

Are Three Squares Impossible? I — The Background

Bill Smyth

Algorithms Research Group, Department of Computing & Software
McMaster University, Hamilton, Canada
email: smyth@mcmaster.ca

10–12 May 2012

Abstract: Series of Talks

These talks describe work done over the last 30 years or so both to understand and to compute repetitions in strings — especially since 1999. We will discover that, although much has been learned, much combinatorial insight gained, there remains much more that is unknown about the occurrence of repetitions in strings and the restrictions they are subject to. I present combinatorial results discovered only recently, and I suggest that **possibly** extensions of these results can be used to compute repetitions in an entirely new way. I hope that members of the audience will be motivated to work on some of the many open problems that remain, thus to extend combinatorial knowledge even further.

Abstract of Talk I

We begin with an introduction to basic notation, terminology and data structures related to strings and associated algorithms. We review the basic algorithms for computing repetitions and runs, as well as the various approaches to determining bounds on the number of runs in a string. We argue that these methods, though ingenious, do not provide enough insight into the possible behaviours of runs in strings and the relationships among them. We propose a new way of thinking about runs (squares) that raises questions for which currently there are only partial answers.

Outline

1. Terminology, Notation & Data Structures
2. Computing Repetitions & Runs
3. The Maximum Number $\rho(n)$ of Runs
4. Combinatorial Insight vs. Brute Force
5. The Three Squares Lemma

Terminology & Notation I

- * A **string** is a finite sequence of symbols (**letters**) drawn from some finite or infinite set Σ 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 x in **mathbold**, and we represent it as an array $x[1..n]$ for some $n \geq 0$. We call $n = x$ the **length** of x . For example,

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

is a string of length $x = 10$ on $\Sigma = \{a, b\}$. 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 $\mathbf{x} = \mathbf{uv}$, then \mathbf{vu} is said to be the u^{th} **rotation** of \mathbf{x} , written $R_u(\mathbf{x})$ or \mathbf{x}_u .
- * If $\mathbf{x} = \mathbf{uv} = \mathbf{wu}$ for $u < x$, then \mathbf{u} is a **border** of \mathbf{x} , and \mathbf{x} has **period** $p = x - u$; that is, for every $i \in 1..u$, $\mathbf{x}[i] = \mathbf{x}[i+p]$. The string (1) has borders \mathbf{abaab} and \mathbf{ab} , hence corresponding periods 5 and 8, respectively.

Terminology & Notation II

- * If $\mathbf{x} = \mathbf{vu}^e\mathbf{w}$, where $e > 1$ and \mathbf{u} is neither a suffix of \mathbf{v} nor a prefix of \mathbf{w} (e is maximum), then \mathbf{u}^e is said to be a **repetition** in \mathbf{x} . The integers u and e are the **period** and **exponent**, respectively, of the repetition.
- * If a repetition \mathbf{u}^e occurs at position i in \mathbf{x} — that is, $\mathbf{x}[i..i+eu-1] = \mathbf{u}^e$ — it can be described by the triple (i, u, e) .
- * For example, in

$$\mathbf{x} = \underline{\underline{abaabababab}},$$

there are repetitions $(1, 3, 2) = (aba)^2$, $(1, 5, 2) = (abaab)^2$,
 $(3, 1, 2) = (8, 1, 2) = a^2$, $(4, 2, 2) = (ab)^2$, $(5, 2, 2) = (ba)^2$. Each of these repetitions is a **square** ($e = 2$). In general, every repetition has a square prefix.

- * An interesting string is the **Fibonacci string** \mathbf{f}_k :

$$\mathbf{f}_0 = b, \mathbf{f}_1 = a; k \geq 2 \implies \mathbf{f}_k = \mathbf{f}_{k-1}\mathbf{f}_{k-2}.$$

It contains $O(f_k \log f_k)$ repetitions [C81, FS99, IS97].

Terminology & Notation III

- * If $\mathbf{v} = \mathbf{x}[i..j]$ has period u , where $v/u \geq 2$, and if neither $\mathbf{x}[i-1..j]$ nor $\mathbf{x}[i..j+1]$ (whenever these are 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** $e = \lfloor v/u \rfloor$.
- * If a run of period u and exponent e occurs at position i in \mathbf{x} , it can be described by the 4-tuple (i, u, e, t) , where the **tail** $t = v \bmod u$. Thus $0 \leq t < u$; when $t = 0$, the run is also a repetition.
- * All of the repetitions in

$$\mathbf{x} = \underline{\underline{abaababaab}}$$

are runs except for $(ab)^2$ and $(ba)^2$: these are substrings of the run $\mathbf{v} = (4, 2, 2, 1) = ababa$.

- * In general, every repetition is a substring of some run; thus computing all the runs **implicitly** computes all the repetitions.

Data Structures: ST, SA, LCP

	1	2	3	4	5	6	7	8
$x =$	a	b	a	a	b	a	b	a
$SA_x =$	8	3	6	1	4	7	2	5
$LCP_x =$	0	1	1	3	3	0	2	2

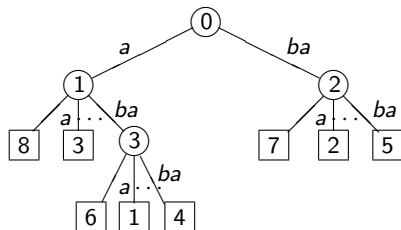


Figure: Suffix tree, suffix array, LCP array

ST Checklist

- ▶ leaf nodes in lexorder
- ▶ internal nodes give lcp
- ▶ $O(x \log \sigma)$ construction time, where σ is the alphabet size
- ▶ $O(x)$ space
- ▶ for construction and search: data structure at each node
- ▶ pattern-matching in time proportional to pattern-length m
- ▶ essential for repeats, repetitions, LZ

Marvellous!

BUT ... too much space!!

Data Structures: LZ

A factorization $x = \mathbf{w}_1\mathbf{w}_2 \cdots \mathbf{w}_k$ is **LZ** (for Lempel-Ziv [LZ76, ZL77]) if and only if each \mathbf{w}_j , $j \in 1..k$, is

- (a) a letter that does *not* occur in $\mathbf{w}_1\mathbf{w}_2 \cdots \mathbf{w}_{j-1}$; or otherwise
- (b) the longest substring that occurs at least twice in $\mathbf{w}_1\mathbf{w}_2 \cdots \mathbf{w}_j$.

We observe that $\mathbf{w}_1 = x[1]$, further that a factor \mathbf{w}_j may overlap with its previous occurrence in x . For the string

$$x = abaababa$$

the factorization LZ $_x$ is given by $\mathbf{w}_1 = a$, $\mathbf{w}_2 = b$, $\mathbf{w}_3 = a$, $\mathbf{w}_4 = aba$, $\mathbf{w}_5 = ba$.

All of these data structures can be computed in time linear in x (well, almost, for ST): ST [W73, M76, U95, F97],

SA [MM90, MM93, PST07, NZC09, M09],

LCP [KLAAP01, M04, PT08, KMP09], LZ [ACIKSTY12].

SA [AKO04] can be used at least as efficiently as ST [A85]. Orders other than lexicographic (e.g., V-order) can also be used for SA construction [DDS12].

Computing LZ is Brute Force!

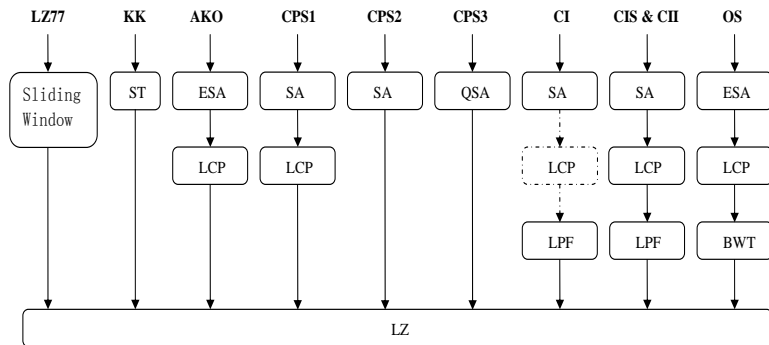


Figure: From [ACIKSTY12]

Computing Repetitions

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

- * Crochemore [C81] describes a method of **successive refinement** that identifies all equal substrings of lengths $1, 2, \dots$ until for some length ℓ every substring is unique. As remarked in [S03], his method is essentially an algorithm for suffix tree construction, with repetitions appearing as adjacent common prefixes of suffixes of \mathbf{x} .
- * Apostolico & Preparata [AP83] use suffix trees plus auxiliary data structures.
- * Main & Lorentz [ML84] use a **divide-and-conquer** approach based on prior computation of $LZ_{\mathbf{x}}$.

Computing Runs

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

$$\rho(n) \leq k_1 n - k_2 \sqrt{n} \log_2 n \quad (2)$$

for some universal positive constants k_1 and k_2 .

- * They provided computational evidence (up to $n = 60$) that $\rho(n) \leq n$ — this was their conjecture.

BUT their method yielded no upper bound on k_1 or k_2 — what if $k_1 = 10^{10^{10}}$?

And Furthermore ...

The K&K method requires **BRUTE FORCE**, the computation of global data structures:

- * SA
- * LCP
- * LZ

when the **expected** number of runs in a string of length n is small (Puglisi & Simpson [PS08]):

- * $0.41n$ 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 in most strings are **SPARSE!**

Bounding $\rho(n)/n$ (I)

Gradually, as researchers realized its importance [S00], the estimation of bounds on $\rho(n)/n$ became a growth industry:

- * In 2006 Rytter [R06] divided runs into two classes: **highly periodic** (hp) and **weakly periodic**. Using separate estimates for these two classes, he showed that $\rho(n)/n < 5$.
- * In 2008 Puglisi, Simpson & Smyth [PSS08] extended Rytter's work: they introduced **θ -hp** runs in which the generator of the run is itself periodic and has length at least θ times its period. They were able to show that $\rho(n)/n \leq 3.48$.
- * Also in 2008 Crochemore & Ilie [CI08] showed that $\rho(n)/n \leq 1.6$ by counting runs based not on their starting positions but on their "centres". For given p , they divide runs into those of period $\geq p$ (counted by $\rho_{\geq p}(n) \leq 6n/p$) and "microruns" of period $\leq p$ (counted by $\rho_{\leq p}(n) \leq bn$ for some computed b). The achieved bound relates to the choice $p = 9$.

Bounding $\rho(n)/n$ (II)

- * Still in 2008 Giraud [G08, G09] established using an elegant argument that $\lim_{n \rightarrow \infty} \rho(n)/n$ in fact exists but is never attained. Moreover, he proved that $\rho(n)/n < 1.52$.
- * Finally, in papers published in 2008 [CIT08] and to appear in 2012 [CIT12], Crochemore, Ilie & Tinta, using the same approach as in [CI08], raised the value of p , the value separating “large” from “small” periods, and introduced massive computation to determine upper bounds on the number of microruns (essentially the determination of “ b ”). They were thus able to claim that $\rho(n)/n \leq 1.048$ using $p = 50$ [CIT08], later that $\rho(n)/n \leq 1.029$ using $p = 60$ [CIT12] — and three **years** of CPU time on the Canadian *SHARCNET* high performance computer!!!

Bounding $\rho(n)/n$ (III)

Meanwhile, lower bounds on $\rho(n)/n$ have also been established:

- * In 2003 Franek, Simpson & Smyth [FSS03] exhibited an infinite family of strings for which $\rho(n)/n > 0.927$ for n sufficiently large.
- * In 2008 Matsubara, Kusano, Ishino, Bannai & Shinohara [MKIBS08] found an infinite family for which $\rho(n)/n > 0.944565$.
- * In 2010 Simpson [S10] used Padovan words to show that $\rho(n)/n > 0.944575!!$

So what we think we know in 2012 is that, for sufficiently large n ,

$$0.944575 < \rho(n)/n \leq 1.029.$$

I Have Three Problems With These Results:

1. The “proof” that $\rho(n)/n \leq 1.029$ suffers from Four Colour Theorem Syndrome [AH77] — does the program really work?
2. We don't really know **why** $\rho(n)/n \leq 1.029$ — shortage of combinatorial insight!
3. We get no algorithm out of it — still the only way to compute runs is by brute force!

What we want is 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.

A Tiny 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? Not very much – but we do have a little!

What We Know:

Lemma (Crochemore & Rytter (1995) [CR95])









Suppose \mathbf{u} is not a repetition, and suppose $\mathbf{v} \neq \mathbf{u}^j$ for any $j \geq 1$. If \mathbf{u}^2 is a prefix of \mathbf{v}^2 , in turn a proper prefix of \mathbf{w}^2 , then $w \geq u+v$.

The Fibostring 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
 $\mathbf{x} = a b a a b \underline{a} b a a \underline{b} a a b a b \underline{a} a b a b a a b a a \underline{b}$

This tells us that if three squares occur at the same position, then one of them has to be “large”. But we want to know more: what if the three squares just overlap, just occur in the same neighbourhood? What then???

In the next lecture we begin to attack this problem.

-  Mohamed I. Abouelhoda, Stefan Kurtz, & Enno Ohlebusch, **Replacing suffix trees with enhanced suffix arrays**, *J. Discrete Algorithms* 2 (2004) 53–86.
-  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* (2012) to appear.
-  Alberto Apostolico, **The myriad virtues of suffix trees**, *Combinatorial Algorithms on Words* (NATO ASI Series F12), Springer-Verlag (1985) 85–96.
-  Alberto Apostolico & Franco P. Preparata, **Optimal off-line detection of repetitions in a string**, *Theoret. Comput. Sci.* 22 (1983) 297–315.
-  Kenneth Appel & Wolfgang Haken, **Solution of the four color map problem**, *Scientific American* 237–4 (1977) 108–121.
-  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, **Towards a solution to the “runs” conjecture**, *Proc. 19th Annual Symp. Combinatorial Pattern Matching*

P. Ferragina & G. Landau (eds.), Lecture Notes in Computer Science, LNCS 5029, Springer-Verlag (2008) 290–302.



Maxime Crochemore, Lucian Ilie & Liviu Tinta, **The “runs” conjecture**, *TCS* (2012) to appear.



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



David E. Daykin, Jacqueline W. Daykin & W. F. Smyth, **A linear partitioning algorithm for hybrid Lyndons using V-order**, *Theoret. Comput. Sci.* (2012) to appear.








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, **The exact number of squares in Fibonacci words**, *Theoret. Comput. Sci.* 218–1 (1999) 95–106.



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.

-  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.), 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.
-  Costas S. Iliopoulos & W. F. Smyth, **A characterization of the squares in a Fibonacci string**, *Theoret. Comput. Sci.* 172 (1997) 281–291.
-  Juha Kärkkäinen, Giovanni Manzini & Simon J. Puglisi, **Permuted longest-common-prefix array**, *Proc. 20th Annual Symp. Combinatorial Pattern Matching*, Gregory Kucherov & Esko Ukkonen (eds.), Lecture Notes in Computer Science, LNCS 5577 (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*, Amihood Amir & Gad M. Landau (eds.), 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.
-  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.), 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.), 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, **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.), LNCS 2884, Springer-Verlag (2006) 184–195.
-  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.
-  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.