Speeding up *q*-gram mining on grammar based compressed text

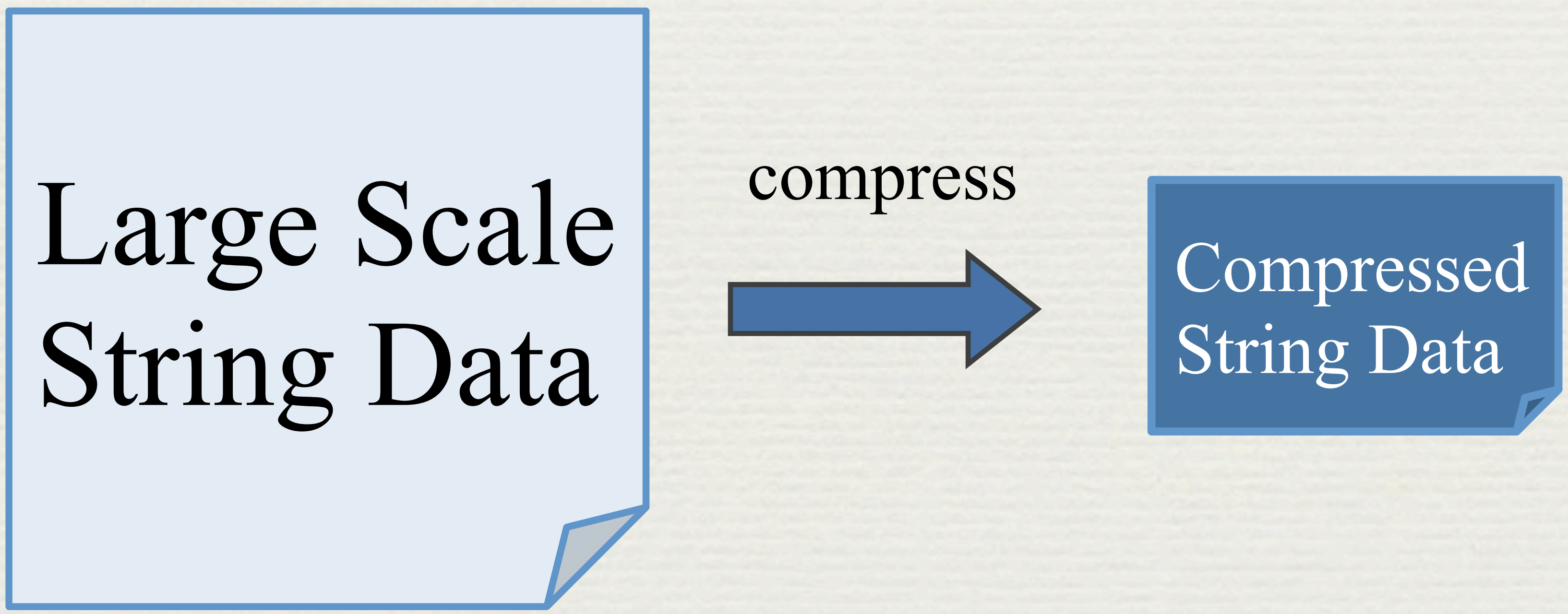
Kyushu University

◯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.

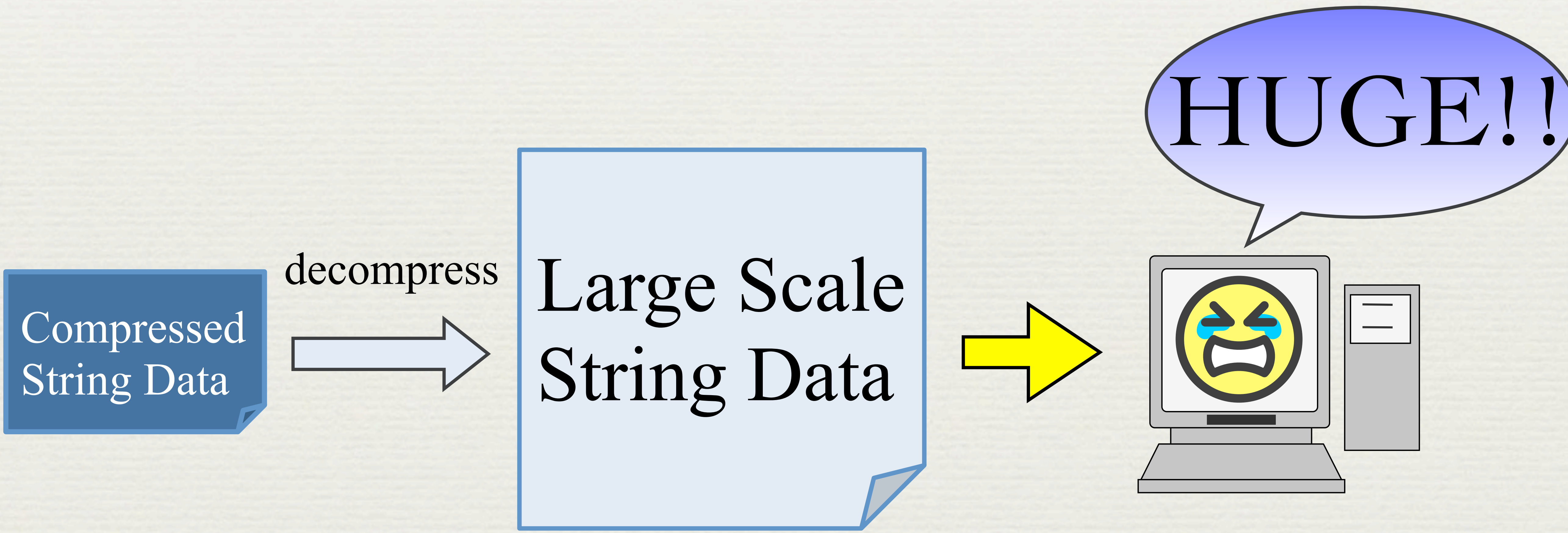
Compressed String Data

Background: Processing large scale string data

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

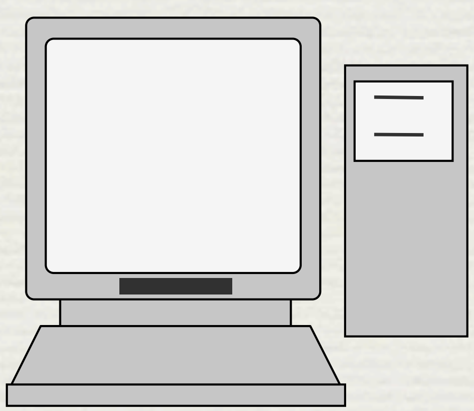
Background: Processing large scale string data

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



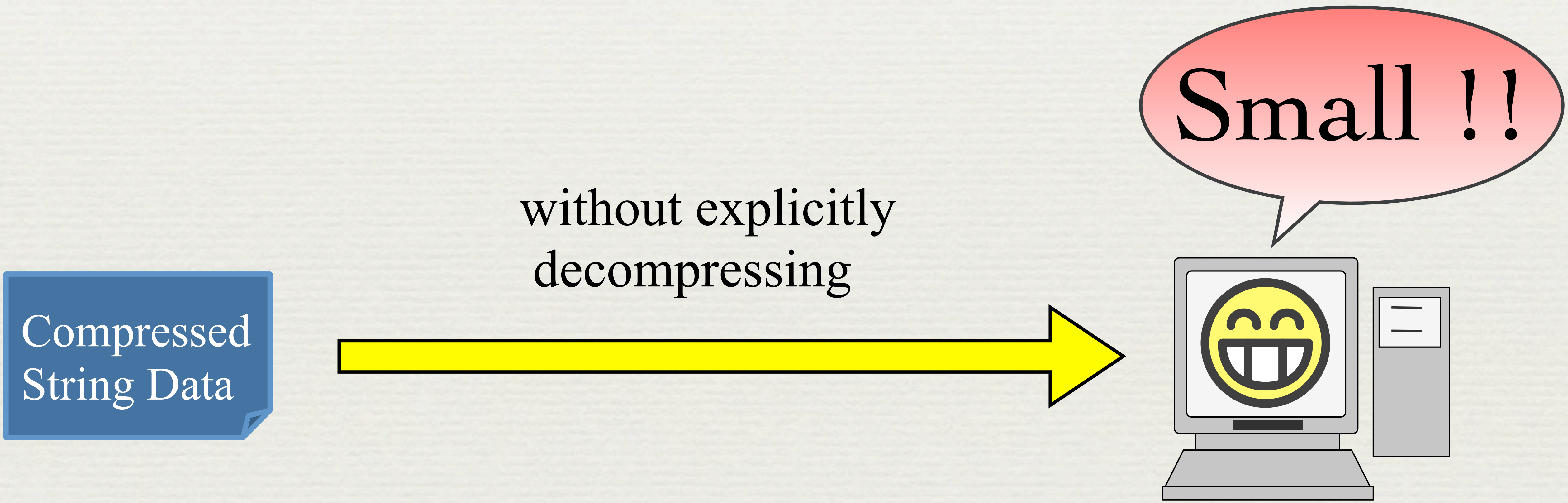
Background: Processing large scale string data

- 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
```

$Freq(T,$ "aba"$)$ $= 3$

13年2月21日木曜日

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
```

$Freq(T,$ "aba"$) = 3$

$Freq(T,$ "baa"$) = 2$

$Freq(T,$ "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

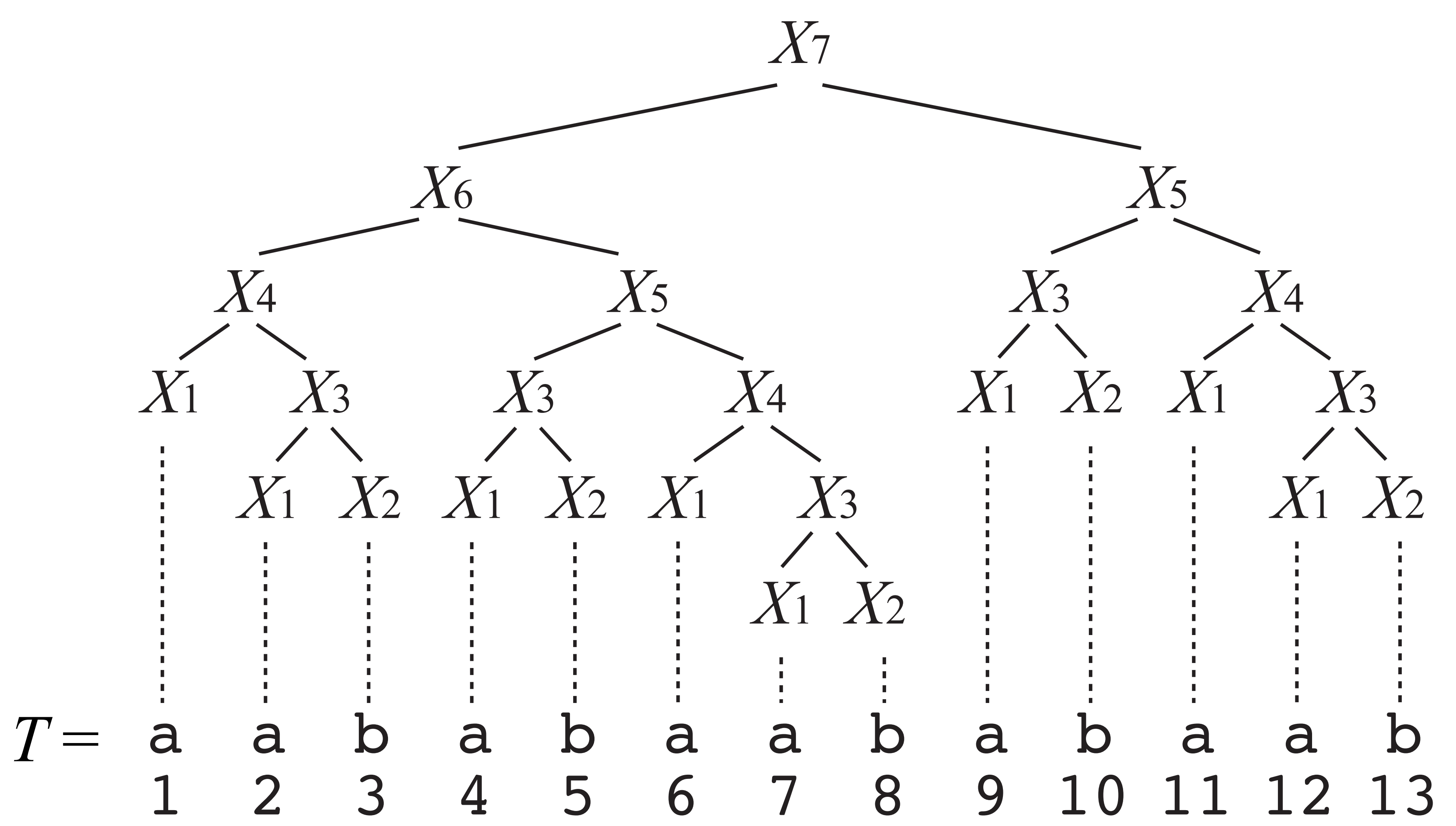
Example of SLP

SLP: D

$$X_1 = a$$
$$X_2 = b$$
$$X_3 = X_1 X_2$$
$$X_4 = X_1 X_3$$
$$X_5 = X_3 X_4$$
$$X_6 = X_4 X_5$$
$$X_7 = X_6 X_5$$

$$n = |D| = 7$$

Derivation Tree of D



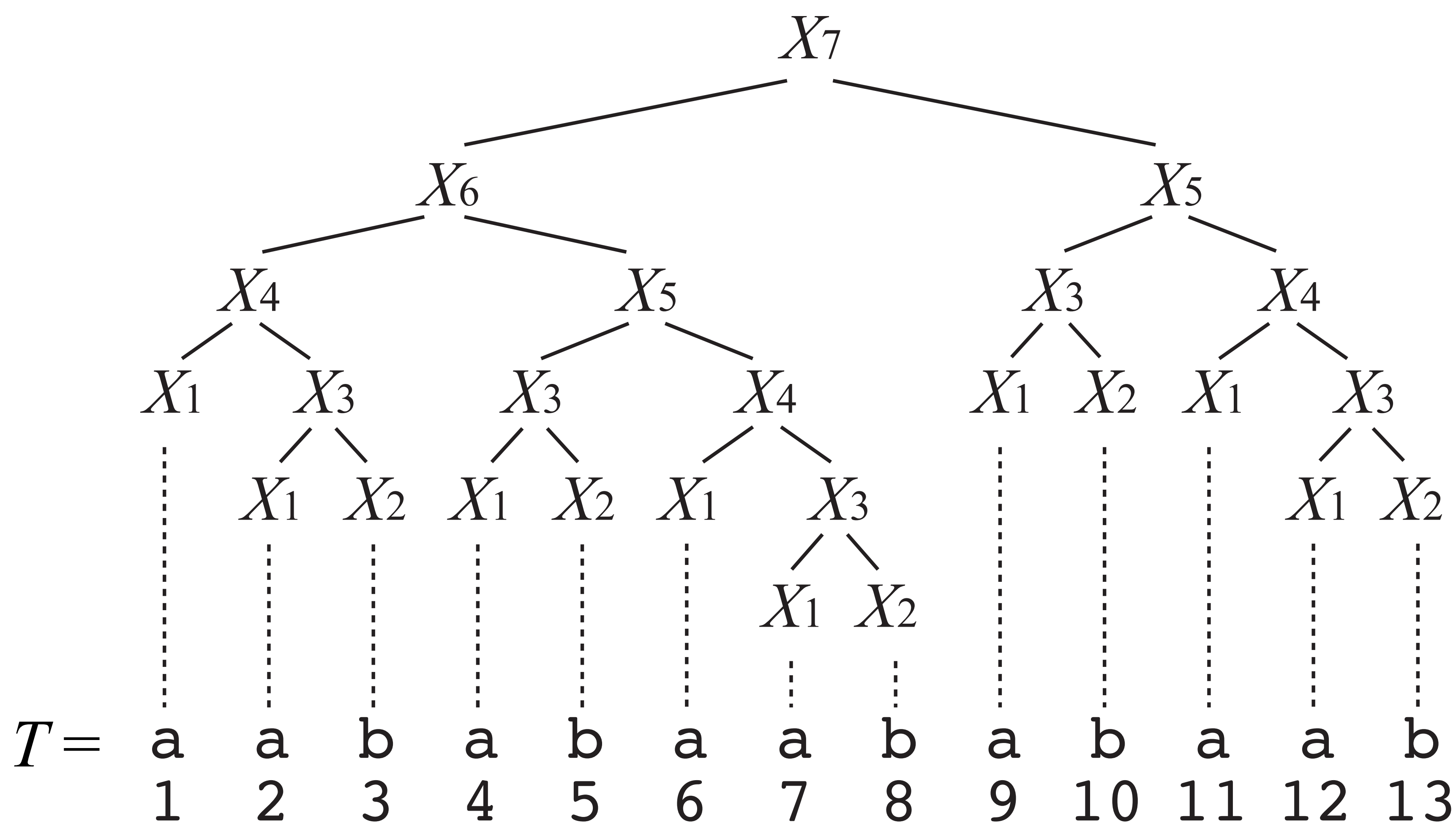
Example of SLP

SLP: D

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

$n = |D| = 7$

Derivation Tree of D



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

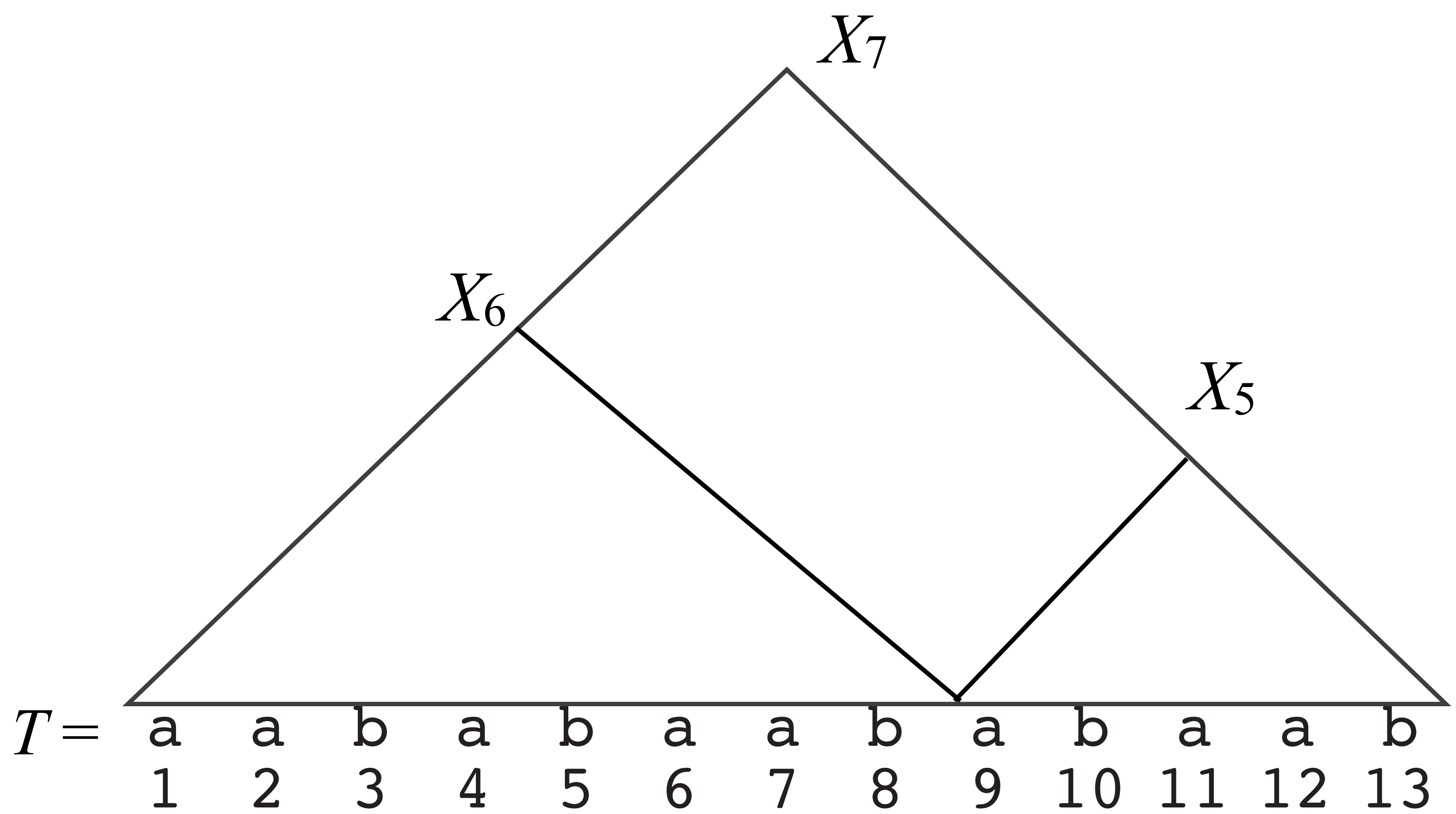
Example of SLP

SLP: D

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

$n = |D| = 7$

Derivation Tree of D



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

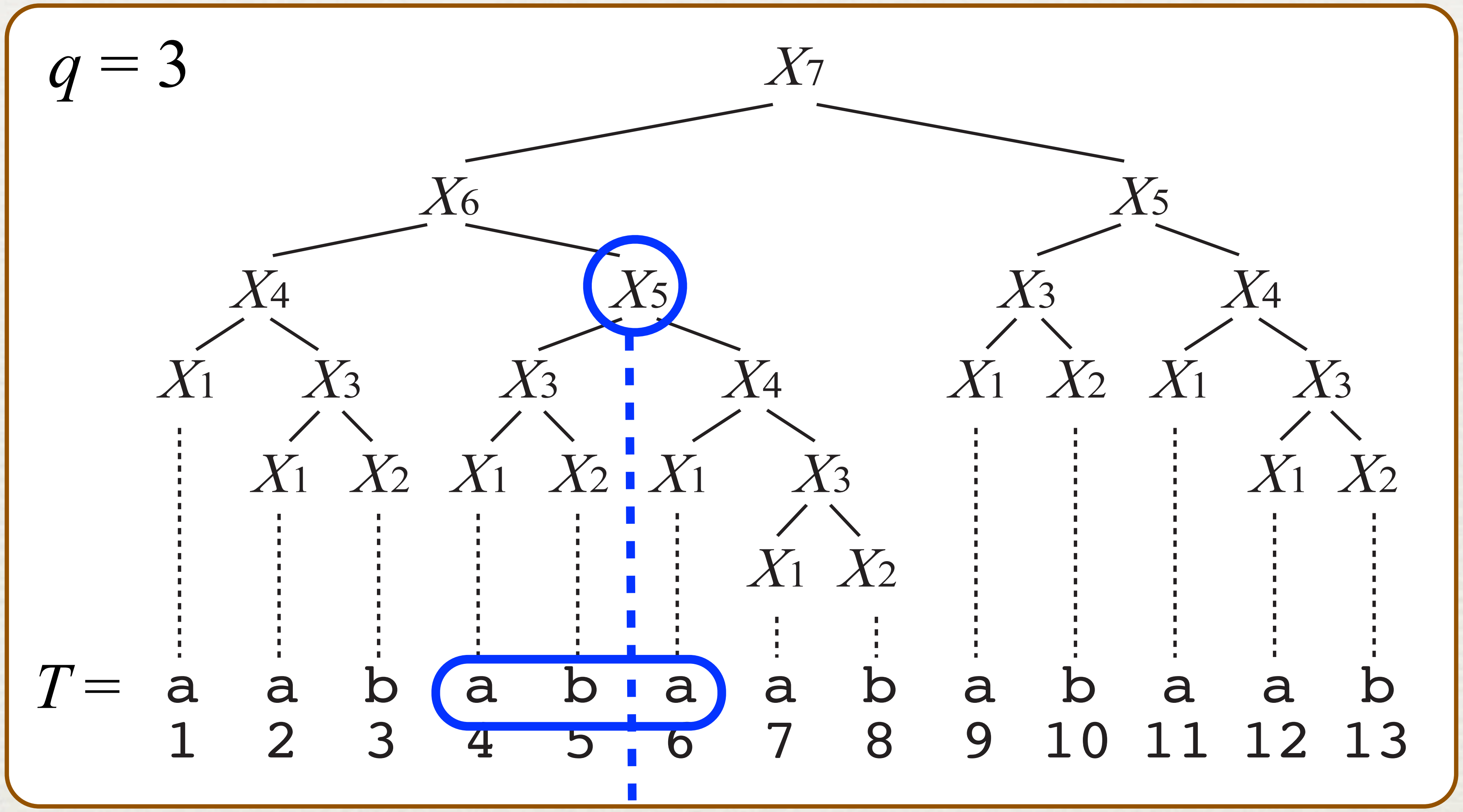
$O(qn)$ algorithm for q-gram frequencies problem on SLP

[Goto et al., SPIRE 2011]

Important Observation: stabbing

Definition

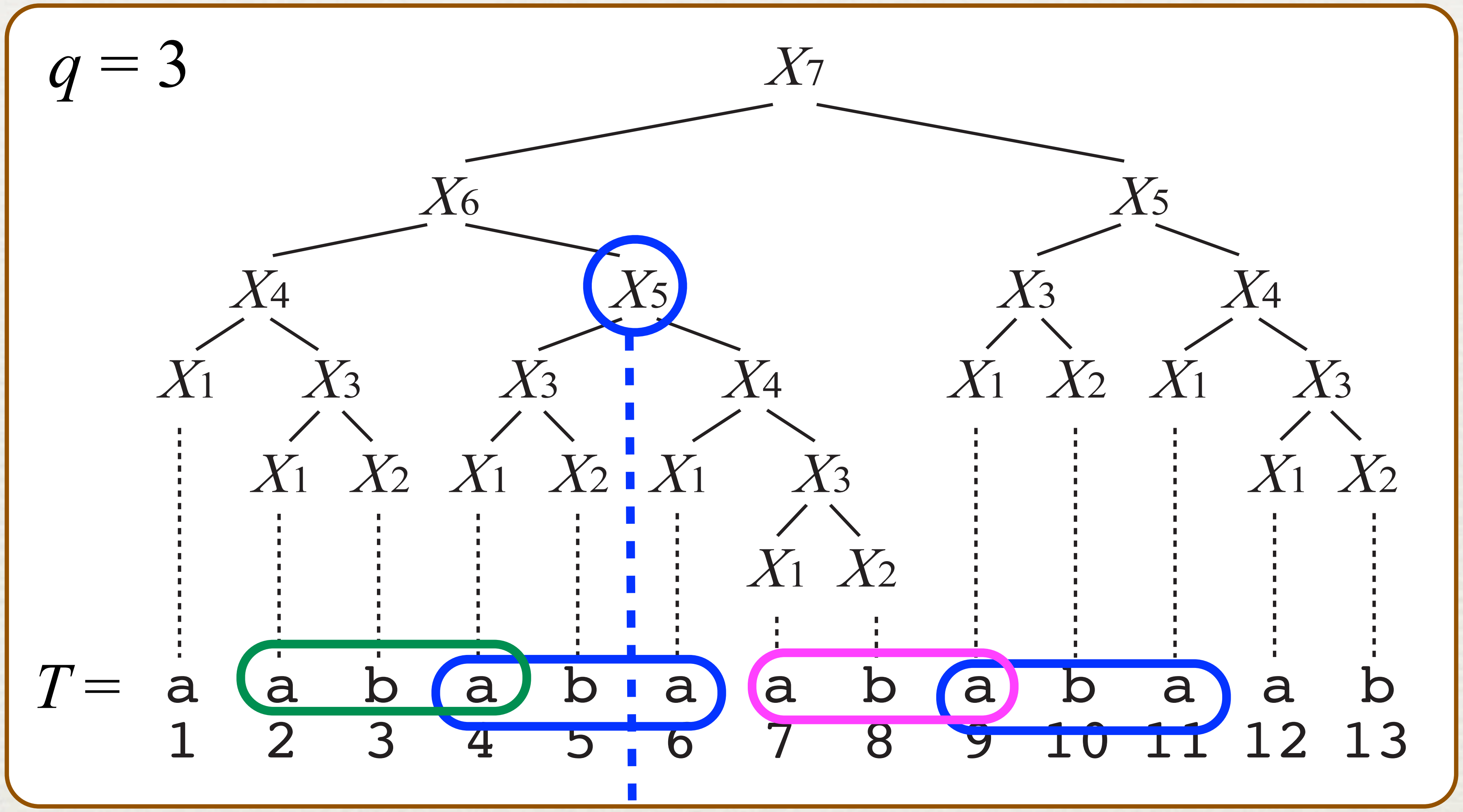
For $X_i = X_l X_r$, X_i stabs an occurrence of $P \Leftrightarrow P$ starts in X_l and ends in X_r



Important Observation: stabbing

Definition

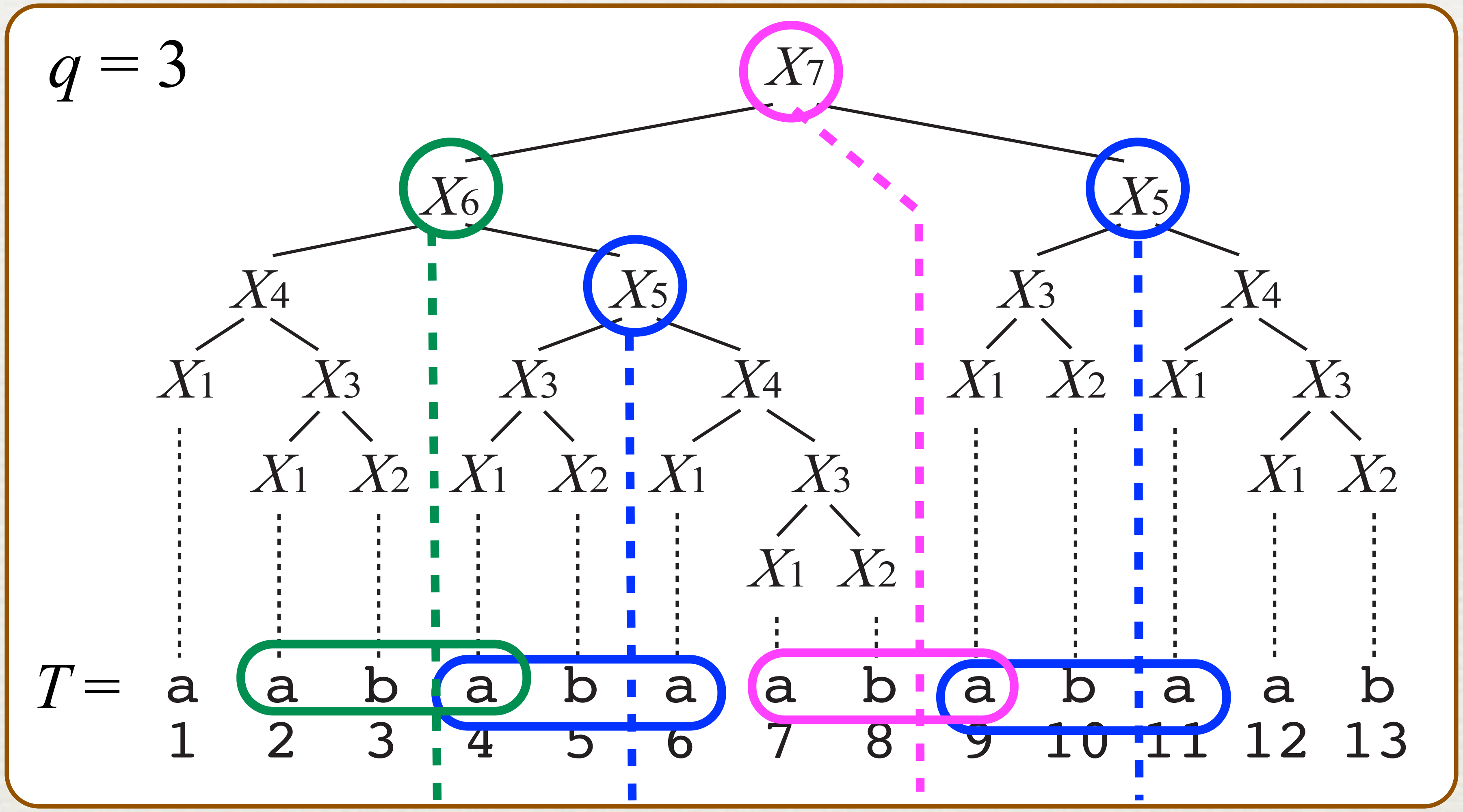
For $X_i = X_l X_r$, X_i stabs an occurrence of P $\Leftrightarrow$ P starts in X_l and ends in X_r



Important Observation: stabbing

Definition

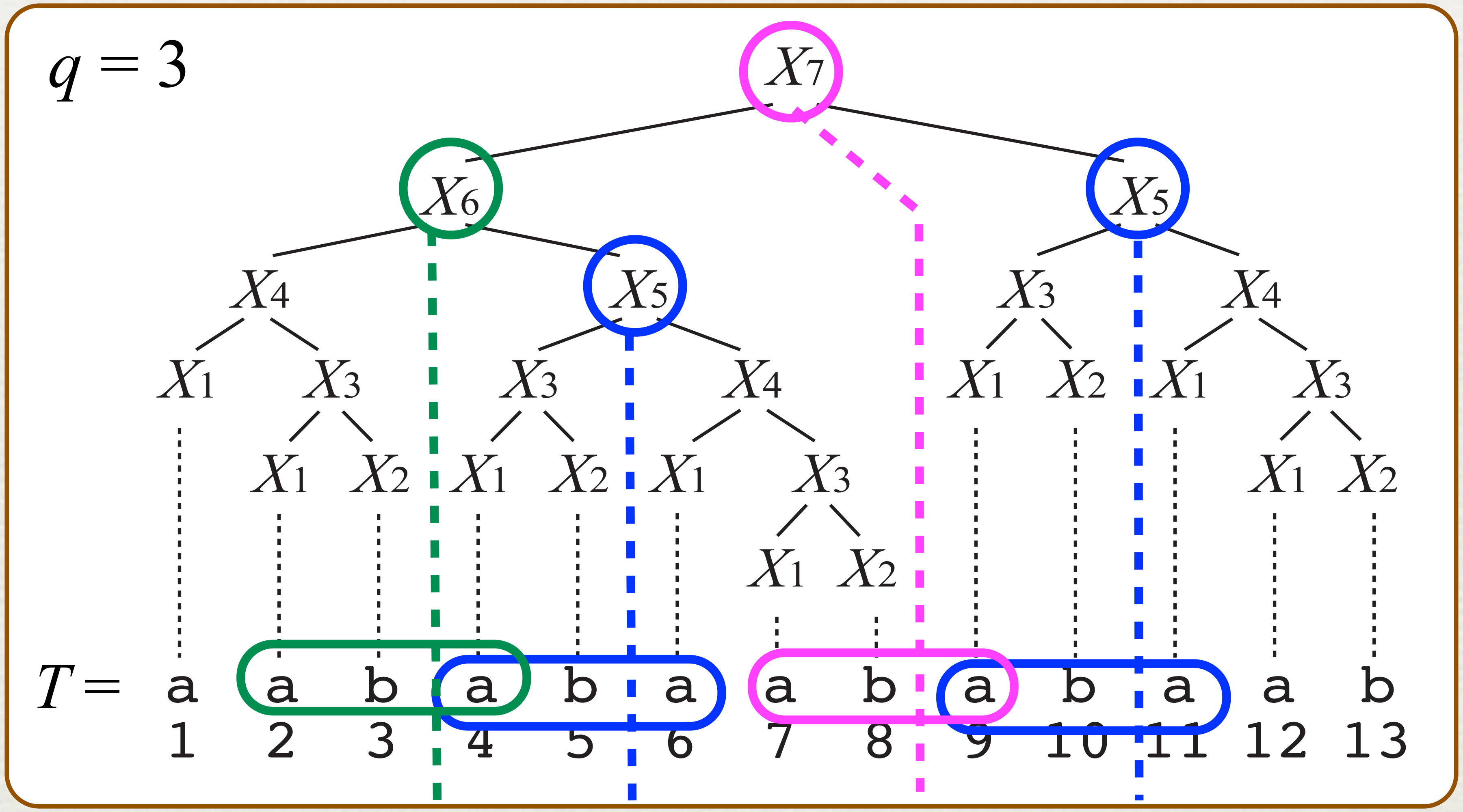
For $X_i = X_l X_r$, X_i stabs an occurrence of P $\Leftrightarrow$ P starts in X_l and ends in X_r



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



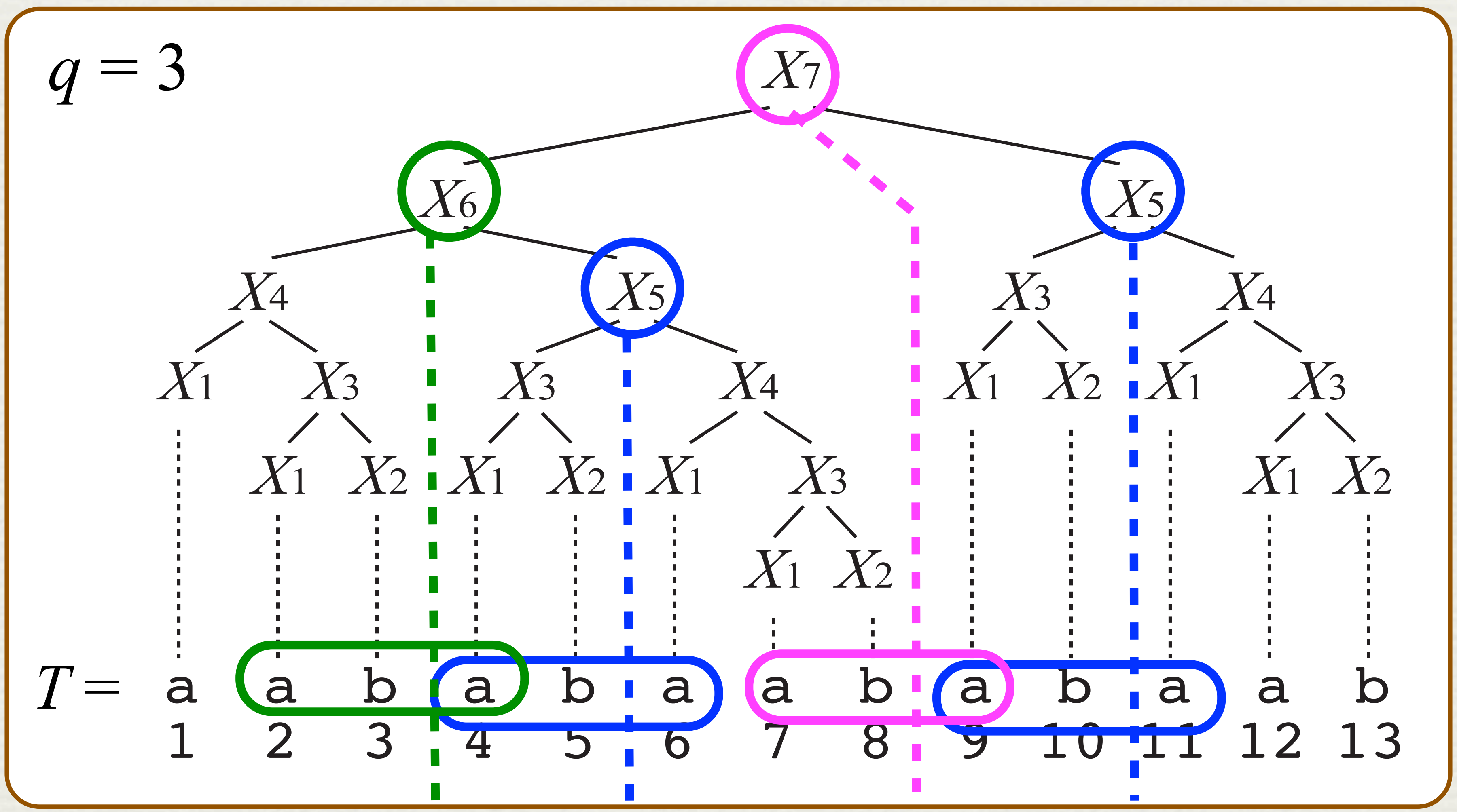
Observation

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

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

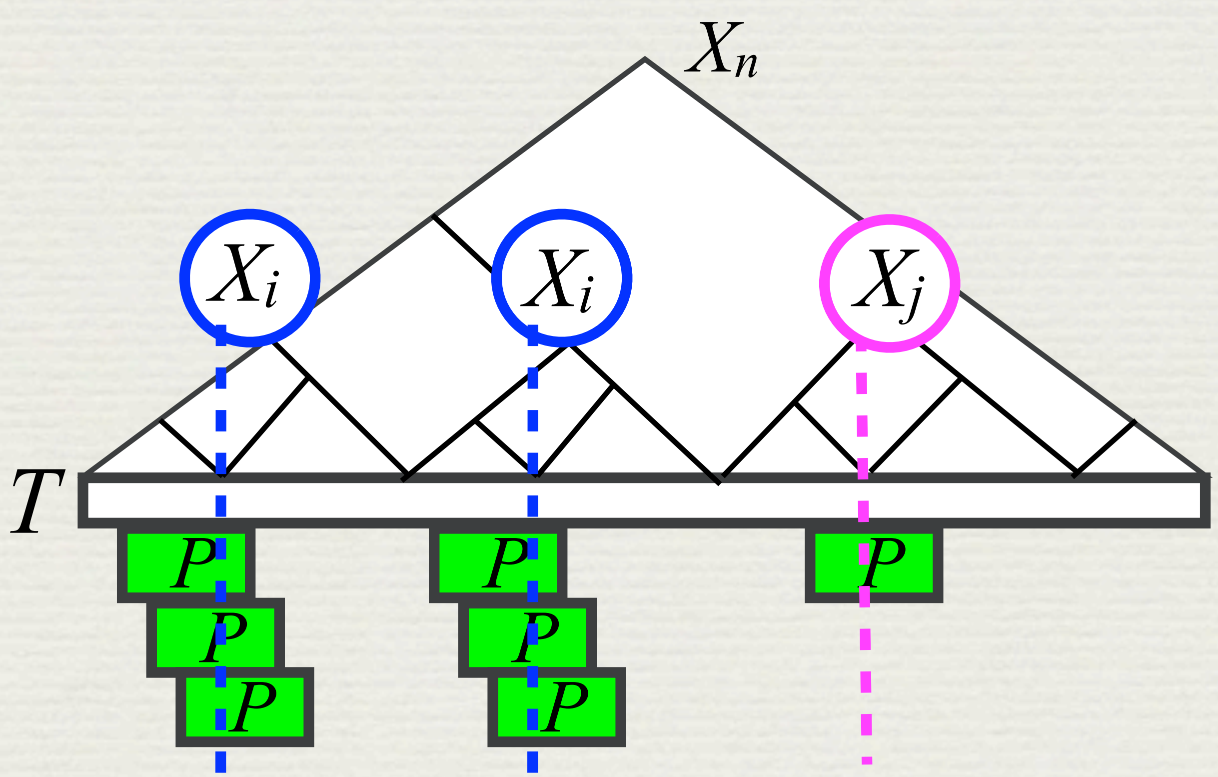


More formal description

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.



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.

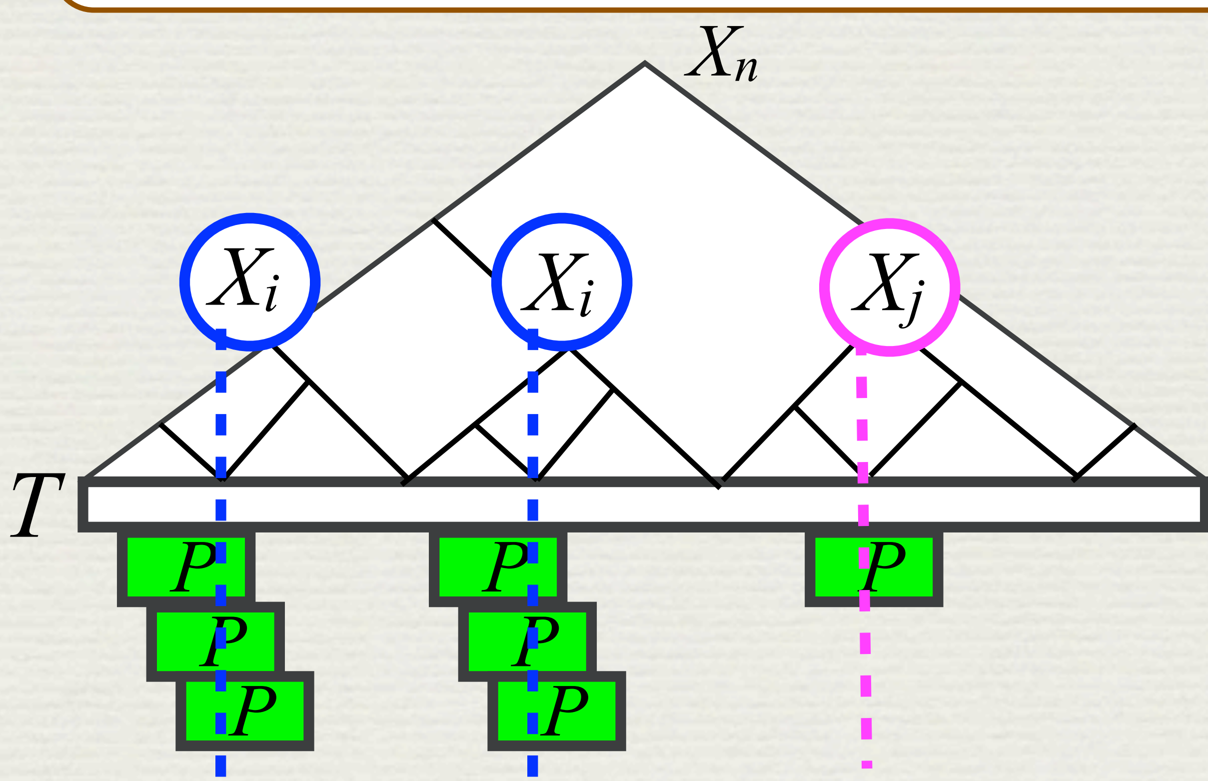
串(Kushi):

Japanese skewer,
used to stab foods

BBQ

Yakitori

Oden



More formal description

Definition

For each variable X_i,

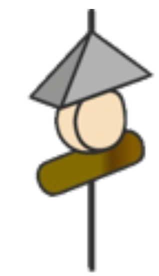
- $Freq^{串}(X_i, P)$: # occurrences of P stabbed by X_i in the string derived from X_i.
- $vOcc(X_i)$: # nodes labeled by X_i in the derivation tree of the last variable X_n.

串(Kushi):

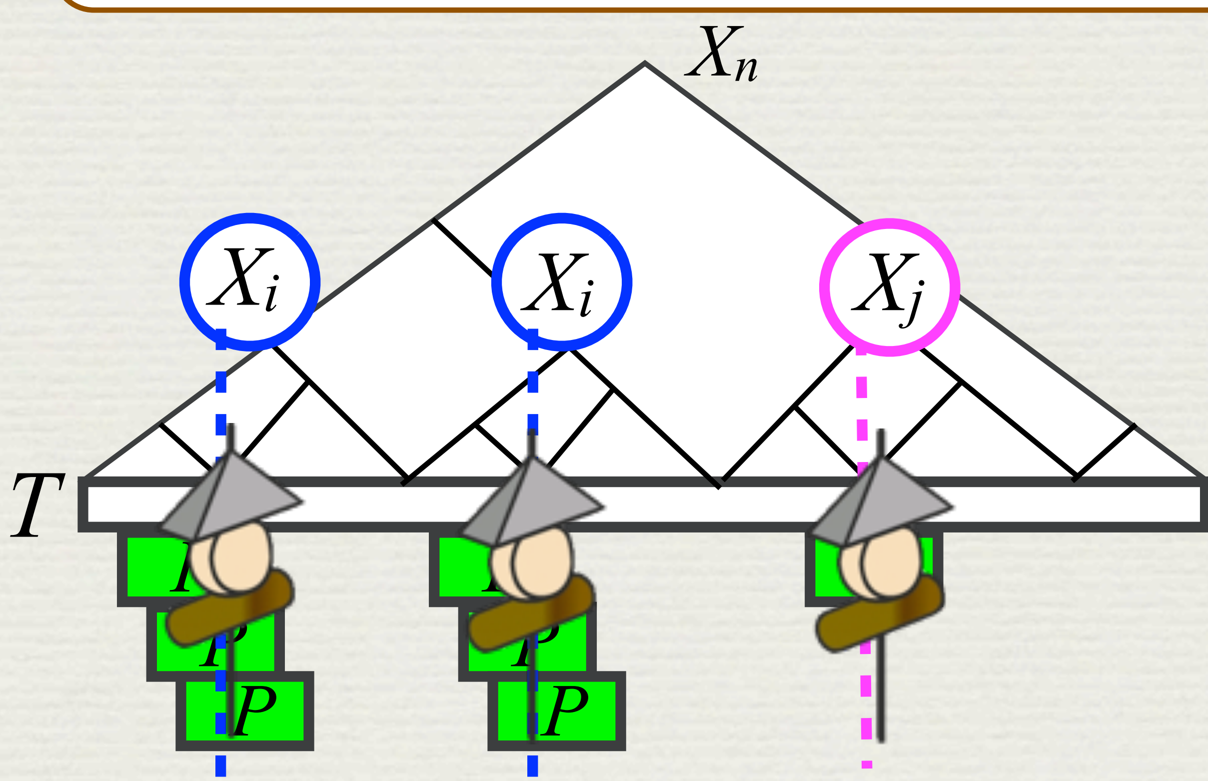
Japanese skewer,
used to stab foods

BBQ

Yakitori



Oden

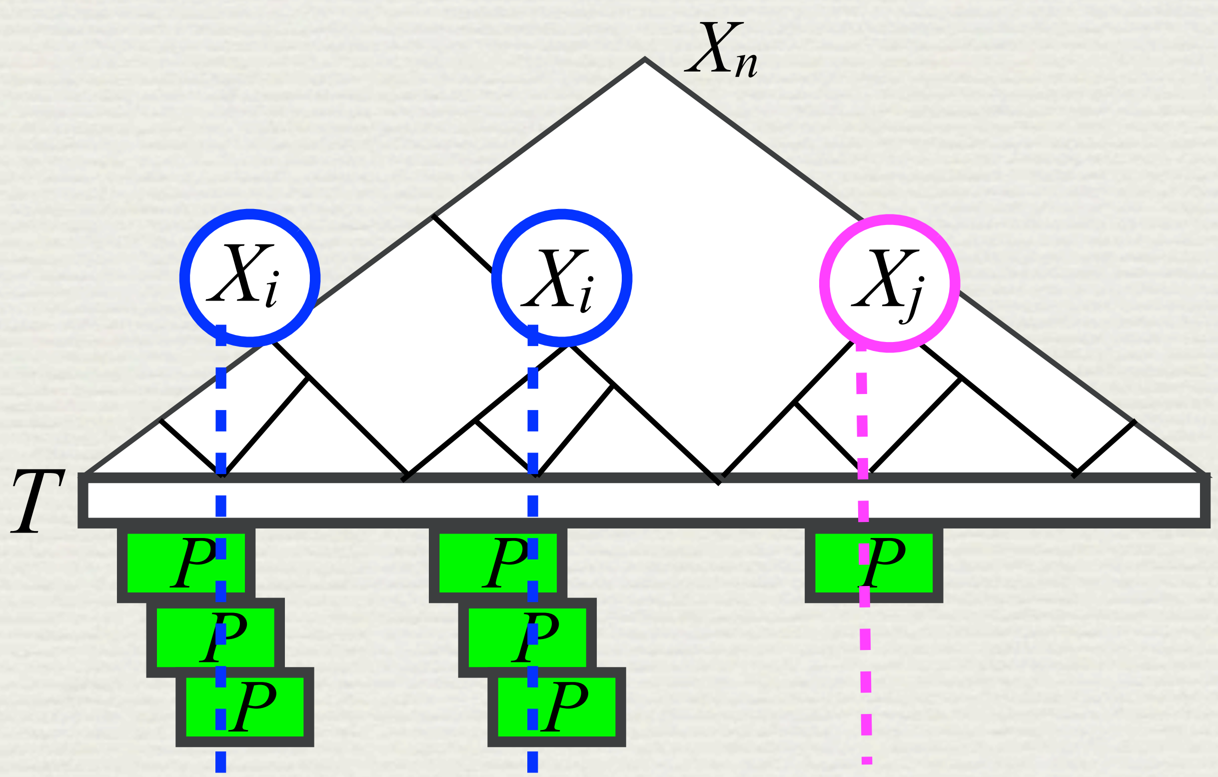


More formal description

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.



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

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

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

More formal description

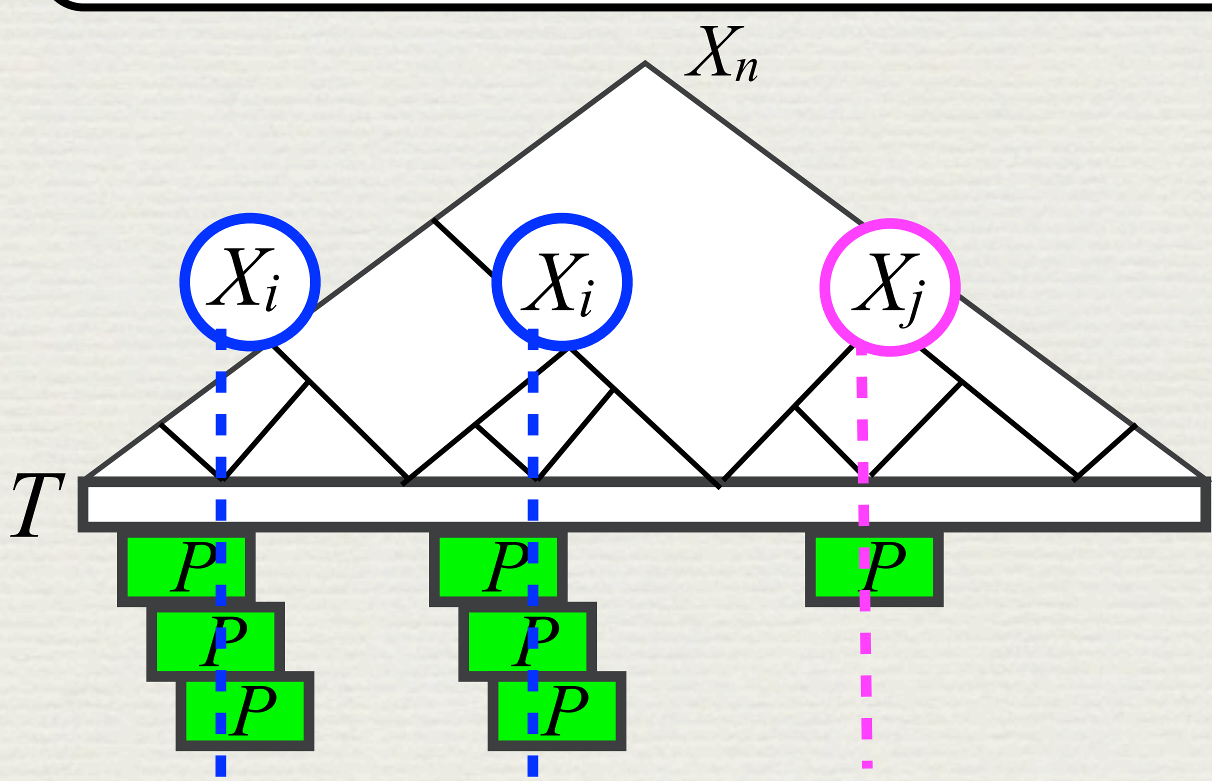
Definition

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

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

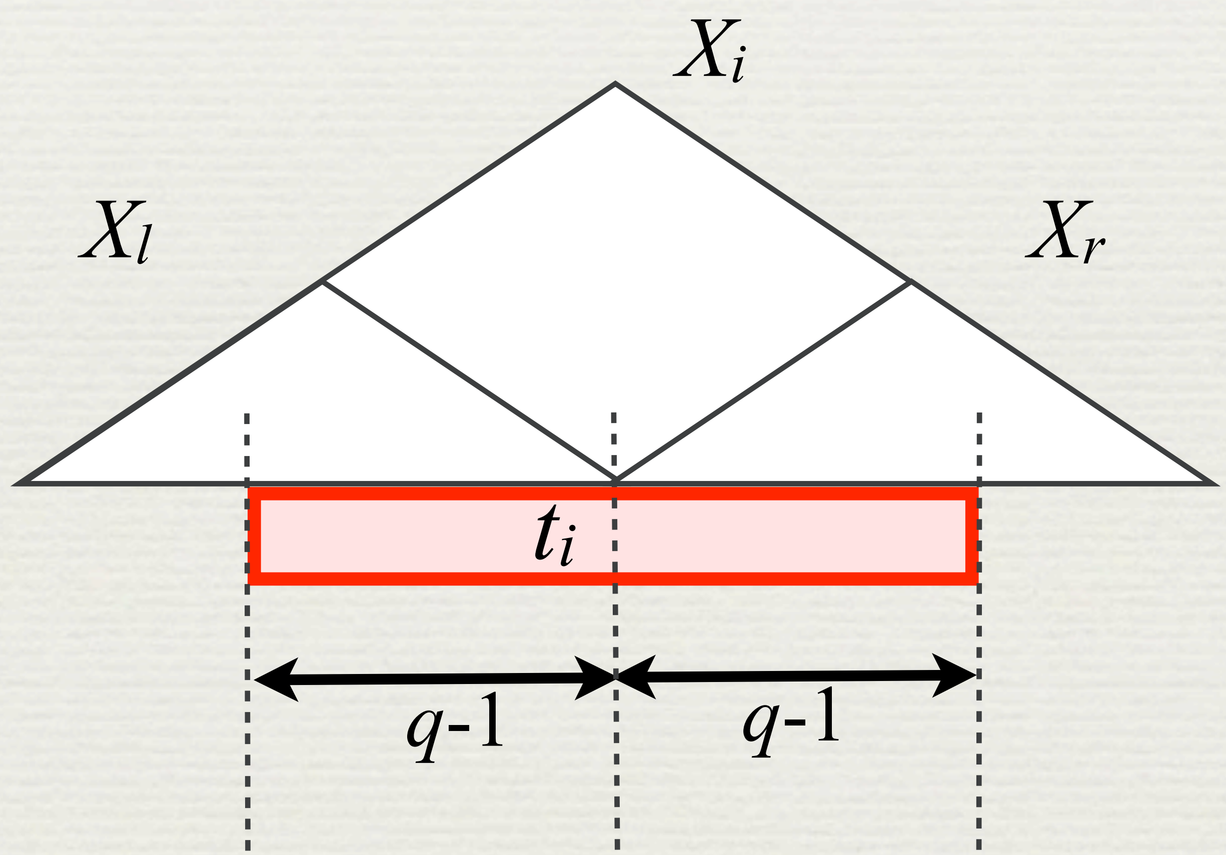


$Freq^{串}(X_i, P) = 3$, $Freq^{串}(X_j, P) = 1$

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

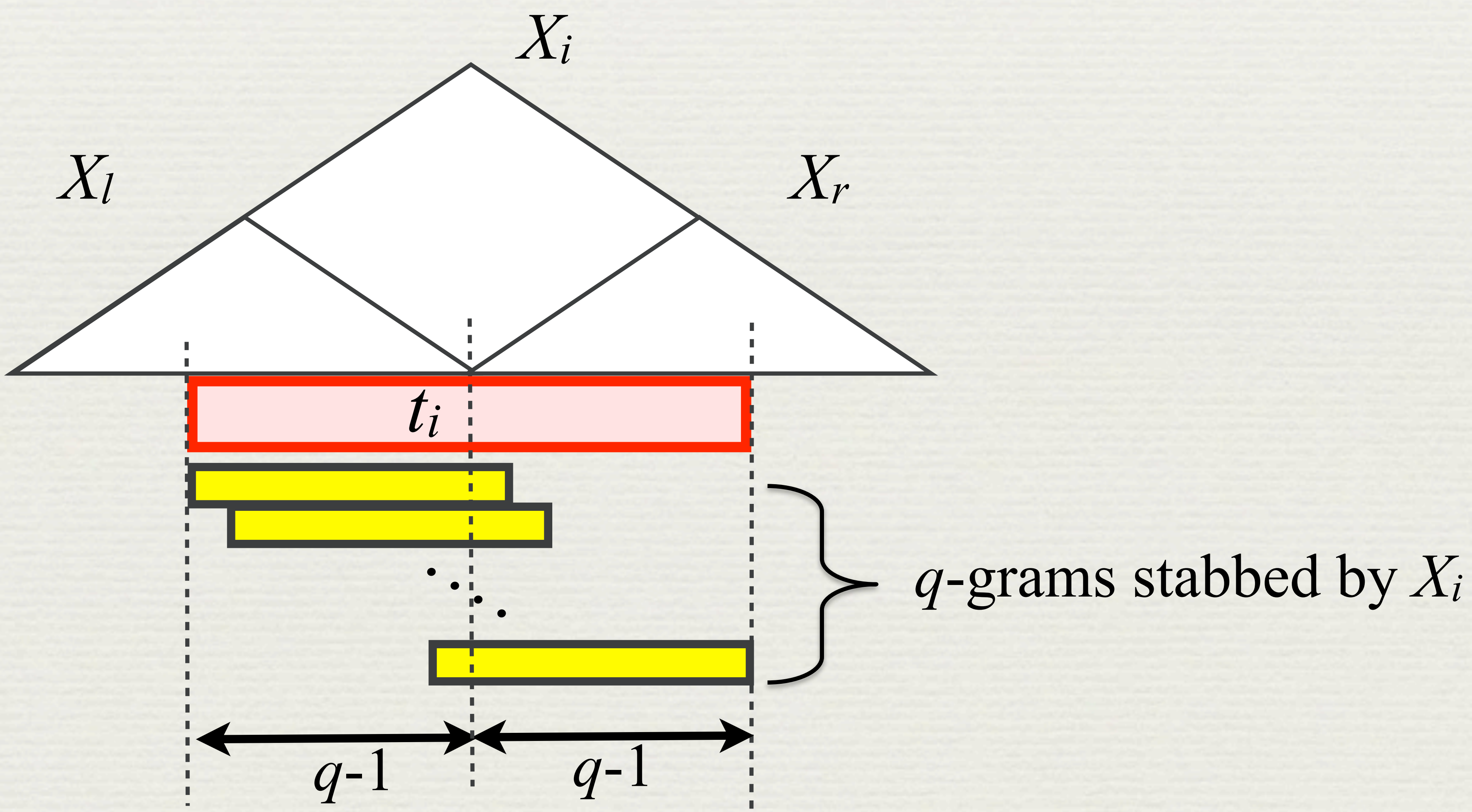
Frequency of P = 3 ・ 2 + 1 ・ 1 = 7

Computing $Freq^{串}(X_i, P)$



Computing $Freq^{串}(X_i, P)$

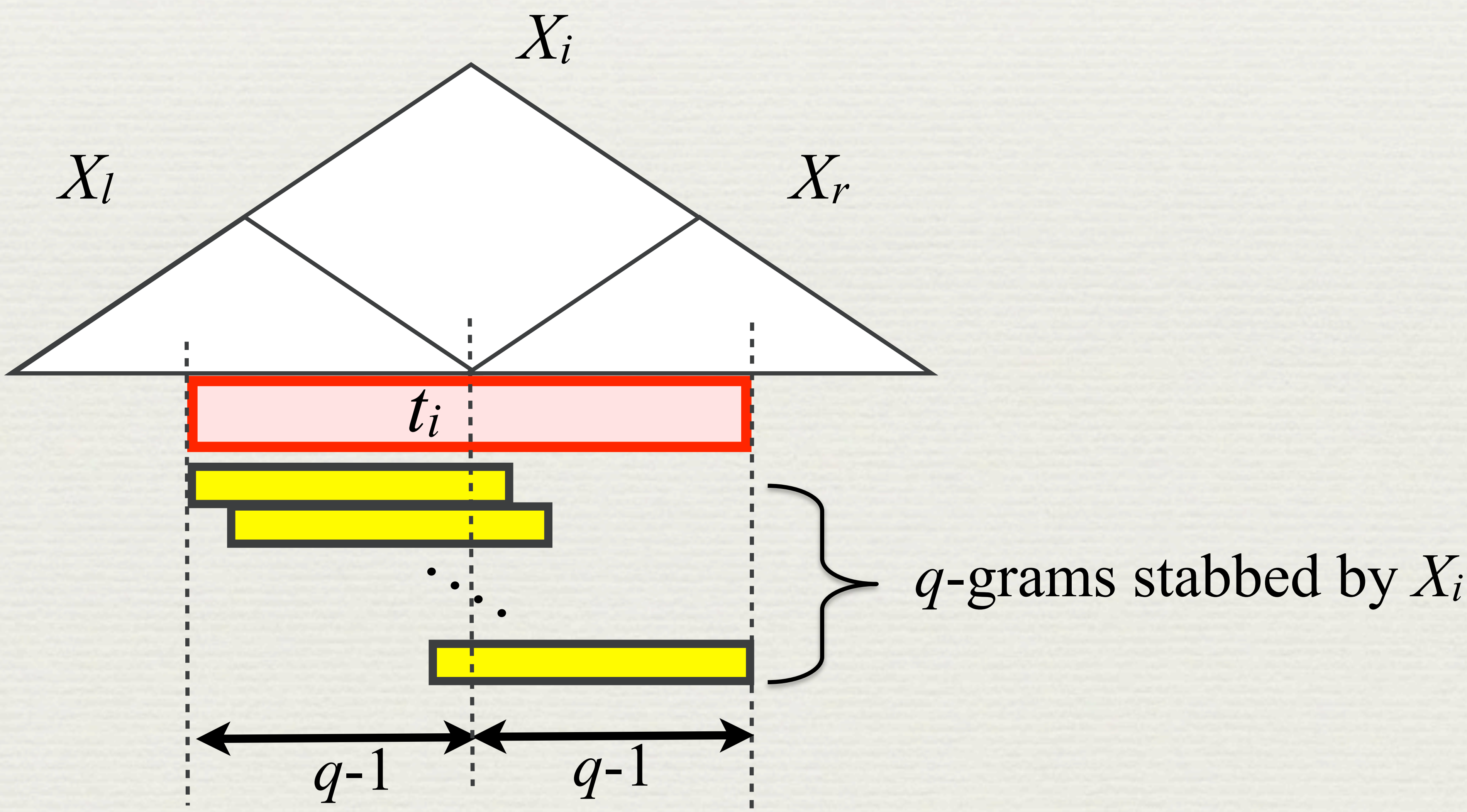
X_i stabs $P \Leftrightarrow$ P starts in X_l and ends in X_r



Computing $Freq^{串}(X_i, P)$

Observation

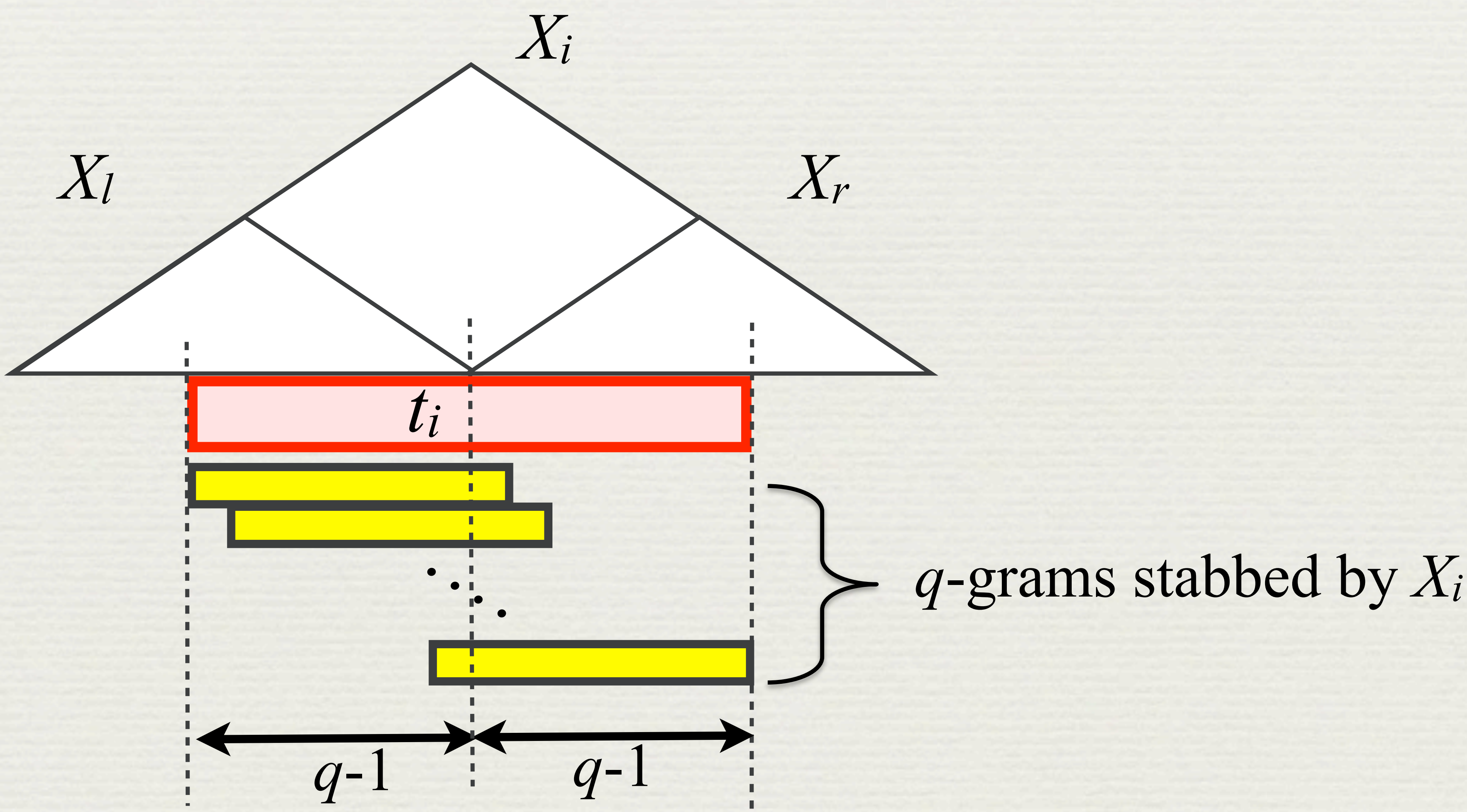
For any $P \in \Sigma^q$, $Freq^{串}(X_i, P) = Freq(t_i, P)$



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

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

$O(qn)$ time and space in total

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

$O(n)$ time and space in total

Lemma

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

$O(qn)$ time and space in total

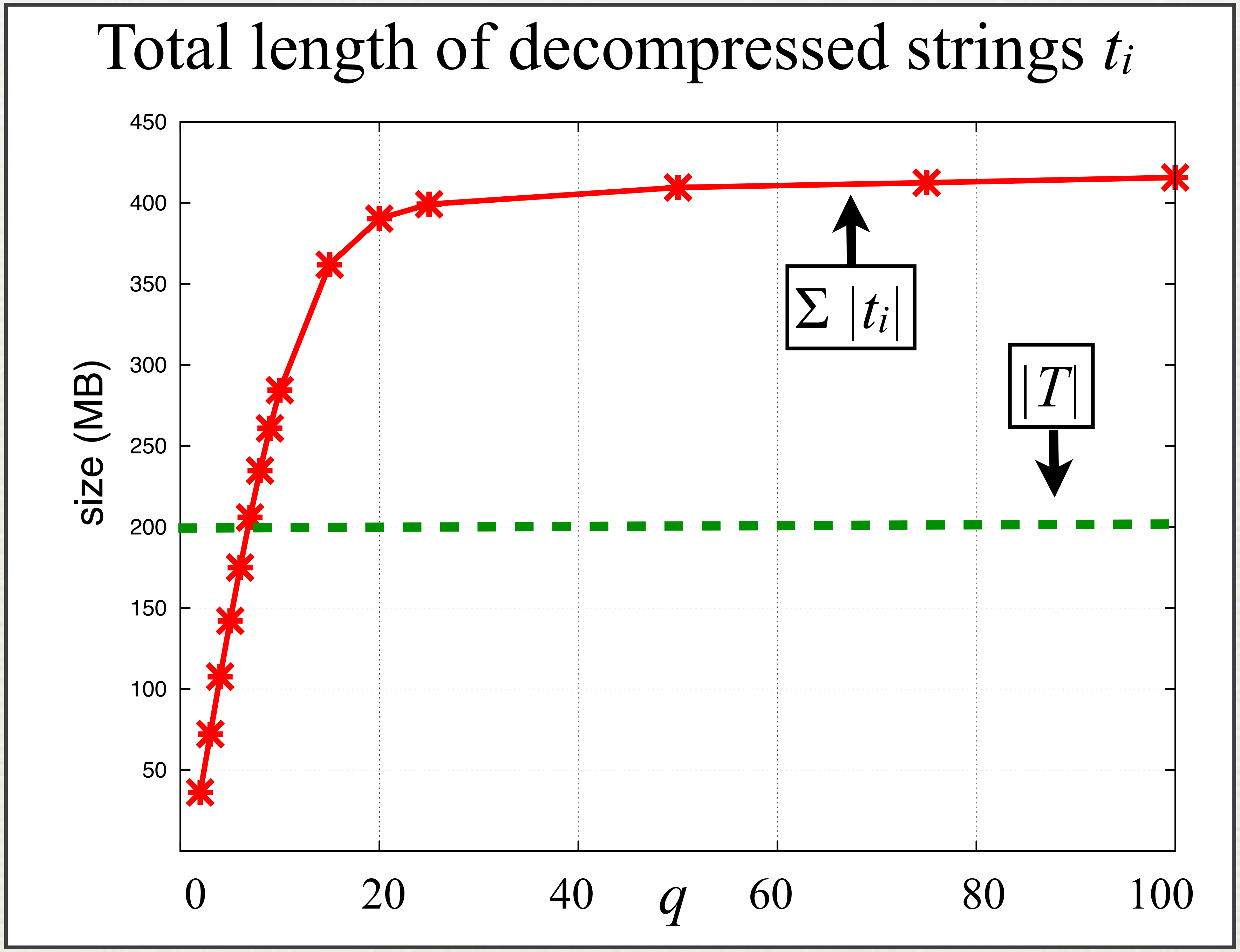
Theorem

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

Sketch of proof:

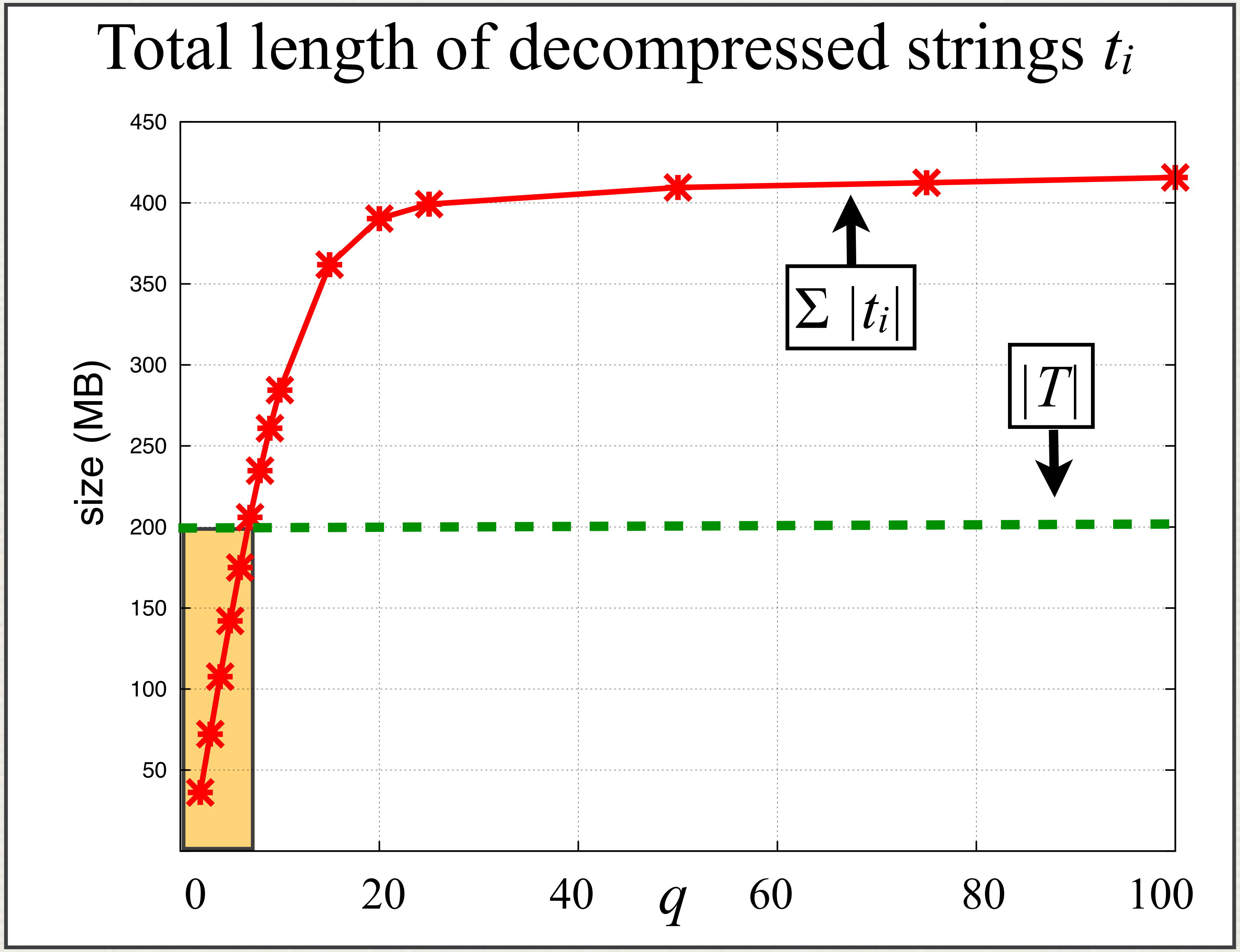
Using the suffix array of the concatenation of all t_i's, we can compute all q-gram frequencies in $O(qn)$ time and space.

Efficiency & Inefficiency of $O(qn)$ algorithm



ENGLISH data of 200MB from pizza & chili corpus

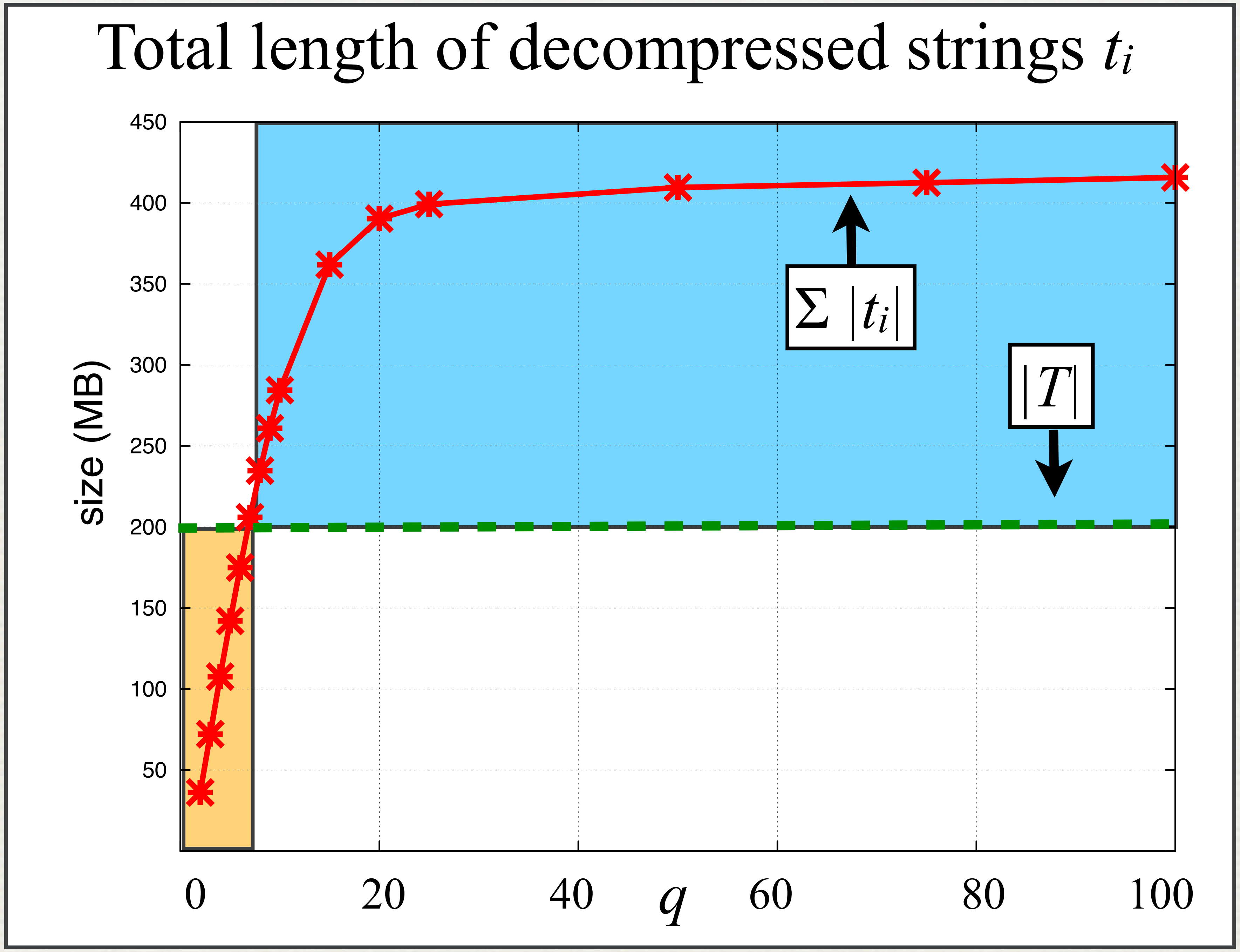
Efficiency & Inefficiency of $O(qn)$ algorithm



ENGLISH data of 200MB from pizza & chili corpus

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

Efficiency & Inefficiency of $O(qn)$ algorithm



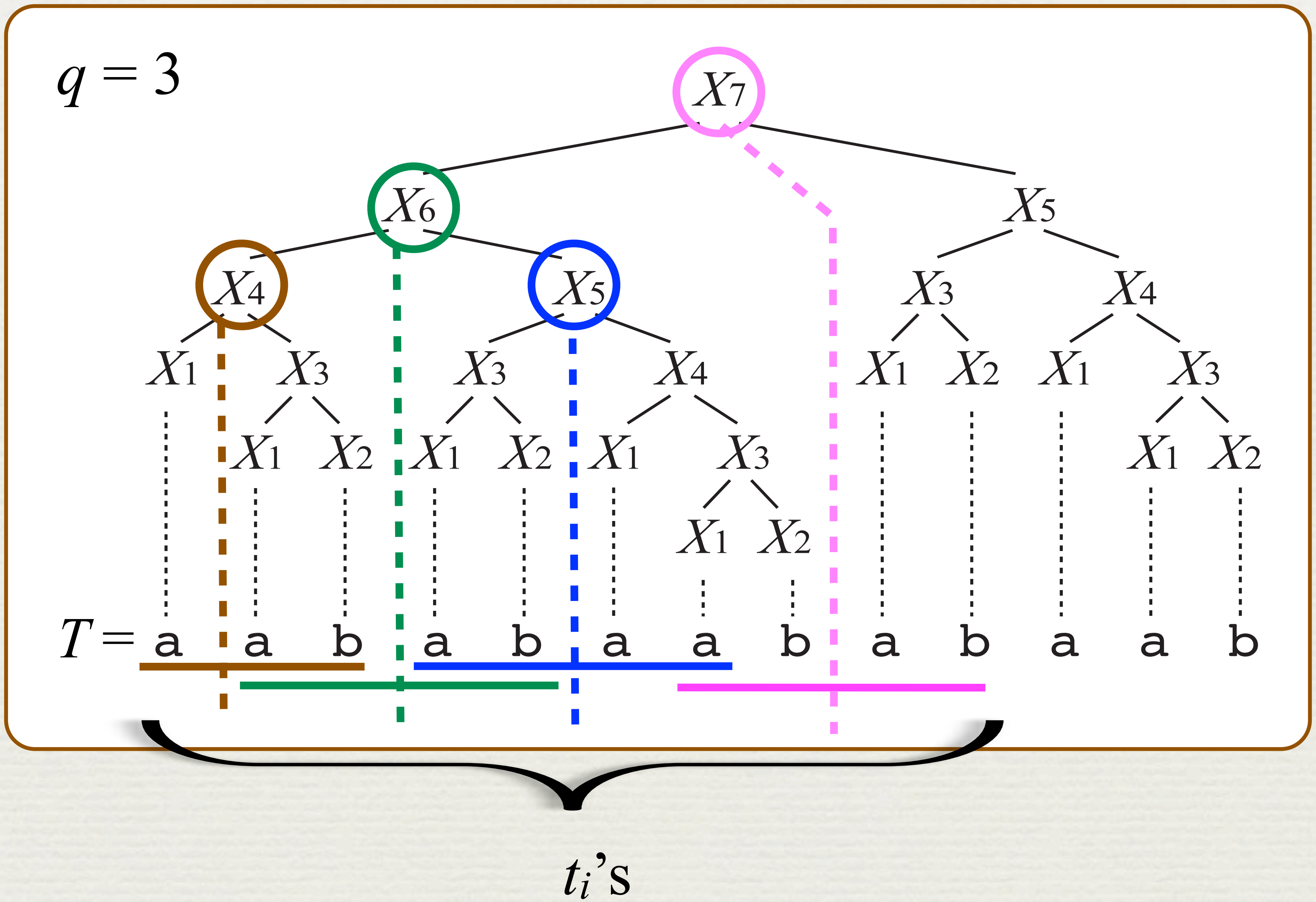
- when q is **large**, the algorithm runs **slower**
- when q is **small**, the algorithm runs **faster**

ENGLISH data of 200MB from pizza & chili corpus

New algorithm

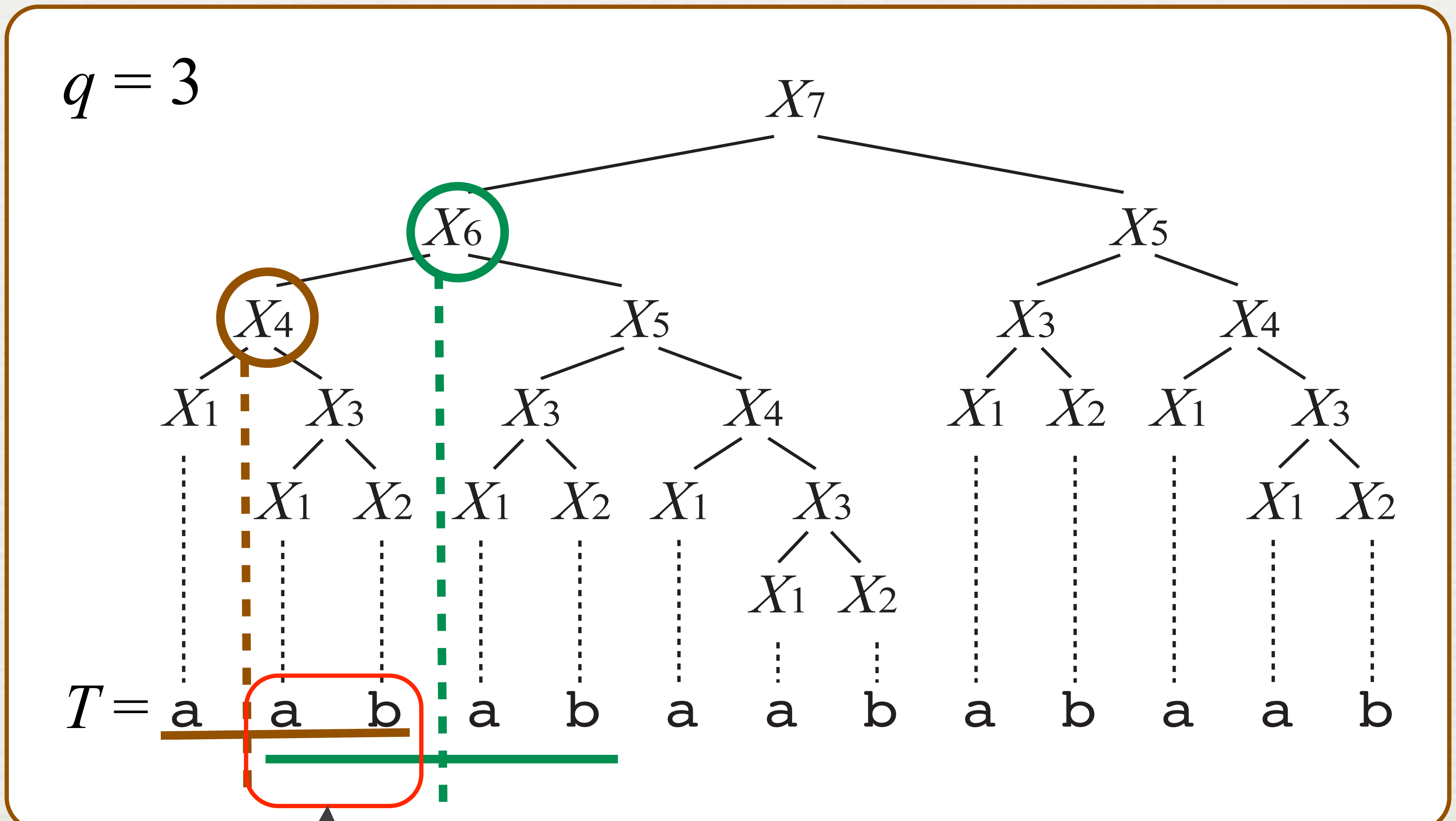
Inefficiency of $O(qn)$ algorithm

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



Inefficiency of $O(qn)$ algorithm

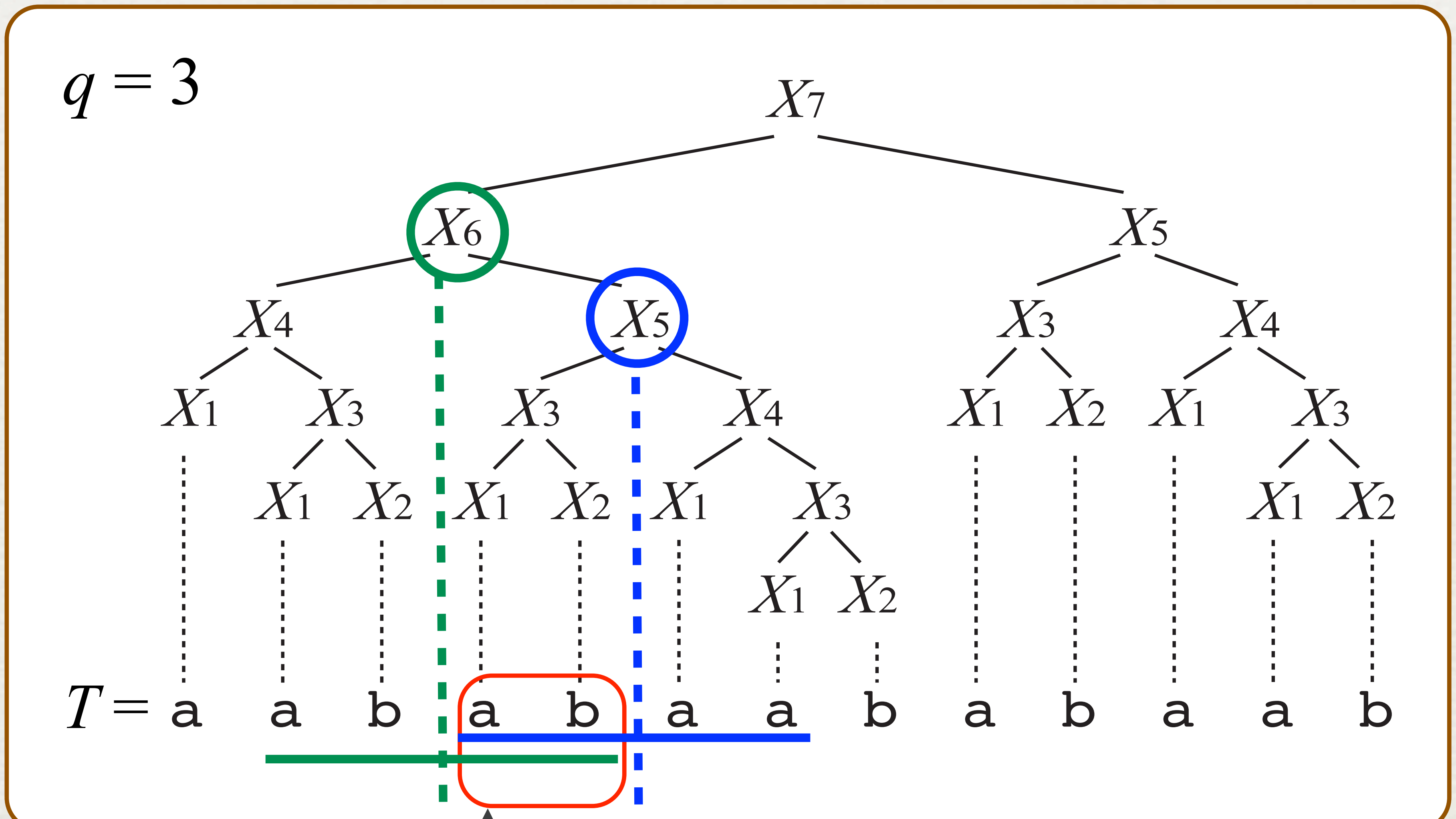
- There are overlaps between partially decompressed strings t_i



t_4 and t_6 overlap with "ab"

Inefficiency of $O(qn)$ algorithm

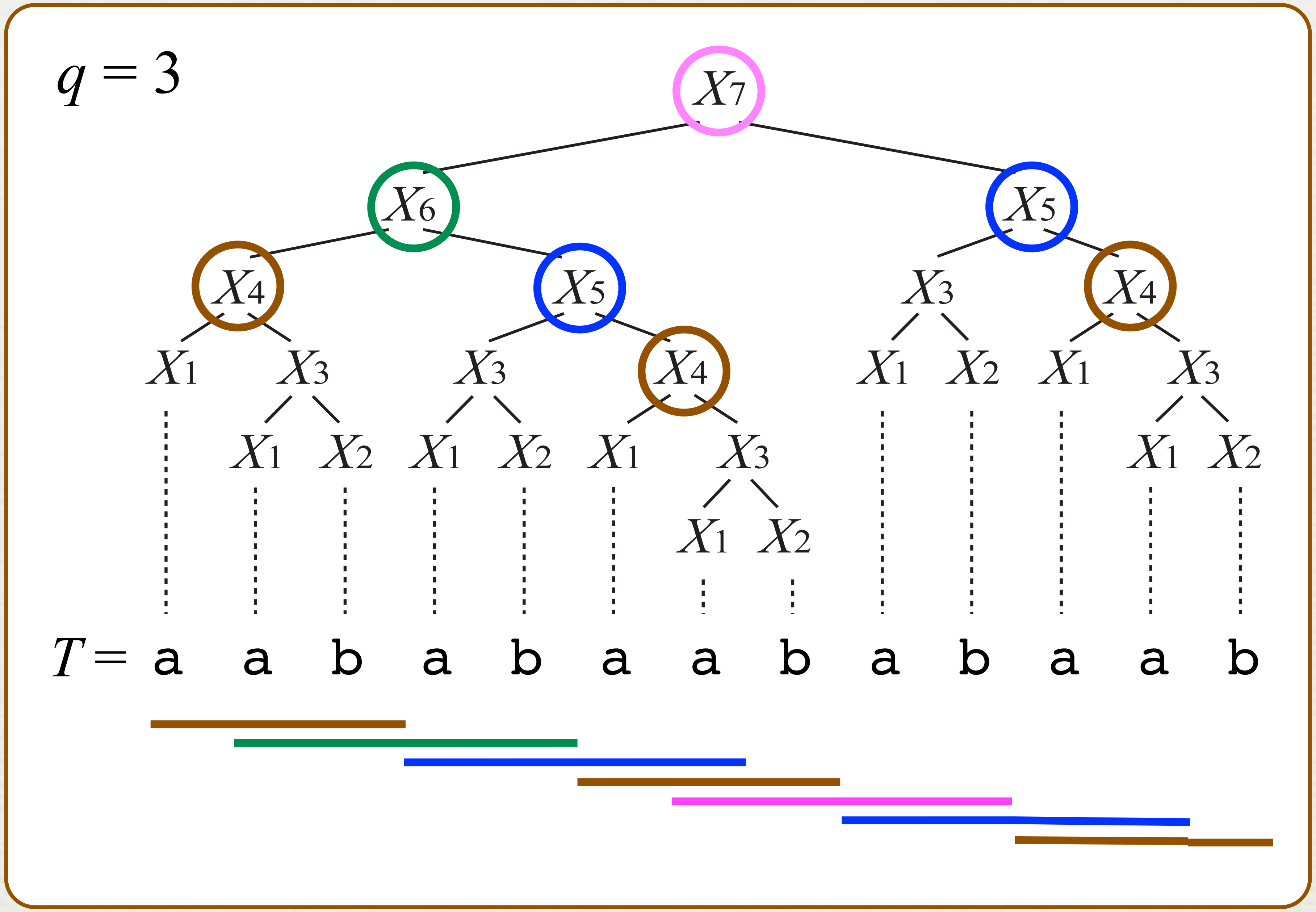
- There are overlaps between partially decompressed strings t_i



t_6 and t_5 overlap with "ab"

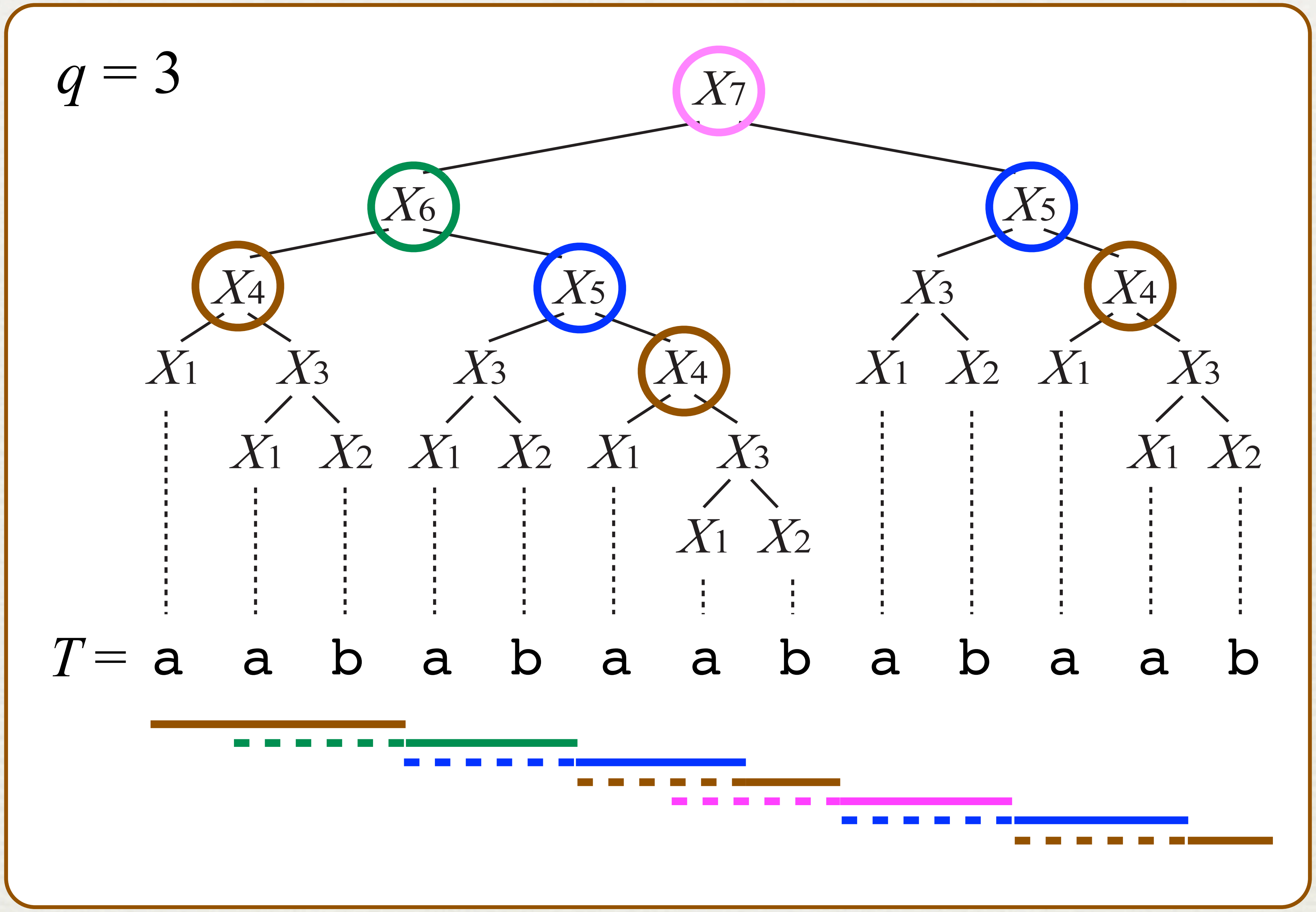
Identifying the redundancies

- Consider all partially decompressed strings t_i in derivation tree



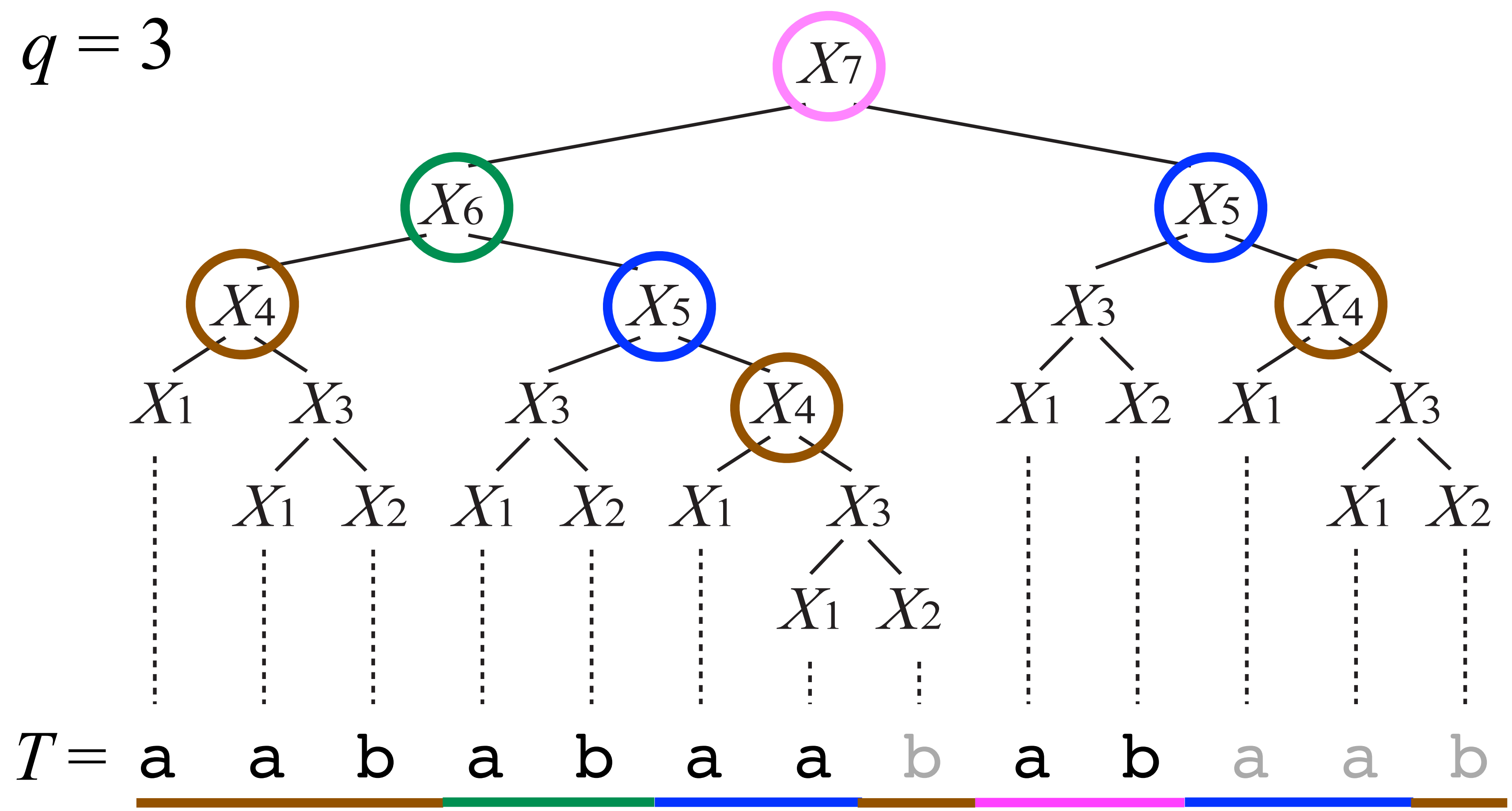
Removing overlaps of neighboring t_i's

- Eliminate length-(q-1) prefix of all t_i's except for leftmost one
- Concatenation of remaining strings equals to T



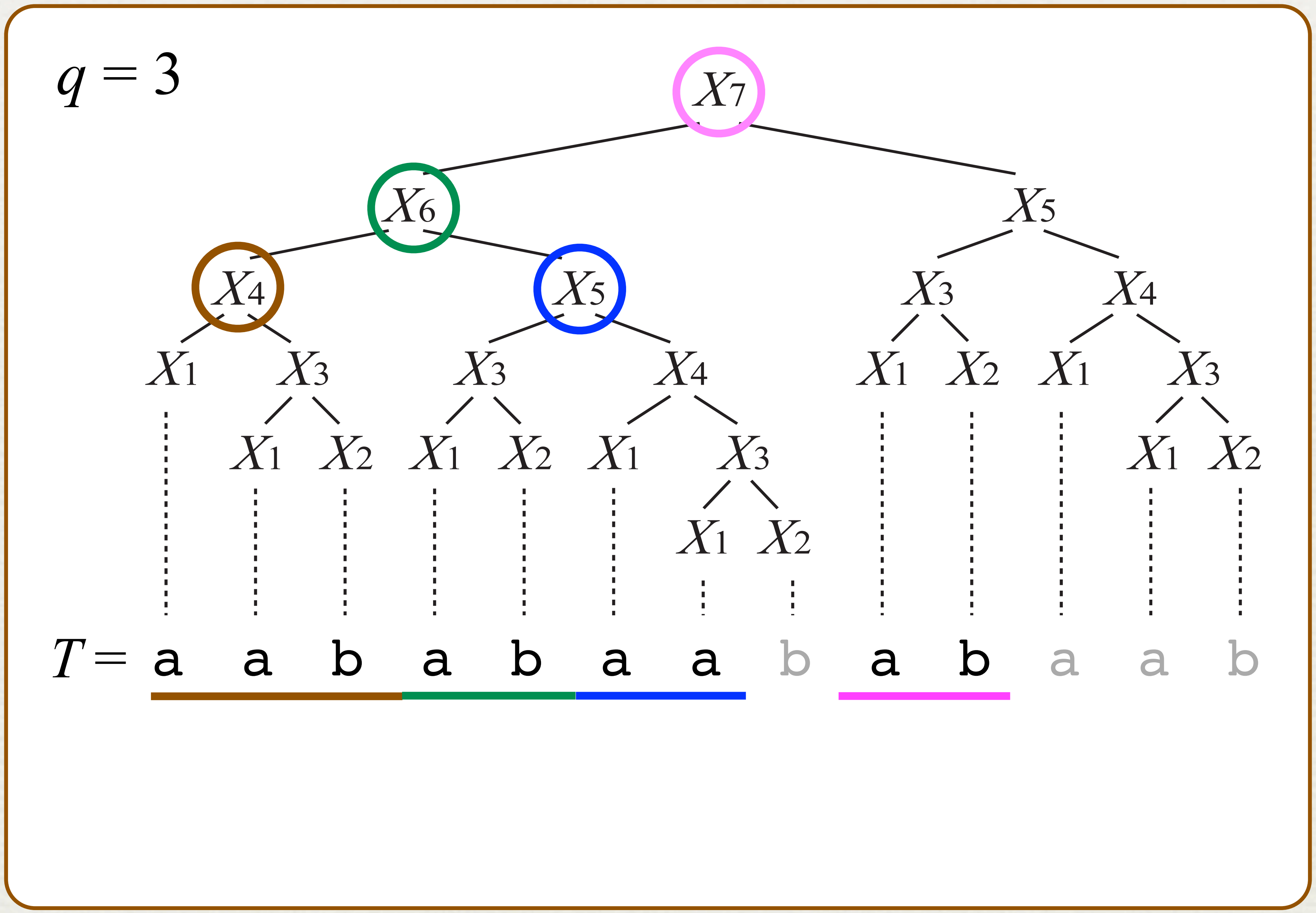
Removing duplicate t_i's

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



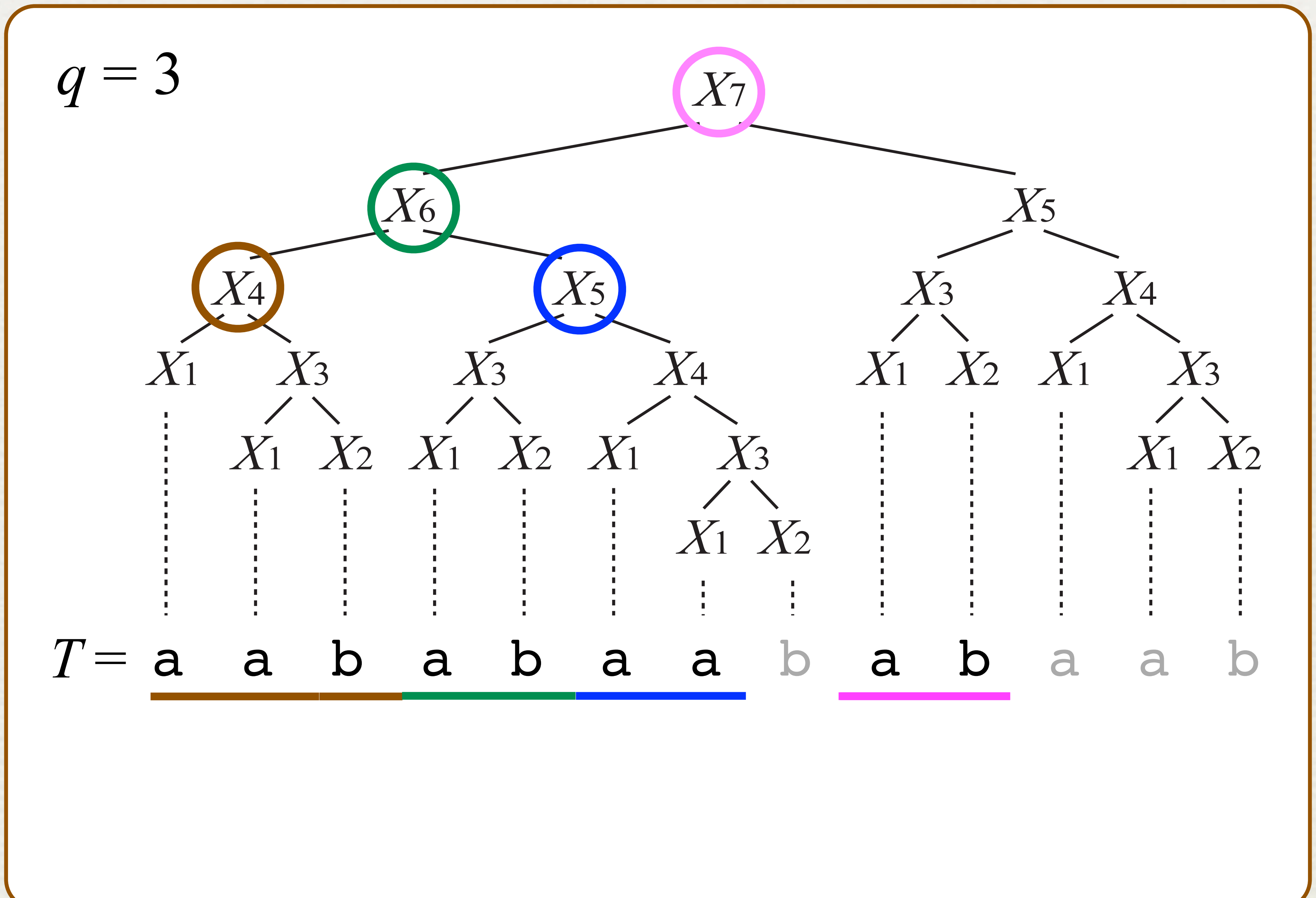
Removing duplicate t_i's

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



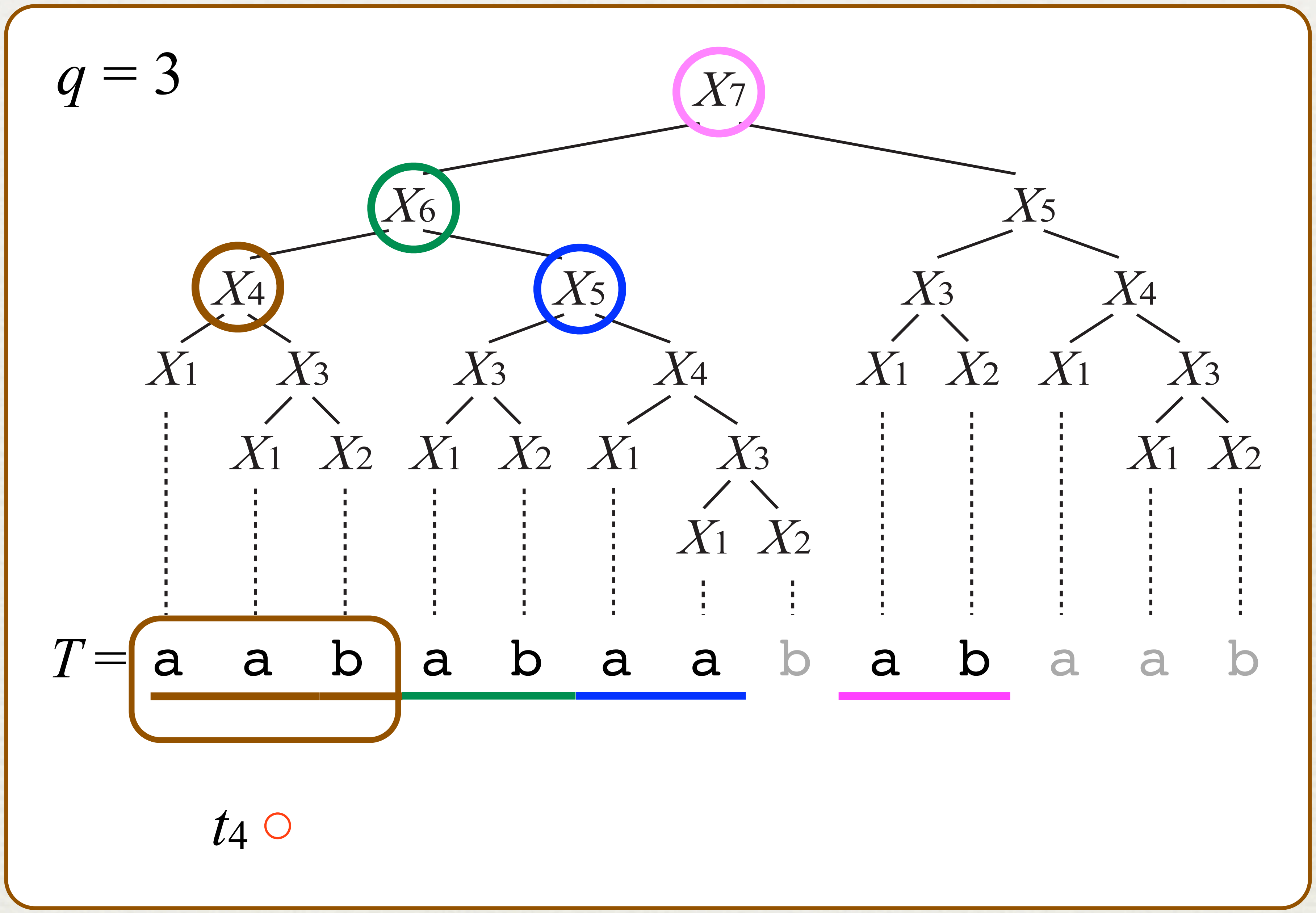
What we have left

- Compact representation of all t_i's



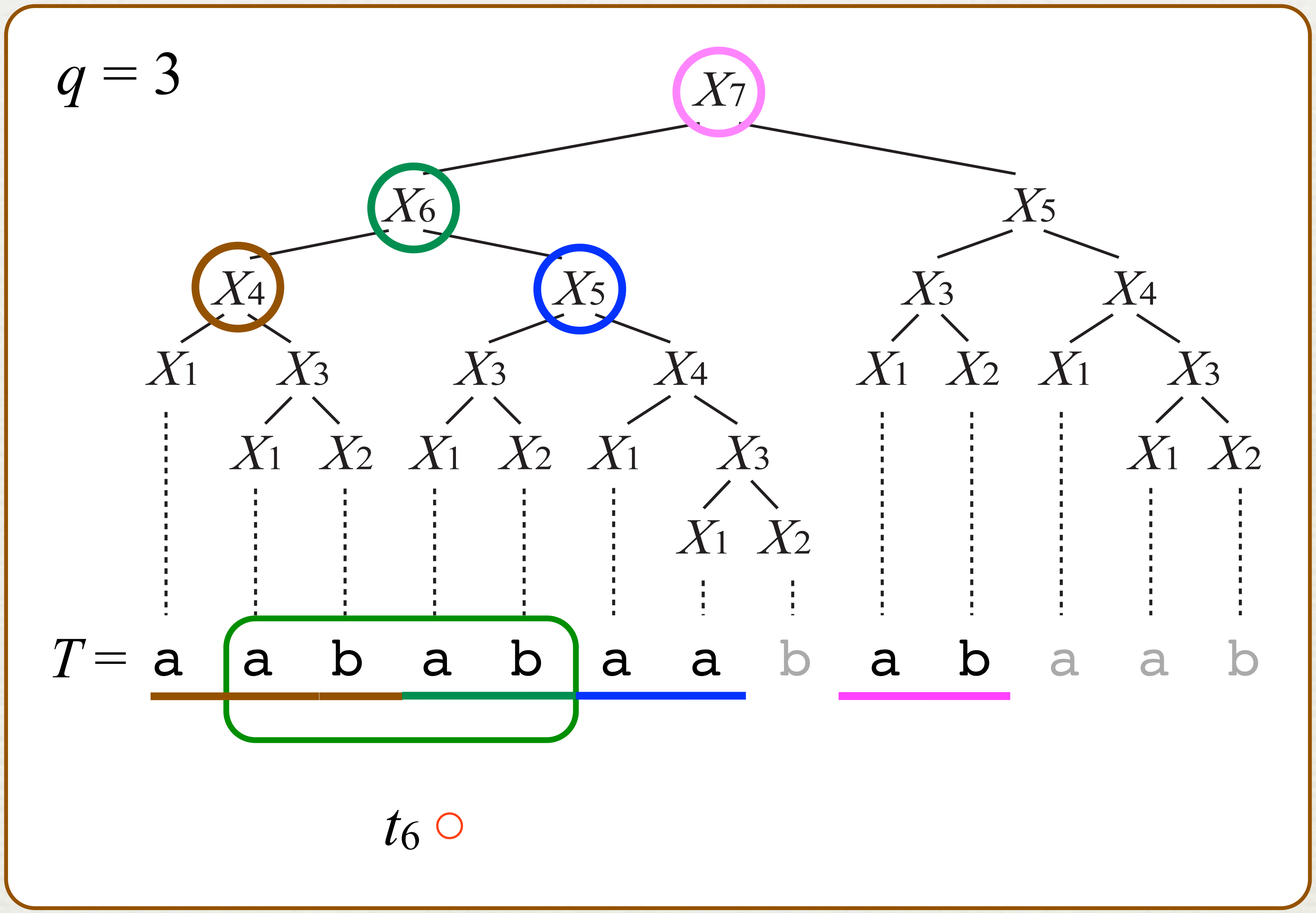
What we have left

- Compact representation of all t_i's



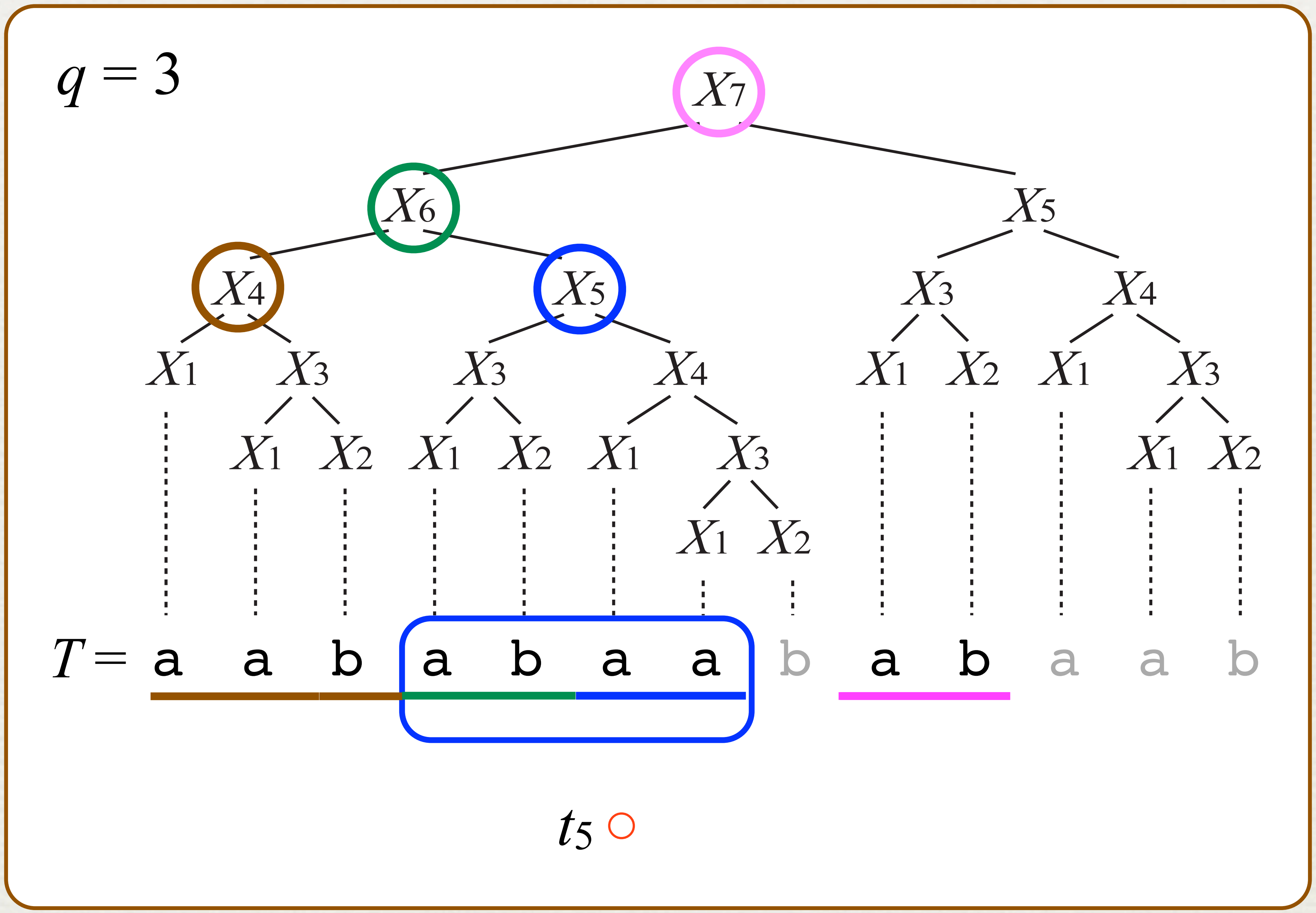
What we have left

- Compact representation of all t_i's



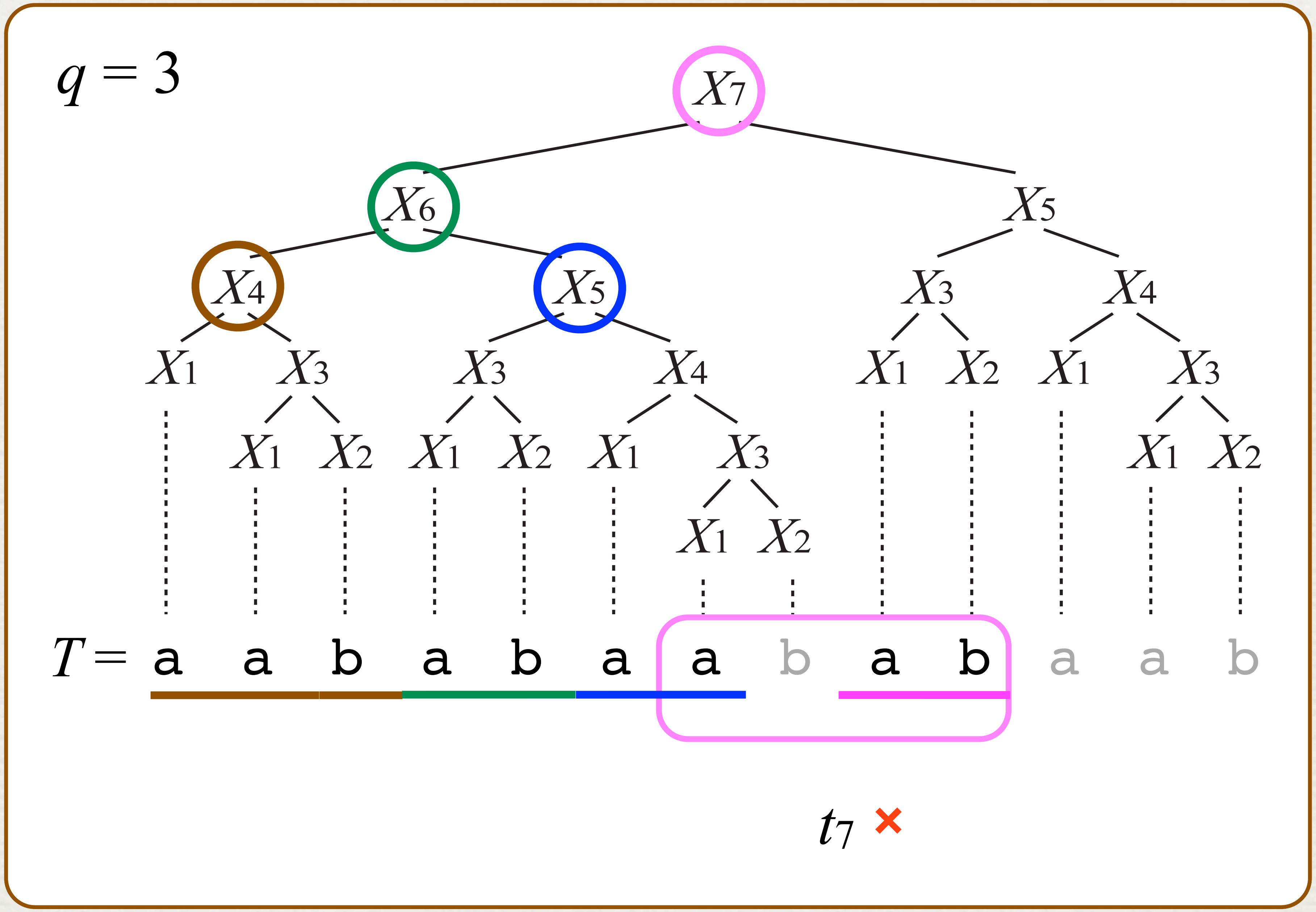
What we have left

- Compact representation of all t_i's



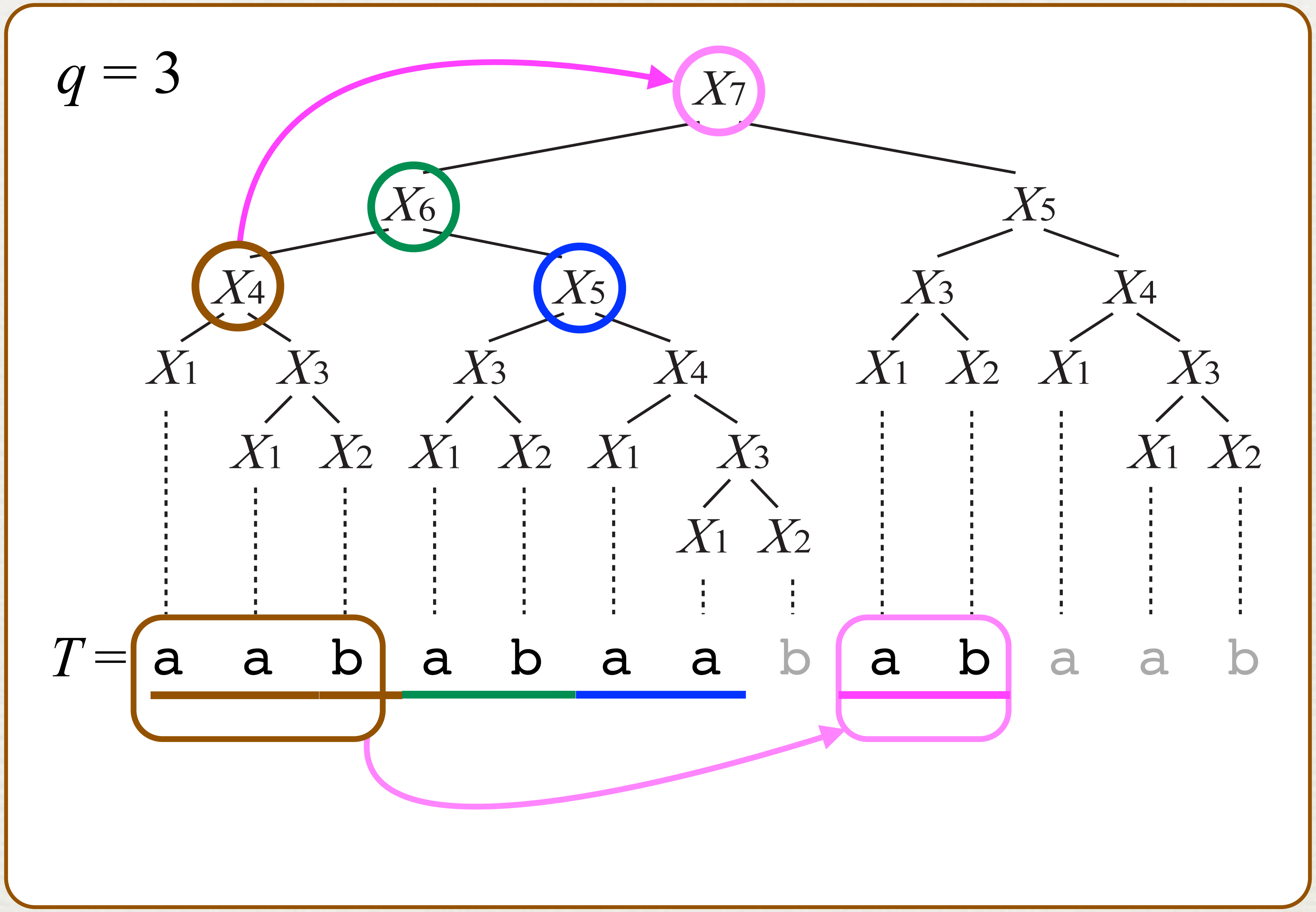
What we have left

- Compact representation of all t_i's



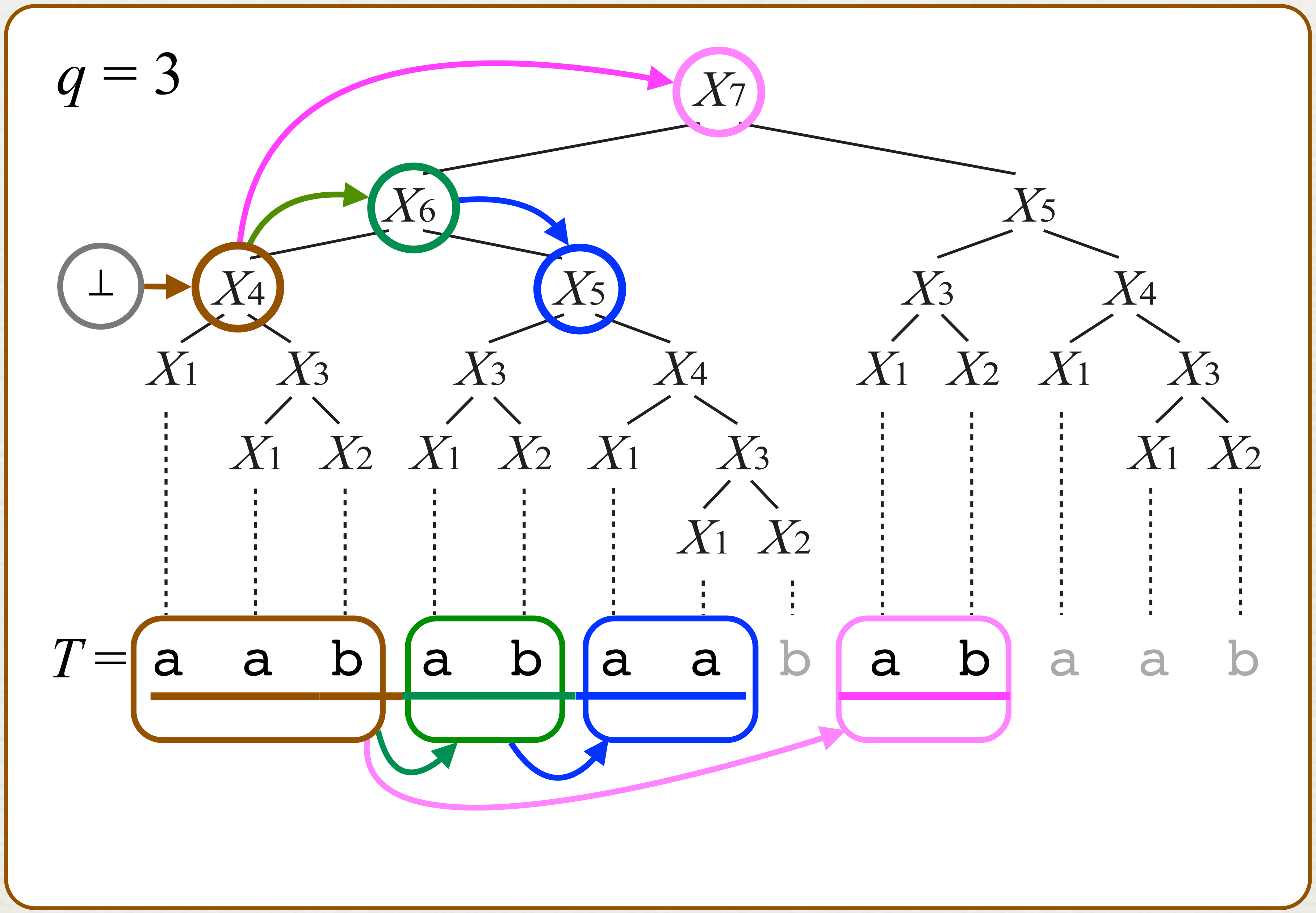
Neighbor tree

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



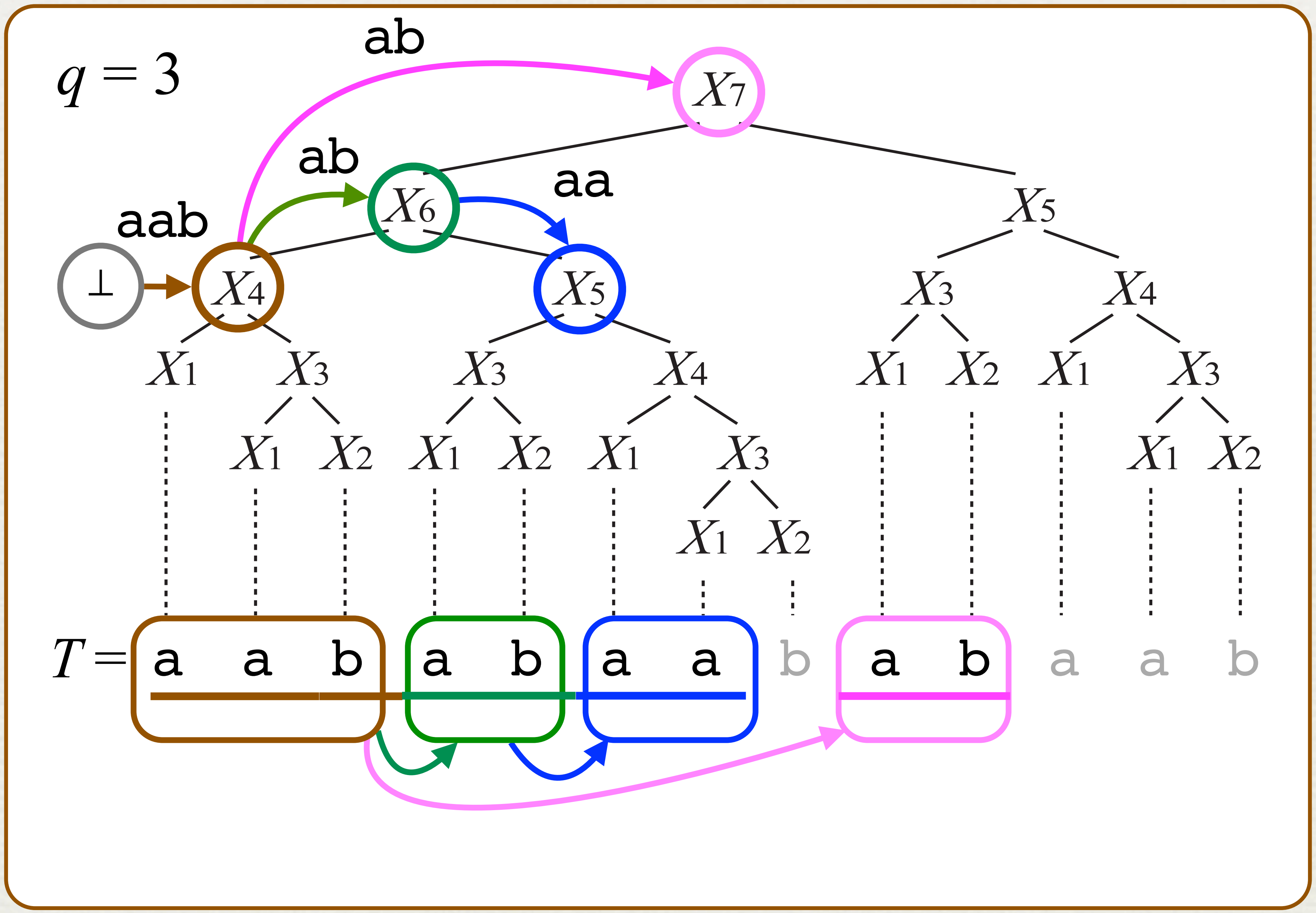
Neighbor tree

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



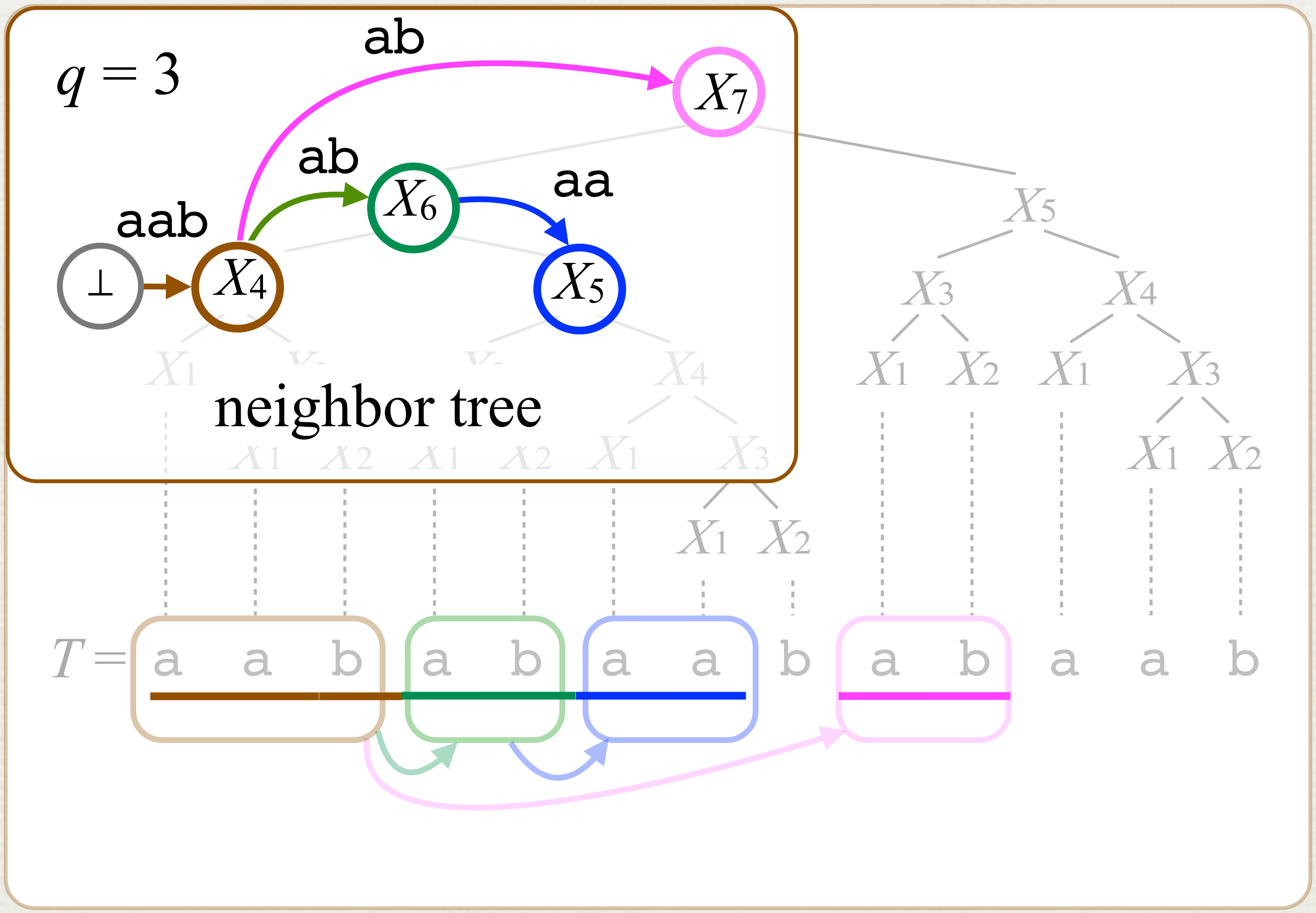
Neighbor tree

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



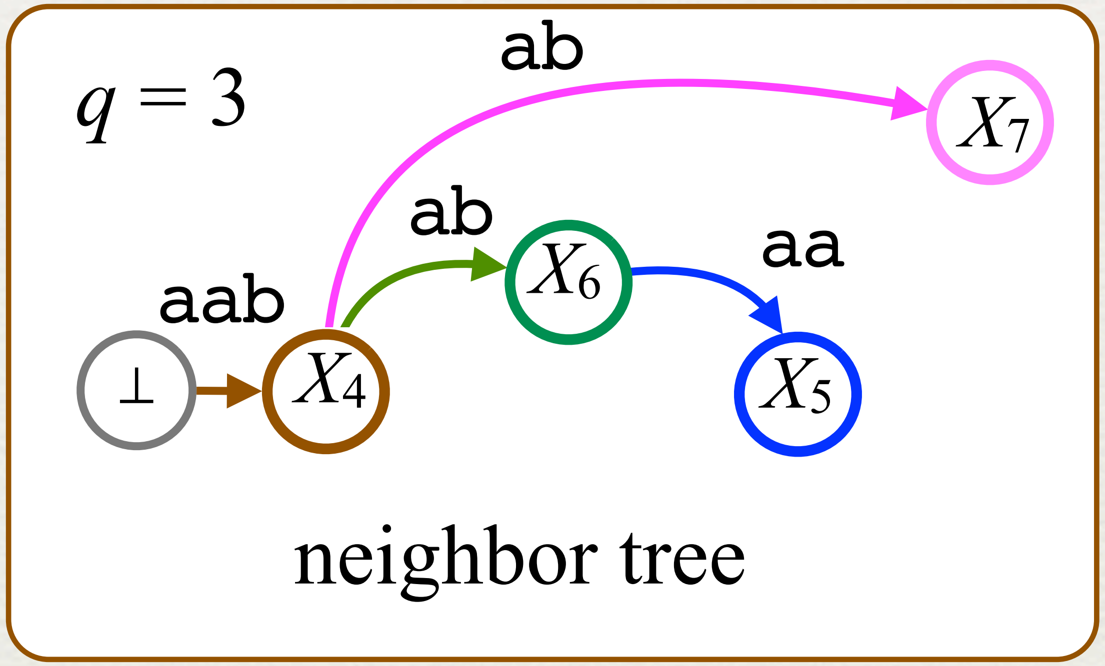
Neighbor tree

- 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

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

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

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 tree 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]

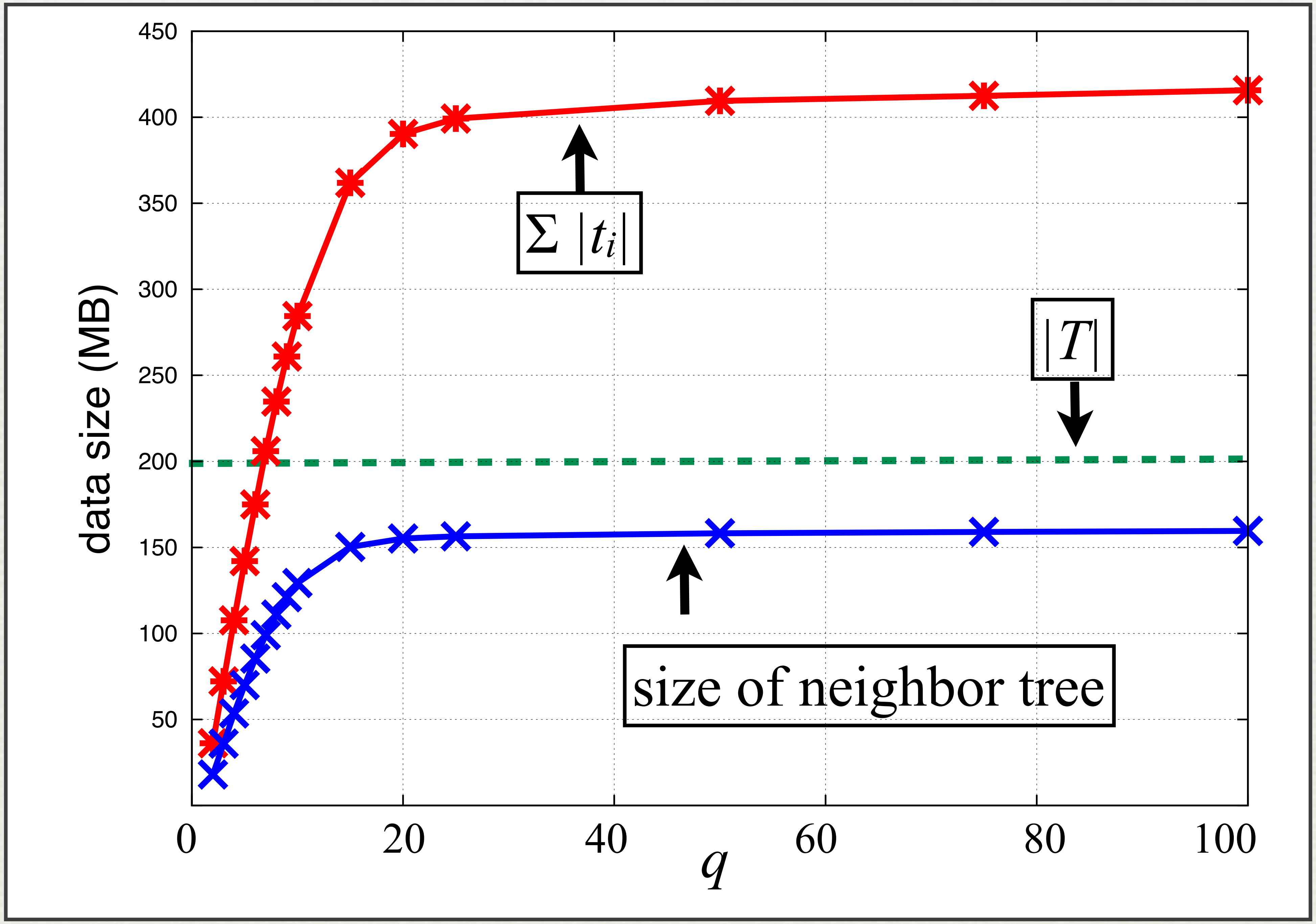
The suffix tree for a tree can be constructed in time linear in its size

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.

New Algorithm

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)

想定される質問

- Q.木を作る時，ラベルはどうやって計算するの？
  - A. right q-gram neighborの種類は２つ，neighborが子孫か祖先か．祖先に遷移してから次に祖先に遷移するまでのパスを考えると，各ノードのneighborは右辺のleft most pathの変数になる．つまりこのパスのラベルは最初の変数（一番祖先）右辺のprefixになる．仮定より，パスの各変数の左辺の長さはq-1以下，よって各変数の左辺をナイーブに展開すればパスのラベルを求めることが出来る．

想定される質問

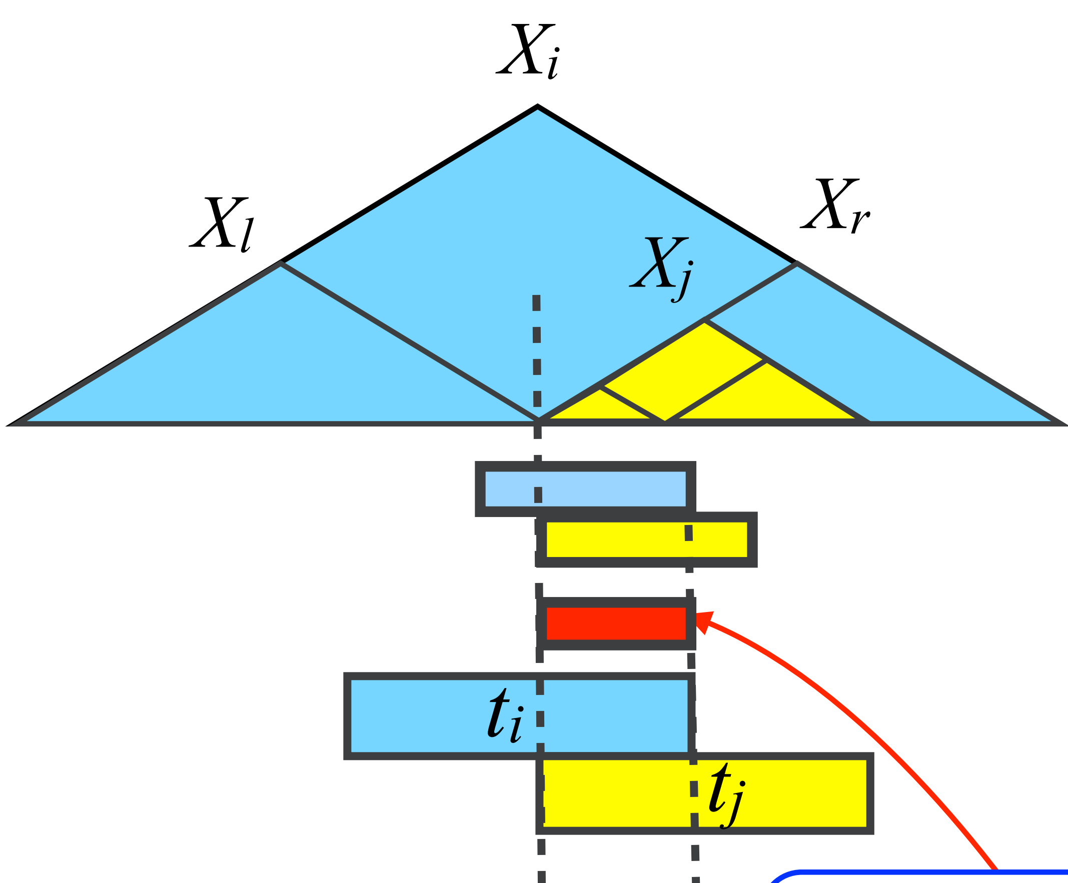
- Q.木を作る時，ラベルはどうやって計算するの？
  - A. right q-gram neighborの種類は２つ，neighborが子孫か祖先か．祖先に遷移してから次に祖先に遷移するまでのパスを考えると，各ノードのneighborは右辺のleft most pathの変数になる．つまりこのパスのラベルは最初の変数（一番祖先）右辺のprefixになる．仮定より，パスの各変数の左辺の長さはq-1以下，よって各変数の左辺をナイーブに展開すればパスのラベルを求めることが出来る．

Observation: relation of t_i's

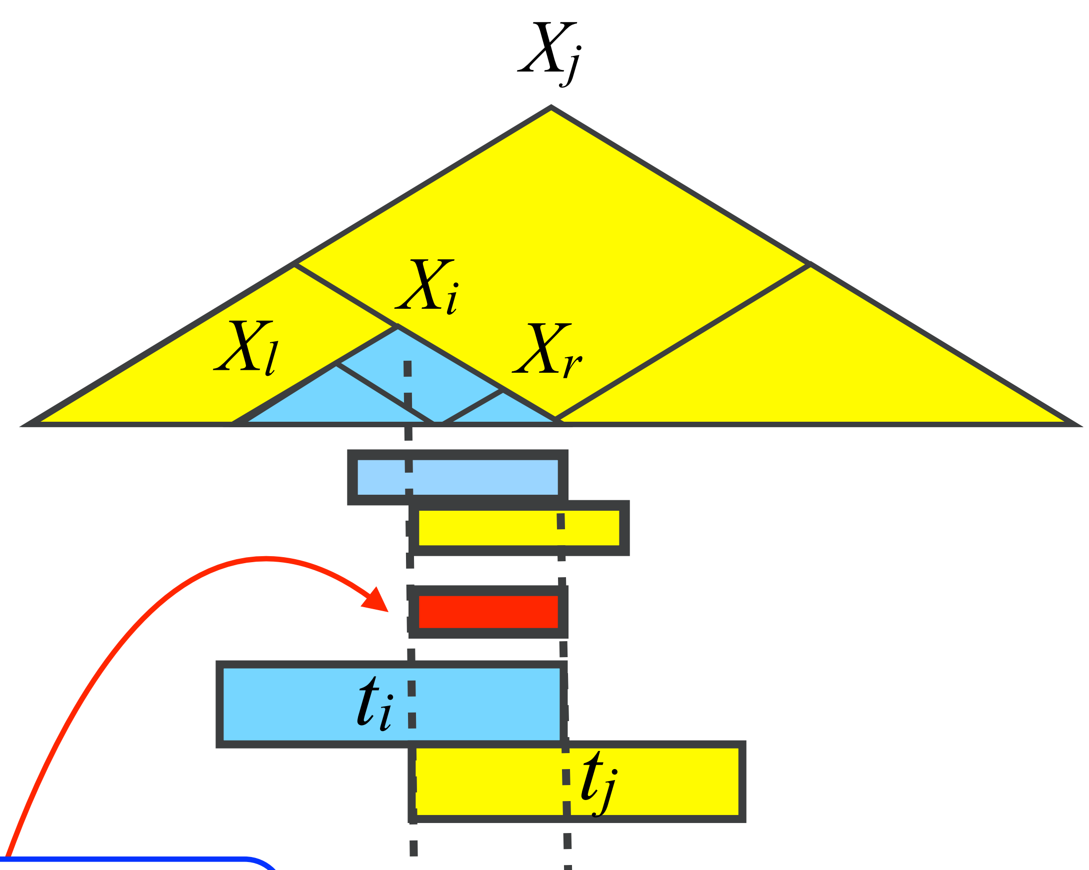
Observation

For any text position u,
if X_i stabs q-gram $T[u...u+q-1]$ and X_j stabs q-gram $T[u+1...u+q]$,
then t_i and t_j share the substring $t_j[1...q-1]$.

if X_i is an ancestor of X_j



if X_i is an descendant of X_j



t_i and t_j share the
(q-1)-length substring

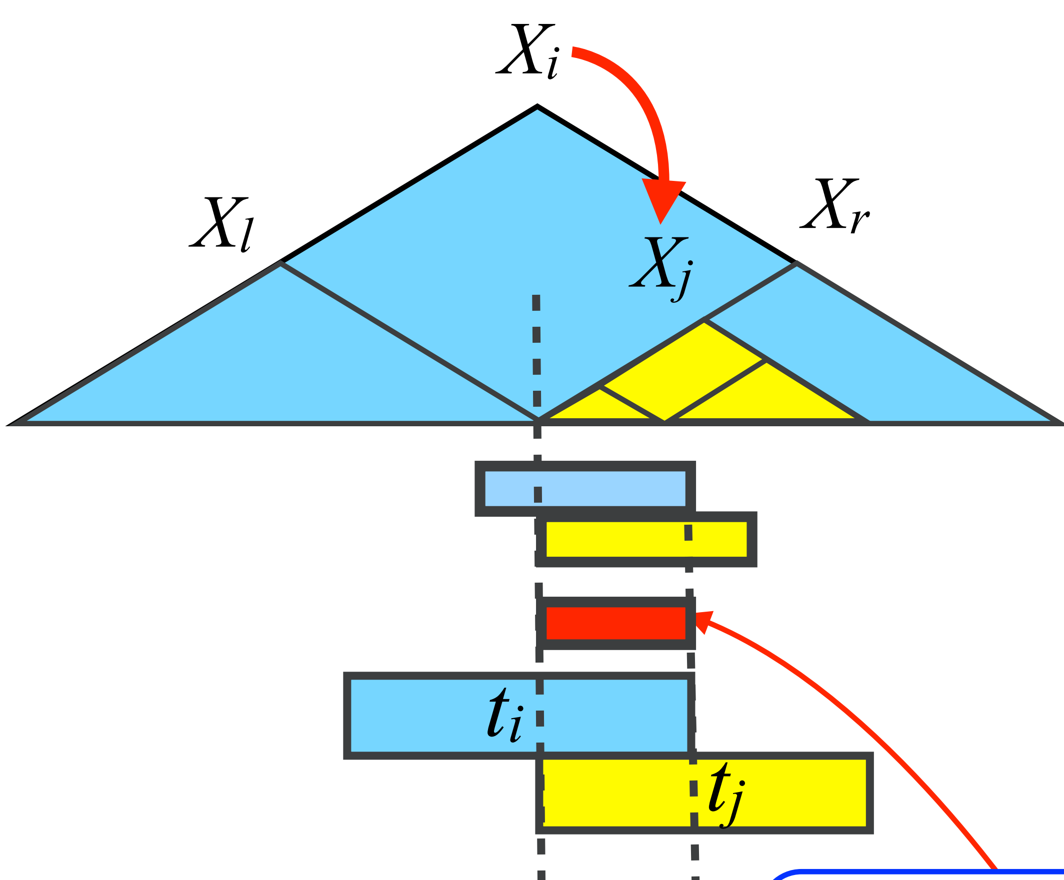
13年2月21日木曜日

right q-gram neighbor

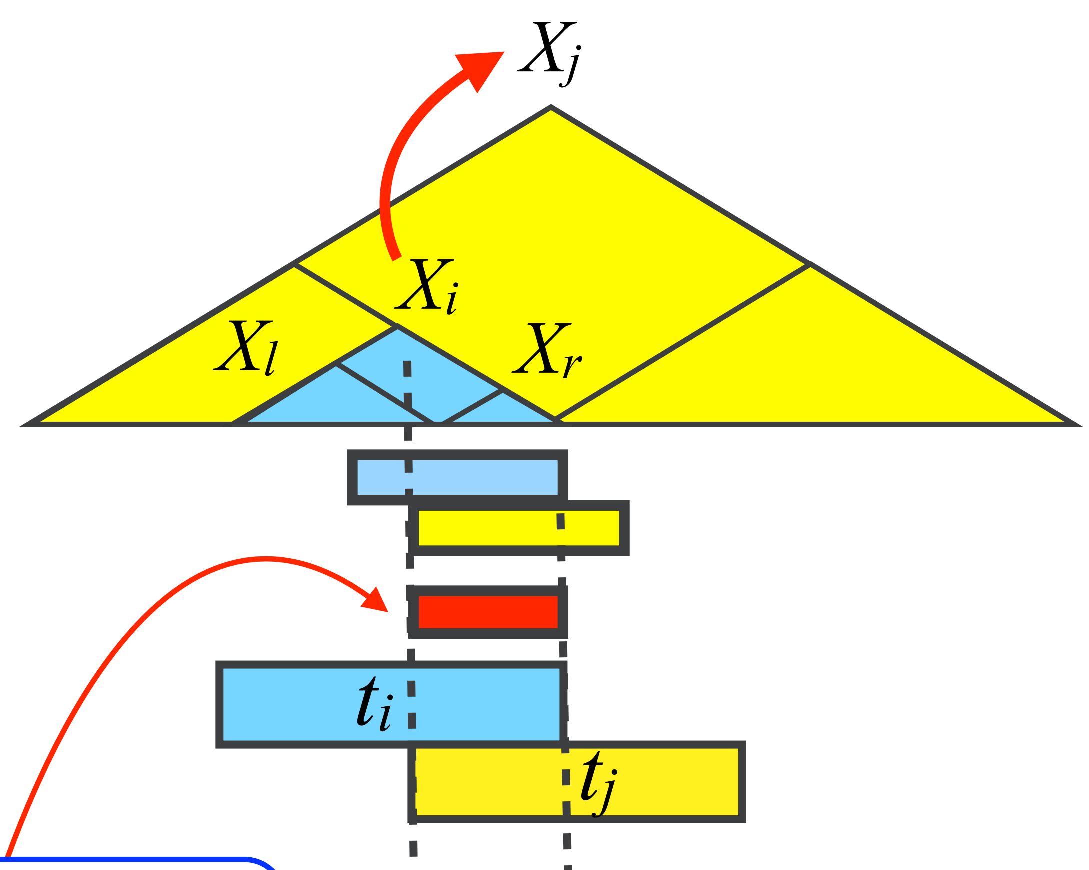
Definition

For any text position u, if X_i stabs q-gram $T[u...u+q-1]$ and X_j stabs q-gram $T[u+1...u+q]$,
We say X_j is a right q-gram neighbor of X_i.

if X_i is an ancestor of X_j



if X_i is an descendant of X_j



t_i and t_j share the
(q-1)-length substring

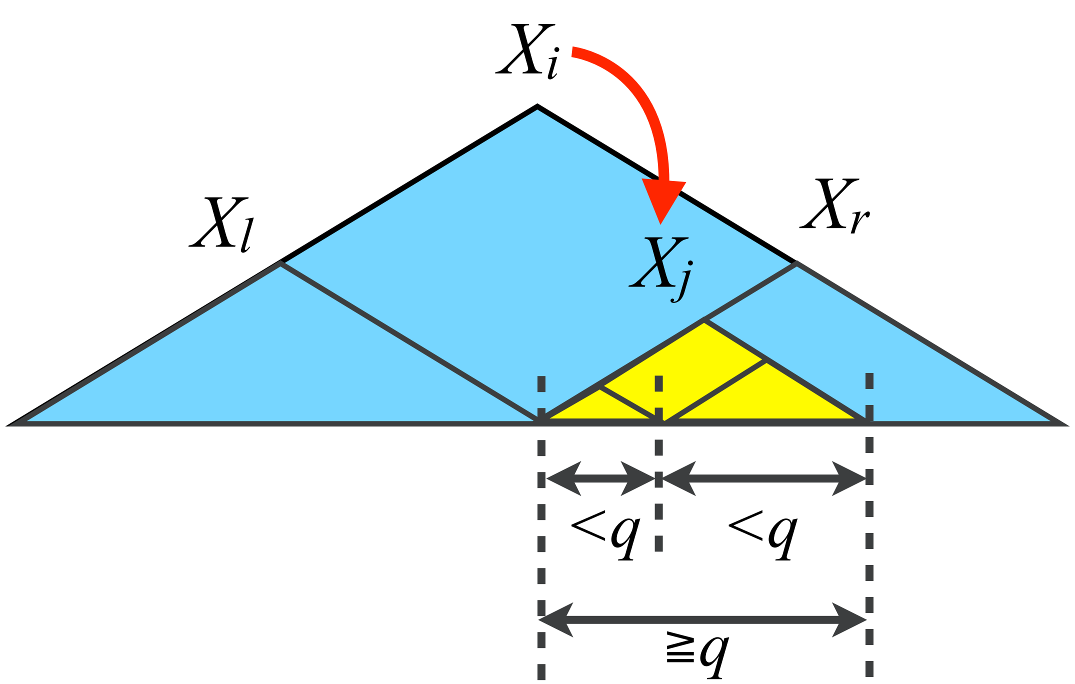
right q-gram neighbor

Definition

For any text position u, if X_i stabs q-gram $T[u...u+q-1]$ and X_j stabs q-gram $T[u+1...u+q]$,

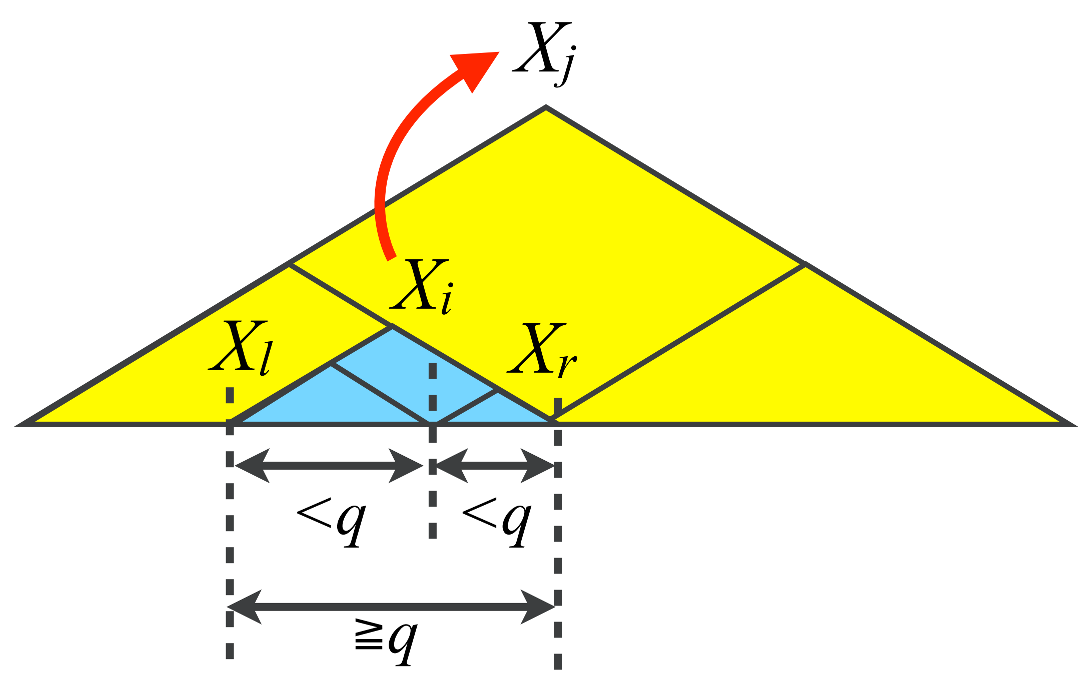
We say X_j is a right q-gram neighbor of X_i.

if X_i is an ancestor of X_j



X_j is the deepest variable on the left most path of X_r, whose length is at most q

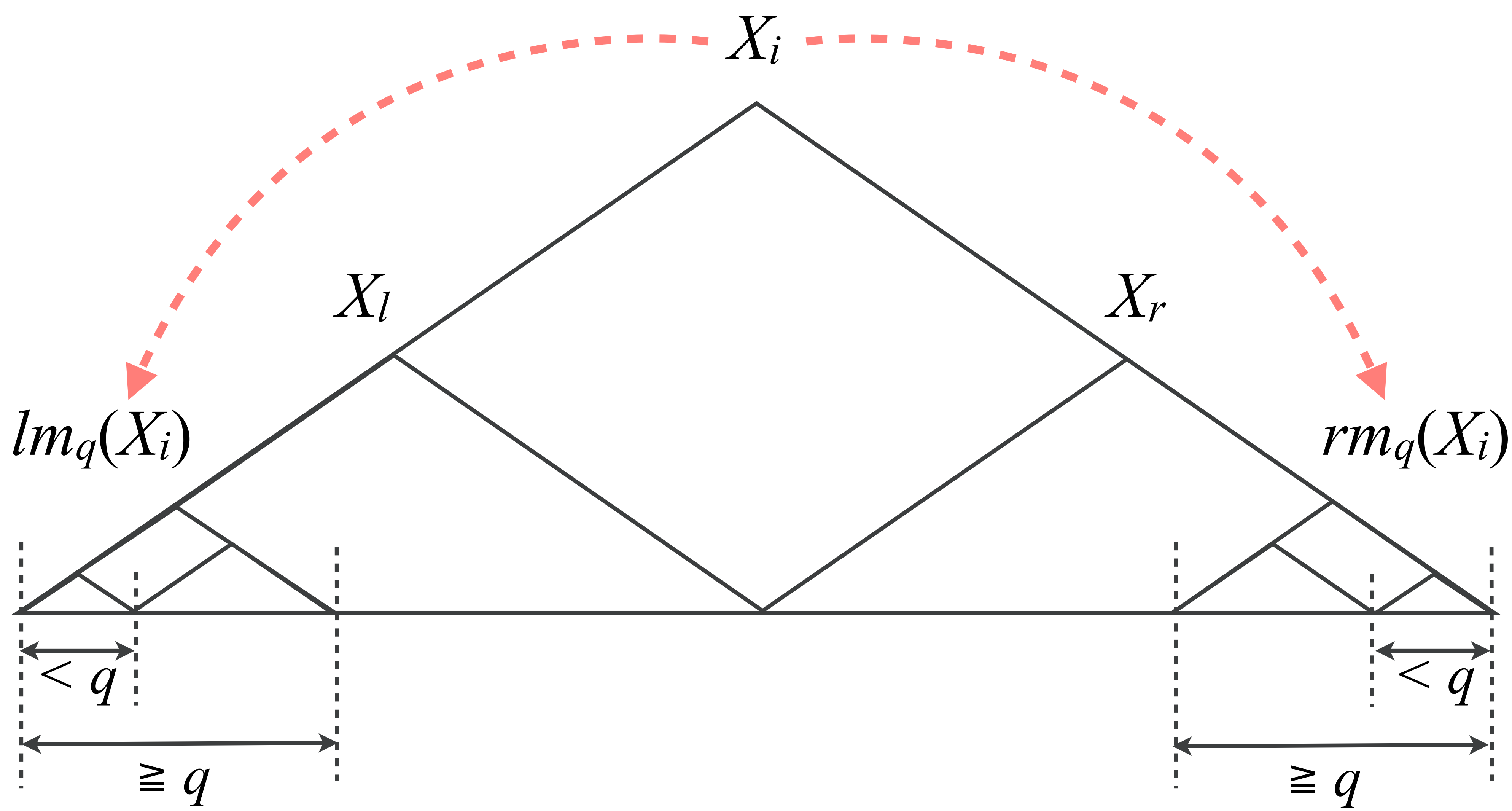
if X_i is an descendant of X_j



X_i is the deepest variable on the right most path of X_j, whose length is at most q

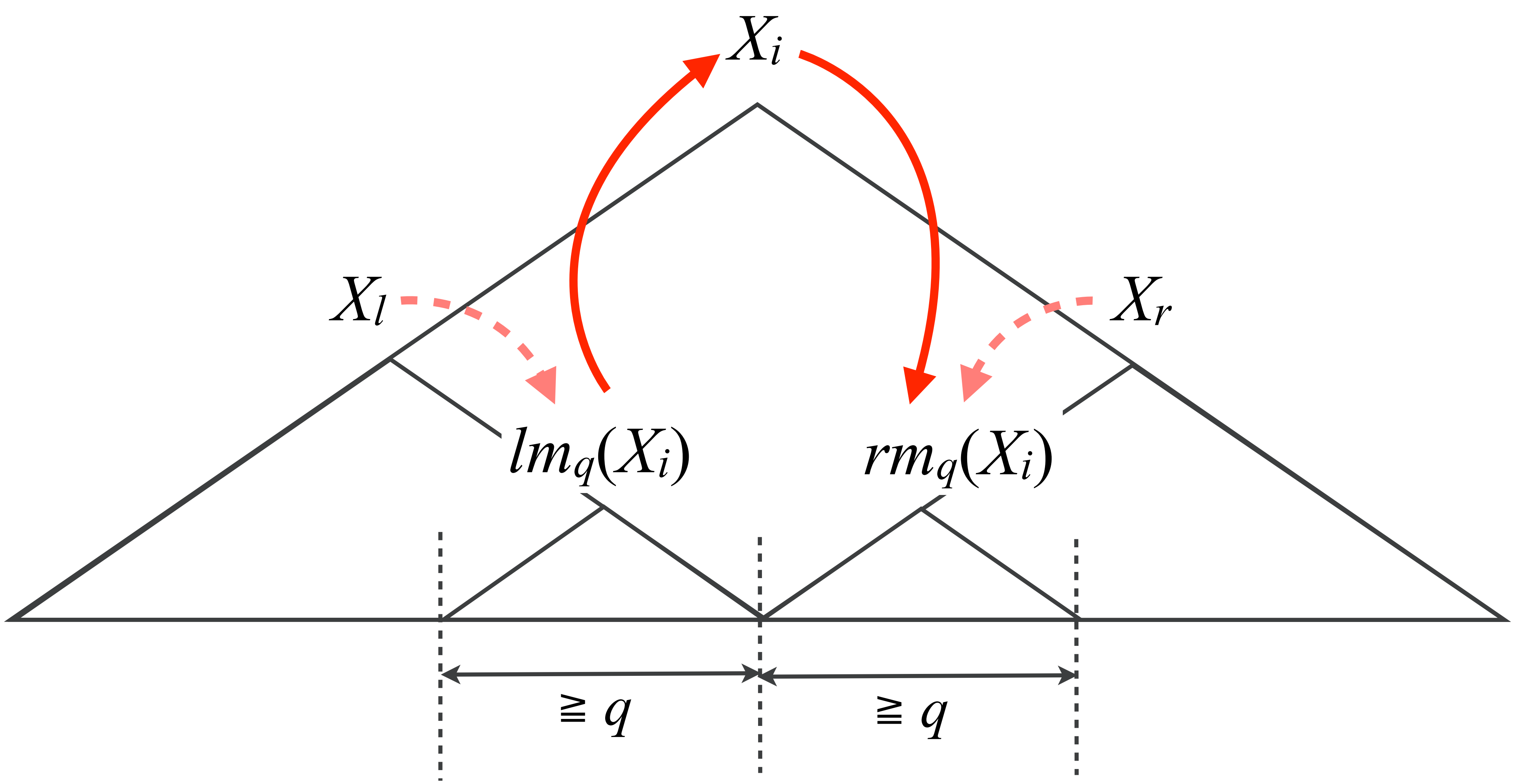
Finding the right q-gram neighbor

For all X_i, store the link from X_i to $lm_q(X_i)$, $rm_q(X_i)$ such that
- $lm_q(X_i)$ is the leftmost and deepest variable of X_i s.t $|X_{l(i)}| \geqq q$
- $rm_q(X_i)$ is the rightmost and deepest variable of X_i s.t $|X_{r(i)}| \geqq q$



Finding the right q-gram neighbor

For all X_i, store the link from X_i to $lm_q(X_i)$, $rm_q(X_i)$ such that
- $lm_q(X_i)$ is the leftmost and deepest variable of X_i s.t $|X_{l(i)}| \geqq q$
- $rm_q(X_i)$ is the rightmost and deepest variable of X_i s.t $|X_{r(i)}| \geqq q$

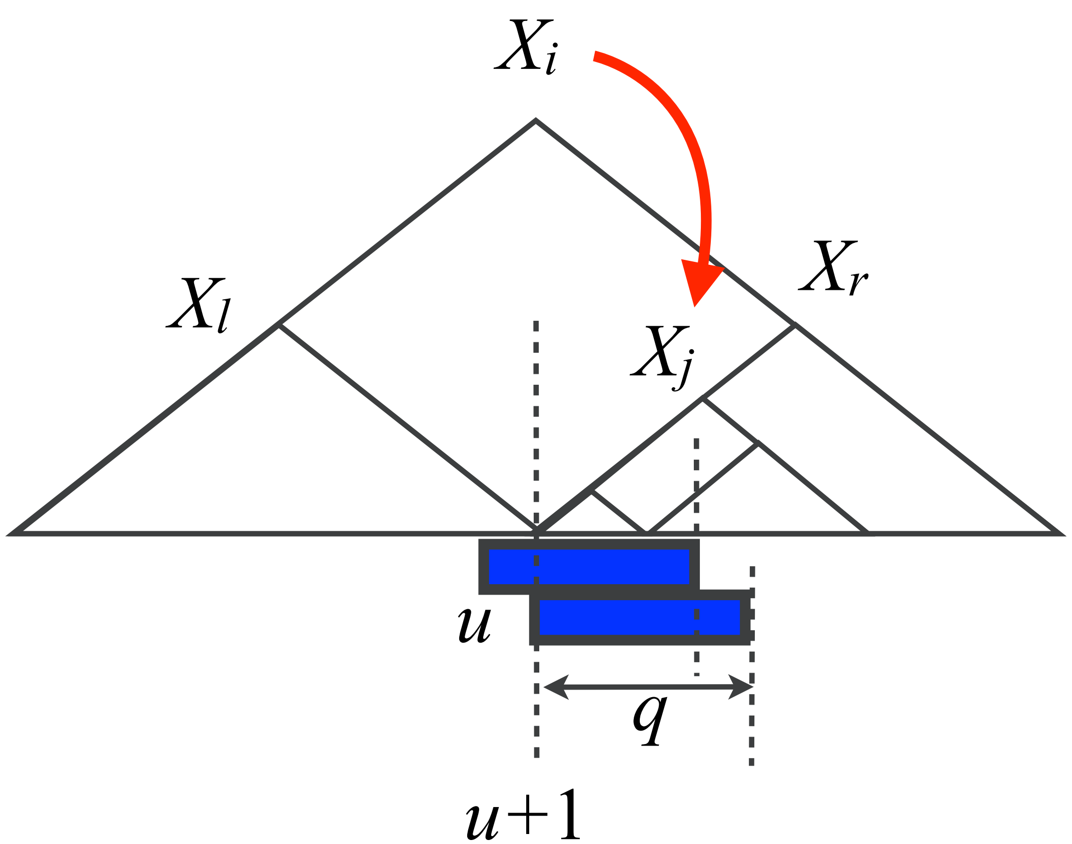


q-gram right neighbor can be constructed in $O(n)$ time and space

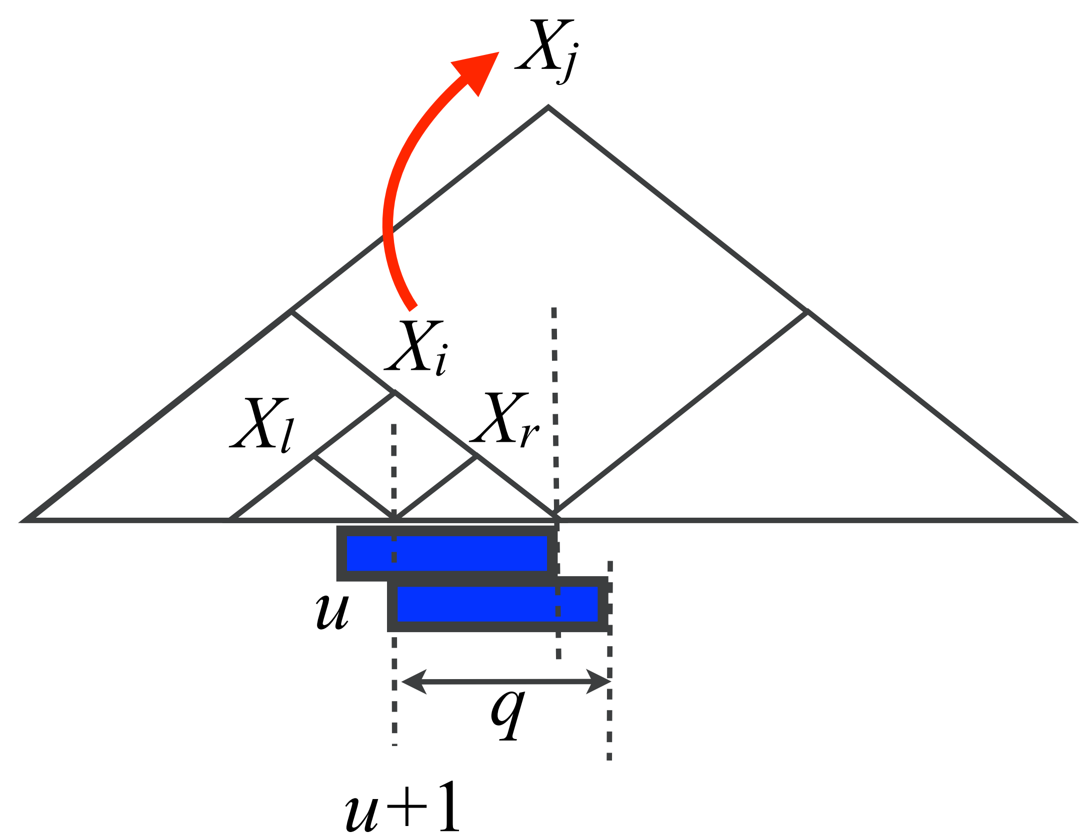
Important idea: q-gram neighbor

For some u, if X_i stabs q-gram $T[u...u+q-1]$ and
X_j stabs q-gram $T[u+1...u+q]$,
we say X_j is a q-gram neighbor of X_i

if $|X_r| \geqq q$



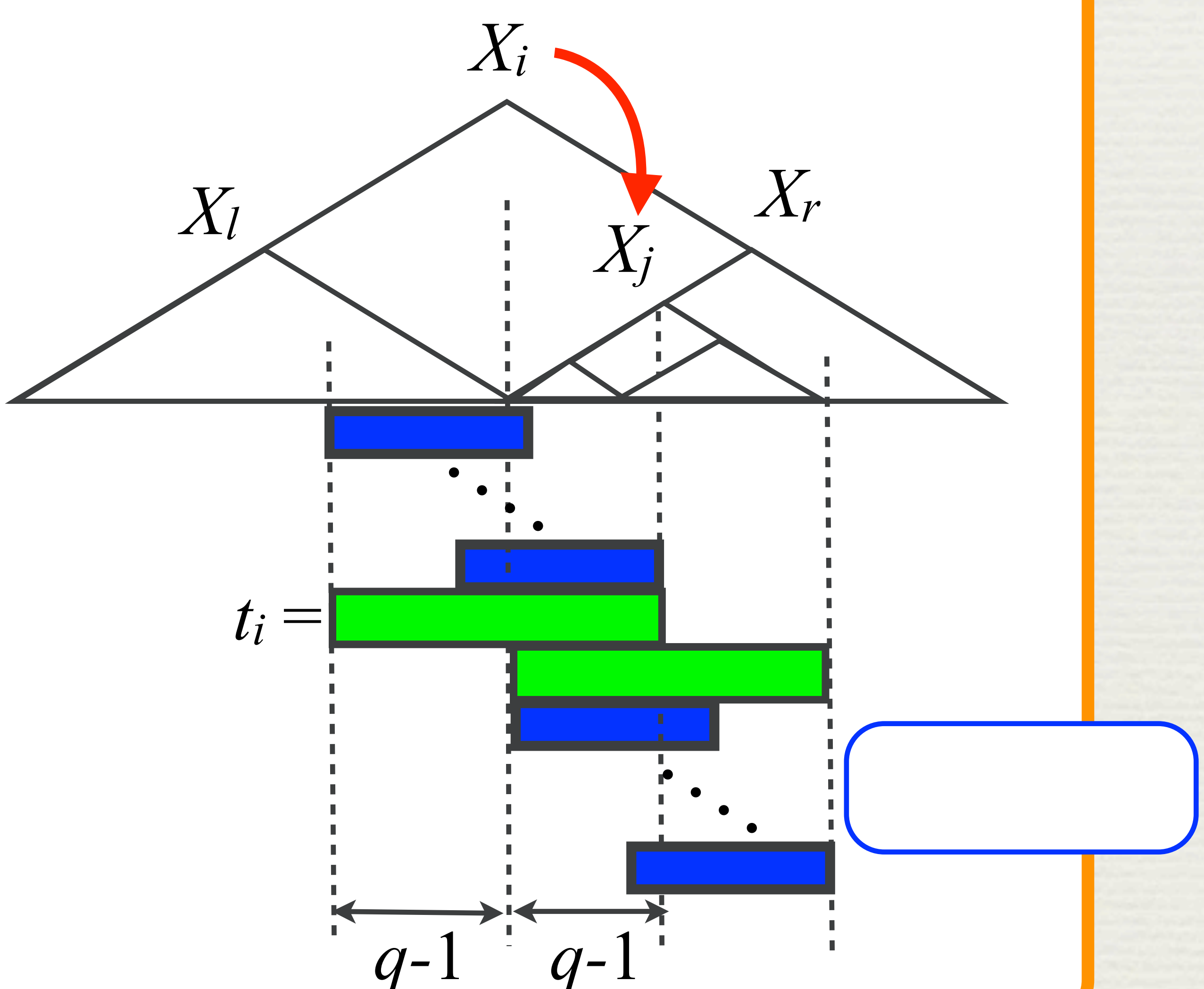
if $|X_r| < q$



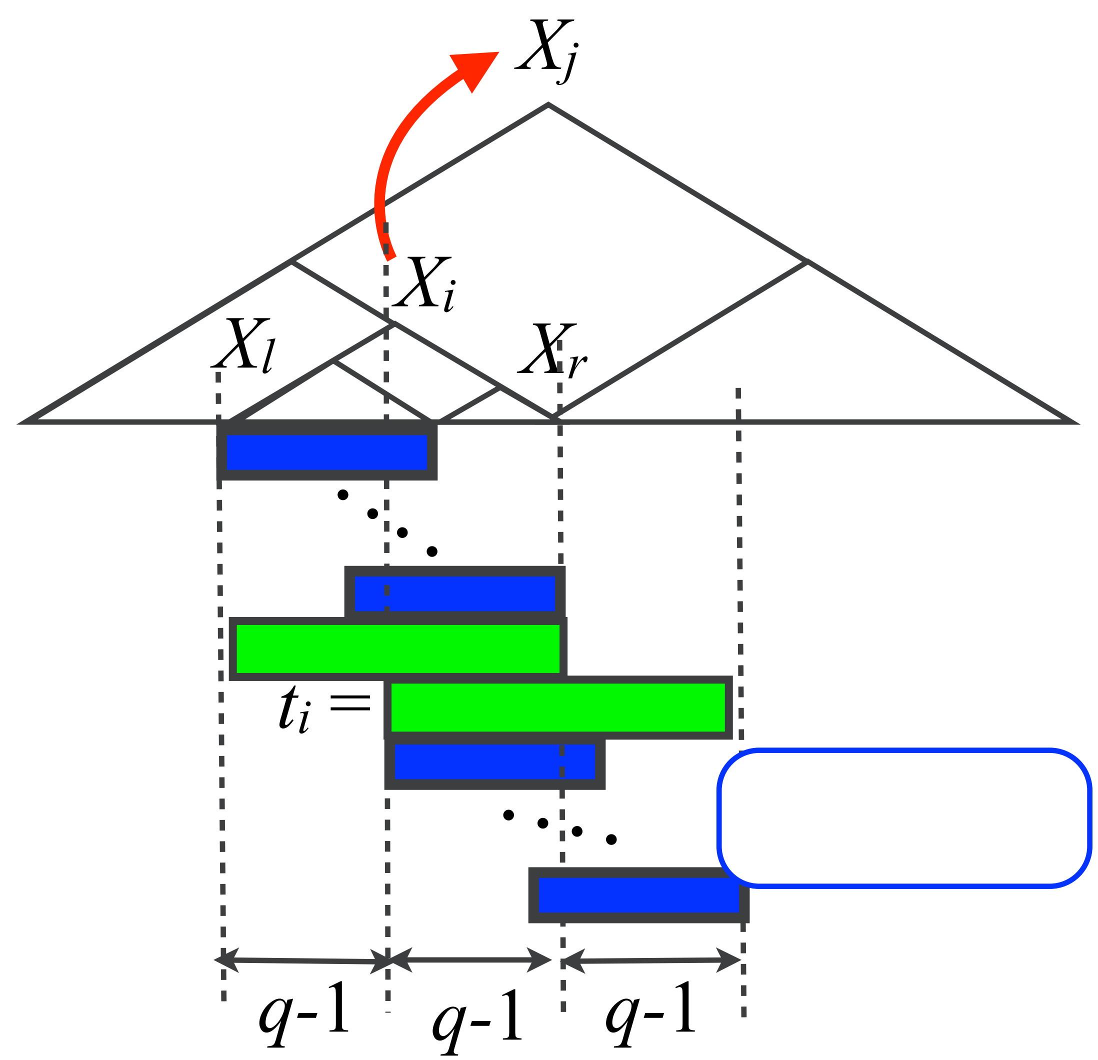
Important idea: q-gram neighbor

For X_i and its q-gram neighbor X_j, the length of overlap between t_i and t_j is q-1.

if $|X_r| \geqq q$

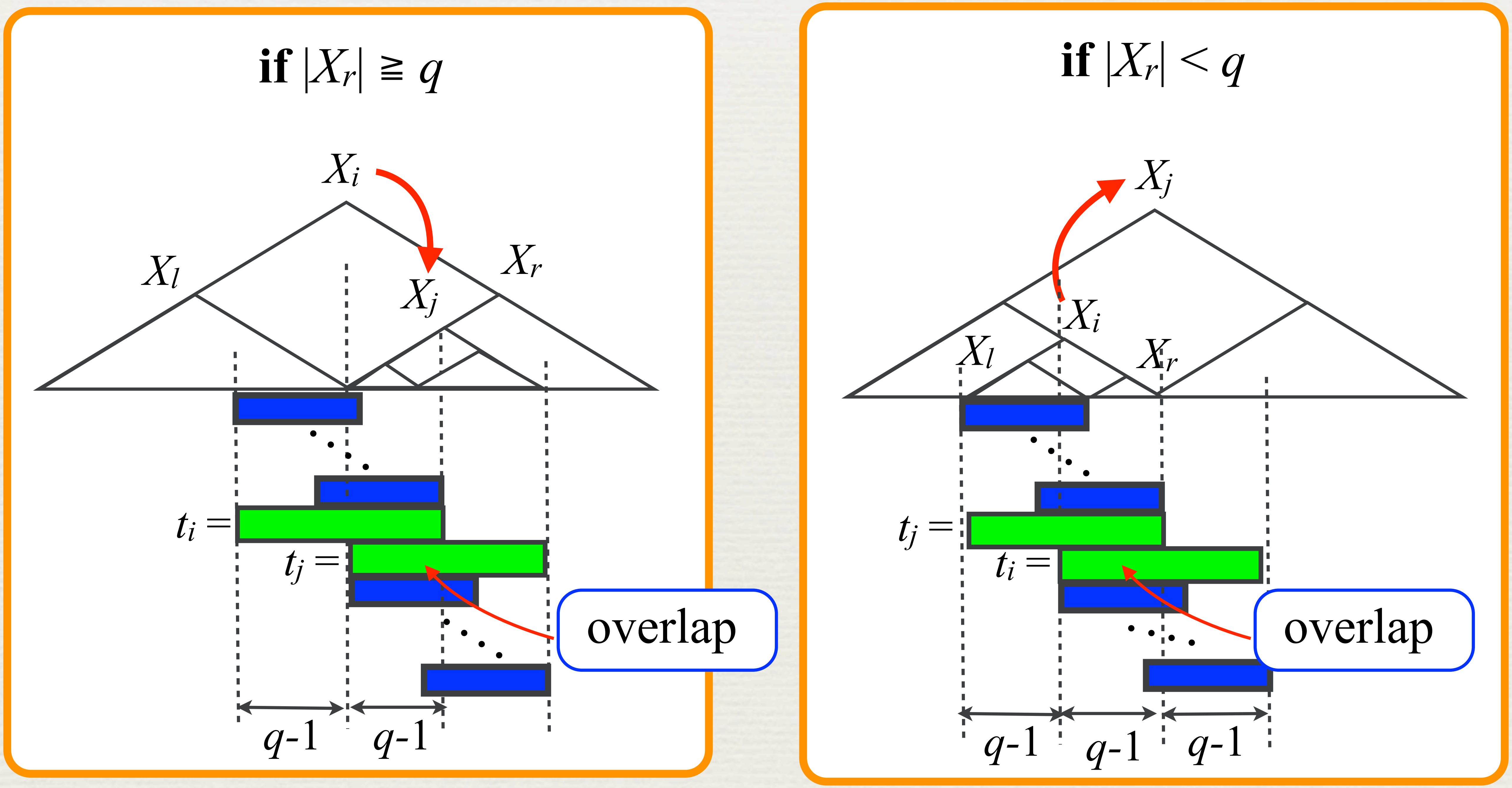


if $|X_r| < q$



Important idea: q-gram neighbor

For X_i and its q-gram neighbor X_j, the length of overlap between t_i and t_j is q-1.

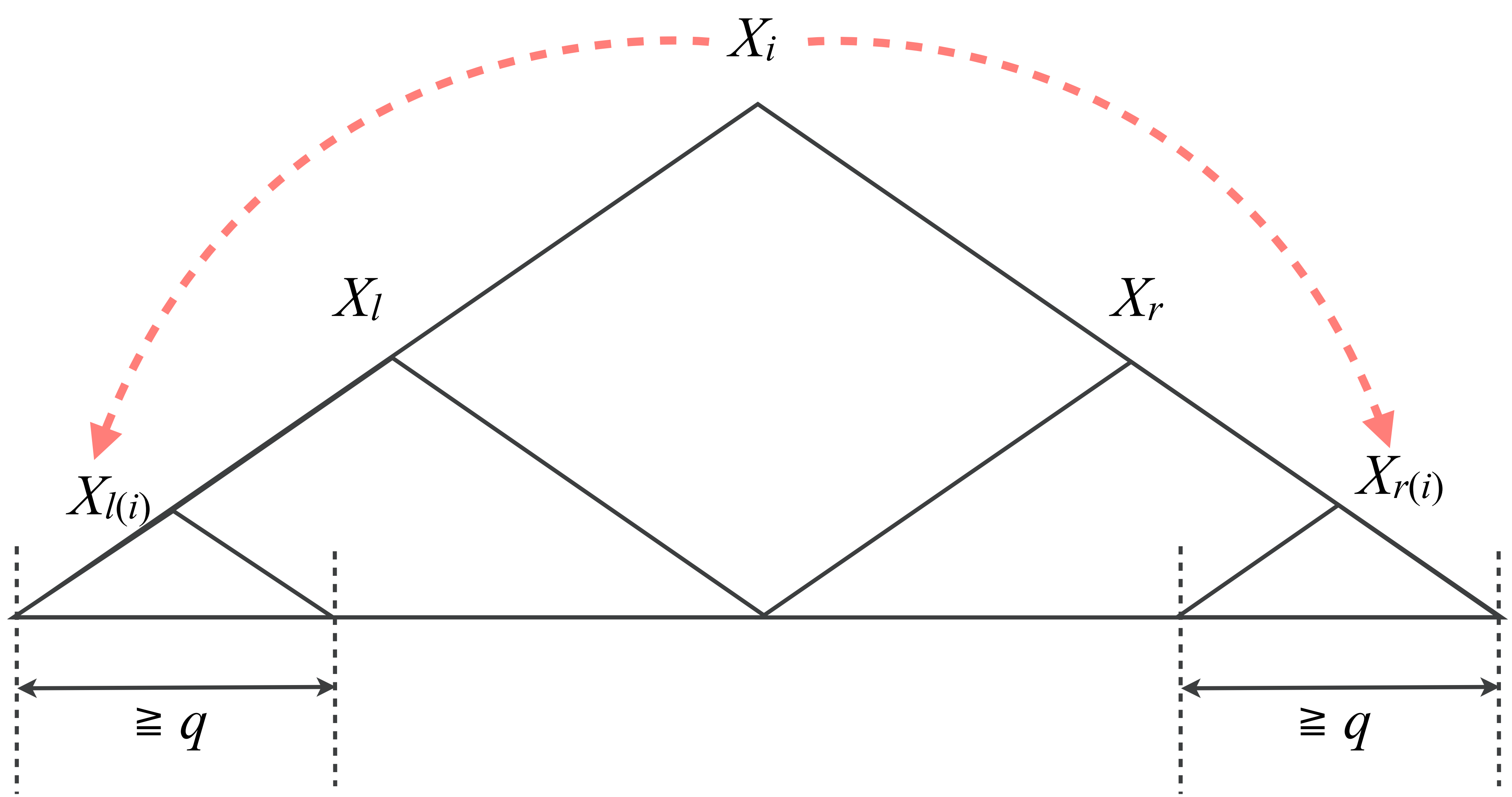


New Algorithm

Constructing q-gram neighbor graph

For all X_i, Store the link from X_i to $X_{l(i)}$, $X_{r(i)}$ such that

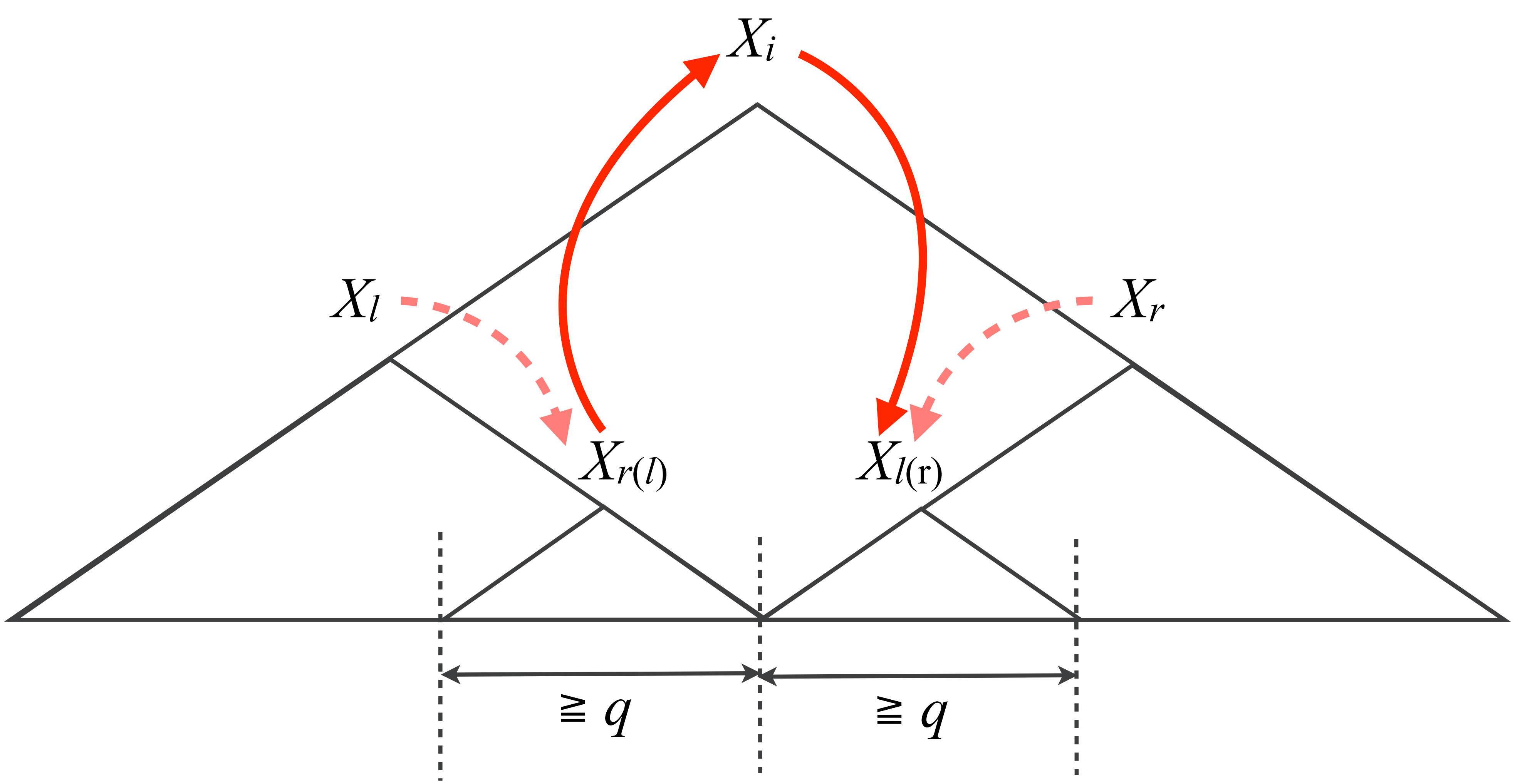
- $X_{l(i)}$ is the leftmost and deepest variable of X_i s.t $|X_{l(i)}| \geqq q$
- $X_{r(i)}$ is the rightmost and deepest variable of X_i s.t $|X_{r(i)}| \geqq q$



New Algorithm

Constructing q-gram neighbor graph

For all X_i, Store the link from X_i to $X_{l(i)}$, $X_{r(i)}$ such that
- $X_{l(i)}$ is the leftmost and deepest variable of X_i s.t $|X_{l(i)}| \geqq q$
- $X_{r(i)}$ is the rightmost and deepest variable of X_i s.t $|X_{r(i)}| \geqq q$



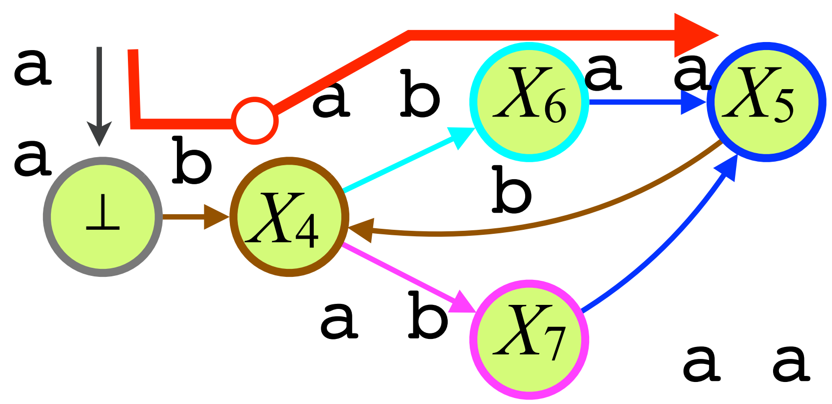
q-gram right neighbor can be constructed in $O(n)$ time and space

New Algorithm

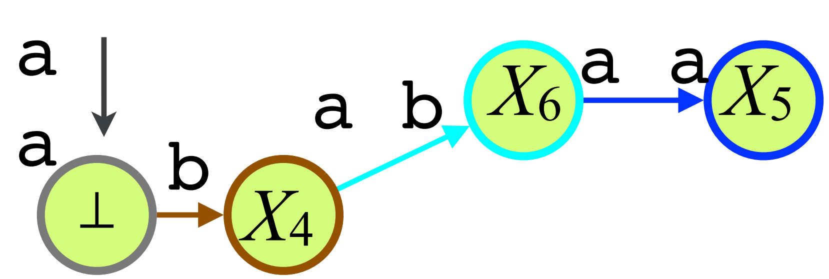
Constructing q-gram neighbor tree

Doing the DFS in neighbor graph,
neighbor tree can be constructed in $O(n)$ time

neighbor graph G



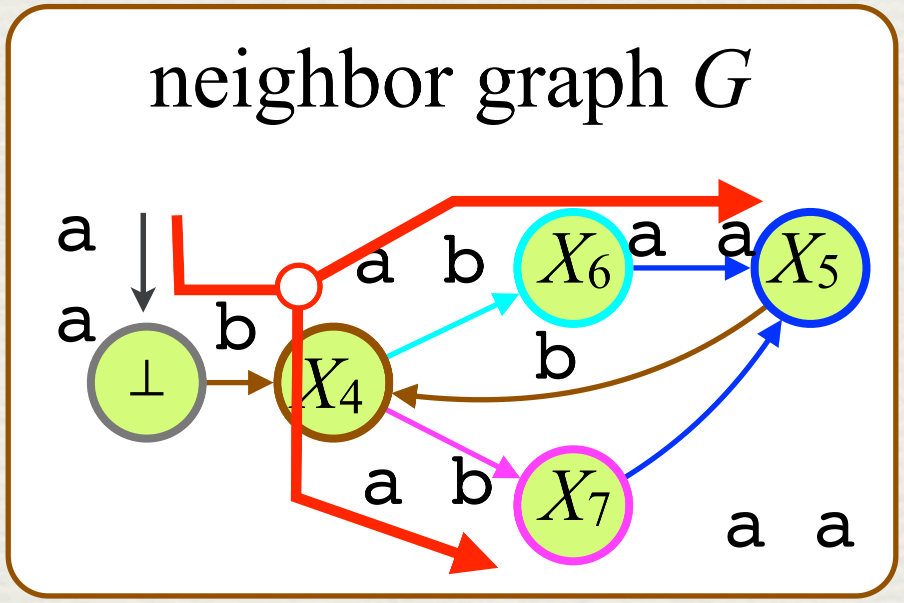
neighbor tree of G

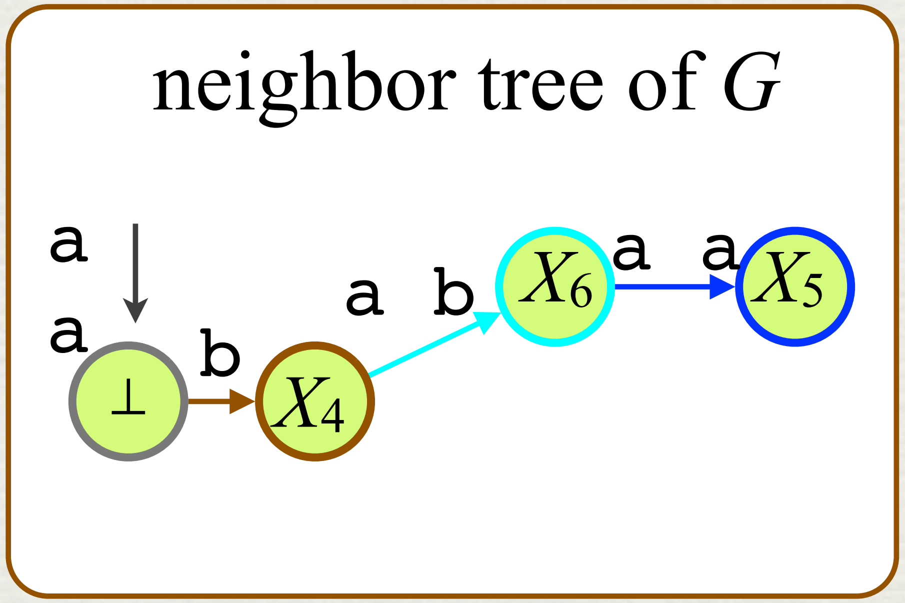


New Algorithm

Constructing *q*-gram neighbor tree

Doing the DFS in neighbor graph,
neighbor tree can be constructed in $O(n)$ time

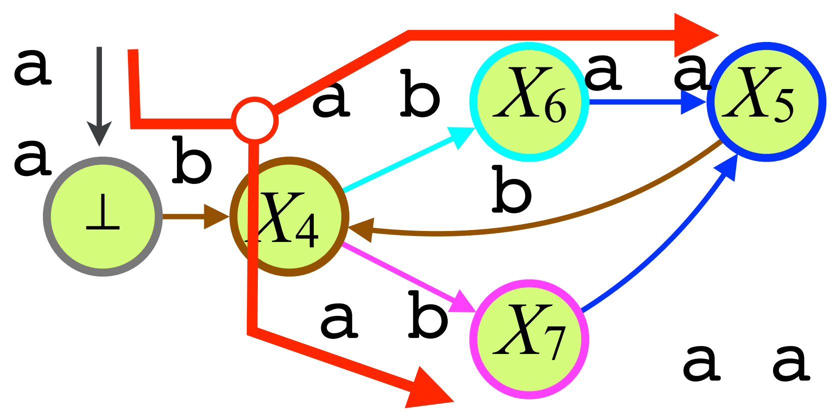




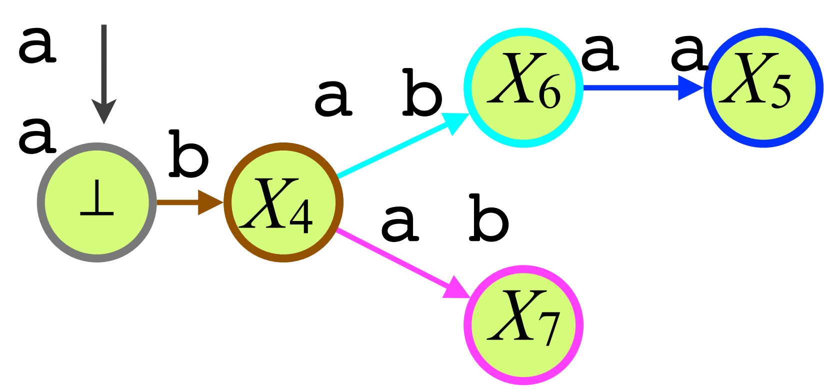
Constructing q-gram neighbor tree

Doing the DFS in neighbor graph,
neighbor tree can be constructed in $O(n)$ time

neighbor graph G



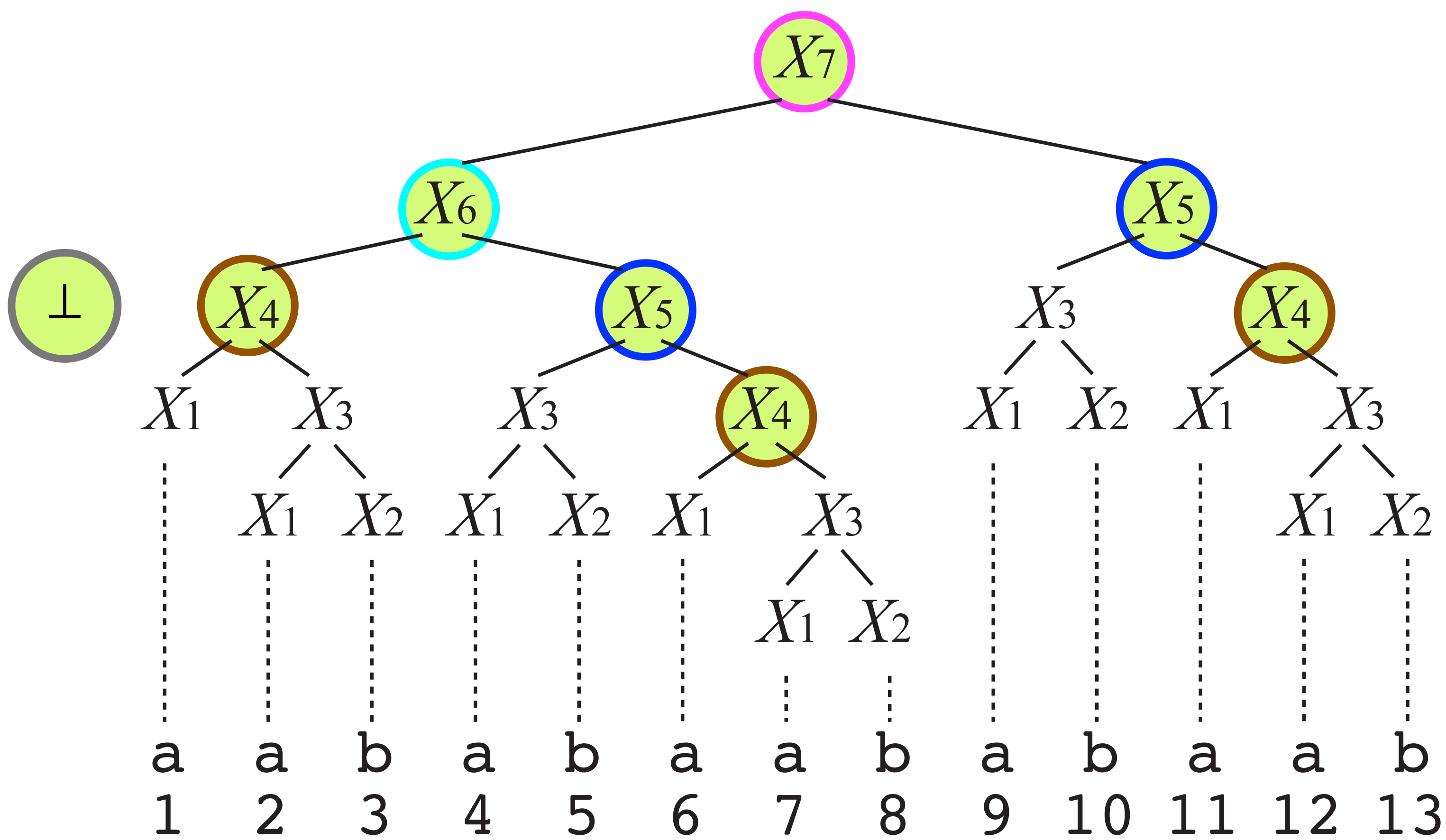
neighbor tree of G



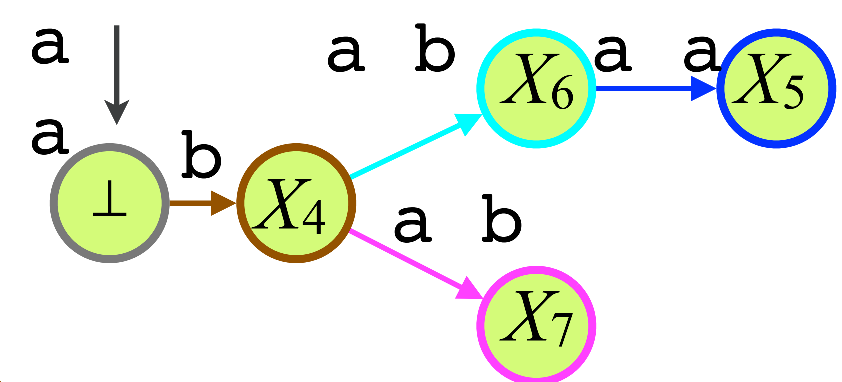
Calculating neighbor tree label

Label can be computed by decompressing prefix of variables

3-gram neighbor in derivation tree



neighbor tree of G

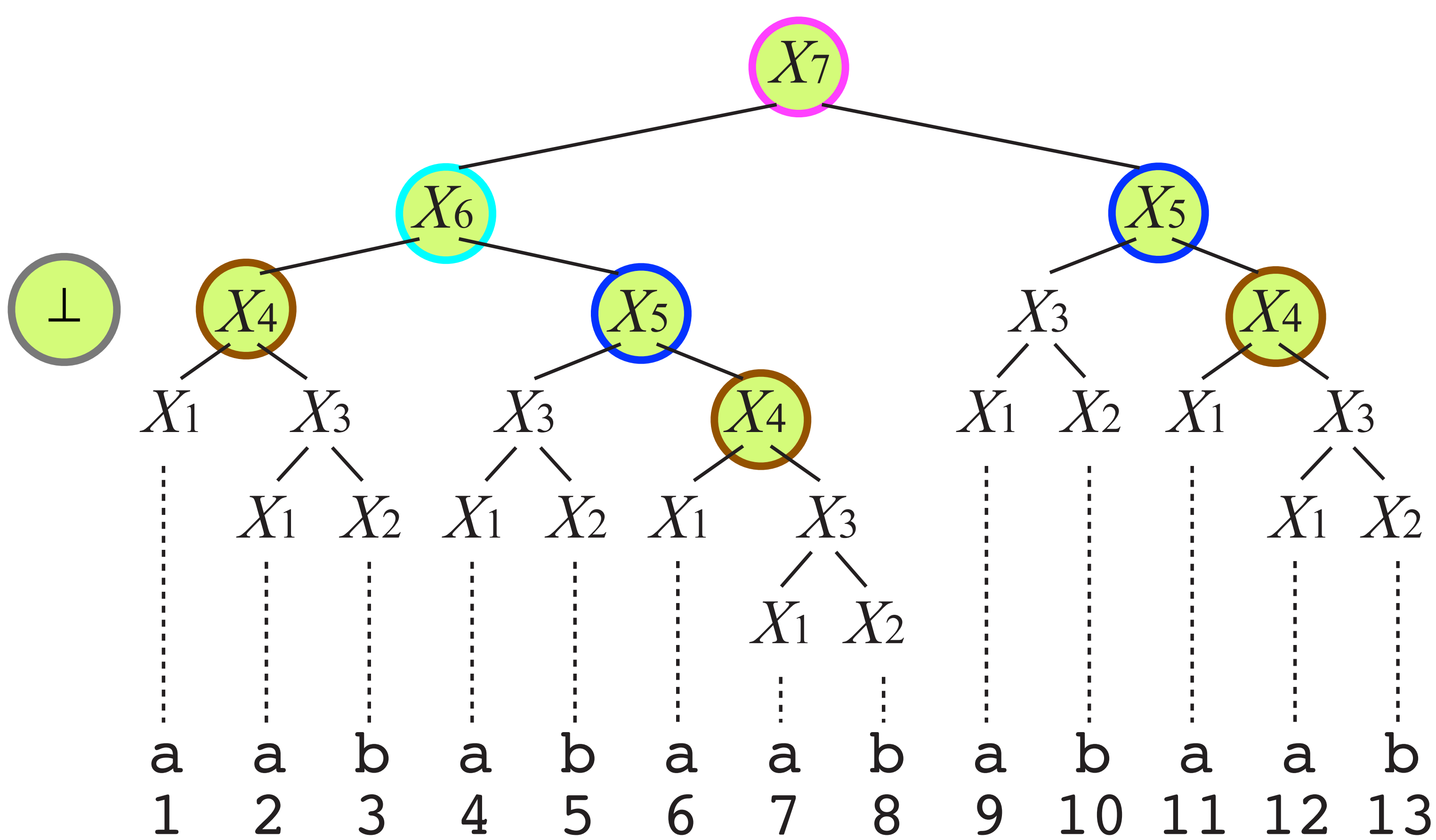


We compute the label by decompressing prefix of variables

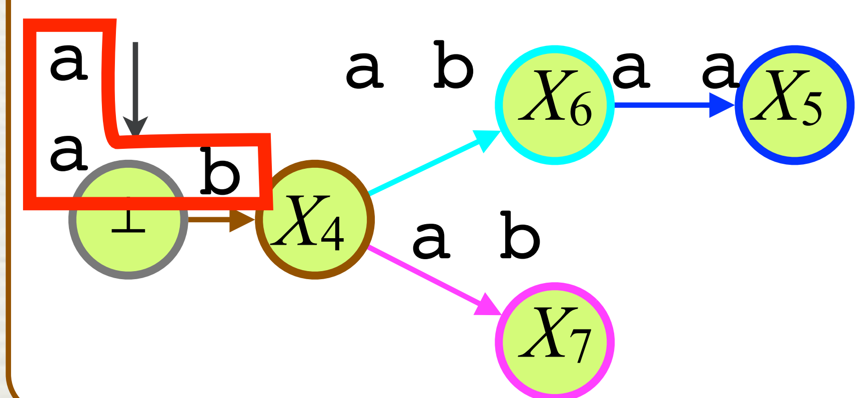
Calculating neighbor tree label

Label can be computed by decompressing prefix of variables

3-gram neighbor in derivation tree



neighbor tree of G



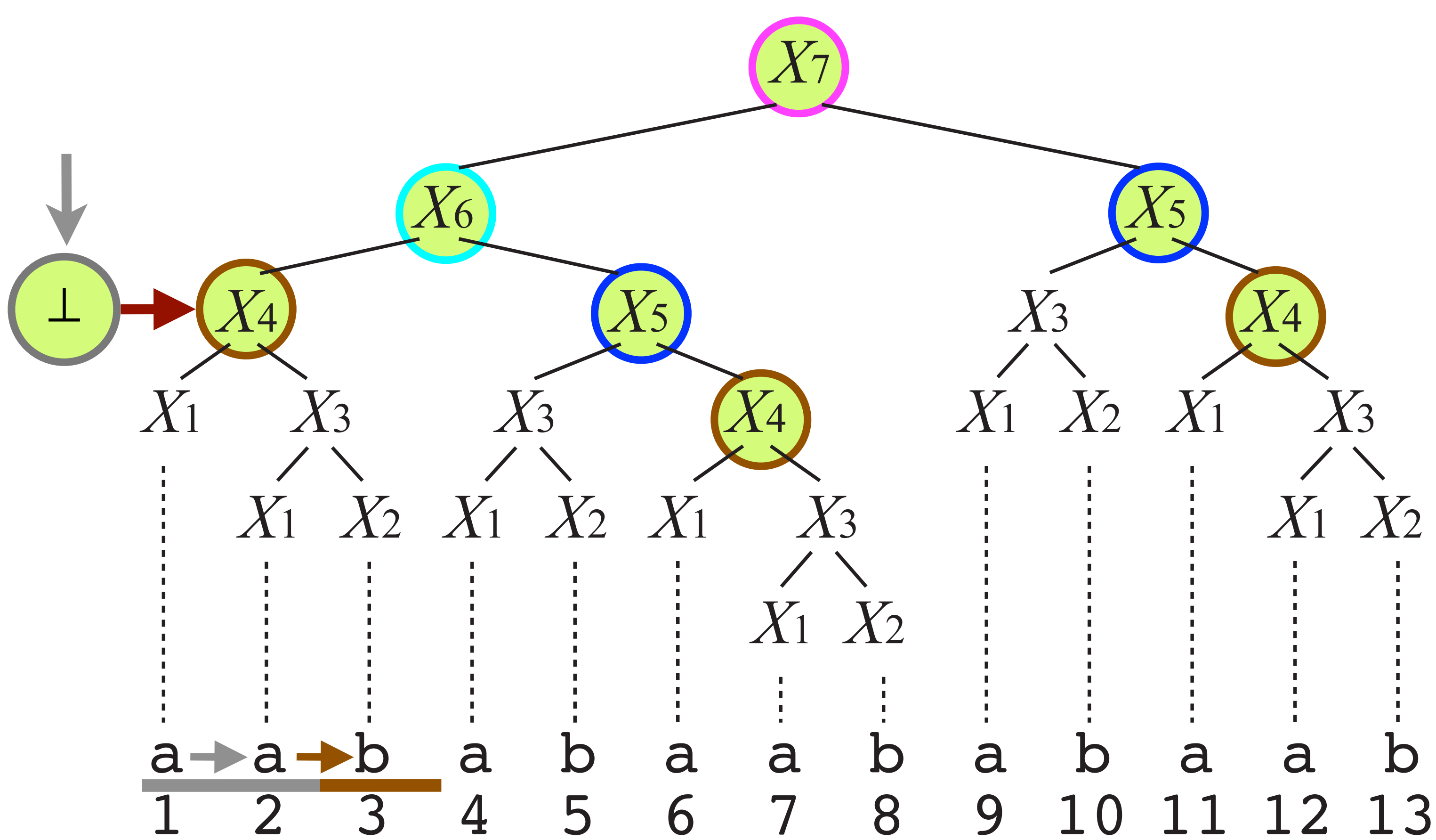
Prefix of X_7 = aab

We compute the label by decompressing prefix of variables

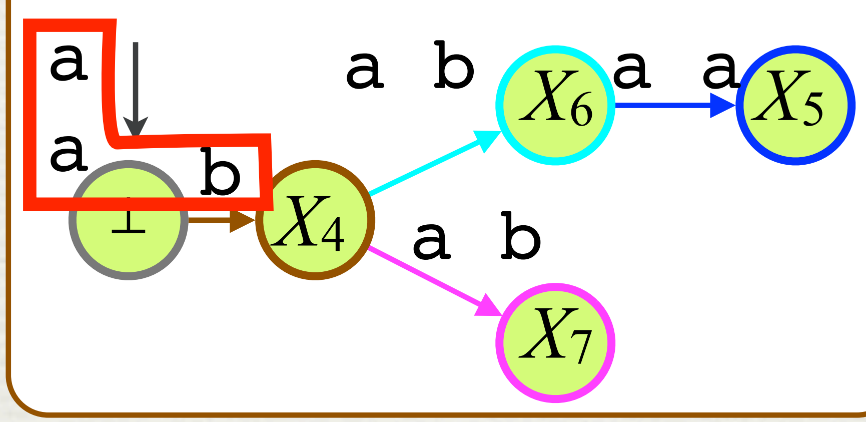
Calculating neighbor tree label

Label can be computed by decompressing prefix of variables

3-gram neighbor in derivation tree



neighbor tree of G



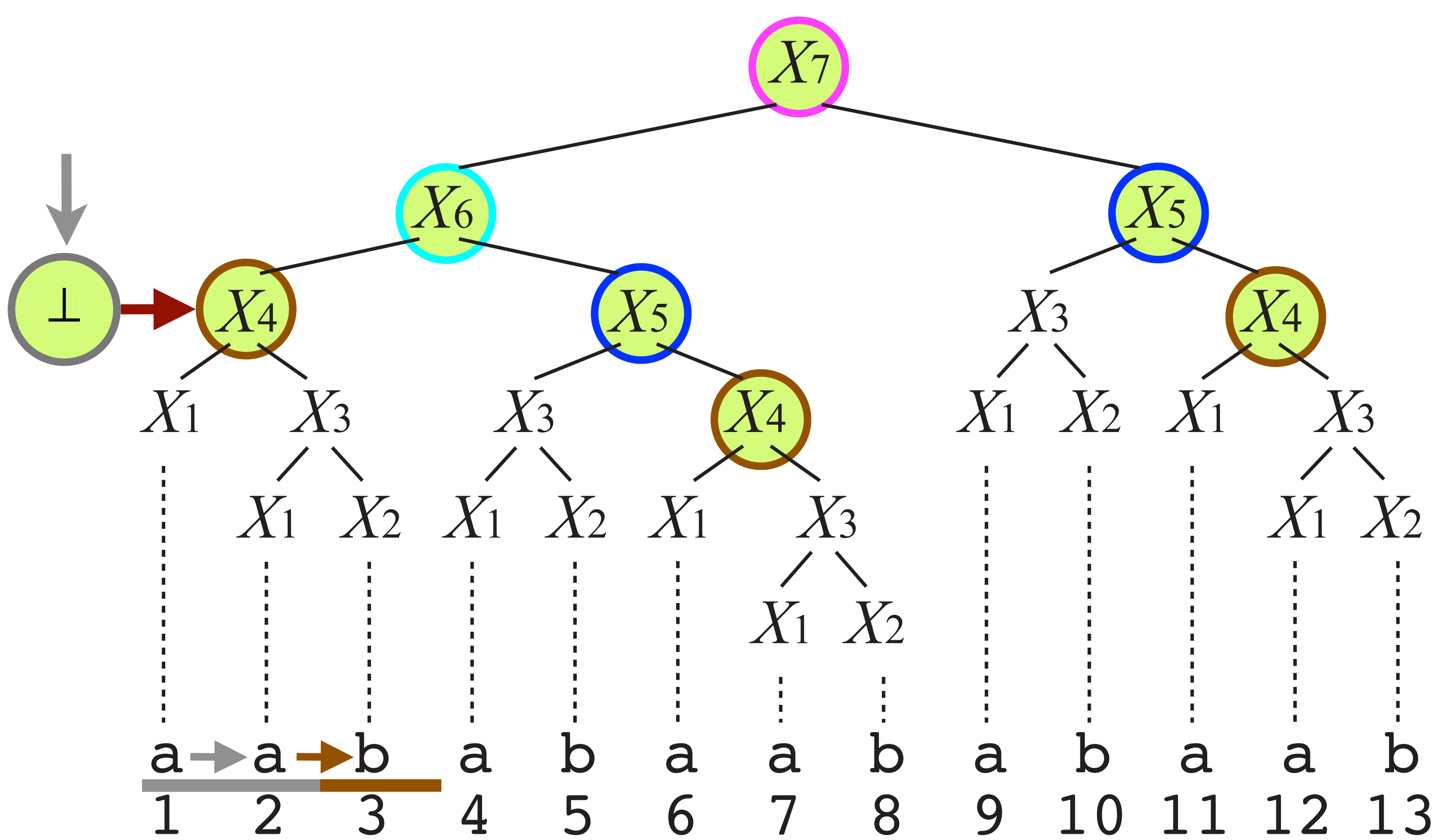
Prefix of X_7 = aab

We compute the label by decompressing prefix of variables

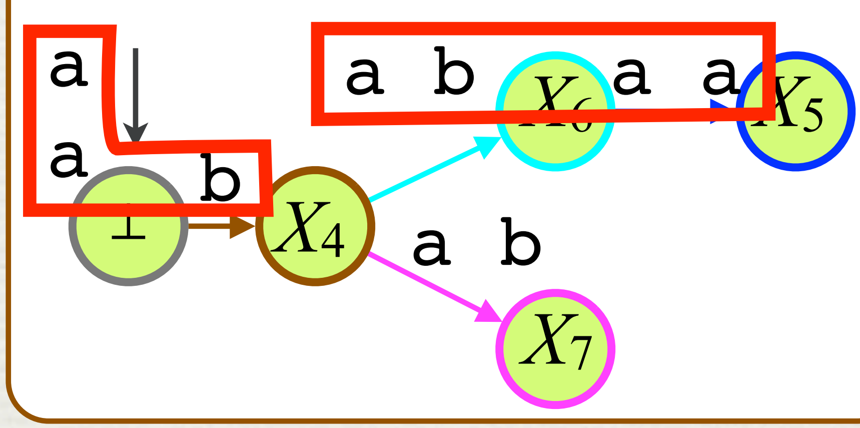
Calculating neighbor tree label

Label can be computed by decompressing prefix of variables

3-gram neighbor in derivation tree



neighbor tree of G



Prefix of X_7 = aab

Prefix of X_5 = abaa

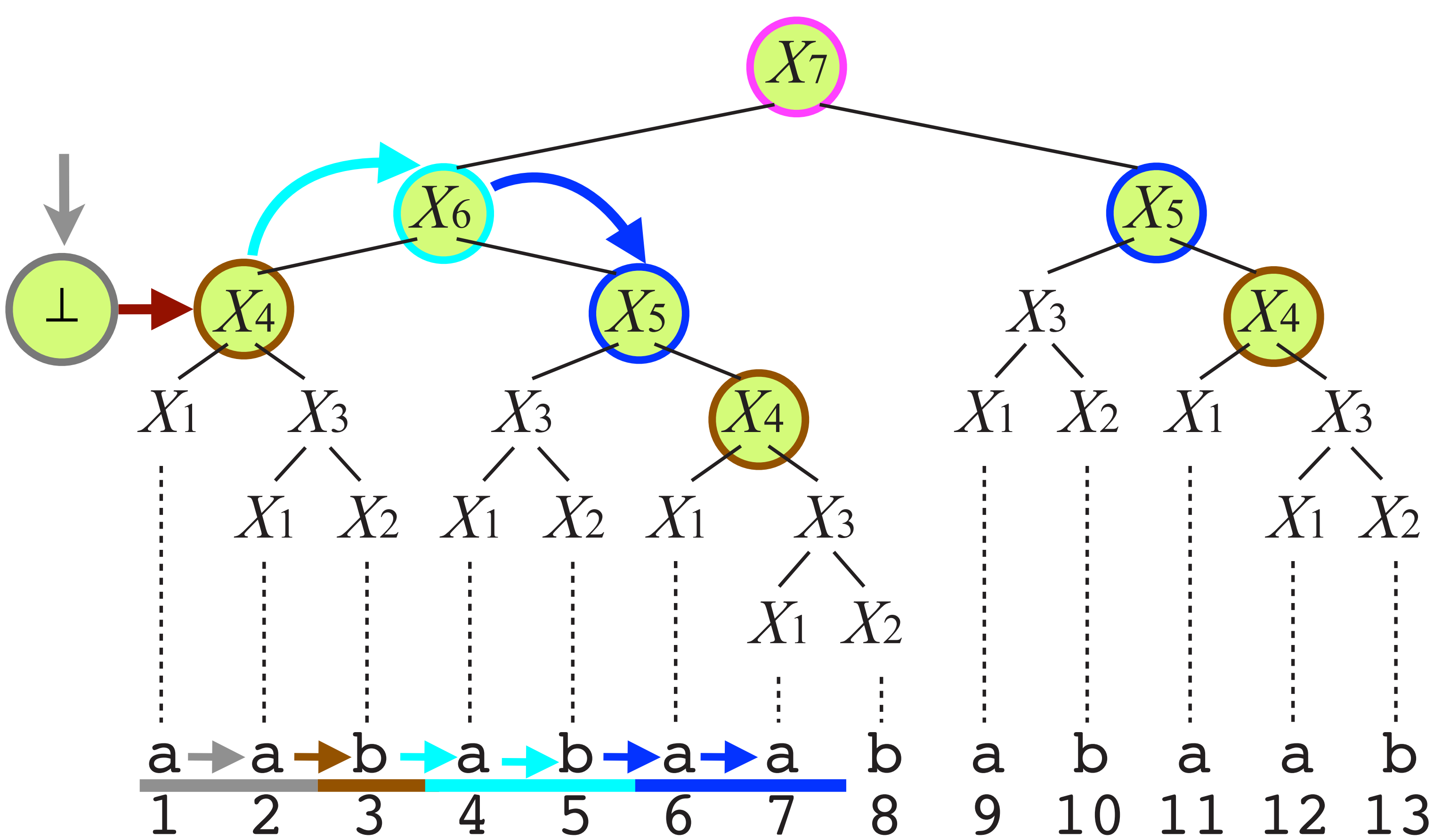
We compute the label by decompressing prefix of variables

13年2月21日木曜日

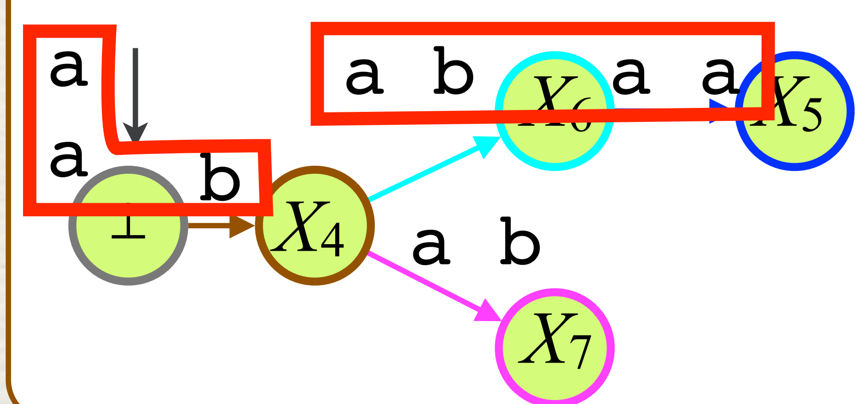
Calculating neighbor tree label

Label can be computed by decompressing prefix of variables

3-gram neighbor in derivation tree



neighbor tree of G



Prefix of X_7 = aab

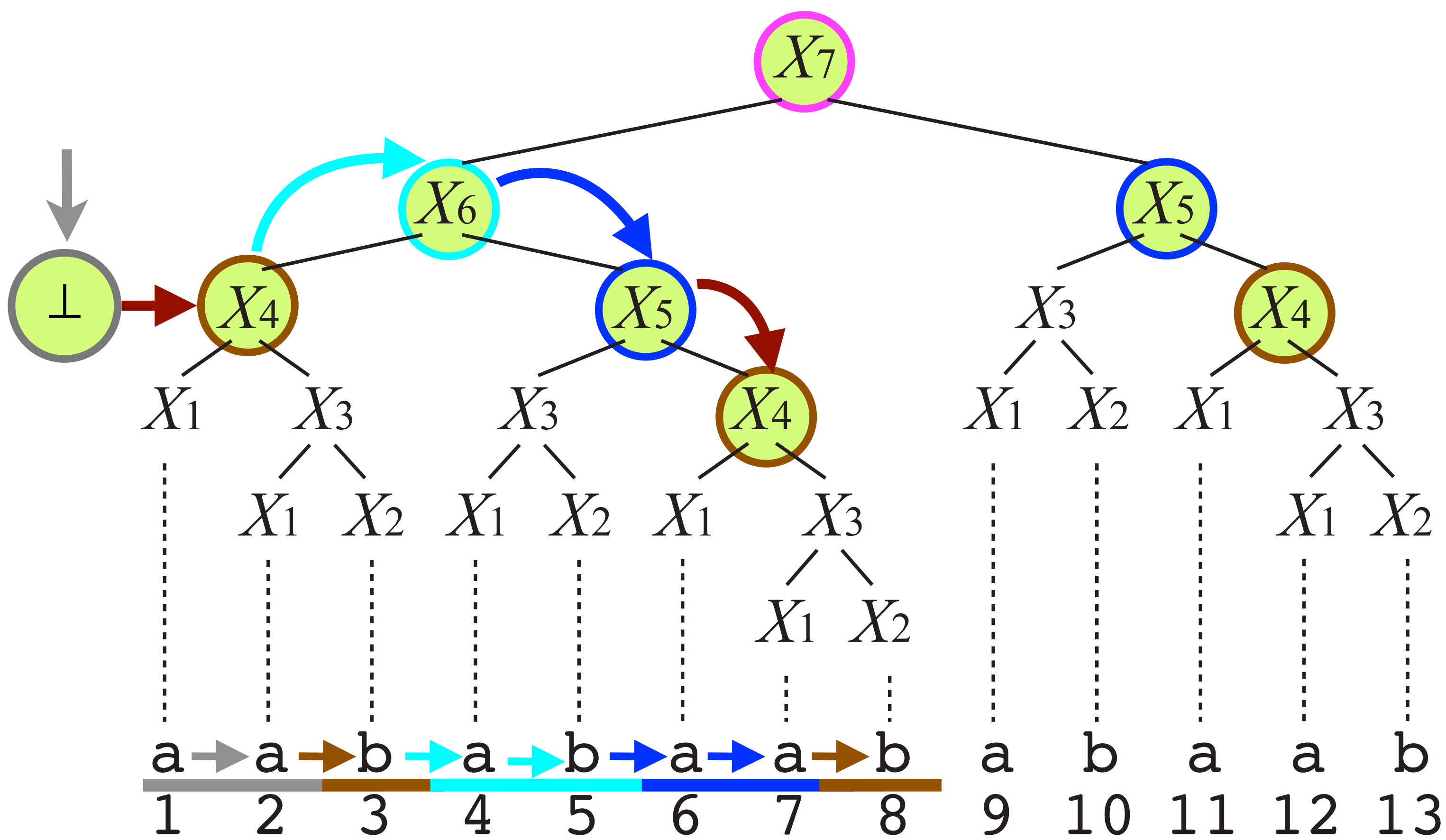
Prefix of X_5 = abaa

We compute the label by decompressing prefix of variables

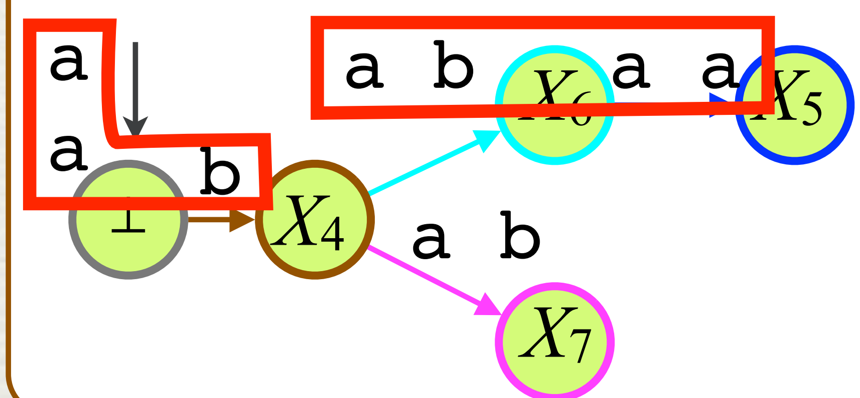
Calculating neighbor tree label

Label can be computed by decompressing prefix of variables

3-gram neighbor in derivation tree



neighbor tree of G



Prefix of X_7 = aab

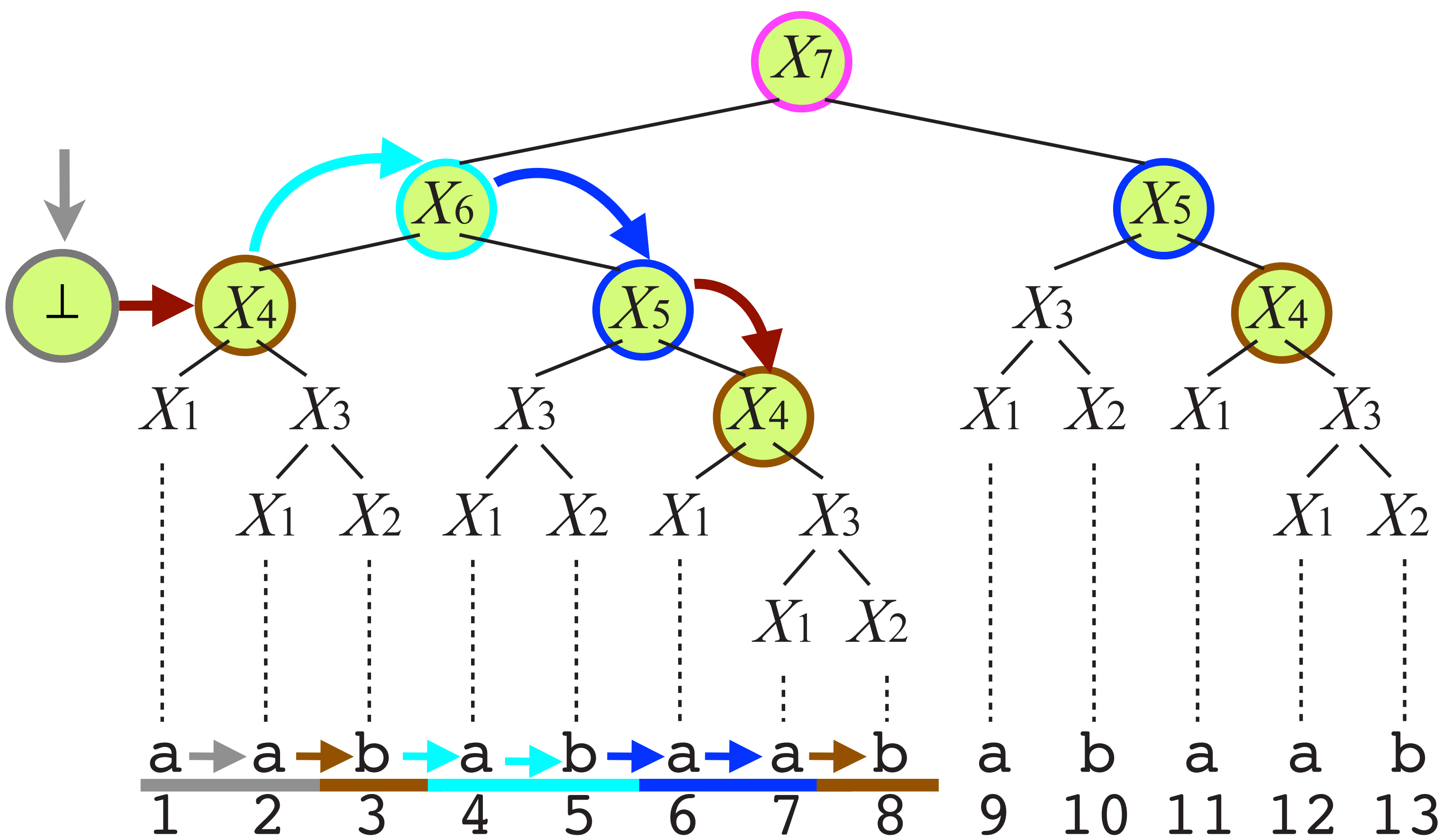
Prefix of X_5 = abaa

We compute the label by decompressing prefix of variables

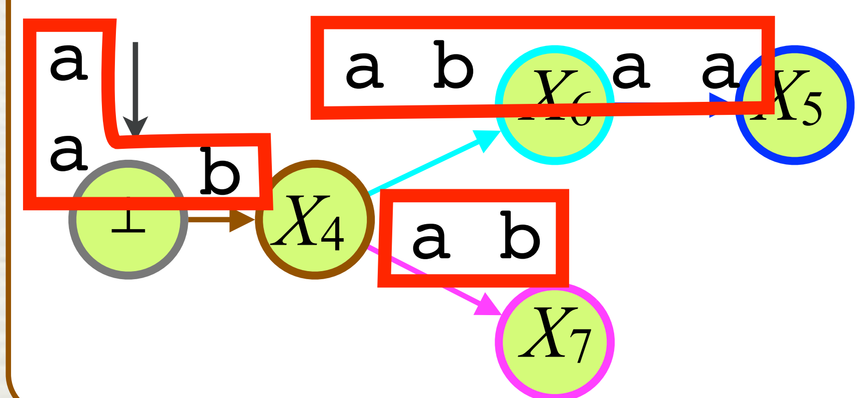
Calculating neighbor tree label

Label can be computed by decompressing prefix of variables

3-gram neighbor in derivation tree



neighbor tree of G



Prefix of X_7 = aab

Prefix of X_5 = abaa

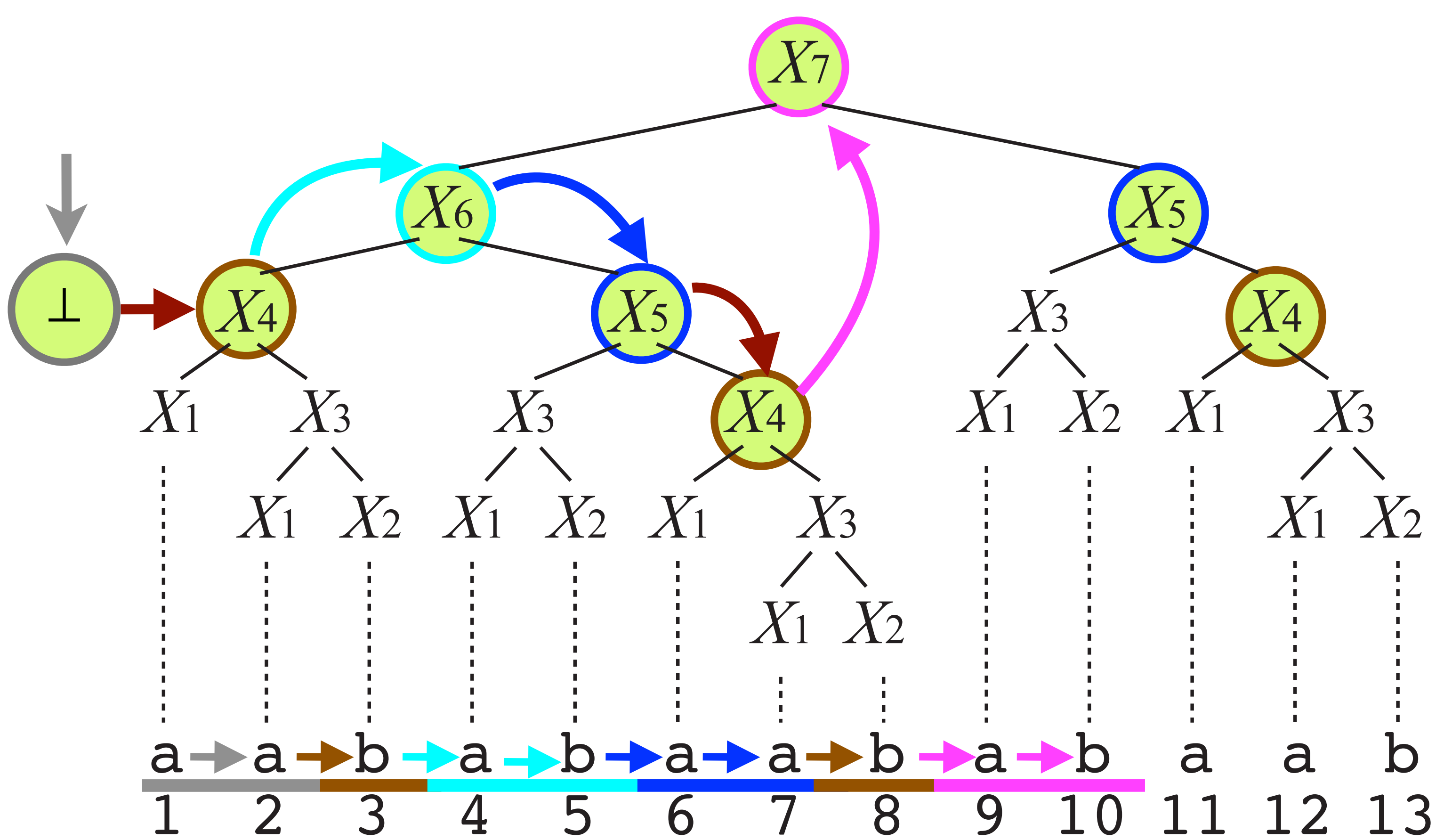
Prefix of X_5 = ab

We compute the label by decompressing prefix of variables

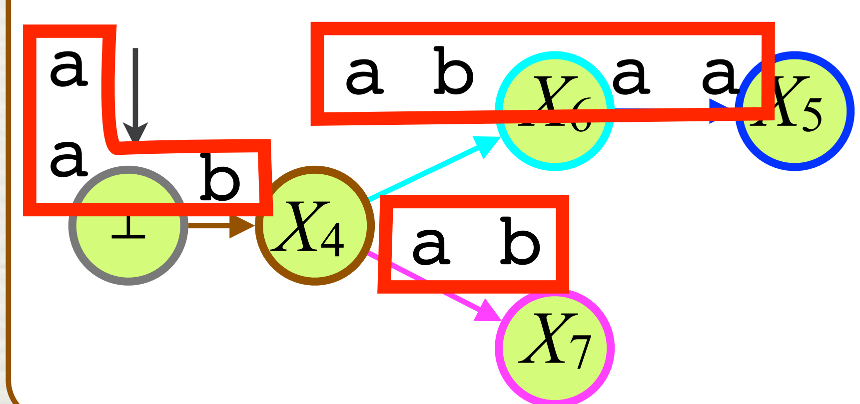
Calculating neighbor tree label

Label can be computed by decompressing prefix of variables

3-gram neighbor in derivation tree



neighbor tree of G



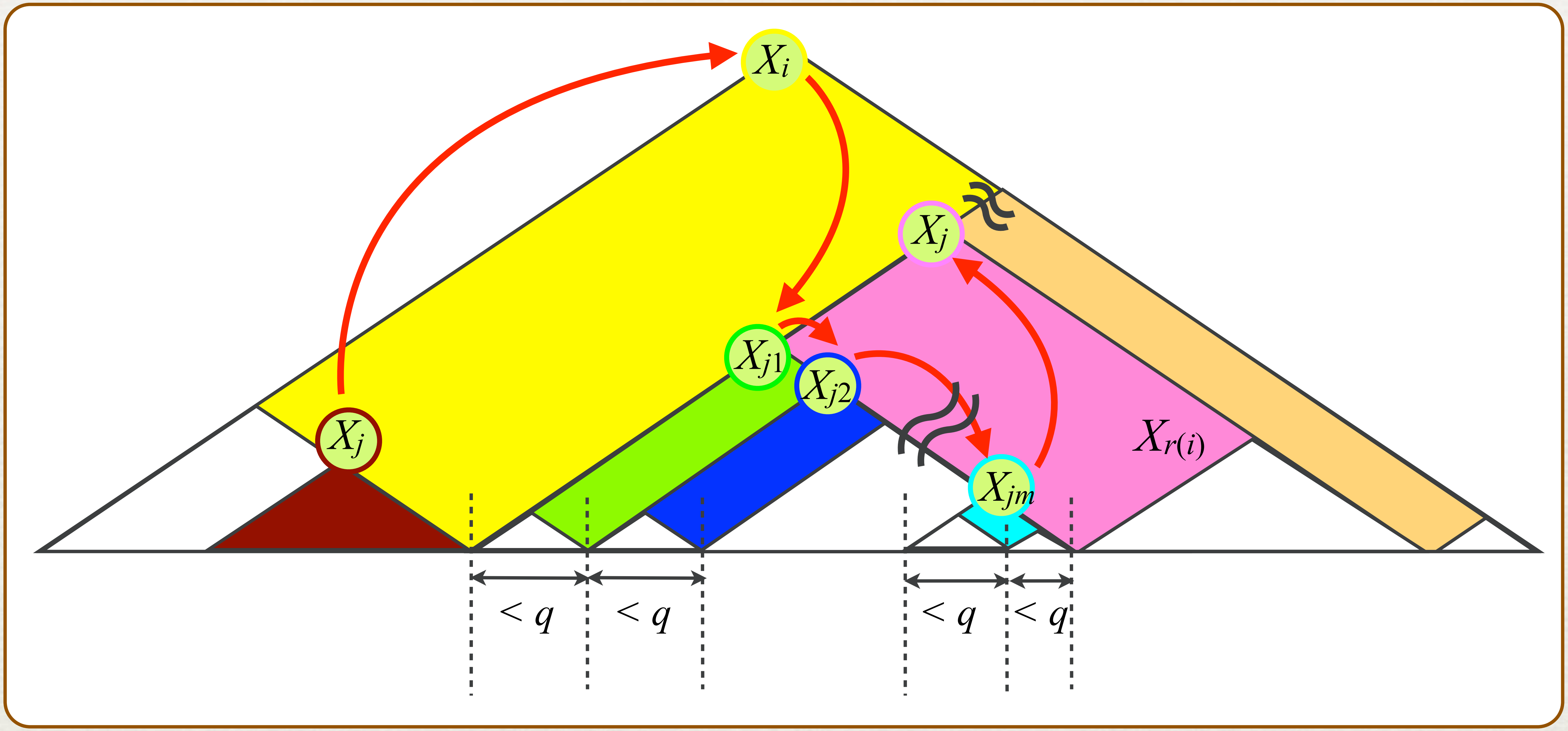
Prefix of X_7 = aab

Prefix of X_5 = abaa

Prefix of X_5 = ab

We compute the label by decompressing prefix of variables

Calculating neighbor tree label



can be decompress by its label size