J. KSIAM **Vol.9**, No.1, 1-7, 2005

$O(N \log N)$ ALGORITHM FOR FINDING PRIMARY TANDEM REPEATS IN A DNA GENOMIC SEQUENCE

SANGBACK MA, HYEONG-HWA JUN

ABSTRACT. The genomes of organism are being published in an enormous speed. The genomes has a lot of intronic regions, and repeats constitute a substantial part of that. Repeats play a crucial role in DNA finger-printing, and detecting certain genomic diseases, such as Huntington disease, which has a high number of CAG repeats. Also, they throw important clues about the evolutionary history. Repeats are in two types, *Tandem Repeats* and *Interspersed Repeats*. In this paper we address ourselves to the problem of detecting *Primary Tandem Repeats*, which are tandem repeats that are not contained in any tandem repeats. We show that our algorithm takes $O(n \log n)$ time, where n is the length of genome.

1. INTRODUCTION

Ever since human genome sequencing has begun the amount of genome data is increasing at an enormous speed. How to interpret the genome is critical to understanding the secrets of life.

DNA sequences has many repeats interspersed throughout the genome. Repeats are classified by tandem .epeats and interspersed repeats. Tandem repeat is arbitrary length of nucleotides repeated two or more times, for example, aattggaattgg.

Primary tandem repeats is tandem repeats that are not contained in any tandem repeats. e.g., TCATTCATTCATTCAT(sequence) has TCATTCAT(primary tandem repeat).

We suggest an algorithm for finding all primary tandem repeats in a DNA genomic sequence based on a recursive algorithm. The time complexity of our algorithm is $O(N \log N)$.

2. METHODS

Basic algorithm is algorithm that finds all repetitions in a string in time $O(n \log n)$, where n is the strings length[1].

Our algorithmis based on a linear procedure for finding all new repetitions that are formed when two strings are concatenated. The following is a variant of Knuth-Morris-Pratt pattern matching algorithm [2]. We compute the following arrays through preprocessing phase. Let *pattern* and *text* be strings of length m and n, respectively.

: *lppattern* : *lppattern*[*i*] is the length of the longest substring of *pattern* which begins at position i and is a prefix of *pattern* $(1 < i \leq m)$.

: *lptext* : *lptext*[*i*] is the length of the longest substring of *text* which begins at position i and is a prefix of *pattern* $(1 \leq i \leq n)$.

: *lspattern* : *lspattern*[*i*] is the length of the longest substring of *pattern* which ends at position i and is a suffix of *pattern* $(1 \leq i < m)$.

: *lstext* : *lstext*[*i*] is the length of the longest substring of *text* which ends at position i and is a suffix of *pattern* $(1 \leq i \leq n)$.

ALGORITHM 1. *Procedure to Find All Primary Tandem Repeats in String uv.*

Input : u, v - string. q - queue of repetition ranges.
Output : the positions of all primary tandem repeats in string uv.

```
procedure $newptreps(u, v)$ begin
  calculate $rlstext[1] \cdots rlstext[|v|]$ and $rlppattern[2] \cdots rlppattern[|v|+1]$. /* right
*/
  calculate $llspattern[1] \cdots llspattern[|u|]$ and $llptext[1] \cdots llptext[|u|]$. /* left */
  $rightflag \leftarrow true$;
  $leftflag \leftarrow true$;
  for $n \leftarrow \lfloor(|u| + |v|)/2\rfloor$ downto 1 do begin
    /* right */
    if $(rightflag = true)$ then begin
      $b \leftarrow |u| - n + 1$;
      $e \leftarrow |u| + 2n - 1$;
      if $(b < 1)$ then $b \leftarrow 1$;
      if $(e > |u| + |v|)$ then $e \leftarrow |u| + |v|$;
      if $(b \geq right.begin$ and $e \leq right.end)$ then $rightflag \leftarrow flase$;
      else begin
        $first \leftarrow 2n - rlstext[n]$;
        $last \leftarrow$ minimum$(2n - 1, n + rlppattern[n + 1])$;
        if $(last < first)$ then there are no new right repetitions of length $2n$.
        else begin
          new right repetitions of length $2n$ end at $first$ through $last$ in $v$.
          find new primary tandem repeats ranges that are not included in $q$.
          add new primary tandem repeats ranges in $q$.
```

```
            /* adjust $right.begin$ and $right.end$. */
            $right.begin \leftarrow$ leftmost position in new primary tandem repeats ranges.
            $right.end \leftarrow$ rightmost position in new primary tandem repeats ranges.
          end
        end
      end
      /* left */
      if ($leftflag = true$) then begin
        $b \leftarrow |u| - 2n + 2$;
        $e \leftarrow |u| + n$;
        if ($b < 1$) then $b \leftarrow 1$;
        if ($e > |u| + |v|$) then $e \leftarrow |u| + |v|$;
        if ($b \geq left.begin$ and $e \leq left.end$) then $leftflag \leftarrow false$;
        else begin
          $first \leftarrow \textbf{maximum}(|u| - 2n + 2, |u| - n + 1 - llspattern[|u| - n])$;
          $last \leftarrow |u| - 2n + 1 + llptext[|u| - n + 1]$;
          if ($last < first$) then there are no new left repetitions of length $2n$.
          else begin
            new left repetitions of length $2n$ begin at $first$ through $last$ in $u$.
            find new primary tandem repeats ranges that are not included in $q$.
            add new primary tandem repeats ranges in $q$.
            /* adjust $left.begin$ and $left.end$. */
            $left.begin \leftarrow$ leftmost position in new primary tandem repeats ranges.
            $left.end \leftarrow$ rightmost position in new primary tandem repeats ranges.
          end
        end
      end
    end
end
```

ALGORITHM **2.** *Procedure to Find All Primary Tandem Repeats in a String w.*

Input : w - string. q - queue of range of primary tandem repeats.
Output : the positions of all primary tandem repeats in string w.
/* *left* : left queue of range of primary tandem repeats.
right : right queue of range of primary tandem repeats. */

```
procedure $findptreps(w, q)$ begin
  if ($|w| \leq 1$) then $w$ is repetition-free
  else begin
```

```
$newptreps(w[1] \cdots w[\lfloor |w|/2 \rfloor], w[\lfloor |w|/2 \rfloor + 1] \cdots w[|w|], q)$;
$leftflag \leftarrow true$;
$rightflag \leftarrow true$;
for $n \leftarrow 1$ to size of $q$ do begin
  $r \leftarrow q[n]$;
  if ($leftflag = true$) then begin
    if ($r.begin = 1$ and $r.end \geq \lfloor |w|/2 \rfloor$) then $leftflag \leftarrow false$;
    else begin
      if (($r.begin \geq 1$ and $r.begin \leq \lfloor |w|/2 \rfloor$) or ($r.end \geq 1$ and $r.end \leq \lfloor |w|/2 \rfloor$)) then begin
        $temp \leftarrow r$;
        if ($temp.end > \lfloor |w|/2 \rfloor$) then $temp.end \leftarrow \lfloor |w|/2 \rfloor$;
        $forflag \leftarrow true$;
        for $k \leftarrow 1$ to size of $left$ do begin
          if ($temp.begin \geq left[k].begin$ and $temp.end \leq left[k].end$) then begin
            $forflag \leftarrow false$;
            break;
          end
          else if ($temp.begin \leq left[k].begin$ and $temp.end \geq left[k].end$) then begin
            pop $left[k]$ in left queue.
            $k \leftarrow k - 1$;
          end
        end
        if ($forflag = true$) then push $temp$ in left queue.
      end
    end
  end
  if ($rightflag = true$) then begin
    if ($r.begin \geq \lfloor |w|/2 \rfloor + 1$ and $r.end = |w|$) then $rightflag \leftarrow false$;
    else begin
      if (($r.begin \geq \lfloor |w|/2 \rfloor + 1$ and $r.begin \leq |w|$) or ($r.end \geq \lfloor |w|/2 \rfloor + 1$ and $r.end \leq |w|$)) then begin
        $temp \leftarrow r$;
        if ($temp.begin < \lfloor |w|/2 \rfloor + 1$) then $temp.begin \leftarrow \lfloor |w|/2 \rfloor + 1$;
        $forflag \leftarrow true$;
        for $k \leftarrow 1$ to size of $right$ do begin
```

```
                    if ($temp.begin \geq right[k].begin$ and $temp.end \leq right[k].end$) then
begin
                      $forflag \leftarrow false$;
                      break;
                    end
                    else if ($temp.begin \leq right[k].begin$ and $temp.end \geq right[k].end$)
then begin
                      pop $right[k]$ in right queue.
                      $k \leftarrow k - 1$;
                    end
                  end
                  if ($forflag = true$) then push $temp$ in right queue.
                end
              end
            end
          end
          if ($leftflag = true$) then $findptreps(w[1] \cdots w[\lfloor |w|/2 \rfloor], left)$;
          if ($rightflag = true$) then $findptreps(w[\lfloor |w|/2 \rfloor + 1] \cdots w[|w|], right)$;
        end
end
```

3. Results and Discussion

Data : human chromosome 5 (the Build 35 finished human genome assembly (hg17, May 2004), http://hgdownload.cse.ucsc.edu/goldenPath/hg17/chromosomes/chr5.fa.gz)

```
Output :
begin:64791, end:64802, length:6
repeat:
ctaacc

begin:64806, end:64817, length:6
repeat:
aaccct

begin:65465, end:65812, length:174
repeat:
gcgccgcgccggcgcaggcgcagagaggcgcgccgcgccggcgcaggcgc
agagaggcgcgccgcgccggcgcaggcgcagagaggcgcgccgcgccggc
gcaggcgcagagaggcgcgccgcgccggcgcaggcgcagagaggcgcgcc
```

```
gcgccggcgcaggcgcagagaggc

begin:180520455, end:180520754, length:150
repeat:
tgggggtggggccaggacgagcatcgtcgttgggggtggggccaggacga
gcatcgtcgttgggggtggggccaggacgagcatcgtcgttgggggtggg
gccaggacgagcatcgtcgttgggggtggggccaggacgagcatcgtcgt

begin:63479, end:64570, length:546
repeat:
cctaaccctaaccctaaccctaaccctaaccctaaccctaaccctaaccc
taaccctaaccctaaccctaaccctaaccctaaccctaaccctaacccta
accctaaccctaaccctaaccctaaccctaaccctaaccctaaccctaac
cctaaccctaaccctaaccctaaccctaaccctaaccctaaccctaaccc
taaccctaaccctaaccctaaccctaaccctaaccctaaccctaacccta
accctaaccctaaccctaaccctaaccctaaccctaaccctaaccctaac
cctaaccctaaccctaaccctaaccctaaccctaaccctaaccctaaccc
taaccctaaccctaaccctaaccctaaccctaaccctaaccctaacccta
accctaaccctaaccctaaccctaaccctaaccctaaccctaaccctaac
cctaaccctaaccctaaccctaaccctaaccctaaccctaaccctaaccc
taaccctaaccctaaccctaaccctaaccctaaccctaaccctaac

begin:3691868, end:3692329, length:231
repeat:
tcttgcactggagacgcccagcatccctgtgtgtcttgcactggagacgc
ccagcatccctgtgtgtcttgcactggagacgcccagcatccctgtgtgt
cttgcactggagacgcccagcatccctgtgtgtcttgcactggagacgcc
cagcatccctgtgtgtcttgcactggagacgcccagcatccctgtgtgtc
ttgcactggagacgcccagcatccctgtgtg
```

References

[1] Michael G. Main and Richard J. Lorentz, "An $O(N \log N)$ Algorithm for Finding All Repetitions in a String", Journal of Algorithms 5, pp. 422-432, 1984.

[2] D. E. Knuth, J. H. Morris, Jr and V. R. Pratt, "Fast pattern matching in strings", SIAM J. Computing, Vol. 6, pp. 323-350, 1977.

School of Electrical Engineering and Computer Science
Hanyang University, Ansan, Korea

School of Electrical Engineering and Computer Science
Hanyang University, Ansan, Korea