Abstract

We consider a fully compressed pattern matching problem, where both text $T$ and pattern $P$ are given by its succinct representation, in terms of straight-line programs and its variant. The length of the text $T$ and pattern $P$ may grow exponentially with respect to its description size $n$ and $m$, respectively. The best known algorithm for the problem runs in $O(n^2m^2)$ time using $O(nm)$ space. In this paper, we introduce a variant of straight-line programs, called balanced straight-line programs so that we establish a faster fully compressed pattern matching algorithm. Although the compression ratio of balanced straight-line programs may be worse than the original straight-line programs, they can still express exponentially long strings. Our algorithm runs in $O(nm)$ time using $O(nm)$ space.

1 Introduction

Pattern matching is a task to find all occurrences of a pattern $P$ in a text $T$. It is one of the most fundamental problems in string processing. In the last decade, pattern matching on compressed objects attracts more and more interests. The basic problems are the compressed pattern matching and the fully compressed pattern matching [8]:

Compressed Pattern Matching
Instance: pattern $P$ and $T = \text{Compress}(T)$, representing the compressed text.
Question: does $P$ occur in $T$?

Fully Compressed Pattern Matching
Instance: $P = \text{Compress}(P)$ and $T = \text{Compress}(T)$, representing the compressed pattern and compressed text.
Question: does $P$ occur in $T$?

For recent developments on this topic, refer to an excellent survey paper [8].

Studies on (fully) compressed pattern matching have introduced a new measure to compare the performance of various compression methods. In addition to the traditional measures such as compression ratio, compression time and decompression time, a new measure “(fully) compressed pattern matching time” is also considered to be an important factor. For example, byte pair encoding [3], a simple universal text compression scheme based on the pattern substitution, has been reconsidered as a practically good choice from the new viewpoint of compressed pattern matching, despite the fact that the compression ratio is not as good as other competitors such as Lempel-Ziv type compression [9, 10].

In this paper, we concentrate on strings described in terms of straight-line programs and its variant. A straight-line program is a context-free grammar in the Chomsky normal form that derives only one string. It is a sequence of assignments of either a character or a concatenation of two variables defined before. The length of the string represented by a straight-line program can be exponentially long with respect to the size of the straight-line program.
We note that the class of straight-line programs is the central subclass of collage systems, that was recently introduced by Kida et al. [6] as a unifying framework for various (not fully) compressed pattern matching algorithms. In addition to the concatenation operation, collage systems have expressions of repetition and truncation. Their results imply that the compressed pattern matching for straight-line programs can be solved in $O(n + m^2)$ time using $O(n + m^2)$ space, where $n$ and $m$ are the description sizes of text and pattern, respectively. According to the fully compressed pattern matching, among several studies [4, 5, 7], the best known algorithm for straight-line programs runs in $O(n^2m^2)$ time using $O(nm)$ space. Concerning with the 2D-text described by 2-dimensional straight-line programs, see [1, 8].

In this paper, we introduce a variant of straight-line programs where faster pattern matching is possible. Here, we only allow concatenations of variables which represent strings of the same length except the last assignment. We call them balanced straight-line programs, since the evaluation trees form complete binary trees, except the root node. Because of this restriction, the compression ratio becomes potentially worse than the original straight-line programs, while balanced straight-line programs can still express exponentially long strings. Moreover, compression process in balanced straight-line programs is easier and faster than that in the original straight-line programs.

We show a fully compressed pattern matching algorithm for balanced straight-line programs which runs in $O(nm)$ time using $O(nm)$ space. In this sense, balanced straight-line programs have their own advantage compared with the original straight-line programs as a succinct representation of strings. In Fig. 1, we illustrate the relation of balanced SLP to collage systems and their subclasses. Regular collage systems directly correspond to straight-line programs. Simple collage systems consist of the concatenation of a string with a single character only. Note that LZ78-family compression can be described as simple collage systems. We summarize in Table 1 the running times of related algorithms.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight-line programs</td>
<td>$O(n \text{height}(T) + m^2)$ [6]</td>
<td>unknown</td>
</tr>
<tr>
<td>Balanced straight-line programs</td>
<td>$O(n + m^2)$ [6]</td>
<td>$O(n^2m^2)$ [7]</td>
</tr>
<tr>
<td></td>
<td>$O(n + m^2)$ [6]</td>
<td>$O(nm)$</td>
</tr>
</tbody>
</table>

A straight-line program $\mathcal{R}$ is a sequence of assignments as follows:

$$X_1 = \text{expr}_1; \ X_2 = \text{expr}_2; \ldots; \ X_n = \text{expr}_n,$$

where $X_i$ are variables and $\text{expr}_i$ are expressions of the form:

- $\text{expr}_i$ is a symbol of a given alphabet $\Sigma$, or
- $\text{expr}_i = X_\ell \cdot X_r$, $(\ell, r < i)$, where $\cdot$ denotes the concatenation of $X_\ell$ and $X_r$.

In this paper, both text and pattern are described in terms of straight-line programs and its variant. Denote by $X[d]$ the string obtained by removing the length $d$ suffix from the string represented by variable $X$. Denote by $\mathcal{R}$ the string which is derived from the last variable $X_n$ of the program $\mathcal{R}$. The size of the straight-line program $\mathcal{R}$, denoted by $||\mathcal{R}||$, is the number $n$ of assignments in $\mathcal{R}$. The length of a string $w$ is denoted by $|w|$. We identify a variable $X_i$ with the string represented by $X_i$ if it is clear from the context.

We define the height of a variable $X$ in a straight-line program $\mathcal{R}$ by

$$\text{height}(X) = \begin{cases} 1 & \text{if } X = a \in \Sigma, \\ 1 + \max(\text{height}(X_\ell), \text{height}(X_r)) & \text{if } X = X_\ell \cdot X_r. \end{cases}$$

It corresponds to the length of the longest path that connects $X$ to a leaf in the evaluation tree.

For a string $w$ denote by $w[f : t](1 \leq f \leq t \leq |w|)$ the subword of $w$ starting at $f$ and ending at $t$. We often abbreviate $w[1 : t]$ to $w[t]$, and $w[f : |w|]$ to $w[f :]$. The fully compressed pattern matching for strings in terms of straight-line programs is, given straight-line programs $\mathcal{P}$ and $\mathcal{T}$ which are the descriptions of pattern $P$ and text $T$ respectively, to find all occurrences of $P$ in $T$. Namely, we will find a compact representation of the following set:

$$\text{Occ}(\mathcal{T}, \mathcal{P}) = \{i \mid |T[i : i + |P| - 1] = P\}$$

Hereafter, we use $X_i$ for a variable in $\mathcal{T}$ and $Y_j$ for a variable in $\mathcal{P}$. We assume $||T|| = n$ and $||P|| = m$. For

2 Preliminary

In this section, we introduce basic notations and definitions, and briefly summarize the known results.
a set $U$ of integers and an integer $k$, we denote $U \oplus k = \{ i + k : i \in U \}$ and $U \ominus k = \{ i - k : i \in U \}$.

We now give an overview that is common to our algorithm and that in [7]. First we consider a compact representation of the set $\text{Occ}(X,Y)$. Suppose $X = X_\ell \times X_r$. We define $\text{Occ}^\Delta(X,Y)$ to be the set of occurrences of $Y$ in $X$ such that $Y$ covers the boundary between $X_\ell$ and $X_r$.

$$\text{Occ}^\Delta(X,Y) = \left\{ s + |Y| - |X_\ell| - 1 \mid s \in \text{Occ}(X,Y) \text{ and } |X_\ell| - |Y| + 1 \leq s \leq |X_\ell| + 1 \right\}.$$  

It corresponds to the length from the boundary between $X_\ell$ and $X_r$ to the right edge of $Y$ (see Fig. 2). For the sake of convenience, let $\text{Occ}^\Delta(X,Y) = \text{Occ}(X,Y)$ for $X = a \in \Sigma$.

The following lemmas hold for any straight-line program.

**Lemma 1 (Miyazaki et al. [7])** For any $X$ in $T$ and any $Y$ in $P$, $\text{Occ}^\Delta(X,Y)$ forms a single arithmetic progression.

**Lemma 2 (Miyazaki et al. [7])** For $X_i = X_{l(i)} \times X_{r(i)}$ in $T$ and $Y$ in $P$,

$$\text{Occ}(X_i,Y) = \text{Occ}(X_{l(i)},Y) \cup \{ \text{Occ}^\Delta(X_i,Y) \oplus |X_{l(i)}| \oplus |Y| \} \cup \{ \text{Occ}(X_{r(i)},Y) \oplus |X_{l(i)}| \}.$$  

(See Fig. 3)

The above lemma suggests that $\text{Occ}(X_m,Y)$ can be represented by a combination of $\{ \text{Occ}^\Delta(X_i,Y) \}_{i=1}^n$. By Lemma 1, each $\text{Occ}^\Delta(X_i,Y)$ forms a single arithmetic progression, which can be stored in $O(1)$ space as a triple of the first element, the last element, and the step of the progression. Thus the desired output, a compact representation of the set $\text{Occ}(T,P) = \text{Occ}(X_m,Y_m)$ is some combination of $\{ \text{Occ}^\Delta(X_i,Y_m) \}_{i=1}^n$, which occupies $O(n)$ space. Therefore the computation of the set $\text{Occ}(T,P)$ is reduced to the computation of each set $\text{Occ}^\Delta(X_i,Y_m)$, $i = 1, \ldots, n$.

The following is the best known result for fully compressed pattern matching on straight-line programs.

**Theorem 1 (Miyazaki et al. [7])** Fully compressed pattern matching problem for straight-line programs can be solved in $O(n^2m^2)$ time using $O(nm)$ space.

## 3 Balanced SLP

In this section, we introduce a variant of straight-line programs we call balanced straight-line programs (Balanced SLP) for which we can develop a faster fully compressed pattern matching algorithm.

**Definition 1** A balanced straight-line programs $\mathcal{B}$ is a sequence of assignments as follows:

$$X_1 = \text{expr}_1; \ X_2 = \text{expr}_2; \ldots; \ X_n = \text{expr}_n,$$

where $X_i$ are variables and $\text{expr}_i$ are expressions of the form:
presentation, the membership to the set $\text{Occ}^\Delta(T\cup O, O)$ can be determined. We can see that $B = X_{10} = \text{ababbaaabaa}$. The evaluation tree is shown in Fig. 4. Note that the subtrees rooted in $X_9$ and $X_8$ form complete binary trees. This enables us to develop a faster, fully compressed pattern matching algorithm.

We will prove the following main theorem in the rest of this paper.

**Theorem 2** Given two balanced straight-line programs $T$ and $P$, we can compute an $O(n)$ size representation of the set $\text{Occ}(T, P)$ of all occurrences of the pattern $P$ in the text $T$, in $O(nm)$ time using $O(nm)$ work space. For this representation, the membership to the set $\text{Occ}(T, P)$ can be determined in $O(n)$ time.

### 4 Computation for complete binary trees

In this section, we will describe the method to compute $\text{Occ}(X_i, Y_j)$ for $i < n$ and $j < m$. For $X_i = X_{\ell(i)} \cdot X_{r(i)}$ and $h < \text{height}(X_i)$, we recursively define the rightmost descendant of $X_i$ at height $h$, denoted by $rmd(X_i, h)$, as follows.

$$ rmd(X_i, h) = \begin{cases} rmd(X_{\ell(i)}, h) & \text{if } \text{height}(X_i) > h + 1 \\ X_{\ell(i)} & \text{if } \text{height}(X_i) = h + 1. \end{cases} $$

In the same way, we define the leftmost descendant of $X_i$ at height $h$ by

$$ lmd(X_i, h) = \begin{cases} lmd(X_{r(i)}, h) & \text{if } \text{height}(X_i) > h + 1 \\ X_{r(i)} & \text{if } \text{height}(X_i) = h + 1. \end{cases} $$

For example, $rmd(X_9, 2) = X_5$, $rmd(X_9, 1) = X_1$, and $lmd(X_9, 2) = X_7$ (in Fig 4). We compute $rmd(X_i, h)$ and $lmd(X_i, h)$ for each variable $X_i$ with $h < \text{height}(Y)$ and store them as a table in advance, so that we can look up in $O(1)$ time. The construction of the table can be done in $O(nm)$ time.

The next lemma gives a recursive relation of $\text{Occ}(X_i, Y_j)$. The proof is rather straightforward from the illustration in Fig. 5.

**Lemma 3** Assume that both $T$ and $P$ are balanced straight-line programs. Let $X_{\ell(i)} = rmd(X_{r(i)}, \text{height}(Y_j))$ and $X_{r(i)} = lmd(X_{\ell(i)}, \text{height}(Y_j))$. For $X_i$ in $T$ and $Y_j = Y_{\ell(j)} \cdot Y_{r(j)}$ in $P$,

$$ \text{Occ}^\Delta(X_i, Y_j) = \text{Occ}^\Delta(X_{\ell(i)}, Y_j) \cup \text{Occ}^\Delta(X_i, Y_{r(i)}), $$

where

$$ \text{Occ}^\Delta(X_{\ell(i)}, Y_j) = \text{Occ}^\Delta(X_{\ell(i)}, Y_{\ell(j)}) \cap \text{Occ}^\Delta(X_{r(i)}, Y_{r(j)}), $$

and

$$ \text{Occ}^\Delta(X_i, Y_{r(j)}) = \text{Occ}^\Delta(X_i, Y_{\ell(j)}) \cap \text{Occ}^\Delta(X_{r(i)}, Y_{r(j)}) \oplus |Y_{r(j)}|. $$

Since $\text{Occ}^\Delta(X_i, Y_j)$ forms a single arithmetic progression, union operation can be answered in $O(1)$ time. Thus, the problem to be overcome is to perform the intersection operation efficiently. Lemma 5 is a key to solve this problem. It utilizes the periodicity lemma below.

**Lemma 4 (Periodicity Lemma) (see [2], p. 29)** Let $p$ and $q$ be two periods of a string $w$. If $p + q \equiv \gcd(p, q) \leq |w|$, then the greatest common divisor $\gcd(p, q)$ is also a period of $w$.

Let $\langle a, d, b \rangle$ be the set formed by an arithmetic progression where the first element is $a$, the step is $d$, and the last element is $b$.

**Lemma 5** Assume $\langle a_1, d_1, b_1 \rangle = \text{Occ}^\Delta(X_{\ell(i)}, Y_{\ell(j)})$ and $\langle a_2, d_2, b_2 \rangle = \text{Occ}^\Delta(X_{r(i)}, Y_{r(j)})$ are given. Then we can compute $\langle a, d, b \rangle = \langle a_1, d_1, b_1 \rangle \cap \langle a_2, d_2, b_2 \rangle$ in $O(1)$ time.

**Proof.** Since it is trivial for the case $d_1 = d_2$, we can assume that $d_1 < d_2$ without loss of generality. We will prove the following two claims.
Claim 1. \( (a, d, b) \) contains at most one element.

Claim 2. \( (a, d, b) \subseteq \{a_1, a_2, b_1, b_2\} \).

By Claim 2, we can compute \( (a, d, b) \) by checking whether \( a_1, b_1 \in \{a_2, d_2, b_2\} \) and \( a_2, b_2 \in \{a_1, d_1, b_1\} \). Claim 1 guarantees that at most one checking will succeed. Thus we can compute \( (a, d, b) \) from \( \{a_1, d_1, b_1\} \) and \( \{a_2, d_2, b_2\} \) in \( O(1) \) time. In order to prove these claims, we pay our attention to the strings \( T_1 \) and \( T_2 \) as follows (See Fig. 6), where \( X^{R(i)} = X_L \cdot X_R \).

\[
\begin{align*}
T_1 &= X^{R(i)}[a_1 + 1 : b_1 + |X_L|] \\
T_2 &= X_R[a_2 + 1 : X^{R(i)}[b_2]]
\end{align*}
\]

Let \( T' \) be the common substring of \( T_1 \) and \( T_2 \). Then \( T' = X_R[a_2 + 1 : b_1] \).

Proof of Claim 1. Remark that \( d_1 \) and \( d_2 \) are the smallest periods of \( T_1 \) and \( T_2 \), respectively. Suppose \( (a, d, b) \) contains two elements \( t_1 \) and \( t_2 \) \((t_1 < t_2)\). Since \( t_1 \in \langle a_2, d_2, b_2 \rangle \) and \( t_2 \in \langle a_1, d_1, b_1 \rangle \), we have \( t_1 \geq a_2 \) and \( t_2 \leq b_1 \). These imply that \( |T'| = b_1 - a_2 \geq t_2 - t_1 \). We now show that \( t_2 - t_1 \geq d_1 + d_2 - \gcd(d_1, d_2) \). Since \( t_1, t_2 \in \langle a_1, d_1, b_1 \rangle \cap \langle a_2, d_2, b_2 \rangle \), we have \( t_2 - t_1 = d_1 \cdot c_1 = d_2 \cdot c_2 \) for some \( c_1, c_2 \geq 1 \). We have two cases based on whether \( c_1 \) is divided by \( c_2 \) or not. (Case 1) If \( c_1 \) is divided by \( c_2 \), we have \( \gcd(d_1, d_2) = d_1 \). (Case 2) Otherwise, \( c_2 > 1 \). Then \( t_2 - t_1 = d_2 \cdot c_2 \geq 2d_2 > d_1 + d_2 \).

In both cases, we have \( t_2 - t_1 \geq d_1 + d_2 - \gcd(d_1, d_2) \), which implies \( |T'| \geq d_1 + d_2 - \gcd(d_1, d_2) \). Since both \( d_1 \) and \( d_2 \) are periods of \( T' \), we know that \( \gcd(d_1, d_2) \) is also a period of \( T' \) by Periodicity Lemma. Since \( \gcd(d_1, d_2) \leq d_1 < d_2 \), this contradicts with the fact \( d_2 \) is the smallest period of \( T_2 \). Therefore, \( (a, d, b) \) cannot contain more than one element.

Proof of Claim 2. Suppose there exists an element \( t \in \langle a, d, b \rangle - \{a_1, a_2, b_1, b_2\} \). Let \( k_1 \) and \( k_2 \) be the cardinalities of the sets \( \{a_1, d_1, b_1\} \) and \( \{a_2, d_2, b_2\} \), respectively. Remark that \( b_1 = a_1 + (k_1 - 1) \cdot d_1 \) and \( b_2 = a_2 + (k_2 - 1) \cdot d_2 \). Since \( \langle a, d, b \rangle = \langle a_1, d_1, b_1 \rangle \cap \langle a_2, d_2, b_2 \rangle \), for some integers \( i \) and \( j \) with \( 1 \leq i \leq k_1 - 2 \) and \( 1 \leq j \leq k_2 - 2 \), we have \( t = a_1 + i \cdot d_1 = a_2 + j \cdot d_2 \). We see that

\[
|T'| = b_1 - a_2 = a_1 + (k_1 - 1) \cdot d_1 - a_2 = (t - a_2) + a_1 + (k_1 - 1) \cdot d_1 - t = (a_2 + j \cdot d_2 - a_2 + a_1 + (k_1 - 1) \cdot d_1 - a_1 + i \cdot d_1) = j \cdot d_2 + (k_1 - 1 - i) \cdot d_1 \geq d_1 + d_2.
\]

Since both \( d_1 \) and \( d_2 \) are periods of \( T' \), we know that \( \gcd(d_1, d_2) \) is also a period of \( T' \) by Periodicity Lemma. Since \( \gcd(d_1, d_2) \leq d_1 < d_2 \), we see \( \gcd(d_1, d_2) \) is a smaller period of \( T_2 \), which is a contradiction. Therefore, \( (a, d, b) \subseteq \{a_1, a_2, b_1, b_2\} \).

This completes the proof of Lemma 5.
The above Lemma 5 enables us to perform the intersection operation to compute \( \text{Occ}^\Delta_i(X_i, Y_j) \) and \( \text{Occ}^\Delta_n(X_i, Y_j) \) in \( O(1) \) time. When computing each \( \text{Occ}^\Delta(X_i, Y_j) \) recursively, we may often refer to the same set \( \text{Occ}^\Delta(X_i', Y_j') \) repeatedly for \( i' < i \) and \( j' < j \). We take dynamic programming strategy. Let us consider an \( n \times m \) table \( \text{App} \), where each entry \( \text{App}[i, j] \) at row \( i \) and column \( j \) stores the triple representation of the set \( \text{Occ}^\Delta(X_i, Y_j) \). We compute each \( \text{App}[i, j] \) in bottom-up manner, for \( i = 1, \ldots, n-1 \) and \( j = 1, \ldots, m-1 \). We can construct the whole table \( \text{App} \) in \( O(nm) \). The size of the whole table is \( O(nm) \), since each triple occupies \( O(1) \) space.

### 5 Computation for the last assignment

In this section, we will show how to compute \( \text{Occ}(X_n, Y_m) \) and \( \text{Occ}^\Delta(X_i, Y_j) \) for \( 1 \leq i \leq n \) and \( 1 \leq j \leq m \).

In the sequel, we assume that \( Y_{r(i)} \succ Y_{r(j)} \), since the other case is symmetric. \( \text{Occ}(X_n, Y_m) \) can be computed by next lemma.

**Lemma 6** For \( X_n = X_{r(n)}^{d_r(n)} \cdot X_{r(m)}^{d_r(m)} \) in \( T \) and \( Y_m = Y_{r(i)}^{d_r(i)} \cdot Y_{r(j)}^{d_r(j)} \) in \( P \), the following recursive relation holds.

\[
\text{Occ}(X_n, Y_m) = \begin{cases} 
\text{Occ}(X_{r(n)}^{d_r(n)}, Y_m) \cup \text{Occ}(X_{r(n)}^{d_r(n)} \cdot X_{r(m)}^{d_r(m)}, Y_m) \cup |X_{r(n)}| \oplus d_x \oplus |Y_m|, & \text{if } d_x < |Y_m|, \\
\text{Occ}(X_{r(n)}^{d_r(n)}, Y_m) \cup \text{Occ}(X_{r(n)}^{d_r(n)} \cdot X_{r(m)}^{d_r(m)}, Y_m) \cup |X_{r(n)}| \oplus d_x, & \text{if } d_x > |Y_m|. 
\end{cases}
\]

For \( 1 \leq i \leq n \) and \( 1 \leq j < m \),

\[
\text{Occ}(X_i, Y_j) = \begin{cases} 
\text{Occ}(X_{r(i)}^{d_r(i)}, Y_j) \cup \text{Occ}(X_{r(i)}^{d_r(i)} \cdot Y_j^{d_r(j)}), & \text{if height}(X_i) > \text{height}(Y_j), \\
\text{Occ}^\Delta(X_i, Y_{r(j)}^{d_r(j)}) \cap (\text{Occ}^\Delta(X_i, Y_{r(j)}^{d_r(j)} \cdot Y_{r(j)}^{d_r(j)}), & \text{if height}(X_i) = \text{height}(Y_j). 
\end{cases}
\]

**Lemma 7** For \( X_i = X_{r(i)}^{d_r(i)} \cdot X_{r(i)}^{d_r(i)} \) in \( T \) and \( Y_j = Y_{r(j)}^{d_r(j)} \cdot Y_{r(j)}^{d_r(j)} \) in \( P \), let \( X_{r(i)}^{d_r(i)} \cap Y_{r(j)}^{d_r(j)} \) be the union operation to compute \( \text{Occ}^\Delta(X_{r(i)}^{d_r(i)}, Y_{r(j)}^{d_r(j)}) \) and \( \text{Occ}^\Delta(X_{r(i)}^{d_r(i)} \cdot Y_{r(j)}^{d_r(j)}) \).

(See Fig. 8)

\( \text{Occ}^\Delta(X_i, Y_j) \) can be computed by next lemma.

We would like to thank Professor Apostolico for insightful comments.

### References

[1] P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complex-


### Appendix

In this appendix, we show the detail of the method to compute the set $\text{Occ}^X_{\Delta}(X_i, Y_j)$ in Lemma 7 and estimate its cost.

First, we describe the method of computing the following set.

$$S(X, Y, k) = \left\{ t \mid X[t : t + |Y| - 1] = Y \text{ and } t < k < t + |Y| - 1 \right\}$$

$S(X, Y, k)$ is the set of the occurrences of a pattern $Y$ that covers some position $k$ in the string $X$. It also forms a single arithmetic progression. This set can be computed from the following recursive relation and the operation requires $O(\text{height}(X))$ time.

$$S(X, Y, k) = \begin{cases} S(X_k, Y, k) & \text{if } k + |Y| < |X_k|, \\ S(X_r, Y, k - |X_r|) \oplus |X_r| & \text{if } k - |Y| > |X_r|, \\ \text{Occ}_X^Y \oplus |X_r| \cup \text{Occ}_X^Y (\text{round}(X_r, Y_k + 1), Y) & \text{if } k \leq |X_r| \leq k + |Y|, \\ \text{Occ}_X^Y \oplus |X_r| \cup \text{Occ}_X^Y (\text{round}(X_r, Y_k + 1), Y) & \text{if } k - |Y| \leq |X_r| < k. \\ \end{cases}$$

Remind that $d_y$ is the length of overlaps between $Y_{t(j)}$ and $Y_{r(j)}$. In the equation of $\text{Occ}^X_{\Delta}(X, Y)$ in Lemma 7, we denote $\text{Occ}^X_{\Delta}(X, Y_{t(j)}) \oplus |Y_{t(j)}|$ by triple $\langle a, d, k \rangle$ and denote $\text{Occ}^X_{\Delta}(X, Y_{r(j)}) \oplus |Y_{r(j)}|$ by $C$. The smallest element $e$ in $C$ can be computed from the table $\text{App}$ in $O(\text{height}(X_{t(j)}))$ time, by Lemma 2. We now consider the partitions $A$ and $B$ of $\langle a, d, k \rangle$, defined as follows.

$$A = \{ x \mid x \in \langle a, d, k \rangle \text{ and } x \leq b + |Y_t| - d_y \}$$

$$B = \{ x \mid x \in \langle a, d, k \rangle \text{ and } x > b + |Y_t| - d_y \}$$

Since $d$ is a period of $X[a : b + |Y|]$, we have

$$A \cap C = \left\{ x \mid x \in \langle c, d, b \rangle \text{ and } x \leq b + |Y_{t(j)}| - d_y \right\}$$

Moreover, we have

$$B \cap C = B \cap S(X_i, Y_{r(j)}, b + |Y_{r(j)}|).$$

Since both of $A$ and $B$ form single arithmetic progressions, we can answer the intersection operation in $O(1)$ time by Lemma 5.

The whole operation runs in $O(\text{height}(X_{t(j)}))$ time. Therefore, we can compute the set of $\text{Occ}^X_{\Delta}(X_i, Y_j)$ in $O(m)$ time.