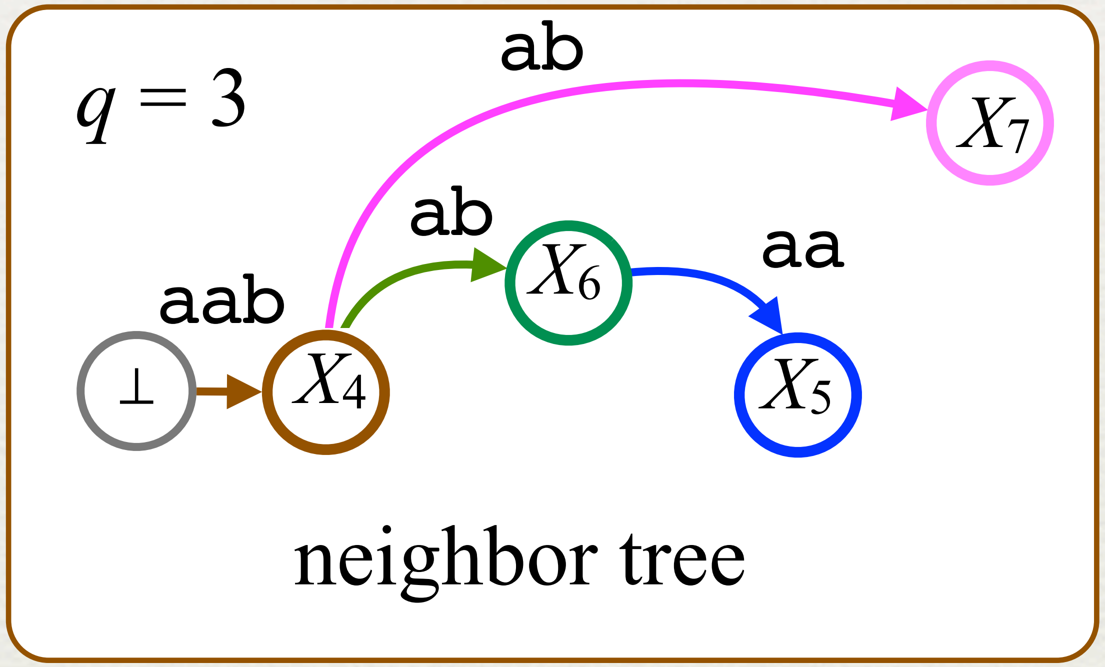
Neighbor tree

- Edge from X_i to X_j $\Leftrightarrow$ t_i and t_j are neighboring



Size of neighbor tree

- Edge from X_i to X_j $\Leftrightarrow$ t_i and t_j are neighboring



Lemma

The total length of edge labels in neighbor tree of G is

$(q\text{-}1) + \Sigma\ \{|t_i| - (q\text{-}1) \mid |X_i| \geqq q,\ i = 1, \ldots, n\}$

$= |T| - dup(q, D)$

where $dup(q, D) = \sum\{(vOcc(X_i) - 1) \cdot (|t_i| - (q-1)) \mid |X_i| \geq q, i = 1, \ldots, n\}$

Summary of Improved algorithm

Lemma

The neighbor tree from SLP D can be constructed in $O(\min\{qn, |T|\text{-}dup(q, D)\})$

Summary of Improved algorithm

Lemma

The neighbor tree from SLP D can be constructed in $O(\min\{qn, |T|\text{-}dup(q, D)\})$

Lemma [Shibuya, 2003]

The suffix tree for a trie can be constructed in time linear in its size

Summary of Improved algorithm

Lemma

The neighbor tree from SLP D can be constructed in $O(\min\{qn, |T|\text{-}dup(q, D)\})$

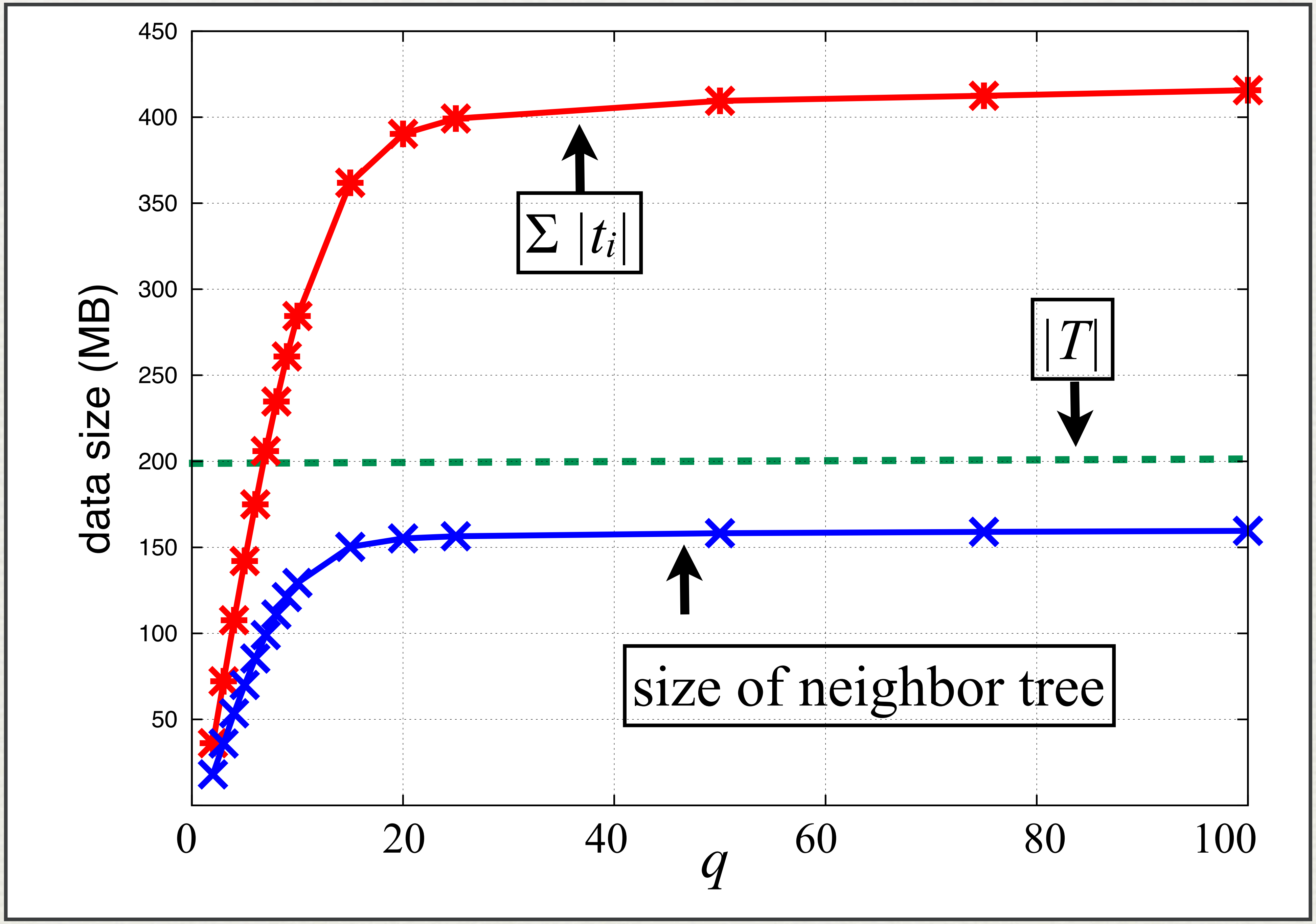
Lemma [Shibuya, 2003]

The suffix tree for a trie can be constructed in time linear in its size

Theorem

The q-gram frequencies problem on a SLP D of size n, representing string T can be solved in $O(\min\{qn, |T|\text{-}dup(q, D)\})$ time and space.

Preliminary Experiment (ENGLISH 200MB)
size of neighbor tree and $\Sigma|t_i|$



Example of ENGLISH data of 200MB from pizza & chili corpus

Summary

	Uncompressed String	SLP (SPIRE 2011)	SLP (This work)
q-gram Freq	$O(\|T\|) = O(2^n)$ time and space	$O(qn)$ time and space	$O(\min\{qn, \|T\|\text{-}dup(q, D)\})$ time and space

Future work:

Other applications of neighbor tree
(e.g. one paper accepted to SPIRE 2012)