Online Multiple Palindrome Pattern Matching*,**

Hwee Kim and Yo-Sub Han

Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
{kimhwee,emmous}@cs.yonsei.ac.kr

Abstract. A palindrome is a string that reads the same forward and backward. We say that two strings of the same length are pal-equivalent if for each possible center they have the same length of the maximal palindrome. Given a text T of length n and a set of patterns $P_1, \ldots, P_k$, we study the online multiple palindrome pattern matching problem that finds all pairs of an index i and a pattern P_j such that $T[i-|P_j|+1 : i]$ and P_j are pal-equivalent. We solve the problem in $O(m_k M)$ preprocessing time and $O(m_k n)$ query time using $O(m_k M)$ space, where M is the sum of all pattern lengths and m_k is the longest pattern length.

1 Introduction

A palindrome is a string that reads the same forward and backward. If a substring of a string is a palindrome, we say that the string has a palindromic substring or palindromic structure. It is crucial to find palindromes and identify similar palindromic structures in bio sequence analysis [8]. Many researchers examined the properties of palindromic structures in strings [2–6] and proposed efficient algorithms on palindromic structures [7, 10, 12]. We focus on the palindrome pattern matching problem introduced by I et al. [11]—they define two strings of the same length to be pal-equivalent if for each possible center they have the same length of the maximal palindrome. Given a text T of length n and a pattern P of length m, the palindrome pattern matching problem is to find all indices i such that $T[i-m+1 : i]$ and P are pal-equivalent. I et al. [11] presented two algorithms that solve the palindrome pattern matching for an arbitrary size alphabet: One solves the problem in $O(n + m)$ time and the other solves the problem in $O((n + m) \log \sigma + r)$ time, where σ is the alphabet size and r is the number of matching occurrences.

We notice that both algorithms by I et al. [11] require a preprocessing step of T, which makes algorithms unsuitable for an extremely large text or a stream text. This motivates us to consider the online pattern matching, where we should report the matching for each index i while reading T online. We tackle the

* This research was supported by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562).

** Kim was supported by NRF (National Research Foundation of Korea) Grant funded by the Korean Government (NRF-2013-Global Ph.D. Fellowship Program).

E. Moura and M. Crochemore (Eds.): SPIRE 2014, LNCS 8799, pp. 173–178, 2014.

online multiple palindrome pattern matching based on a modification of the Aho-Corasick automaton [1]. For multiple patterns $P_1, \ldots, P_k$, our algorithm requires $O(m_k M)$ preprocessing time and runs in $O(m_k n)$ query time using $O(m_k M)$ space, where $M = \sum_{i=1}^{k} |P_i|$ and $m_k = \max(|P_i|)$.

2 Preliminaries

Given a finite set Σ of characters and a string w over Σ, let $|w|$ be the length of w and $w[i]$ be the symbol of w at position i, for $1 \leq i \leq |w|$. We define the empty string λ as a string of length 0. We use $w[i : j]$ to denote a substring $w[i]w[i{+}1] \cdots w[j]$, where $1 \leq i \leq j \leq |w|$. A language over Σ is a set of strings over Σ. A finite-state automaton (FA) $\mathcal{A}$ is specified by $\mathcal{A} = (Q, \Sigma, \delta, s, F)$, where Q is a set of states, Σ is an alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. A string w is accepted by $\mathcal{A}$ if there is a labeled path from s to a state in F such that the path spells out w. The language $L(A)$ of an FA $\mathcal{A}$ is the set of all strings accepted by $\mathcal{A}$. For more background knowledge in automata theory, the reader may refer to textbooks [9, 13].

For a string w, let w^R denote the reversed string of w. A string w is called a *palindrome* if $w = w^R$. The *radius* of a palindrome w is $\frac{|w|}{2}$. The *center* of a palindromic substring $w[i : j]$ of a string w is $\frac{i+j}{2}$. We call a palindromic substring $w[i : j]$ the *maximal palindrome* at the center $\frac{i+j}{2}$ if no other palindromes at the center $\frac{i+j}{2}$ have a larger radius than $w[i : j]$. Let $Pals(w)$ be the set of pairs of the center and the radius of all center-distinct maximal palindromes [10]. For two strings w and z of the same length, we say that w and z are *pal-equivalent* if $Pals(w) = Pals(z)$.

Definition 1 (Online Multiple Palindrome Pattern Matching). *Given a text T of length n and patterns $P_1, \ldots, P_k$ of length $m_1, \ldots, m_k$, find all pairs of an index i and a corresponding pattern P_j such that $Pals(P_j) = Pals(T[i{-}m_j{+}1 : i])$ after reading each character $T[i]$.*

For the online pattern matching, we call the time to preprocess the patterns *preprocessing time*, and the time to read the text to find matchings *query time*.

3 The Algorithm

The basic idea of the algorithm is to process multiple patterns at once with a single automaton based on the idea of the Aho-Corasick automaton [1]. Assume that given patterns $P_1, \ldots, P_k$ of length $m_1, \ldots, m_k$ are sorted by ascending order with respect to the length of the pattern and M is the sum of all pattern lengths. Before we design an algorithm, we have the following observation:

Observation 1. *For strings w, z and an index i, if there exists $(c, r) \in Pals(w)$ where $c \leq i$ and $c+r-0.5 \geq i$, then $z[i] = z[2r-i]$. If there is no (c, r) satisfying the condition, then $z[i] \notin \{z[2r-i] \mid (c, r) \in Pals(w)$ and $c + r - 0.5 = i - 1\}$.*

Note that $z[i]$ is computed based on $z[l]$'s for $l < i$, instead of characters in w. Based on Observation 1, we define a *variable pattern* of a pattern P as follows:

Definition 2. *For a pattern P of length m over Σ of size t, a* variable pattern *P' is defined by an array $\mathbb{A}[m]$ of variables and an array $\mathbb{B}[m]$ of unequal conditions satisfying the following conditions:*

1. *$P'[i] = \mathbb{A}[l_i]$ for $1 \leq i, l_i \leq m$.*
2. *If there exists $(c, r) \in Pals(P)$ where $c \leq i$ and $c+r-0.5 \geq i$, then $l_i = l_{2r-i}$, and thus, $P'[i] = P'[2r-i]$.*
3. *Otherwise, for all $j \in \{2r - i \mid (c, r) \in Pals(P)$ and $c + r - 0.5 = i - 1\}$, $\mathbb{B}[i] = j$ and $\mathbb{B}[j] = i$, and thus, $P'[i] \neq P'[j]$.*

Now we construct $P'_1, \ldots, P'_k$ simultaneously by Algorithm 1. All variable patterns share $\mathbb{A}$ while each variable pattern P'_j has a distinct array $\mathbb{B}[j][m]$ of unequal conditions in the algorithm. Fig. 1 shows an example of P' and $\mathbb{B}$.

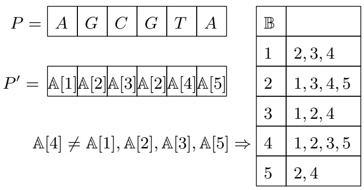

Fig. 1. A variable pattern P' and an array $\mathbb{B}$ of unequal conditions for $P = AGCGTA$

Based on Observation 1 and Definition 2, we establish the following result:

Lemma 1. *After running Algorithm 1, if there is a surjection of $\mathbb{A}$ to Σ where $\mathbb{A}[i] \neq \mathbb{A}[j]$ holds for all i, j such that $j \in \mathbb{B}[l][i]$, then $Pals(P'_l) = Pals(P_l)$. Moreover, given a string w such that $Pals(w) = Pals(P_l)$, there exists a surjection of $\mathbb{A}$ to Σ such that $P'_l = w$.*

Once we have $P'_1, \ldots, P'_k$, we can construct a special automaton $\mathcal{B} = (Q, \mathbb{A} \cup \{\#\}, \delta : Q \times \mathbb{A} \to Q, s, F, \Sigma, \mathbb{B}, \delta_f : Q \to Q, \mathcal{H} : Q \to 2^{\mathbb{A} \times (\mathbb{A} \cup \{\#\})}, \delta_p : Q \to Q)$. Note that five parameters—$\Sigma, \mathbb{B}, \delta_f, \mathcal{H}, \delta_p$—are added to the definition of a traditional FA. The automaton $\mathcal{B}$ simulates the Aho-Corasick algorithm [1], using $P'_1, \ldots, P'_k$ as patterns. In the Aho-Corasick algorithm, when there occurs a mismatch, the algorithm checks the longest suffix of the prefix of T read so far. The automaton $\mathcal{B}$ simulates the process by δ_f, and additionally, changes surjection of $\mathbb{A}$ to Σ according to $\mathcal{H}$. The suffix transition function δ_p contains transitions to find multiple matching occurrences on a single state. Algorithm 2 constructs

$\mathcal{B}$. We use a supplementary function StateForVP to return the state denoting the end of a given variable pattern. Fig. 2 shows an example of $\mathcal{B}$.

Algorithm 1. ConstructMultiVariablePattern

Input: Patterns $P_1, \ldots, P_k$ of length $m_1, \ldots, m_k$ over Σ of size t
Output: $P'_1, \ldots, P'_k$,$\mathbb{A}[m_k]$,$\mathbb{B}[k][m_k]$

for $j \leftarrow 1$ **to** k **do**
  compute $Pals(P_j)$ `// we insert` $(0.5, 0)$ `to` $Pals(P_j)$ `for convenience`
  $c \leftarrow 0.5$, $d \leftarrow 0$, $s \leftarrow 0$
  for $i \leftarrow 1$ **to** m_j **do**
    find r such that $(c, r) \in Pals(P_j)$
    if $d \geq i$ **then** $P'_j[i] \leftarrow P'_j[2r{-}i]$ **else**
      $s \leftarrow s + 1$, $P'_j[i] \leftarrow \mathbb{A}[s]$
      for each $(c', r') \in Pals(P_j)$ **do**
        if $c' + r' - 0.5 = i - 1$ **then**
          add $2r' - i$ to $\mathbb{B}[j][i]$, add i to $\mathbb{B}[j][2r'{-}i]$
    find r_1, r_2 such that $(c + 0.5, r_1), (c + 1, r_2) \in Pals(P_j)$
    $d \leftarrow \max(d, c + r_1, c + r_2 + 0.5)$, $r \leftarrow r + 1$
return $P'_1, \ldots, P'_k, \mathbb{A}, \mathbb{B}$

Algorithm 2. ConstructMultiAutomaton

Input: Patterns $P_1, \ldots, P_k$ of length $m_1, \ldots, m_k$ over Σ of size t
Output: $\mathcal{B} = (Q, \mathbb{A} \cup \{\#\}, \delta, s, F, \Sigma, \mathbb{B}, \delta_f, \mathcal{H}, \delta_p)$

ConsturctMultiVariablePattern($P_1, \ldots, P_k$)
add q_λ to Q and let $p_1, \ldots, p_k \leftarrow q_\lambda$
for $i \leftarrow 1$ **to** $m_k + 1$ **do**
  for each P'_j *where* $i \leq m_j + 1$ **do**
    let $P'_j[i] = \mathbb{A}[l]$ and $p_j = q_s$
    if $i \neq m_j + 1$ **then** $\delta(q_s, \mathbb{A}[l]) \leftarrow q_{s \cdot l}$, add $q_{s \cdot l}$ to Q **if** $i = 2$ **then** $\delta_f(q_s) \leftarrow q_\lambda$, add $(\mathbb{A}[1] \leftarrow \#)$ to $\mathcal{H}(q_s)$ **else if** $i > 2$ **then**
      find the smallest i' and corresponding j' such that $Pals(P'_{j'}[1 : i{-}i']) = Pals(P'_j[i' : i{-}1])$
      $\delta_f(q_s) \leftarrow$StateForVP($P'_{j'}[1 : i{-}i']$)
      for $g \leftarrow 1$ **to** $i - i'$ **do**
        add $(\mathbb{A}[h] \leftarrow \mathbb{A}[h'])$ to $\mathcal{H}(q_s)$ for $P'_{j'}[g] = \mathbb{A}[h]$ and $P'_j[g{+}i'{-}1] = \mathbb{A}[h']$
      for each $\mathbb{A}[h]$ *in* $P'_j[1 : i{-}1]$ *without injective function in* $\mathcal{H}(q_s)$ **do**
        add $(\mathbb{A}[h] \leftarrow \#)$ to $\mathcal{H}(q_s)$
    if $i = m_j$ **then** add $q_{s \cdot l}$ to F find the largest i' and corresponding j' such that $Pals(P'_{j'}[1 : i']) = Pals(P'_j[i{-}i'{+}1 : i])$
    if $i' = m_{j'}$ **then** $\delta_p(p_j) \leftarrow$StateForVP($P'_{j'}$) $p_j \leftarrow q_{s \cdot l}$
return $(Q, \mathbb{A} \cup \{\#\}, \delta, q_\lambda, F, \Sigma, \mathbb{B}, \delta_f, \mathcal{H}, \delta_p)$

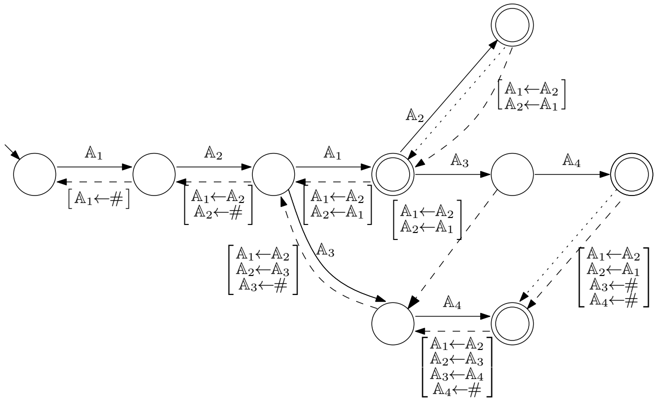

Fig. 2. An automaton $\mathcal{B}$ for $P_1 = AGA, P_2 = ACTG, P_3 = ATAT, P_4 = TCTGC$. Variables $\mathbb{A}[i]$ are written as $\mathbb{A}_i$ for better readability. Dashed transitions are failure transitions and dotted transitions are suffix transitions.

Algorithm 3. FindMultiPalindromeMatching

Input: Patterns $P_1, \ldots, P_k$ of length $m_1, \ldots, m_k$ over Σ of size t
Output: (i, P_j) such that $Pals(P_j) = Pals(T[i{-}m_j{+}1 : i])$

ConstructMultiAutomaton($P_1, \ldots, P_k$)
for $i \leftarrow 1$ **to** m_k **do** $\mathbb{A}[i] \leftarrow \#$ $q_l \leftarrow q_\lambda$ `// current state`
for $i \leftarrow 1$ **to** n **do**
  while *one of the following conditions holds for all* $\mathbb{A}[j]$ *such that* $\delta(q_l, \mathbb{A}[j]) \neq \emptyset$
    1. $q_l \in F$
    2. $\mathbb{A}[j] \neq T[i], \#$
    3. $\mathbb{A}[j] = \#$ *and there exists* $j' \in \mathbb{B}[j][g]$ *such that* $\mathbb{A}[j'] = T[i]$ *and* $\delta(q_l, \mathbb{A}[j]) = \mathit{StateForVP}(P'_g[1 : |l|{+}1])$
  do
    for each $(\mathbb{A}[h] \leftarrow \mathbb{A}[h']) \in \mathcal{H}(q_l)$ **do** $\mathbb{A}[h] \leftarrow \mathbb{A}[h']$ $q_l \leftarrow \delta_f(q_l)$
  if $\mathbb{A}[j] = \#$ **then** $\mathbb{A}[j] \leftarrow T[i]$ $q_l \leftarrow \delta(q_l, \mathbb{A}[j])$
  if $q_l \in F$ **then return** $(i, P_{j'})$ where StateForVP$(P'_{j'}) = q_l$ $p_l \leftarrow q_l$
  while $\delta_p(p_l) \neq \emptyset$ **do**
    $p_l \leftarrow \delta_p(p_l)$
    return $(i, P_{j'})$ where StateForVP$(P'_{j'}) = p_l$

Now we are ready to design an algorithm that solves the problem using $\mathcal{B}$. Algorithm 3 processes T in $\mathcal{B}$ and reports all matching end-indices and the corresponding matching patterns.

Lemma 2. *Algorithm 3 returns all pairs of an index i and a pattern P_j such that $Pals(P_j) = Pals(T[i{-}m_j{+}1 : i])$.*

Theorem 2. *Given a text T of length n and a pattern P of length m, we can solve the online multiple palindrome pattern matching problem with $O(m_k M)$ preprocessing time and $O(m_k n)$ query time using $O(m_k M)$ space.*

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Communications of the ACM 18(6), 333–340 (1975)
2. Allouche, J.-P., Baake, M., Cassaigne, J., Damanik, D.: Palindrome complexity. Theoretical Computer Science 292(1), 9–31 (2003)
3. Anisiu, M.-C., Anisiu, V., Kása, Z.: Total palindrome complexity of finite words. Discrete Mathematics 310(1), 109–114 (2010)
4. Brlek, S., Hamel, S., Nivat, M., Reutenauer, C.: On the palindromic complexity of infinite words. International Journal of Foundations of Computer Science 15(2), 293–306 (2004)
5. Droubay, X., Justin, J., Pirillo, G.: Episturmian words and some constructions of de luca and rauzy. Theoretical Computer Science 255(1-2), 539–553 (2001)
6. Glen, A., Justin, J., Widmer, S., Zamboni, L.Q.: Palindromic richness. European Journal of Combinatorics 30(2), 510–531 (2009)
7. Groult, R., Prieur, É., Richomme, G.: Counting distinct palindromes in a word in linear time. Information Processing Letters 110(20), 908–912 (2010)
8. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press (1997)
9. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison–Wesley (1979)
10. Tomohiro, I., Inenaga, S., Bannai, H., Takeda, M.: Counting and verifying maximal palindromes. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 135–146. Springer, Heidelberg (2010)
11. Tomohiro, I., Inenaga, S., Bannai, H., Takeda, M.: Palindrome pattern matching. Theoretical Computer Science 483, 162–170 (2013)
12. Manacher, G.: A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. Journal of the ACM 22(3), 346–351 (1975)
13. Wood, D.: Theory of Computation. Harper & Row (1986)