Strings, periods, and borders

String Algorithms
A reminder from the first lecture
A formal definition

A string is a collection of elements that obeys the following rules

1. every element has a label that is unique
2. every element with some label \( x \) (except at most one, called the \textit{leftmost}) has a unique determinable \textit{predecessor} labelled \( p(x) \)
3. every element with some label \( x \) (except at most one, called the \textit{rightmost}) has a unique determinable \textit{successor} labelled \( s(x) \)
4. whenever an element with label \( x \) is not leftmost, then \( x = s(p(x)) \)
5. whenever an element with label \( x \) is not rightmost, then \( x = p(s(x)) \)
6. for any two distinct elements with labels \( x \) and \( y \), there exist a positive integer \( k \) such that either \( x = s^k(y) \) or \( x = p^k(y) \)
But ...

... a string is still what it has always been ....

... a finite sequence of elements from some alphabet \( A \),
where the label of an element is its position/index ...

**Definition:** An element of \( A^+ \) is called a *linear string* on alphabet \( A \).
An element of \( A^* \) is called a *finite string* on \( A \).
Notation and terminology

\( x: \text{array} [1..n] \) of \( A \) is a \text{string} on alphabet \( A \) of \text{length} \( n=|x| \).
The \text{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 \text{substring} of \( x \) of length \( j-i+1 \), and a \text{proper substring} if \( j-i+1 < n \). It occurs in \( x \) at position \( i \)

The substring \( x[1..i] \) is a \text{prefix} of \( x \), and a \text{proper prefix} if \( i < n \), and \( x[i..n] \) is a \text{suffix} of \( x \), and a \text{proper suffix} if \( i > 1 \)
Today

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 will be used later in the class for **fast algorithms for exact pattern matching**.
Borders

A border of $x$ is any proper prefix of $x$ that equals a suffix of $x$

- Empty border
- Border $ab$
- Border $abaab$

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

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

1 \hspace{2cm} \beta \hspace{2cm} n
Periodicities

\[ x = (\text{blue part})^3 \]
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
Computing borders

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

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

Running time?
Computing borders

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

\[ \beta = 0 \]

\[ \text{for } i=1 \text{ to } n-1 \text{ do} \]

\[ \text{if } x[1..i] = x[n-i+1..n] \text{ then} \]

\[ \beta = i \]

\[ \text{return } \beta \]

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

... can we do better? Yes, by computing more ...
In the *border array* $\beta[1..n]$ of $x$, entry $\beta[i]$ is the length of the 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 $x[1..i]$

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

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. $\beta[i+1] \leq \beta[i]+1$

**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 ...
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. $\beta[i+1] \leq \beta[i]+1$

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 ...
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 $x[1..i]$
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 $x[1..n]$

\[
\beta[1] = 0 \\
\text{for } i = 1 \text{ to } n-1 \text{ do} \\
\quad b = \beta[i] \\
\quad \text{while } b>0 \text{ and } x[i+1] \neq x[b+1] \text{ do} \\
\quad \quad b = \beta[b] \\
\quad \text{if } x[i+1] = x[b+1] \text{ then} \\
\quad \quad \beta[i+1] = b+1 \\
\quad \text{else} \\
\quad \quad \beta[i+1] = 0
\]
Border array $\beta$ of $x[1..n]$

\[
\begin{align*}
\beta[1] &= 0 \\
\text{for } i = 1 \text{ to } n-1 \text{ do} \\
& \quad b = \beta[i] \\
& \quad \text{while } b > 0 \text{ and } x[i+1] \neq x[b+1] \text{ do} \\
& \quad \quad b = \beta[b] \\
& \quad \text{if } x[i+1] = x[b+1] \text{ then} \\
& \quad \quad \beta[i+1] = b + 1 \\
& \quad \text{else} \\
& \quad \quad \beta[i+1] = 0
\end{align*}
\]

**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 $x[1..n]$**

\[\beta[1] = 0\]

\[\text{for } i = 1 \text{ to } n-1 \text{ do}\]

\[b = \beta[i]\]

\[\text{while } b > 0 \text{ and } x[i+1] \neq x[b+1] \text{ do}\]

\[b = \beta[b]\]

\[\text{if } x[i+1] = x[b+1] \text{ then}\]

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

\[\text{else}\]

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

**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?
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 $x$

```
aabaababaabaabaabaabaabaab
aabaabaab
```

$p = 9$

```
aabaababaab
```
**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 \( x \)

\[
\begin{align*}
\text{aabaab} & \text{aabaabaabaabaabaabaab} \\
\text{aabaabaab} & \text{aabaab} \\
p = 9 & \quad \text{aabaabaab} \\
\text{aabaab} & \text{aabaab} \\
q = 6 & \quad \text{aabaab} \\
\text{aabaab} & \text{aabaab}
\end{align*}
\]
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 $x$.

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