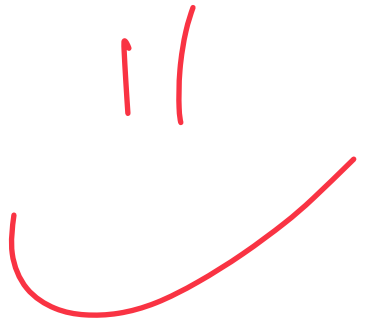
# String Algorithms and Data Structures

# Z-values and the Z-algorithm

CS 199-225
Brad Solomon

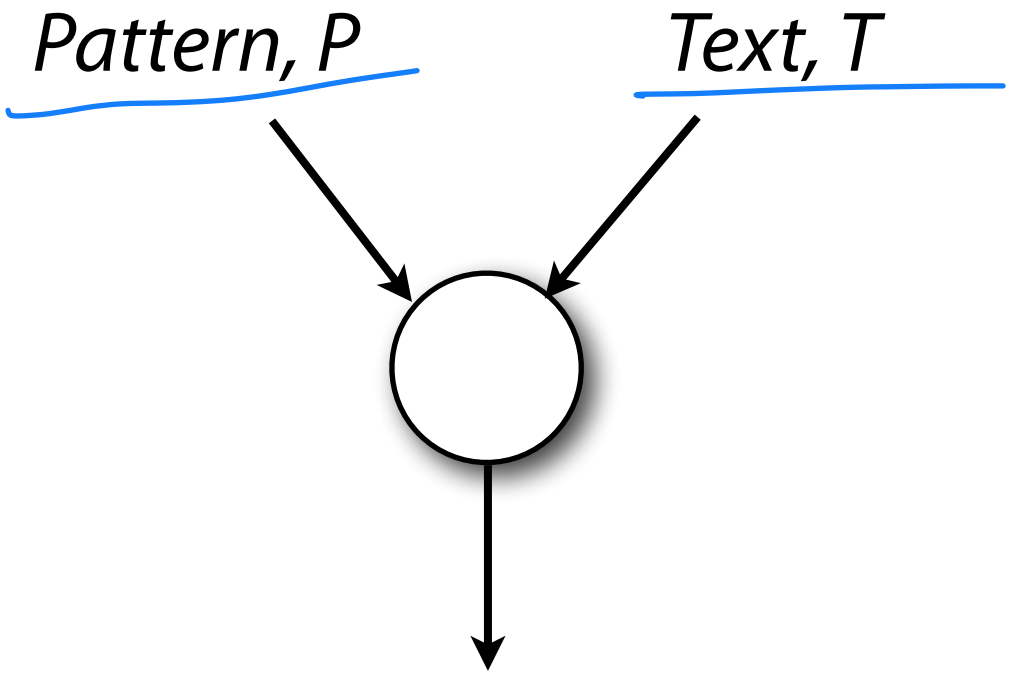
February 2, 2026

This is a two-part assignment (this week + next week)



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

# Exact Pattern Matching




'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?

- Very easy to code! / to follow / to debug
- Its correct
- No preprocessing required

What is bad?

- Pretty slow

# 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)

$n - m$ proposed

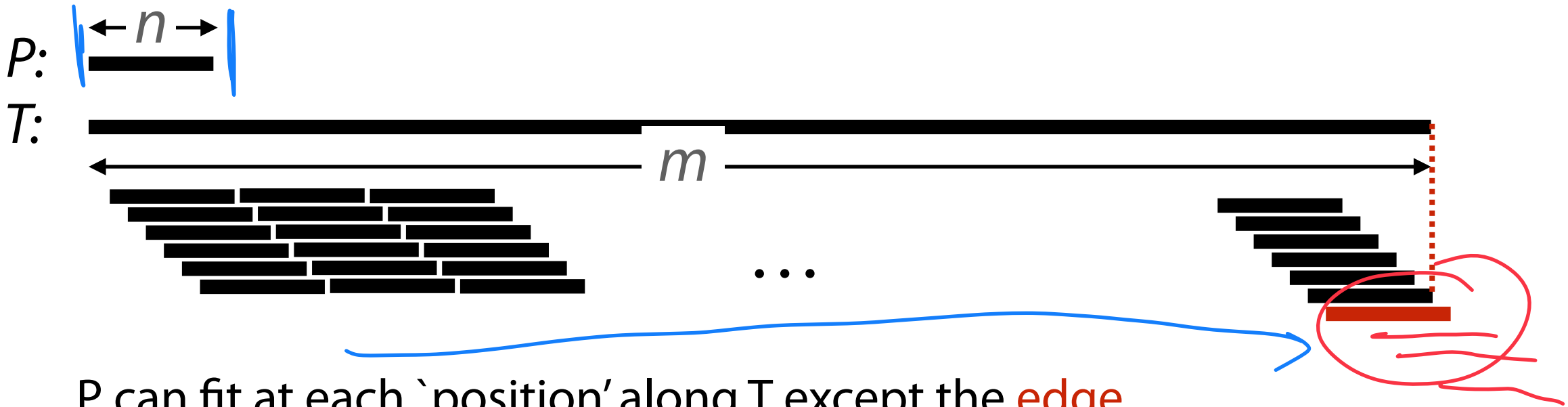
$n / m$ proposed

$n - m + 1$ proposed

# 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

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

( _____ ) x (cost of an alignment)

$m - n$

$m - n + 1$

*P:* **aaaa**

*T:* **aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa**

aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa

aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa **aaaa**

aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa **aaaa**

aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa **aaaa**

aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa

$(n-1)$ alignments which go out of bounds

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)

n

*P*: **aaaa**

*T*: **aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa**

**aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa**
**aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa**
**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? $(n = |P|, \quad m = |T|)$

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

$m \cdot n - n^2 + n$ $\qquad$ $m >> n$

$O(m \cdot n)$

$O(|P|\,|T|)$

# String Algorithms in Genomics

## P: Read ( n = ~50-150)

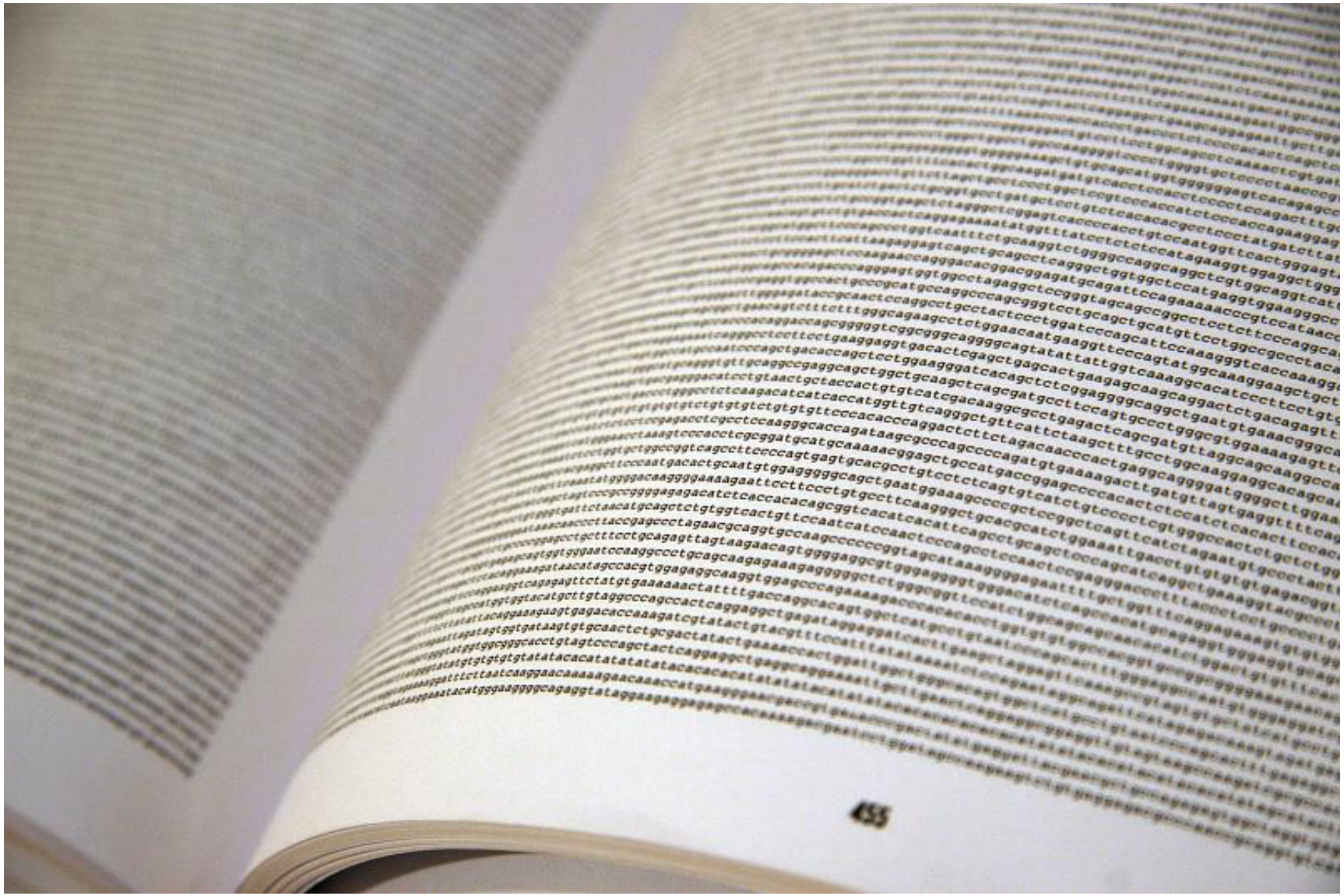
CTCAAACTCCTGACCTTTGGTGATCCACCCGCCTAGGCCTTC

42

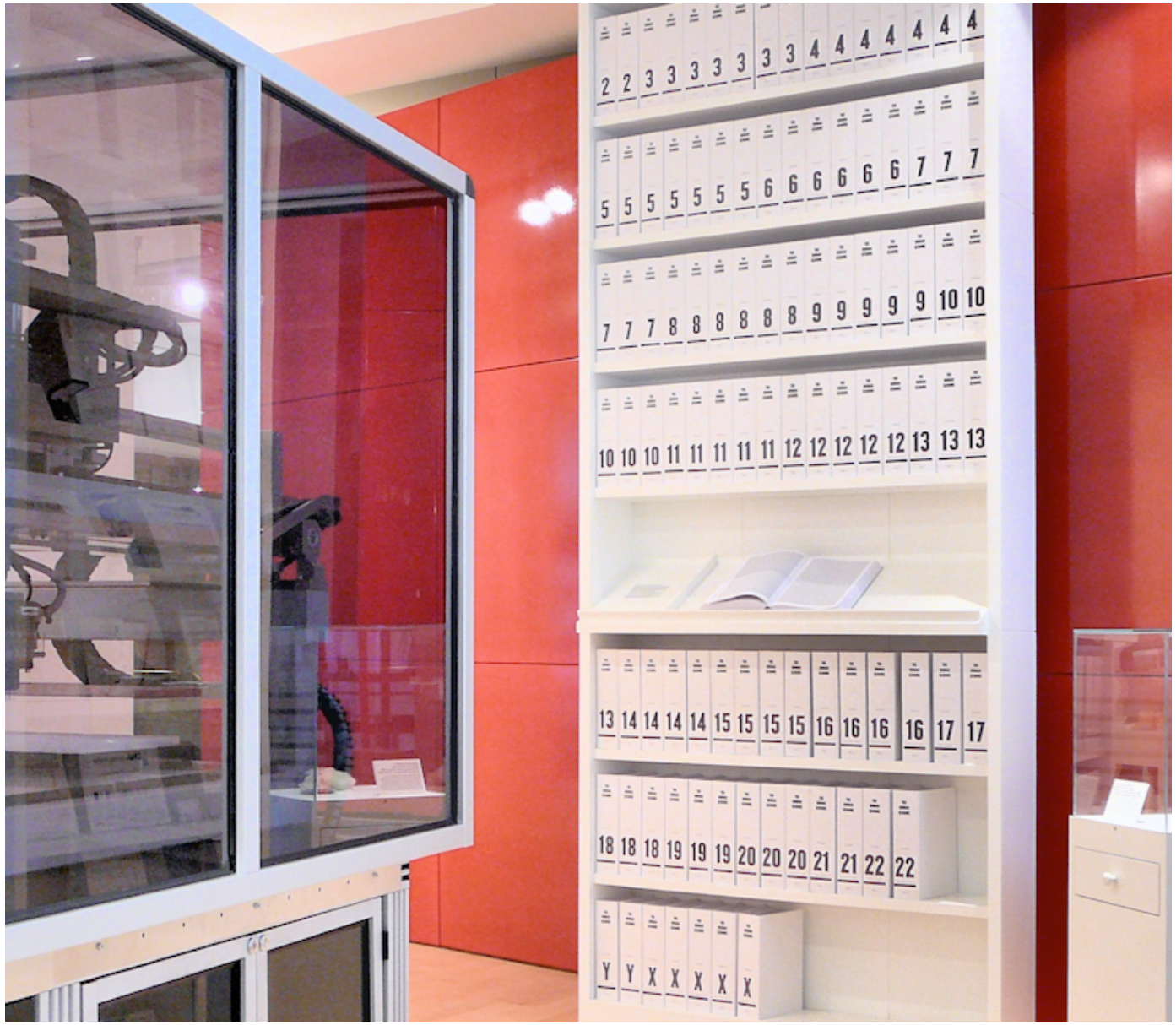
## T: Reference ( m = ~3 billion)

GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTT
CGTCTGGGGGGTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTC
GCAGTATCTGTCTTTGATTCCTGCCTCATCCTATTATTTATCGCACCTACGTTCAATATT
ACAGGCGAACATACTTACTAAAGTGTGTTAATTAATTAATGCTTGTAGGACATAATAATA
ACAATTGAATGTCTGCACAGCCACTTTCCACACAGACATCATAACAAAAAATTTCCACCA
AACCCCCCCTCCCCCGCTTCTGGCCACAGCACTTAAACACATCTCTGCCAAACCCCAAAA
ACAAAGAACCCTAACACCAGCCTAACCAGATTTCAAATTTTATCTTTTGGCGGTATGCAC
TTTTAACAGTCACCCCCCAACTAACACATTATTTTCCCCTCCCACTCCCATACTACTAAT
CTCATCAATACAACCCCCGCCCATCCTACCCAGCACACACACACCGCTGCTAACCCCATA
CCCCGAACCAACCAAACCCCAAAGACACCCCCCACAGTTTATGTAGCTTACCTCCTCAAA
GCAATACACTGACCCGCTCAAACTCCTGGATTTTGGATCCACCCAGCGCCTTGGCCTAAA
CTAGCCTTTCTATTAGCTCTTAGTAAGATTACACATGCAAGCATCCCCGTTCCAGTGAGT
TCACCCTCTAAATCACCACGATCAAAAGGAACAAGCATCAAGCACGCAGCAATGCAGCTC
AAAACGCTTAGCCTAGCCACACCCCCACGGGAAACAGCAGTGATTAA[illegible]TAA
ACGAAAGTTTAACTAAGCTATACTAACCCCAGGGTTGGTCAATTT[illegible]GTGCCAGCCAC[illegible]
GGTCACACGATTAACCCAAGTCAATAGAAGCCGGCGTAAAGAG[illegible]GTTTTAGATCACCCC[illegible]
TCCCCAATAAAGCTAAAACTCACCTGAGTTGTAAAAAACTCC[illegible]GTTGACACAAAATAGAC
TACGAAAGTGGCTTTAACATATCTGAACACACAATAGCTAAG[illegible]CCCAAACTGGGATTAGA
TACCCCACTATGCTTAGCCCTAAACCTCAACAGTTAAATCAA[illegible]AAAACTGCTCGCCAGAA
CACTACGAGCCACAGCTTAAAACTCAAAGGACCTGGCGGTGCT[illegible]CATATCCCTCTAGAGG
AGCCTGTTCTGTAATCGATAAACCCCGATCAACCTCACCACCTC[illegible]TGCTCAGCCTATAT[illegible]
CCGCCATCTTCAGCAAACCCTGATGAAGGCTACAAAGTAAGCGCAA[illegible]TACCCACGTA[illegible]
ACGTTAGGTCAAGGTGTAGCCCATGAGGTGGCAAGAAATGGGCTACATTTT[illegible]CTACCCCA[illegible]
AAAACTACGATAGCCCTTATGAAACTTAAGGGTCGAAGGTGGATTTAGCAGTAAACTAAG
AGTAGAGTGCTTAGTTGAACAGGGCCCTGAAGCGCGTACACACCGCCCGTCACCCTCCTC
AAGTATACTTCAAAGGACATTTAACTAAAACCCCTACGCATTTATATAGAGGAGACAAGT
CGTAACCTCAAACTCCTGCCTTTGGTGATCCACCCGCCTTGGCCTACCTGCATAATGAAG

# String Algorithms in Genomics



# String Algorithms in Genomics



# Improving exact pattern matching

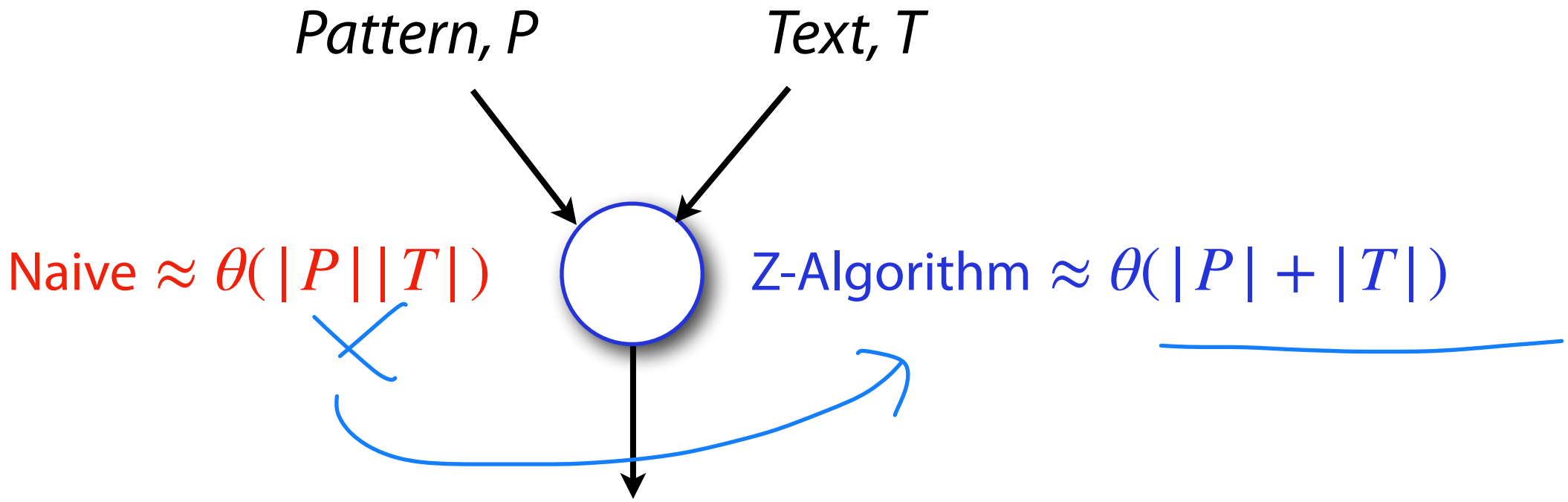
How can we do better than the naïve algorithm?

… 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




'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 $S$.

$S$: **T T C G T T A G C G**

0 1 2 3 4 5 6 7 8 9

$Z_0(S) =$ 10

$Z_1(S) =$ 1

$Z_2(S) =$ 0

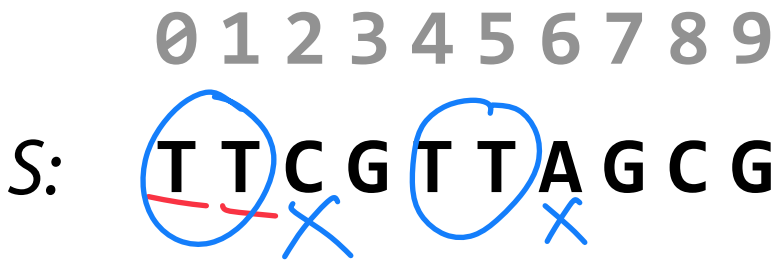
$Z_3(S) =$

$Z_4(S) =$

$Z_5(S) =$

# 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 $S$.




$Z_0(S) = 10$

$Z_1(S) = 1$

$Z_2(S) = 0$

$Z_3(S) =$ 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

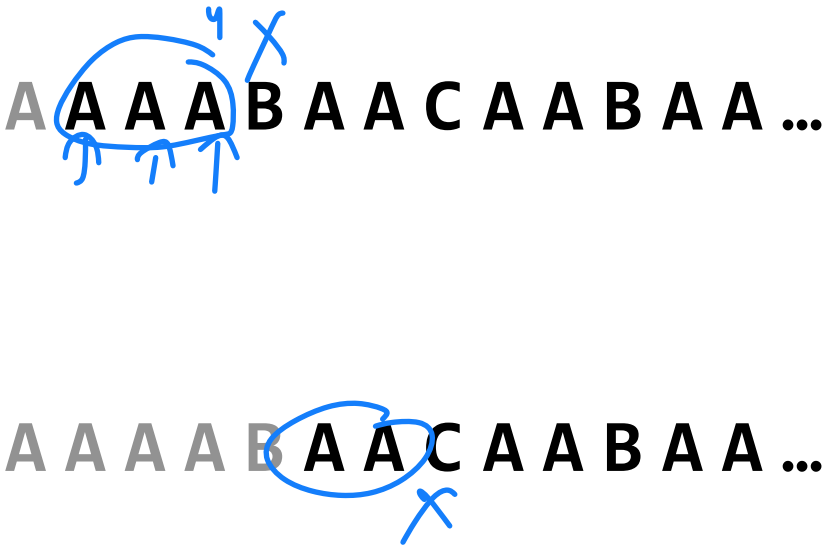
**Naive:** Compute the Z-values by *explicitly* comparing characters (left-to-right scan):

$Z_1 =$ 3

A A A A B A A C A A B A A …

$Z_5 =$ 2

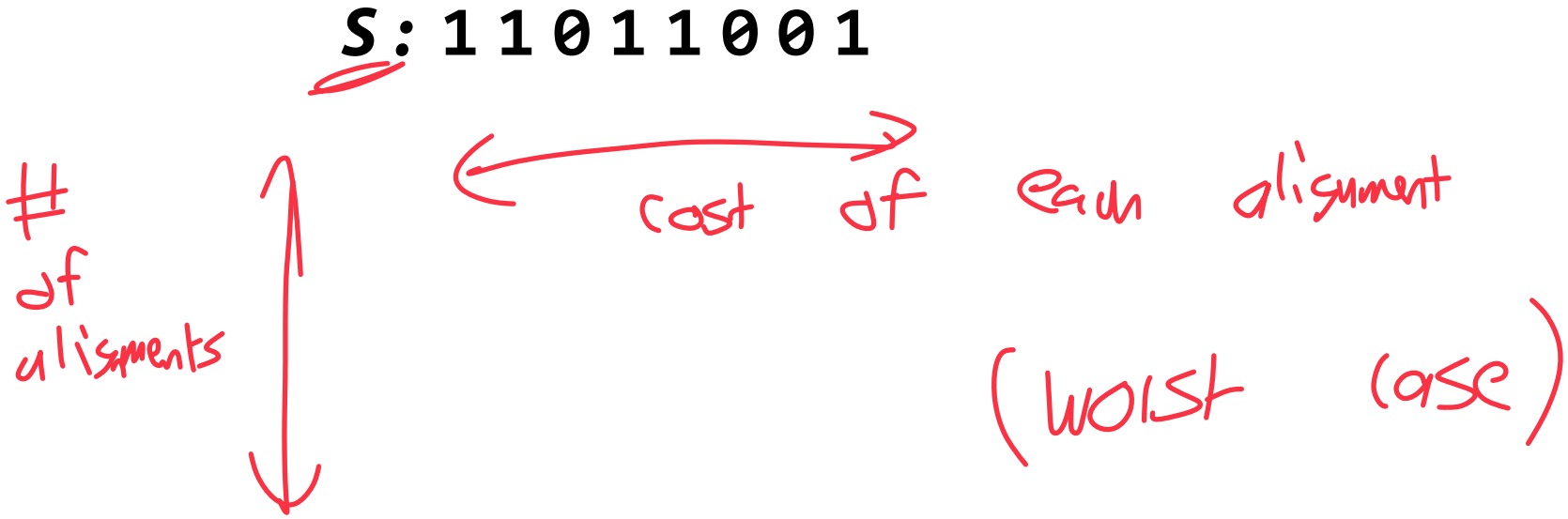
A A A A B A A C A A B A A …




*What is our time complexity?*

# Calculating the Z-values

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




*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      ← S-1
   0 1 1 0 0 1        ← S-2
   1 1 0 0 1          ← S-3
   1 0 0 1
   0 0 1
   0 1
   1                  ≤ 1
```

# alignments

S - 1

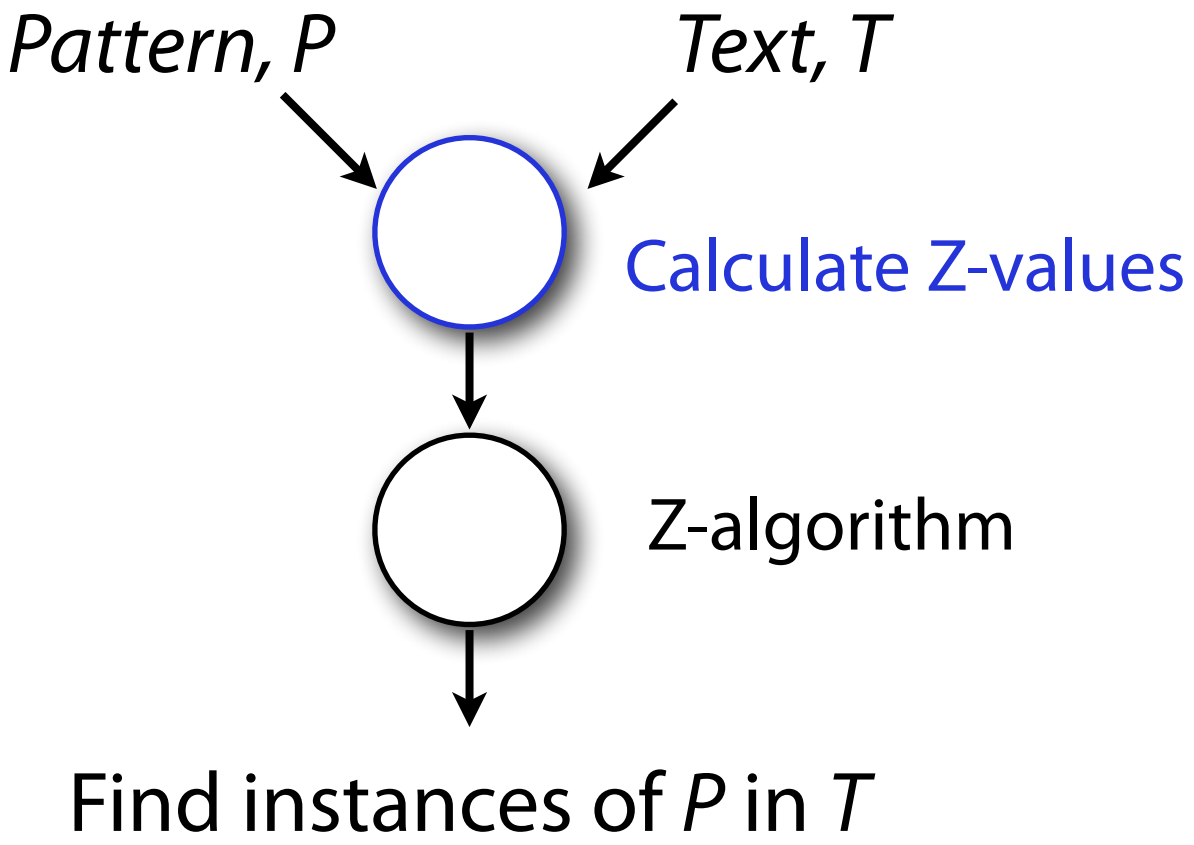
$\sum_{i=1}^{S-1} i = \frac{S(S+1)}{2}$

$O(S^2)$

*What is our time complexity?*

# Pattern matching with the Z-value

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

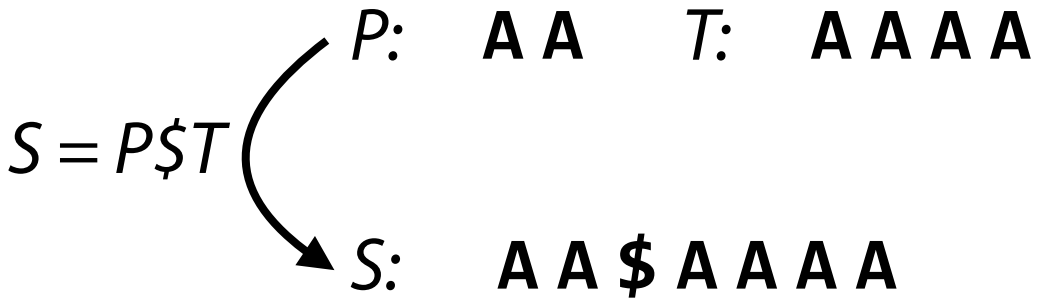


# Z-value Pattern Matching

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

**$** = 'terminal character', outside alphabet

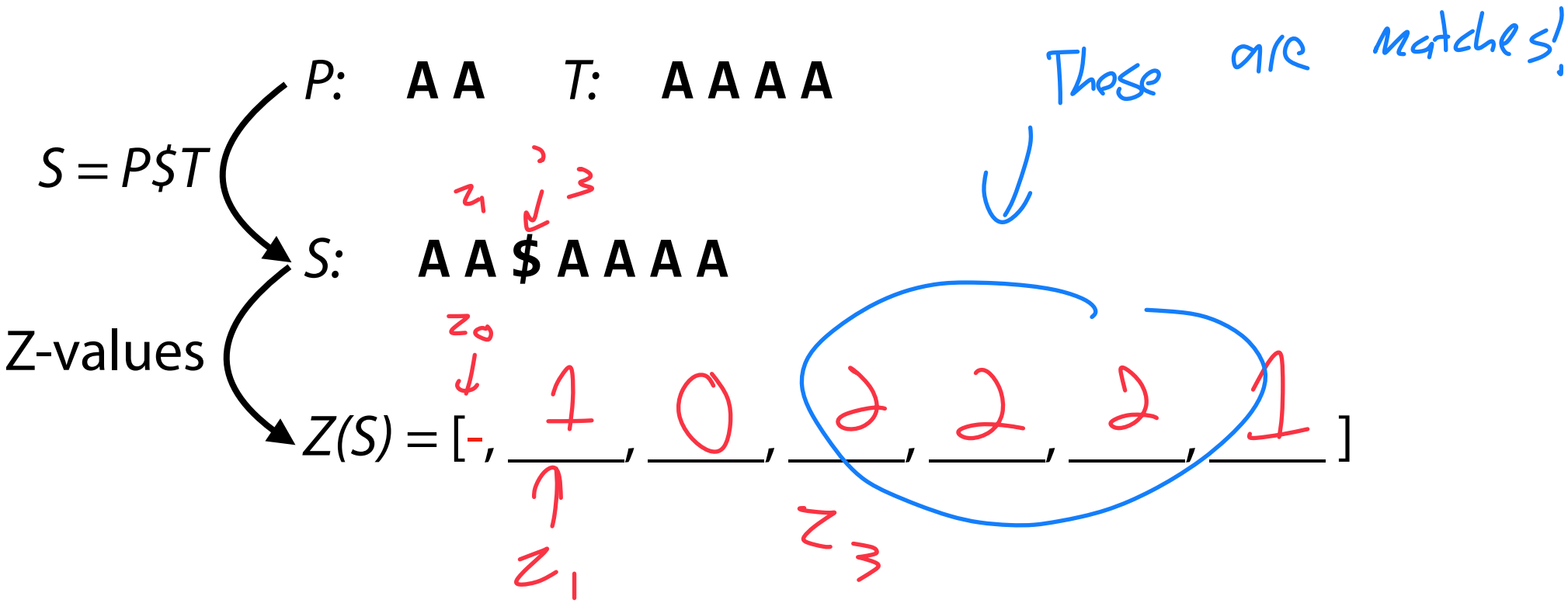
will never match anything



# Z-value Pattern Matching

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

**$** = 'terminal character', outside alphabet



# 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
```

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

What $Z_i$ values are matches? $Z_3, Z_4, Z_5$ (length of P)

What are the matching indices in *T*? $i - |P| - 1$ ($)

# 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? Any |P| value in Z(S)

What are the matching indices in *T*? 0, 1, 2 which are (i - |P| -1)

# Z-value Pattern Matching

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

*S:* T T $ C T T A

*Z(S):* (-, 2, 0, 0, 2, 1, 0)
$z_1$

**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

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?

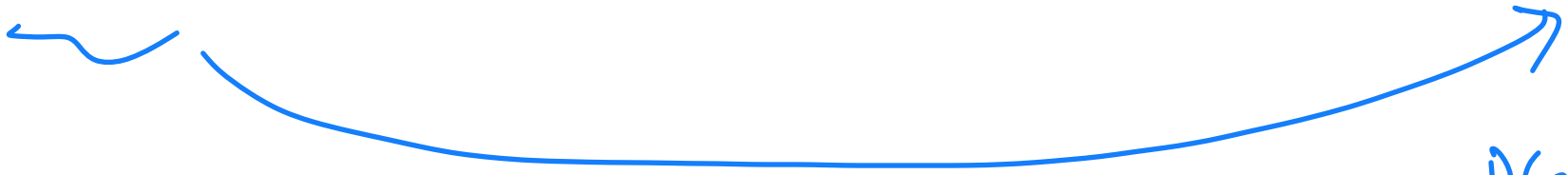
↳ Z value calculation is simply too slow!

$\Theta(\lceil |P| \cdot (|P| + |T|) \rceil)$

# 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 $Z_i(S)$?



Past calc

present index

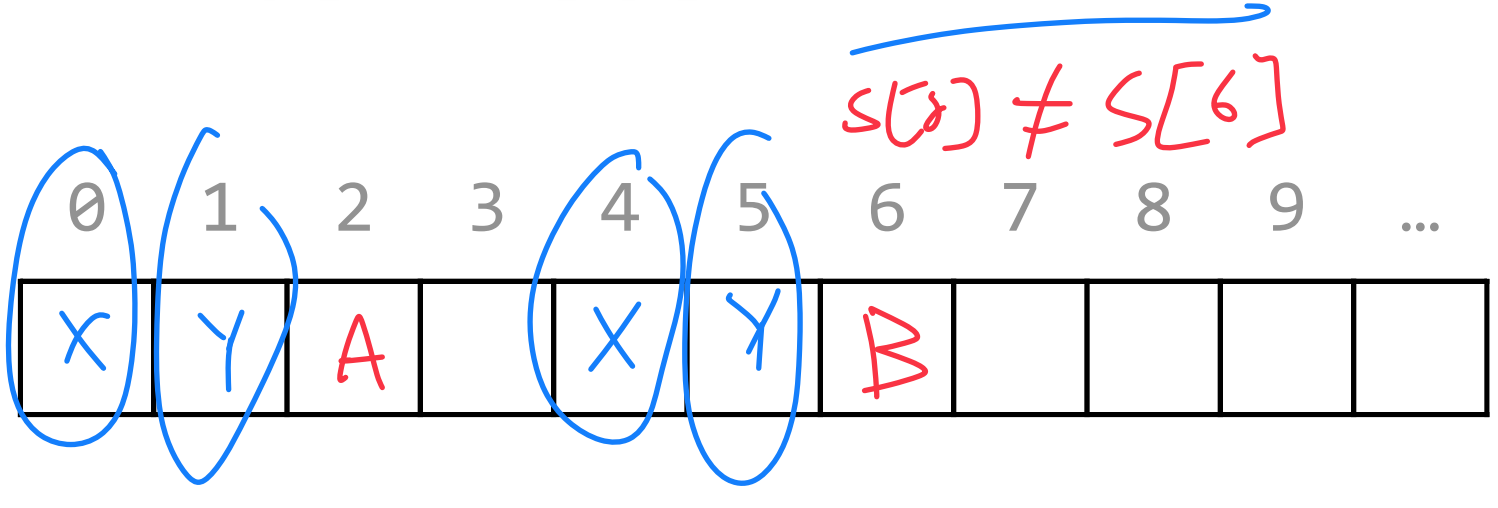
# The Z-value (Take 2)

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:* [redacted]

$Z_4 = 2$



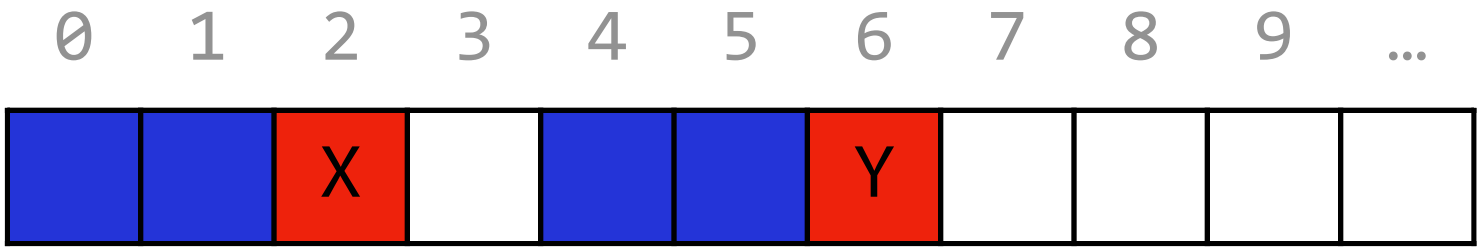
# The Z-value (Take 2)

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:* [illegible]

$Z_4 = 2$



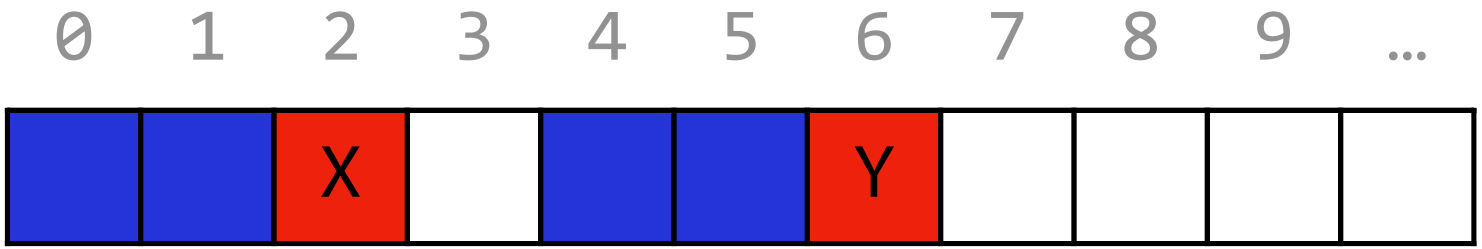
# The Z-value (Take 2)

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?

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

$Z_4 = 2$

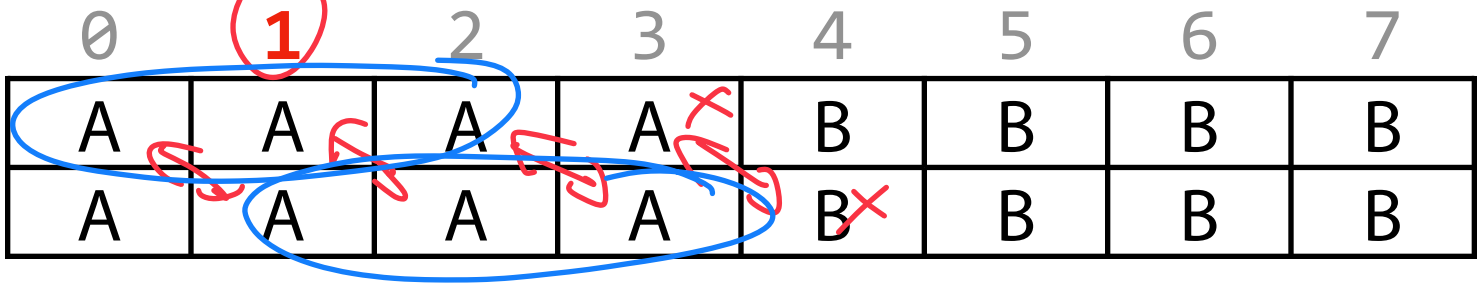


# The Z-Algorithm

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

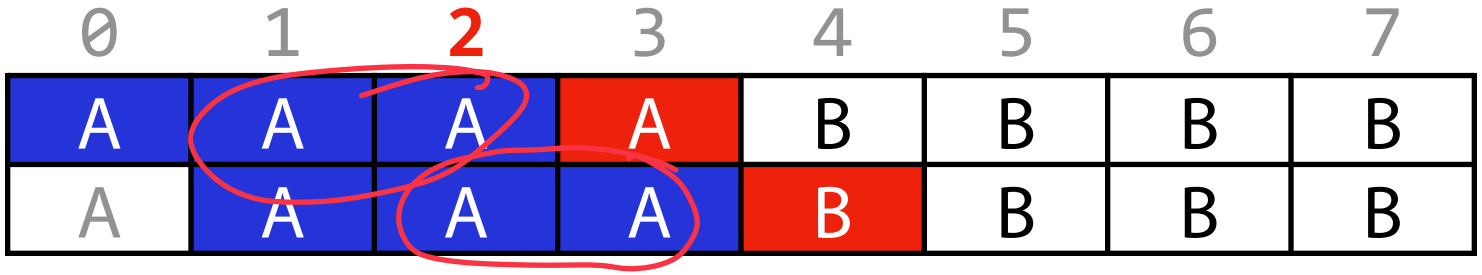
**Case 1:** We know nothing about the characters at S[i]

$Z_1 = ?$




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

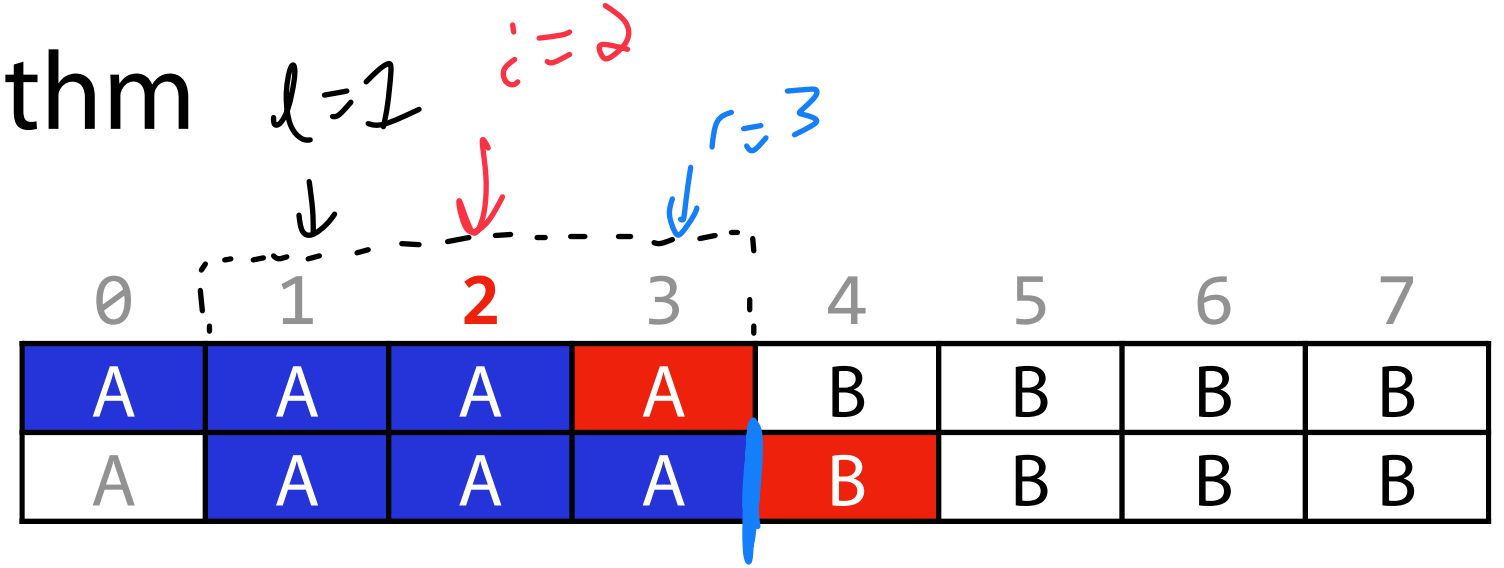
$Z_2 = ?$



# The Z-Algorithm

$Z_1 = 3$

$Z_2 = ?$




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 | i++ | End |
|---|---|---|---|
| $i$, the current index = | 1 | always | 2 |
| $r$, the furthest matching 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

Join Code: 225

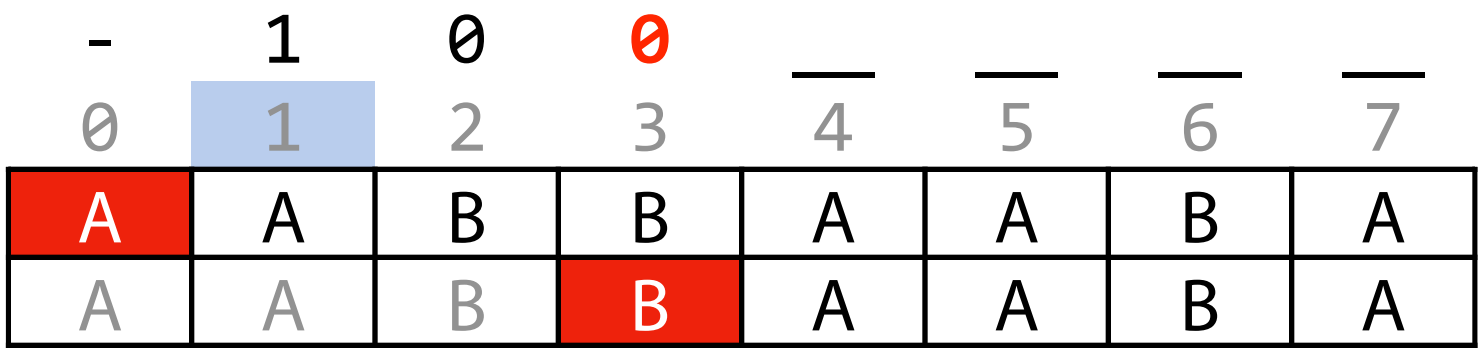
| | Start | End |
|---|---|---|
| $i$, the current index = | 2 | 3 |
| $r$, the furthest matching char = | 1 | 1 |
| $l$, the furthest reaching Z-value = | 1 | 1 |

| - | 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 |

r only updates if new characters matches

# The Z-Algorithm

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



# The Z-Algorithm

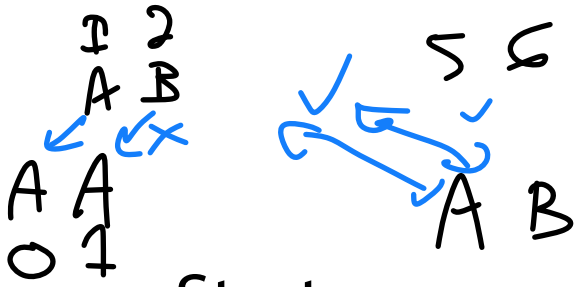
Join Code: 225

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

| - | 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

cool!




$i$, the current index =

$r$, the furthest matching char =

$l$, the furthest reaching Z-value =

| | Start | End |
|---|---|---|
| $i$ | 5 | 6 |
| $r$ | 6 | 6 |
| $l$ | 4 | 4 |

1) B/c $z_4 = 3$ I know 4, 5, 6 is 1, 2, 3
2) Index 5 matches index 1

$z_1 = 1$

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

claim: $z_5$ is $z_1$

No work needed

?.???

# The Z-Algorithm

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

Z6

1

| - | 1 | 0 | 0 | 3 | 1 | 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 =

$r$, the furthest matching 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.

claim: If $i$ is monotonically increasing then we never compare same character twice

↳ $O(|P| + |T|)$