Dagstuhl Seminar 13232

Algorithms on grammar compressed strings

Shunsuke Inenaga
Kyushu University, Japan

What we did after Dagstuhl Seminar 08261

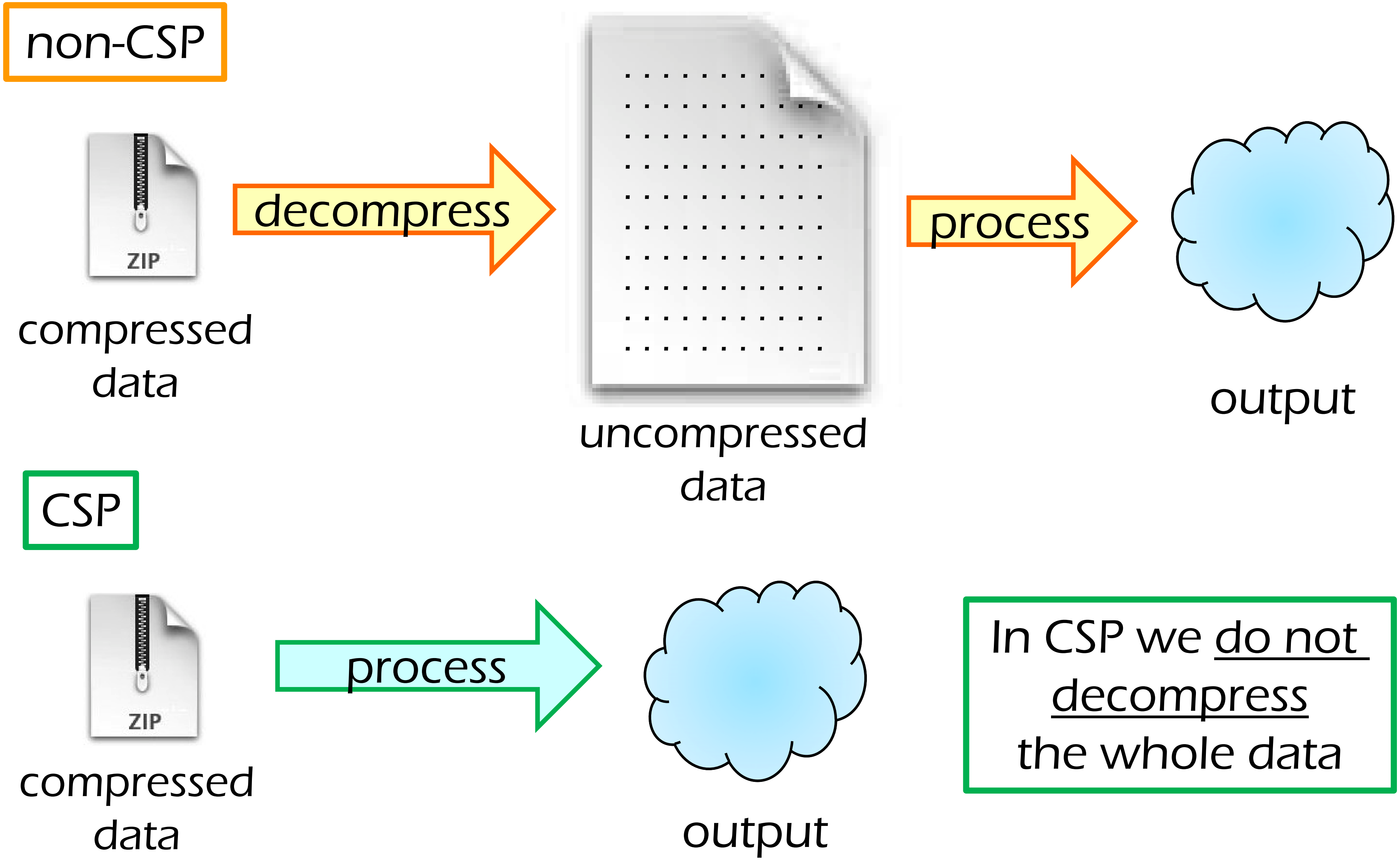
- ✓ In Dagstuhl Seminar 08261 (in 2008), I gave a survey talk about algorithmic results on grammar-based compressed strings, which were achieved before 2008.
- ✓ Today, I will talk about our new(er) results we achieved after 2008.

Collaborations

- ✓ Japanese:
  Hideo Bannai, Tomohiro I, Masayuki Takeda,
  Keisuke Goto, Yuto Nakashima, Kouji Shimohira,
  Takanori Yamamoto (Kyushu U.),
  Ayumi Shinohara, Kazuyuki Narisawa,
  Wataru Matsubara (Tohoku U.)

- ✓ International:
  Paweł Gawrychowski (Max Planck),
  Travis Gagie (U. Helsinki), Gad M. Landau (U. Haifa),
  Moshe Lewenstein (Bar Ilan U.)

Compressed String Processing (CSP)



Compressed String Processing [Cont.]

- ✓ Suppose that huge string data is stored in a compressed form.
- ✓ Given a compressed string, our goal is to perform various kinds of processing on the compressed string, without decompressing the whole string.
- ✓ Our input is a <u>straight-line program (SLP)</u>.

Straight Line Program (SLP)

An SLP is a sequence of productions

$$X_1 = expr_1,\ X_2 = expr_2,\ \cdots,\ X_n = expr_n$$

- $expr_i = a \qquad (a \in \Sigma)$
- $expr_i = X_l X_r \quad (l, r < i)$

✓ The size of the SLP is the number n of productions.

✓ An SLP is essentially a CFG deriving a single string.

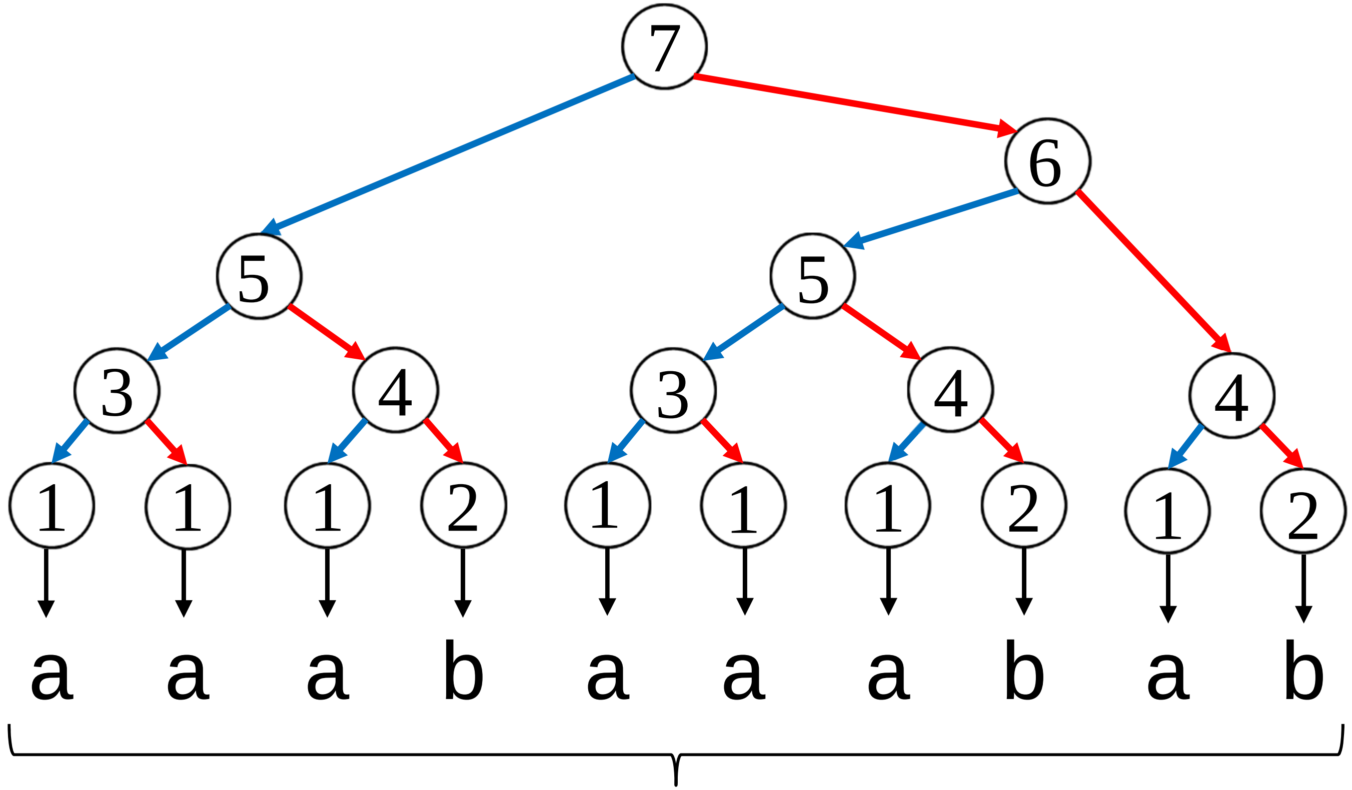
✓ SLPs model outputs of grammar-based compression algorithms (e.g., Re-pair, Sequitur, LZ78, etc).

Example of SLP

SLP S

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

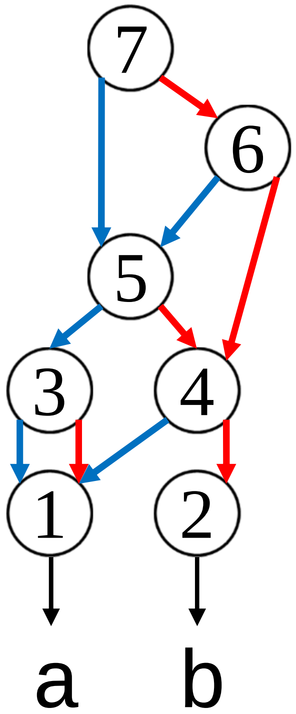
Derivation tree T of SLP S



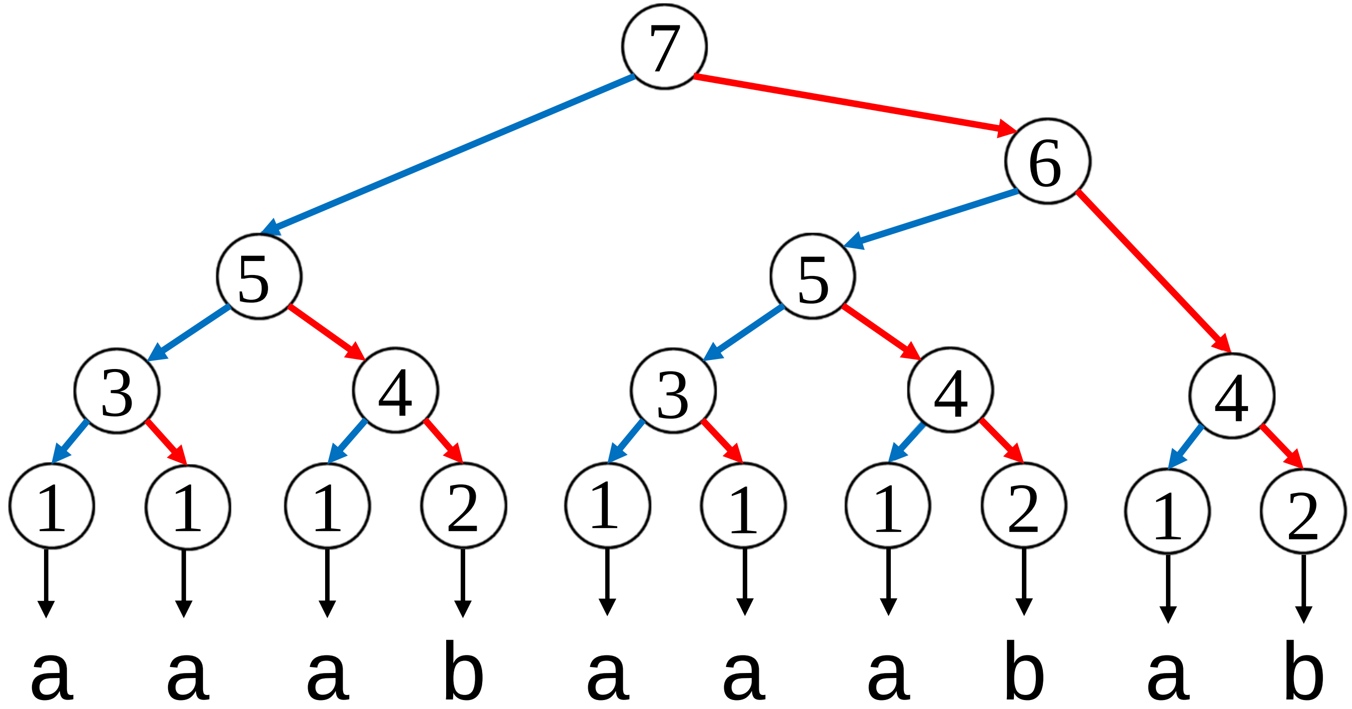
string represented by SLP S

DAG view of SLP

DAG for SLP S



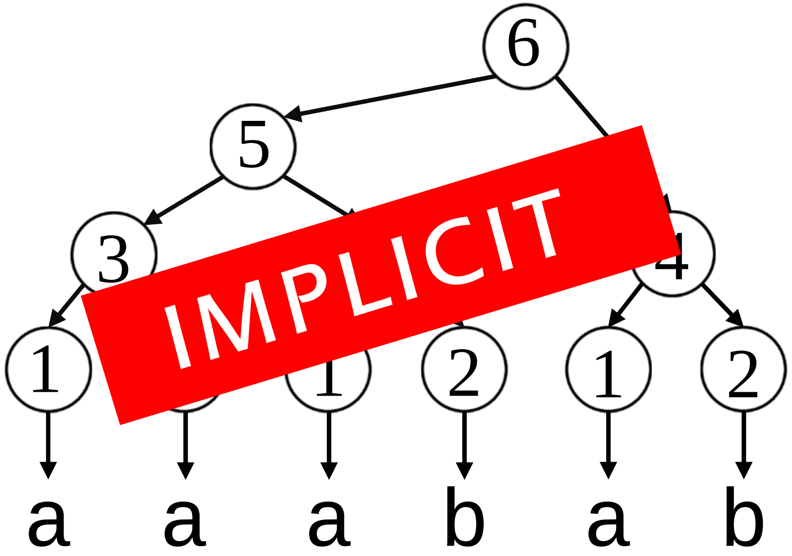
Derivation tree T of SLP S

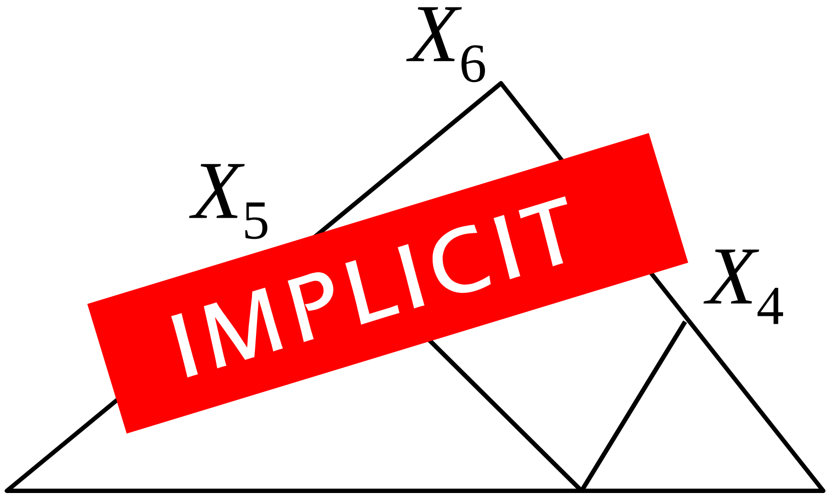


✓ DAG is compressed representation of derivation tree.

✓ SLP is compressed representation of string.

Important Remark





✓ Derivation trees are used only for explanations, and are never constructed in our algorithms.

✓ CSP on SLPs can be seen as algorithmic technique to perform various kinds of operations on the DAG for SLP, not on the derivation tree.

Notations

n : the size of a given SLP S

h : the height of the derivation tree T of S

N : the length of the decompressed string w
that is represented by SLP S

- ✓ $\log_2 N \leq h \leq n$ always holds.
- ✓ In theory, $N = O(2^n)$.
  - ➢ Solutions polynomial in n are beneficial.

Pattern Mining

problem	time	space (words)
q-gram frequencies	$O(qn)$	$O(qn)$
q-gram frequencies	$O(N\text{-}\alpha)$	$O(N\text{-}\alpha)$
q-gram non-overlapping frequencies	$O(q^2 n)$	$O(qn)$
longest repeating substring	$O(n^4 \log n)$	$O(n^3)$

$N\text{-}\alpha \leq \min(qn, N)$ always holds

SLP Text v.s Uncompressed Pattern

problem	time	space (words)
(window) subsequence matching	$O(nM)$	$O(nM)$
(window) VLDC pattern matching	$O(nM)$	$O(nM)$
convolution	$O((N\text{-}\beta) \log M)$	$O((N\text{-}\beta) \log M)$

- M is the length of uncompressed pattern
- $N\text{-}\beta \leq \min(nM, N)$ always holds

String Regularities

problem	time	space (words)
square freeness	$O(n^3 h \log N)$	$O(n^2)$
repetitions (runs & squares)	$O(n^3 h)$	$O(n^2)$
palindromes	$O(nh\,(n + h \log N))$	$O(n^2)$
gapped palindromes	$O(nh\,(n^2 + g \log N))$	$O(n\,(n + g))$
periods	$O(n^2 h)$	$O(n^2)$
covers	$O(nh\,(n + \log^2 N))$	$O(n^2)$

g is the fixed gap length

Factorization

problem	time	space (words)
LZ78 factorization	$O(n\sqrt{N} + s \log N)$	$O(n\sqrt{N} + s)$
LZ78 factorization	$O(n + s \log s)$	$O(n + s \log s)$
LZ77 factorization	$O(zn^2h \log N)$	$O(n^2 + z)$
Lyndon factorization	$O(n^4 + mn^3h)$	$O(n^2)$
Lyndon factorization	$O(nh\,(n + \log^2 N))$	$O(n^2)$

- s is the number of LZ78 factors
- z is the number of LZ77 factors
- m is the number of Lyndon factors

And Some Others

problem	time	space (words)
longest common substring	$O(n^4 \log n)$	$O(n^2 \log N)$
longest common extension	$O(n^3 h)$ preprocess $O(h \log N)$ query	$O(n^2)$
Aho-Corasick automaton	$O(n^4 \log n)$	$O(n^2 \log N)$

Our SLP-based Aho-Corasick automaton runs in $O(|u|\ (k + h + \log|\Sigma|))$ time on uncompressed text u, where k is the number of patterns.

q-gram Frequency on SLP

Problem 1 (q-gram frequencies on SLP)

Given an SLP S representing string w and a positive integer q, compute $Occ(w, p)$ for all substrings p of w of length q.

$Occ(w, p)$: the number of occurrences of p in w

Solution for Uncompressed String

✓ Given the uncompressed string w, we can solve the q-gram frequencies problem in $O(N)$ time, using the suffix array and LCP array of w.

$q = 3$

SA	LCP	
8	-	$
7	0	a$
5	1	aba$
3	3	ababa$
1	5	abababa$
6	0	ba$
4	2	baba$
2	4	bababa$

Solution for Uncompressed String

✓ Given the uncompressed string w, we can solve the q-gram frequencies problem in $O(N)$ time, using the suffix array and LCP array of w.

$q = 3$

SA	LCP	
8	-	$
7	0	a$
5	1	aba$
3	3	ababa$
1	5	abababa$
6	0	ba$
4	2	baba$
2	4	bababa$

Solution for Uncompressed String

✓ Given the uncompressed string w, we can solve the q-gram frequencies problem in $O(N)$ time, using the suffix array and LCP array of w.

$q = 3$

SA	LCP		
8	-		$
7	0		a$
5	1	<3	aba$
3	3	≥ 3	ababa$
1	5		abababa$
6	0		ba$
4	2		baba$
2	4		bababa$

Solution for Uncompressed String

✓ Given the uncompressed string w, we can solve the q-gram frequencies problem in $O(N)$ time, using the suffix array and LCP array of w.

$q = 3$

SA	LCP		
8	-		$
7	0		a$
5	1	<3	aba$
3	3	≥ 3	ababa$
1	5	≥ 3	abababa$
6	0		ba$
4	2		baba$
2	4		bababa$

Solution for Uncompressed String

✓ Given the uncompressed string w, we can solve the q-gram frequencies problem in $O(N)$ time, using the suffix array and LCP array of w.

$q = 3$

SA	LCP			Output $(pos, q, \#occ)$
8	-		\$	
7	0		a\$	
5	1	<3	aba\$	(5, 3, 3)
3	3	≥3	ababa\$	
1	5	≥3	abababa\$	
6	0	<3	ba\$	
4	2		baba\$	
2	4		bababa\$	

Solution for Uncompressed String

✓ Given the uncompressed string w, we can solve the q-gram frequencies problem in $O(N)$ time, using the suffix array and LCP array of w.

$q = 3$

SA	LCP		Suffix	Output (pos, q, $\#occ$)
8	-		\$	
7	0		a\$	
5	1		aba\$	(5, 3, 3)
3	3		ababa\$	
1	5		abababa\$	
6	0		ba\$	
4	2	<3	baba\$	(4, 3, 2)
2	4	≥ 3	bababa\$	

Solution for Uncompressed String

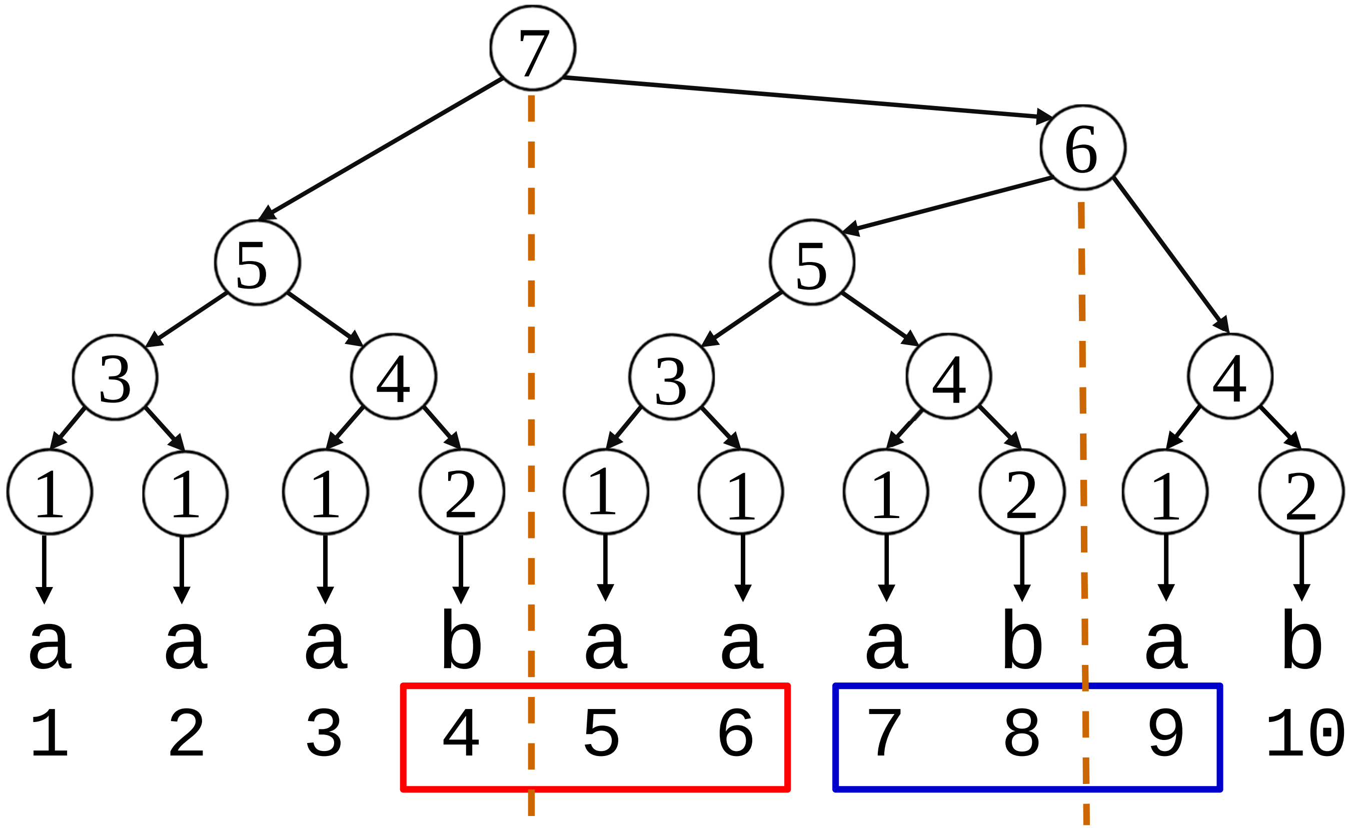
In the sequel, I will show how to simulate this $O(N)$-time algorithm in $O(qn)$ time.

$q = 3$

SA	LCP			Output $(pos, q, \#occ)$
8	-		\$	
7	0		a\$	
5	1		aba\$	(5, 3, 3)
3	3		ababa\$	
1	5		abababa\$	
6	0		ba\$	
4	2	<3	baba\$	(4, 3, 2)
2	4	≥ 3	bababa\$	

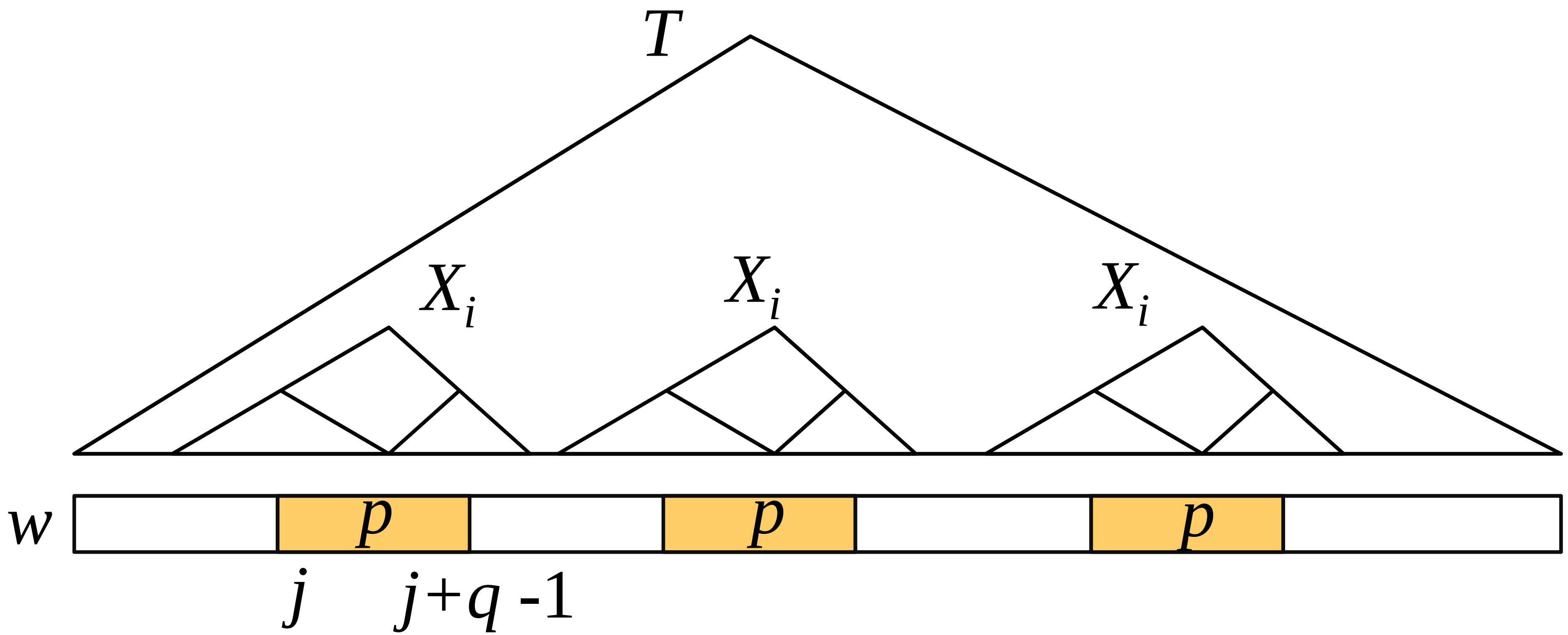
Stab

An integer interval $[b, e]$ $(1 \leq b \leq e \leq N)$ is said to be stabbed by a variable X_i, if the LCA of the bth and eth leaves of the derivation tree T is labeled by X_i.



Observation

- ✓ Assume that the occurrence of a q-gram p starting at position j is stabbed by variable X_i.
- ✓ Then, in any other occurrence of X_i in T, there is another stabbed occurrence of p.



Sub-problems

- ✓ Hence, the q-gram frequencies problem on SLP reduces to the following sub-problems:

Problem 2

For each variable X_i, count the number of occurrences of X_i in the derivation tree T.

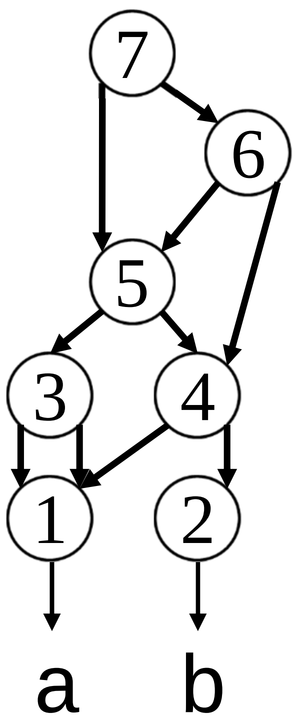
Problem 3

For each variable X_i, count the number of occurrences of each q-gram stabbed by X_i.

Solving Problem 2

Lemma 1

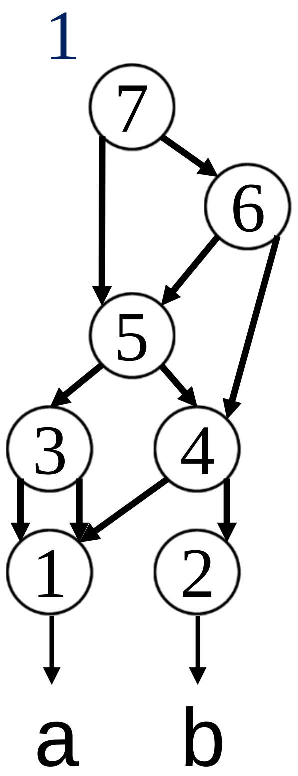
Problem 2 can be solved in $O(n)$ time.



Solving Problem 2

Lemma 1

Problem 2 can be solved in $O(n)$ time.

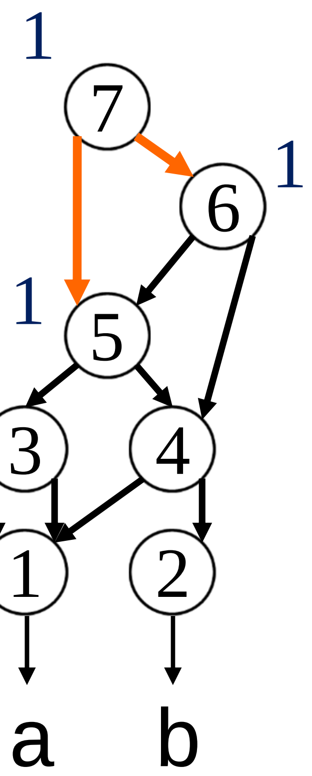


✓ The root occurs exactly once.

Solving Problem 2

Lemma 1

Problem 2 can be solved in $O(n)$ time.

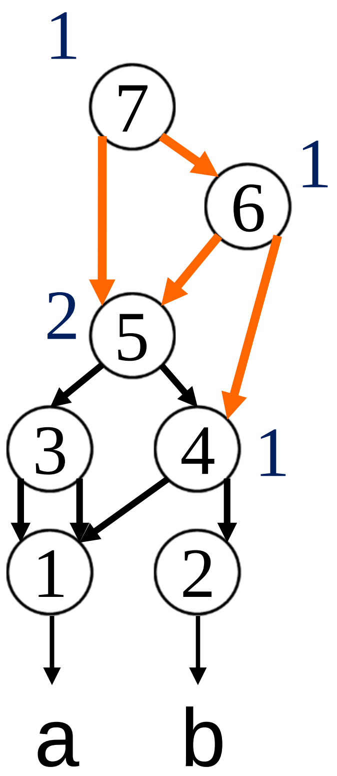


✓ For each node in a topological order, propagate its number of occurrences to its children.

Solving Problem 2

Lemma 1

Problem 2 can be solved in $O(n)$ time.

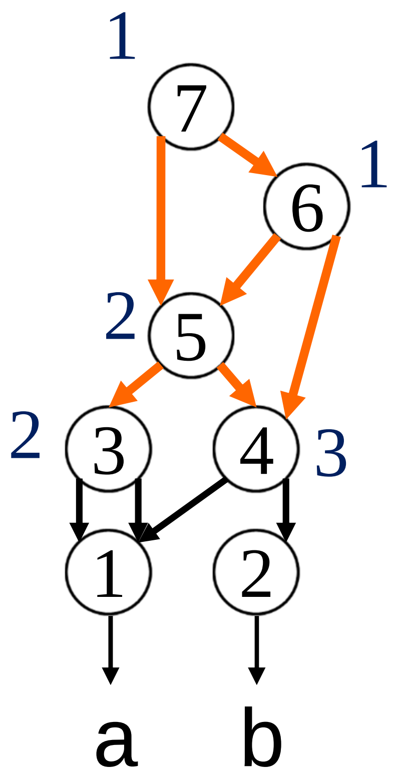


- ✓ For each node in a topological order, propagate its number of occurrences to its children.

Solving Problem 2

Lemma 1

Problem 2 can be solved in $O(n)$ time.

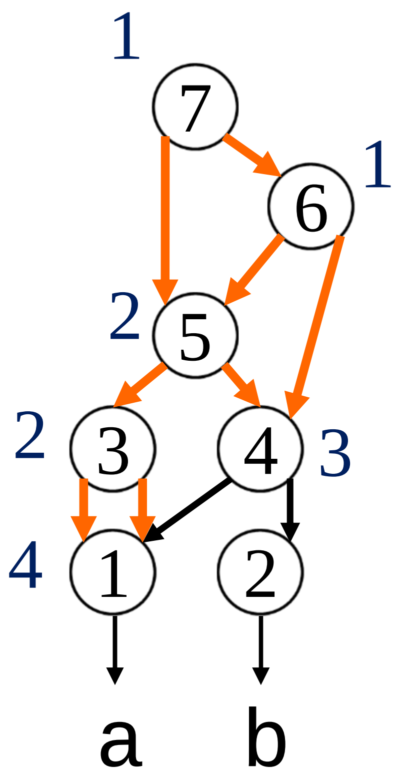


- ✓ For each node in a topological order, propagate its number of occurrences to its children.

Solving Problem 2

Lemma 1

Problem 2 can be solved in $O(n)$ time.

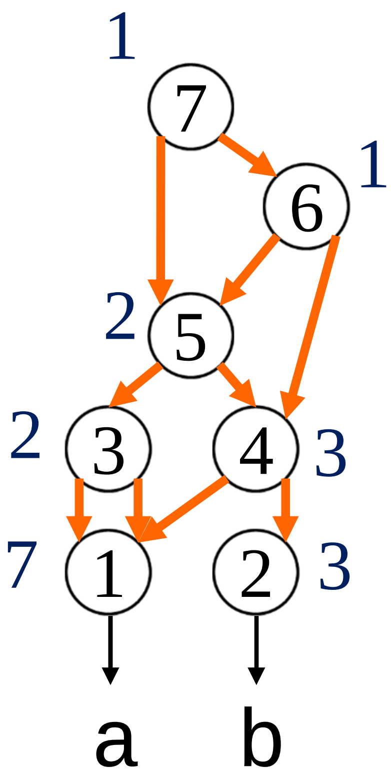


- ✓ For each node in a topological order, propagate its number of occurrences to its children.

Solving Problem 2

Lemma 1

Problem 2 can be solved in $O(n)$ time.

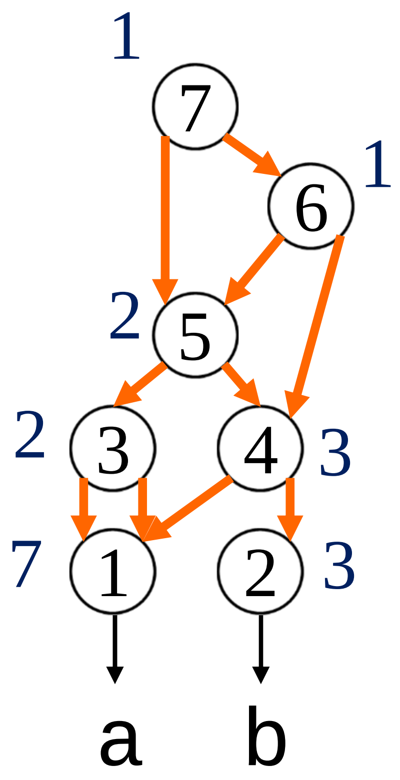


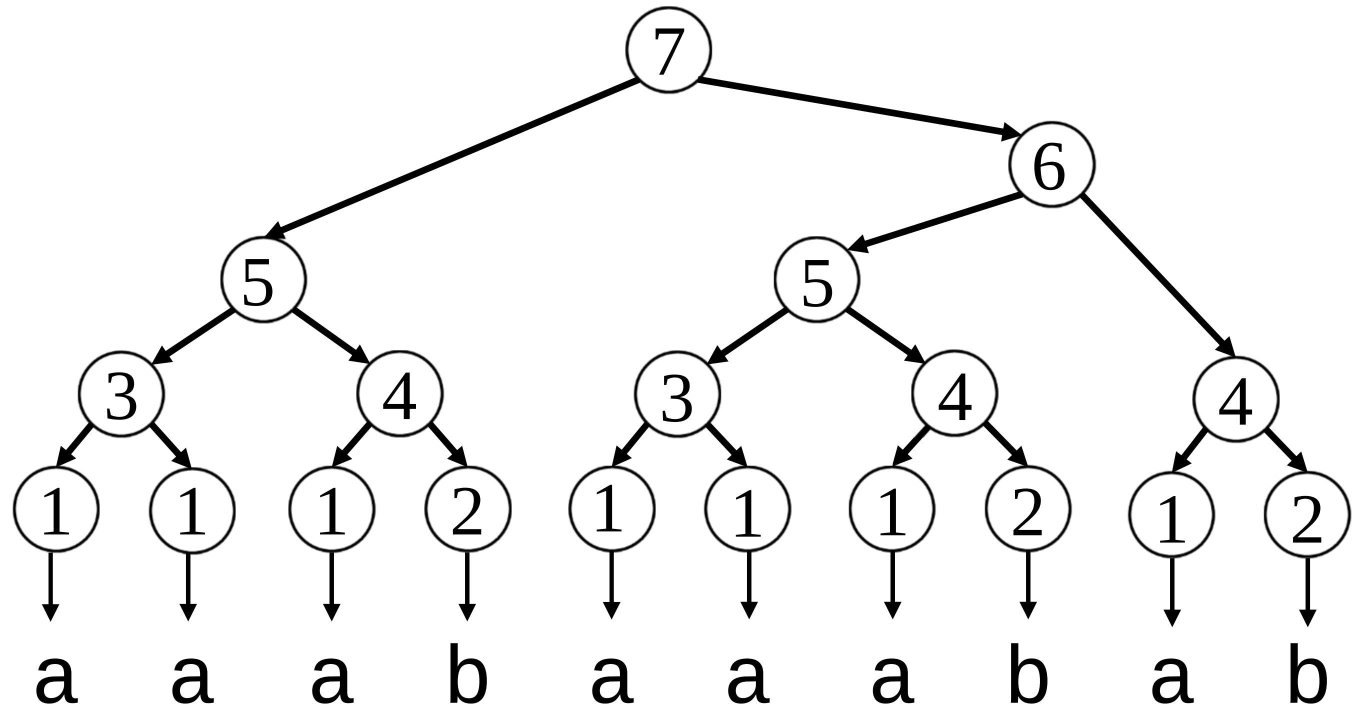
- ✓ For each node in a topological order, propagate its number of occurrences to its children.

Solving Problem 2

Lemma 1

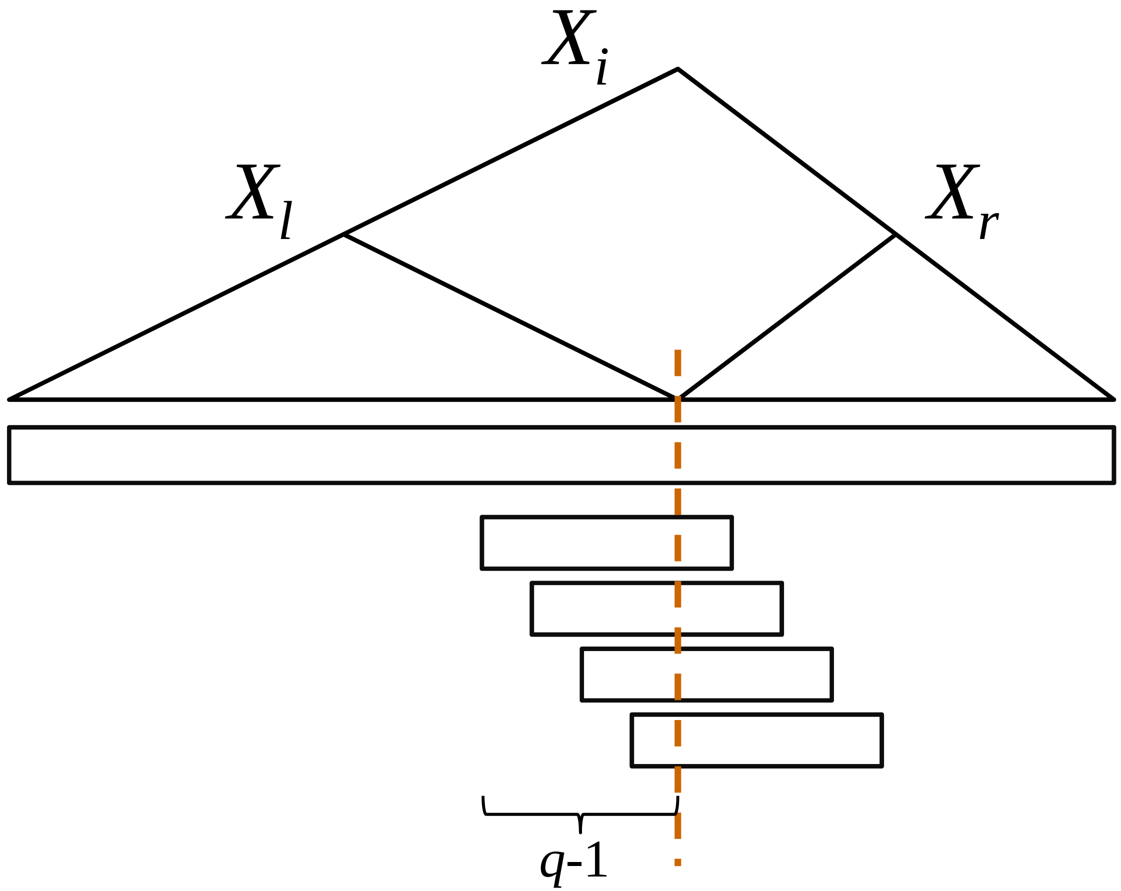
Problem 2 can be solved in $O(n)$ time.





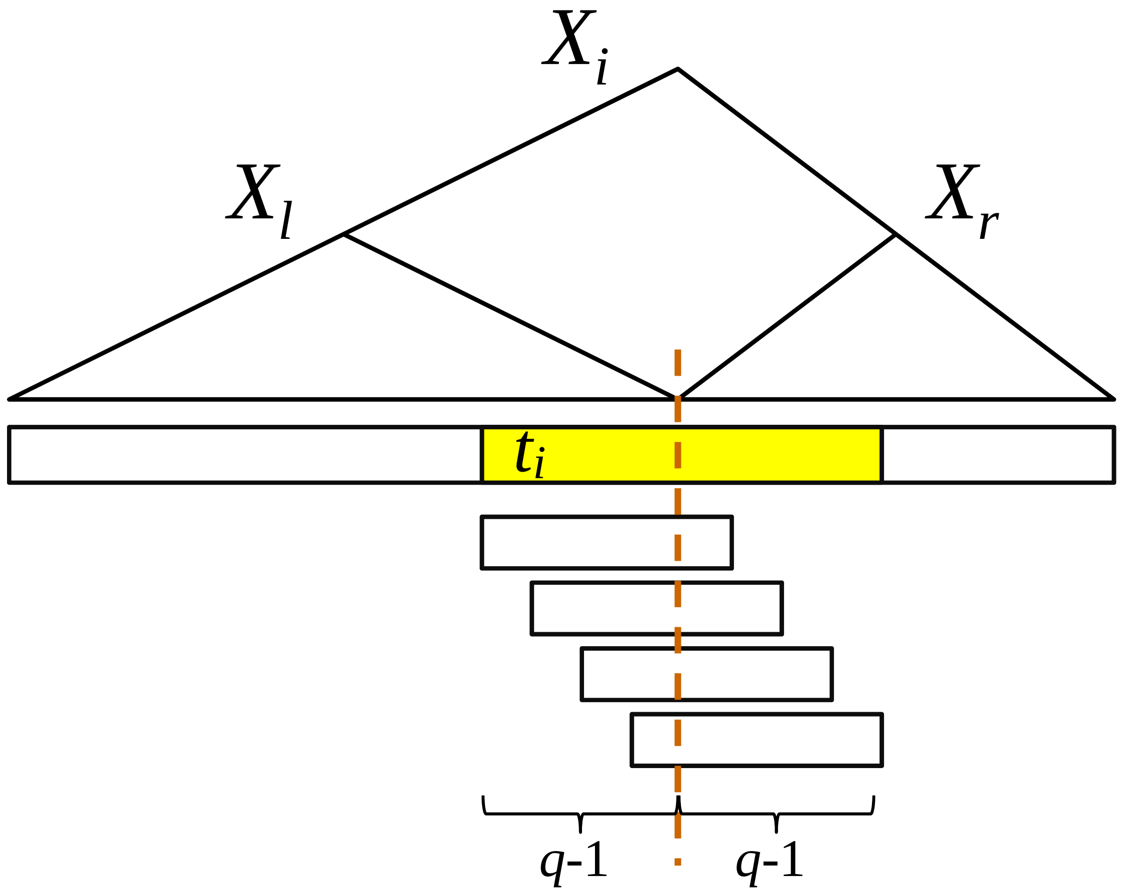
Solving Problem 3

- ✓ Each variable X_i can stab at most q-1 occurrences of q-grams.



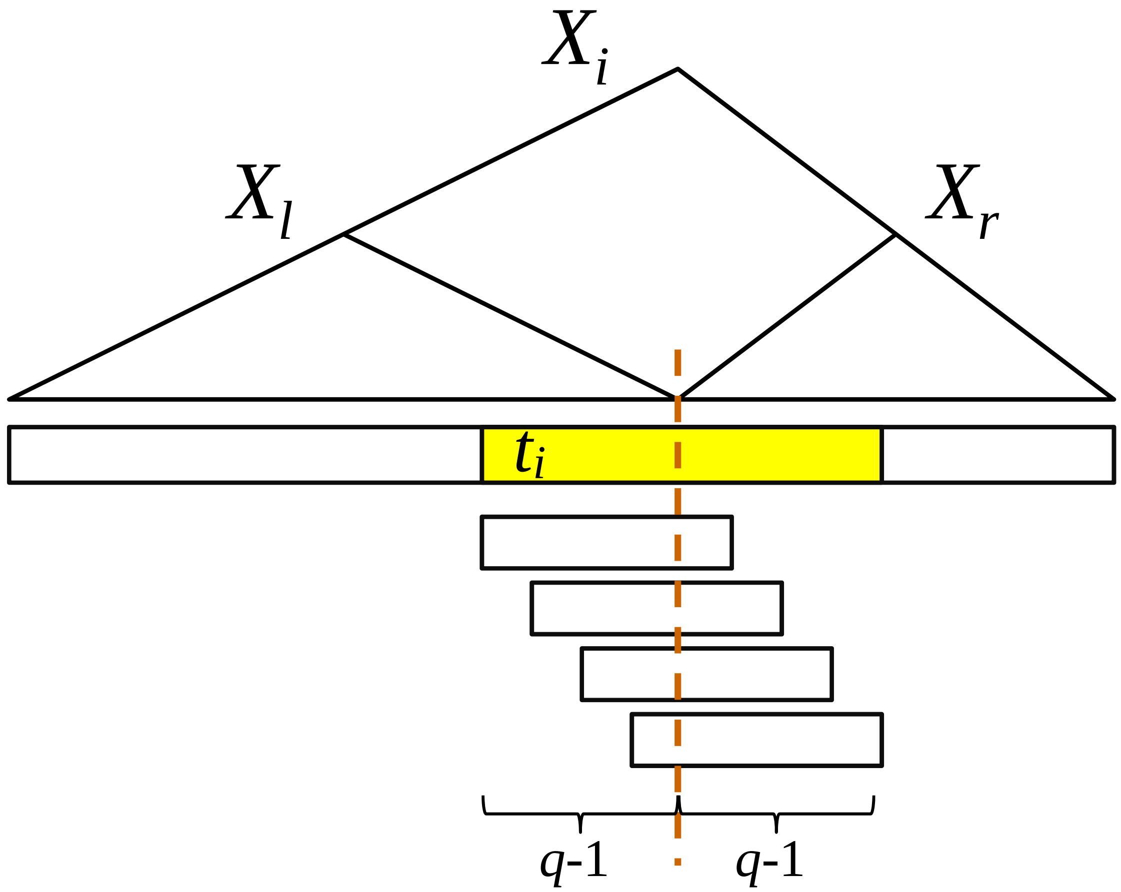
Solving Problem 3

✓ We decompress substring $t_i = X_l[|X_l|\text{-}q+2..|X_l|]\ X_r[1..q\text{-}1]$ of length $2q$-2.



Solving Problem 3

- ✓ Clearly, all q-grams stabbed by X_i occur inside t_i.



Solving Problem 3

Lemma 2

Problem 3 can be solved in $O(qn)$ time.

- ✓ For all variables X_i, substring t_i can be computed in a total of $O(qn)$ time, by a simple DP.
- ✓ We construct the suffix array and LCP array for string $z = t_1 t_2 \ldots t_n$, in $O(|z|) = O(qn)$ time.

q-gram Frequency on SLP

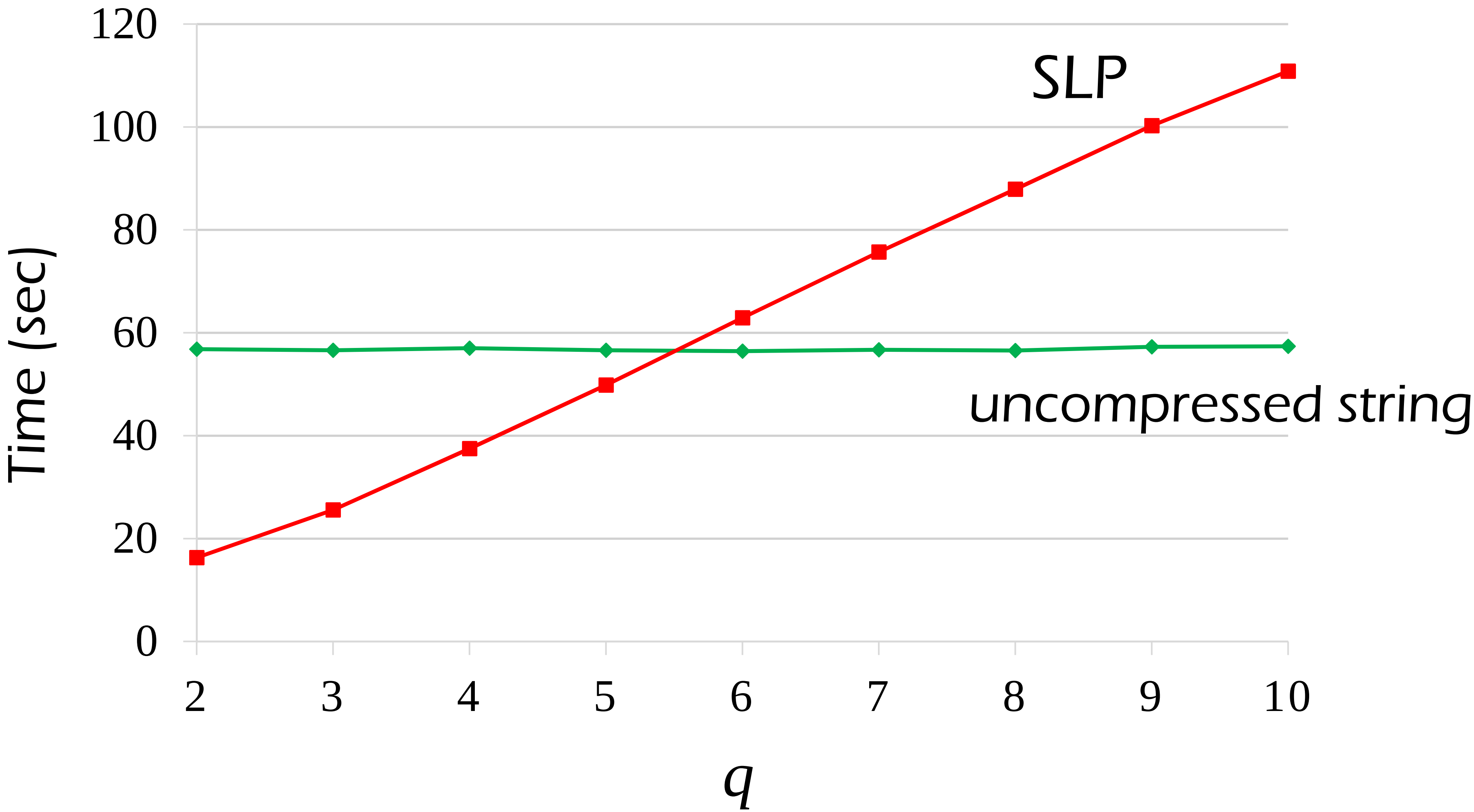
Theorem 1 [JDA 2013]

Problem 1 (q-gram frequencies on SLP) can be solved in $O(qn)$ time.

✓ Easily follows from Lemma 1 and Lemma 2.

Experimental Result

English text (200MB) from Pizza & Chili corpus



Improved Algorithm for Larger q

- ✓ For smaller values of q,
  our $O(qn)$ solution overcomes the $O(N)$ solution
  <u>both in theory and in practice</u>.

- ✓ Is it possible to improve our solution so that
  it works efficiently for larger values of q?

- ✓ At least in theory, the answer is yes!

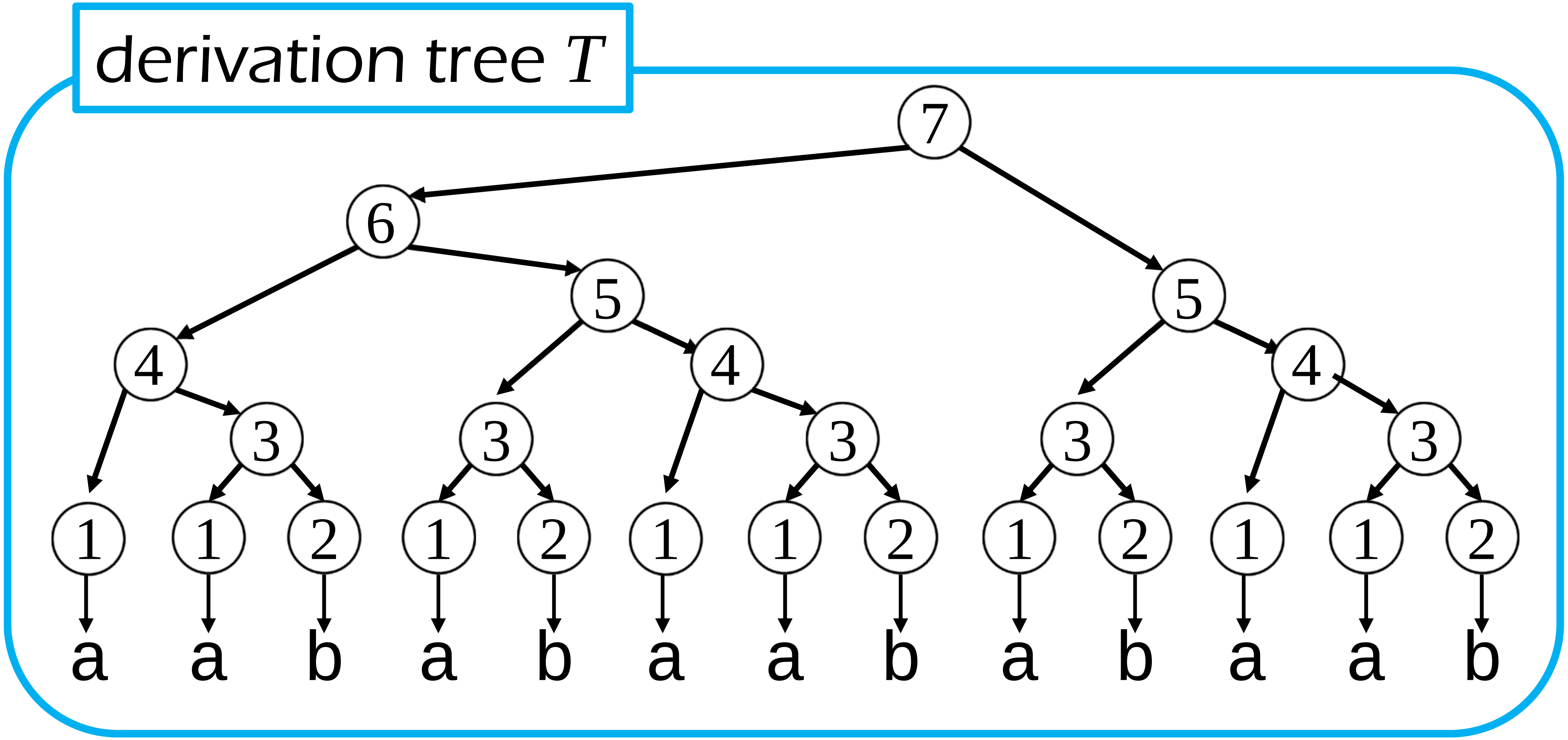
Improved Algorithm for Larger q

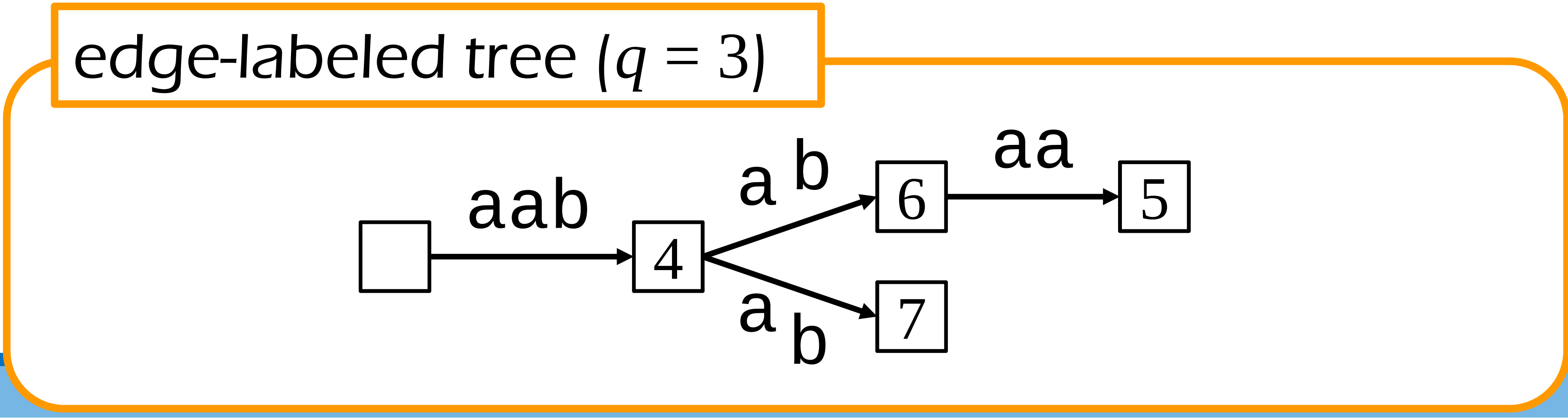
Lemma 3

We can construct, in linear time, an edge-labeled tree of size $O(N\text{-}\alpha)$ representing all q-grams which occur in w.

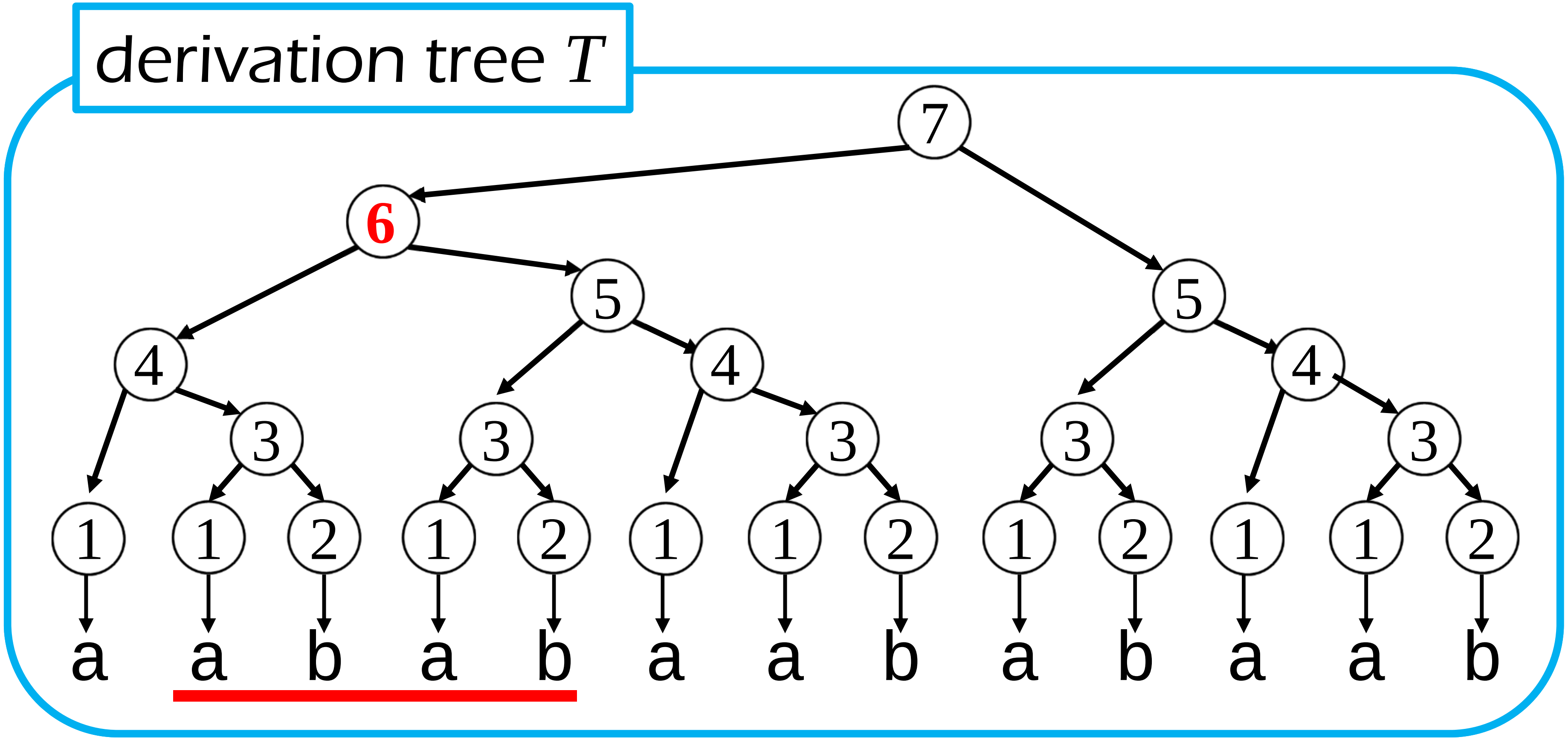
- ✓ $N\text{-}\alpha \leq \min(qn, N)$ denotes the total length of the edge labels of the tree.

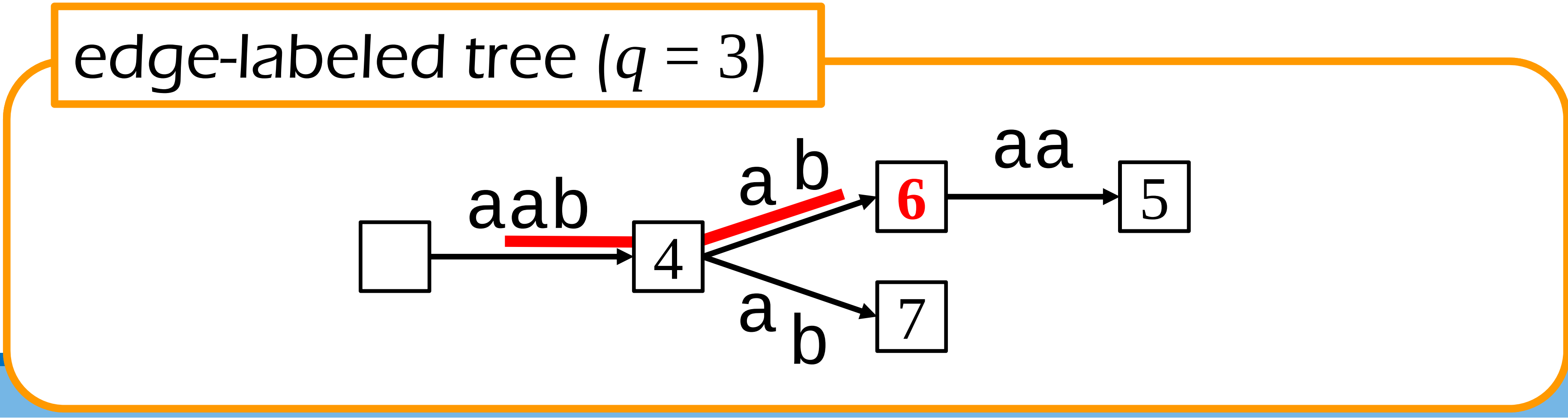
Example of Edge-Labeled Tree



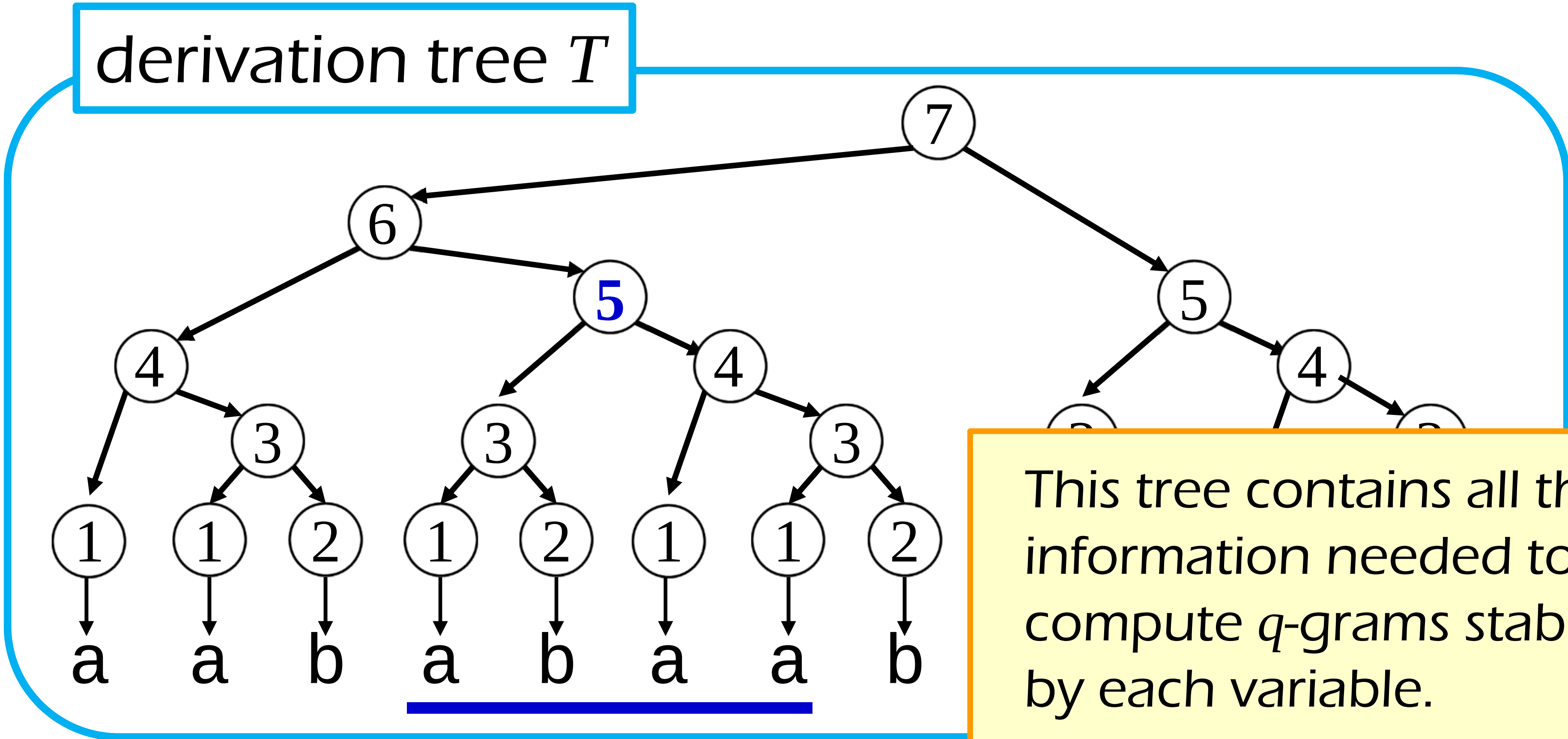


Example of Edge-Labeled Tree

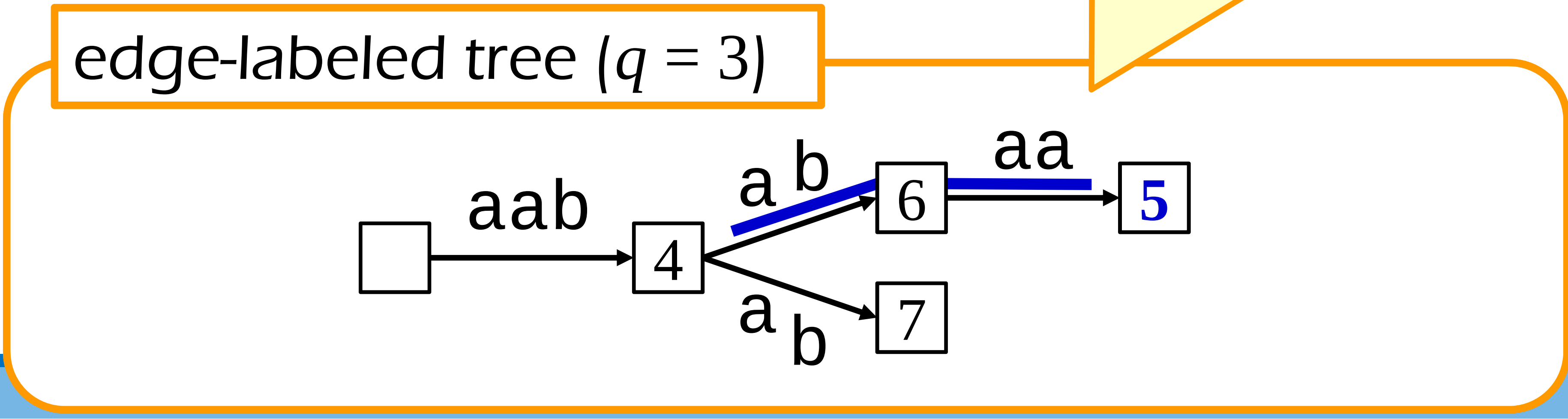




Example of Edge-Labeled Tree



This tree contains all the information needed to compute q-grams stabbed by each variable.



Improved Algorithm for larger q

Theorem 1 [CPM 2012]

Problem 1 (q-gram frequencies on SLP) can be solved in $O(N\text{-}\alpha) = O(\min(qn, N))$ time.

- ✓ We use a linear-time algorithm to construct the suffix tree of a tree (cf. Shibuya 2003).
- ✓ Our improved solution is at least as efficient as the $O(N)$-time solution, and can be much faster when q and n are small.

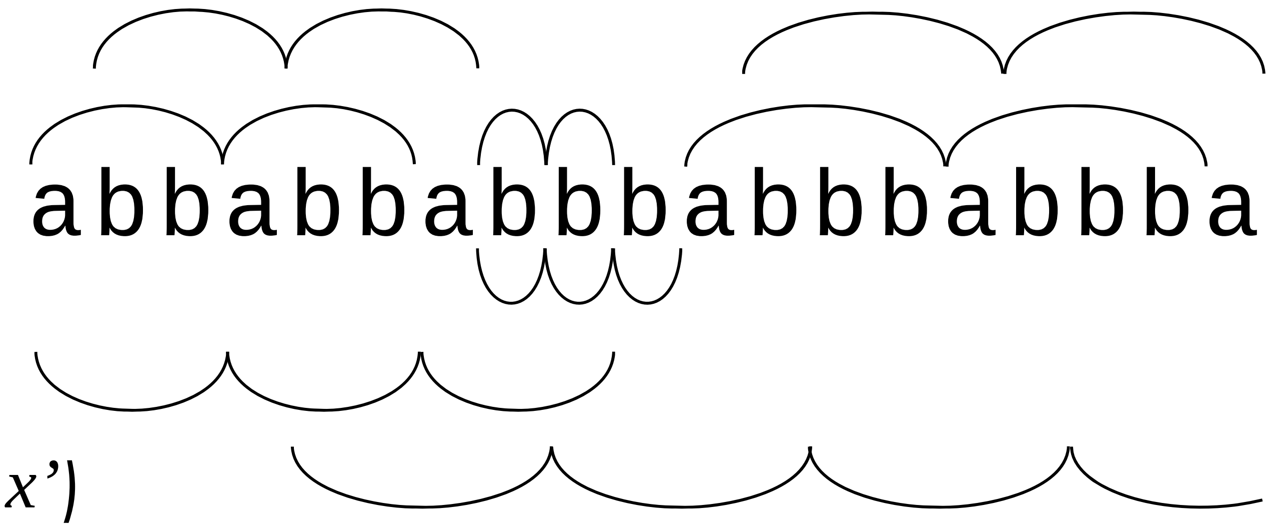
Finding Repetitions on SLP

Problem 4 (finding repetitions on SLP)

Given an SLP S representing string w, compute squares and runs that occur in w.

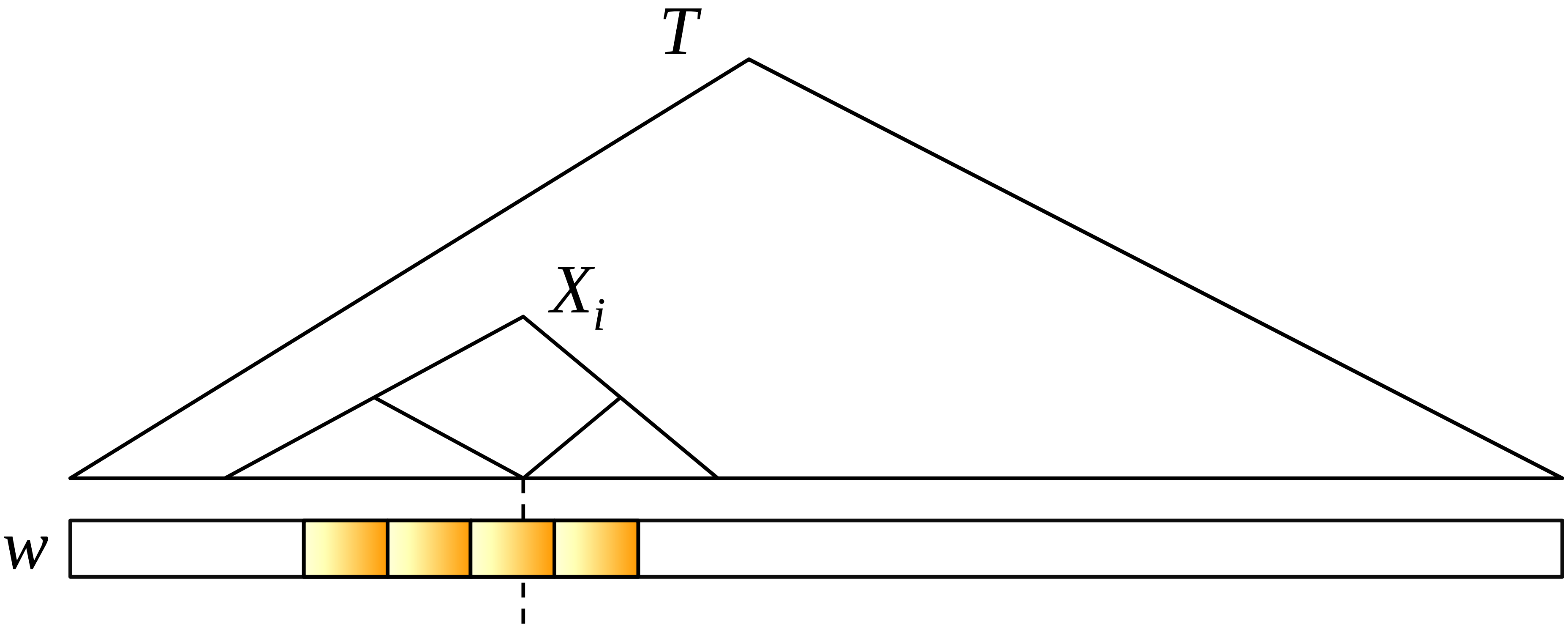
squares
(of form xx)

runs
(maximal repetition $x^k x'$)



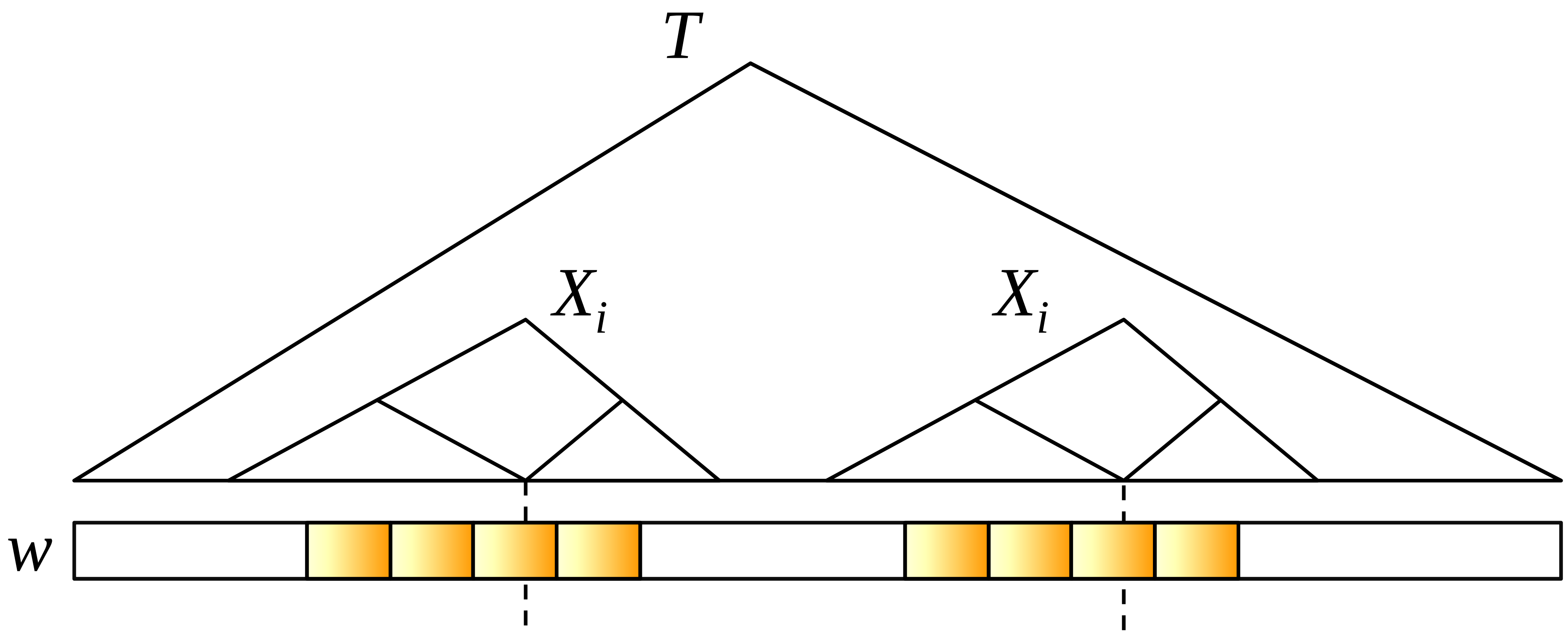
Stabbed Runs

- ✓ For each run in the string w, there is a unique variable X_i that stabs the run.



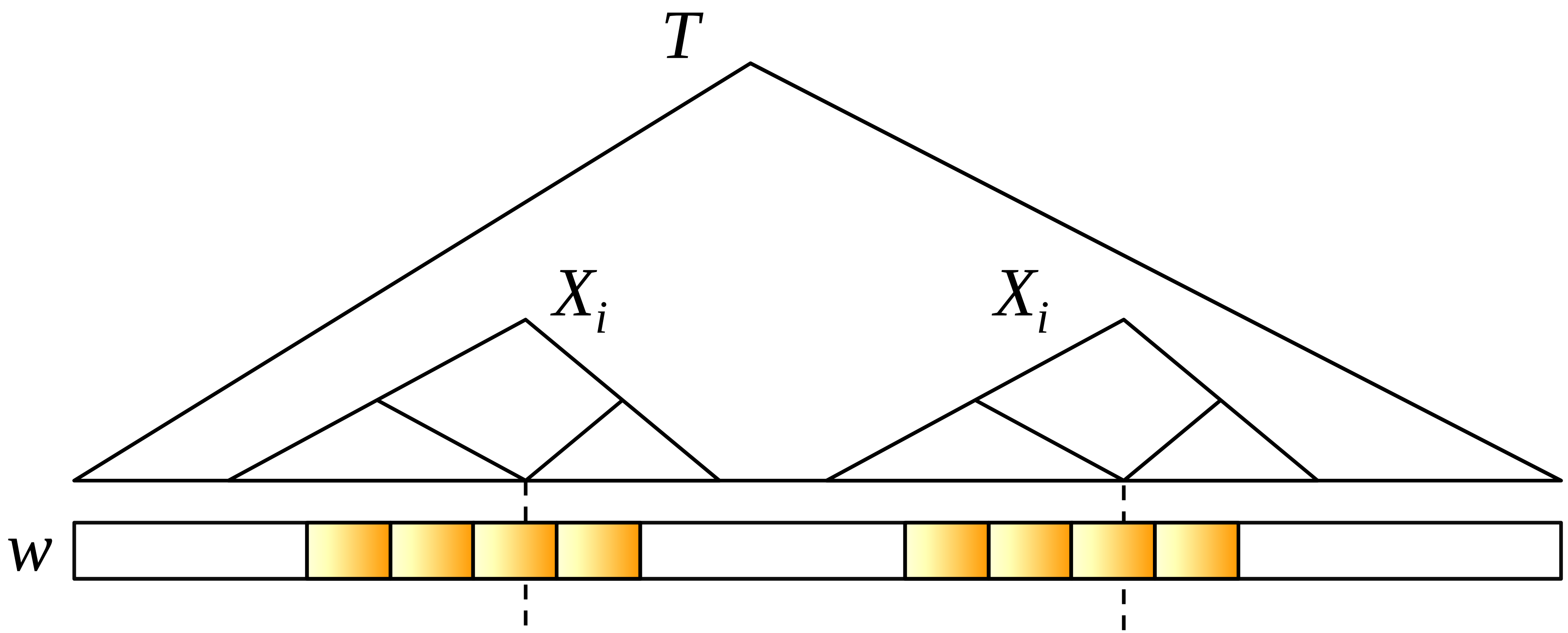
Stabbed Runs [Cont.]

- ✓ In other occurrences of X_i in the derivation tree, the same run is stabbed by X_i.



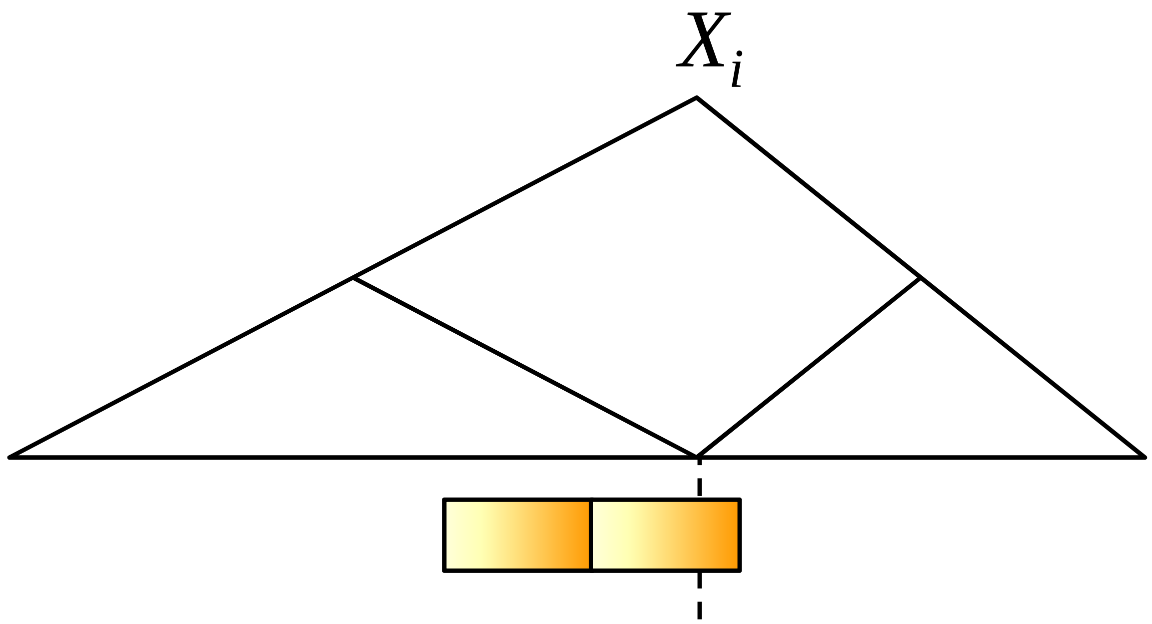
Stabbed Runs [Cont.]

- ✓ Computing runs in string w reduces to computing stabbed runs for each variable X_i.



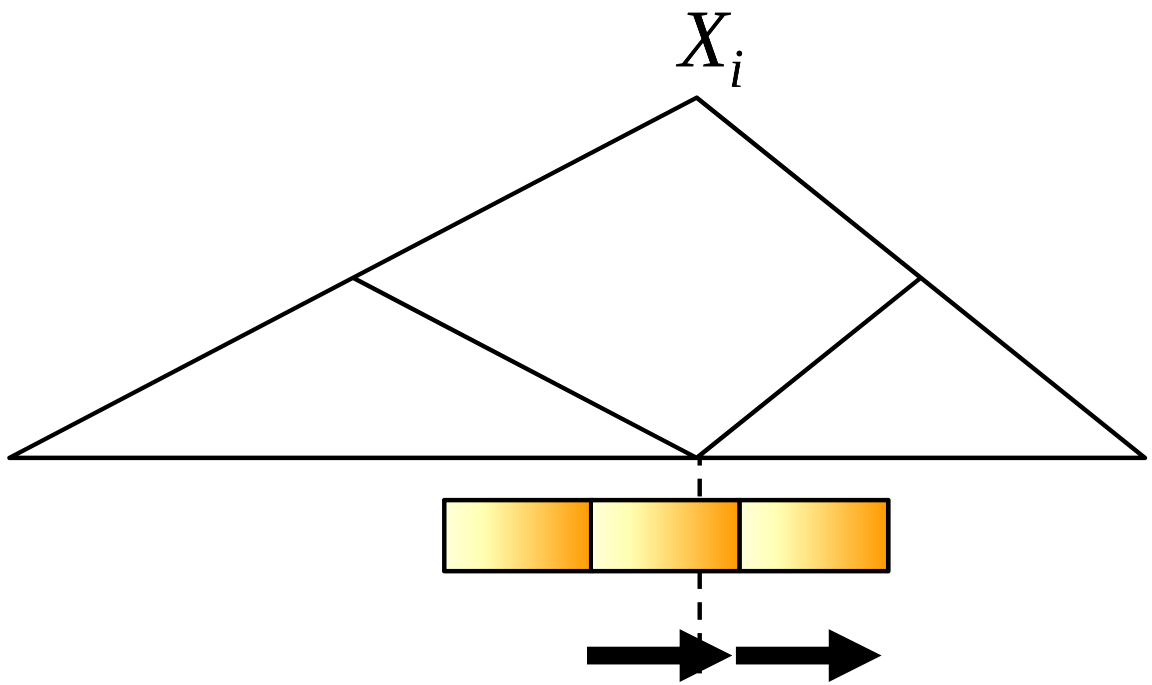
Stabbed Runs [Cont.]

- ✓ For each variable X_i, firstly we compute (the beginning and ending positions of) stabbed squares.



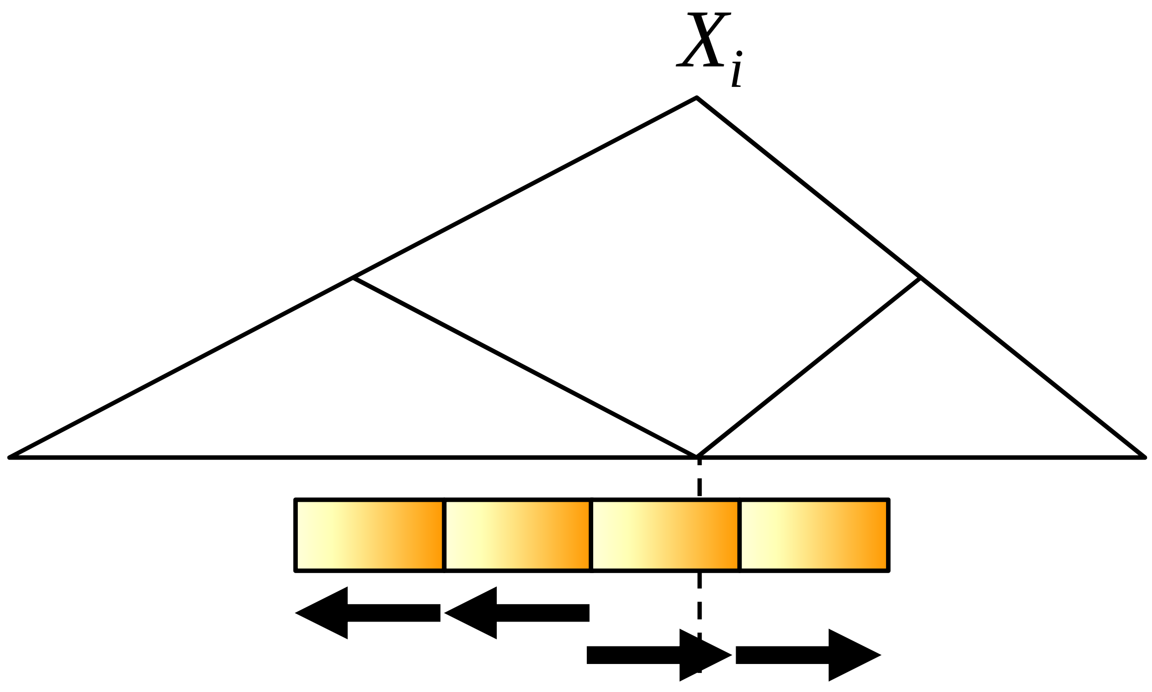
Stabbed Runs [Cont.]

- ✓ We then determine how long the periodicity continues to the right and to the left.
  - ➢ We can efficiently do this without decompressing X_i.



Stabbed Runs [Cont.]

- ✓ We then determine how long the periodicity continues to the right and to the left.
  - ➢ We can efficiently do this without decompressing X_i.



Finding Repetitions on SLP

Theorem 2 [MFCS 2013]

$\underline{O(n \log N)\text{-size representation}}$ of all runs and squares can be computed in $O(n^3 h)$ time using $O(n^2)$ space.

- ✓ There are $\Theta(N)$ runs in a string of length N.
  - ➢ Naïve representation of runs requires $O(2^n)$ space in the worst case.
- ✓ Hence we need a compact representation of output.

Compact Representation of Runs

Lemma 4

Our $O(n \log N)$-size representation of runs supports the following query in $O(h \log N)$ time:

Given an interval $[b, e]$ with $1 \leq b \leq e \leq N$, count the number of runs and squares that occur in the substring $w[b..e]$.

Finding Palindromes on SLP

Problem 5 (finding palindromes on SLP)

Given an SLP S representing string w,
compute maximal palindromes of w.

maximal
palindromes

a b b b a a b b b b a b b b a a b

Finding Palindromes on SLP

Problem 5 (finding palindromes on SLP)

Given an SLP S representing string w, compute maximal palindromes of w.

maximal palindromes

a b b b a a b b b b a b b b a a b

Finding Palindromes on SLP

Problem 5 (finding palindromes on SLP)

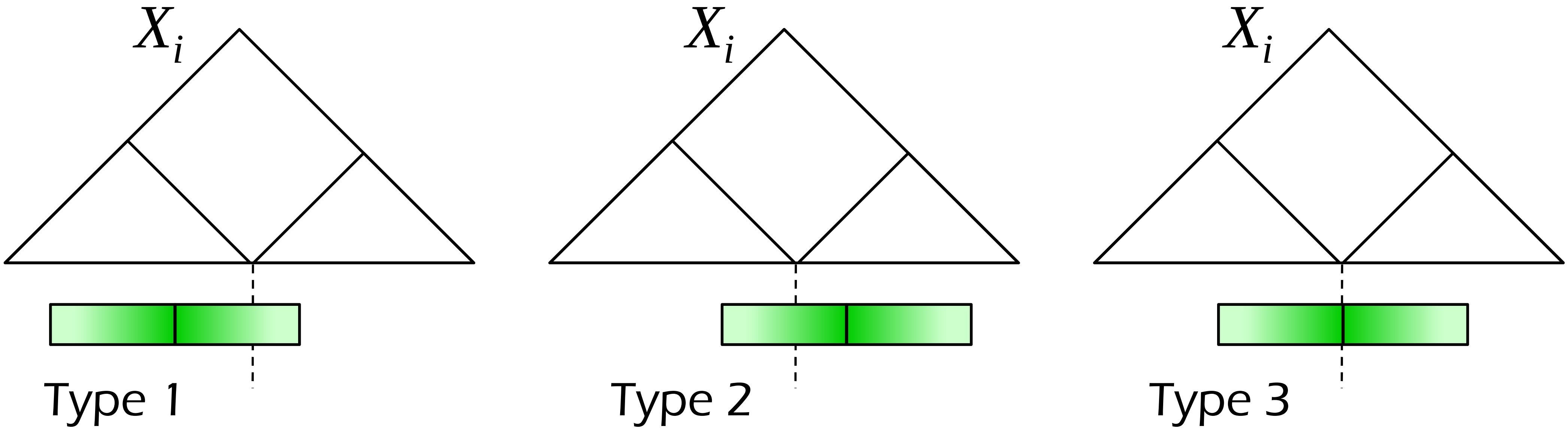
Given an SLP S representing string w, compute maximal palindromes of w.

maximal palindromes

a b b b a a b b b b a b b b a a b

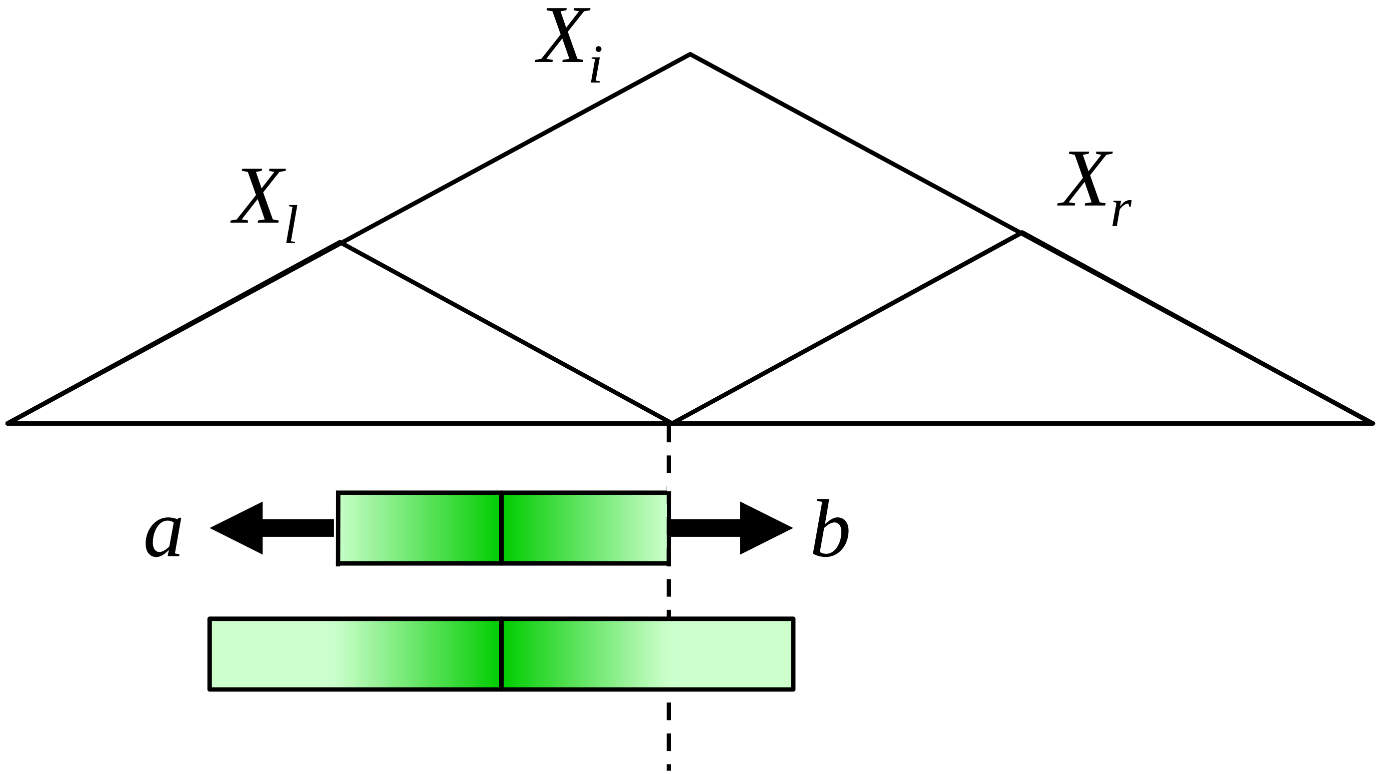
Stabbed Palindromes

✓ For each variable X_i, there can be 3 different types of stabbed maximal palindromes.



Computing Type 1 Palindromes

- ✓ Type 1 maximal palindromes of X_i can be computed by extending the arms of the <u>suffix palindromes</u> of X_l.



Suffix Palindromes

Lemma 5 [Apostolico et al., 1995]

For any string of length N, the lengths of its suffix palindromes can be represented by $O(\log N)$ arithmetic progressions.

- ✓ We can extend the suffix palindromes belonging to the same arithmetic progression in a batch, efficiently, using the periodicity.

Finding Palindromes on SLP

Theorem 3 [TCS 2009]

$\underline{O(n \log N)\text{-size representation}}$ of all maximal palindromes can be computed in $O(nh\,(n + h \log N))$ time using $O(n^2)$ space.

✓ The above time complexity is improved to $O(nh\,(n + \log^2 N))$ by using our recent LCE algorithm on SLP.

Finding Gapped Palindromes on SLP

Problem 6 (finding gapped palindromes on SLP)

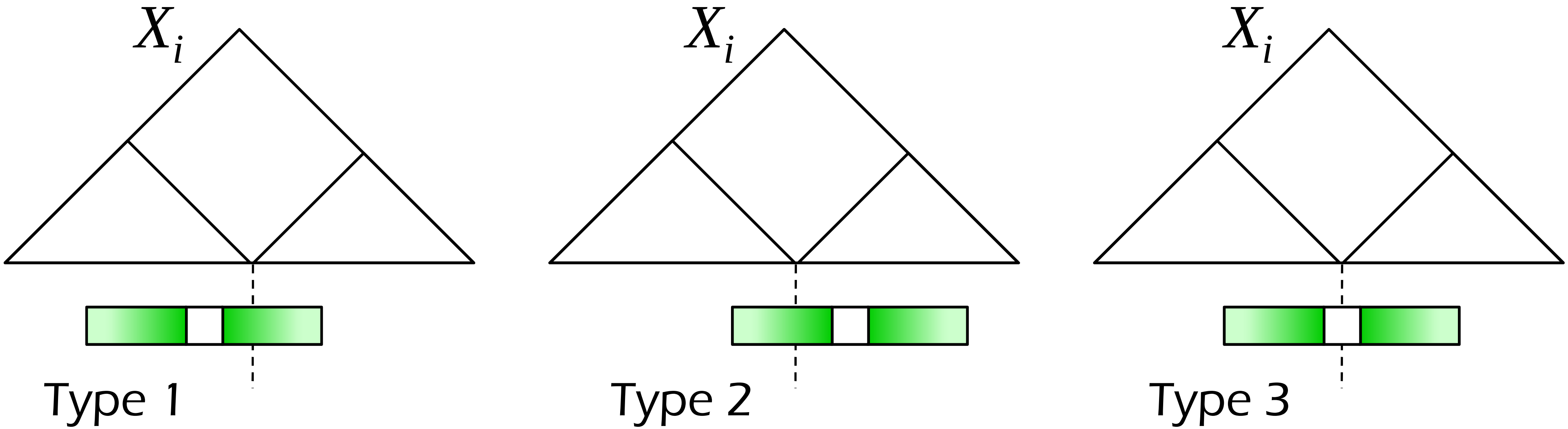
Given an SLP S representing string w and a positive integer g, compute g-gapped palindromes that occur in w.

3-gapped palindromes

a b a b a b c b a b a a b b a b c a

Stabbed *g*-gapped Palindromes

- ✓ There are 3 types of g-gapped palindromes stabbed by variable X_i.



Finding Gapped Palindromes on SLP

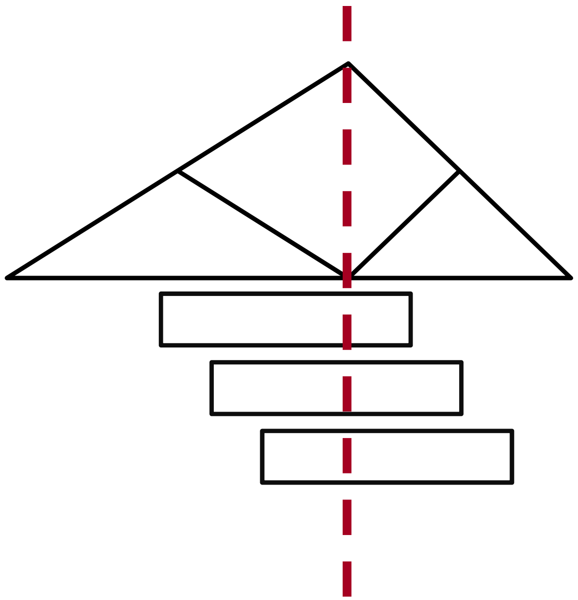
Theorem 4 [MFCS 2013]

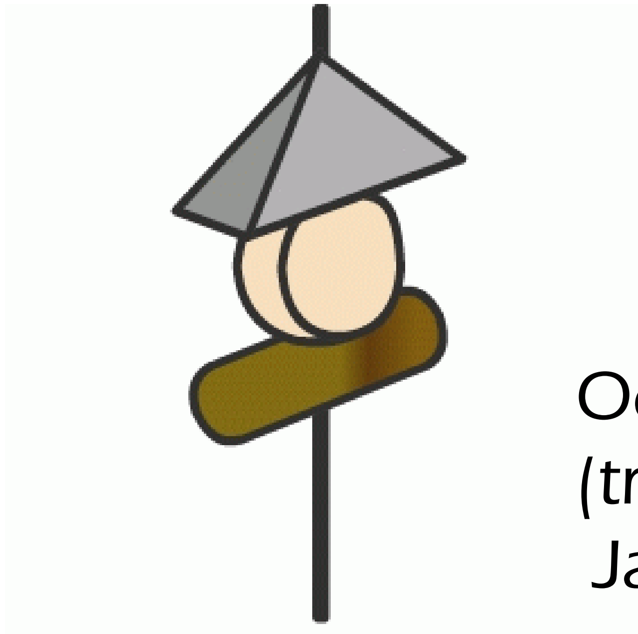
<u>$O(n\,(\log N + g))$-size representation</u> of all g-gapped palindromes can be computed in $O(nh\,(n^2 + g \log N))$ time using $O(n^2)$ space.

- ✓ Because of the gap between arms, we cannot use Lemma 5 (ar. pr. suffix palindromes).
- ✓ Instead, we used a similar technique to our solution for computing stabbed squares.

Concluding Remarks

- ✓ A number of string problems can be efficiently solved on SLP-compressed strings.
- ✓ The common key concept is stabbing, which we call “串 (kushi)”, a Japanese meaning a skewer.





Oden
(traditional
Japanese food)