Strings, periods, and borders

String Algorithms
Recall: Notation and terminology

\(x: \text{array} \ [1..n] \) of \(A\) is a \textit{string} on alphabet \(A\) of \textit{length} \(n=|x|\). The \textit{letter} at position \(i\) is \(x[i]\), i.e. \(x = x[1]x[2]...x[n]\)

\(x[i..j] = x[i] \ x[i+1] \ ... \ x[j]\) is a \textit{substring} of \(x\) of length \(j-i+1\), and a \textit{proper substring} if \(j-i+1 < n\). It occurs in \(x\) at position \(i\)

The substring \(x[1..i]\) is a \textit{prefix} of \(x\), and a \textit{proper prefix} if \(i < n\), and \(x[i..n]\) is a \textit{suffix} of \(x\), and a \textit{proper suffix} if \(i > 1\)
A non-trivial algorithm for computing the border array of string. The algorithm is based on a (semi-)clever insight into the structure of borders, it has a nice time analysis, and border arrays are important for fast algorithms for exact pattern matching.
A border of $x$ is any *proper prefix* of $x$ that equals a suffix of $x$.

... overlapping borders of a string imply that the string is periodic ...

$$abaabaab = (aba)(aba)(ab) = (aba)^2ab = (aba)^{8/3}$$
Periodicities

1 \beta n
Periodicities

$\beta$

1 $\rightarrow$ $\rightarrow$ $n$
Periodicities

1 \quad \beta \quad n
Periodicities

\[ x = (1)^3 \]
Let $p = n - \beta$, then $x = u^{n/p} = u^{[n/p]}u'$, where $u' = x[1..n-[n/p] \cdot p]$

The normal form of $x[1..n]$ is $u^{n/p^*}$, where $p^*$ is the minimum period
Periodicities

... one of the classic lemmas concerning properties of strings ...

“The Periodicity Lemma”: Let \( p \) and \( q \) be two periods of \( x = x[1..n] \), and let \( d = \gcd(p, q) \). If \( p+q \leq n+d \), then \( d \) is also a period of \( x \)

\[
\begin{align*}
p = 9 & \quad \text{aabaabaabaabaabaabaabaab} \\
q = 6 & \quad \text{aabaabaab} \\
d = \gcd(9, 6) = 3 & \quad \text{aab aab aab aab aab}
\end{align*}
\]
Computing borders

... computing the length of the longest border implies normal form ...

$$\beta = 0$$

for $i=1$ to $n-1$ do

if $x[1..i] = x[n-i+1..n]$ then

$$\beta = i$$

return $\beta$

Running time?
Computing borders

... computing the length of the longest border implies normal form ...

\[
\beta = 0 \\
\textbf{for } i=1 \text{ to } n-1 \text{ do} \\
\quad \textbf{if } x[1..i] = x[n-i+1..n] \text{ then} \\
\quad \quad \beta = i \\
\textbf{return } \beta
\]

Running time? \(O(n^2)\)

... can we do better? Yes, by computing more ...
Border array

In the **border array** $\beta[1..n]$ of $x$, entry $\beta[i]$ is the length of the longest border of $x[1..i]$.
Border array

In the **border array** $\beta[1..n]$ of $x$, entry $\beta[i]$ is the length of the longest border of $x[1..i]$

We know: (1) $\beta[1] = 0$, (2) if $\beta[i+1] = b$, then $\beta[i] \geq b - 1$, i.e. $\beta[i+1] \leq \beta[i] + 1$
In the **border array** $\beta[1..n]$ of $x$, entry $\beta[i]$ is the length of the longest border of $x[1..i]$

We know: (1) $\beta[1] = 0$, (2) if $\beta[i+1] = b$, then $\beta[i] \geq b - 1$, i.e. $\beta[i+1] \leq \beta[i] + 1$

Observation: if $\beta[i+1] = b$, then $x[1..i]$ has a border of length $b - 1$

How can we compute $\beta[i+1]$ from $\beta[i]$?
In the *border array* $\beta[1..n]$ of $x$, entry $\beta[i]$ is the length of the longest border of $x[1..i]$

We know: (1) $\beta[1] = 0$, (2) if $\beta[i+1] = b$, then $\beta[i] \geq b-1$, i.e. $\beta[i+1] \leq \beta[i]+1$

Observation: if $\beta[i+1] = b$, then $x[1..i]$ has a border of length $b-1$

How can we compute $\beta[i+1]$ from $\beta[i]$?

if $x[\beta[i] + 1] = x[i+1]$, then $\beta[i+1] = \beta[i]+1$, otherwise ...

Observation: if $\beta[i+1] = b$, then $x[1..i]$ has a border of length $b-1$
In the \textit{border array} $\beta[1..n]$ of $x$, entry $\beta[i]$ is the length of the longest border of $x[1..i]$

Try to extend the \textit{second longest} border of $x[1..i]$. 
In the border array $\beta[1..n]$ of $x$, entry $\beta[i]$ is the length of the longest border of $x[1..i]$

Try to extend the **second longest** border of $x[1..i]$

i.e. the **longest border** of $x[1..\beta[i]]$

if $x[\beta[\beta[i]]+1] = x[i+1]$, then $\beta[i+1] = \beta[\beta[i]]+1$, otherwise ...
In the *border array* $\beta[1..n]$ of $x$, entry $\beta[i]$ is the length of the longest border of $x[1..i]$

Try to extend the *third longest* border of $x[1..i]$

If $x[\beta[\beta[\beta[i]]]+1] = x[i+1]$, then $\beta[i+1] = \beta[\beta[\beta[i]]]+1$, otherwise ...
Border array $\beta$ of $x[1..n]$

$\beta[1] = 0$

for $i = 1$ to $n-1$ do

$b = \beta[i]$

while $b > 0$ and $x[i+1] \neq x[b+1]$ do

$b = \beta[b]$

if $x[i+1] = x[b+1]$ then

$\beta[i+1] = b+1$

else

$\beta[i+1] = 0$
Border array $\beta$ of $x[1..n]$

$\beta[1] = 0$

for $i = 1$ to $n-1$ do
    $b = \beta[i]$
    while $b > 0$ and $x[i+1] \neq x[b+1]$ do
        $b = \beta[b]$
    if $x[i+1] = x[b+1]$ then
        $\beta[i+1] = b+1$
    else
        $\beta[i+1] = 0$

Correctness: Follows by construction ...

$\beta[i+1]$ is set to $b+1$ (or 0), where $b$ is the length of the longest border of $x[1..i]$ which can be extended, i.e. $x[i+1] = x[b+1]$
Border array $\beta$ of $x[1..n]$

$$\beta[1] = 0$$

**for** $i = 1$ to $n-1$ **do**

$$b = \beta[i]$$

**while** $b > 0$ and $x[i+1] \neq x[b+1]$ **do**

$$b = \beta[b]$$

**if** $x[i+1] = x[b+1]$ **then**

$$\beta[i+1] = b+1$$

**else**

$$\beta[i+1] = 0$$

**Running time?**
Border array $\beta$ of $x[1..n]$

$\beta[1] = 0$

for $i = 1$ to $n-1$ do

$\ b = \beta[i]$

while $b > 0$ and $x[i+1] \neq x[b+1]$ do

$\ b = \beta[b]$

if $x[i+1] = x[b+1]$ then

$\beta[i+1] = b+1$

else

$\beta[i+1] = 0$

Running time? If we disregard the while-loop, then time is $O(n)$, i.e. total time is “$O(n) +$ total time for while-loop” ....
Border array $\beta$ of $x[1..n]$

\[
\beta[1] = 0 \\
\textbf{for} \ i = 1 \ \textbf{to} \ n-1 \ \textbf{do} \\
\quad b = \beta[i] \\
\quad \textbf{while} \ b>0 \ \textbf{and} \ x[i+1] \neq x[b+1] \ \textbf{do} \\
\quad \quad b = \beta[b] \\
\quad \textbf{if} \ x[i+1] = x[b+1] \ \textbf{then} \\
\quad \quad \quad \beta[i+1] = b+1 \\
\quad \textbf{else} \\
\quad \quad \quad \beta[i+1] = 0
\]

Observations: (1) $b$ is initialized to 0, (2) $b$ is increased by at most 1 in each iteration of the for-loop, (3) $b$ is decreased by at least 1 in every iteration of the while-loop, i.e. at most $n-2$ iterations of the while-loop

Time: $O(n)$, space: $O(n)$ for $x$ plus $O(1)$ additional
Exercise: Borders and suffix trees?

**Rule of thumb:** Anything (almost) related to finding regularities in a string can be solved using a suffix tree.

Can we find (the length of) the longest border using a suffix tree? How long does it take?

Why, or why not, use the suffix tree approach?
**Problem:** Given text $x[1..n]$ and pattern $p[1..m]$, find all occurrences of $p$ in $x$, i.e. all $i$ where $x[i..i+m-1] = p[1..m]$. 
Border arrays and exact pattern matching

Problem: Given text $x[1..n]$ and pattern $p[1..m]$, find all occurrences of $p$ in $x$, i.e. all $i$ where $x[i..i+m-1] = p[1..m]$.

Solution: Can be solved naively in time $O(nm)$. More efficient solutions (both in theory and practice) exists.

Today we will see a (surprisingly) simple solution based on border arrays that has worst-case running time $O(n+m)$, and we will see the classic KMP algorithm that also has worst-case running time $O(n+m)$.

Later we will see the BM and SHIFT-and-OR algorithms that might be faster in practice.
Exact pattern matching: BA algorithm

Step 1: Construct $s[1..n+m+1] = p$x and its borders array $\beta[1..n+m+1]$

Observation 1: No border of $s$ has length more than $m$ because of $\$

Observation 2: $p$ occurs in $x$ at pos. $i-m+1-(m+1) = i-2m$ iff $\beta[i]=m$

Step 2: Report an occurrence of $p$ in $x$ at position $i-2m$ iff $\beta[i]=m$

Time and space $O(n+m)$