Lyndon factors and periodicities in strings

Franya Franek

Advanced Optimization Laboratory
Department of Computing and Software
McMaster University, Hamilton, Ontario, Canada

AdvOL seminar talk
April 2016

Outline

1. Motivation and background
2. Lyndon words
3. Lyndon roots and L-roots
4. Lyndon arrays

A ***repetition*** of a substring (i.e. a ***tandem repeat***) in a string is a simple concept:

$$\cdots \underbrace{abba}\,\underbrace{abba}\cdots$$

A very natural question is how many repetitions there can be in a string? Why this is not simple?

$$\cdots \underbrace{\underbrace{aba}\underbrace{ab}}\underbrace{\underbrace{a}baab}\cdots$$

$O(\log(n))$ repetitions can start at the same position!

$$\cdots \underbrace{\underbrace{aba}}\,\underbrace{\underbrace{\underbrace{aba}}}\,\underbrace{\underbrace{aba}}\cdots$$

A bigger repetition represents several repetitions

It is enough to determine (count) maximal repetitions:

Crochemore 1978 – there are at most $O(n\log(n))$ maximal repetitions in a string of length n. It is an optimal upper bound, as it is attained by Fibonacci strings.

(*Let* $F_0 = a$ *and* $F_1 = ab$. *Define* $F_n = F_{n-1}F_{n-2}$.)

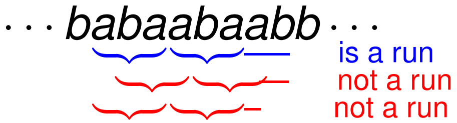
$$\begin{array}{l}\cdots abaabaab \cdots \\ \underbrace{}\underbrace{} \\ \underbrace{}\underbrace{} \\ \underbrace{}\underbrace{}\end{array}$$

Even incomplete repetitions represent several repetitions

$$\cdots \underbrace{\underbrace{aba}\underbrace{aba}}\underbrace{\underbrace{aba}\underbrace{aba}} \cdots$$

Moreover, non-primitive repetitions represent several repetitions

This leads to the notion of run: maximal fractional repetition that can be extended neither to the left nor to the right.



Moreover the ***root*** (the repeating part) must be ***primitive*** (not a repetition)

Run can be fully described by 3 integers: starting position, ending position, and the period.

Now the question becomes more interesting: how many runs there can be in a string? $O(n\log(n))$? $O(n)$?

In 1999 *Kolpakov+Kucherov* proved that the maximum number of runs in a string is linear in the string's length and conjectured that it is in fact bounded by the length.

Many additional researchers (*Bannai, Crochemore, Ilie, Ishino, Kusano, Matsubara, Puglisi, Rytter, Shinohara, Simpson, Smyth, F.*) contributed to improving the asymptotic lower and upper bounds

$$0.944565n \leq \rho(n) \leq 1.029n$$

where $\rho(n) = \max\{r(x) : |x| = n\}$ and $r(x)$ denotes the # of runs in x

In 2013, *Dez+F.* introduced d-step approach:

considering the role played by the size of the alphabet of the string and investigated the function $\rho_d(n)$, i.e. the maximum number of respectively runs, over all strings of length n containing exactly d distinct symbols.

They conjectured that

- $\rho_d(n) \leq n - d$ for all $n \geq d \geq 2$
- There is unique (up to relabeling) run-maximal string of length $2d$ with d symbols: $(aa)(bb)(cc)\cdots$

In 2014, *Bannai* et al. introduced a powerful method of mapping runs into partitions of the positions via L-roots of the runs, thus proving that $\rho(n) \leq n-1$ for any $n \geq 1$, i.e. gave a universal upper bound.

In 2015, *Deza+F.*, and simultaneously and independently *Bannai* et al. proved the d-step conjectures utilizing the L-roots, i.e.:

(a) $\rho_d(n) \leq n-d$ for $n \geq d \geq 2$

(b) $\rho_d(n) \leq n-d-1$ for $n > 2d \geq 4$

This gives $\rho(n) \leq n-3$ for $n > 4$ and (a)+(b) give in turn the uniqueness of the run-maximal $(d, 2d)$-strings.

Recently, *Deza+F.*, refined the method and showed that in addition

$\rho_d(n) \leq n - d - 2$ for $n > 2d + 4 \geq 8$

which gives

$\rho(n) \leq n - 4$ for $n > 8$

and *Fischer+Holub+I+Lewenstein* used L-roots to improve the upper bound for runs for binary strings from

$\rho_2(n) \leq n - 4$ for $n > 8$ to $\rho_2(n) \leq \frac{22}{23} n \leq 0.957n$

The L-roots of a run are those Lyndon roots of the run that do not start at the beginning of the run, which brings us to the topic of Lyndon words.

Lyndon words

Definition

A string x is a ***Lyndon word*** if x is lexicographically strictly smaller than any non-trivial rotation of x. Trivially true when $|x| = 1$, so-called ***trivial*** Lyndon word.

The following are all equivalent:

- x is a non-trivial Lyndon word
- $x[1..n] \prec x[i..n]$ for any $1 < i \leq n$
- $x[1..i] \prec x[i{+}1..n]$ for any $1 \leq i < n$
- there is $1 \leq i < n$ so that $x[1..i] \prec x[i{+}1..n]$ and both $x[1..i]$ and $x[i{+}1..n]$ are Lyndon (standard factorization *when $x[i{+}1..n]$ is the longest*)

abb is Lyndon (*abb bba bab*)

aba is not (*aba baa aab*)

abab is not (none of the rotations is strictly smallest: *abab baba abab baba*)

Lyndon $\Rightarrow$ unbordered $\Rightarrow$ aperiodic $\Rightarrow$ primitive

- Lyndon words have an application to the description of free Lie algebras, this was *Lyndon's* original motivation for introducing these words.

- Linear time constant space generation of Lyndon words provides an efficient method for constructing a particular de Bruijn sequence in linear time and logarithmic space.

- Radford's theorem states that the Lyndon words are algebraically independent elements of the shuffle algebra, and generate it.

- Lyndon words correspond to aperiodic necklace class representatives and can thus be counted with Moreau's necklace-counting function.

- *Given a string s, find its Lyndon rotation*. Problem arises in chemical databases for circular molecules.The canonical representation is the lexicographically smallest rotation.

Theorem (*Chen+Fox+Lyndon*, 1958, Lyndon factorization)

For any string x there are unique Lyndon words u_1, ..., u_k so that $u_{i+1} \preceq u_i$ and $x = u_1 u_2 ... u_k$.

Duval, 1983, presented an efficient elegant linear time constant space algorithm to compute Lyndon factorization.

Each u_i is in fact a ***maximal Lyndon factor***, some may even be ***non-extendible Lyndon factors***.

Lyndon roots and L-roots

A repetition in a string can be considered a sequence of right cyclic shifts of its root:

$\ldots \textit{babbababba} \ldots$

$\ldots \textit{babbababba} \ldots$

$\ldots \textit{babbababba} \ldots$

$\ldots \textit{babbababba} \ldots$

$\ldots \textit{babbababba} \ldots$

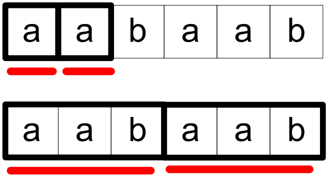
$\ldots \textit{babbababba} \ldots$

If the root of the repetition is primitive, one of the shifts is guaranteed to be Lyndon !

Hence, every run has one or more Lyndon roots !

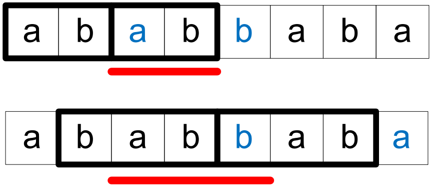
L-roots of a run are defined as the Lyndon roots of the run except the one that starts at the beginning of the run. Why?

We want to achieve that no two L-roots start at the same position.

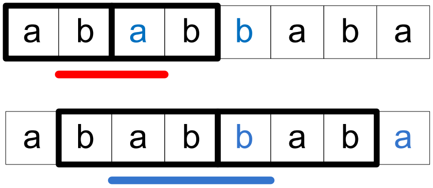


Note that we can afford it – it only affects the runs whose root is Lyndon, and in this case we are guaranteed at least two Lyndon roots, hence at least one L-root.

But this is still not good enough:



but when this happen, the blue symbols “disagree” suggesting a way out: if the blue letters are in this way $\prec$, we use to determine the Lyndoness by the reverse order $\preceq^{-1}$, otherwise by the order $\preceq$.



reverse the order

keep the order

Theorem (*Bannai et al.*, 2015)

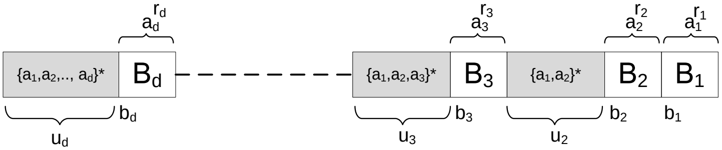
For any string x, there are no two L-roots that start at the same position.

The proof is simple – the trick with reversing the order makes every L-root a non-extendible Lyndon factor !

An easy consequence – $\rho(n) \leq n$ – as we can map every run to the starting positions of all its L-roots. These sets of positions are mutually disjoint.

Since the very first position is excluded by definition from hosting an L-root, in fact $\rho(n) \leq n - 1$.

Canonical form of a string (*Deza+F.*):



Any string can be modified to be in canonical form, by relabeling:

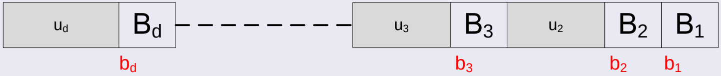
abcbcaabccc	c -> a
ababaaabaaa	b -> b
cbabaccbaaa	a -> c

cbaba	cc	b	aaa
u_3	B_3	B_2	B_1

Lemma (*Deza+F.*)

$\rho_d(n) \leq n - d$ *for any* $n \geq d \geq 2$.

Proof.

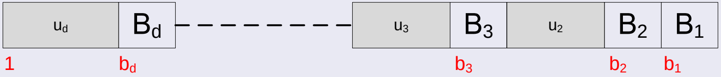


There cannot be an L-root starting at any b_k □

Lemma (*Deza+F.*)

$\rho_d(n) \leq n - d - 1$ *for any* $n > 2d$, $d \geq 2$.

Proof.

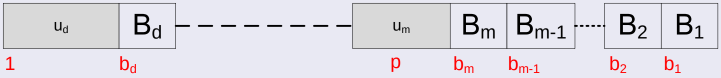


The size guarantees that u_d is not empty. There cannot be an L-root starting at any b_k, and no L-root at position 1 □

Lemma (*Deza+F.*)

$\rho_d(n) \leq n - d - 2$ *for* $n > 2d + 4$, $d \geq 2$.

Proof.



The size guarantees that u_d is not empty and u_m is big enough to have a position p with no L-root. There cannot be an L-root starting at any b_k, and no L-root at position 1 □

Lyndon arrays

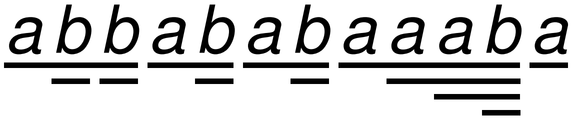
Bannai et. al used the knowledge of all maximal Lyndon factors of a string w.r.t. $\preceq$ and $\preceq^{-1}$ to compute all runs in linear time.

This gives credence to the notion of Lyndon array of a string x and the efforts to compute the Lyndon array in linear time.

Definition

$x = x[1..n]$ a string over ordered alphabet $(\mathcal{A}, \preceq)$.
An integer array $L[1..n]$ (or alternatively $\lambda[1..n]$) is a ***Lyndon array*** of x w.r.t. $\preceq$ iff $L[i]$ is the length of the maximal Lyndon factor of x starting at i (or $\lambda[i]$ is the end position of the maximal Lyndon factor of x starting at i).

Lyndon factors:



Lyndon array:

3 1 1 2 1 2 1 4 3 2 1 1

Lemma (*Hohlweg+Reutenauer*, 2003)

For any string $x = x[1..n]$, $x[i..j]$ is a maximal Lyndon factor of x iff $x[j{+}1..n] \prec x[i..n]$.

Definition

Suffix array $s[i]$ of a string $x = x[1..n]$ is an integer array so that $s[i] = j$ iff $x[i..n]$ is the j-th suffix in the lexicographic ordering.

Suffix array can be computed in linear time!! (*Kărkkăinen+ Sanders*, *Kim+Sim+Park+Park*, *Ko+Aluru*).

Inverse suffix array $s^{-1}[j] = i$ iff $s[i] = j$.

Can also be computed from the suffix array in linear time.

Thus, $s^{-1}[i] < s^{-1}[j]$ iff $x[i..n] \prec x[j..n]$.

Lyndon array can be computed in linear time using stack from the inverse suffix array by NSV (next smallest value) algorithm.

Lyndon array can be computed in linear time

A small beauty fault: computing suffix array in linear time is quite laborious and involved. Hence: a goal is to find a direct linear time algorithm to compute Lyndon array bypassing the computation of the suffix array altogether.

But, is it possible? What if by computing Lyndon array one actually sorts out the suffixes?

Holub+Islam+Smyth+F. : For a binary alphabet, except a special case when the Lyndon array is all 1's, one can determine in linear time the unique string of which it is the Lyndon array, i.e. in linear time we can sort the suffixes from the Lyndon array.

Thus, for binary strings, computing the Lyndon array is as hard as sorting suffixes.

This is not true for alphabet of bigger sizes.

Mantaci+Restivo+Rosone+Sciortino: sorting suffixes from Lyndon decomposition.

No complexity given explicitly, but looks like $O(n^2)$.

Proposition

Let $u_1..u_k$ be the Lyndon factorization of x. Then $sort(u_1..u_k) = merge(sort(u_1..u_r), sort(u_{r+1}..u_k))$.

Can easily be reformulated in terms of Lyndon array as it gives more information than just Lyndon factorization.

The real question is whether it can be done in linear time

Which array can be a Lyndon array of some string?

Holub+Islam+Smyth+F. : Any integer array satisfying ***Monge condition*** is a Lyndon array of some string.

Definition

Monge condition for $L[1..n]$:
For any i, $1 \leq i < n$ and any j, $i < j < i + L[i]$, $j + L[j] \leq i + L[i]$.

THANK YOU