Revising Crochemore's Repetitions Algorithm to Compute Runs in a String

Mei Jiang

Department of Computing and Software
McMaster University
March 5th, 2009

Outline

- Introduction
- Some Basic Definitions
  - repeat/repetition
  - run
- *Crochemore*'s Repetitions Algorithm
- Revising *Crochemore*'s Algorithm to Compute Runs
- Work in Progress

Introduction

- A string is a sequence of “letters” (symbols) drawn from some (finite or infinite) “alphabet” (set) [1].

  i.e. a word, a text file, a DNA sequence, etc.
- The stringology is a science of algorithms on strings . There are many areas that utilize the results of the stringology such as information retrieval, DNA processing, etc.
- Repetition problem has been significantly used in many different fields, such as data mining, pattern-matching, data compression, and computational biology, etc.
- In today’s talk, we will be focusing on the algorithm that computes all the repetitions in a string.

Definition - Repeat/Repetition

- Repeat: a collection of identical repeating substrings.
- Repetition: adjacent repeats, no overlap, no spilt.
- Left Extendible (LE), Right Extendible (RE), Non Extendible (NE).

		0	1	2	3	4	5	6	7	8
f	=	a	a	b	a	a	b	a	b	a

$$f = a\,\underbrace{a\,b\,a}_{\text{generator (must be irreducible)}}\,\underbrace{a\,b\,a}\,b\,a$$

- Encoded as (s, l, p)
  - s: starting position of the repetition
  - l: length of the generator, period
  - p: power of the repetition , exponent ($p \geq 2$)

i.e. (0, 1, 2) (0, 3, 2) (1, 3, 2)

Definition - Run

- Introduced by *Main* (1989), also called “Maximal Periodicity” [2].
- Represent repetitions, in a more compact way.
- Computing all the runs specifies all the repetitions in a string.

$$\begin{array}{cccccccccccc} & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ f & = & a & a & a & b & a & a & b & a & a & a \end{array}$$

Non LE   generator   tail (proper prefix of generator)   Non RE

- Encoded as (s, l, p, t)
  - s: starting position of the repetition
  - l: length of the generator, period
  - p: power of the repetition, exponent (p ≥ 2)
  - t: length of the tail

  i.e. (1, 3, 2, 2) is equivalent to (1, 3, 2) (2, 3, 2) (3, 3, 2)

Crochemore's Repetitions Algorithm

- *1981 Crochemore* designed the first $O(n \log n)$ algorithm to compute all the repetitions in a string [3].
- The main ideas of this approach is to successively refine the indices of the string into equivalent classes.
- We define two indices at level l are equivalent if two identical substring of length l start there.
  - i.e. f = abcab $\{0, 3\}_{ab}$ at level 2

Crochemore's Repetitions Algorithm - Example

	0	1	2	3	4	5	6	7	8	9	10
f =	a	b	a	a	b	a	a	b	a	b	a

l=1 $\{0, 2, 3, 5, 6, 8, 10\}_{a}$ $\{1, 4, 7, 9\}_{b}$

l=2 $\{0, 3, 6, 8\}_{ab}$ $\{2, 5\}_{aa}$ $\{10\}_{a\$}$ $\{1, 4, 7, 9\}_{ba}$

l=3 $\{0, 3, 6, 8\}_{aba}$ $\{2, 5\}_{aab}$ $\{1, 4\}_{baa}$ $\{7\}_{bab}$ $\{9\}_{ba\$}$

l=4 $\{0, 3\}_{abaa}$ $\{6\}_{abab}$ $\{8\}_{aba\$}$ $\{2, 5\}_{aaba}$ $\{1, 4\}_{baab}$

l=5 $\{0, 3\}_{abaab}$ $\{2\}_{aabaa}$ $\{5\}_{aabab}$ $\{1, 4\}_{baaba}$

l=6 $\{0, 3\}_{abaaba}$ $\{1\}_{baabaa}$ $\{4\}_{baabab}$

l=7 $\{0\}_{abaabaa}$ $\{3\}_{abaabab}$

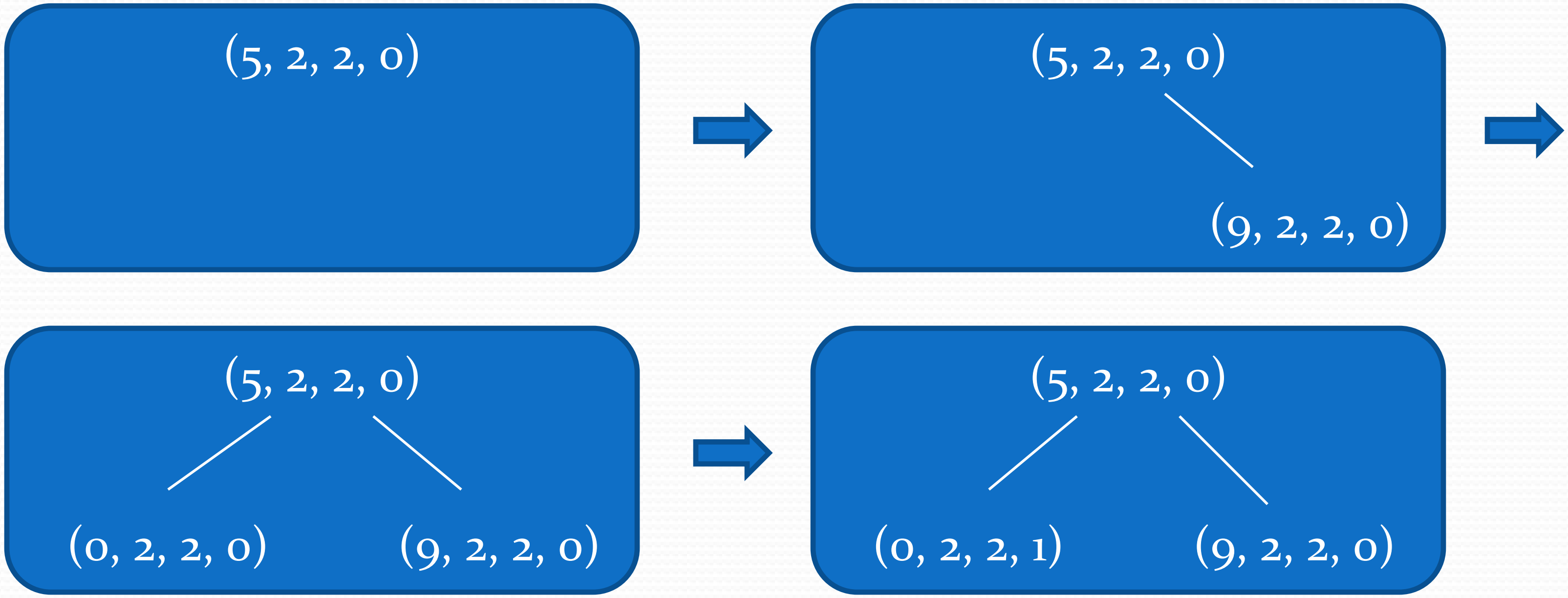
Revising *Crochemore*'s Algorithm to Compute Runs

- The main approach is to combine the repetitions into runs.
- At each level of refinement, we build a binary search tree base on the starting position of the repetitions to collect the runs.
- Every repetition is rewritten in form of run and initialized with tail size of zero.
  - i.e. (0, 3, 2) is equivalent to (0, 3, 2, 0)
- At each level, when a new repetition is computed, we traverse the tree to find a run to join:
  - If find, join the run
  - If not, insert it into the tree

Revising *Crochemore*'s Algorithm to Compute Runs - Example

		0	1	2	3	4	5	6	7	8	9	10	11	12
f	=	c	d	c	d	c	a	b	a	b	c	d	c	d

Repetitions at level 2: (5, 2, 2) (9, 2, 2) (0, 2, 2) (1, 2, 2)



Revising *Crochemore's* Algorithm to Compute Runs - Implementation

	0	1	2	3	4	5	6	7	8	9		n-1
Run_Left				4								
Run_Right				6								
Run_s				7	0		15					
Run_l		0	2	7								
Run_p				2	3		4					
Run_t				1	0		1					

i.e. (7, 2, 2, 1) with left child (0, 2, 3, 0) and right child (15, 2, 4, 1)

Work in Progress

- Implementation of revised algorithm is based on *Franek & Smyth & Xiao's* FSX10 (2003) approach of *Crochemore's* repetitions algorithm [4].
- In 2007 *Chen & Puglisi & Smyth* showed a collection of fast and space efficient algorithms (CPS) to compute runs [5].
- Testing above two algorithms on a set of various strings to get an overview of their performance and possibly memory usage comparison.
  - Testing data includes the sample strings from:
    - DNA, English, Fibonacci, periodic, protein, random

References

1. Bill Smyth, **Computing Patterns in Strings**, *Pearson Addison-Wesley* (2003), 423 pp.
2. Michael G. Main, **Detecting leftmost maximal periodicities**, *Discrete Applied Maths.* 25 (1989) 145–153.
3. Maxime Crochemore, **An optimal algorithm for computing the repetitions in a word**, *Inform. Process. Lett.* 12–5 (1981) 244–250.
4. Frantisek Franek& W. F. Smyth&Xiangdong Xiao, **A note on Crochemore's repetitions algorithm - a fast space-efficient approach**, *Nordic Journal of Computing* 10 (2003) 21–28.
5. Gang Chen& Simon J. Puglisi&W. F. Smyth, **Fast and Practical Algorithms for Computing All the Runs in a String**, *Lecture Notes in Computer Science* (2007) 307 - 315.

Thank you!