String Algorithms and Data Structures

Z-values and the Z-algorithm

CS 199-225
Brad Solomon

February 6, 2023
Exact Pattern Matching

Find instances of $P$ in $T$

‘instances’: An exact, full length copy
Exact Pattern Matching

What’s a simple algorithm for exact matching?

P: word

T: There would have been a time for such a word
word word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word

Try all possible alignments. For each, check if it matches. This is the naive algorithm.
Exact Pattern Matching

What is good about the naive solution?

What is bad?
Exact Pattern Matching

What is our time complexity? \( (n = |P|, \quad m = |T|) \)

(# of alignments) x (cost of an alignment)
Exact Pattern Matching

What is our time complexity? \( n = |P|, \ m = |T| \)  

\((\# \text{ of alignments}) \times \text{(cost of an alignment)}\)

\(P:\)  

\(T:\)

P can fit at each `position' along \(T\) except the edge
Exact Pattern Matching

What is our time complexity? \( n = |P|, \quad m = |T| \) 

\( (\text{__________}) \times \text{(cost of an alignment)} \)

\( P: \) aaaa

\( T: \) aaaaaaaaaaaaaaaaaa

aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa

There are ______ positions which extend past the edge of T
Exact Pattern Matching

What is our time complexity? \( (n = |P|, \quad m = |T|) \)

\[
(m-n+1) \times \text{(cost of an alignment)}
\]

\(P: \ aaaa\)

\(T: \ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Exact Pattern Matching

What is our time complexity? \( (n = |P|, \ m = |T|) \)

\[ \theta((m - n + 1) \times n) \]
String Algorithms in Genomics

P: Read (n = ~50-150)

T: Reference (m = ~3 billion)
String Algorithms in Genomics
Improving exact pattern matching

How can we do better than the naïve algorithm?

… If we have infinite space?

… If I tell you the pattern ahead of time?

… If I tell you the text ahead of time?
Exact Pattern Matching with Z-algorithm

Pattern, $P$  
Text, $T$

Naive $\approx \theta(|P| + |T|)$  
Z-Algorithm $\approx \theta(|P| + |T|)$

Find instances of $P$ in $T$

‘instances’: An exact, full length copy
The Z-value \([ Z_i(S) ]\)

Given a string \(S\), \(Z_i(S)\) is the length of the longest substring in \(S\), starting at position \(i\), that matches a prefix of \(S\).

\[
\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 & \quad 6 & \quad 7 & \quad 8 & \quad 9 \\
S: & \quad T & \quad T & \quad C & \quad G & \quad T & \quad T & \quad A & \quad G & \quad C & \quad G \\
Z_0(S) & = & \quad Z_3(S) & = \\
Z_1(S) & = & \quad Z_4(S) & = \\
Z_2(S) & = & \quad Z_5(S) & =
\end{align*}
\]
The Z-value \([ Z_i(S) ]\)

Given a string \(S\), \(Z_i(S)\) is the length of the longest substring in \(S\), starting at position \(i\), that matches a prefix of \(S\).

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
S: & T & T & C & G & T & T & A & G & C & G \\
\end{array}
\]

\[
\begin{align*}
Z_0(S) &= 10 \\
Z_1(S) &= 1 \\
Z_2(S) &= 0 \\
Z_3(S) &= \_ \\
Z_4(S) &= \_ \\
Z_5(S) &= \_
\end{align*}
\]
The Z-value \([ Z_i(S) ]\)

Given a string \(S\), \(Z_i(S)\) is the length of the longest substring in \(S\), starting at position \(i > 0\), that matches a prefix of \(S\).

\[
\begin{align*}
S &: \quad \text{T T C G T T A G C G} \\
Z_0(S) &= 10 \\
Z_1(S) &= 1 \\
Z_2(S) &= 0 \\
Z_3(S) &= 0 \\
Z_4(S) &= 2 \\
Z_5(S) &= 1
\end{align*}
\]
Calculating the Z-values

**Naive:** Compute the Z-values by *explicitly* comparing characters (left-to-right scan):

\[ Z_1 = \]

\[
\begin{array}{c}
A A A A B A A C A A B A A \\
\end{array}
\]

\[
\begin{array}{c}
A A A A B A A C A A B A A \\
\end{array}
\]

\[ Z_5 = \]

\[
\begin{array}{c}
A A A A B A A C A A B A A \\
\end{array}
\]

\[
\begin{array}{c}
A A A A B A A C A A B A A \\
\end{array}
\]

*What is our time complexity?*
Calculating the Z-values

**Naive**: Compute the Z-values by *explicitly* comparing characters (left-to-right scan):

```
S: 1 1 0 1 1 0 0 1
```

What is our time complexity?
Calculating the Z-values

**Naive:** Compute the Z-values by *explicitly* comparing characters (left-to-right scan):

\[
S : 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \\
1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \\
0 \ 1 \ 1 \ 0 \ 0 \ 1 \\
1 \ 1 \ 0 \ 0 \ 1 \\
1 \ 0 \ 0 \ 1 \\
1 \ 0 \ 0 \ 1 \\
0 \ 0 \ 1 \\
0 \ 1 \\
0 \ 1 \\
1
\]

*What is our time complexity?*
Pattern matching with the Z-value

Given a $Z_i$ value calculator, how do we solve pattern matching?

Pattern, $P$  
Text, $T$

Calculate Z-values

Z-algorithm

Find instances of $P$ in $T$
Z-value Pattern Matching

To solve pattern matching (given $P$ and $T$), let $S = P$T

$S = \text{‘terminal character’, outside alphabet}$
Z-value Pattern Matching

To solve pattern matching (given $P$ and $T$), let $S = P$\$T$

$\$ = ‘terminal character’, outside alphabet

\[
\begin{align*}
S & = P$T \\
S & = A A$A A A A
\end{align*}
\]

$Z(S) = [\_, \_, \_, \_, \_, \_, \_, \_, \_ ]$
Z-value Pattern Matching

To solve pattern matching (given $P$ and $T$), let $S = P$\$T$

$\$ = ‘terminal character’, outside alphabet

$P: \text{A A}$ \hspace{1cm} $T: \text{A A A A}$

$0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6$

$S: \text{A A }\text{}\text{\$ A A A A}$ \hspace{1cm} $Z(S) = [-, 1, 0, 2, 2, 2, 1 ]$

$0 \ 1 \ 2 \ 3$

What $Z_i$ values are matches?

What are the matching indices in $T$?
Z-value Pattern Matching

\( P: \ T \ T \ \ T: \ C \ T \ T \ A \)

\( S: \)

\( Z(S): \)

Z-value search pseudo-code

1. Concatenate \( (S=P$T) \)

2. Calculate Z-values for \( S \)

3. For \( i<0 \), match if \( Z_i = \) __________
   Match is not at \( i \), but instead at ________________
Assignment 2: a_zval

Learning Objective:

Construct a Z-value calculator and measure its efficiency

Demonstrate use of Z-values in pattern matching

Consider: Our goal is $\theta(|P| + |T|)$. Does Z-value search match this?
End-of-class brainstorm

What information does a single Z-value tell us?

If I know $Z_{i-1}(S)$, can I use that information to help me compute $Z_i(S)$?
The Z-value (Take 2)

Given a string $S$, $Z_i(S)$ is the length of the longest substring in $S$, starting at position $i$, that matches a prefix of $S$.

What information does this give us?

$S$: TTCGTTAGCG

$Z_4 = 2$
The Z-value (Take 2)

Given a string \( S \), \( Z_i(S) \) is the length of the longest substring in \( S \), starting at position \( i \), that matches a prefix of \( S \).

What information does this give us?

\[
S: \quad \text{TTCCGTTAGCG} \quad \quad Z_4 = 2
\]
The Z-value (Take 2)

Given a string $S$, $Z_i(S)$ is the length of the longest substring in $S$, starting at position $i$, that matches a prefix of $S$.

What information does this give us?

$S$: $\text{TTCGTTAGCG}$

$Z_4 = 2$
The Z-Algorithm

Assume we’ve computed $Z_1, \ldots, Z_{i-1}$ and need to calculate $Z_i$

**Case 1:** We know nothing about the characters at $S[i]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z_1$ = ?</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

**Case 2:** We know something about the characters at $S[i]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z_2$ = ?</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>
The Z-Algorithm

We track our current knowledge of $S$ using three values: $i$, $r$, $l$

$i$, the current index position being calculated

$r$, the index of the rightmost character which has ever been matched

$l$, the index of Z-value which $r$ belongs too

$Z_1 = 3$

$Z_2 = ?$
The Z-Algorithm

\[ i, \text{ the current index } = \]

\[ r, \text{ the furthest match char } = \]

\[ l, \text{ the furthest reaching Z-value } = \]

<table>
<thead>
<tr>
<th>-</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

Start End
The Z-Algorithm

\( i, \) the current index =

\( r, \) the furthest match char =

\( l, \) the furthest reaching Z-value =

\[
\begin{array}{cccccccc}
- & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & 1 & A & A & B & B & A & A \\
\end{array}
\]
The Z-Algorithm

\( i \), the current index =

\( r \), the furthest match char =

\( l \), the furthest reaching Z-value =

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>l</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>
The Z-Algorithm

\(i\), the current index = 

\(r\), the furthest match char = 

\(l\), the furthest reaching Z-value =

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
The Z-Algorithm

\( i \), the current index = \\
\( r \), the furthest match char = \\
\( l \), the furthest reaching Z-value = 

\[
\begin{array}{cccccccc}
- & 1 & 0 & 0 & 3 & \_ & \_ & \_ \\
0 & 1 & 2 & 3 \_ & 4 & 5 & 6 & 7 \\
\end{array}
\]
The Z-Algorithm

\( i \), the current index =

\( r \), the furthest match char =

\( l \), the furthest reaching Z-value =
The Z-Algorithm

\( i \), the current index =

\( r \), the furthest match char =

\( l \), the furthest reaching Z-value =

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>B</th>
<th>A</th>
<th>A</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>
The Z-Algorithm

**Intuition:** We can use the previous $Z_1, \ldots, Z_i$ to compute $Z_{i+1}$!

Track ‘what we know’ using three integers: $i, r, l$

Next week: Review how integers are updated to define specific cases.