String Algorithms and Data Structures

Z-values and the Z-algorithm

CS 199-225
Brad Solomon

September 16, 2024

UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

Department of Computer Science

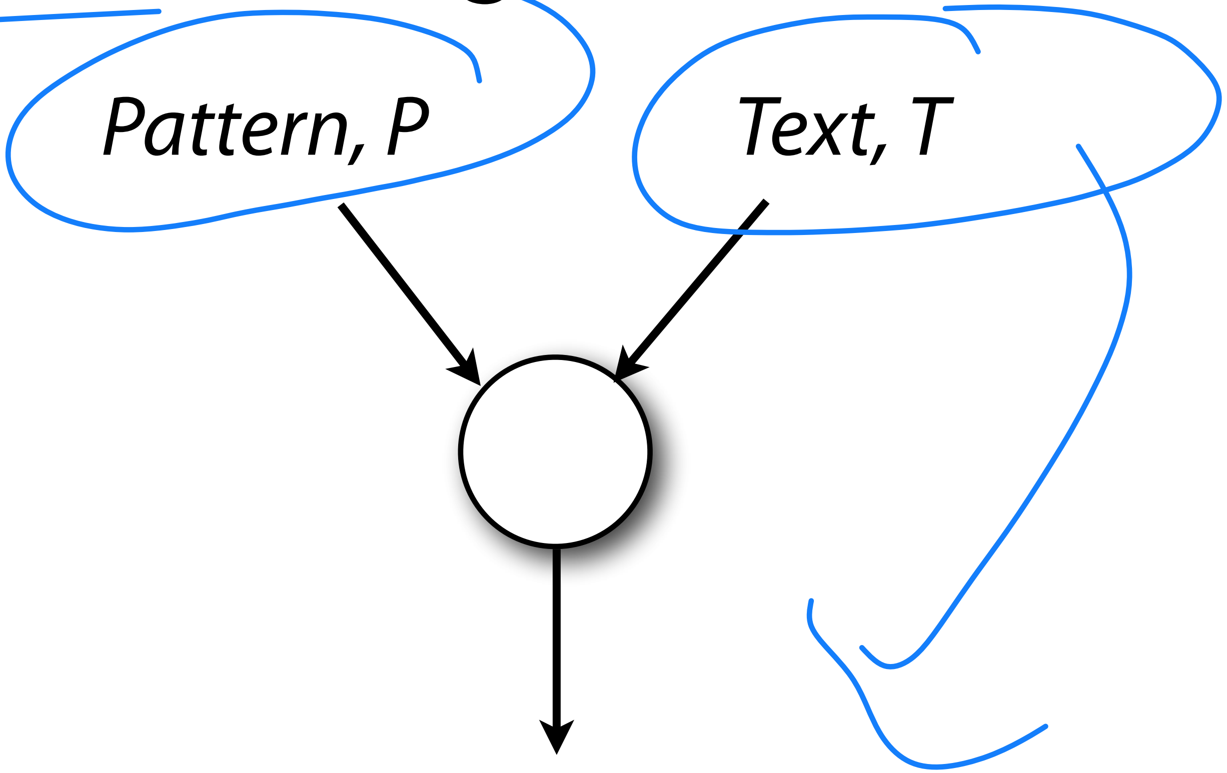
B B B B

A A

A A

A A

Exact Pattern Matching



Pattern, P

Text, T

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
```

One occurrence

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?

↳ Easy to think about / implement

↳ O(1) space

↳ Finds all matches (correct)

What is bad?

↳ Inefficient! (Slow) ← "Bad"

Exact Pattern Matching

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

Exact Pattern Matching

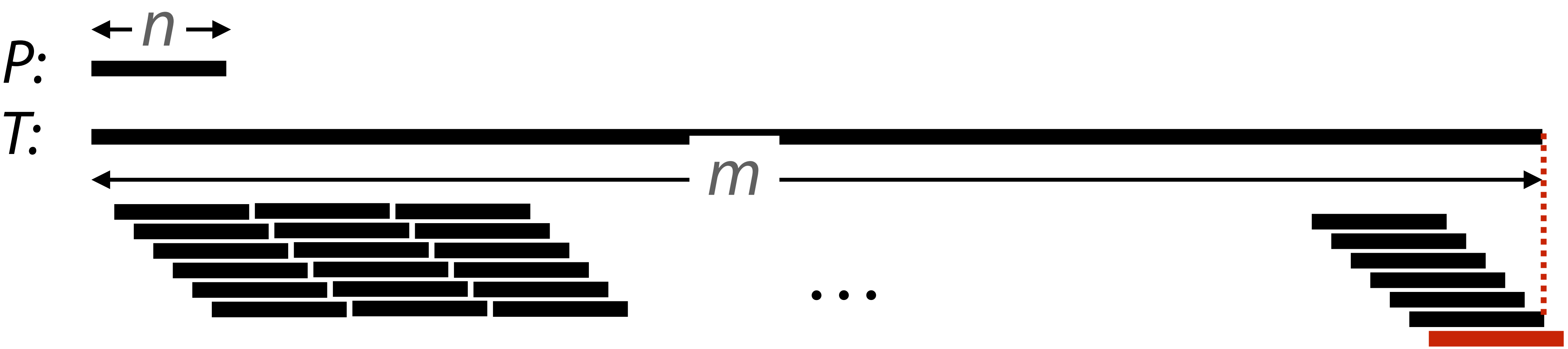
What is our time complexity? $(n = |P|, \quad m = |T|)$

(# of alignments) x (cost of an alignment)

Exact Pattern Matching

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

(# of alignments) x (cost of an alignment)



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

Exact Pattern Matching

p = 4

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

(m - n + 1) x (cost of an alignment)

n-1 characters

P: **aaaa**

xxx

T: **aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa**

```
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
```

3

There are n-1 positions which extend past the edge of T

Exact Pattern Matching

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

(m-n+1) x (cost of an alignment)

$m - (n-1)$

$= m-n+1$

P: **aaaa**

T: **aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa**

m

xxx

$(n-1)$

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 ____n____ characters.

Exact Pattern Matching

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

$$\theta\big((m - n + 1) \times n\big)$$

$$O\left(m \cdot n - n^2 + n\right)$$

↓

Bad ☹

$$O\left(m \cdot n\right)$$

$m >> n$

String Algorithms in Genomics

P: Read (n = ~50-150)

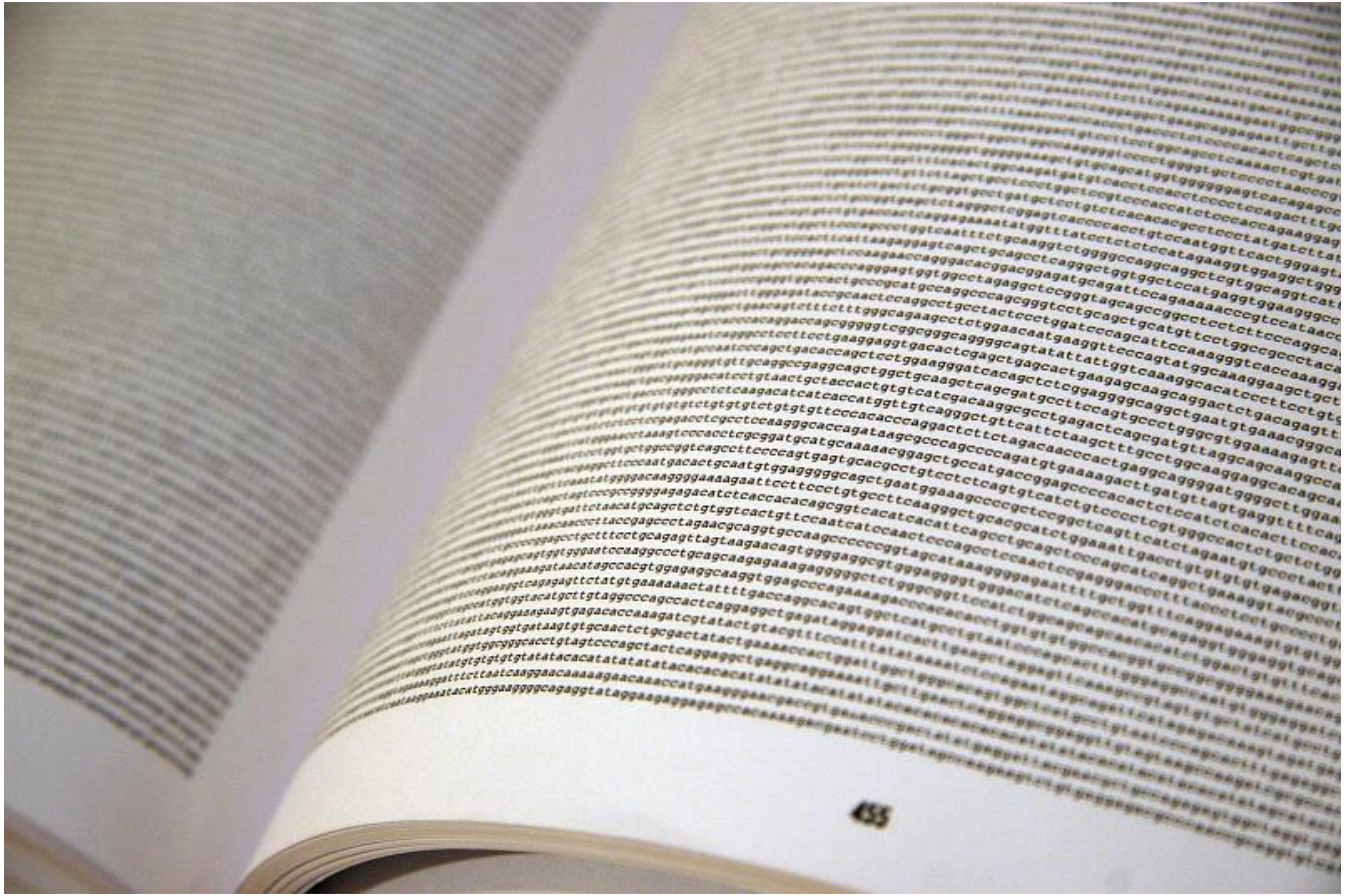
CTCAAACTCCTGACCTTTGGTGATCCACCCGCCTAGGCCTTC

42

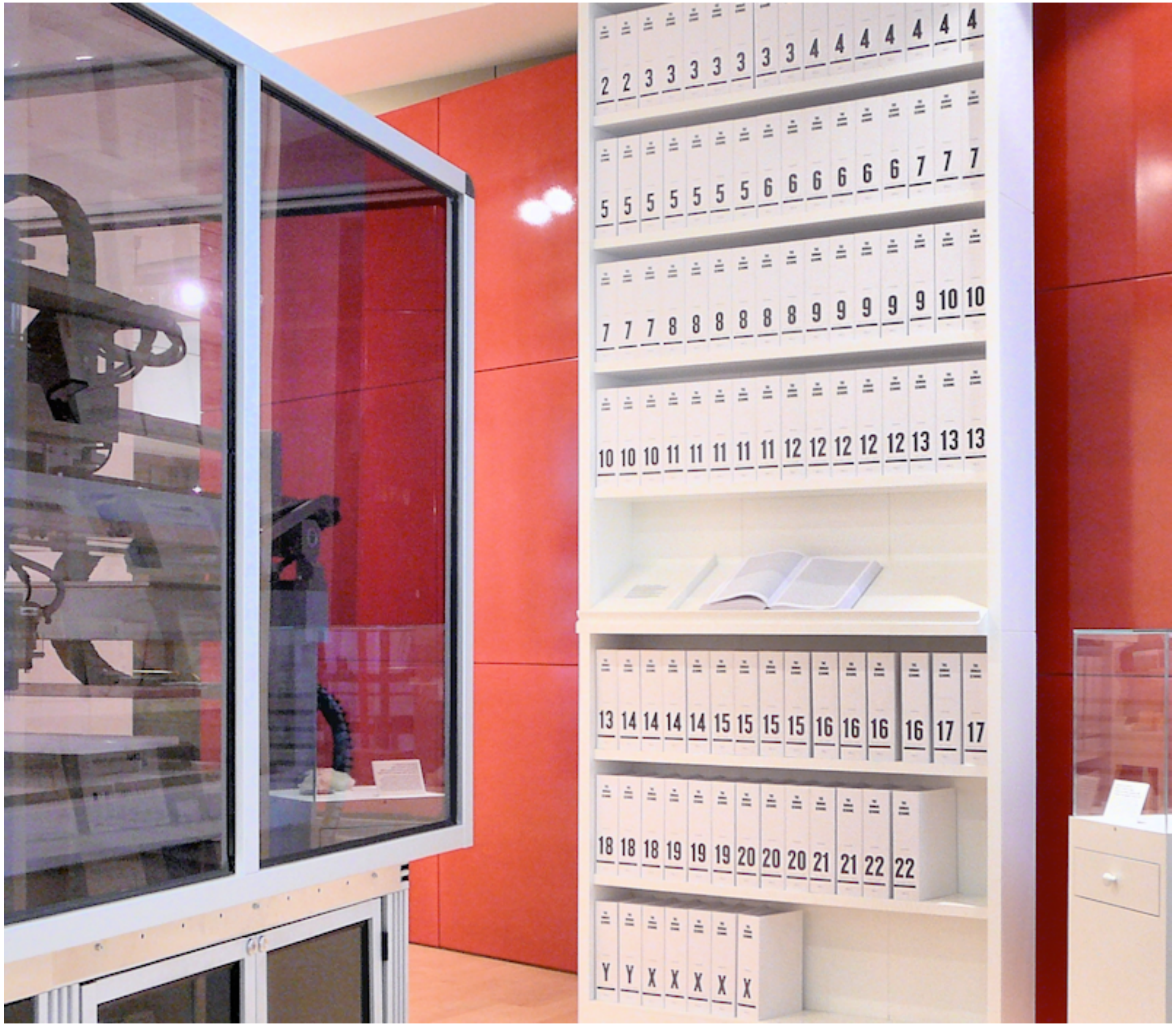
T: Reference (m = ~3 billion)

```
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTT
CGTCTGGGGGGTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTC
GCAGTATCTGTCTTTGATTCCTGCCTCATCCTATTATTTATCGCACCTACGTTCAATATT
ACAGGCGAACATACTTACTAAAGTGTGTTAATTAATTAATGCTTGTAGGACATAATAATA
ACAATTGAATGTCTGCACAGCCACTTTCCACACAGACATCATAACAAAAAATTTCCACCA
AACCCCCCCTCCCCCGCTTCTGGCCACAGCACTTAAACACATCTCTGCCAAACCCCAAAA
ACAAAGAACCCTAACACCAGCCTAACCAGATTTCAAATTTTATCTTTTGGCGGTATGCAC
TTTTAACAGTCACCCCCCAACTAACACATTATTTTCCCCTCCCACTCCCATACTACTAAT
CTCATCAATACAACCCCCGCCCATCCTACCCAGCACACACACACCGCTGCTAACCCCATA
CCCCGAACCAACCAAACCCCAAAGACACCCCCCACAGTTTATGTAGCTTACCTCCTCAAA
GCAATACACTGACCCGCTCAAACTCCTGGATTTTGGATCCACCCAGCGCCTTGGCCTAAA
CTAGCCTTTCTATTAGCTCTTAGTAAGATTACACATGCAAGCATCCCCGTTCCAGTGAGT
TCACCCTCTAAATCACCACGATCAAAAGGAACAAGCATCAAGCACGCAGCAATGCAGCTC
AAAACGCTTAGCCTAGCCACACCCCCACGGGAAACAGCAGTGATTAACCTTTAGCAATAA
ACGAAAGTTTAACTAAGCTATACTAACCCCAGGGTTGGTCAATTTCGTGCCAGCCACCGC
GGTCACACGATTAACCCAAGTCAATAGAAGCCGGCGTAAAGAGTGTTTTAGATCACCCCC
TCCCCAATAAAGCTAAAACTCACCTGAGTTGTAAAAAACTCCAGTTGACACAAAATAGAC
TACGAAAGTGGCTTTAACATATCTGAACACACAATAGCTAAGACCCAAACTGGGATTAGA
TACCCCACTATGCTTAGCCCTAAACCTCAACAGTTAAATCAACAAAACTGCTCGCCAGAA
CACTACGAGCCACAGCTTAAAACTCAAAGGACCTGGCGGTGCTTCATATCCCTCTAGAGG
AGCCTGTTCTGTAATCGATAAACCCCGATCAACCTCACCACCTCTTGCTCAGCCTATATA
CCGCCATCTTCAGCAAACCCTGATGAAGGCTACAAAGTAAGCGCAAGTACCCACGTAAAG
ACGTTAGGTCAAGGTGTAGCCCATGAGGTGGCAAGAAATGGGCTACATTTTCTACCCCAG
AAAACTACGATAGCCCTTATGAAACTTAAGGGTCGAAGGTGGATTTAGCAGTAAACTAAG
AGTAGAGTGCTTAGTTGAACAGGGCCCTGAAGCGCGTACACACCGCCCGTCACCCTCCTC
AAGTATACTTCAAAGGACATTTAACTAAAACCCCTACGCATTTATATAGAGGAGACAAGT
CGTAACCTCAAACTCCTGCCTTTGGTGATCCACCCGCCTTGGCCTACCTGCATAATGAAG
AAGCACCCAACTTACACTTAGGAGATTTCAACTTAACTTGACCGCTCTGAGCTAAACCTA
```

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?

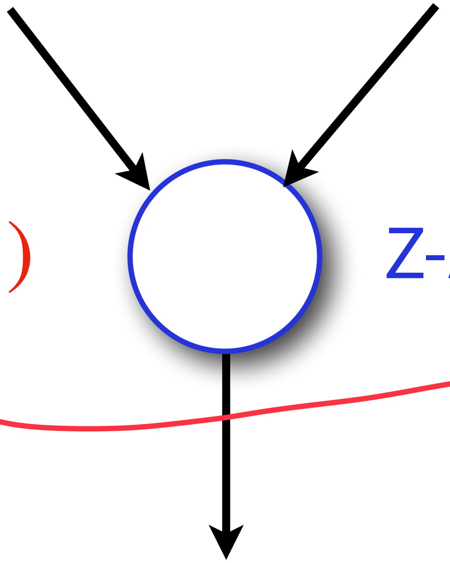
… If I tell you the pattern ahead of time?

… If I tell you the text ahead of time?

Exact Pattern Matching w/ 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.

0 1
T T
T C
1 2

	0	1	2	3	4	5	6	7	8	9
S:	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

$Z_4(S) =$

$Z_5(S) =$

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*.

	0	1	2	3	4	5	6	7	8	9
S:	**T**	**T**	**C**	**G**	**T**	**T**	**A**	**G**	**C**	**G**

$Z_0(S) = 10$

$Z_1(S) = 1$

$Z_2(S) = 0$

$Z_3(S) = 0$

$Z_4(S) = 2$

$Z_5(S) = 1$

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.

```
    0 1 2 3 4 5 6 7 8 9
S:  T T C G T T A G C G
```

~~$Z_0(S) = 10$~~

$Z_1(S) = 1$

$Z_2(S) = 0$

$Z_3(S) = 0$

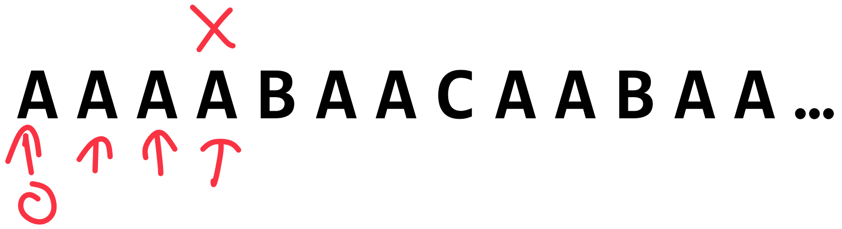
$Z_4(S) = 2$

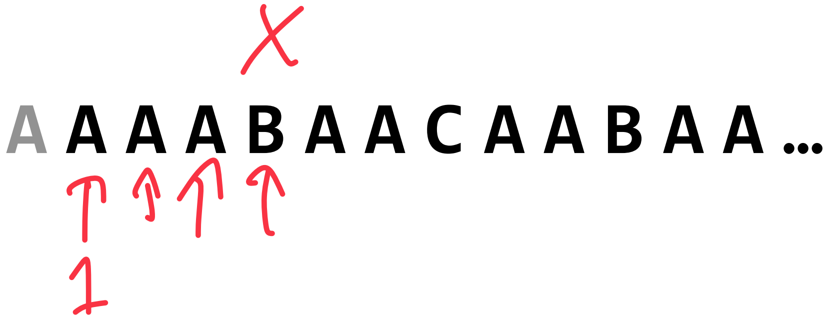
$Z_5(S) = 1$

Calculating the Z-values

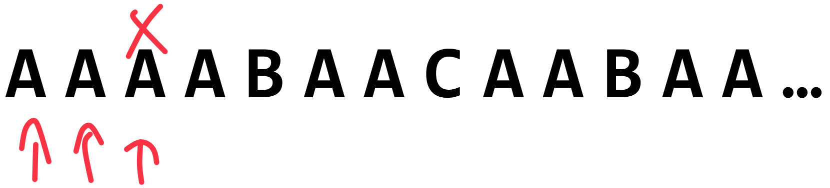
Naive: Compute the Z-values by *explicitly* comparing characters (left-to-right scan):

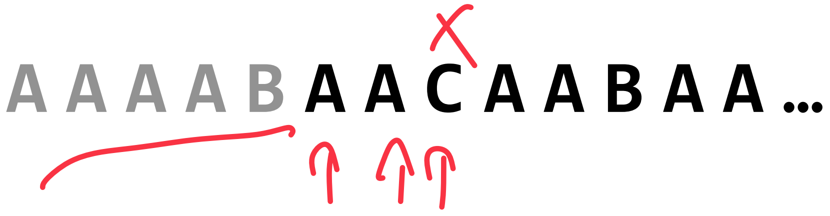
$Z_1 =$ 3


A A A A B A A C A A B A A …


A A A A B A A C A A B A A …

$Z_5 =$ 2


A A A A B A A C A A B A A …

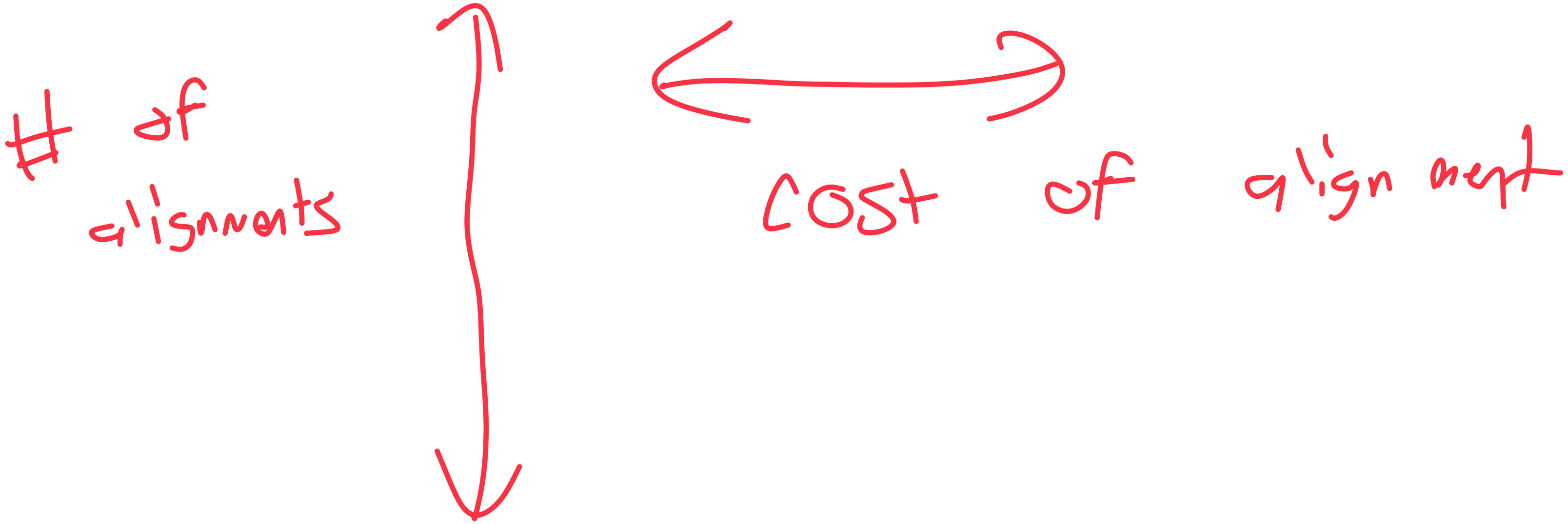

A A A A B A A C A A B A A …

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: 11011001      |S|-1
   1011001   7   cost of alignment
   011001    6   |S|-2
   11001     5
   1001      4
   001       3
   01        2
   1         1
```

# of alignments

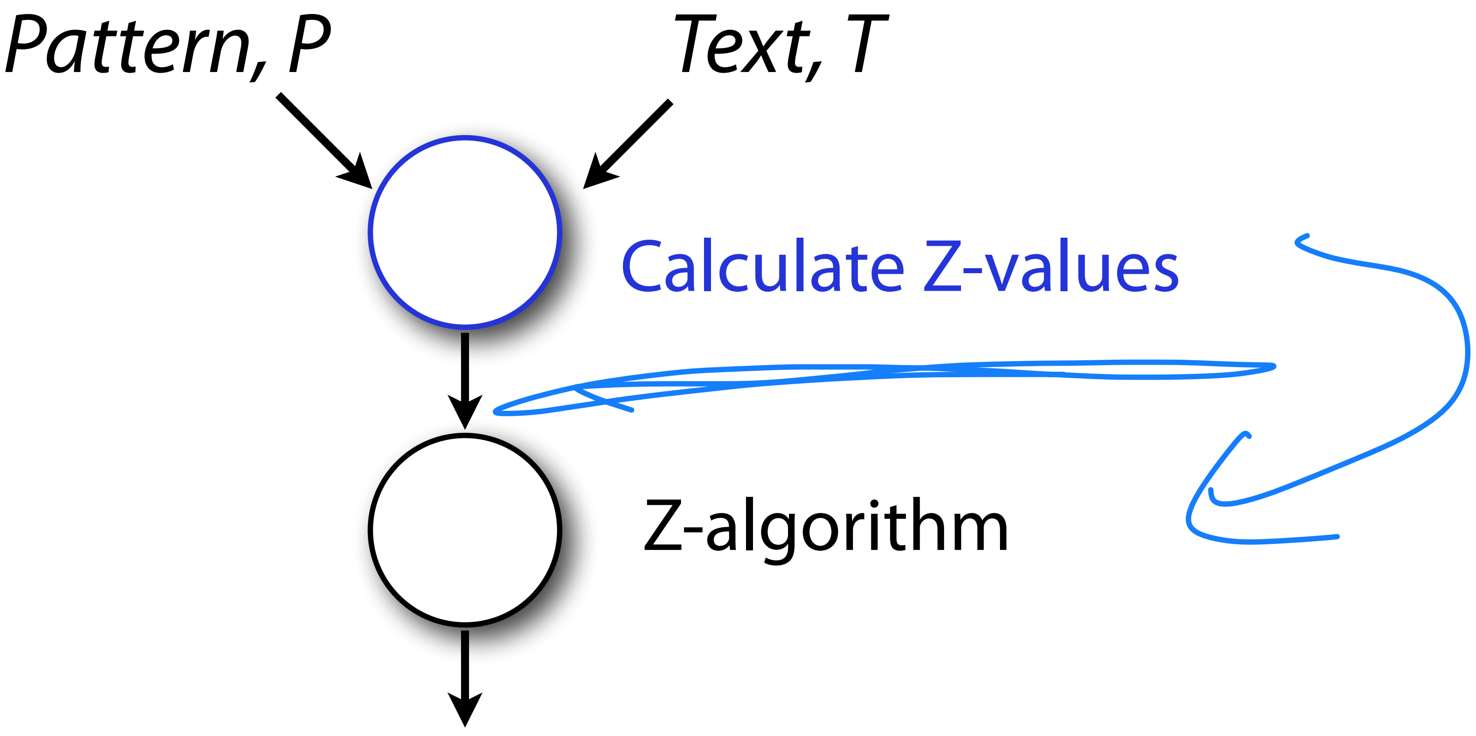
$$\frac{S(S-1)}{2}$$

What is our time complexity?

$O(S^2)$:(

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

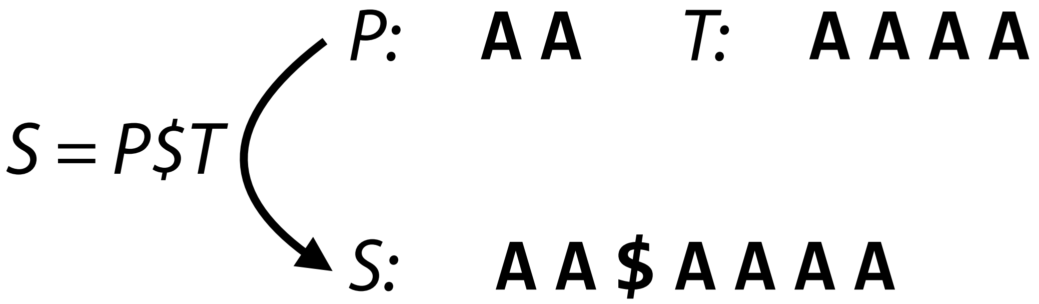
Find instances of *P* in *T*

Z-value Pattern Matching

$ Cant match anything!

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

$ = 'terminal character', outside alphabet



S = *P*$*T*

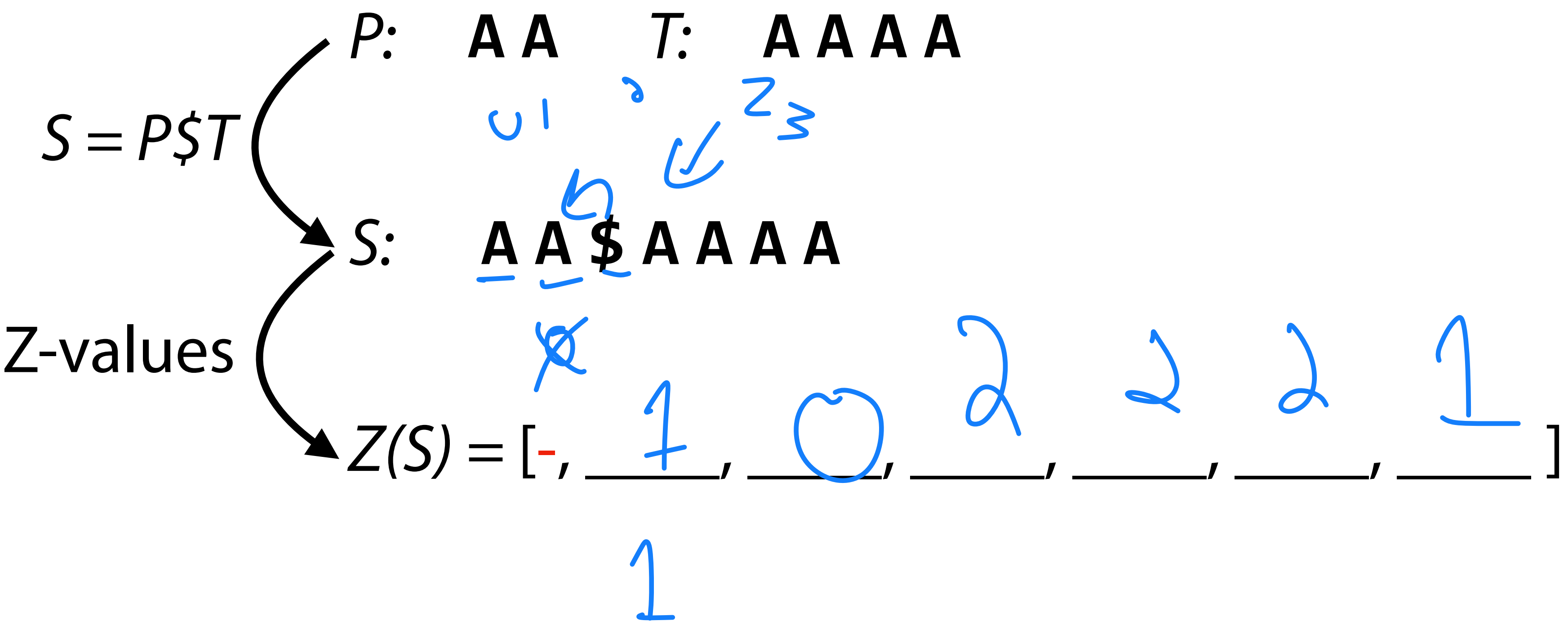
P: **A A** *T*: **A A A A**

S: **A A $ A A A A**

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**
S = *P$T*
S: **A A $ A A A A**
Z-values
Z(S) = [-, 1, 0, 2, 2, 2, 1]
u1
2
Z3
Ø
1

Z-value Pattern Matching

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

$ = 'terminal character', outside alphabet

is our terminal character

P: **A A** *T*: **A A A A**

i - |P| - 1

```
    0 1 2 3 4 5 6
S:  A A $ A A A A
          0 1 2 3
```

3, 4, 5

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

What Z_i values are matches?

Any value w/ 2

What are the matching indices in *T*?

Z-value Pattern Matching

P: **T T** *T:* **C T T A**

0 1

S: T T $ C T T A

0 1 2 3 4

Z(S): [-, 1, 0, 0, 2, 1, 0]

n

4 - 2 - 1 = 1

Z-value search pseudo-code

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

2. *Calculate Z-values for S*

3. For i ~~<~~ > 0, match if Z_i = |P|

Match is ***not*** at i, but instead at

T[i - |P| - 1]

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?

0 1 2 3 4

S: [redacted]

$Z_4 = 2$

A ≠ B

0	1	2	3	4	5	6	7	8	9	...
X	Y	A		X	Y	B				

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

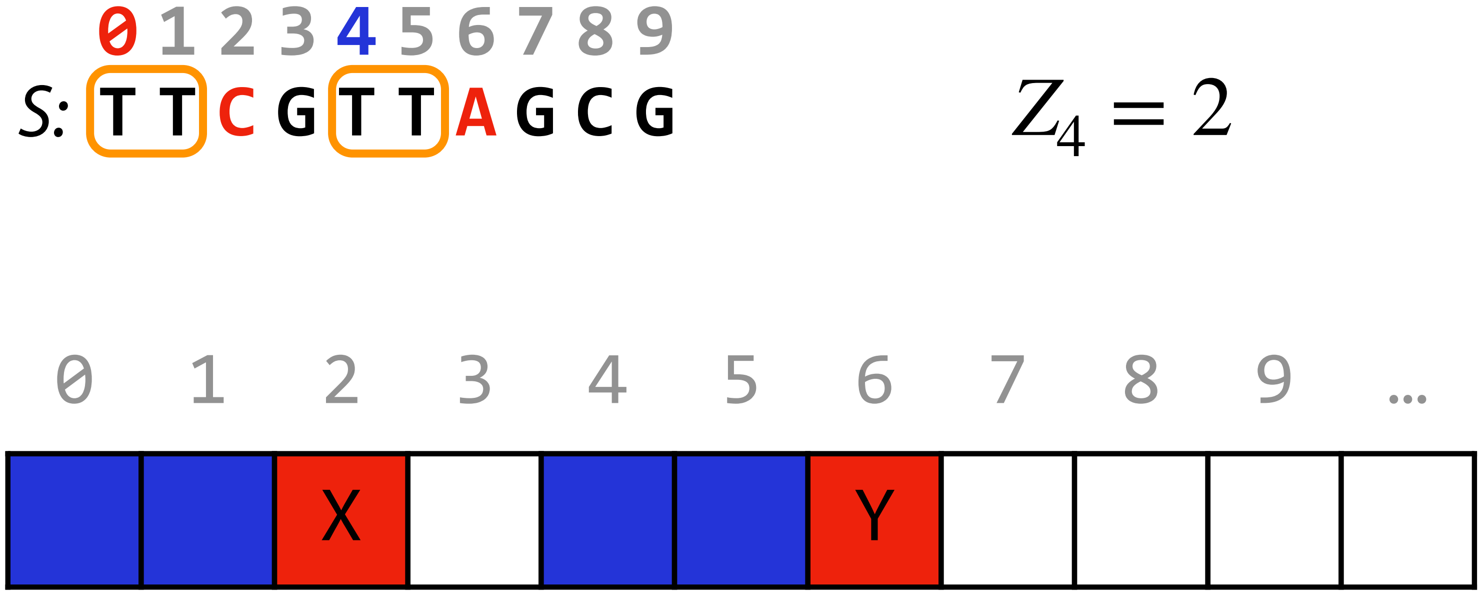
$$Z_4 = 2$$

0	1	2	3	4	5	6	7	8	9	…
🟦	🟦	X		🟦	🟦	Y				

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?



0 1 2 3 4 5 6 7 8 9

S: T T C G T T A G C G

$$Z_4 = 2$$

0	1	2	3	4	5	6	7	8	9	…
		X				Y				

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]

$Z_1 = ?$

0	**1**	2	~~3~~	~~4~~	5	6	7
A	A	A	A	B	B	B	B
A	A	A	A	B	B	B	B

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

$Z_2 = ?$

0	1	**2**	3	4	5	6	7
A	A	A	A	B	B	B	B
A	A	A	A	B	B	B	B

Z_1

Z_2?

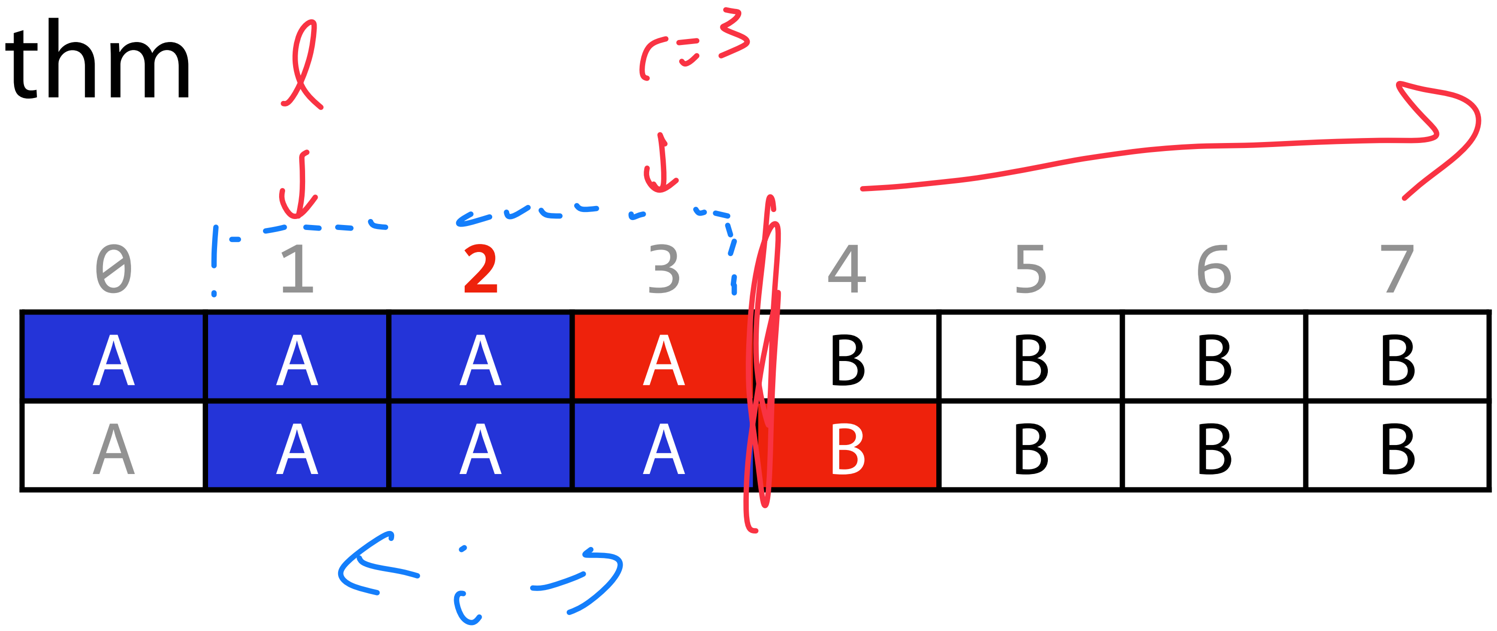
The Z-Algorithm

$l=1$

$Z_1 = 3$

$Z_2 = ?$

$i=2$



0	1	2	3	4	5	6	7
A	A	A	A	B	B	B	B
A	A	A	A	B	B	B	B

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

The Z-Algorithm

	Start	End
i, the current index =	1	2
r, the furthest match char =	0	1
l, the furthest reaching Z-value =	0	1

-	1						
0	1	2	3	4	5	6	7
A	A	B	B	A	A	B	A
A	A	B	B	A	A	B	A

The Z-Algorithm

	Start	End
i, the current index =	2	3
r, the furthest match char =	1	1
l, the furthest reaching Z-value =	1	1

-	1	___	___	___	___	___	___
0	1	2	3	4	5	6	7
A	A	B	B	A	A	B	A
A	A	B	B	A	A	B	A

Don't change r (and l) if no match

The Z-Algorithm

	Start	End
i, the current index =	3	4
r, the furthest match char =	1	1
l, the furthest reaching Z-value =	1	1

-	1	0	0	___	___	___	___
0	1	2	3	4	5	6	7
A	A	B	B	A	A	B	A
A	A	B	B	A	A	B	A

The Z-Algorithm

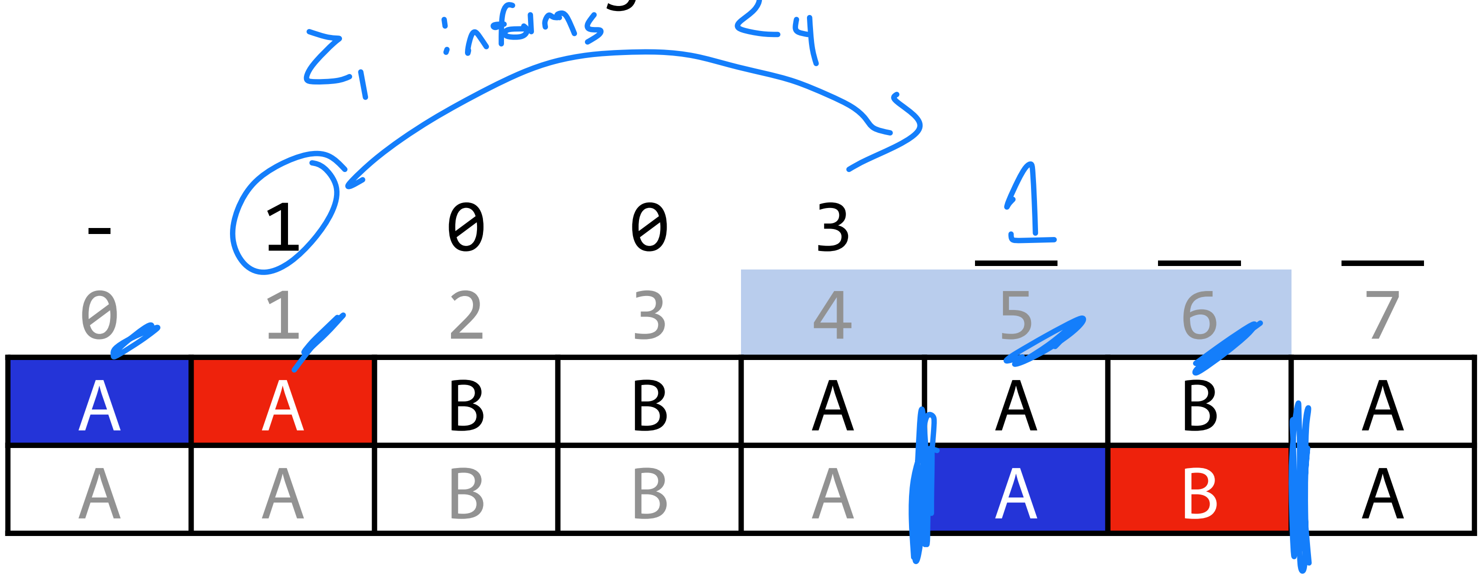
	Start	End
i, the current index =	4	5
r, the furthest match char =	1	6
l, the furthest reaching Z-value =	1	4

$z_4 = 3$

-	1	0	0	__	__	__	__
0	1	2	3	4	5	6	7
A	A	B	B	A	A	B	A
A	A	B	B	A	A	B	A

The Z-Algorithm

	Start	End
i, the current index =	5	6
r, the furthest match char =	6	6
l, the furthest reaching Z-value =	4	4



-	1	0	0	3	1	___	___
0	1	2	3	4	5	6	7
A	A	B	B	A	A	B	A
A	A	B	B	A	A	B	A

Cool step

↳ No work needed!

The Z-Algorithm

	Start	End
i, the current index =	7	8
r, the furthest match char =	6	7
l, the furthest reaching Z-value =	4	7

-	1	0	0	3	1	0	2
0	1	2	3	4	5	6	7
A	A	B	B	A	A	B	A
A	A	B	B	A	A	B	A

The Z-Algorithm

Start End

i, the current index =

r, the furthest match char =

l, the furthest reaching Z-value =

-	1	0	0	3	1	0	1
0	1	2	3	4	5	6	7
A	A	B	B	A	A	B	A
A	A	B	B	A	A	B	A

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.

↳ Z-alg never compares same character twice!

$O(|P| + |T|)$