Compact q-Gram Profiling of Compressed Strings

Philip Bille, Patrick Hagge Cording, and Inge Li Gørtz

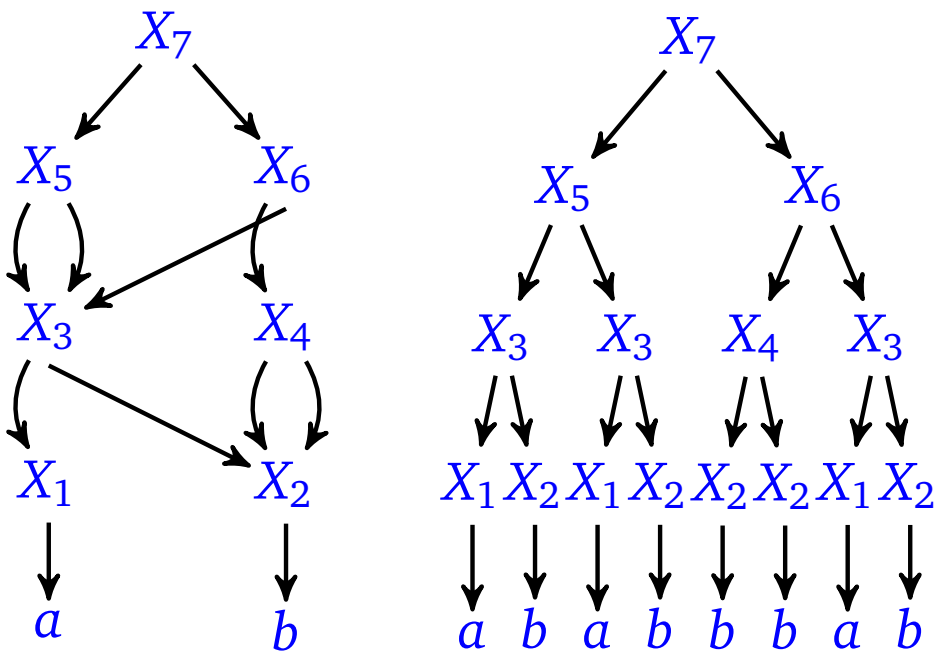
DTU Compute, Technical University of Denmark, phaco@dtu.dk

24th CPM, Bad Herrenalb
June 19, 2013

Compressed strings

- We consider *Straight Line Programs* (SLPs)
- An SLP is a grammar in Chomsky normal form that derives one string only
- It consists of production rules $X_1, \ldots, X_n$ of the form $X_i = X_l X_r$ (nonterminal) or $X_i = \mathtt{a}$ (terminal)

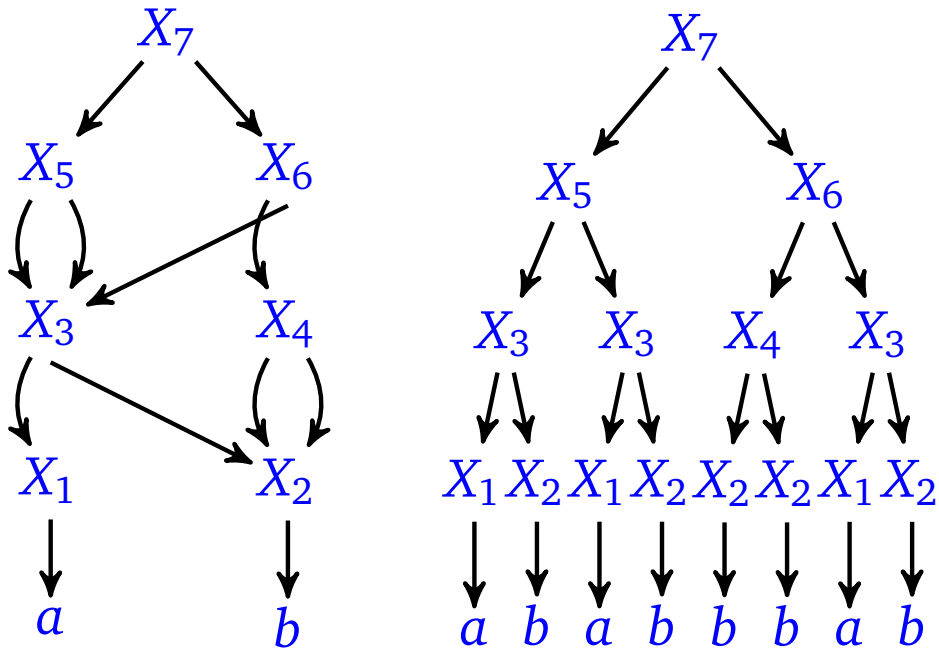
Example:



Straight Line Programs

- X_i derives the string t_{X_i} and $|X_i|$ is the length of t_{X_i}
- $occ(X_i)$ is the number of times X_i occurs in the parse tree of the SLP

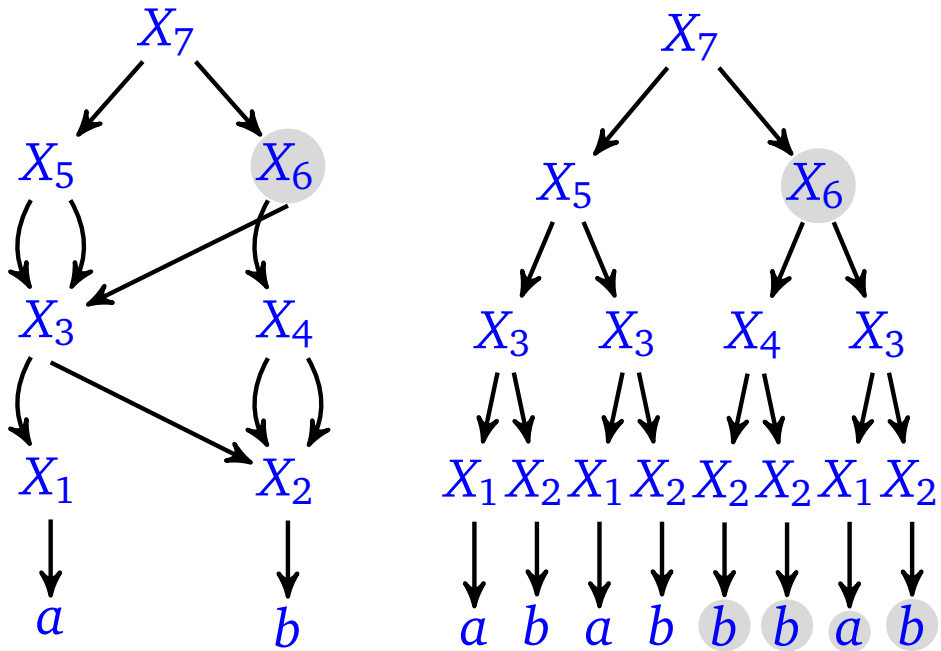
Example:



Straight Line Programs

- X_i derives the string t_{X_i} and $|X_i|$ is the length of t_{X_i}
- $occ(X_i)$ is the number of times X_i occurs in the parse tree of the SLP

Example:

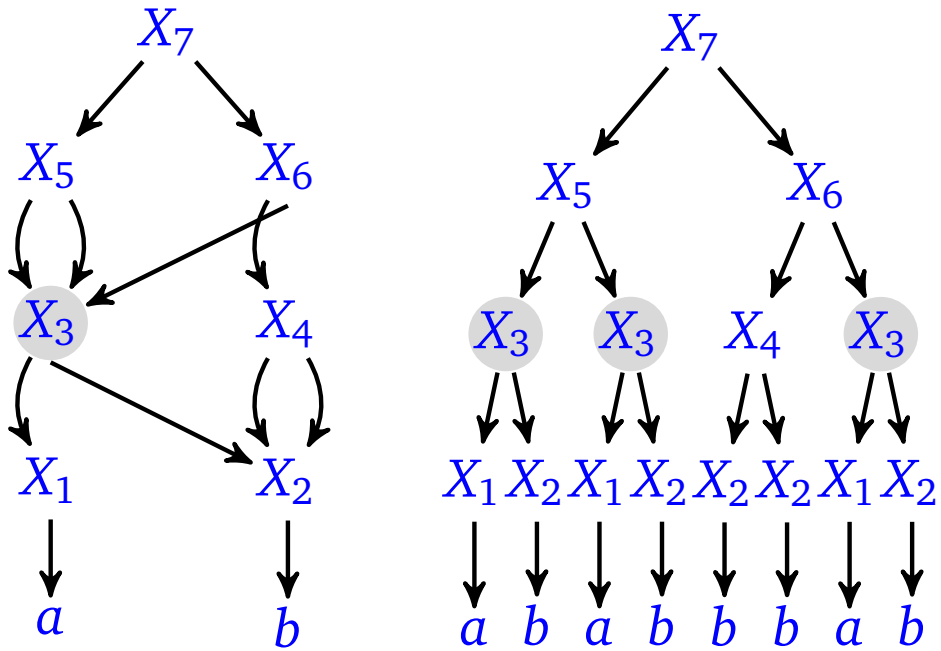


$|X_6| = 4$

Straight Line Programs

- X_i derives the string t_{X_i} and $|X_i|$ is the length of t_{X_i}
- $occ(X_i)$ is the number of times X_i occurs in the parse tree of the SLP

Example:



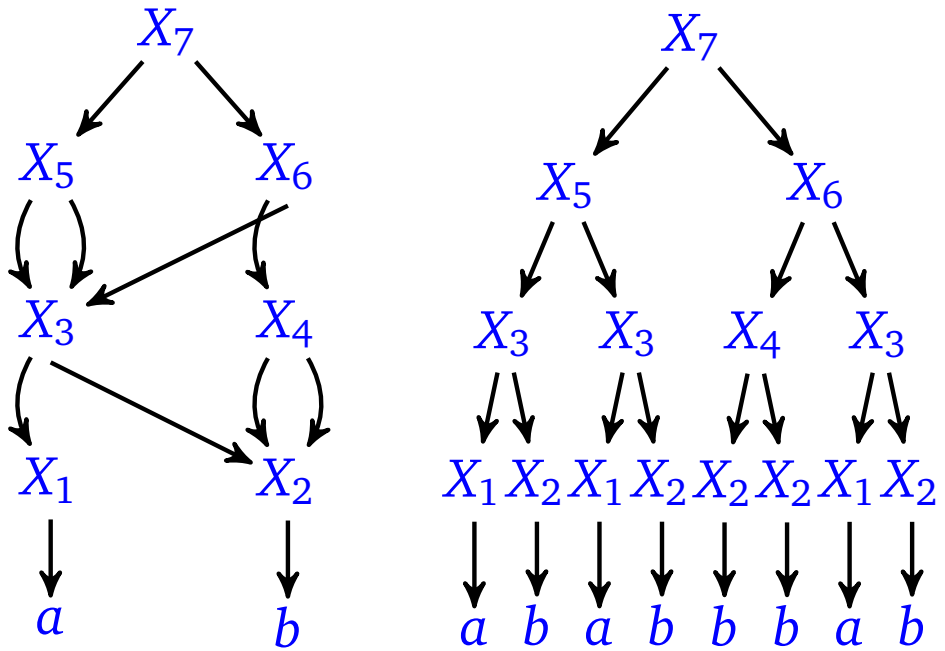
$|X_6| = 4$

$occ(X_3) = 3$

Straight Line Programs

- X_i derives the string t_{X_i} and $|X_i|$ is the length of t_{X_i}
- $occ(X_i)$ is the number of times X_i occurs in the parse tree of the SLP

Example:



$|X_6| = 4$

$occ(X_3) = 3$

Using $O(n)$ preprocessing time and $O(n)$ space, and given a node X_i, we may

- get $|X_i|$ and $occ(X_i)$ in constant time, and
- decompress the j length prefix of t_{X_i} in $O(j)$ time [Gąsieniec et al., DCC '05]

Problem definition

Input: An SLP $\mathcal{S}$ of size n compressing a string T of size N, an integer $q \geq 2$.

Output: The *q-gram* profile of T, i.e., a data structure that supports *substring frequency queries* for strings of length q in $O(q)$ time.

Example:

$q = 3$
$T = \texttt{aababaababaab}$

$\texttt{aab} \rightarrow 3$
$\texttt{aba} \rightarrow 4$
$\texttt{bab} \rightarrow 2$
$\texttt{baa} \rightarrow 2$

Problem definition

Input: An SLP $\mathcal{S}$ of size n compressing a string T of size N, an integer $q \geq 2$.

Output: The *q-gram* profile of T, i.e., a data structure that supports *substring frequency queries* for strings of length q in $O(q)$ time.

Example:

$q = 3$
$T = \underline{\texttt{aab}}\texttt{ab}\underline{\texttt{aab}}\texttt{ab}\underline{\texttt{aab}}$

$\underline{\texttt{aab}} \rightarrow 3$
$\texttt{aba} \rightarrow 4$
$\texttt{bab} \rightarrow 2$
$\texttt{baa} \rightarrow 2$

Problem definition

Input: An SLP $\mathcal{S}$ of size n compressing a string T of size N, an integer $q \geq 2$.

Output: The *q-gram* profile of T, i.e., a data structure that supports *substring frequency queries* for strings of length q in $O(q)$ time.

Example:

$q = 3$
$T =$ a<u>ababaababa</u>ab

aab $\rightarrow 3$
<u>aba</u> $\rightarrow 4$
bab $\rightarrow 2$
baa $\rightarrow 2$

Problem definition

Input: An SLP $\mathcal{S}$ of size n compressing a string T of size N, an integer $q \geq 2$.

Output: The *q-gram* profile of T, i.e., a data structure that supports *substring frequency queries* for strings of length q in $O(q)$ time.

Example:

$q = 3$
$T =$ aababaababaab

aab $\rightarrow 3$
aba $\rightarrow 4$
bab $\rightarrow 2$
baa $\rightarrow 2$

Problem definition

Input: An SLP $\mathcal{S}$ of size n compressing a string T of size N, an integer $q \geq 2$.

Output: The *q-gram* profile of T, i.e., a data structure that supports *substring frequency queries* for strings of length q in $O(q)$ time.

Example:

$q = 3$
$T = \texttt{aababaababaab}$

$\texttt{aab} \rightarrow 3$
$\texttt{aba} \rightarrow 4$
$\texttt{bab} \rightarrow 2$
$\texttt{baa} \rightarrow 2$

Results

	Time	Space
Decompression + suffix tree	$O(N)$	$O(N)$
Goto et al. [SPIRE '11]	$O(qn)$	$O(qn)$
Goto et al. [CPM '12]	$O(N - \alpha)$	$O(N - \alpha)$
Our algorithm	$O(N - \alpha)$	$O(n + q + k_{T,q})$

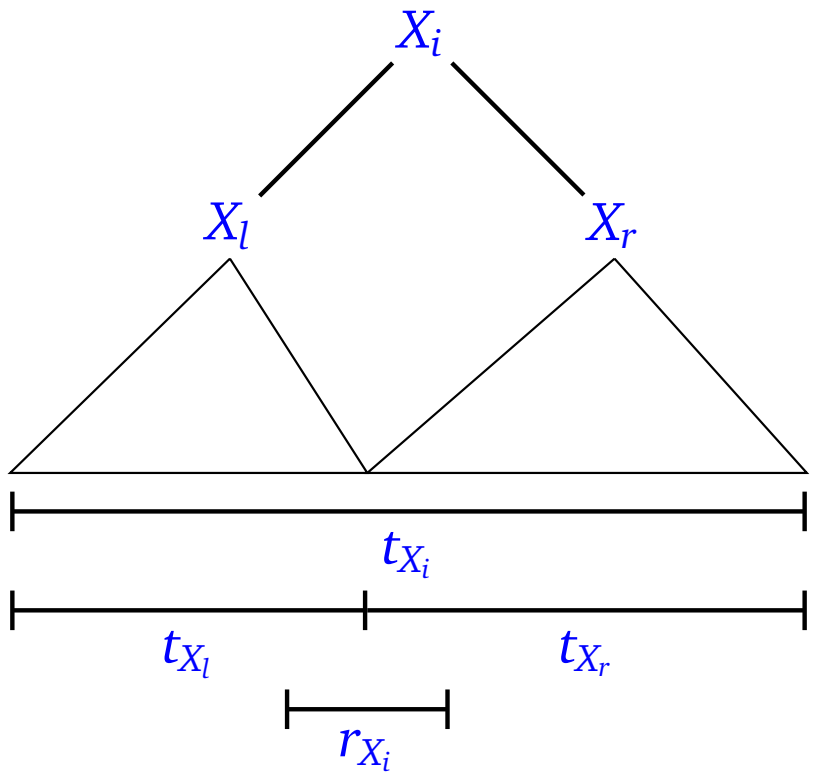
- $N - \alpha < qn$ is the number of characters needed to decompress to ensure that all distinct q-grams are in the profile
- $1 \leq k_{T,q} \leq N - \alpha$ is the number of distinct q-grams in T

$\Rightarrow$ Our algorithm is space efficient regardless of how well the string is compressed.

Relevant substrings

The main observation

The *relevant substring* r_{X_i} of a node $X_i = X_l X_r$ is the substring of t_{X_i} that contains q-grams starting in t_{X_l} and ending t_{X_r}.



"Relevant substring lemma":

$$socc(s, T) = \sum_{X_i \in \mathcal{S}} socc(s, r_{X_i}) \cdot occ(X_i).$$

Outline of our algorithm

Step 1: Build the *q-gram graph* from the SLP.

Step 2: Turn the the q-gram graph into a tree.

Step 3: Construct a suffix tree of strings in the tree.

Step 4: ~~Verification. Going from Monte Carlo to Las Vegas.~~

In the paper.

Step 1: Building the q-gram graph from the SLP

Karp-Rabin fingerprints

A fingerprint function ϕ maps strings to integers.

$$\phi(S) = \sum_{k=1}^{|S|} S[k]b^k \bmod p\,.$$

Example:

$T = \texttt{ababbbab} = \texttt{01011101}$

$\phi(T[1..3]) = \phi(\texttt{bab}) = 1 \cdot b^1 + 0 \cdot b^2 + 1 \cdot b^3 \mod p$

Properties:

- $O(1)$ space per fingerprint
- For the right choice of b and p, collisions are very unlikely
- $\phi(T[i+1\,..\,j+1])$ can be computed from $\phi(T[i\,..\,j])$ in $O(1)$ time

Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T = \texttt{ababbbab}$

Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T =$ abababbbab

$\phi(ab)$

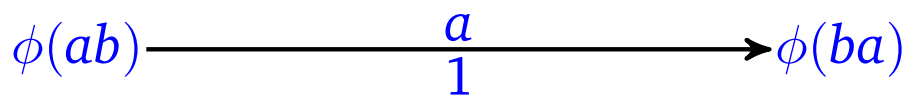
Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T = \texttt{ababbbab}$



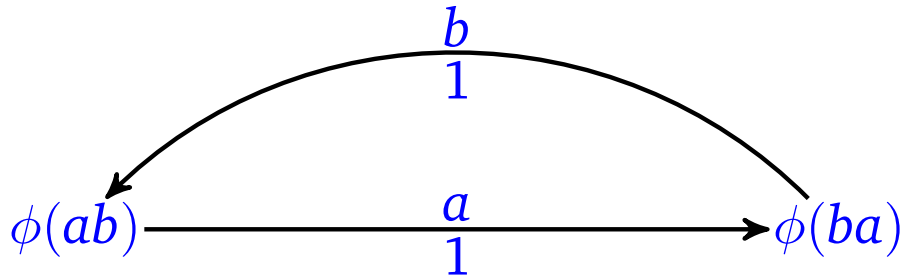
Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T =$ `ababbbab`



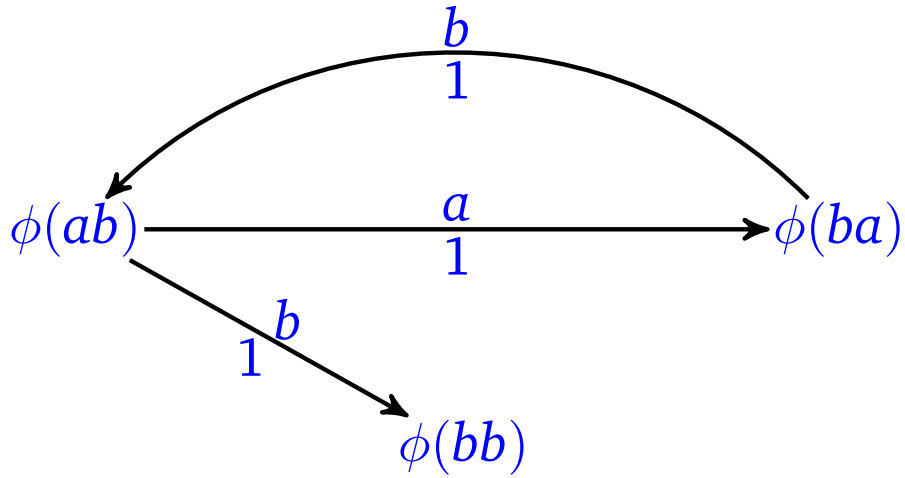
Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T =$ aba<u>bb</u>bab



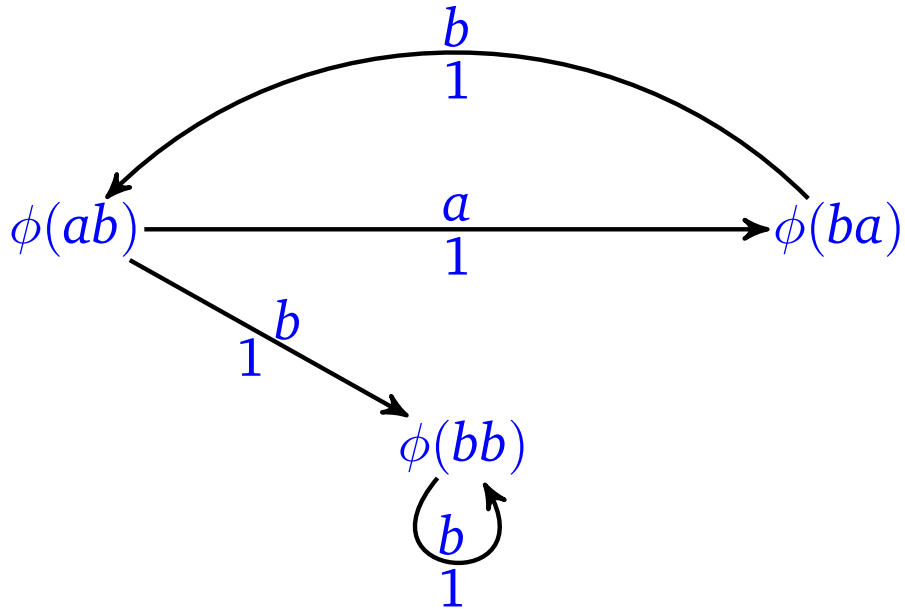
Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T =$ ababbbab



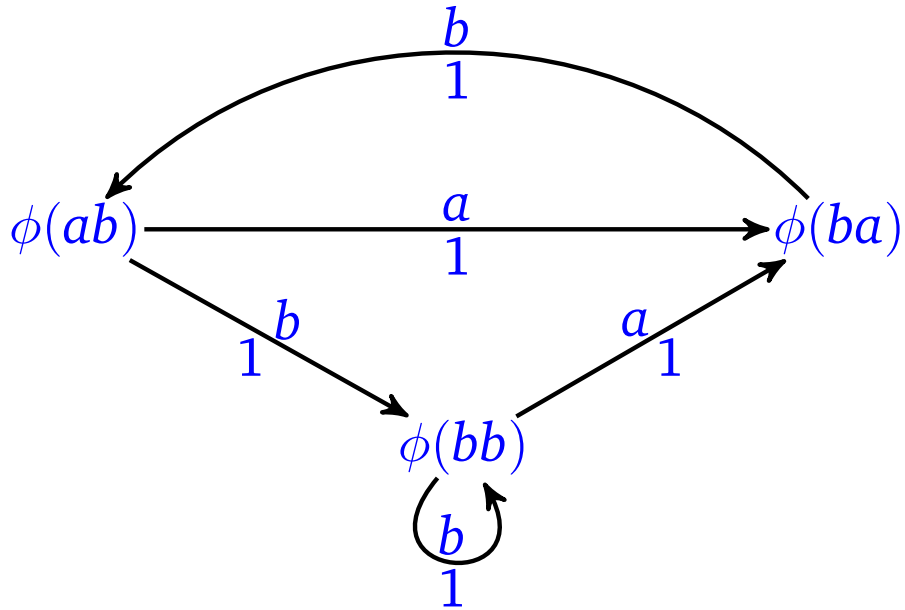
Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T =$ `ababbbab`



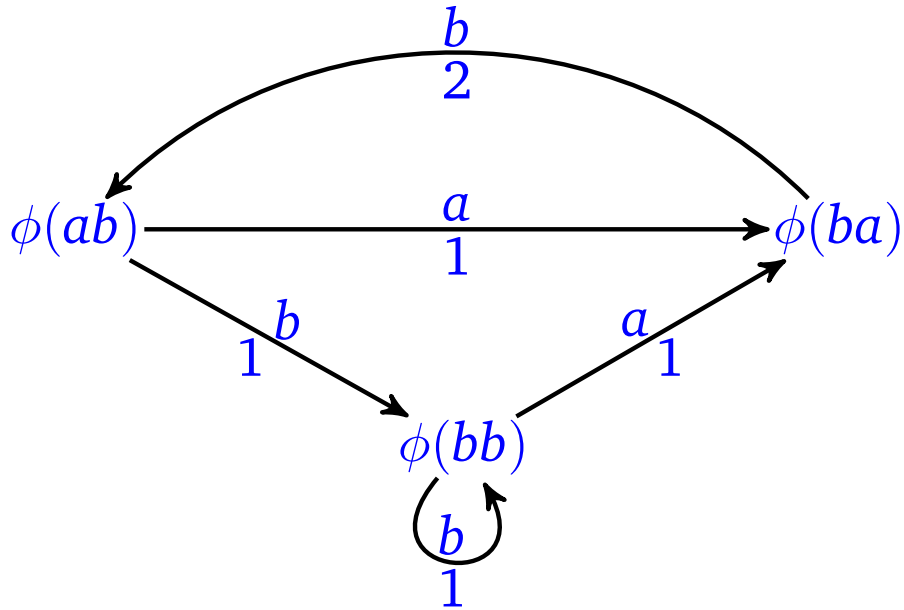
Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T =$ `ababbbab`



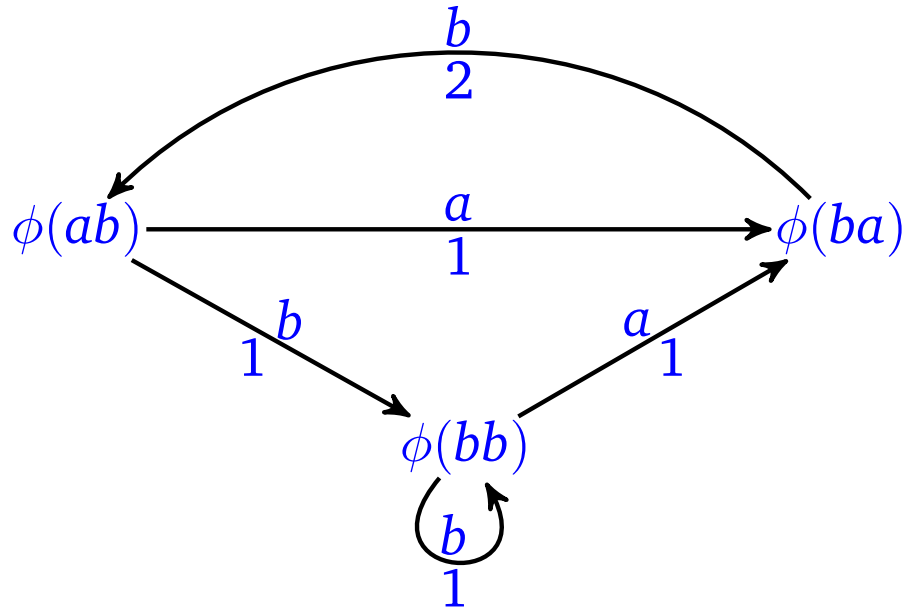
Step 1: Building the q-gram graph from the SLP

The q-gram graph

Example:

$q = 3$

$T = \texttt{ababbbab}$



Key property: The size of the q-gram graph is $O(k_{T,q})$.

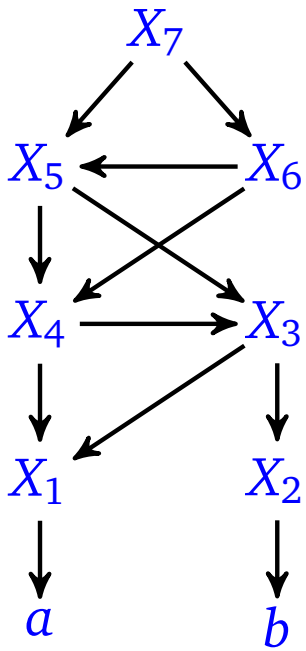
Step 1: Building the q-gram graph from the SLP

Initialization:

- Compute $occ(X_i)$ values and data structure for linear time prefix decompression
- Compute the number of q-grams in each relevant substring: $Q(X_i) = |r_{X_i}| - (q-1)$

Main part:

- Assume we are at node $X_i = X_l X_r$ and have the q-gram graph for t_{X_l}
- Decompress the $Q(X_i)$ length prefix of t_{X_r}



$q = 3$

$T = \texttt{aababaababaab}$

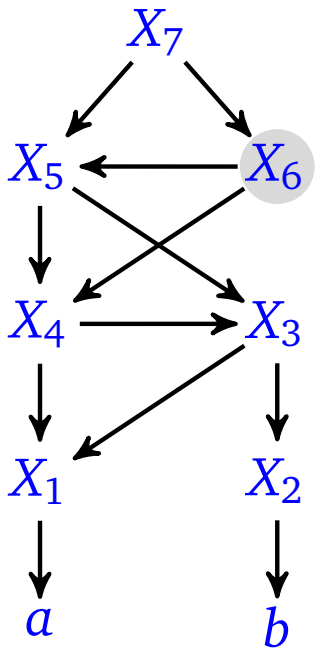
Step 1: Building the q-gram graph from the SLP

Initialization:

- Compute $occ(X_i)$ values and data structure for linear time prefix decompression
- Compute the number of q-grams in each relevant substring: $Q(X_i) = |r_{X_i}| - (q-1)$

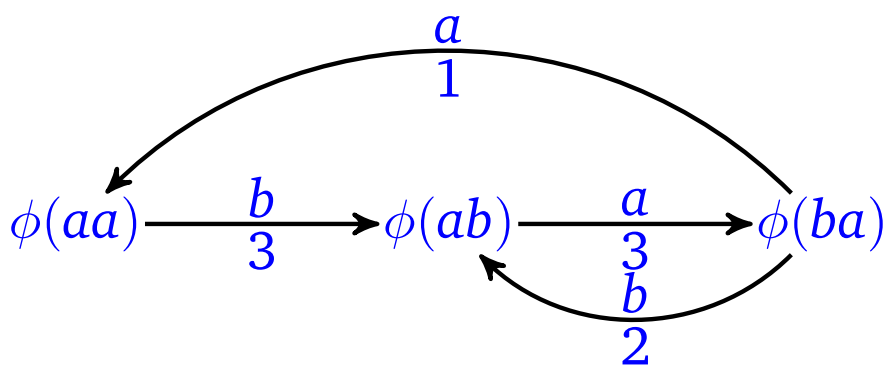
Main part:

- Assume we are at node $X_i = X_l X_r$ and have the q-gram graph for t_{X_l}
- Decompress the $Q(X_i)$ length prefix of t_{X_r}



$q = 3$

$T =$ aababaababaab



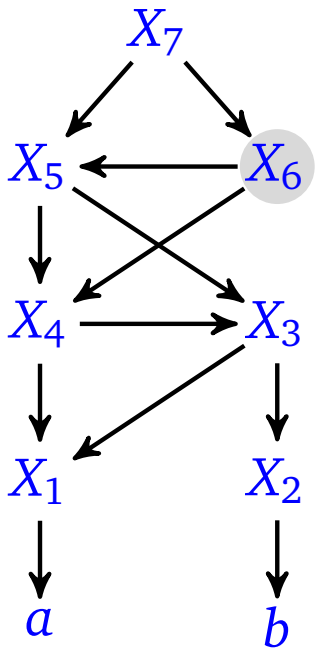
Step 1: Building the q-gram graph from the SLP

Initialization:

- Compute $occ(X_i)$ values and data structure for linear time prefix decompression
- Compute the number of q-grams in each relevant substring: $Q(X_i) = |r_{X_i}| - (q-1)$

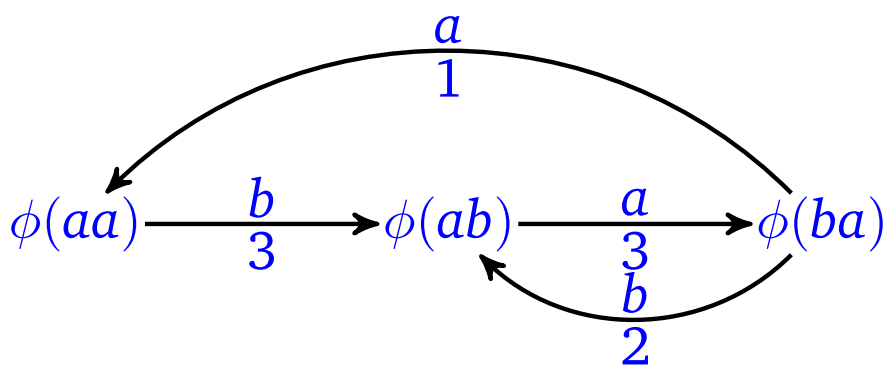
Main part:

- Assume we are at node $X_i = X_l X_r$ and have the q-gram graph for t_{X_l}
- Decompress the $Q(X_i)$ length prefix of t_{X_r}



$q = 3$

$T =$ aababaababaaab



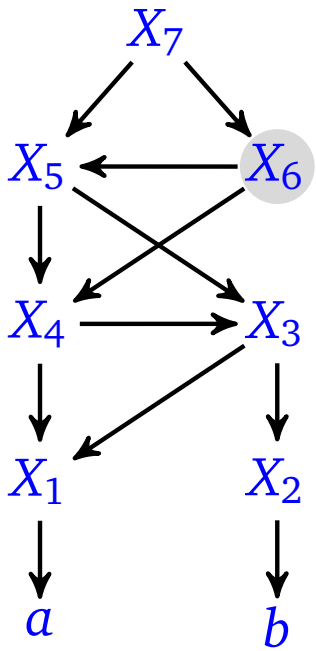
Step 1: Building the q-gram graph from the SLP

Initialization:

- Compute $occ(X_i)$ values and data structure for linear time prefix decompression
- Compute the number of q-grams in each relevant substring: $Q(X_i) = |r_{X_i}| - (q-1)$

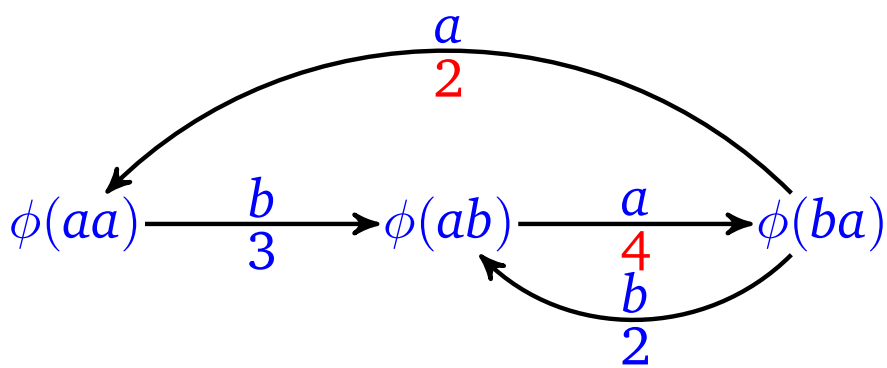
Main part:

- Assume we are at node $X_i = X_l X_r$ and have the q-gram graph for t_{X_l}
- Decompress the $Q(X_i)$ length prefix of t_{X_r}



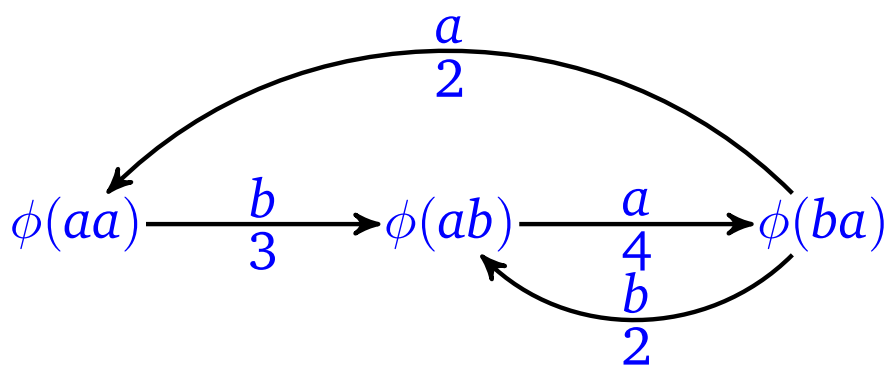
$q = 3$

$T =$ aababaababaaab



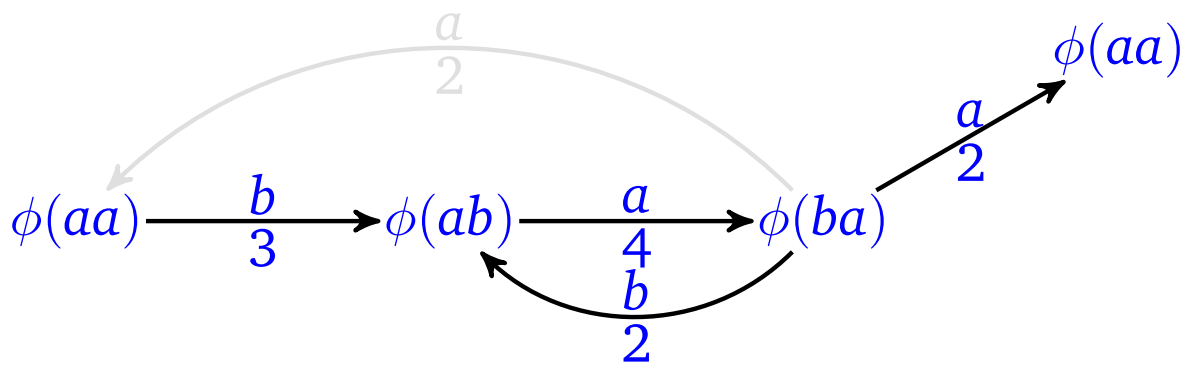
Step 2: Turning the q-gram graph into a tree

- Do a depth-first traversal of the q-gram graph; start from the first node created
- Create a new leaf when reaching a previously visited node
- Keep labels on nodes and edges
- Extend the tree with the first $q-1$ characters of T



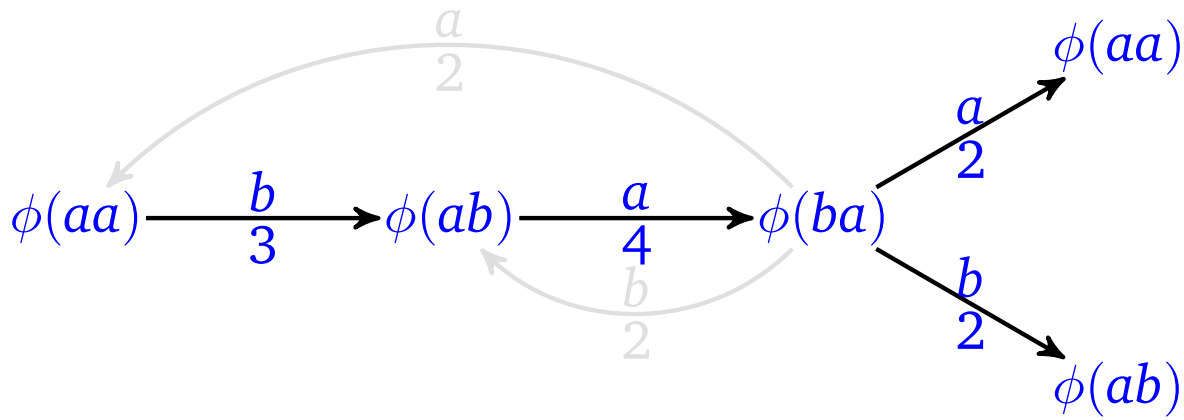
Step 2: Turning the q-gram graph into a tree

- Do a depth-first traversal of the q-gram graph; start from the first node created
- Create a new leaf when reaching a previously visited node
- Keep labels on nodes and edges
- Extend the tree with the first $q - 1$ characters of T



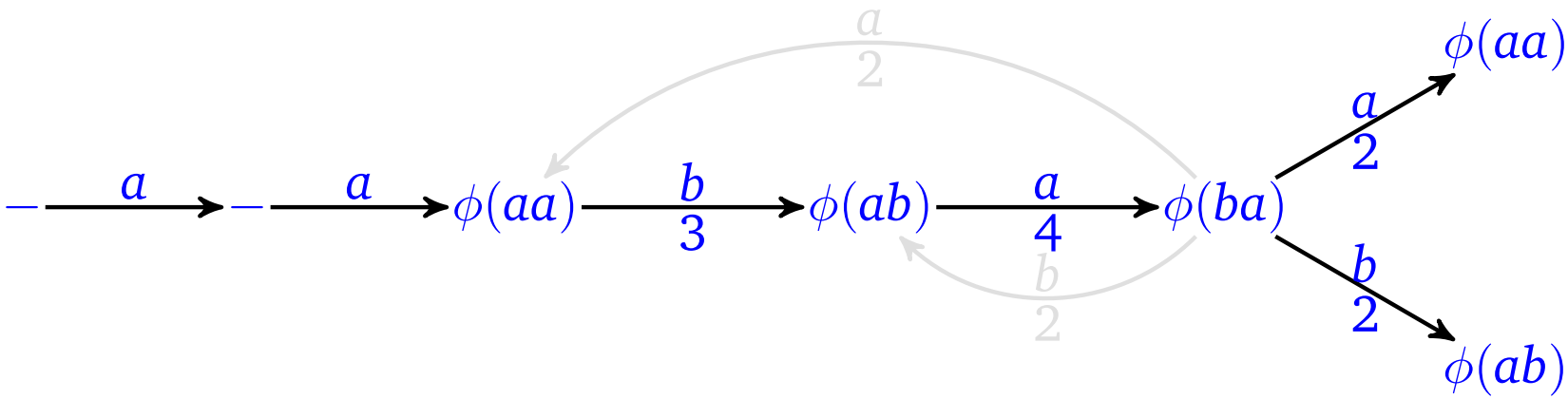
Step 2: Turning the q-gram graph into a tree

- Do a depth-first traversal of the q-gram graph; start from the first node created
- Create a new leaf when reaching a previously visited node
- Keep labels on nodes and edges
- Extend the tree with the first $q - 1$ characters of T



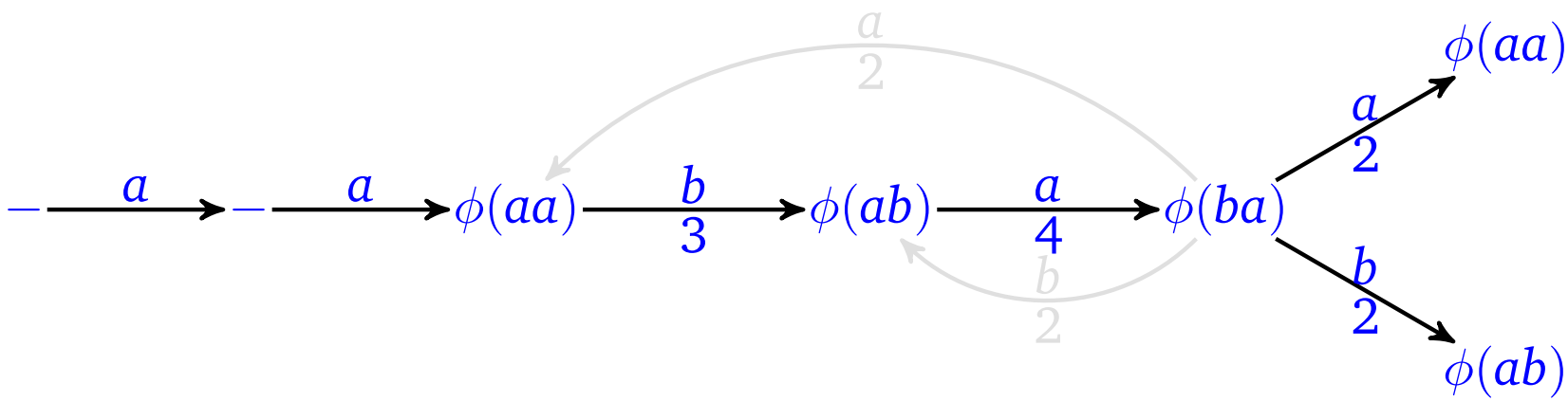
Step 2: Turning the q-gram graph into a tree

- Do a depth-first traversal of the q-gram graph; start from the first node created
- Create a new leaf when reaching a previously visited node
- Keep labels on nodes and edges
- Extend the tree with the first $q-1$ characters of T



Step 2: Turning the q-gram graph into a tree

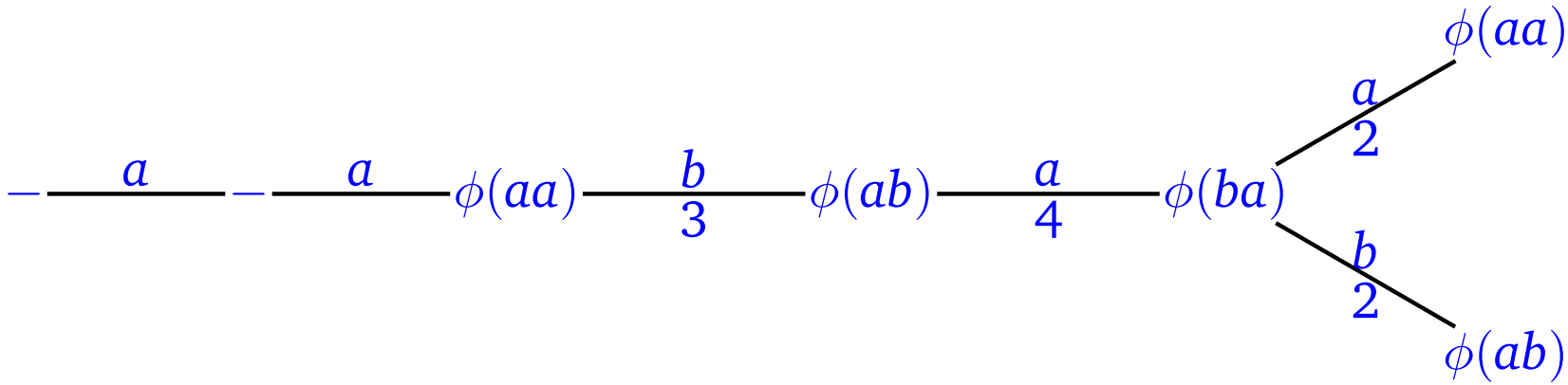
- Do a depth-first traversal of the q-gram graph; start from the first node created
- Create a new leaf when reaching a previously visited node
- Keep labels on nodes and edges
- Extend the tree with the first $q-1$ characters of T



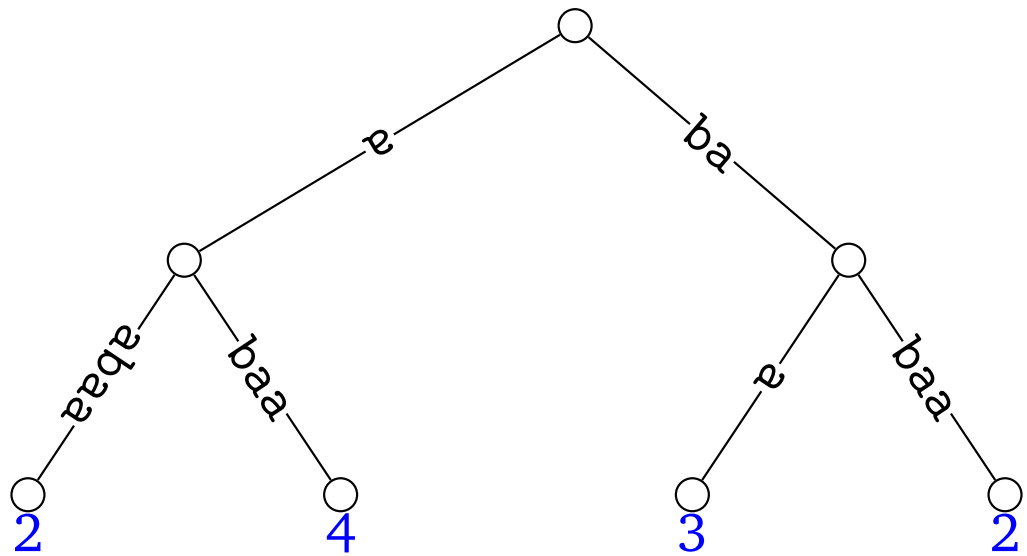
Key property: Size of the tree is $O(q + k_{T,q})$.

Step 3: Constructing a suffix tree of q-grams in the tree

Tree of q-gram graph. Tree of the strings {aabaa, babaa}.



Generalized suffix tree of the strings {aabaa, babaa}.



Analysis of the algorithm

Time

Step 1: Build the q-gram graph from the SLP.

- Let $\mathcal{S}_q$ be the set of nodes with a relevant substring
- $(q-1) + \sum_{X_i \in \mathcal{S}_q} Q(X_i) = (q-1) + \sum_{X_i \in \mathcal{S}_q} |r_{X_i}| - (q-1) = O(N - \alpha)$

Analysis of the algorithm

Time

Step 1: Build the q-gram graph from the SLP.

- Let $\mathcal{S}_q$ be the set of nodes with a relevant substring
- $(q-1)+\sum_{X_i \in \mathcal{S}_q} Q(X_i) = (q-1)+\sum_{X_i \in \mathcal{S}_q} |r_{X_i}| - (q-1) = O(N-\alpha)$

Step 2: Turning the q-gram graph into a tree.

- Size of the q-gram graph is $O(k_{T,q})$
- Traversal visits a node at most twice and the tree is extended with q nodes
- $O(q + k_{T,q})$ time

Analysis of the algorithm

Time

Step 1: Build the q-gram graph from the SLP.

- Let $\mathcal{S}_q$ be the set of nodes with a relevant substring
- $(q-1)+\sum_{X_i \in \mathcal{S}_q} Q(X_i) = (q-1)+\sum_{X_i \in \mathcal{S}_q} |r_{X_i}| - (q-1) = O(N-\alpha)$

Step 2: Turning the q-gram graph into a tree.

- Size of the q-gram graph is $O(k_{T,q})$
- Traversal visits a node at most twice and the tree is extended with q nodes
- $O(q+k_{T,q})$ time

Step 3: Constructing a suffix tree from the tree.

- The suffix tree of a tree can be constructed in linear time in the size of the tree (for integer alphabets) [Shibuya, 2003]
- Size of tree is $O(q+k_{T,q})$ so $O(q+k_{T,q})$ time

Analysis of the algorithm

Time

Step 1: Build the q-gram graph from the SLP.

- Let $\mathcal{S}_q$ be the set of nodes with a relevant substring
- $(q-1)+\sum_{X_i \in \mathcal{S}_q} Q(X_i) = (q-1)+\sum_{X_i \in \mathcal{S}_q} |r_{X_i}| - (q-1) = O(N-\alpha)$

Step 2: Turning the q-gram graph into a tree.

- Size of the q-gram graph is $O(k_{T,q})$
- Traversal visits a node at most twice and the tree is extended with q nodes
- $O(q+k_{T,q})$ time

Step 3: Constructing a suffix tree from the tree.

- The suffix tree of a tree can be constructed in linear time in the size of the tree (for integer alphabets) [Shibuya, 2003]
- Size of tree is $O(q+k_{T,q})$ so $O(q+k_{T,q})$ time

Step 4: Verification. Going from Monte Carlo to Las Vegas.

- Takes linear time in the size of the suffix tree so $O(q+k_{T,q})$ time

Analysis of the algorithm

Space

Step 1: Build the q-gram graph from the SLP.

- Data structures required to build the q-gram graph from the SLP: $O(n)$ space
- Size of the q-gram graph is $O(k_{T,q})$ so $O(n + k_{T,q})$

Analysis of the algorithm

Space

Step 1: Build the q-gram graph from the SLP.

- Data structures required to build the q-gram graph from the SLP: $O(n)$ space
- Size of the q-gram graph is $O(k_{T,q})$ so $O(n + k_{T,q})$

Step 2: Turning the q-gram graph into a tree.

- $O(q + k_{T,q})$ space

Analysis of the algorithm

Space

Step 1: Build the q-gram graph from the SLP.

- Data structures required to build the q-gram graph from the SLP: $O(n)$ space
- Size of the q-gram graph is $O(k_{T,q})$ so $O(n + k_{T,q})$

Step 2: Turning the q-gram graph into a tree.

- $O(q + k_{T,q})$ space

Step 3: Constructing a suffix tree from the tree.

- The number of leaves in the suffix tree is equal to the number of edges in the tree from step 2
- $O(q + k_{T,q})$ space

Analysis of the algorithm

Space

Step 1: Build the q-gram graph from the SLP.

- Data structures required to build the q-gram graph from the SLP: $O(n)$ space
- Size of the q-gram graph is $O(k_{T,q})$ so $O(n + k_{T,q})$

Step 2: Turning the q-gram graph into a tree.

- $O(q + k_{T,q})$ space

Step 3: Constructing a suffix tree from the tree.

- The number of leaves in the suffix tree is equal to the number of edges in the tree from step 2
- $O(q + k_{T,q})$ space

Step 4: Verification. Going from Monte Carlo to Las Vegas.

- $O(q + k_{T,q})$ space

Analysis of the algorithm

Space

Step 1: Build the q-gram graph from the SLP.

- Data structures required to build the q-gram graph from the SLP: $O(n)$ space
- Size of the q-gram graph is $O(k_{T,q})$ so $O(n + k_{T,q})$

Step 2: Turning the q-gram graph into a tree.

- $O(q + k_{T,q})$ space

Step 3: Constructing a suffix tree from the tree.

- The number of leaves in the suffix tree is equal to the number of edges in the tree from step 2
- $O(q + k_{T,q})$ space

Step 4: Verification. Going from Monte Carlo to Las Vegas.

- $O(q + k_{T,q})$ space

In total: $O(n + q + k_{T,q})$ space