Vol.2011-AL-134 No.11
2011/3/7

圧縮テキスト上での q-gram 非重複頻度の効率的な計算とその応用

Keisuke Goto ,[†1] Nami Fukui ,[†2] Hideo Bannai ,[†1]
Shunsuke Ikenaga [†3] and Masayuki Takeda [†1]

In many problems concerning text data, length-q substrings, or q-grams, can represent important characteristics of the data. Determining the frequencies of all q-grams contained in the data is an important problem with many applications. In this paper, we consider the problem of calculating the non-overlapping frequencies of all q-grams in a text represented as a straight line program (SLP). We show that the problem can be solved in $O(q^2n)$ time, where n is the size of the SLP. We also show an interesting application of the algorithm, which converts an arbitrary SLP to an SLP that is constructed based on frequency information.

1. Introduction

Storing text data in compressed form is becoming increasingly important in many applications, due to the huge amount of data that is being produced. To utilize the stored data, one would usually have to decompress it, and then run some algorithm on the uncompressed text. However, it would be much better if we could omit the decompression step, and efficiently operate on the compressed representation, without explicitly decompressing it. This concept of processing compressed text was first considered by Amir and Benson[1] for the pattern matching problem. Since then, there have been a lot of work on efficient processing on compressed texts, not only aiming for reduced space complexities, but for constructing algorithms which are practically faster than those which work on the uncompressed text.

A *Straight Line Program* (*SLP*)[7] is a context free grammar in the Chomsky normal form that derives a single string. SLPs are a widely accepted abstract model of various text compression schemes, since texts compressed by any grammar-based compression algorithm (e.g.[12),14)]) can be represented as SLPs, and those compressed by the LZ-family (e.g.[16),17)]) can be quickly transformed to SLPs. The size of an SLP of a string of length N can be as small as $O(\log N)$. SLPs are a promising representation of a given string, not only for reducing the storage size of the data, but for efficiently conducting various string processing operations[15]. Recently, even *self indices* based on SLPs have been developed[3].

The problem of calculating q-gram frequencies on SLP compressed strings has been considered previously[5]. When overlapping occurrences are *allowed* to be counted, an algorithm running in $O(|\Sigma|^q qn^2)$ time and $O(n^2)$ space was presented. Later, a much more efficiently algorithm running in $O(qn)$ time and space was developed[4]. For calculating *non-overlapping* q-gram frequencies, an algorithm that works only for 2-grams is known, which runs in $O(n^4 \log n)$ time and $O(n^3)$ space[5]. In this paper, we give an efficient solution for the non-overlapping q-gram frequency problem on SLP which runs in $O(q^2n)$ time and space, generalizing and greatly improving the time/space complexities of the previous solution. Our method builds on a similar idea used in the efficient algorithm allowing overlapping occurrences, but overcomes the difficulty that is introduced when not counting overlapping occurrences.

The algorithm can be used as a basis for efficiently performing interesting operations on compressed text, such as re-compressing compressed text as if using the Re-Pair algorithm[12].

2. Preliminaries

2.1 Notation

Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string T is denoted by $|T|$. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. A string of length $q > 0$ is called a *q-gram*. The set of q-grams is denoted by Σ^q. For a string $T = XYZ$, X, Y and Z are called a *prefix*, *substring*, and *suffix* of T, respectively. The i-th character of a string T is denoted by $T[i]$ for $1 \leq i \leq |T|$, and the substring of a string T that begins at position i and ends at position j is denoted by $T[i:j]$ for $1 \leq i \leq j \leq |T|$. For convenience, let $T[i:j] = \varepsilon$ if $j < i$. Let T^R denote the reversal of T, namely,

†1 Department of Informatics
†2 Department of Electrical Engineering and Computer Science
†3 Graduate School of Information Science and Electrical Engineering Kyushu University

$T^R = T[N]T[N-1]\cdots T[1]$, where $N = |T|$.

For an integer i and a set of integers A, let $i \oplus A = \{i + x \mid x \in A\}$ and $i \ominus A = \{i - x \mid x \in A\}$. If $A = \emptyset$, then let $i \oplus A = i \ominus A = \emptyset$.

2.2 Occurrences and Frequencies

For any strings T and P, let $Occ(T, P)$ be the set of occurrences of P in T, i.e.,

$$Occ(T, P) = \{k > 0 \mid T[k : k + |P| - 1] = P\}.$$

The number of occurrences of P in T, or the *frequency* of P in T is, $|Occ(T, P)|$. Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be *overlapping* if $k_1 + |P| - 1 \geq k_2$. Otherwise, they are said to be *non-overlapping*. The *non-overlapping frequency* $nOcc(T, P)$ of P in T is defined as the size of a largest subset of $Occ(T, P)$ where any two occurrences in the set are non-overlapping.

For any strings T and P, we define the sets of *right and left priority non-overlapping occurrences* of P in T, respectively, as follows:

$$RnOcc(T, P) = \begin{cases} \emptyset & \text{if } Occ(T, P) = \emptyset, \\ \{i\} \cup RnOcc(T[1 : i - 1], P) & \text{otherwise,} \end{cases}$$

$$LnOcc(T, P) = \begin{cases} \emptyset & \text{if } Occ(T, P) = \emptyset, \\ \{j\} \cup j \oplus LnOcc(T[j + 1 : |T|], P) & \text{otherwise.} \end{cases}$$

where $i = \max Occ(T, P)$ and $j = |P| - 1 + \min Occ(T, P)$. Note that $RnOcc(T, P) \subseteq Occ(T, P)$, $LnOcc(T, P) \subseteq (|P| - 1) \oplus Occ(T, P)$, and $RnOcc(T, P) = (|T| + 1) \ominus RnOcc(T^R, P^R)$.

Lemma 1 $nOcc(T, P) = |RnOcc(T, P)| = |LnOcc(T, P)|$

Proof. We prove $nOcc(T[1 : i], P) = |LnOcc(T[1 : i], P)|$ by induction on i. For $i \leq 1$, the statement clearly holds. Now, assume that the statement holds for $i < k$, where $k \geq 2$. For $i = k$, notice that $0 \leq nOcc(T[1 : k], P) - |LnOcc(T[1 : k], P) \leq 1$, since there can be at most one new occurrence of P ending at position i, which may or may not be counted for $nOcc(T[1 : k], P)$. If we assume on the contrary that the statement does not hold for $i = k$, then $nOcc(T[1 : k], P) - nOcc(T[1 : k - 1], P) = nOcc(T[1 : k], P) - |LnOcc(T[1 : k], P)| = 1$. Since the change was caused by the new occurrence, we have $nOcc(T[1 : k]) = nOcc(T[1 : k - |P|]) + 1$. By the inductive hypothesis, we have $nOcc(T[1 : k - |P|], P) = |LnOcc(T[1 : k - |P|], P)|$. Also, $|LnOcc(T[1 : k], P)| = |LnOcc(T[1 : k - |P|], P)| + 1$, since the new occurrence does not overlap with any occurrences in $LnOcc(T[1 : k - |P|])$. This leads to $nOcc(T[1 : k]) = |LnOcc(T[1 : k], P)|$, a contradiction. $nOcc(T, P) = |RnOcc(T, P)|$ can be shown symmetrically. □

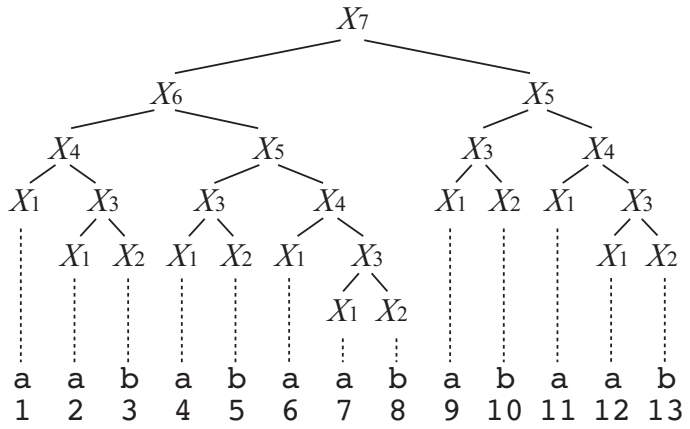

Fig. 1 The derivation tree of SLP $\mathcal{T} = \{X_i\}_{i=1}^{7}$ with $X_1 = \mathtt{a}$, $X_2 = \mathtt{b}$, $X_3 = X_1X_2$, $X_4 = X_1X_3$, $X_5 = X_3X_4$, $X_6 = X_4X_5$, and $X_7 = X_6X_5$, representing string $T = val(X_7) = \mathtt{aababaababaab}$.

Lemma 2 For any strings T and P, and any integer i with $1 \leq i \leq |T|$, let $ep = \max LnOcc(T[1 : i - 1], P)$ and $bp = i - 1 + \min RnOcc(T[i : |T|], P)$. Then $nOcc(T, P) = |LnOcc(T[1 : i - 1], P)| + |RnOcc(T[i : |T|], P)| + nOcc(T[ep + 1 : bp - 1], P)$.

Proof. By Lemma 1 and definition of ep, bp, $LnOcc$ and $RnOcc$, we have

$$\begin{aligned} nOcc(T, P) &= |LnOcc(T[1 : ep], P)| + |LnOcc(T[ep + 1 : |T|], P)| \\ &= |LnOcc(T[1 : ep], P)| + |RnOcc(T[ep + 1 : |T|], P)| \\ &= |LnOcc(T[1 : ep], P)| + |RnOcc(T[ep + 1 : bp - 1], P)| \\ &\quad + |RnOcc(T[bp : |T|], P)| \\ &= |LnOcc(T[1 : i - 1], P)| + nOcc(T[ep + 1 : bp - 1], P) \\ &\quad + |RnOcc(T[i : |T|], P)|. \end{aligned}$$

□

2.3 Straight Line Programs

In this paper, we treat strings described in terms of *straight line programs*

(*SLPs*). A straight line program $\mathcal{T}$ is a sequence of assignments such that

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

where each X_i is a variable and each $expr_i$ is an expression, where

- $expr_i = a \quad (a \in \Sigma)$, or
- $expr_i = X_\ell X_r \quad (\ell, r < i)$.

For assignment $X_i = X_\ell X_r$, we say that variables X_ℓ and X_r are children of variable X_i. Denote by T the string derived from the last variable X_n of the program $\mathcal{T}$. The *size* of the program $\mathcal{T}$ is the number n of assignments in $\mathcal{T}$. Note that $|T| = O(2^n)$.

Example 1 SLP $\mathcal{T} = \{X_i\}_{i=1}^{7}$ with $X_1 = \mathtt{a}$, $X_2 = \mathtt{b}$, $X_3 = X_1X_2$, $X_4 = X_1X_3$, $X_5 = X_3X_4$, $X_6 = X_4X_5$, and $X_7 = X_6X_5$ generates string $T = \mathtt{aababaababaab}$. (See Fig 1)

Let $val(X_i)$ represent the string derived from X_i. When it is not confusing, we identify a variable X_i with $val(X_i)$. Then, $|X_i|$ denotes the length of the string X_i derives.

The *substring intervals* of the string that each variable derives can be defined recursively as follows: $itv(X_n) = \{[1 : |T|]\}$, and $itv(X_i) = \{[u + |X_\ell| : v] \mid X_k = X_\ell X_i, [u : v] \in itv(X_k)\} \cup \{[u : u + |X_i| - 1] \mid X_k = X_i X_r, [u : v] \in itv(X_k)\}$ for $i < n$. For example, $itv(X_5) = \{[4 : 8], [9 : 13]\}$ in Fig. 1. Considering the transitive reduction of set inclusion, the intervals $\cup_{i=1}^{n} itv(X_i)$ naturally form a binary tree (the derivation tree).

Let $vOcc(X_i) = |itv(X_i)|$ denote the number of times a variable X_i occurs in the derivation of T. $vOcc(X_i)$ for all $1 \le i \le n$ can be computed in $O(n)$ time by a simple iteration on the variables, since $vOcc(X_n) = 1$ and for $i < n$, $vOcc(X_i) = \sum\{vOcc(X_k) \mid X_k = X_\ell X_i\} + \sum\{vOcc(X_k) \mid X_k = X_i X_r\}$.

2.4 Suffix Arrays and LCP Arrays

We will make use of the *suffix array* and *lcp array*. It is well known that the suffix array for any string of length $|T|$ can be constructed in $O(|T|)$ time[6),9),11)] assuming an integer alphabet. Given the text and suffix array, the lcp array can also be calculated in $O(|T|)$ time[8)].

Definition 1 (Suffix Arrays) The *suffix array*[13)] SA of any string T is an array of length $|T|$ such that $SA[i] = j$, where $T[j : |T|]$ is the i-th lexicographically smallest suffix of T.

Definition 2 (LCP Arrays) The *lcp* array of any string T is an array of length $|T|$ such that $LCP[i]$ is the length of the longest common prefix of $T[SA[i-1] : |T|]$ and $T[SA[i] : |T|]$ for $2 \le i \le |T|$, and $LCP[1] = 0$.

Problem 1 (weighted overlapping q-gram frequencies) Given a string T, an integer q, and integer array w ($|w| = |T|$), compute $\sum_{i \in Occ(T,P)} w[i]$ for all q-grams $P \in \Sigma^q$.

Theorem 1 Problem 1 can be solved in $O(|T|)$ time.

Proof. We can calculate the overlapping q-gram frequencies of string T using suffix array SA and lcp array LCP. $SA[i]$ represents an occurrence of a q-gram $T[SA[i] : SA[i] + q - 1]$. Since the suffixes are lexicographically sorted in the suffix array, intervals on the suffix array where the values of lcp array are at least q represent occurrence of the same q-gram. The sum of $w[SA[i]]$ in this interval is the desired value for the q-gram. Constructing SA, LCP can be done in $O(|T|)$ time, and summing up $w[SA[i]]$ for each interval where $LCP[i] \ge q$ can easily be done in $O(|T|)$ by a simple scan. □

3. Computing q-gram Non-Overlapping Frequencies

In this section we consider the following problem:

Problem 2 (Non-overlapping q-gram frequencies on SLP) Given an SLP $\mathcal{T}$ that describes string T and an integer q, compute $nOcc(T, P)$ for all q-grams $P \in \Sigma^q$.

If $q = 1$, then since no occurrences of a 1-gram overlap, the 1-gram non-overlapping frequency is simply the number of occurrences of the corresponding character in string T. This can be computed in a total of $O(n)$ time, since $nOcc(T, a) = \sum_{X_i = a} vOcc(X_i)$ for each $a \in \Sigma$.

If $q \ge 2$, we make use of Lemma 3 below. The idea is similar to the *mk Lemma*[2)] but Lemma 3 is more specifically tailored for our needs.

Lemma 3 Let $\mathcal{T} = \{X_i\}_{i=1}^{n}$ be an SLP that represents string T. For an interval $[u : v]$ ($1 \le u < v \le |T|$), there exists exactly one variable $X_i = X_\ell X_r$ such that for some $[u' : v'] \in itv(X_i)$, the following holds: $[u : v] \subseteq [u' : v']$, $u \in [u' : u' + |X_\ell| - 1] \in itv(X_\ell)$ and $v \in [u' + |X_\ell| : v'] \in itv(X_r)$.

Proof. Consider length 1 intervals $[u : u]$ and $[v : v]$ corresponding to leaves in the derivation tree. X_i corresponds to the lowest common ancestor of these intervals

in the derivation tree. □

In the following subsections, we firstly describe how to compute 2-gram non-overlapping frequencies, and secondly describe how to compute q-gram non-overlapping frequencies for $q \geq 3$, from a given SLP.

3.1 2-gram Non-Overlapping Frequencies

For any variable X_i and integer $1 \leq k \leq |X_i|$, let $pre(X_i, k) = val(X_i)[1 : \min\{k, |X_i|\}]$ and $suf(X_i, k) = val(X_i)[|X_i| - \min\{k, |X_i|\} + 1 : |X_i|]$. That is, $pre(X_i, k)$ and $suf(X_i, k)$ are the prefix and the suffix of $val(X_i)$ of length k, respectively. For all variables X_i, $pre(X_i, k)$ can be computed in a total of $O(nk)$ time and space, as follows:

$$pre(X_i, k) = \begin{cases} val(X_i) & \text{if } |X_i| \leq k, \\ pre(X_\ell, k)pre(X_r, k - |X_\ell|) & \text{if } X_i = X_\ell X_r \text{ and } |X_\ell| < k < |X_i|, \\ pre(X_\ell, k) & \text{if } X_i = X_\ell X_r \text{ and } k \leq |X_\ell|. \end{cases}$$

$suf(X_i, k)$ can be computed similarly in $O(nk)$ time and space.

Let $plen(X_i) = \max\{k \mid val(X_i)[j] = pre(X_i, 1), 1 \leq \forall j \leq k\}$ and $slen(X_i) = \min\{k \mid val(X_i)[j] = suf(X_i, 1), k \leq \forall j \leq |val(X_i)|\}$. That is, $plen(X_i)$ and $slen(X_i)$ are the length of the maximum runs of the first and the last characters of $val(X_i)$, respectively. We can compute $plen(X_i)$ for all variables X_i in a total of $O(n)$ time, as follows:

$$plen(X_i) = \begin{cases} 1 & \text{if } X_i = a, \\ |X_\ell| + plen(X_r) & \text{if } X_i = X_\ell X_r, plen(X_\ell) = |X_\ell|, X_\ell[1] = X_r[1], \\ plen(X_\ell) & \text{otherwise.} \end{cases}$$

$slen(X_i)$ can be computed similarly in $O(n)$ time.

Theorem 2 For $q = 2$, Problem 2 can be solved in $O(n)$ time.

Proof. Fig. 3 shows a pseudo-code of our algorithm to compute non-overlapping frequences of 2-grams from a given SLP.

Firstly, let us consider a 2-gram of form ab, where $a \neq b \in \Sigma$. It is clear that no occurrences of such a 2-gram overlap. Therefore, we simply compute the number of occurrences of ab. By Lemma 3, we have $nOcc(T, ab) = \sum vOcc(X_i)$, where $X_i = X_\ell X_r$ is any variable such that $suf(X_\ell, 1) = a$ and $pre(X_r, 1) = b$. This is done in line 8.

Now we consider a 2-gram of form aa, where $a \in \Sigma$. A key idea is to find an interval that corresponds to a maximal repetition of a in T. Namely, if there is an interval $[u, v]$ $(1 \leq u \leq v \leq |T|)$ such that $T[u : v] = a^{v-u+1}$, $T[u - 1] \neq a$, and $T[v + 1] \neq a$, then we know that there are at most $\lfloor (v - u + 1)/2 \rfloor$ non-overlapping occurrences of aa in $T[u - 1 : v + 1]$. By summing up this value for all such intervals, we obtain $nOcc(T, aa)$. To find such intervals, we process variables $X_i = X_\ell X_r$ in increasing order of i. There are three cases to consider:

(1) When $suf(X_\ell, 1) = pre(X_r, 1) = a$, $slen(X_\ell) < |X_\ell|$ and $plen(X_r) < |X_r|$. For any interval $[u', v'] \in itv(X_i)$, it holds that $T[u' + |X_\ell| - slen(X_\ell) - 1] \neq a$ and $T[u' + |X_\ell| + |plen(X_r)|] \neq a$. Since there are at most $\lfloor (slen(X_\ell) + plen(X_r))/2 \rfloor \geq 1$ non-overlapping occurrences of aa in $T[u' + |X_\ell| - slen(X_\ell) - 1 : u' + |X_\ell| + |plen(X_r)|]$, we increment the value of non-overlapping occurrenes of aa in T by $vOcc(X_i) \times \lfloor (slen(X_\ell) + plen(X_r))/2 \rfloor$. This is done in line 18.

(2) When $suf(X_\ell, 1) \neq pre(X_r, 1)$ and $1 < slen(X_\ell) < |X_\ell|$. Let $suf(X_\ell, 1) = a$. For any interval $[u', v'] \in itv(X_i)$, it holds that $T[u' + |X_\ell| - slen(X_\ell) - 1] \neq a$ and $T[u' + |X_\ell|] \neq a$. Since there are at most $\lfloor slen(X_\ell)/2 \rfloor \geq 1$ non-overlapping occurrences of aa in $T[u' + |X_\ell| - slen(X_\ell) - 1 : u' + |X_\ell|]$, we increment the value of non-overlapping occurrenes of aa in T by $vOcc(X_i) \times \lfloor slen(X_\ell)/2 \rfloor$. This is done in line 12.

(3) When $suf(X_\ell, 1) \neq pre(X_r, 1)$ and $1 < plen(X_r) < |X_r|$. This is symmetric to Case 2, and we increment the value of non-overlapping occurrences of bb in T by $vOcc(X_i) \times \lfloor plen(X_r)/2 \rfloor$, where $b = pre(X_r, 1)$. This is done in line 15.

For convenience, we assume that T starts and ends with special characters # and \$ that do not occur anywhere else in T, respectively. Then we can cope with the last variable X_n as described above. By Lemma 3, we are guaranteed to obtain the non-overlapping frequencies for all 2-grams.

For all variables X_i, $pre(X_i, 1)$, $suf(X_i, 1)$, $plen(X_i)$, and $slen(X_i)$ can be computed in a total of $O(n)$ time, as descrived above. We can reduce the problem to Problem 1, by constructing a string z and array w, appending the 2-gram to count to z, and two values, its weight and 0, to w. The rest can be solved in $O(n)$ time by Theorem 1. □

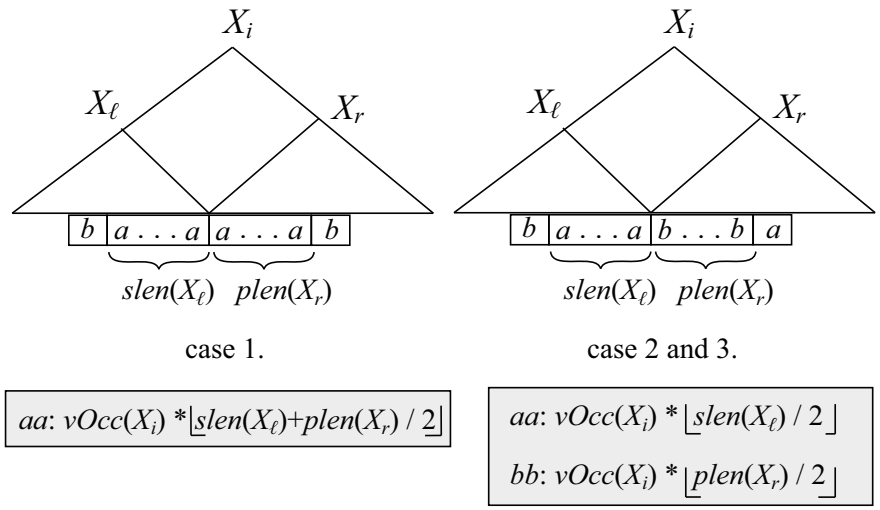

Fig. 2 non-overlapping frequencies correspond to X_i

3.2 q-gram Non-Overlapping Frequecies

Solving Problem 2 for $q \geq 3$ is essentially more difficult than when $q \leq 2$, since q-grams with $q \geq 3$ can have more than 1 period. This implies that computing $plen(X_i)$ and $slen(X_i)$ does not help. To deal with the general case $q \geq 3$, we introduce an extended notion of $plen(X_i)$ and $slen(X_i)$, called *longest overlapping covers*. For any string T and positive integers q and j $(1 \leq j \leq j+q-1 \leq |T|)$, the *longest overlapping cover* of the q-gram $T[j : j+q-1]$ w.r.t. position j of T is an ordered pair $loc_q(T, j) = (sp, ep)$ of positions in T $(1 \leq sp \leq ep \leq |T|)$, which satisfies the following conditions:

(1) if $T[k : k+q-1] = T[j : j+q-1]$ for some k $(sp < k < ep-q+1,\ k \neq j)$, then there exists an integer d $(1 \leq d < q)$ such that $T[k+d : k+d+q-1] = T[j : j+q-1]$ or $T[k-d : k-d+q-1] = T[j : j+q-1]$,

(2) $T[sp-x : sp-x+q-1] \neq T[j : j+q-1]$ for any x $(1 \leq x < q)$, and

(3) $T[ep-q+1+x : ep+x] \neq T[j : j+q-1]$ for any x $(1 \leq x < q)$.

Namely, the longest overlapping cover $loc_q(T, j)$ represents the maximum chain of overlapping occurrences of q-gram $T[j : j+q-1]$ that contains position j.

Lemma 4 Given a string w of length m and integers q, j, the longest cover $loc_q(w, j)$ can be computed in $O(m)$ time.

Proof. We compute $SP = Occ(w[1 : j+q-1], w[j : j+q-1])$ and $EP = Occ(w[j : |T|], w[j : j+q-1])$. Let sp_x (resp. ep_x) denote the x-th smallest element of SP (resp. EP). Let $sp = \min\{sp_x \in SP \mid sp_{y+1} - sp_y < q, 1 \leq \forall y \leq x\}$ if such exists, and $sp = i$ otherwise. Similarly, let $ep = i - 1 + \max\{ep_x \in EP \mid ep_y - ep_{y-1} < q, 2 \leq \forall y \leq x\}$ if such exists, and $ep = i$ otherwise. Then $loc_q(T, i) = (sp, ep)$.

Input: SLP $\mathcal{T} = \{X_i\}_{i=1}^n$ representing string T.
Output: $nOcc(T, P)$ for all 2-grams $P \in \Sigma^2$.

```
1  Compute plen(X_i), slen(X_i), pre(X_i, 1), and suf(X_i, 1) for all 1 ≤ i ≤ n;
2  z ← ε; w ← [];
3  for i ← 1 to n do
4      if X_i = X_ℓ X_r then
5          a ← suf(X_ℓ, 1);
6          b ← pre(X_r, 1);
7          if a ≠ b then
8              z.append(ab) ;
9              w.append(vOcc(X_i)) w.append(0);
10             if 1 < slen(X_ℓ) < |X_ℓ| then
11                 z.append(aa) ;
12                 w.append(vOcc(X_i) × ⌊slen(X_ℓ)/2⌋) w.append(0) ;
13             if 1 < plen(X_r) < |X_r| then
14                 z.append(bb) ;
15                 w.append(vOcc(X_i) × ⌊plen(X_r)/2⌋) w.append(0);
16         else if slen(X_ℓ) < |X_ℓ| and plen(X_r) < |X_r| then /* now a = b */
17             z.append(aa) ;
18             w.append(vOcc(X_i) × ⌊(slen(X_ℓ) + plen(X_r))/2⌋) ;
19             w.append(0);
20 Calculate q-gram frequencies in z, where each q-gram starting at position i is
   weighted by w[i].
```

Fig. 3 Algorithm for computing 2-gram non-overlapping frequencies from SLP

This can be computed in $O(m)$ time using, e.g., the KMP algorithm[10]. □

For any variable X_i and integer q and j $(1 \leq j < q)$,

$$lpc_q(X_i, j) = \begin{cases} (j, |X_i|) & \text{if } j+q-1 \leq |X_i|, \\ (j, ep) & \text{otherwise,} \end{cases}$$

$$lsc_q(X_i, j) = \begin{cases} (1, |X_i|-j+1) & \text{if } |X_i|-j-q+2 \leq 1, \\ (sp', |X_i|-j-q+2) & \text{otherwise,} \end{cases}$$

where $(sp, ep) = (j-1) \oplus loc_q(val(X_i)[j : |X_i|], 1)$ and $(sp', ep') = loc_q(val(X_i)[1 : |X_i|-j-q+2], |X_i|-j+1)$. Namely, $lpc_q(X_i, j)$ is the last part of the longest overlapping cover of the q-gram $X_i[j : j+q-1]$ that starts at position j $(1 \leq j < q)$ in X_i, and $lsc_q(X_i, j)$ is the first part of the longest overlapping cover of the q-gram $X_i[|X_i|-j-q+2 : |X_i|-j+1]$ that ends at position $|X_i|-j+1$ in X_i.

Lemma 5 For all $1 \leq i \leq n$ and $1 \leq j < q$, $lpc_q(X_i, j)$ and $lsc_q(X_i, j)$ can be computed in a total of $O(q^2 n)$ time.

Proof. For any $X_i = X_\ell X_r$ and $1 \leq j < q$, let $s_j = val(X_i)[j : j+q-1]$. Let $(sp_\ell, ep_\ell) = lpc_q(X_\ell, j)$, $(sp_r, ep_r) = lpc_q(X_r, k)$ where $k = \min Occ(pre(X_r, 2q-1), s)$, $(sp_i, ep_i) = loc_q(X_i[|X_\ell|-q+2 : |X_\ell|+q-1], h)$, and for some h $(|X_\ell|-q+2 \leq h \leq |X_\ell|)$ such that $X_i[h : h+q-1] = X_i[s_j : s_j+q-1]$. Then we have

$$lpc_q(X_i, j) = \begin{cases} (sp_\ell, ep_\ell) & \text{if } ep_\ell < sp_i, \\ (sp_\ell, ep_i) & \text{if } sp_i \leq ep_\ell \text{ and } ep_i < sp_r, \\ (sp_\ell, ep_r) & \text{otherwise.} \end{cases}$$

For all variables X_i we pre-compute $pre(X_i, 2q-1)$ and $suf(X_i, 2q-1)$. This can be done in a total of $O(qn)$ time as previously stated in Section 3.1. Then we can compute $lpc_q(X_i, j)$ recursively, using the KMP algorithm and Lemma 4, in a total of $O(q^2 n)$ time for all $1 \leq i \leq n$ and $1 \leq j < q$. $lsc_q(X_i, j)$ can be computed similarly. □

Lemma 6 For all $1 \leq i \leq n$ and $1 \leq j < q$, let $(sp, ep) = loc_q(X_i, |X_\ell|-j+1)$. We can compute $nOcc(val(X_i)[sp : ep], val(X_i)[|X_\ell|-j+1 : |X_\ell|-j+q])$ in a total of $O(q^2 n)$ time, provided that $lsc(X_i, j)$ and $lpc(X_i, j)$ are already computed.

We can extend the algorithm of Fig. 3 so as to compute q-gram non-overlapping frequencies for $q \geq 3$, based on Lemma 5. Instead of computing $slen(X_i)$ and $plen(X_i)$, we compute $lpc_q(X_i, j)$ and $lsc_q(X_i, j)$ for each $1 \leq j < q$, in $O(q^2 n)$ time. By Lemma 6, we compute non-overlapping frequencies of the q-grams in $O(q^2 n)$ time, essentially the same way as computing the 2-grams non-overlapping frequencies. The next theorem holds.

Theorem 3 Problem 2 can be solved in $O(q^2 n)$ time.

4. Applications

Below we shown an example of interesting applications of the proposed algorithms.

4.1 Converting SLP to MFF-SLP

An SLP $\{Y_i\}_{i=1}^m$ which generates a string T is said to be *most-frequent first* (*MFF*) if $nOcc(T, val(Y_i)) \geq nOcc(T, val(Y_j))$ for any $1 \leq j < i \leq m$. Given a string T, an MFF-SLP can be constructed by the Re-Pair algorithm[12], in $O(|T|)$ time. The algorithm recursively replaces the most non-overlapping frequent 2-grams in the string with a new variable, until there exist no 2-grams having more than 1 non-overlapping occurrence.

The problem we consider in this section is: given an SLP $\mathcal{T} = \{X_i\}_{i=1}^n$ that generates a string T, construct an MFF-SLP $\mathcal{S} = \{Y_j\}_{j=1}^m$ that generates T. As previously stated, we can compute MFF-SLP $\mathcal{S}$ by first decompressing the input SLP $\mathcal{T}$, and then running the Re-Pair algorithm on the decompressed string T. However, this takes $O(2^n)$ time and space in the worst case, as $|T| = O(2^n)$.

Fortunately, we can compute MFF-SLP without decompression: We recursively run our algorithm for computing the 2-gram non-overlapping frequencies of Section 3.1, and find the most frequent one amongst them. We then replace non-overlapping occurrences of the 2-gram by a new variable. This is repeated until all 2-grams have at most 1 non-overlapping occurrence. A pseudo-code is shown in Fig. 4. The following theorem holds.

Lemma 7 Given an SLP $\mathcal{T}$ of size n that derives string T, we can compute an MFF-SLP of size m that derives T in $O(mn \times height(\mathcal{T}))$ time, where $height(\mathcal{T})$ is the height of the derivation tree of $\mathcal{T}$.

Input: SLP $\mathcal{T} = \{X_i\}_{i=1}^{n}$ representing string T.
Output: MFF-SLP $\mathcal{S} = \{Y_j\}_{j=1}^{m}$ representing string T.

```
cur ← 1 ;
while |T| > 1 do
    2gram ← MostFrequent2gram(T) ;
    create new variable Y_cur = 2gram ;
    cur ← cur + 1 ;
    forall the variable X_i = X_ℓ X_r s.t. suf(X_ℓ, 1)pre(X_r, 1) = 2gram do
        updateTree(X_i, Y_cur) ;
```

Output: $\mathcal{T}$

Fig. 4 Algorithm for computing MFF-SLP from given SLP

References

1) Amir, A. and Benson, G.: Efficient Two-Dimensional Compressed Matching, *Data Compression Conference*, pp.279–288 (1992).
2) Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A. and abhi shelat: The Smallest Grammar Problem, *IEEE Transactions on Information Theory*, Vol.51, No.7, pp.2554–2576 (2005).
3) Claude, F. and Navarro, G.: Self-indexed Text Compression Using Straight-Line Programs, *MFCS*, pp.235–246 (2009).
4) Goto, K., Bannai, H., Inenaga, S. and Takeda, M.: Fast q-gram Mining on SLP Compressed Strings. submitted.
5) Inenaga, S. and Bannai, H.: Finding Characteristic Substring from Compressed Texts, *International Journal of Foundations of Computer Science* (accepted for publication).
6) Kärkkäinen, J. and Sanders, P.: Simple Linear Work Suffix Array Construction, *Proc. ICALP'03*, Lecture Notes in Computer Science, Vol.2719, Springer-Verlag, pp.943–955 (2003).
7) Karpinski, M., Rytter, W. and Shinohara, A.: An Efficient Pattern-Matching Algorithm for Strings with Short Descriptions, *Nordic Journal of Computing*, Vol.4, pp.172–186 (1997).
8) Kasai, T., Lee, G., Arimura, H., Arikawa, S. and Park, K.: Linear-time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications, *Proc. of CPM'01*, LNCS, Vol.2089, Springer-Verlag, pp.181–192 (2001).
9) Kim, D.K., Sim, J.S., Park, H. and Park, K.: Linear-Time Construction of Suffix Arrays, *Proc. Combinatorial Pattern Matching*, Lecture Notes in Computer Science, Vol.2676, pp.186–199 (2003).
10) Knuth, D.E., Morris, J.H. and Pratt, V.R.: Fast pattern matching in strings, *SIAM J. Comput*, Vol.6, No.2, pp.323–350 (1977).
11) Ko, P. and Aluru, S.: Space Efficient Linear Time Construction of Suffix Arrays, *Proc. Combinatorial Pattern Matching*, Lecture Notes in Computer Science, Vol. 2676, pp.200–210 (2003).
12) Larsson, N.J. and Moffat, A.: Off-Line Dictionary-Based Compression, *Proceedings of the IEEE*, Vol.88, No.11, pp.1722–1732 (2000).
13) Manber, U. and Myers, G.: Suffix arrays: A new method for on-line string searches, *SIAM J.Computing*, Vol.22, No.5, pp.935–948 (1993).
14) Nevill-Manning, C.G., Witten, I.H. and Maulsby, D.L.: Compression by Induction of Hierarchical Grammars, *Data Compression Conference 1994*, IEEE Computer Society, pp.244–253 (1994).
15) Rytter, W.: Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input, *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, Lecture Notes in Computer Science, Vol.3142, Springer Berlin / Heidelberg, pp.15–27 (2004).
16) Ziv, J. and Lempel, A.: A Universal Algorithm for Sequential Data Compression, *IEEE Transactions on Information Theory*, Vol.IT-23, No.3, pp.337–349 (1977).
17) Ziv, J. and Lempel, A.: Compression of Individual Sequences via Variable-length Coding, *IEEE Transactions on Information Theory*, Vol. 24, No. 5, pp. 530–536 (1978).