String Algorithms and Data Structures

Z-values and the Z-algorithm

CS 199-225
Brad Solomon

September 12, 2022
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

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

What is good about the naive solution?

What is bad?
Exact Pattern Matching

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

\((# \ of \ alignments) \times (cost \ of \ an \ alignment)\)
Exact Pattern Matching

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

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

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

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

\((\phantom{________}) \times \text{(cost of an alignment)}\)

\(P: aaaa\)

\(T: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\)

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

\( T: \) aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

  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

Each alignment compares \_______\ characters.
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)
CTCAAACTCCGTACCTTGGTATCCACCGCCTAGGCCTTC

T: Reference (m = ~3 billion)
GATCACAGGTCATATCACCTATTACCACTACCGGAGGCTCCTCCATGCATTTTGGTATTTT
CGTCTGGGGGATAGCAGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTC
GCAGTAGCTGGGTAGTTCTGATTTGCTCCTGACTATTTATATTATGACACTTAGCTTTAT
ACAGGGCACCACCTCCTACTTTAATTGATTATTAlerGCTTGTAGAGCAATATAATA
ACATTGATGATGCTGACCGCCACTTCTCACACACAGACATCTAAACATATTACATTTTTAC
ACAGGCAAACCTATACACCCGCTACCTCAGATTTTGGCGGTATGCACGTTTTAACAGTCAC
CTCAAACTCCAACCCGCTATCTCCACCCGGCACTACCCATTTTATACCTTCTCTCCAA
CTCAGCTTCTACTTATAGCTTCTAGAGATACATTACAACACACCGCTTCCCCTTCTGGT
TCACCCTCTAAATCACCAGTACAATGAAACACCGATCACAGCACGACATCAGCTAAAG
ACGTTAGGTCAAGGTGTAGCCCATGAGGTGGCAAGAAATGGGCTACATTTTCTACCCAG
AAAACTACGATAGCCCTTATGAAACTTAAGGGTCGAAGGTGGATTTAGCAGTAAACTAAG
AGTAGAGTGCTAGTTGAACAGGGCCCTGAAGCGCGTACACACCGCCCGTCACCCTCCTC
AAGTATACTTCAAAGGACATTTAACTAAAACCCCTACGCATTTATATAGAGGAGACAGT
CGTAACCTCAAACTCCTGCTTTGGTGATCCACCCGCCTTAGGCCTTC

CCTCAAACTCCTGACCTTTGGTGATCCACCCGCCTTGGCCTACCTGCTAAATGAAGAAGCACCCAACTTACACTTAGGAGATTTCAACTTAACTTGACCGCTCTGAGCTAAACCTA
String Algorithms in Genomics
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 \textbf{w/ Z-algorithm}

- **Pattern, \( P \)**
- **Text, \( T \)**

Naive \( \approx \Theta(|P| \cdot |T|) \)

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 \(S\).

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

\[
\begin{align*}
Z_0(S) &= \\
Z_1(S) &= \\
Z_2(S) &= \\
Z_3(S) &= \\
Z_4(S) &= \\
Z_5(S) &= \\
\end{align*}
\]
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 \( i \), that matches a prefix of \( S \).

\[
\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 & \quad 6 & \quad 7 & \quad 8 & \quad 9 \\
S & : & T & T & C & G & T & T & A & G & C & G \\
Z_0(S) & = & 10 & & & & & \text{ } & & \text{ } & & \text{ } \\
Z_1(S) & = & 1 & & & & & \text{ } & & \text{ } & & \text{ } \\
Z_2(S) & = & 0 & & & & & \text{ } & & \text{ } & & \text{ } \\
Z_3(S) & = & & & & & \text{ } & & \text{ } & & \text{ } \\
Z_4(S) & = & & & & & \text{ } & & \text{ } & & \text{ } \\
Z_5(S) & = & & & & & \text{ } & & \text{ } & & \text{ } \\
\end{align*}
\]
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 \(S\).

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
S: & T & T & C & G & T & T & A & G & C & G \\
Z_0(S) = 10 & Z_3(S) = 0 \\
Z_1(S) = 1 & Z_4(S) = 2 \\
Z_2(S) = 0 & Z_5(S) = 1 \\
\end{array}
\]
Calculating the Z-values

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

$$Z_1 =$$

```
AAAABAACAABAA...
```

$$Z_5 =$$

```
AAAABAACAABAA...
```

What is our time complexity?
Calculating the Z-values

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

```
S: 11011001
```
Calculating the Z-values

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

\[
S : \begin{array}{c}
1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 \\
0 & 1 \\
0 & 1 \\
1 &
\end{array}
\]

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

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

Pattern, $P$  \quad 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 $S = P \cdot T$

$\$ = ‘terminal character’, outside alphabet

\[
S = P \cdot T
\]

\[
P: \text{ A A } \quad T: \text{ A A A A}
\]

\[
S: \text{ A A } \$ \text{ A A A A}
\]
Z-value Pattern Matching

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

$\$ = ‘terminal character’, outside alphabet

\[
S = P$T
\]

\[
P: \quad \text{A A} \quad T: \quad \text{A A A A}
\]

\[
S: \quad \text{A A }\$\text{A A A A}
\]

Z-values

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

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

$\$ = ‘terminal character’, outside alphabet

$P$: A A  $T$: A A A A

0 1 2 3 4 5 6

$S$: A A $\$ A A A A

0 1 2 3

$Z(S) = [-, 1, 0, 2, 2, 2, 1 ]$

What $Z_i$ values are matches?

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

\( P: \quad T \quad T \quad T \quad T \quad A \quad T \quad T \quad A \)

\( T: \quad C \quad T \quad T \quad A \quad T \quad T \quad A \quad T \quad A \)

\( S: \)

\( Z(S): \)

**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 $Z_4(S) = 2$
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?

$S$: TCGTTAGCG $\quad Z_4 = 2$
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{TTCGTATTAGCG}

$Z_4 = 2$
The Z-Algorithm

Assume we’ve computed $Z_0, \ldots, Z_{i-1}$ 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>A</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>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

A red box indicates a change in the pattern.
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

<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>$Z_2$</td>
<td>$A$</td>
<td>$A$</td>
<td>$A$</td>
<td>$A$</td>
<td>$A$</td>
<td>$B$</td>
<td>$B$</td>
<td>$B$</td>
</tr>
</tbody>
</table>

$Z_1 = 3$

$Z_2 = ?$
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>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 =

<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></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></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>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>0</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></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 =

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

*i*, the current index =

*r*, the furthest match char =

*l*, the furthest reaching Z-value =

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

Start   End
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>3</th>
<th>1</th>
<th>0</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>
</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>
</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>
</tr>
</tbody>
</table>
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>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>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

**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.