A Minimal Periods Algorithm with Applications

Zhi Xu

The University of Western Ontario,
Bio-computing Laboratory, Computer Science Department

CPM2010
21 June 2010, Brooklyn, NY, USA

Organization

Western
School of Graduate and Postdoctoral Studies

Repetitions in Words

We assume the alphabet is fixed in the entire discussion.

Definitions

square word of form $w = uu$, $u \neq \epsilon$, e.g., **gaga**.

cube word of form $w = u^3$, $u \neq \epsilon$, e.g., **chachacha**.

(integer) power word of form $w = u^k = \overbrace{uu \cdots u}^{k}$, $u \neq \epsilon$.

fractional power word of form $w = u^k v$, v is a prefix of u, $u \neq \epsilon$. We call $|w|/|u|$ exponent and u period, e.g., **alfalfa**=(**alf**)(**alf**)**a** is a $7/3$ power.

Repetitions in Words

We assume the alphabet is fixed in the entire discussion.

Definitions

square word of form $w = uu$, $u \neq \epsilon$, e.g., **gaga**.

cube word of form $w = u^3$, $u \neq \epsilon$, e.g., **chachacha**.

(integer) power word of form $w = u^k = \overbrace{uu \cdots u}^{k}$, $u \neq \epsilon$.

fractional power word of form $w = u^k v$, v is a prefix of u, $u \neq \epsilon$. We call $|w|/|u|$ exponent and u period, e.g., **alfalfa**=(**alf**)(**alf**)**a** is a $7/3$ power.

repetition fractional power with exponent ≥ 2.

square-free word not of the form $w = xuuy$, $u \neq \epsilon$, e.g., **square** is square-free, while **square-free** is not.

X-free for various X is defined similarly.

Research on Repetitions in Words

- Dating back to as early as 1900's (Thue 1906, Thue 1902);
- Testing square-freeness (Crochemore 1983, Main and Lorentz 1985);

Research on Repetitions in Words

- Dating back to as early as 1900's (Thue 1906, Thue 1902);
- Testing square-freeness (Crochemore 1983, Main and Lorentz 1985);
- Counting the number of syntactically distinct squares (Fraenkel and Simpson 1998, Ilie 2007);
- Finding appearance of primitively-rooted squares (Crochemore 1981, Apostolico and Preparata 1983, Main and Lorentz 1984, Franek, Smyth and Tang 2003)

Research on Repetitions in Words

- Dating back to as early as 1900's (Thue 1906, Thue 1902);
- Testing square-freeness (Crochemore 1983, Main and Lorentz 1985);
- Counting the number of syntactically distinct squares (Fraenkel and Simpson 1998, Ilie 2007);
- Finding appearance of primitively-rooted squares (Crochemore 1981, Apostolico and Preparata 1983, Main and Lorentz 1984, Franek, Smyth and Tang 2003)
- Finding/counting syntactically distinct maximal repetitions (Slisenko 1983, Gusfield and Stoye 1998, Rytter 2006);
- Finding appearance of maximal repetitions (Main 1989, Kolpakov and Kucherov 1999);

From the Local Point of View

Definitions

The prefix period of w with respect to exponent α and integer s is the shortest word x in $\{ x : |x| > s, w = x^{\beta} y, \beta \geq \alpha \}$. We denote its length by $pp_s^{\alpha}(w)$. If no such x exists, we write $pp_s^{\alpha}(w) = +\infty$.

The strict prefix period with respect to α, s is the shortest word x in $\{ x : |x| > s, w = x^{\alpha} y \}$, and we denote its length by $\overline{pp}_s^{\alpha}(w)$.

From the Local Point of View

Definitions

The prefix period of w with respect to exponent α and integer s is the shortest word x in $\{ x : |x| > s, w = x^\beta y, \beta \geq \alpha \}$. We denote its length by $pp_s^\alpha(w)$. If no such x exists, we write $pp_s^\alpha(w) = +\infty$.

The strict prefix period with respect to α, s is the shortest word x in $\{ x : |x| > s, w = x^\alpha y \}$, and we denote its length by $\overline{pp}_s^\alpha(w)$.

Examples

Let $w = \mathbf{0100101001}$. Then $pp_0^2(w) = 3$, $pp_0^3(w) = +\infty$, $pp_3^2(w) = 5$, $pp_0^{3/2}(w) = 2$, $pp_0^{5/4}(w) = 2$, and $\overline{pp}_0^{5/4}(w) = 8$.

From the Local Point of View (Cont.)

Definitions

The right minimal period array $rmp_s^\alpha(w)$ is defined as

$$rmp_s^\alpha(w)[i] = pp_s^\alpha(w[i\,..\,|w|]),\ \overline{rmp}_s^\alpha(w)[i] = \overline{pp}_s^\alpha(w[i\,..\,|w|]),$$

and the left minimal period array $lmp_s^\alpha(w)$ is defined as

$$lmp_s^\alpha(w)[i] = pp_s^\alpha(w[1\,..\,i]^R),\ \overline{lmp}_s^\alpha(w)[i] = \overline{pp}_s^\alpha(w[1\,..\,i]^R).$$

From the Local Point of View (Cont.)

Definitions

The right minimal period array $rmp_s^\alpha(w)$ is defined as

$$rmp_s^\alpha(w)[i] = pp_s^\alpha(w[i\,..\,|w|]),\ \overline{rmp}_s^\alpha(w)[i] = \overline{pp}_s^\alpha(w[i\,..\,|w|]),$$

and the left minimal period array $lmp_s^\alpha(w)$ is defined as

$$lmp_s^\alpha(w)[i] = pp_s^\alpha(w[1\,..\,i]^R),\ \overline{lmp}_s^\alpha(w)[i] = \overline{pp}_s^\alpha(w[1\,..\,i]^R).$$

Examples

Let $w = \mathbf{0100101001}$. Then

$$rmp_0^2(w) = [3, +\infty, 1, 2, 2, +\infty, +\infty, 1, +\infty, +\infty],$$
$$rmp_1^{3/2}(w) = [2, 3, 5, 2, 2, 2, +\infty, +\infty, +\infty, +\infty],$$
$$\overline{rmp}_1^{3/2}(w) = [2, +\infty, +\infty, 2, 2, 2, +\infty, +\infty, +\infty, +\infty].$$

Finding the Minimal Period Starting at Each Index

Problem

Compute $rmp_s^\alpha(w), lmp_s^\alpha(w), \overline{rmp}_s^\alpha(w), \overline{rmp}_s^\alpha(w)$ *of word* w.

Finding the Minimal Period Starting at Each Index

Problem

Compute $rmp_s^\alpha(w), lmp_s^\alpha(w), \overline{rmp}_s^\alpha(w), \overline{rmp}_s^\alpha(w)$ *of word* w.

Global V.S. Local

The computation of $rmp_s^\alpha(w)$ can not be approached directly by the computation of maximal repetition with the same efficiency, since there may be $\Omega(\log |w|)$ primarily-rooted maximal repetitions starting from the same position in w. On the other hand, the maximal repetitions can not be obtained by computing $rmp_s^\alpha(w)$.

Finding the Minimal Period Starting at Each Index

Problem

Compute $rmp_s^\alpha(w), lmp_s^\alpha(w), \overline{rmp}_s^\alpha(w), \overline{rmp}_s^\alpha(w)$ *of word* w.

Global V.S. Local

The computation of $rmp_s^\alpha(w)$ can not be approached directly by the computation of maximal repetition with the same efficiency, since there may be $\Omega(\log |w|)$ primarily-rooted maximal repetitions starting from the same position in w. On the other hand, the maximal repetitions can not be obtained by computing $rmp_s^\alpha(w)$.

Research on Repetitions from a Local Point of View

The case rmp_0^2 (Kosaraju 1994); another setting (Duval, Kolpakov, Kucherov, Lecroq, and Lefebvre 2004).

Suffix Tree

Definition

A suffix tree $\mathcal{T}_w$ for $w = w[1..n]$ is a labeled and rooted tree that satisfies the following conditions.

1. All internal nodes excluding the root have ≥ 2 children;
2. labels on edges from same node begin with different letters;
3. there are exactly n leaves, $leaf_i$, and $\tau(leaf_i) = w[i..n]\$$,

where $\tau(v)$ is the concatenation of labels along the path from the root to the node v, and \$ is a special letter not in w.

Suffix Tree

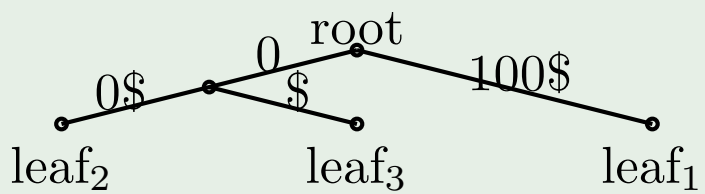
Definition

A suffix tree $\mathcal{T}_w$ for $w = w[1..n]$ is a labeled and rooted tree that satisfies the following conditions.

1. All internal nodes excluding the root have ≥ 2 children;
2. labels on edges from same node begin with different letters;
3. there are exactly n leaves, $leaf_i$, and $\tau(leaf_i) = w[i..n]\$$,

where $\tau(v)$ is the concatenation of labels along the path from the root to the node v, and \$ is a special letter not in w.

Example (the suffix tree for **100**)



Suffix Tree Construction

Algorithm

Construction of $\mathcal{T}_w$ for w can be in linear time (for fixed alphabet) (Weiner 1973, McCreight 1976, Ukkonen 1992).

Suffix Tree Construction

Algorithm

Construction of $\mathcal{T}_w$ for w can be in linear time (for fixed alphabet) (Weiner 1973, McCreight 1976, Ukkonen 1992).

Framework of Weiner's Algorithm (Weiner 1973)

begin function `make_suffix_tree(`w`)`
  $T_n \longleftarrow$ suffix tree for $w[n..n]$;
  for i **from** $n-1$ **to** 1 **do** $T_i \longleftarrow$`extend(`T_{i+1}, $w[i..n]$`)` ;
  return T_1 ;
end

Suffix Tree Construction (Cont.)

Framework of Weiner's Algorithm (Cont.)

begin function `extend`(*tree*, $word[i \,..\, n]$)
  find the proper position y in *tree* to insert the new node $leaf_i$;
  if needed, split an edge $x \longrightarrow z$ to $x \longrightarrow y, y \longrightarrow z$ by a new y ;
  create and label the edge $y \longrightarrow leaf_i$ by $word[i + |\tau(y)| \,..\, n]\$$;
end

Details of locating the x, y, z is not relevant and thus omitted.

Suffix Tree Construction (Cont.)

Framework of Weiner's Algorithm (Cont.)

begin function `extend`(*tree*, *word*$[i \,..\, n]$)
  find the proper position y in *tree* to insert the new node $leaf_i$;
  if needed, split an edge $x \longrightarrow z$ to $x \longrightarrow y, y \longrightarrow z$ by a new y ;
  create and label the edge $y \longrightarrow leaf_i$ by *word*$[i + |\tau(y)| \,..\, n]\$$;
end

Details of locating the x, y, z is not relevant and thus omitted.

Example (construct suffix tree for **100**)

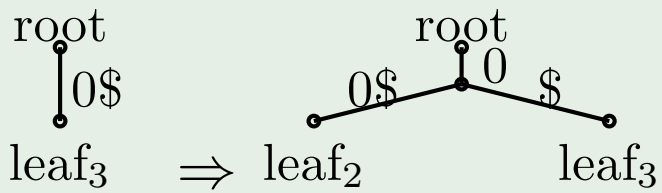
Suffix Tree Construction (Cont.)

Framework of Weiner's Algorithm (Cont.)

begin function `extend`(*tree*, *word*$[i \,..\, n]$)
  find the proper position y in *tree* to insert the new node $leaf_i$;
  if needed, split an edge $x \longrightarrow z$ to $x \longrightarrow y, y \longrightarrow z$ by a new y ;
  create and label the edge $y \longrightarrow leaf_i$ by *word*$[i + |\tau(y)| \,..\, n]\$$;
end

Details of locating the x, y, z is not relevant and thus omitted.

Example (construct suffix tree for **100**)



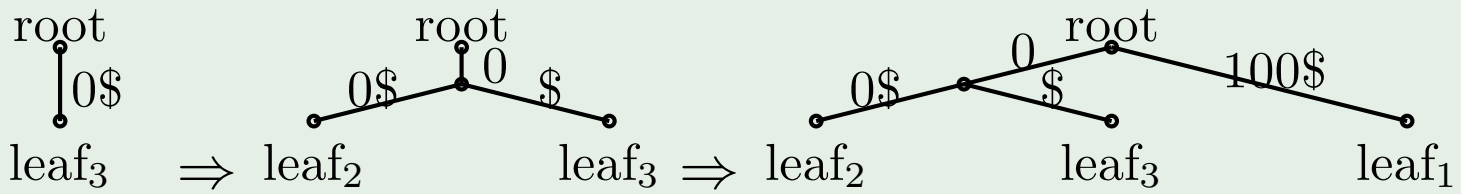
Suffix Tree Construction (Cont.)

Framework of Weiner's Algorithm (Cont.)

begin function `extend`(*tree*, *word*$[i \,..\, n]$)
  find the proper position y in *tree* to insert the new node $leaf_i$;
  if needed, split an edge $x \longrightarrow z$ to $x \longrightarrow y, y \longrightarrow z$ by a new y ;
  create and label the edge $y \longrightarrow leaf_i$ by *word*$[i + |\tau(y)| \,..\, n]$\$;
end

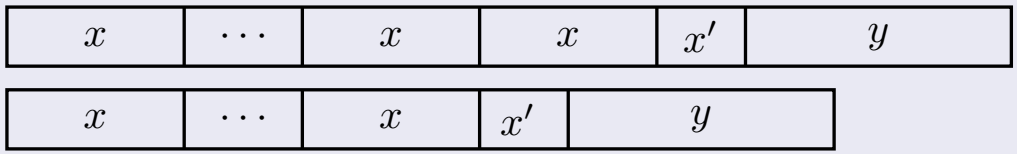
Details of locating the x, y, z is not relevant and thus omitted.

Example (construct suffix tree for **100**)



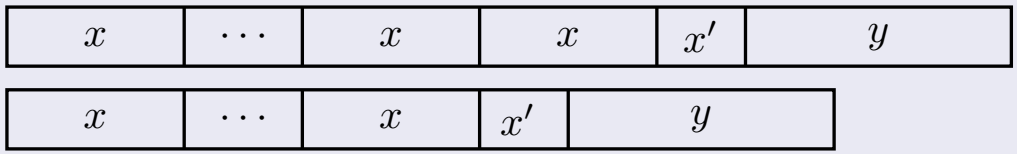
Compute $pp_s^{\alpha}(w)$ from the Suffix Tree of w

Prefix of w has a period x of length $i-1$ with exponent α if and only if $|\mathrm{lcp}(w[1\,..\,|w|], w[i\,..\,|w|])| \geq (\alpha-1)(i-1)$:



Compute $pp_s^{\alpha}(w)$ from the Suffix Tree of w

Prefix of w has a period x of length $i-1$ with exponent α if and only if $|\mathrm{lcp}(w[1..|w|], w[i..|w|])| \geq (\alpha-1)(i-1)$:



Compute Prefix Period in $O(|w|)$ Time

begin function `compute_pp`(*tree*, *s*, α)
  pp $\longleftarrow +\infty$; preprocessing *tree* for lca operation ;
  foreach $leaf_i$ *other than* $leaf_1$ **do**
    if $|\tau(\mathrm{lca}(leaf_1, leaf_i))| \geq (\alpha-1)(i-1)$ *and* $i-1 > s$ **then**
      if $pp > i-1$ **then** $pp \longleftarrow i-1$;
  return *pp* ;
end

Compute $pp_s^\alpha(w)$ from the Suffix Tree of w (Cont.)

Compute Prefix Period in $O(|w|/\min\{s+1, pp_0^\alpha(w)\})$ Time

begin function `compute_pp`(*tree*, s, α)
  if $\alpha(s+1) > n$ **then return** $+\infty$; **else** $h \longleftarrow leaf_1$;
  while $|\tau(p(h))| \geq (\alpha-1)(s+1)$ **do** $h \longleftarrow p(h)$;
  $pp \longleftarrow +\infty$; preprocessing the tree rooted at h for lca ;
  foreach $leaf_i$ *being a descendent of* h *other than* $leaf_1$ **do**
    if $|\tau(\mathrm{lca}(leaf_1, leaf_i))| \geq (\alpha-1)(i-1)$ *and* $i-1 > s$ **then**
      if $pp > i-1$ **then** $pp \longleftarrow i-1$;
  return pp ;
end

Compute $\overline{pp}_s^\alpha(w)$ from the Suffix Tree of w

Compute Strict Prefix Period in $O(|w|/\min\{s+1, pp_0^\alpha(w)\})$ Time

begin function `compute_pp_strict`(*tree*, *s*, α)
  if $\alpha(s+1) > n$ **then return** $+\infty$; **else** $h \longleftarrow leaf_1$;
  while $|\tau(p(h))| \geq (\alpha-1)(s+1)$ **do** $h \longleftarrow p(h)$;
  $pp \longleftarrow +\infty$; preprocessing the tree rooted at h for lca ;
  foreach $leaf_i$ *being a descendent of* h *other than* $leaf_1$ **do**
    if $|\tau(\mathrm{lca}(leaf_1, leaf_i))| \geq (\alpha-1)(i-1)$ *and* $i-1 > s$ **then**
      if $i-1 \bmod den = 0$ *and* $pp > i-1$ **then** $pp \longleftarrow i-1$;
  return *pp* ;
end

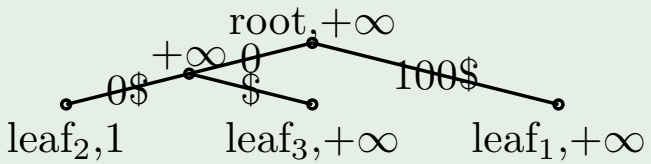
den is the denominator of $\alpha = num/den$, $\gcd(num, den) = 1$.

Suffix Tree with Prefix Periods

Definitions

For given $\alpha > 1, s \geq 0$, a suffix tree with prefix periods is a suffix tree with each node v labeled by $\pi(v) = pp_s^{\alpha}(\tau(v))$. The suffix tree with strict periods is defined analogously by $\overline{\pi}(v) = \overline{pp}_s^{\alpha}(\tau(v)))$.

Example (the suffix tree with (non-strict) prefix periods for **100**)

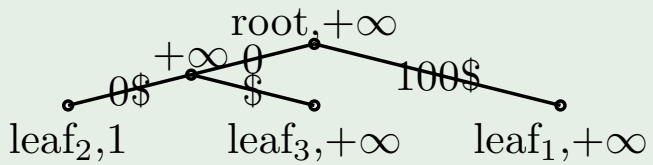


Suffix Tree with Prefix Periods

Definitions

For given $\alpha > 1, s \geq 0$, a suffix tree with prefix periods is a suffix tree with each node v labeled by $\pi(v) = pp_s^\alpha(\tau(v))$. The suffix tree with strict periods is defined analogously by $\overline{\pi}(v) = \overline{pp}_s^\alpha(\tau(v)))$.

Example (the suffix tree with (non-strict) prefix periods for **100**)



Let u be the parent of v in a suffix tree with prefix periods. 1 If $\pi(u) \neq +\infty$, then $\pi(v) = \pi(u)$. 2 If $\pi(v) \neq +\infty$ and $|\tau(u)| \geq \alpha\pi(v)$, then $\pi(u) = \pi(v)$; otherwise $\pi(u) = +\infty$.

This is also true for suffix tree with strict prefix periods.

Compute $rmp_s^{\alpha}(w)$ Using Suffix Tree

Compute Right Minimal Period Array in $O(|w|^2)$ Time

begin function `compute_rmp`(w, s, α)
  $T_n \longleftarrow$ suffix tree for $w[n..n]$ with $\pi(root), \pi(leaf_n) \longleftarrow +\infty$;
  for i **from** $n-1$ **to** 1 **do**
    $T_i \longleftarrow$`extend`(T_{i+1}, $w[i..n]$) ;
    if *splitting* **then** // y, z are obtained from `extend()`
      if $|\tau(y)| \geq \alpha\pi(z)$ **then** $\pi(y) \longleftarrow \pi(z)$; **else** $\pi(y) \longleftarrow +\infty$;
    if $\pi(y) \neq +\infty$ **then** $\pi(leaf_i) \longleftarrow \pi(y)$;
    else $\pi(leaf_i) \longleftarrow$`compute_pp`($T_i$, s, α) ;
    $rmp[i] \longleftarrow \pi(leaf_i)$;
  $rmp[n] \longleftarrow +\infty$ and **return** rmp ;
end

Compute $rmp_s^{\alpha}(w)$ Using Suffix Tree (Cont.)

Instead of `compute_pp`$(\mathcal{T}_{w[i\,..\,|w|]}, s, \alpha)$, we compute `compute_pp`$(\mathcal{T}_{w[i\,..\,j]}, s', \alpha)$ for some chosen j and s'.

Compute Right Minimal Period Array in $O(|w|^2)$ Time

begin function `compute_rmp`(w, s, α)
  $T_n \longleftarrow$ suffix tree for $w[n\,..\,n]$ with $\pi(root), \pi(leaf_n) \longleftarrow +\infty$;
  for i **from** $n-1$ **to** 1 **do**
    $T_i \longleftarrow$`extend`$(T_{i+1}, w[i\,..\,n])$;
    if *splitting* **then** // y, z `are obtained from extend()`
      if $|\tau(y)| \geq \alpha\pi(z)$ **then** $\pi(y) \longleftarrow \pi(z)$; **else** $\pi(y) \longleftarrow +\infty$;
    if $\pi(y) \neq +\infty$ **then** $\pi(leaf_i) \longleftarrow \pi(y)$;
    else $\pi(leaf_i) \longleftarrow$`compute_pp`$(T_i, s, \alpha)$;
    $rmp[i] \longleftarrow \pi(leaf_i)$;
  $rmp[n] \longleftarrow +\infty$ and **return** rmp ;
end

Compute $rmp_s^{\alpha}(w)$ Using Suffix Tree (Cont.)

Compute Right Minimal Period Array, $O(\alpha|w|)$ Time for $s = 0$

begin function `compute_rmp`(w, s, α)

$T_n \longleftarrow$ suffix tree for $w[n\,..\,n]$ with $\pi(root), \pi(leaf_n) \longleftarrow +\infty$;
$A \longleftarrow$ *empty*, $j \longleftarrow n$, and $d \longleftarrow 0$;
for i **from** $n-1$ **to** 1 **do**
  $T_i \longleftarrow$`extend`$(T_{i+1}, w[i\,..\,n])$;
  if *splitting* **then** // y, z are obtained from `extend()`
    if $|\tau(y)| \geq \alpha\pi(z)$ **then** $\pi(y) \longleftarrow \pi(z)$; **else** $\pi(y) \longleftarrow +\infty$;
  if $j - i + 1 > 2\alpha d/(\alpha - 1)$ *or* $|\tau(y)| < d/2$ **then** $A \longleftarrow$ *empty* ;
  if $\pi(y) \neq +\infty$ **then**
    $\pi(leaf_i) \longleftarrow \pi(y)$;
    if $A \neq$ *empty* **then** $A \longleftarrow$`extend`$(A, w[i\,..\,j])$;
  else
    if $A =$ *empty* **then**
      $d \longleftarrow |\tau(y)|$ and $j \longleftarrow i + (\alpha+1)d/(\alpha-1) - 1$;
      $A \longleftarrow$`make_suffix_tree`$(w[i\,..\,j])$;
    else
      $A \longleftarrow$`extend`$(A, w[i\,..\,j])$;
    $\pi(leaf_i) \longleftarrow$`compute_pp`$(A, \max\{s, |\tau(y)|/\alpha\}, \alpha)$;
  $rmp[i] \longleftarrow \pi(leaf_i)$;
$rmp[n] \longleftarrow +\infty$ and **return** rmp ;

end

Compute $rmp_s^\alpha(w)$ Using Suffix Tree — Correctness

In the algorithm, the auxiliary tree A is the suffix tree for $w[i\,..\,j]$.

Theorem

The algorithm is correct.

The only improvement is

$$\pi(leaf_i) \longleftarrow \texttt{compute\_pp}(A,\ \max\{s, |\tau(y)|/\alpha\},\ \alpha)\,.$$

By the construction in the algorithm, $pp_s^\alpha(w[i\,..\,n]) = pp_s^\alpha(w[i\,..\,j])$ and $pp_s^\alpha(w[i\,..\,n]) > |\tau(y)|/\alpha$ holds.

Compute $rmp_s^{\alpha}(w)$ Using Suffix Tree — Complexity

The algorithm is in $O(|w|^2)$ time as the one without using tree A.

Theorem

For $s = 0$, the algorithm is in $O(\alpha|w|)$ time.

When $s = 0$, each call of `compute_pp`$(A, \max\{s, |\tau(y)|/\alpha\}, \alpha)$ is in $O(\alpha)$ time and, by the construction in the algorithm, the total cost in the construction of trees A and tree T is in $O(|w|)$. So the total time is $O(\alpha|w|) + O(|w|) = O(\alpha|w|)$.

Compute $\overline{rmp}_s^{\alpha}(w)$ Using Suffix Tree

Compute Right Minimal Period Array, $O(\alpha|w|)$ Time for $s = 0$

begin function `compute_rmp`(w, s, α)

$T_n \longleftarrow$ suffix tree for $w[n\,..\,n]$ with $\pi(root), \pi(leaf_n) \longleftarrow +\infty$;
$A \longleftarrow$ *empty*, $j \longleftarrow n$, and $d \longleftarrow 0$;
for i **from** $n-1$ **to** 1 **do**
  $T_i \longleftarrow$`extend`$(T_{i+1}, w[i\,..\,n])$;
  if *splitting* **then** // y, z are obtained from `extend()`
    if $|\tau(y)| \geq \alpha\pi(z)$ **then** $\pi(y) \longleftarrow \pi(z)$; **else** $\pi(y) \longleftarrow +\infty$;
  if $j - i + 1 > 2\alpha d/(\alpha - 1)$ *or* $|\tau(y)| < d/2$ **then** $A \longleftarrow$ *empty* ;
  if $\pi(y) \neq +\infty$ **then**
    $\pi(leaf_i) \longleftarrow \pi(y)$;
    if $A \neq$ *empty* **then** $A \longleftarrow$`extend`$(A, w[i\,..\,j])$;
  else
    if $A =$ *empty* **then**
      $d \longleftarrow |\tau(y)|$ and $j \longleftarrow i + (\alpha+1)d/(\alpha-1) - 1$;
      $A \longleftarrow$`make_suffix_tree`$(w[i\,..\,j])$;
    else
      $A \longleftarrow$`extend`$(A, w[i\,..\,j])$;
    $\pi(leaf_i) \longleftarrow$`compute_pp_strict`$(A, \max\{s, |\tau(y)|/\alpha\}, \alpha)$;
  $rmp[i] \longleftarrow \pi(leaf_i)$;
$rmp[n] \longleftarrow +\infty$ and **return** rmp ;

end

Pseudo-Powers

In the Setting of DNA Sequence

DNA sequences can be viewed as words over alphabet { **A**, **C**, **G**, **T** }. There is a Watson-Crick complementarity **A**↔**T**, **C**↔**G**, which induces an antimorphic involution θ, e.g., $\theta(\mathbf{AC})$=**GT**.

Definitions

antimorphic involution a function $\theta : \Sigma^* \to \Sigma^*$ that satisfies $\theta(\theta(w)) = w$ and $\theta(uv) = \theta(v)\theta(u)$ for $w, u, v \in \Sigma^*$, e.g., the Watson-Crick complementarity, mirror image (reverse of word).

pseudo-power word of the form $w = u_1 u_2 \cdots u_k$ such that either $u_i = u_j$ or $u_i = \theta(u_j)$ for $1 \leq i, j \leq k$, e.g., **ACGCGTCGTACG**=**ACG**θ(**ACG**)θ(**ACG**)**ACG** is a pseudo 4th power.

Testing Pseudo-Power-Freeness

Algorithms (Chiniforooshan, Kari, Xu 2009)

- Testing pseudo-square-freeness in $O(|w|)$ time;
- Testing pseudo-cube-freeness in $O(|w|^2)$ time;
- Testing pseudo-kth-power-freeness in $O(|w|^2 \log |w|)$ time.

Problem

Testing whether a word w contains any pseudo kth power of the form $\theta(u)\underbrace{uu\cdots u}_{k-1}$ (and of the form $\underbrace{uu\cdots u}_{k-1}\theta(u)$).

Centralized Maximal Pseudo-Palindrome Array

Definition

Let θ be an antimorphic involution. The centralized maximal pseudo-palindrome array cmp_w^θ of word w is defined as $cmp_w^\theta[i] = \max\{m : 0 \leq m \leq \min\{i, |w| - i\}, \theta(w[i - m + 1 .. i]) = w[i + 1 .. i + m]\}$ for $0 \leq i \leq |w|$.

Example

Let $w = \mathbf{0100101001}$ and let θ be the mirror image. Then $cmp_w^\theta = [0, 0, 0, 3, 0, 0, 0, 0, 2, 0, 0]$.

Algorithm (Ilie)

cmp_w^θ for w can be computed in $O(|w|)$ time by constructing the suffix tree for the word $w\$\theta(w)$, where letter $\$$ is not in w.

Testing $\theta(u)u^{k-1}$-Freeness

Algorithm

Testing whether a word w contains factor of the form $\theta(u)\overbrace{uu\cdots u}^{k-1}$ can be done in $O(k|w|)$ time.

Algorithm for Testing $\theta(u)u^{k-1}$-Freeness

1. Computing $rmp_0^{k-1}(w)$;
2. Computing cmp_w^{θ};
3. Testing whether $rmp_0^{k-1}(w)[i] \leq cmp_w^{\theta}[i-1]$ for some $1 \leq i \leq |w|$:

$$\cdots \underline{\theta(u)}\overline{uu\cdots u}\cdots$$

Main Points

We generalize Kosaraju's algorithm for computing minimal squares (i.e. $rmp_0^2(w)$) to computing $rmp_s^\alpha(w)$ and $\overline{rmp}_s^\alpha(w)$ for arbitrary rational exponent $\alpha > 1$ and integer $s \geq 0$. The algorithm is in $O(|w|^2)$ time for arbitrary s and is in $O(k|w|)$ time for $s = 0$.

In particular, the following problem is in linear time: for word w, computing the minimal kth power that begins at each letter in w.

In the end, testing whether a word contains a pseudo kth power of the form $\phi(u)uu\cdots u$ can be done in linear time.

Main Points

We generalize Kosaraju's algorithm for computing minimal squares (i.e. $rmp_0^2(w)$) to computing $rmp_s^\alpha(w)$ and $\overline{rmp}_s^\alpha(w)$ for arbitrary rational exponent $\alpha > 1$ and integer $s \geq 0$. The algorithm is in $O(|w|^2)$ time for arbitrary s and is in $O(k|w|)$ time for $s = 0$.

In particular, the following problem is in linear time: for word w, computing the minimal kth power that begins at each letter in w.

In the end, testing whether a word contains a pseudo kth power of the form $\phi(u)uu\cdots u$ can be done in linear time.

Try It Yourself

`http://www.csd.uwo.ca/~zhi_xu/demons/cpm2010xu.html`

This is the end. Thank you!