Introduction
Old results
New Results

Make Pals Your Pals

Arseny M. Shur

Ural Federal University, Ekaterinburg, Russia

Joint work with
K. Borozdin, D. Kosolobov, O. Merkurev, and M. Rubinchik
Palindromes

- Palindrome is a string that reads the same in both directions
  - like rotator
- There is also a generalized version (palindromes with involution) inspired by the Watson–Crick palindromes in DNA/RNA strands

Topics
- Find/count palindromes in a string
- Compare palindromes in two or more strings
- Factorize a string into palindromes
- Strings with maximum number of palindromes
- Expected distribution of palindromes in strings
Notation and Definitions

- Array notation for strings (words): $S = S[1..n]$
  - $n = |S|$, $\sigma = \text{alph}(S)$
- Substring $S[i..j]$, prefix $S[1..i]$, suffix $S[j..n]$
- Reversal: $\overline{S} = S[n]S[n-1]\cdots S[1]$
- Palindrome: $S = \overline{S}$
- Involution: letter-to-letter map $\theta$ such that $\theta^2 = \text{id}$
- Involution palindrome: $S = \theta(\overline{S})$
- Gapped palindrome: $ST\overline{S}$, where $T[1] \neq T[|T|]$
- Subpalindrome: substring $S[i..j]$ which is a palindrome
  - has center $(j + i)/2$ and radius $\lceil (j - i)/2 \rceil$
  - the set of centers is $\{1, \frac{3}{2}, 2, \ldots, n - \frac{1}{2}, n\}$
Agreements

Alphabets:
- general ordered
  - only comparisons; sorting in $n \log n$ time
- integer of polynomial size
  - many tricks including sorting in linear time

Computation:
- Word-RAM model
- input string usually arrives online, symbol by symbol
  - an algorithm solves a problem for a string $S$ online if it gives the answer for every prefix $S[1..i]$ before reading $S[i+1]$
- Streaming model (sublinear space available)

Disclaimer:
- All results are formulated for palindromes, many translate to involution pals, none translates to gapped pals
Outline

1. Introduction

2. Old results
   - Search/count
   - Factorizations

3. New Results
Array of Radiiuses

$Rad[c]$: maximum radius of a subpalindrome centered at $c$

**Theorem**

The array $Rad$ can be computed “almost” online in linear time ([Manacher’s algorithm](https://en.wikipedia.org/wiki/Manacher%27s_algorithm), Manacher, 1975). The algorithm can be made real-time using lazy computation (Galil, 1976).

**Example:** (the red part is computed online)

<table>
<thead>
<tr>
<th>S</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>b</th>
<th>c</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rad</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

As a data structure, the array $Rad$

- Compactly represents all subpalindromes of a string
- Answers the queries “is $S[i..j]$ a palindrome?” in $O(1)$ time
  - Compare $Rad[(j + i)/2]$ to $[(j – i)/2]$
Manacher’s algorithm allows one to compute online
- the longest prefix palindrome, the longest suffix palindrome
  and the longest subpalindrome of a string
  - in particular, to check whether the string is a palindrome
- the total number of (occurrences of) palindromes in a string
... but gives no information about the number of distinct
palindromes which occur in the string
Palindrome or Not?

Checking whether a string is a palindrome

- In the RAM model:
  - online in linear/real time by Manacher’s algorithm/ Galil’s modification

- On a multi-tape Turing machine:
  - online in linear time (Slisenko 1973, simplified by Galil 1975)

- In the streaming model:
  - in real time, w.h.p. (by Karp–Rabin hashes)
Factorizations into Palindromes

Checking whether a string can be factorized into
- Even-length palindromes
  - online in real time (Knuth, Morris, Pratt 1977 + later improvements)
- Palindromes of length > 1
  - online in linear time (Galil, Seiferas 1978)
- Two palindromes
  - online in linear time (Galil, Seiferas 1978)
- Three/four palindromes
  - linear time (Galil, Seiferas 1978)
- $k$ palindromes, for any constant $k$
  - conjectured to be linear time
Outline

1 Introduction

2 Old results

3 New Results
   - Combinatorics
   - Eertree and Factorizations
   - Streaming
Distribution of Palindromes

\( E(n, \sigma) \) is the expected number of palindromes in a \( \sigma \)-ary string of length \( n \)

**Theorem (RS 2016)**

For any fixed \( \sigma > 1 \), \( E(n, \sigma) = \Theta(\sqrt{n}) \).

The function \( E(n, \sigma)/\sqrt{n} \) oscillates between the values of size \( \Theta(1) \) and \( \Theta(\sqrt{\sigma}) \).

\( L(n, \sigma) \) is the expected length of a subpalindrome of a \( \sigma \)-ary string of length \( n \)

**Proposition (easily follows from RS 2016)**

For any fixed \( \sigma > 1 \), \( L(n, \sigma) = (2 + o(1)) \log_\sigma n \).
Upper bounds for the expected number of distinct palindromes of length $s \in \{2m, 2m+1\}$ are the total number of palindromes and the expected number of subpalindromes of length $s$.

Matching lower bounds from the estimations of the number of strings without a given substring (Guibas, Odlyzko 1981)
Rich String, Poor String

The **minimum** number of distinct palindromes in a $\sigma$-ary long (even infinite) string is constant for every $\sigma > 1$ (poor strings, not very interesting)

The **maximum** number of distinct palindromes in a length-$n$ string is $n$ (rich strings, have many nice properties:

- A substring or reversal of a rich string is rich
- Any rich string can be extended to a longer rich strings
- Include sturmian strings and their generalizations
- Several combinatorial characterizations

The number $R_\sigma(n)$ of rich strings grows with length unusually:

- $R_\sigma(n) > R_2(n) \geq C\sqrt{n}$ for $C \approx 37.6$ (Guo, Shallit, S 2016)
- $R_\sigma(n) = O(2^{\frac{n \log \log n}{\log n}})$ (Rukavička 2017)
Eertree: linear-size tree-like data structure capturing all information about subpalindromes of a string
Introduced by Mikhail Rubinchik (RS, IWOCA 2015)

- Vertices: palindromes + \{0, −1\}
- Edges: \(W \rightarrow aWa\), labeled by \(a\)
  - two trees with roots 0 and −1
- Suffix links: longest suffix palindrome
  - reversed tree with root −1
- Lengths are stored (not strings!)
- Optional: “fast track” suffix links
  - Space is often sublinear
    - \(O(\sqrt{\sigma n})\) for random \(\sigma\)-ary strings
For a string $S[1..n]$, $eertree(S)$ can be built in the time

- $O(n \log \sigma)$ online for a general alphabet
  - log $n$ per step or
  - log $\sigma$ per step + some extra space
- $O(n)$ offline for an integer alphabet
- $O(n \alpha(n))$ online for an integer alphabet
  - $\alpha(n)$: insertion time for a hash-based dictionary
  - randomized: $\alpha(n) = O(1)$
  - deterministic: $\alpha(n) = \Theta((\log \log \sigma)^2)$
Free with Eertree

Some problems can be solved just by building an eertree (possibly with some additional fields stored in vertices)

- Find/count distinct subpalindromes
- Compare palindromes in two strings (build “joint” eertree)
  - number of common palindromes
  - longest common palindrome
  - shortest distinctive palindrome
  - palindromes having the same / different numbers of occurrences
  - …
Factorizations with Eertree

Two main factorization problems:

- **k-factorization**: given $k$, can $S[1..n]$ be factorized into exactly $k$ palindromes?
- **Palindromic length**: what is the minimum $k$ such that $S[1..n]$ admits a $k$-factorization?
  - the first problem does not reduce to the second, because a string with a $k$-factorization can have no $(k+1)$-factorization

Both problems solved with eertree in $O(n \log n)$ time

- same time bound for palindromic length was first obtained by Fici et al and by I et al (2014)
- method: dynamic programming, testing every suffix palindrome of the current string as the last palindrome in the factorization; $O(n^2)$ in a naive way, reduced to $O(n \log n)$ using series of palindromes
Counting Hard with Eertree

Consider the following query problem:

- **Palindromes in substrings**: a string $S$ arrives online, and each symbol is followed by zero or more queries $count(i, j)$ which should be answered by the number of distinct palindromes in $S[i..j]$

**Theorem (RS, SPIRe 2017)**

Palindromes in substrings can be solved in time $O(n \log n)$ plus $O(\log n)$ per query, using $O(n \log n)$ space. Restricted versions (e.g., an offline version and a problem of finding all rich substrings of a string) require $O(n)$ space.

Ingredients: eertree + a persistent lazy version of **segment tree** (a data structure for computing symmetric functions on arrays)
Bit Compression (Four Russians’ trick):

- \( \log n \) bits are assumed to fit into a machine word
- packing a bit array into machine words, we can process subarrays of \( \log n \) bits in \( O(1) \) time using
  - standard operations
  - custom functions given by precomputed tables of size \( o(n) \)

\( \log n \) speed-up of an algorithm can (sometimes) be obtained
Factorization with Bit Compression

Bit Compression (Four Russians’ trick):
- log $n$ bits are assumed to fit into a machine word
- packing a bit array into machine words, we can process subarrays of log $n$ bits in $O(1)$ time using
  - standard operations
  - custom functions given by precomputed tables of size $o(n)$

$\log n$ speed-up of an algorithm can (sometimes) be obtained

Theorem (KRS, SOFSEM 2015)

There is an online algorithm solving the $k$-factorization problem in $O(kn)$ time and $O(n)$ space.

Theorem (BKRS, CPM 2017)

There is an online algorithm solving the palindromic length problem in $O(n)$ time and space.
Palindromic Series Example

aaabaabaabaaabaabaaabaabaaabaabaaabaabaaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaba
Palindromic Series Example

aaabaabaabaaabaabaabaaabaaabaabaabaabaaabaabaabaaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaabaabaabaabaabaaa

period = 1:
   a
Palindromic Series Example

aaabaabaabaaabaabaabaaabaabaabaaabaabaabaabaabaabaabaabaabaabaabaabaaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaaba
Palindromic Series Example

\[aaabaabaabaaabaabaabaaababaabaabaaabaabaabaaabaabaab\]

\[
\begin{align*}
\text{period = 1:} & & \text{period = 2:} & & \text{period = 3:} \\
\text{a} & & \text{aba} & & \text{abaaba} \\
\end{align*}
\]

\[
\begin{align*}
\text{abaaba} & & \text{abaabaaba}
\end{align*}
\]
Palindromic Series Example

\[
\text{period } = 1: \quad \text{period } = 2: \quad \text{period } = 3:
\]
\[
a \quad \text{aba} \quad \text{abaabaavaabaabaabaabaabaabaabaabaabaaba}
\]

\[
\text{period } = 10:
\]
\[
\text{abaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaabaab
\]
Sketch for $O(n \log n)$

- Let $ans[n]$ be the palindromic length of the processed string.
- Let $u_1, \ldots, u_t$ be all its suffix palindromes.
- $ans[n] = 1 + \min_{i \in [1..t]} ans[n-|u_i|]$.
- Let $U_1, \ldots, U_k$ be all its series.
- $ans[n] = 1 + \min_{i \in [1..k]} \min_{u \in U_i} ans[n-|u|]$.
- After appending a symbol:
  - Each internal minimum can be computed in $O(1)$ time, $O(k)$ in total.
  - The list of series can be recalculated in $O(k)$ time.
Good and Bad Iterations

append(a)

append(b)
Streaming

In the streaming model, the input string arrives online symbol by symbol and cannot be stored: the available memory is sublinear

- Still, some pattern matching and search of regularities can be performed (wait for Tanya’s lecture for a comprehensive account...)

Features of streaming model:

- Many problems can be solved only approximately, or w.h.p., or both
- Many trade-offs (e.g., memory vs approximation ratio)
- Real-time algorithms are utterly important
Finding the longest palindrome in a stream “requires both”:

- provably, the problem can be solved only by a Monte Carlo algorithm, reaching the required approximation ratio \( \text{w.h.p.} \)

Tool to detect palindromes: Karp-Rabin hash

- \( \alpha > 0, \ p \in [n^{3+\alpha}, n^{4+\alpha}] \cap \text{PRIMES} \)
- \( r \) is a fixed integer randomly chosen from \( \{1, \ldots, p-1\} \)

For \( S \), its forward and reversed hash are defined as

\[
\phi^F(S) = \left( \sum_{i=1}^{n} S[i] \cdot r^i \right) \mod p; \ \phi^R(S) = \left( \sum_{i=1}^{n} S[i] \cdot r^{n-i+1} \right) \mod p
\]

- condition \( \phi^F(u) = \phi^R(u) \) defines a palindrome modulo the (improbable) collisions of hashes
- \( O(1) \) computation of \( \phi(S[1..i+1]) \) from \( \phi(S[1..i]) \); of \( \phi(S[i..j]) \) from \( \phi(S[1..i]) \) and \( \phi(S[1..j]) \)
Longest Palindrome in a Stream (2)

Theorem (Gawriechowski, Uznanski, MS, CPM2016)

1. Longest palindrome in a stream cannot be found approximately within $o(M \log \min\{\sigma, M\})$ bits of memory, where $M = n/E$ for approximating the answer with additive error $E$ and $M = \frac{\log n}{\log(1+\varepsilon)}$ for approximating with multiplicative error $\varepsilon$.

2. For both additive and multiplicative error, there exist real-time algorithms finding a longest palindrome with a given error within $O(M)$ words of memory.

If a string is close to random, it contains palindromes of length $O(\log n)$ only; a sliding-window real-time modification of Manacher’s algorithm can found a longest palindrome exactly in $O(\log n)$ space!