Right-to-left Online Construction of Parameterized Position Heaps

Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda

Department of Informatics, Kyushu University, Japan
{noriki.fujisato, yuto.nakashima, inenaga, bannai, takeda}@inf.kyushu-u.ac.jp

Abstract. Two strings of equal length are said to *parameterized match* if there is a bijection that maps the characters of one string to those of the other string, so that two strings become identical. The parameterized pattern matching problem is, given two strings T and P, to find the occurrences of substrings in T that parameterized match P. Diptarama et al. [Position Heaps for Parameterized Strings, CPM 2017] proposed an indexing data structure called *parameterized position heaps*, and gave a left-to-right online construction algorithm. In this paper, we present a *right-to-left* online construction algorithm for parameterized position heaps. For a text string T of length n over two kinds of alphabets Σ and Π of respective size σ and π, our construction algorithm runs in $O(n \log(\sigma+\pi))$ time with $O(n)$ space. Our right-to-left parameterized position heaps support pattern matching queries in $O(m \log(\sigma+\pi)+m\pi+\mathit{pocc}))$ time, where m is the length of a query pattern P and pocc is the number of occurrences to report. Our construction and pattern matching algorithms are as efficient as Diptarama et al.'s algorithms.

1 Introduction

Text indexing is the task to preprocess the text string so that subsequent pattern matching queries can be answered efficiently. To date, a numerous number of text indexing structure for exact pattern matching have been proposed, ranging from classical data structures such as suffix trees [14], directed acyclic word graphs [2,3], and suffix arrays [10], to more advanced ones such as compressed suffix arrays [8] and FM index [7], just to mention a few.

Ehrenfeucht et al. [6] proposed a text indexing structure called *position heaps*. Ehrenfeucht et al.'s position heap is constructed in a *right-to-left* online manner, where a new node is incrementally inserted to the current position heap for each decreasing position $i = n, \ldots, 1$ in the input string T of length n. In other words, Ehrenfeucht et al.'s position heap is defined over a sequence $\langle \varepsilon, T[n..], \ldots, T[1..] \rangle$ of the suffixes of T in increasing order of their length, where ε is the empty string of length 0. Kucherov [9] proposed another variant of position heaps. Kucherov's position heap is constructed in a *left-to-right* online manner, where a new node is incrementally inserted to the current position heap for each increasing $i = 1, \ldots, n$. In other words, Kucherov's position heap is defined over a sequence $\langle T[1..], \ldots, T[n..], \varepsilon \rangle$ of the suffixes of T in decreasing order of their length. We will call Ehrenfeucht et al.'s position heap as the RL position heap, and Kucherov's position heap as the LR position heap. Both of the RL and LR position heaps for a text string T of length n require $O(n)$ space and can be constructed in $O(n \log \sigma)$ time, where σ is the alphabet size. By augmenting the RL and LR position heaps of T with auxiliary links called maximal reach pointers, pattern matching queries can be answered in $O(m \log \sigma + \mathit{occ})$ time, where m is the length of a query pattern P and occ is the number of occurrences of P in T.

Noriki Fujisato, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda: Right-to-left Online Construction of Parameterized Position Heaps, pp. 91–102.
Proceedings of PSC 2018, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-06484-9

Nakashima et al. [12] proposed position heaps for a set of strings that is given as a reversed trie, and proposed an algorithm that constructs the position heap of a given trie in $O(\sigma N)$ time and space, where N is the size of the input trie. Later, the same authors showed how to construct the position heap of a trie in $O(N)$ time and space, for integer alphabets of size polynomialy bounded in N [13].

Baker [1] introduced the *parameterized pattern matching* problem, that seeks for the occurrences of substrings of the text T that have the "same" structures as the given pattern P. Parameterized pattern matching is motivated by e.g., software maintenance and plagiarism detection [1]. More formally, we consider two distinct alphabets Σ and Π, and we call an element over $\Sigma \cup \Pi$ a p-string. The parameterized pattern matching problem is, given two p-strings T and P, to find all occurrences of substrings X of T that can be transformed to P by a bijection from $\Sigma \cup \Pi$ to $\Sigma \cup \Pi$ which is identity for Σ. For instance, if $T = \texttt{abzaxxbyaxxbazzax}$ and $P = \texttt{yazzbx}$ where $\Sigma = \{\texttt{a}, \texttt{b}\}$ and $\Pi = \{\texttt{x}, \texttt{y}, \texttt{z}\}$, then the positions to output are 3 and 8. To see why, observe that for the substring $T[3..8] = \texttt{zaxxby}$ there is a bijection $\texttt{z} \rightarrow \texttt{y}$, $\texttt{a} \rightarrow \texttt{a}$, $\texttt{x} \rightarrow \texttt{z}$, $\texttt{b} \rightarrow \texttt{b}$, and $\texttt{x} \rightarrow \texttt{y}$ that maps the substring to P. Also, observe that for the other substring $T[8..13] = \texttt{yaxxbz}$, there is a bijection $\texttt{y} \rightarrow \texttt{y}$, $\texttt{a} \rightarrow \texttt{a}$, $\texttt{x} \rightarrow \texttt{z}$, $\texttt{b} \rightarrow \texttt{b}$, and $\texttt{z} \rightarrow \texttt{x}$ that maps the substring to P as well.

Of various algorithms and indexing structures for the parameterized pattern matching (see [11] for a survey), we focus on Diptarama et al.'s *parameterized position heaps* [5]. Diptarama et al.'s *parameterized position heaps* are based on Kucherov's LR position heaps, which are constructed in a *left-to-right* online manner. Let us call their structure the *LR p-position heaps*. Diptarama et al. showed how to construct the LR p-position heap for a given text of length n in $O(n \log(\sigma + \pi))$ time with $O(n)$ space, where $\sigma = |\Sigma|$ and $\pi = |\Pi|$. They also showed that the LR p-position heap augmented with maximal reach pointers can support parameterized pattern matching queries in $O(m \log(\sigma + \pi) + m\pi + pocc)$ time, where *pocc* is the number of occurrences to report.

In this paper, we propose *RL p-position heaps* which are constructed in a *right-to-left* online manner. We show how to construct our RL position heap for a given text string T of length n in $O(n \log(\sigma + \pi))$ time with $O(n)$ space. Our construction algorithm is based on Ehrenfeucht et al.'s construction algorithm for RL position heaps [6], and Weiner's suffix tree construction algorithm [14]. Namely, we use reversed suffix links defined for the nodes of RL p-position heaps. The key to our algorithm is how to label the reversed suffix links, which will be clarified in Definition 5. Using our RL p-position heap augmented with maximal reach pointers, one can perform parameterized pattern matching queries in $O(m \log(\sigma + \pi) + m\pi + pocc)$ time.

2 Preliminaries

2.1 Notations on strings

Let Σ and Π be disjoint sets called a *static alphabet* and a *parameterized alphabet*, respectively. Let $\sigma = |\Sigma|$ and $\pi = |\Pi|$. An element of Σ is called an *s-character*, and that of Π is called a *p-character*. In the sequel, both an s-character and a p-character are sometimes simply called a *character*. An element of Σ^* is called a *string*, and an element of $(\Sigma \cup \Pi)^*$ is called a *p-string*. The length of a (p-)string S is the number of characters contained in S. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. For a (p-)string $S = XYZ$, X, Y and Z are called a *prefix*, *substring*, and *suffix* of w, respectively. The set of prefixes, substrings, and suffixes of a (p-)string S is denoted by $\mathsf{Prefix}(S)$, $\mathsf{Substr}(S)$, and $\mathsf{Suffix}(S)$, respectively. The i-th character of a (p-)string S is denoted by $S[i]$ for $1 \le i \le |S|$, and the substring of a (p-)string S that begins at position i and ends at position j is denoted by $S[i..j]$ for $1 \le i \le j \le |S|$. For convenience, let $S[i..j] = \varepsilon$ if $j < i$. Also, let $S[i..] = S[i..|S|]$ for any $1 \le i \le |S|$.

2.2 Parameterized pattern matching

For any p-string X and $f : (\Sigma \cup \Pi) \to (\Sigma \cup \Pi)$, let $F(X) = f(X[1]) \cdots f(X[|X|])$. Two p-strings X and Y of length k each are said to *parameterized match* (*p-match*) iff there is a bijection f on $\Sigma \cup \Pi$ such that $f(a) = a$ for any $a \in \Sigma$ and $f(X[i]) = Y[i]$ for all $1 \le i \le k$. For instance, if $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and $\Pi = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$, then $X = \mathtt{axbzzayx}$ and $Y = \mathtt{azbyyaxz}$ p-match since there is a bijection f such that $f(\mathtt{a}) = \mathtt{a}$, $f(\mathtt{b}) = \mathtt{b}$, $f(\mathtt{x}) = \mathtt{z}$, $f(\mathtt{y}) = \mathtt{x}$, and $f(\mathtt{z}) = \mathtt{y}$ and $F(X) = F(\mathtt{axbzzayx}) = \mathtt{azbyyaxz} = Y$. We write $X \approx Y$ iff X and Y p-match.

The *previous encoding* $\mathsf{prev}(S)$ of a p-string S of length n is a sequence of length n such that the first occurrence of each p-character x is replaced with 0 and any other occurrence of x is replaced by the distance to the previous occurrence of x in S, and each s-character remains the same. More formally, $\mathsf{prev}(S)$ is a sequence over $\Sigma \cup [0..n-1]$ of length n such that for each $1 \le i \le n$,

$$\mathsf{prev}(S)[i] = \begin{cases} S[i] & \text{if } S[i] \in \Sigma, \\ 0 & \text{if } S[i] \in \Pi \text{ and } S[i] \ne S[j] \text{ for any } 1 \le j < i, \\ i - j & \text{if } S[i] \in \Pi, S[i] = S[j] \text{ and } S[i] \ne S[k] \text{ for any } j < k < i. \end{cases}$$

Observe that $X \approx Y$ iff $\mathsf{prev}(X) = \mathsf{prev}(Y)$. Using the same example as above, we have that $\mathsf{prev}(\mathtt{axbzzayx}) = \mathsf{prev}(\mathtt{azbyyaxz}) = \mathtt{a0b01a06}$.

Let T and P be p-strings of length n and m, respectively, where $n \ge m$. The *parameterized pattern matching* problem is to find all positions i in T such that $T[i..i+m-1] \approx P$.

3 Parameterized position heaps

Let $\mathcal{S} = \langle S_1, \ldots, S_k \rangle$ be a sequence of strings such that for any $1 < i \le k$, $S_i \notin \mathsf{Prefix}(S_j)$ for any $1 \le j < i$. For convenience, we assume that $S_1 = \varepsilon$.

Definition 1 (Sequence hash trees [4]). *The* sequence hash tree *of a sequence $\mathcal{S} = \langle S_1, \ldots, S_k \rangle$ of strings, denoted $\mathsf{SHT}(\mathcal{S})$, is a trie structure that is recursively defined as follows: Let $\mathsf{SHT}(\mathcal{S})^i = (V_i, E_i)$. Then*

$$\mathsf{SHT}(S)^i = \begin{cases} (\{\varepsilon\}, \emptyset) & \text{if } i = 1, \\ (V_{i-1} \cup \{p_i\}, E_{i-1} \cup \{(q_i, c, p_i)\}) & \text{if } 1 \le i \le k, \end{cases}$$

where q_i is the longest prefix of S_i which satisfies $q_i \in V_{i-1}$, $c = S_i[|q_i| + 1]$, and p_i is the shortest prefix of S_i which satisfies $p_i \notin V_{i-1}$.

Note that since we have assumed that each $S_i \in \mathcal{S}$ is not a prefix of S_j for any $1 \le j < i$, the new node p_i and new edge (q_i, c, p_i) always exist for each $1 \le i \le k$. Clearly $\mathsf{SHT}(\mathcal{S})$ contains k nodes (including the root).

prev($T[17..]$)	0
prev($T[16..]$)	00
prev($T[15..]$)	a00
prev($T[14..]$)	0a03
prev($T[13..]$)	00a33
prev($T[12..]$)	000a33
prev($T[11..]$)	0100a33
prev($T[10..]$)	00104a33
prev($T[9..]$)	010104a33
prev($T[8..]$)	0013104a33
prev($T[7..]$)	01013104a33
prev($T[6..]$)	001313104a33
prev($T[5..]$)	0101313104a33
prev($T[4..]$)	00131313104a33
prev($T[3..]$)	002131313104a33
prev($T[2..]$)	0022131313104a33
prev($T[1..]$)	a0022131313104a33

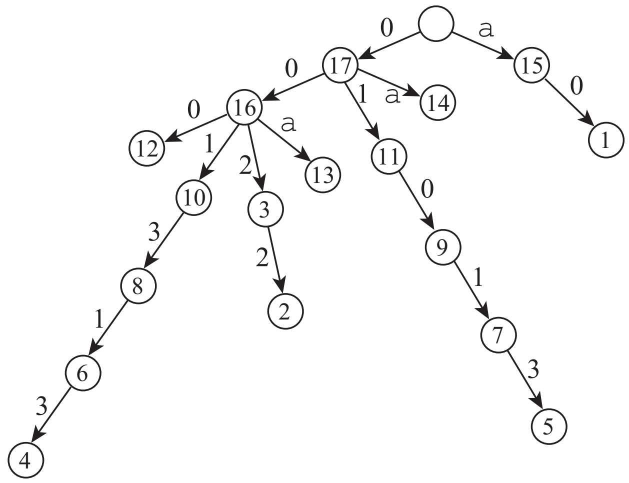

Figure 1. To the left is the list of prev($T[i..]$) for p-string T = axyxyyxxyyxxzyazy of length 17, where $\Sigma = \{\mathtt{a}\}$ and $\Pi = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$. To the right is an illustration for PPH(T). The underlined prefix of each prev($T[i..]$) in the left list denotes the longest prefix of prev($T[i..]$) that was inserted to PPH($T[i+1..]$) and hence, the node with id i represents this underlined prefix of prev($T[i..]$).

In what follows, we will define our indexing data structure for a text p-string T of length n. Let $\mathcal{P}_T = \langle \varepsilon, \mathsf{prev}(T[n..]), \ldots, \mathsf{prev}(T[1..]) \rangle$ be the sequence of previous encoded suffixes of T arranged in *increasing* order of their length. It is clear that $\mathsf{prev}(T[i..]) \notin \mathsf{Prefix}(\mathsf{prev}(T[j..]))$ for any $1 \le j < i$ and $\mathsf{prev}(T[i..]) \notin \mathsf{Prefix}(\varepsilon)$ for any $1 \le i \le n$. Hence we can naturally define the sequence hash tree for $\mathcal{P}_T$, and we obtain our data structure:

Definition 2 (Parameterized positions heaps). *The* parameterized position heap *(*p-position heap*) for a p-string T, denoted* PPH(T)*, is the sequence hash tree of $\mathcal{P}_T$ i.e.,* $\mathsf{PPH}(T) = \mathsf{SHT}(\mathcal{P}_T)$.

See Figure 1 for an example of our p-position heap.

Note that we can obtain $\mathcal{P}_{T[i-1..]}$ by adding $\mathsf{prev}(T[i-1..])$ at the beginning of $\mathcal{P}_{T[i..]}$. This also means that $\mathsf{PPH}(T[i..]) = \mathsf{SHT}(\mathcal{P}_{T[i..]})$ for each $1 \le i \le n$. Hence, we can construct PPH(T) by processing the input string T from right to left. We remark that we can easily compute $\mathsf{prev}(T[i-1..])$ from $\mathsf{prev}(T[i..])$ in a total of $O(n \log \pi)$ time for all $2 \le i \le n$ using $O(\min\{\pi, n\})$ extra space, e.g., by maintaining a balanced search tree that stores the distinct p-characters that have occurred in $T[i..]$ and records the leftmost occurrences of these p-character in the nodes.

Diptarama et al. [5] proposed another version of parameterized position heap for a sequence of previous encoded suffixes of the input p-string T arranged in *decreasing* order of their length. Since their algorithm processes T from left to right, we sometimes call their structure as a *left-to-right p-position heap* (*LR p-position heap*), while we call our PPH(T) as a *right-to-left p-position heap* (*RL p-position heap*) since our construction algorithm processes T from right to left.

For any p-string $P \in (\Sigma \cup [0..n-1])^+$, we say that P is *represented* by PPH(T) iff PPH(T) has a path which starts from the root and spells out P.

Lemma 3. *For any string T of length n,* PPH(T) *consists of exactly $n+1$ nodes. Also, there is a one-to-one correspondence between the positions $1, \ldots, n$ in T and the non-root nodes of* PPH(T).

Proof. Initially, $\mathsf{PPH}(\varepsilon)$ consists only of the root that represents ε. For each $1 \leq i \leq n$, since $|\mathsf{prev}(T[i..])| = n - i + 1 > n - j + 1 = |\mathsf{prev}(T[j..])|$ for any $1 \leq i < j \leq n$, it is clear that there is a prefix of $\mathsf{prev}(T[i..])$ that is not represented by $\mathsf{PPH}(T[i + 1..])$. Therefore, when we construct $\mathsf{PPH}(T[i..])$ from $\mathsf{PPH}(T[i+1..])$, then exactly one node is inserted, which corresponds to position i. □

Let V be the set nodes of $\mathsf{PPH}(T)$. Based on Lemma 3, we define a bijection $\mathsf{id} : V \to [0..n]$ such that $\mathsf{id}(r) = 0$ for the root r and $\mathsf{id}(v) = i$ iff v was the node that was inserted when constructing $\mathsf{PPH}(T[i..])$ from $\mathsf{PPH}(T[i + 1..])$.

Unlike our RL p-position heap, Diptarama et al.'s LR p-position heap can have *double nodes* to which two positions of the text p-string are associated.

We remark that the pattern matching algorithm of Diptarama et al. [5] can be applied to our RL p-position heap $\mathsf{PPH}(T)$ for a text p-string T, and this way one can solve the parameterized pattern matching problem in $O(m \log(\sigma + \pi) + m\pi + occ)$ time, where occ is the number of positions in text T such that the pattern p-string P of length m and the corresponding substring $T[i..i + m - 1]$ p-match. We note that since our RL p-position heap does not have double nodes, the pattern matching algorithm can be somewhat simplified.

The following lemma is an analogue to Lemma 6 of [5] for Diptarama et al.'s LR p-position heap.

Lemma 4. *For any* $1 \leq i \leq j \leq n$ *if* $\mathsf{prev}(T[i..j])$ *is represented by* $\mathsf{PPH}(T)$*, then for any substring* X *of* $T[i..j]$, $\mathsf{prev}(X)$ *is represented by* $\mathsf{PPH}(T)$*.*

Proof. The lemma can be shown in a similar way to Lemma 6 of [5]. For the sake of completeness, we provide a full proof below.

First, we show that for any proper prefix $T[i..i + k]$ of $T[i..j]$ with $0 \leq k < j - i$, $\mathsf{prev}(T[i..i + k])$ is represented by $\mathsf{PPH}(T)$. It follows from the definition of previous encoding that $\mathsf{prev}(T[i..i+k]) = \mathsf{prev}(T[i..j])[1..k+1]$, and hence $\mathsf{prev}(T[i..i+k])$ is a prefix of $\mathsf{prev}(T[i..j])$. Since $\mathsf{prev}(T[i..j])$ is represented by $\mathsf{PPH}(T)$ and $i \leq i+k < j$, $\mathsf{prev}(T[i..i + k])$ is also represented by $\mathsf{PPH}(T)$.

Now it suffices for us to show that for any proper suffix $T[i + h..j]$ of $T[i..j]$ with $0 < h \leq j - i$, $\mathsf{prev}(T[i + h..j])$ is represented by $\mathsf{PPH}(T)$, since then we can inductively apply the above discussion for the prefixes. By the above discussions for the prefixes of $T[i..j]$, there exist positions $i = b_{j-i} < \cdots < b_0 \leq n$ in T such that $\mathsf{prev}(T[i..i + k]) = \mathsf{prev}(T[b_k..b_k + k])$ for $0 \leq k \leq j - i$. By the definition of $\mathsf{PPH}(T)$, the root has an out-going edge labeled by $\mathsf{prev}(T[b_1 + 1..b_1 + 1])$, and this is the base case for our induction. Since $\mathsf{prev}(T[i..i + k]) = \mathsf{prev}(T[b_k..b_k + k])$, we have $\mathsf{prev}(T[i + 1..i + k]) = \mathsf{prev}(T[b_k + 1..b_k + k])$. Now since $\mathsf{prev}(T[b_{k+1} + 1..b_{k+1} + k + 1]) = \mathsf{prev}(T[i + 1..i + k + 1])$ and $\mathsf{prev}(T[b_k + 1..b_k + k]) = \mathsf{prev}(T[i + 1..i + k])$, $\mathsf{prev}(T[b_k + 1..b_k + k])$ is a prefix of $\mathsf{prev}(T[b_{k+1} + 1..b_{k+1} + k + 1])$. This implies that if $\mathsf{prev}(T[b_k + 1..b_k + k])$ is represented by $\mathsf{PPH}(T)$, then $\mathsf{prev}(T[b_{k+1} + 1..b_{k+1} + (k+1)])$ is also represented by $\mathsf{PPH}(T)$. By induction, we have that $\mathsf{prev}(T[b_{j-i} + 1..b_{j-i} + j - i]) = \mathsf{prev}(T[i + 1..j])$ is represented by $\mathsf{PPH}(T)$. Applying the same argument inductively, it is immediate that $\mathsf{prev}(T[i + h...j])$ with $2 \leq h \leq j - i$ are also represented by $\mathsf{PPH}(T)$. □

In the next section, we show how to construct our RL p-position heap $\mathsf{PPH}(T)$ for an input text p-string T of length n in $O(n \log(\sigma + \pi))$ time and $O(n)$ space.

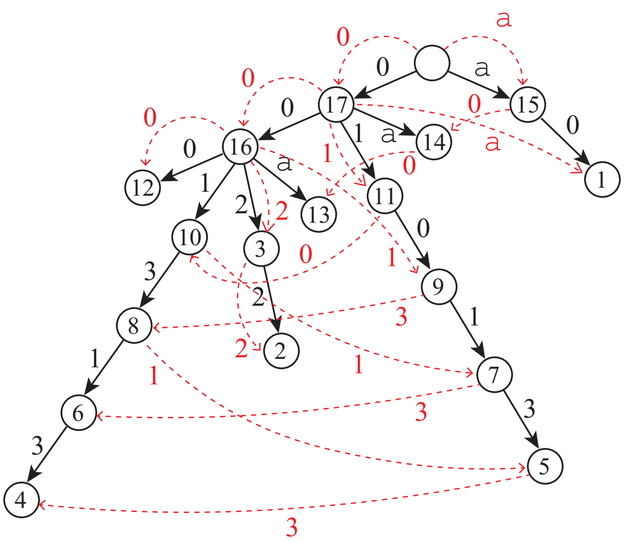

Figure 2. Illustration of the reversed suffix links of $\mathsf{PPH}(T)$ with the same p-string T = `axyxyyxxyyxxzyazy` as in Figure 1. The reversed suffix links and their labels are shown in red.

4 Right to left construction of parameterized position heaps

In this section, we present our algorithm which constructs $\mathsf{PPH}(T)$ of a given p-string T in a right-to-left online manner. The key to our construction algorithm is the use of *reversed suffix links*, which will be defined in the following subsection.

4.1 Reversed suffix links

For convenience, we will sometimes identify each node v of $\mathsf{PPH}(T)$ with the path label from the root to v. In our right-to-left online construction of $\mathsf{PPH}(T)$, we use the *reversed suffix links*, which are a generalization of the *Weiner links* that are used in right-to-left construction of the suffix tree [14] for (standard) string matching:

Definition 5 (Reversed suffix links). *For any node v of $\mathsf{PPH}(T)$ and a character $a \in \Sigma \cup [0..n-1]$, let*

$$\mathsf{rsl}(a, v) = \begin{cases} av & \text{if } a \in \Sigma \cup \{0\} \text{ and } av \text{ is represented by } \mathsf{PPH}(T), \\ u & \text{if } a \in [1..n-1],\ v[a] = 0 \text{ and} \\ & u = 0v[1..a-1]av[a+1..|v|] \text{ is represented by } \mathsf{PPH}(T), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is clear that by taking one rsl link from a node, then the node depth (and hence the string length) increases exactly one.

Observe that the first case of of the definition of $\mathsf{rsl}(a, v)$ is a direct extension of the Weiner links, where $\mathsf{rsl}(a, v)$ points to the node av that is obtained by prepending a to v. The second case, however, is a special case that arises in parameterized pattern matching. The following lemma ensures that our reversed suffix links rsl are well defined:

Lemma 6. *For any node v in $\mathsf{PPH}(T)$ and a character $a \in \Sigma \cup [0..n-1]$, let $\mathsf{rsl}(a, v) = u$, where u is a node of $\mathsf{PPH}(T)$. Then, for any string X such that $\mathsf{prev}(X) = u$, $\mathsf{prev}(X[2..|X|]) = v$.*

Proof. In the first case of the definition of $\mathsf{rsl}(a, v)$ where $a \in \Sigma \cup \{0\}$, we have $\mathsf{prev}(X) = u = av$. Hence, $\mathsf{prev}(X[2..|X|]) = \mathsf{prev}(X)[2..|X|] = u[2..|u|] = v$.

In the second case of the definition of $\mathsf{rsl}(a, v)$ where $a \in [1..n-1]$, we have $\mathsf{prev}(X) = u = 0v[1..a-1]av[a+1..|v|]$, which implies that $X[1] = X[a+1]$ and $X[1] \neq X[i]$ for any $2 \leq i \leq a$. Thus, $\mathsf{prev}(X[2..|X|]) = v[1..a-1]0v[a+1..|v|] = v$. □

The next proposition shows that there is a monotonicity in the labels of the reversed suffix links that come from the nodes in the same path of $\mathsf{PPH}(T)$.

Proposition 7. *Suppose there is a reversed suffix link* $\mathsf{rsl}(a, v)$ *of a node* v *with* $a \in \Sigma \cup [0..n-1]$. *Let* u *be any ancestor of* v. *Then, if* $a \in \Sigma \cup \{0\}$, u *has a reversed suffix link* $\mathsf{rsl}(a, u)$. *Also, if* $a \in [1..n-1]$ *and* $|u| \geq a$, *then* u *has a reversed suffix link* $\mathsf{rsl}(a, u)$, *and if* $a \in [1..n-1]$ *and* $|u| < a$, *then* u *has a reversed suffix link* $\mathsf{rsl}(0, u)$.

Proof. It suffices for us to show that the lemma holds for the parent v' of v, since then the lemma inductively holds for any ancestor of v. Note that $v' = v[1..|v|-1]$. Let $w = \mathsf{rsl}(a, v)$.

If $a \in \Sigma \cup \{0\}$, then $w = av$. Hence, the parent of w is $w[1..|w|-1] = av[1..|v|-1] = av'$. Therefore, there is a reversed suffix link $\mathsf{rsl}(a, v')$.

If $a \in [1..n-1]$ and $|v'| = |v|-1 \geq a$, then it follows from the definition of $\mathsf{rsl}(a, v)$ that $v[a] = 0$ and $w = 0v[1..a-1]av[a+1..|v|]$. Since $|v'| \geq a$, we have that $v'[a] = 0$ and $|v| \geq a+1$. Thus $w[1..|w|-1] = 0v[1..a-1]av[a+1..|v|-1]$ is represented by $\mathsf{PPH}(T)$. Consequently, there is a reversed suffix link $\mathsf{rsl}(a, v')$.

If $a \in [1..n-1]$ and $|v'| = |v|-1 = a-1$, then it follows from the definition of $\mathsf{rsl}(a, v)$ that $v[a] = v[|v|] = 0$ and $w = 0v[1..|v|-1]a$. Thus $w[1..|w|-1] = 0v[1..|v|-1] = 0v'$ is represented by $\mathsf{PPH}(T)$. Consequently, there is a reversed suffix link $\mathsf{rsl}(a, v')$. □

4.2 Adding a new node

Our algorithm processes a given p-string T of length n from right to left and maintains $\mathsf{PPH}(T[i..])$ in decreasing order of $i = n, \ldots, 1$. Initially, we begin with $\mathsf{PPH}(\varepsilon)$ which consists of the root r representing the empty string ε. For convenience, we use an auxiliary node $\perp$ as a parent of the root r, and create reversed suffix links $\mathsf{rsl}(a, \perp) = r$ for every $a \in \Sigma \cup \{0\}$.

Now suppose we have constructed $\mathsf{PPH}(T[i..])$ for $1 < i \leq n$, and we will update it to $\mathsf{PPH}(T[i-1..])$. In so doing, we begin with node v_i such that $\mathsf{id}(v_i) = i$. We know the locus of this node v_i since v_i is the node that was inserted at the last step when $\mathsf{PPH}(T[i..])$ was constructed from $\mathsf{PPH}(T[i+1..])$. Note also that this node v_i is a leaf in $\mathsf{PPH}(T[i..])$. We climb up the path from v_i until finding its lowest ancestor v_i' that satisfies the following. There are three cases:

1. If $T[i-1] \in \Sigma$, then v_i' is the lowest ancestor of v_i such that $\mathsf{rsl}(T[i-1], v_i)$ is defined.
2. If $T[i-1] \in \Pi$ and $T[i-1] \neq T[j]$ for any $i \leq j \leq n$, then v_i' is the lowest ancestor of v_i such that $\mathsf{rsl}(0, v_i)$ is defined.
3. Otherwise, let $d = j - i$ where j is the smallest position such that $i \leq j \leq n$ and $T[i-1] = T[j]$. Then v_i' is the lowest ancestor of v_i such that $\mathsf{rsl}(d, v_i')$ is defined if it exists, and v_i' is the lowest ancestor of v_i such that $\mathsf{rsl}(0, v_i')$ is defined otherwise.

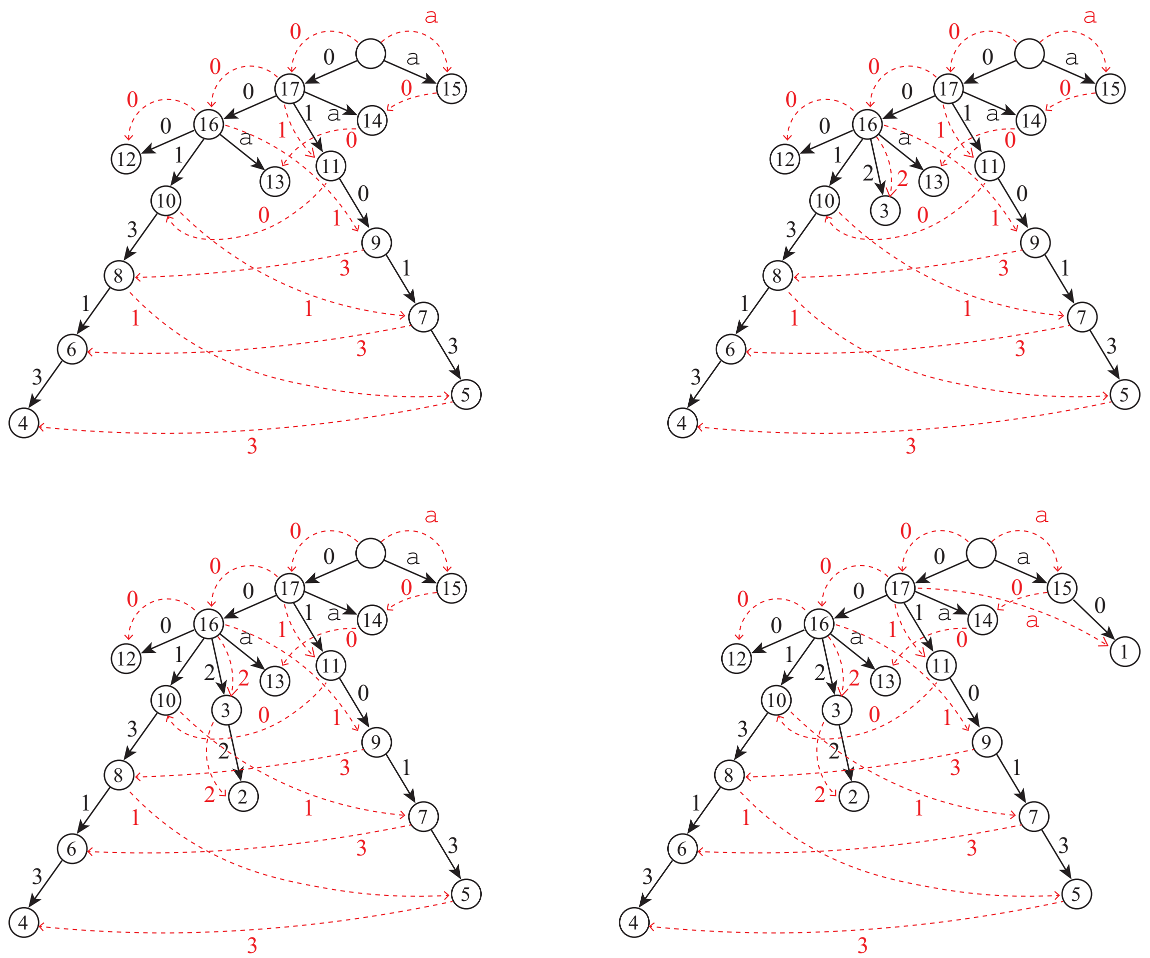

Figure 3. A snapshot of updating $\mathsf{PPH}(T[i..])$ for $i = 4, 3, 2, 1$ with the same p-string T = $\mathtt{axyxyyxxyyxxzyazy}$ as in Figures 1 and 2. First, we update $\mathsf{PPH}(T[4..])$ (upper left) to $\mathsf{PPH}(T[3..])$ (upper right). Since $T[3] = T[5] = \mathtt{y}$ and $d = 5 - 3 = 2$, we first try to find the lowest ancestor of the node with id 4 that has a reversed suffix link labeled with $d = 2$ by climbing up the path. However, it does not exist, and then we arrive at the lowest ancestor with id 17 whose depth is 1 (< 2). Hence the second sub-case of Case 3 is applied, and using its reversed suffix link we move to the node with id 16. The new node with id 3 is inserted as its child. Next, we update $\mathsf{PPH}(T[3..])$ (upper right) to $\mathsf{PPH}(T[2..])$ (lower left). Since $T[2] = T[4] = \mathtt{x}$ and $d = 4 - 2 = 2$, we first try to find the lowest ancestor of the node with id 3 that has a reversed suffix link labeled with $d = 2$ by climbing up the path, and we arrive at the node with id 16. Hence the first sub-case of Case 3 is applied, and using its reversed suffix link we move to the node with id 3. The new node with id 2 is inserted as its child. Finally, we update $\mathsf{PPH}(T[2..])$ (lower left) to $\mathsf{PPH}(T[1..])$ (lower right). Since $T[1] = \mathtt{a} \in \Sigma$, Case 1 is applied. Thus we try to find the lowest ancestor of the node with id 2 that has a reversed suffix link labeled with $\mathtt{a}$ by climbing up the path, and we arrive at the root. Using its reversed suffix link, we move to the node with id 15. The new node with id 1 is inserted as its child.

Let u_i be the node of $\mathsf{PPH}(T[i..])$ that is pointed by the reversed suffix link of v'_i as above. Then, we create a new node v_{i-1} as a child of u_i such that $\mathsf{id}(v_{i-1}) = i - 1$. The new edge (u_i, v_{i-1}) is labeled by $\mathsf{prev}(T[i-1..])[|u_i| + 1]$. We repeat the above procedure for all positions i in T in decreasing order. See also Figure 3 for concrete examples.

Lemma 8. *The above algorithm correctly updates* $\mathsf{PPH}(T[i..])$ *to* $\mathsf{PPH}(T[i-1..])$.

Proof. Note that v_i and v'_i are prefixes of $\mathsf{prev}(T[i..])$. Let a be the character in $\Sigma \cup [0..n-1]$ that is used in the reversed suffix link as above.

In Cases 1 and 2 above, we have $a = T[i-1] \in \Sigma$ or $a = 0$. Then it is clear that av'_i is a prefix of $\mathsf{prev}(T[i-1..])$. Since v'_i is the lowest ancestor of v_i for which $\mathsf{rsl}(a, v'_i)$ is defined, $u_i = av'_i$ is the longest prefix of $\mathsf{prev}(T[i-1..])$ that is represented by $\mathsf{PPH}(T[i..])$. Hence, the new node v_{i-1} and its incoming edge labeled by $\mathsf{prev}(T[i-1..])[|u_i|+1]$ are correctly inserted.

Consider Case 3 above. We first try to find v'_i in the first sub-case, where $a = d \geq 1$. If it exists, then v'_i is the lowest ancestor of v_i such that $\mathsf{rsl}(d, v'_i)$ is defined, and thus $\mathsf{rsl}(d, v'_i) = 0v'_i[1..d-1]dv'_i[d+1..|v'_i|]$. It now follows from Lemma 4 that $u_i = 0v'_i[1..d-1]dv'_i[d+1..|v'_i|]$ is the longest prefix of $\mathsf{prev}(T[i-1..])$ that is represented by $\mathsf{PPH}(T[i..])$. Hence, the new node v_{i-1} and its incoming edge labeled by $\mathsf{prev}(T[i-1..])[|u_i|+1]$ are correctly inserted in this sub-case. It is clear that v'_i in the first sub-case is at least of depth d. Hence, if we arrive at the ancestor of v_i of depth $d-1$ without encountering the lowest ancestor satisfying the condition of the first sub-case, then we try to find the lowest ancestor of v_i that has a reversed suffix link labeled by 0 (second sub-case). Thus, by a similar argument to Case 2, the new node v_{i-1} its incoming edge labeled by $\mathsf{prev}(T[i-1..])[|u_i|+1]$ are correctly inserted in this second sub-case. □

4.3 Adding a new reversed suffix link

After inserting the new node v_{i-1}, we need to maintain the reversed suffix links corresponding to v_{i-1}.

Lemma 9. *There is exactly one reversed suffix link that points to the new node v_{i-1} in $\mathsf{PPH}(T[i-1..])$. Moreover, this reversed suffix link comes from the ancestor of v_i of depth $|v'_i|+1$.*

Proof. Suppose on the contrary that there are two distinct nodes x and y each of which has a reversed suffix link pointing to v_{i-1}. The label of any reversed suffix link that points to v_{i-1} is uniquely determined by the path label from the root to v_{i-1}. Therefore, the reversed suffix links of x and y that point to v_{i-1} are both labeled by the same symbol. This means that $x = y$, however, this contradicts the definition of the p-position heap. Hence, there is at most one node which has a reversed suffix link that points to v_{i-1}.

Let z_i be the ancestor of v_i of depth $|v'_i|+1$. Also, let $x = (T[i..])[|u_i|] = (T[i-1..])[|u_i|+1] = T[i+|u_i|-1]$, namely, x is the text character that corresponds to the label of the edge (v'_i, z_i) that is on the path from the root to v_i, and to the label of the new edge (u_i, v_{i-1}). If $x \in \Pi$ and $i+|u_i|-1$ is the smallest position in $T[i-1..]$ such that $T[i-1] = T[i+|u_i|-1]$, then (v'_i, z_i) is labeled with 0 while (u_i, v_{i-1}) is labeled with $|u_i|$. Otherwise, the label of the new edge (u_i, v_{i-1}) must be equal to that of (v'_i, z_i). It follows from the definition of reversed suffix links that in both cases the reversed suffix link to v_{i-1} comes from z_i. □

Lemma 10. *There is no reversed suffix link that comes from the new node v_{i-1} in $\mathsf{PPH}(T[i-1..])$.*

Proof. Suppose on the contrary that there is a reversed suffix link from v_{i-1} in $\mathsf{PPH}(T[i-1..])$, and let w be the node that is pointed by this reversed suffix link.

Notice that $|w| = |v_{i-1}| + 1$. Let $T[j..]$ be the suffix of T for which this node w was inserted, namely, $\mathsf{id}(w) = j > i-1$. By Lemma 4, for any substring X of $T[j..j+|w|-1]$, $\mathsf{prev}(X)$ is represented by $\mathsf{PPH}(T[j..])$, and hence it is also represented by $\mathsf{PPH}(T[i..])$ since $j \leq i$. Recall that $\mathsf{prev}(T[j+1..j+|w|-1]) = \mathsf{prev}(T[i-1..i+|v_{i-1}|])$, which implies that the node v_{i-1} existed already in $\mathsf{PPH}(T[i..])$. However, this contradicts that v_{i-1} is the node that was inserted when $\mathsf{PPH}(T[i..])$ was updated to $\mathsf{PPH}(T[i-1..])$. □

Due to Lemmas 9 and 10, there is only one reversed suffix link that is newly inserted in $\mathsf{PPH}(T[i-1..])$.

4.4 Complexity analysis

Lemma 11. *The proposed algorithm runs in a total of $O(n \log(\sigma + \pi))$ time with $O(n)$ space.*

Proof. For each $i = n, \ldots, 1$, the algorithm updates $\mathsf{PPH}(T[i..])$ to $\mathsf{PPH}(T[i-1..])$. The update begins with node v_i such that $\mathsf{id}(v_i) = i$, and climbs up the path to v'_i. It takes a reversed suffix link from v'_i and moves to u_i of depth $|v'_i| + 1$, and the new node v_{i-1} of depth $|v'_i| + 2$ with $\mathsf{id}(v_{i-1}) = i - 1$ is inserted. Hence the total number of nodes visited when updating $\mathsf{PPH}(T[i..])$ to $\mathsf{PPH}(T[i-1..])$ is $|v_i| - |v'_i| + 2 = |v_i| - |v_{i-1}| + 4$. Thus, the total number of nodes visited for all $i = n, \ldots, 1$ sums up to $\sum_{i=n}^{2}(|v_i| - |v_{i-1}| + 4) = |v_n| - |v_1| + 4(n-1) = O(n)$. At each node that we visit, it takes $O(\log(\sigma + \pi))$ time to search for the corresponding reversed suffix link, as well as inserting a new edge. Hence, the total time cost is $O(n \log(\sigma + \pi))$.

It is clear that the number of nodes in $\mathsf{PPH}(T)$ is $n+2$, including the root and the auxiliary node $\perp$. It follows from Lemmas 9 and 10 that the number of reversed suffix links coming out from the root, the internal nodes, and the leaves is $n+1$. As for the reversed suffix links that come from $\perp$ to the root, we add a new reversed suffix link labeled with $T[i-1]$ only if $T[i-1] \in \Sigma$ and $T[i-1] \neq T[j]$ for any $j < i-1$. This way, we can maintain these reversed suffix links from $\perp$ in an online manner, using $O(n)$ space. □

We have proven the following theorem, which is the main result of this paper.

Theorem 12. *For an input p-string T of length n, the proposed algorithm constructs $\mathsf{PPH}(T[i..])$ in a right-to-left online manner for $i = n, \ldots, 1$, in a total of $O(n \log(\sigma + \pi))$ time with $O(n)$ space.*

5 Parameterized pattern matching with augmented PPH(T)

Ehrenfeucht et al. [6] introduced *maximal reach pointers*, which used for efficient pattern matching queries on position heaps. Diptarama et al. [5] introduced maximal reach pointers for their LR p-position heaps, and showed how to perform pattern matching queries in $O(m \log(\sigma + \pi) + m\pi + pocc)$ time, where m is the length of a given pattern p-string and *pocc* is the number of occurrences to report. We can naturally extend the notion of maximal reach pointers to our RL p-position heaps, as follows:

Definition 13 (Maximal reach pointers). *For each position $1 \leq i \leq n$ in T, the* maximal reach pointer *of the node v with $\mathsf{id}(v) = i$ points to the deepest node u of $\mathsf{PPH}(T)$ such that u is a prefix of $\mathsf{prev}(T[i..])$.*

$\mathsf{prev}(T[12..])$	a
$\mathsf{prev}(T[11..])$	0a
$\mathsf{prev}(T[10..])$	00a
$\mathsf{prev}(T[9..])$	a00a
$\mathsf{prev}(T[8..])$	0a03a
$\mathsf{prev}(T[7..])$	00a33a
$\mathsf{prev}(T[6..])$	a00a33a
$\mathsf{prev}(T[5..])$	0a03a33a
$\mathsf{prev}(T[4..])$	00a33a33a
$\mathsf{prev}(T[3..])$	a00a33a33a
$\mathsf{prev}(T[2..])$	0a03a33a33a
$\mathsf{prev}(T[1..])$	01a03a33a33a

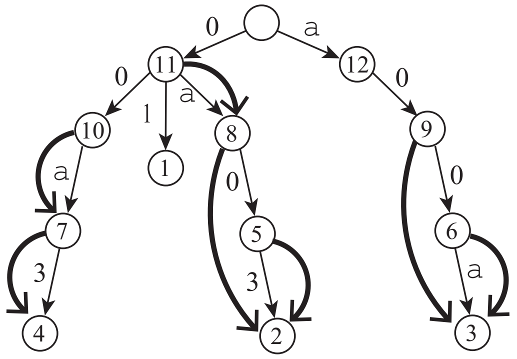

Figure 4. To the left is the list of $\mathsf{prev}(T[i..])$ for p-string $T = \mathtt{xxayxayxayxa}$ of length 12, where $\Sigma = \{\mathtt{a}\}$ and $\Pi = \{\mathtt{x}, \mathtt{y}\}$. To the right is an illustration for augmented $\mathsf{PPH}(T)$, where the maximal reach pointers are indicated by the bold arrows. The wavy underlined prefix of each $\mathsf{prev}(T[i..])$ in the left list denotes the longest prefix of $\mathsf{prev}(T[i..])$ that is represented by $\mathsf{PPH}(T)$, and hence it is the destination of $\mathsf{mrp}(i)$.

We denote by $\mathsf{mrp}(i)$ the pointer of node v such that $\mathsf{id}(v) = i$. The *augmented* $\mathsf{PPH}(T)$ is $\mathsf{PPH}(T)$ with the maximal reach pointers of all nodes. For simplicity, if $\mathsf{mrp}(i)$ points to the node with id i, then we omit this pointer. See Figure 4 for an example of maximal reach pointers and augmented $\mathsf{PPH}(T)$.

Lemma 14. *For every* $1 \le i \le n$, *we can compute* $\mathsf{mrp}(i)$ *in a total of* $O(n \log(\sigma + \pi))$ *time with* $O(n)$ *space.*

Proof. We compute $\mathsf{mrp}(i)$ for each position $i = 1, \ldots, n$ increasing order. In so doing, we use the *forward* suffix link that are the reversals of the reversed suffix links. For simplicity, we will call forward suffix links as suffix links. Since there is exactly one in-coming reversed suffix link to each node, there is also exactly one out-going suffix link from each node. Let $\mathsf{sl}(v)$ denote the node that the suffix link of v points to.

We begin with node v_1 such that $\mathsf{id}(v_1) = 1$. Since we have built $\mathsf{PPH}(T[i..])$ in decreasing order of i, v_1 is a leaf of $\mathsf{PPH}(T)$ and it is the deepest node that is a prefix of $\mathsf{prev}(T[1..])$. Now we take the suffix link of v_1, and let $u_1 = \mathsf{sl}(v_1)$. Since $\mathsf{prev}(T[1..|v_1|]) = v_1$, it follows from Lemma 6 that $u_1 = \mathsf{prev}(T[2..|v_1|])$, which implies that u_1 is a prefix of $\mathsf{prev}(T[2..])$. Then the deepest node v_2 that is a prefix of $\mathsf{prev}(T[2..])$ can be found by traversing the corresponding path from node u_1. Then, we make a pointer to v_2 from the node w with $\mathsf{id}(w) = 2$. We iteratively perform the same procedure for all positions i in increasing order.

To analyze the time complexity, we can use a similar argument as in Lemma 11. For each i, the number of nodes traversed is $|v_{i+1}| - |u_i| + 1 = |v_{i+1}| - |v_i| + 2$. Thus, the total number of nodes visited sums up to $\sum_{i=1}^{n-1}(|v_{i+1}| - |v_i| + 2) = |v_n| - |v_1| + 2(n-1) = O(n)$. Since it takes $O(\log(\sigma + \pi))$ time to search for each corresponding edge in the traversal, the total running time is $O(n \log(\sigma + \pi))$.

The space requirement is clearly $O(n)$. □

It is straightforward that by applying Diptarama et al.'s pattern matching algorithm to our $\mathsf{PPH}(T)$ augmented with maximal reach pointers, parameterized pattern matching can be done in $O(m \log(\sigma + \pi) + m\pi + \mathit{pocc})$ time.

Corollary 15. *Using our augmented* $\mathsf{PPH}(T)$*, one can perform parameterized pattern matching queries in* $O(m\log(\sigma+\pi)+m\pi+pocc)$ *time.*

6 Conclusions and further work

This paper proposed a new indexing structure for parameterized pattern matching, called RL p-position heaps, that are built in a right-to-left online manner. We proposed a Weiner-type construction algorithm for our RL p-position heaps that runs in $O(n\log(\sigma+\pi))$ time with $O(n)$ space, for a given text p-string of length n over a static alphabet Σ of size σ and a parameterized alphabet Π of size π. The key to our efficient construction is how to label the reversed suffix links. By augmenting our position heap with maximal reach pointers, one can perform parameterized pattern matching in $O(m\log(\sigma+\pi)+m\pi+pocc)$ time, where m is the length of a query pattern and $pocc$ is the number of occurrence to report.

Our future work includes the following:

- Would it be possible to shave the $m\pi$ term in the pattern matching time using parameterized position heaps? Other data structures such as parameterized suffix trees achieve better $O(m\log(\sigma+\pi)+pocc)$ time [1].
- Nakashima et al. [12] extended Ehrenfeucht et al.'s right-to-left position heaps [6] to a set of texts given as a trie. We are now working on extending our right-to-left p-position heaps to a set of texts given as a trie.

References

1. B. S. Baker: *Parameterized pattern matching: Algorithms and applications.* J. Comput. Syst. Sci., 52(1) 1996, pp. 28–42.
2. A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. Seiferas: *The smallest automaton recognizing the subwords of a text.* Theoretical Computer Science, 40 1985, pp. 31–55.
3. A. Blumer, J. Blumer, D. Haussler, R. Mcconnell, and A. Ehrenfeucht: *Complete inverted files for efficient text retrieval and analysis.* Journal of the ACM, 34(3) 1987, pp. 578–595.
4. E. Coffman and J. Eve: *File structures using hashing functions.* Communications of the ACM, 13 1970, pp. 427–432.
5. Diptarama, T. Katsura, Y. Otomo, K. Narisawa, and A. Shinohara: *Position heaps for parameterized strings*, in Proc. CPM 2017, 2017, pp. 8:1–8:13.
6. A. Ehrenfeucht, R. M. McConnell, N. Osheim, and S.-W. Woo: *Position heaps: A simple and dynamic text indexing data structure.* Journal of Discrete Algorithms, 9(1) 2011, pp. 100–121.
7. P. Ferragina and G. Manzini: *Indexing compressed text.* J. ACM, 52(4) 2005, pp. 552–581.
8. R. Grossi and J. S. Vitter: *Compressed suffix arrays and suffix trees with applications to text indexing and string matching.* SIAM J. Comput., 35(2) 2005, pp. 378–407.
9. G. Kucherov: *On-line construction of position heaps.* J. Discrete Algorithms, 20 2013, pp. 3–11.
10. U. Manber and G. Myers: *Suffix arrays: A new method for on-line string searches.* SIAM J. Computing, 22(5) 1993, pp. 935–948.
11. J. Mendivelso and Y. Pinzón: *Parameterized matching: Solutions and extensions*, in Proc. PSC 2015, 2015, pp. 118–131.
12. Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda: *The position heap of a trie*, in Proc. SPIRE 2012, vol. 7608 of Lecture Notes in Computer Science, 2012, pp. 360–371.
13. Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda: *Constructing LZ78 tries and position heaps in linear time for large alphabets.* Inf. Process. Lett., 115(9) 2015, pp. 655–659.
14. P. Weiner: *Linear pattern-matching algorithms*, in Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory, 1973, pp. 1–11.