

# Repetitions, Runs, Double Squares & Distinct Squares — A Combinatorial Potpourri

Bill Smyth

Algorithms Research Group, Department of Computing & Software  
McMaster University, Hamilton, Canada

School of Engineering & Information Technology,  
Murdoch University, Perth, Australia

Algorithm Design Group, Department of Informatics  
King's College London

email: [smyth@mcmaster.ca](mailto:smyth@mcmaster.ca)

Murdoch University, 12 & 19 September 2014

## Abstract

Mathematicians solve puzzles.

Sometimes the puzzle has no obvious application (Four Colour Theorem, Fermat's Last Theorem), but it turns out that the methodology needed to solve it opens up marvellous new mathematical vistas.

Perhaps more often the puzzle has application to the real world: calculus, linear algebra.

The puzzles discussed here arise out of a simple practical problem: computing the repetitions (tandem repeats) in a string such as

$$x = \underline{\underline{cgccgcccg}}$$

This bit of DNA is only 10 base pairs long, but nevertheless contains six squares, five of them distinct. We will see that computing repetitions takes us into a new combinatorial world.

## Outline

1. Preliminaries: Terminology, Notation & Data Structures
2. Computing Repetitions & Runs
3. The Number of Runs in a String
4. Double Squares
5. Distinct Squares

## Preliminaries I

- ▶ A **string** is a finite sequence of symbols (**letters**) drawn from some finite or infinite set  $\Sigma$  called the **alphabet**. The alphabet **size** is  $\sigma = |\Sigma|$ . Usually the alphabet is **ordered**, thus inducing **lexorder** (dictionary order) on the strings.
- ▶ We write a string  $\mathbf{x}$  in **mathbold**, and we represent it as an array  $\mathbf{x}[1..n]$  for some  $n \geq 0$ . We call  $n = |\mathbf{x}|$  (non-bold) the **length** of  $\mathbf{x}$ . For example,

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \mathbf{x} = & c & g & c & c & g & c & g & c & c & g \end{array} \quad (1)$$

is a string of length  $x = 10$  on  $\Sigma = \{c, g\}$ . For  $x = 0$ ,  $\mathbf{x} = \epsilon$ , the **empty string**.

- ▶ If  $\mathbf{x} = \mathbf{uvw}$ , then  $\mathbf{u}$  is said to be a **prefix**,  $\mathbf{v}$  a **substring** and  $\mathbf{w}$  a **suffix** of  $\mathbf{x}$ ; if  $\mathbf{vw} \neq \epsilon$ ,  $\mathbf{uw} \neq \epsilon$ ,  $\mathbf{uv} \neq \epsilon$ , respectively, then  $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$  is, respectively, a **proper prefix**, **proper substring**, **proper suffix** of  $\mathbf{x}$ .
- ▶ If  $\mathbf{x} = \mathbf{x}[1..n]$  and there exists an integer  $\pi$  such that  $\mathbf{x}[i] = \mathbf{x}[i+\pi]$  for every  $i \in 1..n-\pi$ , then  $\mathbf{x}$  is said to have **period**  $\pi$  and **generator**  $\mathbf{x}[1..\pi]$ . The string (1) has periods 5 and 8 with corresponding generators  $cgccg$  and  $cgccgcg$ , respectively.

## Preliminaries II

- ▶ If  $\mathbf{x} = \mathbf{v}\mathbf{u}^e\mathbf{w}$ , with integer  $e > 1$  and  $\mathbf{u}$  neither a suffix of  $\mathbf{v}$  nor a prefix of  $\mathbf{w}$  ( $e$  is maximum), then  $\mathbf{u}^e$  is a **repetition** in  $\mathbf{x}$ .  $u$  and  $e$  are the **period** and **exponent**, respectively, of the repetition. A string not a repetition is **primitive**. We suppose that  $\mathbf{u}$  itself is primitive (the period  $u$  is minimum).
- ▶ For example, in

$$\mathbf{x} = \underline{\underline{cgccgcgccg}}, \quad (2)$$

there are repetitions  $c^2$  (twice),  $(cg)^2$  and  $(gc)^2$ ,  $(cgc)^2$ , and  $(cgccg)^2$ . Each of these repetitions is a **square** ( $e = 2$ ). In general, every repetition has a square prefix.

- ▶ If  $\mathbf{v} = \mathbf{x}[i..j]$  has minimum period  $u$ , where  $v/u \geq 2$ , and if neither  $\mathbf{x}[i-1..j]$  nor  $\mathbf{x}[i..j+1]$  (whenever defined) has period  $u$ , then  $\mathbf{x}$  is said to be a **maximal periodicity** or **run** in  $\mathbf{x}$  [M89] and  $\mathbf{v}$  is said to have **exponent**  $e = \lfloor v/u \rfloor$  and **tail**  $t = v \bmod u$ . When  $t = 0$ , the run is also a repetition.
- ▶ All of the repetitions in (2) are runs except for  $(cg)^2$  and  $(gc)^2$ : these are prefix and suffix, respectively, of the run  $\mathbf{v} = cgccg$ .
- ▶ In general, every repetition is a substring of some run; thus computing all the runs **implicitly** computes all the repetitions.

## Global Data Structures: ST, SA, LCP (Sorted Suffixes)

	1	2	3	4	5	6	7	8
$x =$	$c$	$g$	$c$	$c$	$g$	$c$	$g$	$c$
$SA_x =$	8	3	6	1	4	7	2	5
$LCP_x =$	0	1	1	3	3	0	2	2

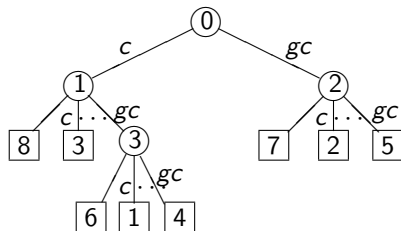


Figure : Suffix Tree, Suffix Array, LCP array

## Global Data Structures: LZ

A factorization  $\mathbf{x} = \mathbf{w}_1\mathbf{w}_2 \cdots \mathbf{w}_k$  is **LZ** (for Lempel-Ziv [LZ76, ZL77]) if each factor (substring)  $\mathbf{w}_j$  is either a letter that has not occurred previously in  $\mathbf{x}$ , or else the longest factor that **has** occurred previously. For the string

$$\mathbf{x} = cgcgcgccg$$

the factorization  $\text{LZ}_{\mathbf{x}}$  is given by  $\mathbf{w}_1 = c$ ,  $\mathbf{w}_2 = g$ ,  $\mathbf{w}_3 = c$ ,  $\mathbf{w}_4 = cgc$ ,  $\mathbf{w}_5 = gccg$ .

All of these data structures can be computed in time linear in  $x$ :

ST [W73, M76, U95, F97],  
 SA [MM90, MM93, PST07, NZC09, M09],  
 LCP [KLAAP01, M04, PT08, KMP09].

And LZ can be computed from some combination of them: ...

## Computing LZ

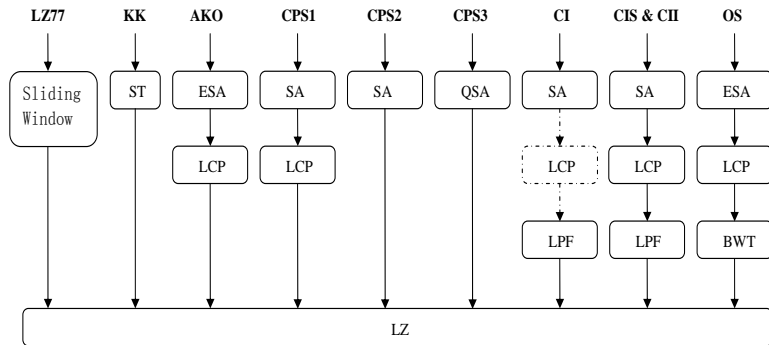


Figure : From [ACIKSTY13]



## Computing Repetitions

In the early 1980s three  $O(x \log x)$ -time algorithms were proposed to compute all the repetitions in a given string  $x$ :

- ▶ Crochemore's method [C81] is essentially an algorithm for suffix tree construction [S03]. He also showed that a string  $x$  can contain as many as  $O(x \log x)$  repetitions — thus all these algorithms are **optimal**.
- ▶ Apostolico & Preparata [AP83] use suffix trees plus auxiliary data structures.
- ▶ Main & Lorentz [ML84] use a **divide-and-conquer** approach based on prior computation of  $LZ_x$ .

\*\*\* ALL USE GLOBAL DATA STRUCTURES \*\*\*

\*\*\* ALL DEPEND ON ORDERING SUFFIXES \*\*\*

## Computing Runs

- ▶ In 1989 Main [M89] showed how to compute all “leftmost” runs, again from  $LZ_x$ , in **linear** time.
- ▶ In 1999 Kolpakov & Kucherov [KK99, KK00] showed how to compute all runs from the leftmost ones, also in **linear** time. To establish linearity, they proved that the maximum number  $\rho(n)$  of runs over all strings of length  $n$  satisfies

$$\rho(n) \leq k_1 n - k_2 \sqrt{n} \log_2 n \quad (3)$$

for some universal positive constants  $k_1$  and  $k_2$ .

- ▶ More recent methods still use suffix arrays [CPS07].

\*\*\* GLOBAL DATA STRUCTURES, LINEAR OUTPUT \*\*\*  
\*\*\* ALGORITHM DEPENDS ON ORDERING SUFFIXES \*\*\*

## BUT ...

- ▶ Puglisi & Simpson [PS08] show that the **expected** number of runs in a string of length  $n$  is **small**:
  - ▶  $0.41n$  runs for alphabet size  $\sigma = 2$ ;
  - ▶  $0.25n$  runs for DNA ( $\Sigma = \{a, c, g, t\}$ );
  - ▶  $0.04n$  for protein ( $\sigma = 20$ );
  - ▶  $0.01n$  for English-language text.

Runs (hence repetitions) in most strings are **sparse**!

- ▶ Runs are generally a **local** phenomenon: why do we need global data structures to compute them?
- ▶ The computation of runs requires no definition of order in strings or substrings — yet the Suffix Tree/Suffix Array data structures depend on an ordering of the suffixes of a string.

Why can't we compute runs more easily???

## Combinatorial Insight vs. Brute Force

- ▶ Using full data structures, linear-time processing (with a high constant of proportionality) is achieved using 12 bytes of storage per input symbol: 36 gigabytes for the human genome (3GB long) [ACIKSTY13].
- ▶ Using compressed data structures, space can be reduced to about 5 bytes per input symbol (15GB for the human genome), but linearity is lost (order of magnitude slower) [ACIKSTY13].
- ▶ **Maybe** this is acceptable using current main memory capacity, but what about processing plant DNA (15-20GB) or 50GB/500GB strings? Only possible using secondary storage and slowing the computation by **several** orders of magnitude.
- ▶ All this to compute something that is generally **sparse** and occurs **locally** in the string: we want to use at most **one** byte per input symbol (only two bits for DNA!) and process an order of magnitude faster.

So we need to understand **why** the maximal periodicities are restricted — **why** there are restrictions on their overlaps — so that we can process a string from left to right in a controlled way, outputting the runs as we go: we need **combinatorial insight!**

## An Idea

If  $\rho(n)/n$  is limited to be near one, it means that on average there is about one run starting at each position. So ... if **TWO** runs start at some position, then there must be some other position, probably nearby, at which **NO** runs start.

Runs always start with squares — what do we know about squares that begin at about the same position? What **COMBINATORIAL INSIGHT** do we have into the restrictions that might be imposed upon occurrences of overlapping squares? Until recently, very little:

## What We Knew (After 90 Years of Stringology)

### Definition

If  $\mathbf{x} = \mathbf{v}^2$  has a proper prefix  $\mathbf{u}^2$ ,  $u < v < 2u$ , we say that  $\mathbf{x}$  is a *double square* and write it  $\mathbf{x} = DS(\mathbf{u}, \mathbf{v})$ .

### Lemma (Three Squares Lemma [CR95])

If  $\mathbf{u}$  is not a repetition and  $DS(\mathbf{u}, \mathbf{v})$  is a proper prefix of  $\mathbf{w}^2$ , then  $w \geq u + v$ .

The Fibonacci string demonstrates that this result is best possible (squares ending at positions 6, 10,  $16 = 6 + 10$ ,  $26 = 10 + 16$ ):

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26  
 $\mathbf{x} = c \ g \ c \ c \ g \ \underline{c} \ g \ c \ c \ \underline{g} \ c \ c \ g \ c \ g \ \underline{c} \ c \ g \ c \ g \ c \ c \ g \ c \ c \ \underline{g}$

Maybe this result is not so surprising. To identify squares and runs, we need to know much more.

## Figuring Out Double Squares

There have been six identifiable approaches taken to the analysis and understanding of double squares:

- ▶ DS I: Bounds on  $\rho(n)$
- ▶ DS II: The New Periodicity Lemma
- ▶ DS III: Combinatorial Restrictions on the Third Square
- ▶ DS IV: The “General Case”
- ▶ DS V: Inversion Factors & the New NPL
- ▶ DS VI: The Magical L-Root

All are interesting and valuable. We look at each one in turn.

## DS I: Bounds on $\rho(n)$ — Upper

- ▶ Kolpakov & Kucherov [KK00]:  $\rho(n) \leq k_1 n - k_2 \sqrt{n} \log_2 n$  for universal positive constants  $k_1$  and  $k_2$ , but without upper bounds on  $k_1$  and  $k_2$ . Linear, but how linear?
- ▶ Rytter [R06]: divided runs into those of “large” and “small” periods, thus showed that  $\rho(n) \leq 5.0n$ .
- ▶ Using similar methods, Puglisi, Simpson & Smyth [PSS08] and Crochemore & Ilie [CI08] reduced the upper bound to  $3.48n$  and  $1.60n$ , respectively.
- ▶ Giraud [G08, G09] showed that  $\lim_{n \rightarrow \infty} \rho(n)/n$  exists, is approached from below, and never attained, while proving  $\rho(n) \leq 1.49n$ .
- ▶ Finally, Crochemore, Ilie & Tinta [CIT11] applied three years of CPU time on a supercomputer to the results of [CI08], yielding  $\rho(n) \leq 1.029n$ .

So far, lots of computation, not so much combinatorial insight.



## DS I: Bounds on $\rho(n)$ — Lower

- ▶ Franek, Simpson & Smyth [FSS03] describe an infinite sequence of strings of increasing lengths  $n_1, n_2, \dots$  such that  $\lim_{i \rightarrow \infty} r(n_i)/n_i = 3/2\phi = 0.927\dots$ , where  $\phi$  is the golden mean and  $r(n_i)$  is the number of runs in the string of length  $n_i$ . Later Franek & Yang [FY08] show that indeed  $\rho(n) \geq 3n/2\phi$ .
- ▶ Matsubara *et al.* [MKIBS08] use simple arguments together with some computation to show that  $\rho(n) \geq 0.9445654n$ .
- ▶ Simpson [S10] uses a construction based on modified Padovan words to show that  $\rho(n) \geq 0.94457n$ .

Here the methods used are based more on number theory than on a detailed understanding of the combinatorial consequences of overlapping squares.

\*\*\* WE WILL HAVE MORE TO SAY ABOUT THIS LATER! \*\*\*

## DS II — NPL

In 2005-2006 the collective efforts of Fan, Puglisi, Simpson, Smyth & Turpin yielded a lemma that imposed constraints on the squares that could occur to the right of a double square:

**Lemma (New Periodicity Lemma [FSS05, PST05, FPST06])**

*Let  $\mathbf{x} = DS(\mathbf{u}, \mathbf{v})$ , where  $\mathbf{u}$  has no square prefix and  $\mathbf{v}$  is not a repetition. Then for all integers  $k$  and  $w$  such that  $0 \leq k < v - u < w < v$  and  $w \neq u$ ,  $\mathbf{x}[k+1..k+2w]$  is not a square.*

But there were serious drawbacks:

- ▶ The conditions imposed on  $\mathbf{u}$  (particularly) and  $\mathbf{v}$  were undesirable: for example, the NPL did not apply to any  $\mathbf{u} = cc \cdots$  or  $cgcg \cdots$ , thus omitting a great many strings.
- ▶ The proof broke down into FOURTEEN messy subcases defined by the sizes of  $k$  and  $w$  relative to  $u$  and  $v$ : ...

## DS II — Subcases 6 & 8

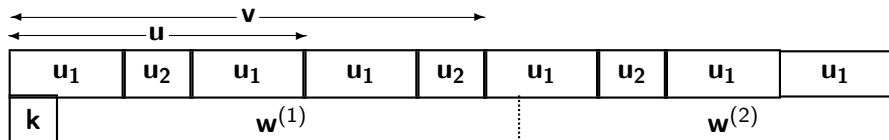


Figure : Subcase 6

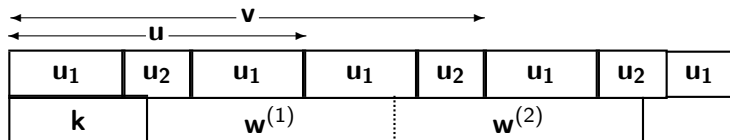


Figure : Subcase 8

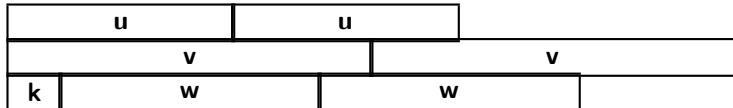
## DS II: The Dreaded Fourteen

Subcase S	k	k+w	k+2w	Special Conditions
1	$0 \leq k \leq u_1$	$k+w \leq u$	$k+2w \leq u+u_1$	$k \geq u_2$
2	$0 \leq k \leq u_1$	$k+w \leq u$	$k+2w \leq u+u_1$	$k < u_2$
3	$0 \leq k \leq u_1$	$k+w \leq u$	$k+2w > u+u_1$	—
4	$0 \leq k \leq u_1$	$u < k+w \leq u+u_1$	—	—
5	$0 \leq k \leq u_1$	$u+u_1 < k+w \leq v$	—	—
6	$0 \leq k \leq u_1$	$v < k+w < 2u$	—	—
7	$u_1 < k < u_1+u_2$	$k+w \leq u+u_1$	$k+2w \leq 2u$	—
8	$u_1 < k < u_1+u_2$	$k+w \leq u+u_1$	$k+2w > 2u$	—
9	$u_1 < k < u_1+u_2$	$u+u_1 < k+w \leq v$	—	$w < u$
10	$u_1 < k < u_1+u_2$	$k+w \leq v$	$k+2w \leq u+v$	$w > u$
11	$u_1 < k < u_1+u_2$	$k+w \leq v$	$u+v < k+2w \leq 2v - u_2$	—
12	$u_1 < k < u_1+u_2$	$k+w \leq v$	$2v - u_2 < k+2w$	—
13	$u_1 < k < u_1+u_2$	$v < k+w \leq 2u$	—	—
14	$u_1 < k < u_1+u_2$	$2u < k+w < 2u+u_2-1$	—	—

Unfortunately the proofs of the 14 cases turned out to be essentially different: it took a lot of work to prove an NPL that was not quite what one wanted ...

## DS III: The Third Square

In an effort to make statements about double squares that were both more precise and more general in their application, a series of papers by Franek, Simpson, Smyth and their students [S07, KS12, FFSS12, BS14] has yielded a characterization of the permissible values of  $w$  for each of the 14 subcases:



It turned out to be useful to consider two main cases:

- (C1)  $u < v \leq 3u/2$  (the easy case: no bounds on  $k$  and  $w$  required)
- (C2)  $3u/2 < v < 2u$  with  $v - u < w < v$ ,  $w \neq u$  (the same 14 subcases!).

## DS III: Case (C1)

### Lemma ([KS12])

For  $DS(\mathbf{u}, \mathbf{v})$  in case (C1):

- (a)  $DS(\mathbf{u}, \mathbf{v}) = \mathbf{u}_1^m \mathbf{u}_2 \mathbf{u}_1^{m+1} \mathbf{u}_2 \mathbf{u}_1$ , where  
 $u_1 = v - u \leq u/2$ ,  $u_2 = u \bmod u_1$ ,  $m = \lfloor u/u_1 \rfloor$ , and  $\mathbf{u}_2$  is a proper prefix of  $\mathbf{u}_1$ ;
- (b)  $DS(\mathbf{u}, \mathbf{v})$  contains no runs of period  $\pi \geq u_1$  other than exactly  $m + 5$  known specified runs.

Thus for (C1) the breakdown of  $DS(\mathbf{u}, \mathbf{v})$  into runs of small period can be established in a uniform way, independent of any subcases. Not so however for (C2): ...

## DS III: Case (C2)

**Table :** Structure of  $\mathbf{x}$  for subcases  $S \in 1..14$ :  $\sigma$  is the largest alphabet size consistent with  $u, v, k, w$  [FFSS12];  $\mathbf{d}$ ,  $\mathbf{d}_1$  and  $\mathbf{d}_3$  are prefixes of  $\mathbf{x}$  with  $d = \gcd(u, v, w)$ ,  $d_1 = \gcd(u-w, v-u)$ ,  $d_2 = \gcd(u, v-w)$ ,  $d_3 = v \bmod d_2$ .

Subcases $S$	Conditions	Breakdown of $\mathbf{x}$
1, 2, 5, 6, 8–10	$(\forall \mathbf{x}, \sigma = d)$	$\mathbf{x} = \mathbf{d}^{x/d}$
3, 4, 7	$(\forall \mathbf{x})$ specified cases	$\mathbf{x} = \mathbf{d}_1^{u/d_1} \mathbf{d}_1^{v/d_1} \mathbf{d}_1^{(v-u)/d_1}$ $\mathbf{x} = \mathbf{d}^{x/d}$
11–14	$\sigma = d$ or $d_2 \leq 2u - v$ otherwise	$\mathbf{x} = \mathbf{d}^{x/d}$ $\mathbf{x} = ((\mathbf{d}_3^{d_2/d_3})^{v/d_2})^2$

## DS III: Case (C2)

### Lemma

*Suppose that in  $\mathbf{x} = DS(\mathbf{u}, \mathbf{v})$ ,  $3u/2 < v < 2u$ ,  $\mathbf{w}^2$  occurs at  $\mathbf{x}[k+1]$ , where  $0 \leq k < v-u < w < v$ ,  $w \neq u$ . Then for each of the Dreaded Fourteen, the corresponding structure of  $\mathbf{x}$  is given in the above table.*

Note:

- ▶ The constraints on  $\mathbf{u}$  and  $\mathbf{v}$  are gone!
- ▶ In every case the assumption that  $\mathbf{w}^2$  exists forces a breakdown into runs of small period, whose generator ( $\mathbf{d}$ ,  $\mathbf{d}_1$  or  $\mathbf{d}_3$ ) is a prefix of  $\mathbf{x}$ ; in all but a few instances (subsubcases of 3,4,7),  $\mathbf{x}$  is a single repetition of small period.
- ▶ This is Structure! What do we do with it?

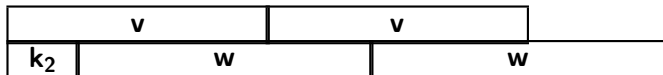


## DS IV: The General Case

To describe all cases that arise from three overlapping squares, we need to put together



and



Such a lemma has been stated and proved [BS14]; it was used to establish the final remaining subcases (3 & 7) of (C2), but its general usefulness is not clear.

## DS V: Inversion Factors & the New NPL

An entirely different approach is taken in [BFS14]: a sequence of simple lemmas is used to show that certain substrings (“inversion factors”) must occur, exactly twice and at predictable locations, within the range of a double square.

### Definition

Given  $\mathbf{x} = \mathbf{x}[1..n]$  and an integer  $j \in 0..n-1$ , the string  $R_j(\mathbf{x}) = \mathbf{x}[j+1..n]\mathbf{x}[1..j]$  is called the  $j^{\text{th}}$  *rotation* of  $\mathbf{x}$ .  $\mathbf{x}$  and  $R_j(\mathbf{x})$  are said to be *conjugate*, written  $\mathbf{x} \sim R_j(\mathbf{x})$ . (For example, *cgat* is conjugate to *gatc*, *atcg*, *tcga* and itself.)

### Lemma (Equal Conjugate Rotations $\iff$ Repetition)

[S03, Lemma 1.4.2] Let  $\mathbf{x}$  be a string of length  $n$  and minimum period  $\pi \leq n$ , and let  $j \in 1..n-1$  be an integer. Then  $R_j(\mathbf{x}) = \mathbf{x}$  if and only if  $\mathbf{x}$  is a repetition ( $\pi < n$ ,  $\pi \mid n$ ) and  $j \mid \pi$ .

## DS V: Basic Lemmas

The repeated application of these two simple structural lemmas is the key to identifying a “canonical form” for double squares:

### Lemma (Synchronization Principle)

*The primitive string  $\mathbf{x}$  occurs exactly  $p$  times in  $\mathbf{x}_2\mathbf{x}^p\mathbf{x}_1$ , where  $p$  is a nonnegative integer and  $\mathbf{x}_1$  (respectively,  $\mathbf{x}_2$ ) is a proper prefix (respectively, proper suffix) of  $\mathbf{x}$ .*

### Lemma (Common Factor Lemma)

*Suppose that  $\mathbf{x}$  and  $\mathbf{y}$  are primitive strings, where  $\mathbf{x}_1$  (respectively,  $\mathbf{y}_1$ ) is a proper prefix and  $\mathbf{x}_2$  (respectively,  $\mathbf{y}_2$ ) a proper suffix of  $\mathbf{x}$  (respectively,  $\mathbf{y}$ ). If for integers  $p$  and  $q$ ,  $\mathbf{x}_2\mathbf{x}^p\mathbf{x}_1$  and  $\mathbf{y}_2\mathbf{y}^q\mathbf{y}_1$  have a common factor of length  $|\mathbf{x}|+|\mathbf{y}|$ , then  $\mathbf{x} \sim \mathbf{y}$ .*

## DS V: A Unique Canonical Form for Double Squares

### Lemma (Two Squares Factorization Lemma)

If  $\mathbf{x} = DS(\mathbf{u}, \mathbf{v})$ , there exists a unique primitive string  $\mathbf{u}_1$  such that  $\mathbf{u} = \mathbf{u}_1^{e_1} \mathbf{u}_2$  and  $\mathbf{v} = \mathbf{u}_1^{e_1} \mathbf{u}_2 \mathbf{u}_1^{e_2}$ , where  $\mathbf{u}_2$  is a possibly empty proper prefix of  $\mathbf{u}_1$  and  $e_1, e_2$  are integers such that  $e_1 \geq e_2 \geq 1$ .

Moreover,

- (a) if  $|\mathbf{u}_2| = 0$ , then  $e_1 > e_2$  (thus  $e_1 \geq 2$ );
- (b) if  $|\mathbf{u}_2| > 0$ , then  $\mathbf{v}$  is primitive, and if in addition  $e_1 \geq 2$ , then  $\mathbf{u}$  also is primitive.

In both cases, the factorization is unique, and so we write the *canonical factorization* as

$$DS(\mathbf{u}, \mathbf{v}) = (\mathbf{u}_1, \mathbf{u}_2, e_1, e_2) = \mathbf{u}_1^{e_1} \mathbf{u}_2 \mathbf{u}_1^{e_1+e_2} \mathbf{u}_2 \mathbf{u}_1^{e_2}.$$

## DS V: Occurrences of the “Inversion Factor” IF

### Definition

Let  $\mathbf{u}_1 = \mathbf{u}_2\overline{\mathbf{u}_2}$ . Then

$$DS(\mathbf{u}, \mathbf{v}) = \mathbf{u}_2(\overline{\mathbf{u}_2}\mathbf{u}_2)^{e_1-1}(IF)\mathbf{u}_2(\overline{\mathbf{u}_2}\mathbf{u}_2)^{e_1+e_2-2}(IF)(\overline{\mathbf{u}_2}\mathbf{u}_2)^{e_2-1}, \quad (4)$$

where  $IF = \overline{\mathbf{u}_2}\mathbf{u}_2\mathbf{u}_2\overline{\mathbf{u}_2} = R_{u_2}(\mathbf{u}_1)\mathbf{u}_1$  is called the *inversion factor*.

### Lemma

Suppose  $\mathbf{x} = DS(\mathbf{u}, \mathbf{v}) = (\mathbf{u}_1, \mathbf{u}_2, e_1, e_2)$ . Then the inversion factor  $IF$  and specified rotations of  $IF$  occur as shown in (4), exactly  $v$  positions apart, and nowhere else in  $\mathbf{x}$ .

These structural lemmas yield a relatively straightforward proof of a New NPL that requires no conditions on  $\mathbf{u}$  and  $\mathbf{v}$  and no bounds on  $k$ ; the only resulting restriction is that it says nothing about values  $w \in v-u+1..u-1$ : ...

## DS V: Old & New NPL

### Lemma (Old NPL)

Let  $\mathbf{x} = DS(\mathbf{u}, \mathbf{v})$ , where  $\mathbf{u}$  has no square prefix and  $\mathbf{v}$  is not a repetition. Then for all integers  $k$  and  $w$  such that  $0 \leq k < v - u < w < v$  and  $w \neq u$ ,  $\mathbf{x}[k+1..k+2w]$  is not a square.

### Lemma (New NPL)

Consider a double square  $DS(\mathbf{u}, \mathbf{v}) = (\mathbf{u}_1, \mathbf{u}_2, e_1, e_2)$ . If  $\mathbf{w}^2$  is a proper substring of  $\mathbf{v}^2$ , then either

- (a)  $w < u$ , or
- (b)  $u \leq w < v$  and the smallest generator of  $\mathbf{w}$  is a conjugate of  $\mathbf{u}_1$ .

## DS VI: The Magical L-Root

It took only a page [BIINTT14] for six Japanese mathematicians

- ▶ Bannai
- ▶ I (I kid you not)
- ▶ Inenaga
- ▶ Nakashima
- ▶ Takeda
- ▶ Tsuruta

to show that  $\rho(n) \leq n-1$ .

Well, at least on a two-letter alphabet  $\Sigma = \{c, g\}$ .

## DS VI: Lyndon Word

### Definition

Consider the two orderings of  $\Sigma = \{c, g\}$ :

- ▶  $F$  (Forward):  $c < g$
- ▶  $B$  (Backward):  $g < c$

and the associated lexicographic (dictionary) orderings  $F$  and  $B$  of strings  $\mathbf{x}$  on  $\Sigma$ . Then a primitive string  $\mathbf{x}$  on  $\Sigma$  is a **Lyndon word**  $L_F$  (respectively,  $L_B$ ) if it is the (unique) least in  $F$ -order (respectively,  $B$ -order) over all rotations  $R_j(\mathbf{x})$ ,  $1 \leq j \leq n-1$ .

For example,  $\mathbf{x} = ccg$  is  $L_F$ ,  $\mathbf{y} = gcc$  is  $L_B$ ,  $\mathbf{z} = cgc$  is not a Lyndon word.



## DS VI: F-Root & B-Root

### Definition

The *F-root* (respectively, *B-root*) of a run in  $\mathbf{x}$  is the position in  $\mathbf{x}$  of the Lyndon word  $L_F$  (respectively,  $L_B$ ) that is conjugate to the (primitive) generator of the run and leftmost in the run, except not the run's first position.

$$\begin{array}{cccccccccc}
 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 \mathbf{x} = & c & g & c & c & g & c & g & c & c & g \\
 & & & & & B & F & & & & 
 \end{array} \tag{5}$$

The F-root of run  $cgccg$  in  $\mathbf{x} = (cgccg)^2$  is position 6, the B-root is position 5.

## DS VI: L-Root

### Definition

Suppose that a sentinel letter  $\$ > g > c$  is appended to  $\mathbf{x}$ . Then the *L-root* of a run in  $\mathbf{x}$  is the *F-root* if the run is followed by  $c$ , the *B-root* otherwise.

$$\begin{array}{cccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
 \mathbf{x} = & c & g & c & c & g & c & g & c & c & g & \$ \\
 & & & & & B & F & & & \uparrow & & \\
 & & & & & & L & & & & & 
 \end{array} \tag{6}$$

### Lemma

The *L-roots* of the runs in  $\mathbf{x}$  are distinct!

### Corollary

$$\rho(n) \leq n - 1.$$

## DS VI: L-Root Example

The runs in  $\mathbf{x} = (\mathit{cgccg})^2$  are

- ▶  $\mathit{cc}$  (twice, period 1)
- ▶  $\mathit{cgcgc}$  (period 2)
- ▶  $(\mathit{cgc})^2$  (period 3)
- ▶  $(\mathit{cgccg})^2$  (period 5)

$$\begin{array}{cccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
 \mathbf{x} = & c & g & c & c & g & c & g & c & c & g & \$ \\
 \text{periods} = & & 3 & & 1 & 5 & 2 & & & 1 & & 
 \end{array} \tag{7}$$

Hey presto!

(And somehow the maximum number of runs is intimately connected with the **ordering** of the letters!?!)

## Distinct Squares I

Recall there can be  $O(n \log n)$  repetitions in  $\mathbf{x}$  (in a Fibostring, for example), while  $\mathbf{x}^n$  has  $O(n^2)$  squares. How many **distinct** squares can there be? Note  $\mathbf{x} = (cgccg)^2$  contains six squares, of which five are distinct:

$$c^2, (cg)^2, (gc)^2, (cgc)^2, (cgccg)^2,$$

even though it contains only four runs.

Let  $\delta(n)$  denote the maximum number of distinct squares in any string of length  $n$ . Fraenkel & Simpson showed that no position in  $\mathbf{x}$  could be the start position of more than two rightmost occurrences of squares, hence:

### Lemma

$$[FS98] \delta(n) < 2n.$$

## Distinct Squares II

In 2007 Ilie showed

### Lemma

$$[I07] \delta(n) \leq 2n - \Theta(\log n).$$

Recently the “double square” methods used to prove the New NPL have been applied to distinct squares (in a 29-page paper). The following is claimed:

### Lemma

$$[DFT13] \delta(n) < \lfloor 11n/6 \rfloor.$$

Everybody believes the true upper bound is less than  $n$  — why is it so hard to prove?

Perhaps another new combinatorial world awaits us!



Anisa Al-Hafeedh, Maxime Crochemore, Lucian Ilie, Evguenia Kopylova, W. F. Smyth, German Tischler & Munina Yusufu, **A comparison of index-based Lempel-Ziv LZ77 factorization algorithms**, *ACM Computing Surveys*45–1 (2013) 5:1–5:17.



Alberto Apostolico & Franco P. Preparata, **Optimal off-line detection of repetitions in a string**, *Theoret. Comput. Sci.* 22 (1983) 297–315.



Haoyue Bai, Frantisek Franek & W. F. Smyth, **The New Periodicity Lemma Revisited**, *Discrete Applied Math.*, submitted for publication (2014).



Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda & Kazuya Tsuruta, **The “runs” theorem**, <http://arxiv.org/abs/1406.0263> (2014).



Widmer Bland & W. F. Smyth, **Three Overlapping Squares: The General Case Characterized & Applications**, *Theoret. Comput. Sci.* , submitted for publication (2014).



Gang Chen, Simon J. Puglisi & W. F. Smyth, **Fast & practical algorithms for computing all the runs in a string**, *Proc. 18th Annual Symp. Combinatorial Pattern Matching*, B. Ma & K. Zhang (eds.), Springer Lecture Notes in Computer Science, LNCS 4580, Springer-Verlag (2007) 307–315.



Maxime Crochemore, **An optimal algorithm for computing all the repetitions in a word**, *Inform. Process. Lett.* 12–5 (1981) 244–248.



Maxime Crochemore & Lucian Ilie, **Maximal repetitions in strings**, *J. Comput. Sys. Sci.* (2008) 796–807.



Maxime Crochemore, Lucian Ilie & Liviu Tinta, **The “runs” conjecture**, *Theoret. Comput. Sci.* 412 (2011) 2931–2941.



Maxime Crochemore and Wojciech Rytter, **Squares, cubes, and time-space efficient strings searching**, *Algorithmica* 13 (1995) 405–425.



Antoine Deza, Frantisek Franek & Adrien Thierry, **How many double squares can a string contain?**, <http://arxiv.org/abs/1310.3429> (2014).



Kangmin Fan, Simon J. Puglisi, W. F. Smyth & Andrew Turpin, **A new periodicity lemma**, *SIAM J. Discrete Math.* 20–3 (2006) 656–668.



Kangmin Fan, R. J. Simpson & W. F. Smyth, **A new periodicity lemma** (preliminary version), *Proc. 16th Annual Symp. Combinatorial Pattern Matching*, Springer Lecture Notes in Computer Science, LNCS 3537, Springer-Verlag (2005) 257–265.



Martin Farach, **Optimal suffix tree construction with large alphabets**, *Proc. 38th IEEE Symp. Found. Computer Science*, IEEE Computer Society (1997) 137–143.



Aviezri S. Fraenkel & Jamie Simpson, **How many squares can a string contain?**, *J. Combinatorial Theory, Series A*82–1 (1998) 112–120.



Frantisek Franek, Robert C. G. Fuller, Jamie Simpson & W. F. Smyth, **More results on overlapping squares**, *J. Discrete Algorithms* 17 (2012) 2–8.



Frantisek Franek, R. J. Simpson & W. F. Smyth, **The maximum number of runs in a string**, *Proc. 14th Australasian Workshop on Combinatorial Algs.*, Mirka Miller & Kunsoo Park (eds.) (2003) 26–35.



Frantisek Franek & Q. Yang, **An asymptotic lower bound for the maximal number of runs in a string**, *Internat. J. Foundations of Computer Science*1 (2008) 195–203.



Mathieu Giraud, **Not so many runs in strings**, *Proc. 2nd Internat. Conf. on Language & Automata Theory & Applications*, Carlos Martín-Vide, Friedrich Otto & Henning Fernau (eds.), Springer Lecture Notes in Computer Science, LNCS 5196, Springer-Verlag (2008) 232–239.



Mathieu Giraud, **Asymptotic behavior of the numbers of runs and microruns**, *Inform. & Computation* 207–11 (2009) 1221–1228.



Lucian Ilie, **A note on the number of squares in a word**, *Theoret. Comput. Sci.* 380–3 (2007) 373–376.





Juha Kärkkäinen, Giovanni Manzini & Simon J. Puglisi, **Permuted longest-common-prefix array**, *Proc. 20th Annual Symp. Combinatorial Pattern Matching*, Gregory Kucherov & Esko Ukkonen (eds.), Springer Lecture Notes in Computer Science, LNCS 5577, Springer Verlag (2009) 181–192.



Toru Kasai, Gunho Lee, Hiroki Akimura, Setsuo Arikawa & Kunsoo Park, **Linear-time longest-common-prefix computation in suffix arrays and its applications**, *Proc. 12th Annual Symp. Combinatorial Pattern Matching*, Amihoud Amir & Gad M. Landau (eds.), Springer Lecture Notes in Computer Science, LNCS 2089, Springer-Verlag (2001) 181–192.



Roman Kolpakov & Gregory Kucherov, **Finding maximal repetitions in a word in linear time**, *Proc. 40th Annual IEEE Symp. Found. Computer Science* (1999) 596–604.



Roman Kolpakov & Gregory Kucherov, **On maximal repetitions in words**, *J. Discrete Algorithms 1* (2000) 159–186.



Evguenia Kopylova & W. F. Smyth, **The three squares lemma revisited**, *J. Discrete Algorithms 11* (2012) 3–14.



Abraham Lempel & Jacob Ziv, **On the complexity of finite sequences**, *IEEE Trans. Information Theory 22* (1976) 75–81.



Michael G. Main, **Detecting leftmost maximal periodicities**, *Discrete Applied Maths. 25* (1989) 145–153.



Michael G. Main & Richard J. Lorentz, **An  $O(n \log n)$  algorithm for finding all repetitions in a string**, *J. Algorithms* 5 (1984) 422–432.



Udi Manber & Gene W. Myers, **Suffix array: a new method for on-line string searches**, *Proc. First Annual ACM-SIAM Symp. Discrete Algs.* (1990) 319-327.



Udi Manber & Gene W. Myers, **Suffix array: a new method for on-line string searches**, *SIAM J. Computing* 22–5 (1993) 935–948.



G. Manzini, **Two space saving tricks for linear time LCP computation**, *Proc. 9th Scandinavian Workshop on Algorithm Theory*, T. Hagerup & J. Katajainen (eds.), Springer Lecture Notes in Computer Science, LNCS 3111, Springer-Verlag (2004) 372–383.



Wataru Matsubara, Kazuhiko Kusano, Akira Ishino, Hideo Bannai & Ayumi Shinohara, **New lower bounds for the maximum number of runs in a string**, *PSC* (2008) 140–145.



Edward M. McCreight, **A space-economical suffix tree construction algorithm**, *J. Assoc. Comput. Mach.* 32–2 (1976) 262–272.



Yuta Mori, **libdivsufsort**: <http://code.google.com/p/libdivsufsort/>



Ge Nong, Sen Zhang & Wai Hong Chan, **Linear time suffix array construction using D-critical substrings**, *Proc. 20th Annual Symp. Combinatorial Pattern*

*Matching*, Gregory Kucherov & Esko Ukkonen (eds.), Springer Lecture Notes in Computer Science, LNCS 5577, Springer-Verlag (2009) 54–67.



Simon J. Puglisi & R. J. Simpson, **The expected number of runs in a word**, *Australasian J. Combinatorics* 42 (2008) 45–54.



Simon J. Puglisi, R. J. Simpson & W. F. Smyth, **How many runs can a string contain?**, *Theoret. Comput. Sci.* 401 (2008) 165–171.



Simon J. Puglisi, W. F. Smyth & Andrew Turpin, **Some restrictions on periodicity in strings**, *Proc. 16th Australasian Workshop on Combinatorial Algs.* (2005) 263–268.











Simon J. Puglisi, W. F. Smyth & Andrew Turpin, **A taxonomy of suffix array construction algorithms**, *ACM Computing Surveys* 39–2 (2007) Article 4, 1–31.



Simon J. Puglisi & Andrew Turpin, **Space-time tradeoffs for longest-common-prefix array computation**, *Proc. 19th Internat. Symp. Algs. & Computation*, S.-H. Hong, H. Nagamochi & T. Fukunaga (eds.) (2008) 124–135.



Wojciech Rytter, **The number of runs in a string: improved analysis of the linear upper bound**, *Proc. 23rd Symp. Theoretical Aspects of Computer Science*, B. Durand & W. Thomas (eds.), Springer Lecture Notes in Computer Science, LNCS 2884, Springer-Verlag (2006) 184–195.

-  R. J. Simpson, **Intersecting periodic words**, *Theoret. Comput. Sci.* 374 (2007) 58–65.
-  Jamie Simpson, **Modified Padovan words and the maximum number of runs in a word**, *Australasian J. Combinatorics* 46 (2010) 129–145.
-  W. F. Smyth, **Repetitive perhaps, but certainly not boring**, *Theoret. Comput. Sci.* 249–2 (2000) 343–355.
-  Bill Smyth, *Computing Patterns in Strings*, Pearson Addison-Wesley (2003) 423 pp.
-  W. F. Smyth, **Computing periodicities in strings — a new approach**, *Proc. 16th Australasian Workshop on Combinatorial Algs.* (2005) 481–488.
-  Esko Ukkonen, **On-line construction of suffix trees**, *Algorithmica* 14 (1995) 249–260.
-  Peter Weiner, **Linear pattern matching algorithms**, *Proc. 14th Annual IEEE Symp. Switching & Automata Theory* (1973) 1–11.
-  Jacob Ziv & Abraham Lempel, **A universal algorithm for sequential data compression**, *IEEE Trans. Information Theory* 23 (1977) 337–343.