

String Algorithms and Data Structures

Boyer-Moore

CS 199-225

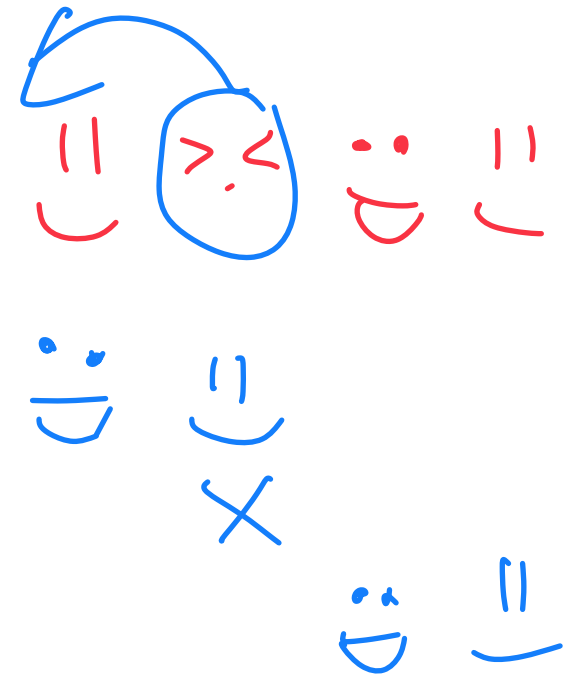
September 30, 2024

Brad Solomon

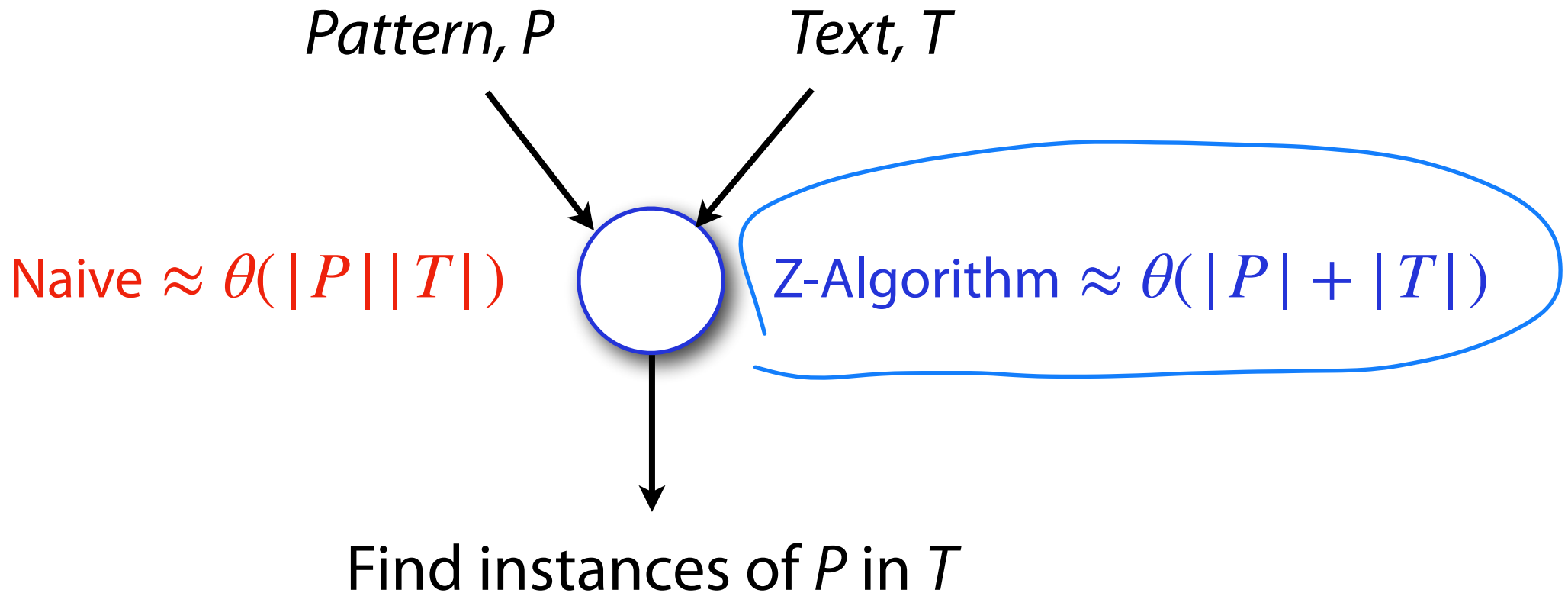


UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science



Exact Pattern Matching w/ Z-algorithm



'instances': An exact, full length copy

Why continue?

The Z-algorithm is:

The Z-algorithm is: $O(|P| + |T|)$ time

An alphabet-independent solution

The Z-algorithm is less good at:

Searching for a **set** of patterns (Aho-Corasick)

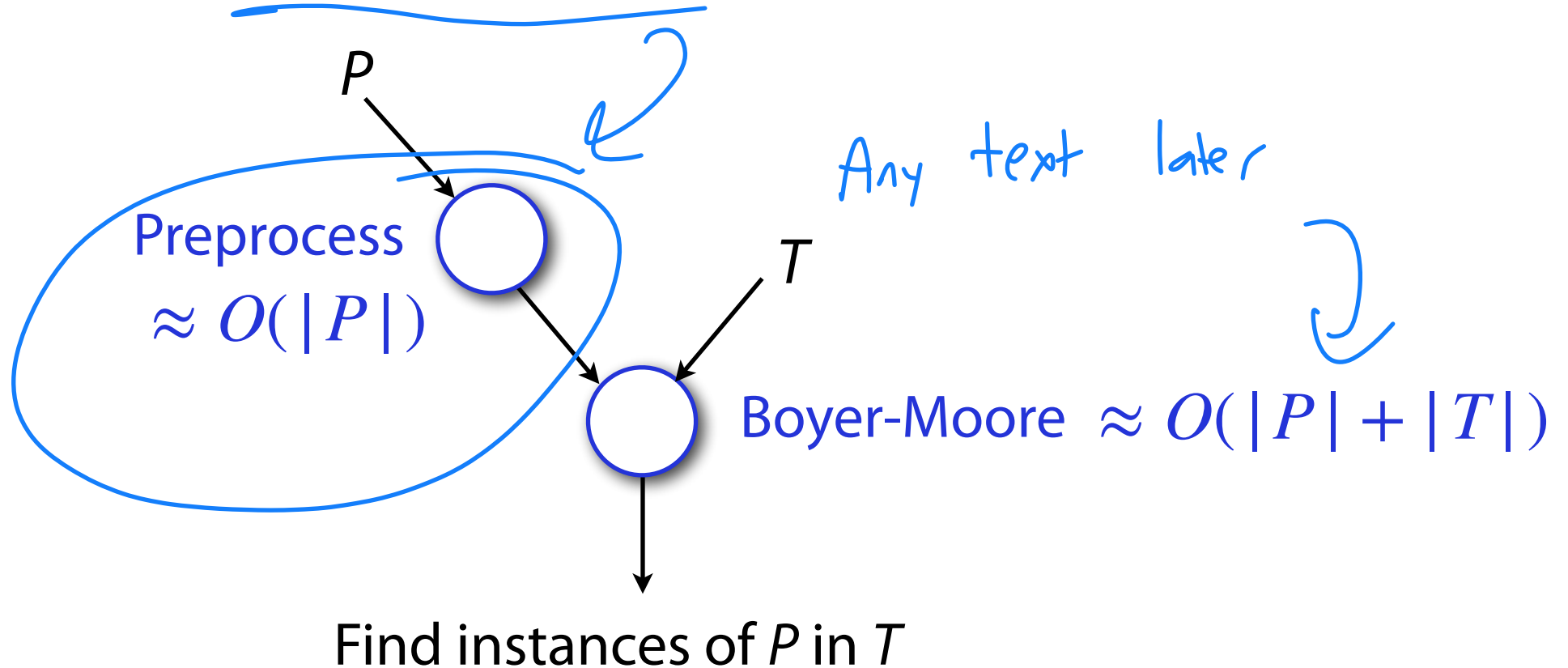
Running in *sub-linear** time (Boyer-Moore)

* — in practice, not theory

Exact pattern matching w/ Boyer-Moore



Boyer Moore **preprocesses** the pattern



'instances': An exact, full length copy

Boyer-Moore

Z-Alg - used prev work to reduce current

Intuition: Learn from alignments to avoid others

Use current align to skip future work

P: c a t

T: | c a r | l c a r r i e d t h e c a t
| c a t |
0 1 2 3 4 5 6 7 8 9 ...

What does this alignment tell us?

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: c a t

T: c a r l c a r r i e d t h e c a t

c a t→

0 1 2 3 4 5 6 7 8 9 ...

What does this alignment tell us?

1) Our pattern doesn't match at this alignment

car

cat

There is no 'r' in
'cat'!

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: c a t

T: c a r | c a r r i e d t h e c a t

c a t

0 1 2 3 4 5 6 7 8 9 ...

c a t

What does this alignment tell us?

2) Our pattern doesn't match at *later* alignments

c a r

c a t

There is no 'r' in 'cat'!

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: c a t

T: c a r l c a r r i e d t h e c a t

c a t ----->

c a t skip!

c a t skip!

What does this alignment tell us?

2) Our pattern doesn't match at *later* alignments

car ← There is no 'r' in
cat 'cat'!

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: w o r d

T: T h e r e w o u l d h a v e b e e n a ...

-----w o r d ----->

0 1 2 3 4 5 6 7 8 9 ...

Boyer-Moore

Intuition: Learn from alignments to avoid others

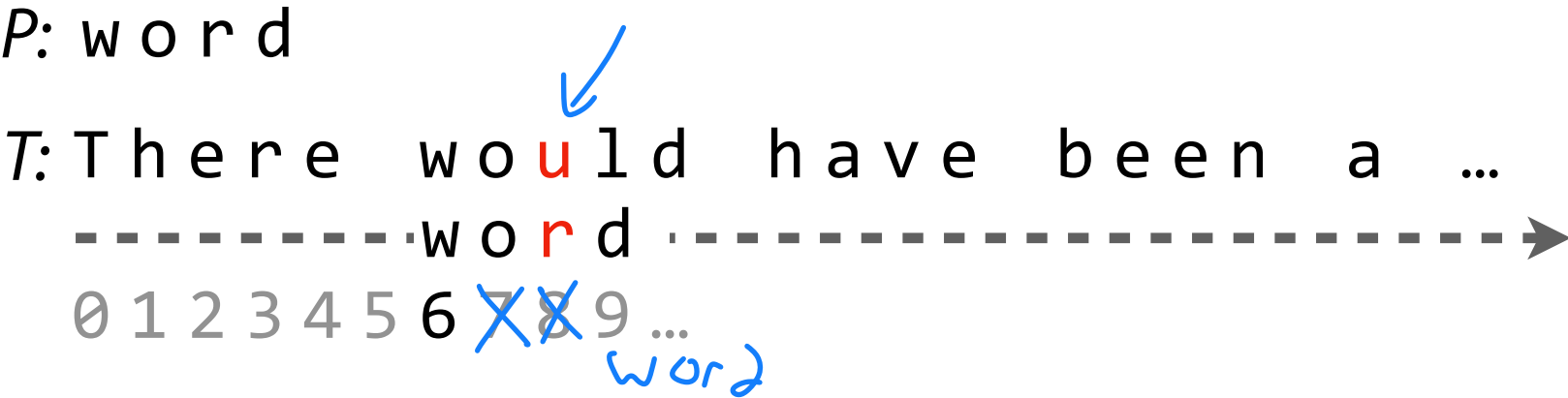
P: w o r d

T: T h e r e w o u l d h a v e b e e n a ...
-----·w o r d ----->
0 1 2 3 4 5 6 7 8 9 ...

How many alignments can we skip?

Boyer-Moore

Intuition: Learn from alignments to avoid others



How many alignments can we skip?

2 is true

1) Our pattern doesn't match at this alignment



Boyer-Moore

Intuition: Learn from alignments to avoid others

P: w o r d

T: T h e r e w o u l d h a v e b e e n a ...
-----w o r d ----->
0 1 2 3 4 5 6 7 8 9 ...

How many alignments can we skip? 2

2) Our pattern doesn't match at *later* alignments

T: w o u l ← There is no 'u' in
P: w o r d 'word'!

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: word

T: There would have been a ...

-----word----->

word skip!

word skip!

word

How many alignments can we skip? 2

2) Our pattern doesn't match at *later* alignments

T: woul

P: word

There is no 'u' in 'word'!

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: T A G A C

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

- TAGAC ----->

X TAGAC

X TAGAC

X TAGAC

TAGAC

How many alignments can we skip?

3

TAGAT



TAGAC

There IS a T in
'TAGAC'!

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: T A G A C

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

- · T A G A C ----->

T A G A C skip!

T A G A C skip!

T A G A C skip!

T A G A C

How many alignments can we skip?

3

TAGAT

TAGAC

There IS a T in
'TAGAC'!

Boyer-Moore

Intuition: Learn from alignments to avoid others

P: A A B B B

T: A A A B A B A A A A A A A A A A A A A A A A

- A A B B B ----->

A A B B B skip!

A A B B B the *first* match we encounter!

A A B B B

How many alignments can we skip? 1

A A B A B

A A B B B

There IS an A in 'AAABB'!

Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character.

Step 1: T : C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
 P : C C T T T T G C

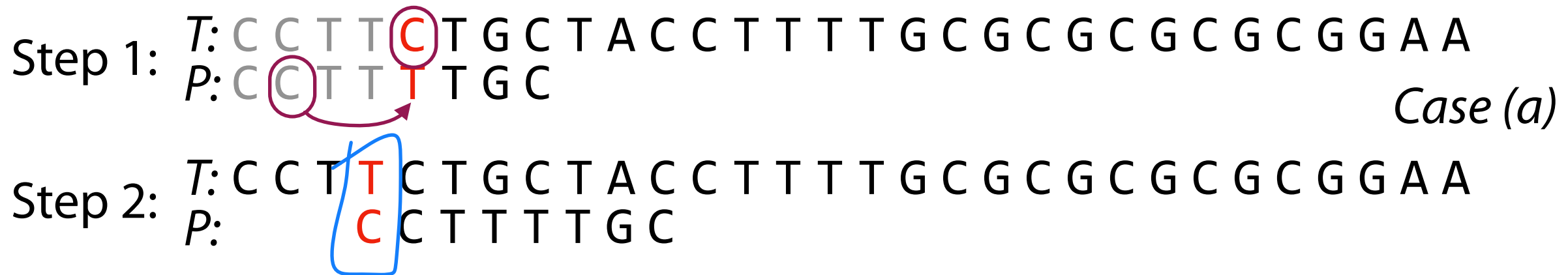
Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character.

Step 1: T : C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
 P : C C T T T G C

Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character.



Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character.

Step 1: T : C C T T **C** T G C T A C C T T T T G C G C G C G C G C G G A A
 P : C **C** T T **T** T G C *Case (a)*

Step 2: T : C C T **T** C T G C T A C C T T T T G C G C G C G C G C G G A A
 P : **C** C T T T T G C

Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character.

Step 1: $T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A$
 $P: C C T T T T G C$ *Case (a)*

Step 2: $T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A$
 $P: C C T T T T G C$ *Case (b)*

Step 3: $T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A$
 $P: C C T T T T G C$ *No T*

Boyer-Moore: Bad Character rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character. (c) If no mismatch, don't skip

Step 1: $T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A$
 $P: C C T T T T G C$ *Case (a)*

Step 2: $T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A$
 $P: C C T T T T G C$ *Case (b)*

Step 3: $T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A$
 $P: C C T T T T G C$ *Case (b)*

(etc)

Step 7: $T: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A$
 $P: C C T T T T G C$ *Case (c)*

Boyer-Moore: Bad Character rule



Step 1: *T:* C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C
C C T T T T G C skip!
C C T T T T G C

Step 2: *T:* C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C
C C T T T T G C skip nothing?

Step 3: *T:* C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C
C C T T T T G C skip!

Boyer-Moore: Bad Character rule

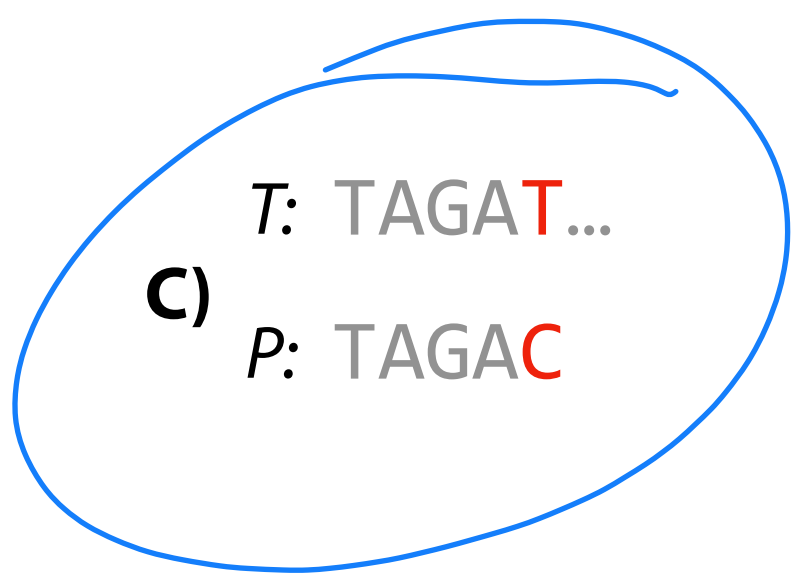
Which of the following alignments skips the most?

A) *T*: TATAT...
P: TAGAC

B) *T*: TTGAT...
P: TAGAC

C) *T*: TAGAT...
P: TAGAC

D) *T*: TAGTT...
P: TAGAC



Boyer-Moore: Bad Character rule

Which of the following alignments skips the most?

A) T: TATAT... Skip 1
P: TAGAC

B) T: TTGAT... Skip 0
P: TAGAC

C) T: TAGAT... Skip 3
P: TAGAC

D) T: TAGTT... Skip 2
P: TAGAC



Mismatches (usually) skip more if occur later in string!

(Heuristic)

Boyer-Moore: Bad Character rule improvement

Continue to test alignment from left-to-right

P: T A G A C

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

- T A G A C ----->

Boyer-Moore: Bad Character rule improvement

Continue to test alignment from left-to-right

... but compare **characters** from right to left.



↳ skip more alignments!

Right-to-left-scanning w/ BC Rule

P: w o r d

T: T h e r e w o u l d h a v e b e e n a ...
-----·w o r d ----->
 <-----

Right-to-left-scanning w/ BC Rule

T: A A A A
P: B A A B

P: word

T: There would have been a ...



Intuit: Multiple characters
can be bad

T: wou**l**

P: wor**d**

There is no 'l' in
'word'!

The character that skips the most is probably the last one

↳ can skip the most // can't skip more than chars to left

Right-to-left-scanning w/ BC Rule

P: word

T: There would have been a ...

-----word----->

←-----

T: wou**l**

P: wor**d**

There is no 'l' in
'word'!

How many alignments do we skip?



Right-to-left-scanning w/ BC Rule

P: word

T: There would have been a ...

-----word----->

word

word

word

SKip future work!

How many alignments do we skip?

3

Right-to-left-scanning w/ BC Rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character. (c) If no mismatch, don't skip



Right-to-left-scanning w/ BC Rule

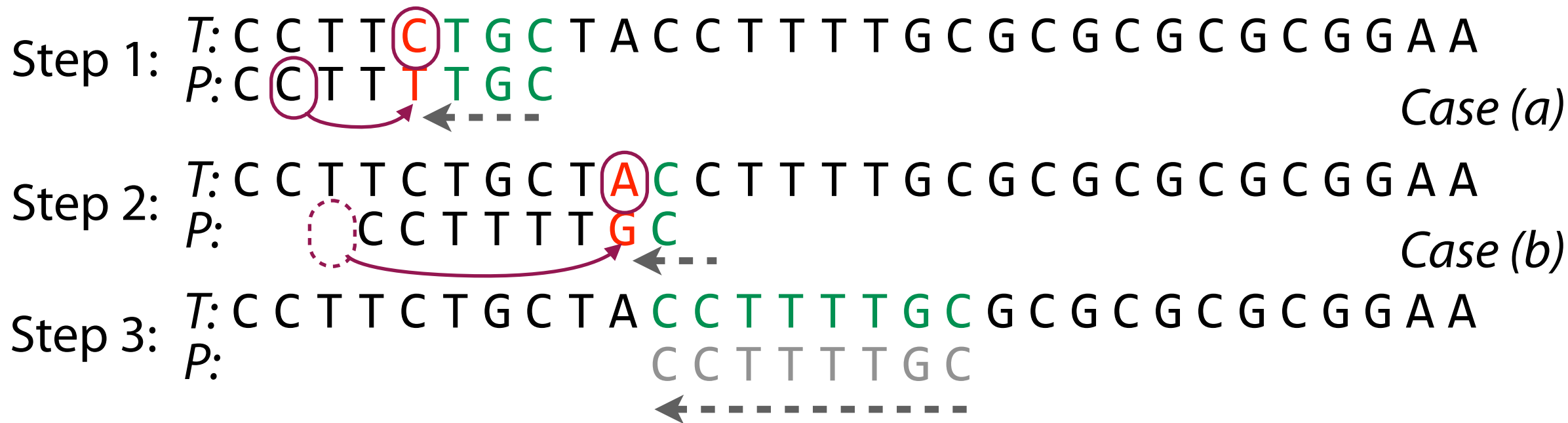
Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character. (c) If no mismatch, don't skip

Step 1: T : C C T T **C** T G C T A C C T T T T G C G C G C G C G C G G A A
 P : C **C** T T **T** T G C *Case (a)*

Step 2: T : C C T **T** C T G C T **A** C C T T T T G C G C G C G C G C G G A A
 P : C C T T T T **G** C

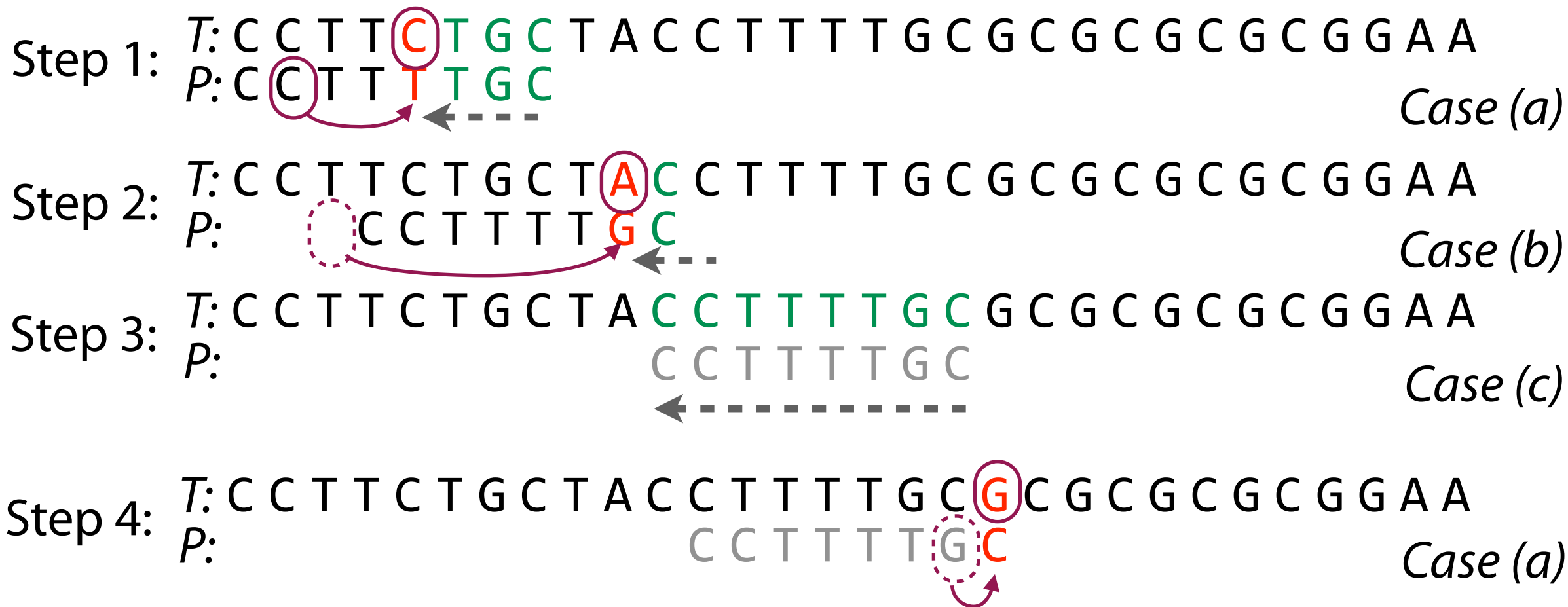
Right-to-left-scanning w/ BC Rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character. (c) If no mismatch, don't skip



Right-to-left-scanning w/ BC Rule

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character. (c) If no mismatch, don't skip



Right-to-left-scanning w/ BC Rule

Step 1: *T*: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C

Step 2: *T*: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C

Step 3: *T*: C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C

Up to step 3, we skipped 8 alignments

C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A

5 characters in *T* were *never* looked at

C C T T C T G C T A C C T T T T G C G C G C G C G C G G A A

Right-to-left-scanning w/ BC Rule



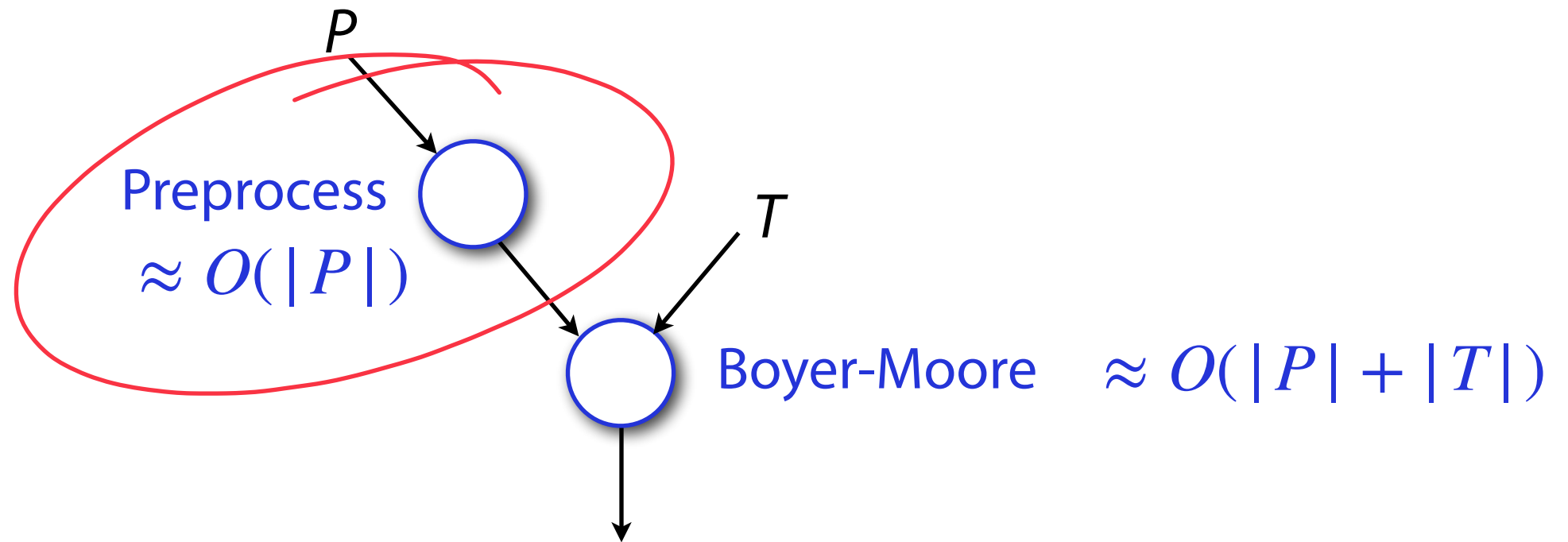
Learn from character comparisons to skip pointless alignments

1. When we hit a mismatch c , move P along until c becomes a match (or P moves past c) “Bad character rule”
2. Try alignments in one direction, but do character comparisons in *opposite* direction “Right-to-left scanning”

How do we put the first two rules in practice?

Exact pattern matching w/ Boyer-Moore

Boyer Moore **preprocesses** the pattern



Find instances of P in T

'instances': An exact, full length copy

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args:

P : T C G C

Σ : A C G T

The goal is to produce a table which tracks *skips*

P

	T	C	G	C
A				
C				
G				
T				

Σ

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: P : T C G C Σ : A C G T

The goal is to produce a table which tracks *skips*

P

	T	C	G	C
A				
C				
G				
T				

Σ

T : ? ? ? T ? ? ? ? ? ?

P : T C G C

T C G C

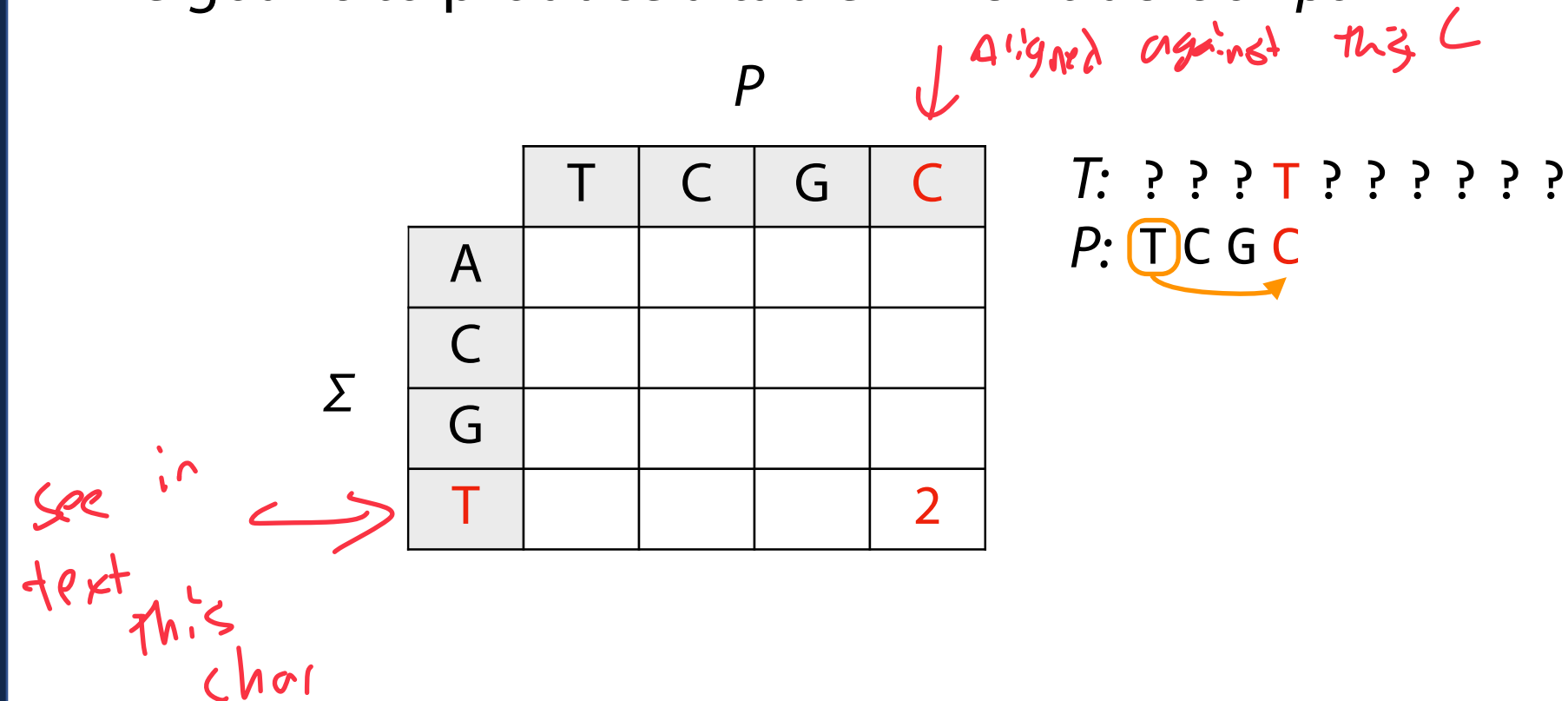
T C G C

T C G C

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: P : T C G C Σ : A C G T

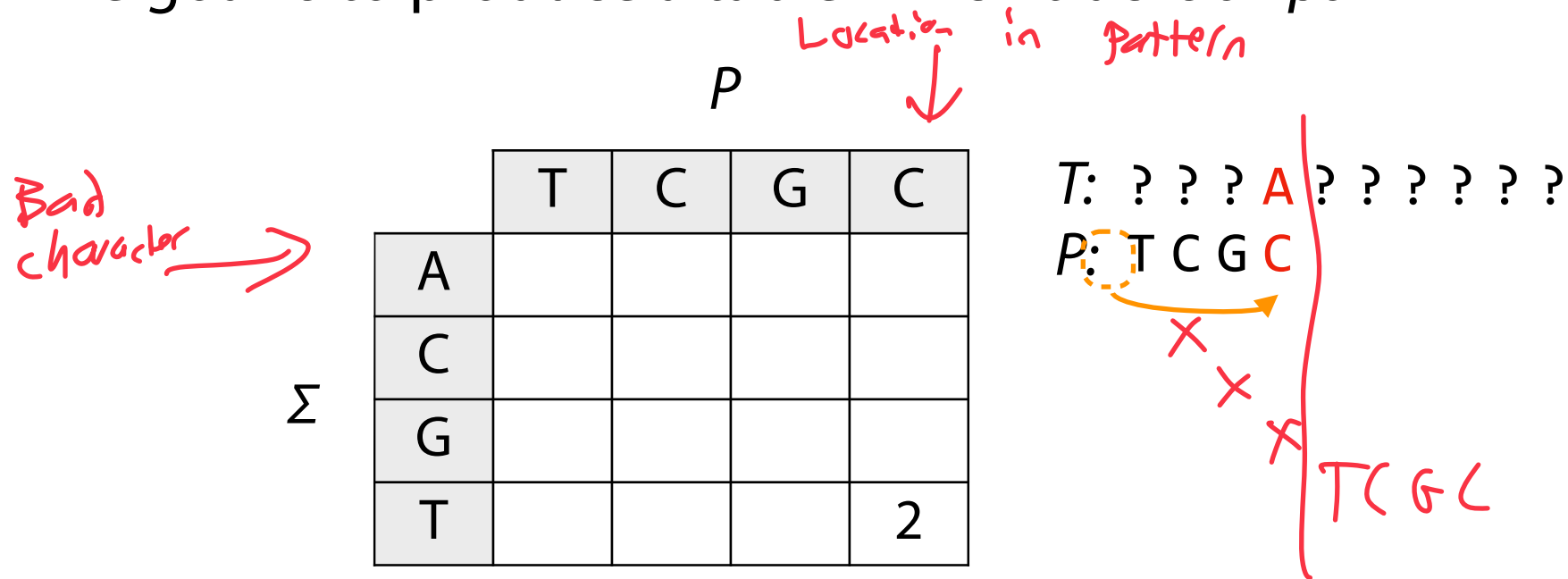
The goal is to produce a table which tracks *skips*



Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: P : T C G C Σ : A C G T

The goal is to produce a table which tracks *skips*



Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: P : T C G C Σ : A C G T

The goal is to produce a table which tracks *skips*

P

	T	C	G	C
A				3
C				
G				
T				2

Σ

T : ? ? ? A ? ? ? ? ? ?
 P : T C G C

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: $P: T C G C$ $\Sigma: A C G T$

The goal is to produce a table which tracks *skips*

		P			
		T	C	G	C
Σ	A	0	1	2	3
	C	0	-	0	-
	G	0	1	-	0
	T	-	0	1	2

$T: ? ? A ? ? ? ? ? ? ?$

$P: T C G C$

$T: ? ? C ? ? ? ? ? ? ?$

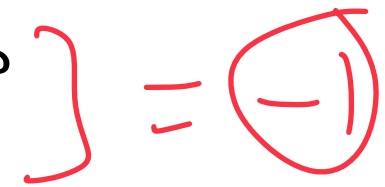
$P: T C G C$

$T: ? ? G ? ? ? ? ? ? ?$

$P: T C G C$

$T: ? ? T ? ? ? ? ? ? ?$

$P: T C G C$



Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: P : B A B A A A B

Σ : A B

Pattern

	B	A	B	A	A	A	B
Σ A							
B							

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: P : B A B A A A B

Σ : A B

For each character p in pattern P

For each character c in alphabet Σ

Find the closest previous instance of p (to the left of c).

Pattern

		B	A	B	A	A	A	B
Σ	A	0	-1	0	-1			
	B	-1	0	-1	0			

Boyer-Moore: BC rule preprocessing

Preprocessing requires two args: P : B A B A A A B

Σ : A B

For each character p in pattern P

For each character c in alphabet Σ

Find the closest previous instance of p (to the left of c).

Pattern

	B	A	B	A	A	A	B
A	0	-1	0	-1	-1	-1	0
B	-1	0	-1	0	1	2	-1

Assignment 4: a_bmoore



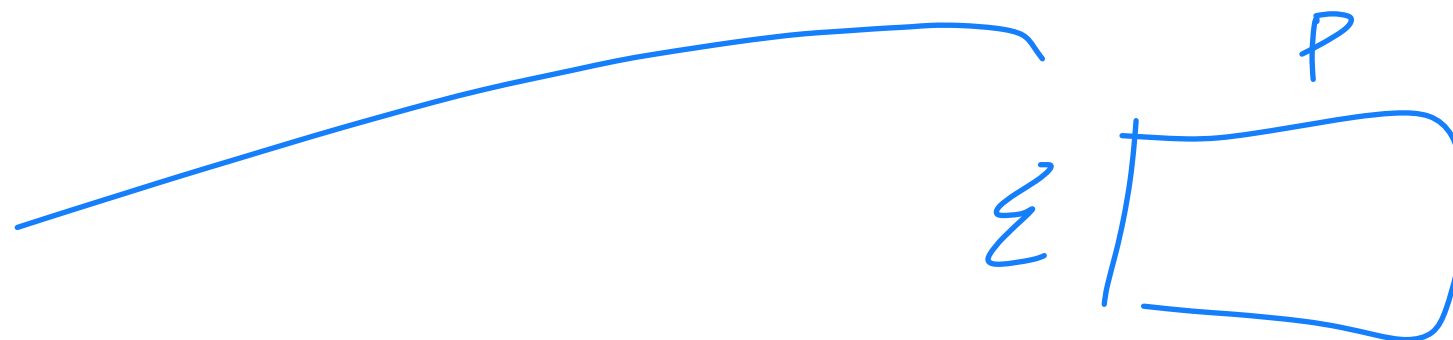
Learning Objective:

Implement preprocessing of patterns with Boyer-Moore*

Observe Boyer-Moore* efficiency *as a heuristic*



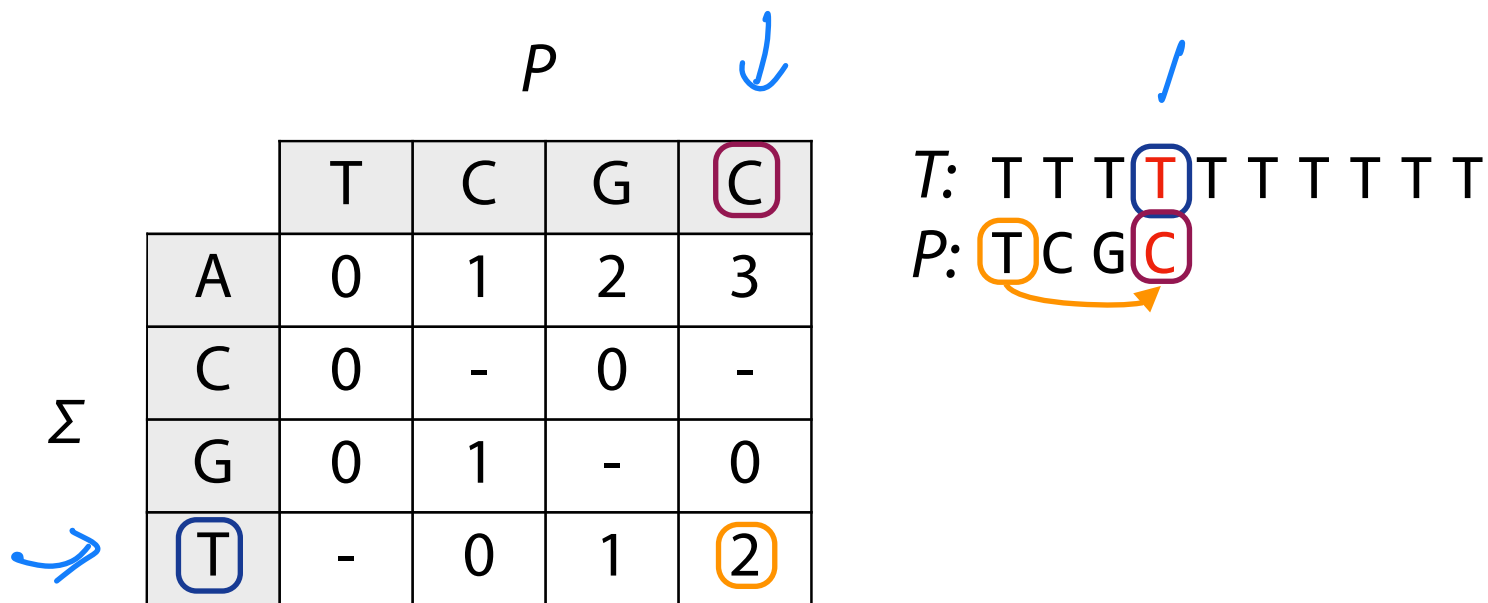
Consider: Optimal preprocessing is $\theta(|P| |\Sigma|)$. Can you code it?



Boyer-Moore: Using the BC Table

Try alignments from left-to-right and match characters from right-to-left

When we encounter a mismatch, skip the calculated number of alignments



Boyer-Moore: Using the BC Table

Try alignments from left-to-right and match characters from right-to-left

When we encounter a mismatch, skip the calculated number of alignments

Text Σ

	<i>P</i>				
	T	C	G	C	
A	0	1	2	3	
C	0	-	0	-	
G	0	1	-	0	
T	-	0	1	2	

T: G G G G G G G G G G
P: T C G C

Handwritten annotations: The 'C' in the header of the BC table is circled in blue. The 'G' in the 'G' row of the BC table is circled in blue. A blue arrow points from the circled 'G' to the circled 'C' in the header. In the alignment example, the 'G' in the pattern 'TCGC' is underlined in blue, and a blue arrow points from the underlined 'G' to the 'G' in the text 'GGGGGGGGGG'.

Boyer-Moore: Using the BC Table

Try alignments from left-to-right and match characters from right-to-left

When we encounter a mismatch, skip the calculated number of alignments

		<i>P</i>			
		T	C	G	C
Σ	A	0	1	2	3
	C	0	-	0	-
	G	0	1	-	0
	T	-	0	1	2

T: A A **T** C A A T A G C ✓
P: T C **G** C
✗
T C G C

Boyer-Moore: Tracking total skips

		P	
		A	A
Σ	A	-1	-1
	B	0	1

\times
 $T: B B B B$
 $A A$

$A A$
 $T: B B B B B$
 $A A$

$A A$
 $T: B B B B B B$
 $A A$

Boyer-Moore: Tracking total skips

		P		
		A	A	A
Σ	A	-1	-1	-1
	B	0	1	2

T: B B (B) B ? ?

A A A

A A A

A A A

B B B ? ?

Assignment 4: a_bmoore



Learning Objective:

Implement preprocessing of patterns with Boyer-Moore*

Observe Boyer-Moore* efficiency *as a heuristic*

Consider: Our Boyer-Moore is theoretically slower than Z-algorithm.

But is it slower in practice? What is our total character comparisons?



A complete bonus lecture!

A better Boyer-Moore

Learn from character comparisons to skip pointless alignments

1. When we hit a mismatch c , move P along until c becomes a match (or P moves past c) “Bad character rule”
2. Try alignments in one direction, but do character comparisons in *opposite* direction “Right-to-left scanning”

Is this $O(|P| + |T|)$?

A better Boyer-Moore

The complete Boyer-Moore algorithm, ***with all refinements***, is $O(|P| + |T|)$.

Refinements include:

- "strong" good suffix rule
- Galil rule

We will be covering the 'weak' good suffix rule

If interested in refinements, see Gusfield textbook (syllabus) or contact me for details

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

P: A C A T A C

T: T A C A ^XG A C A T A C A T G A C A G T G A C C A

- ' A C A T A C ' ----->

←-----

What does this alignment tell us?

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

P: A C A T A C

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

- ' A C A T A C ' ----->

←-----

We only want to look at alignments that are ***at least as good*** as our current alignment

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

P: A C A T A C

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

- 'A C A T A C' ----->

←-----

What does partial match (the suffix 'AC') tell us?

Any alignment that overlaps this region of the text must match the suffix! So we can look for another 'AC' somewhere in the pattern!

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

P: A C A T A C

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

- 'A C A T A C' ----->

A C A T A C

How many alignments do we skip?

3

Any alignment that overlaps this region of the text must match the suffix! So we can look for another 'AC' somewhere in the pattern!

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

P: A T C

T: A G T A G **C** A G C A C A G T A G C A G C T A G A



Any alignment that overlaps this region of the text must match the suffix! So we can look for another _____ somewhere in the pattern!

How many alignments do we skip? α

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

P: A T C

T: A G T A G **C** A G C A C A G T A G C A G C T A G A



Any alignment that overlaps this region of the text must match the suffix! So we can look for another **C** somewhere in the pattern!

How many alignments do we skip?

2

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

P: G C A G C

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

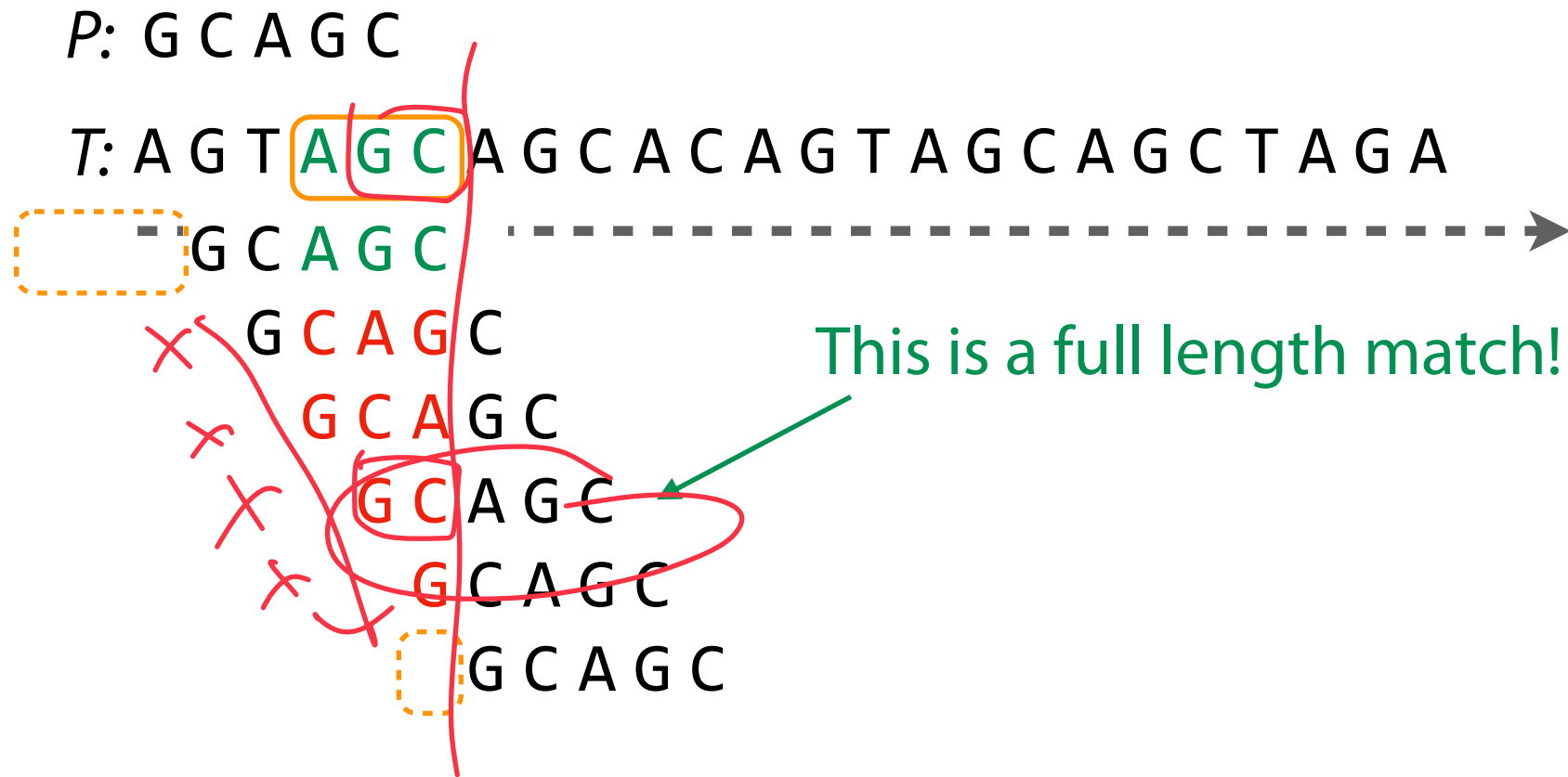
 - ' G C A G C ----->

Any alignment that overlaps this region of the text must match the suffix! So we can look for another ^{AGC} somewhere in the pattern!

How many alignments do we skip?

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others



How many alignments do we skip? 2

“Weak” Good Suffix rule

Intuition: Learn from alignments to avoid others

P: G C A G C

T: A G T **A G C** A G C A C A G T A G C A G C T A G A

G C A G C →
G C A G C

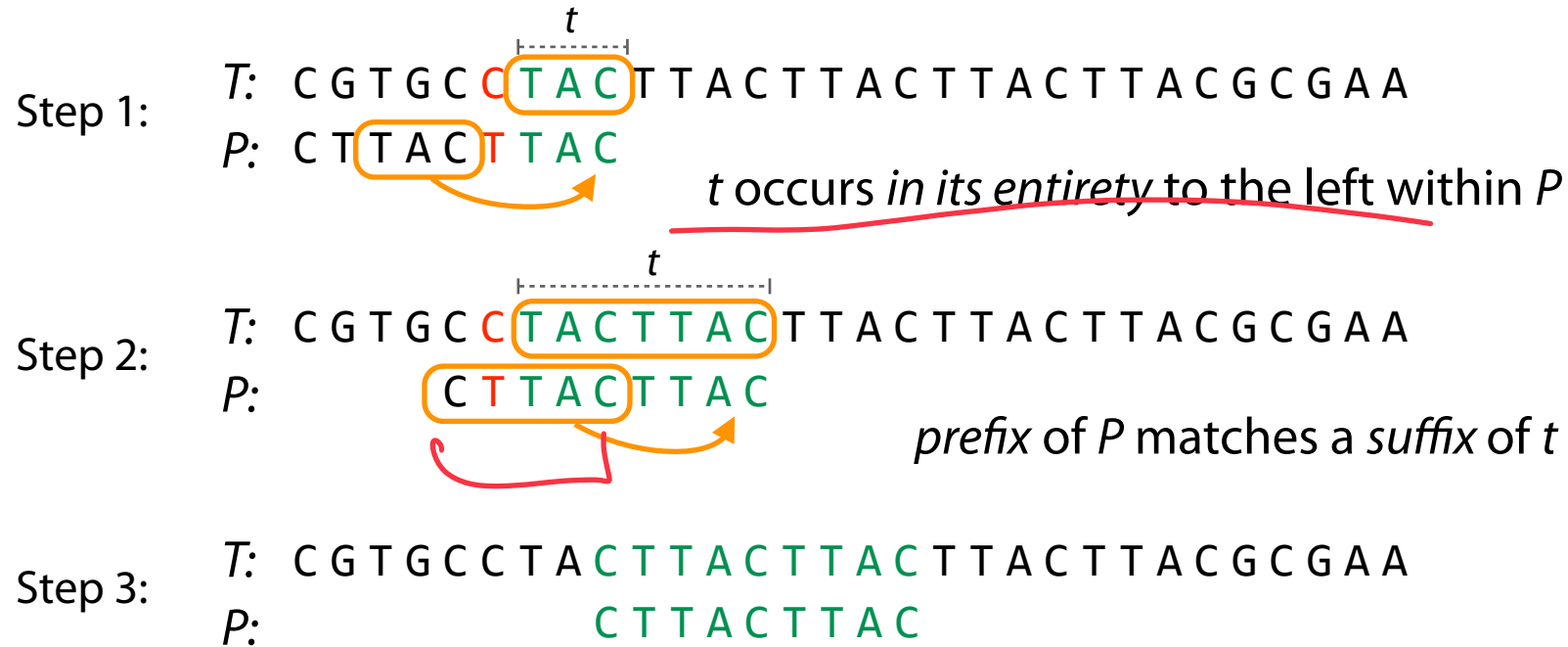
Any alignment that overlaps this region of the text must match the suffix ... *or have a prefix-suffix partial match!*

How many alignments do we skip?



“Weak” Good Suffix rule

Let t = longest suffix match at alignment; skip until (a) we find another **instance** of t or (b) P moves past t



An **instance** of t is either a full match to the left within P or a prefix of P matches a suffix of t

Boyer-Moore: Putting it together

How to combine bad character and good suffix rules?

T: G T T A T A G C T G A T C G C G G C G T A G C G G C G A A
P: G T A G C G G C G

How many characters does bad character skip? 2 characters

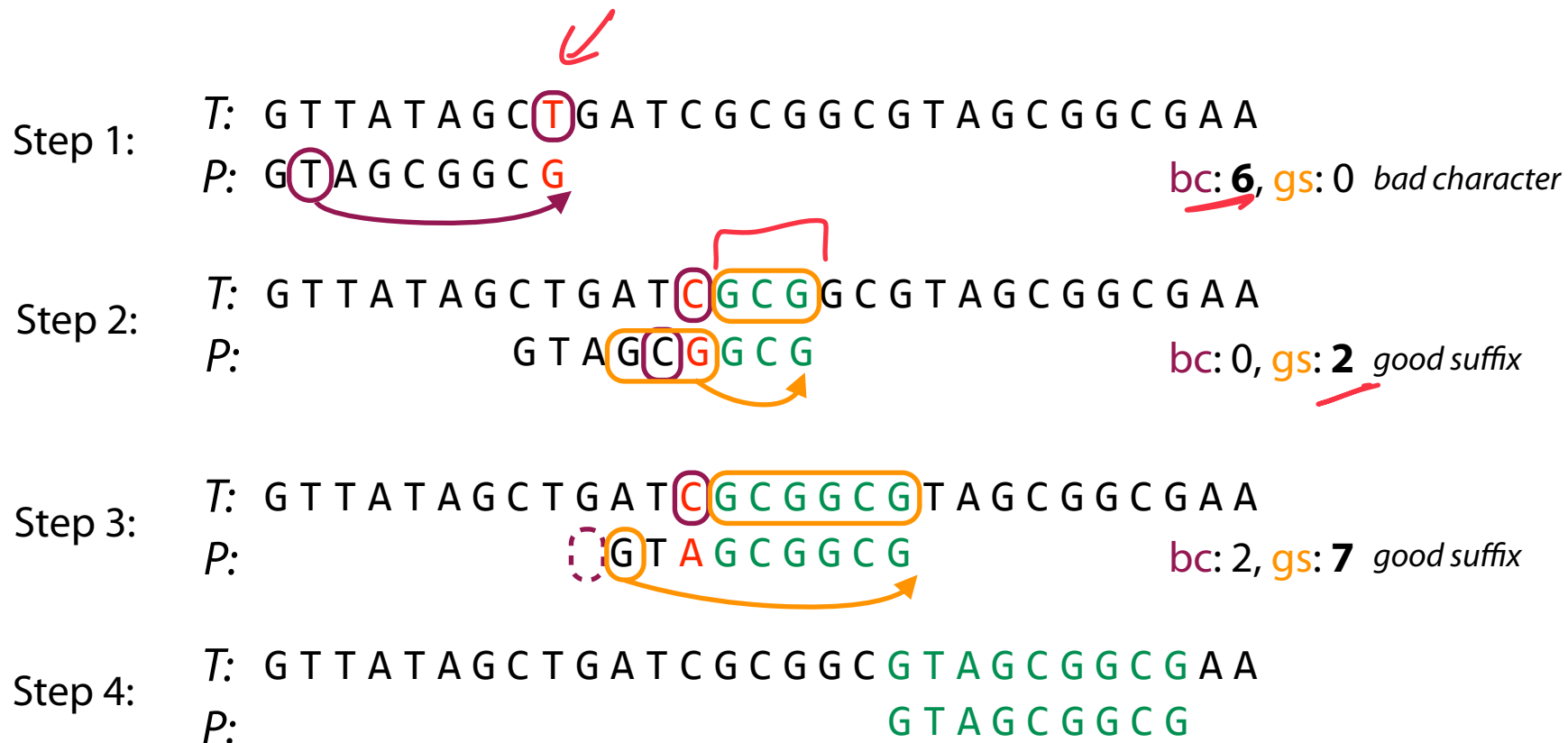
T: G T T A T A G C T G A T C G C G G C G T A G C G G C G A A
P: G T A G C G G C G

How many characters does good suffix skip? 7 characters

Take the maximum (7)!

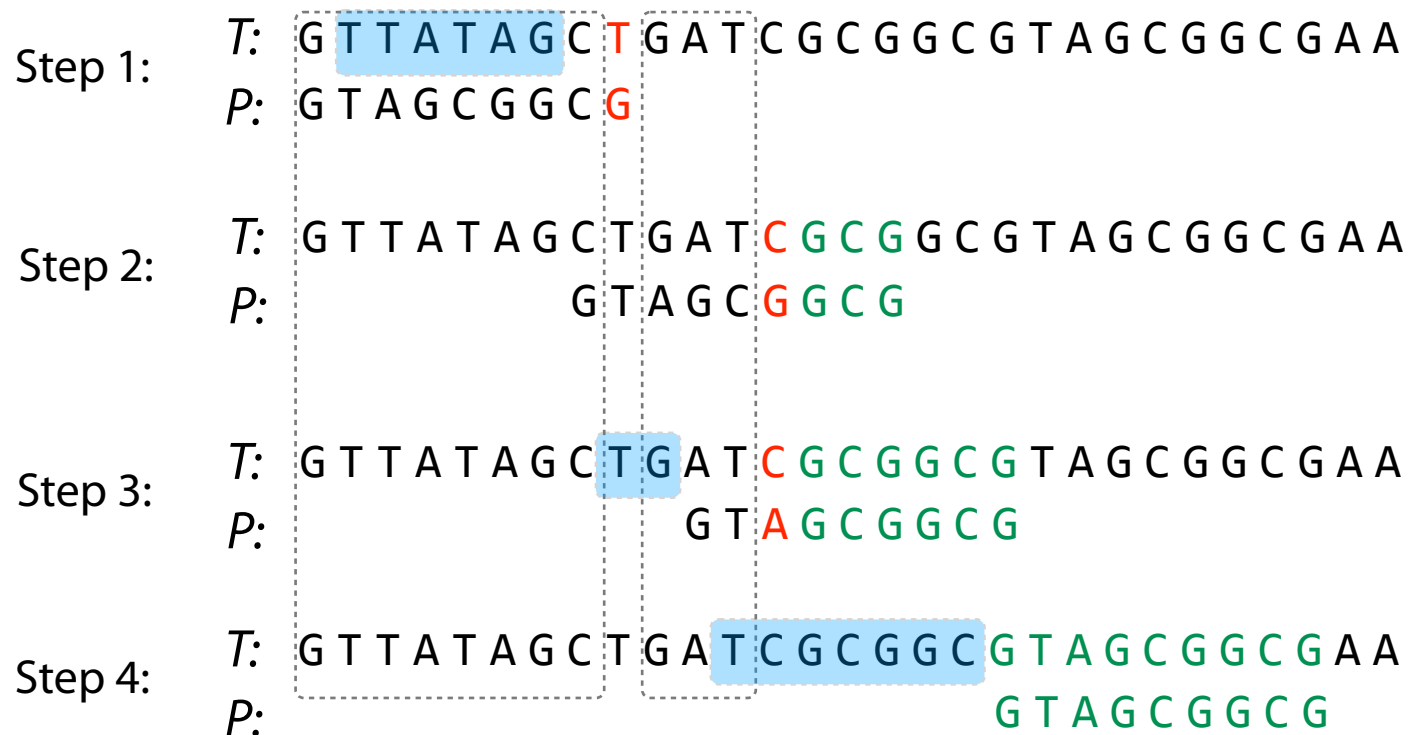
Boyer-Moore: Putting it together

Use bad character or good suffix rule, *whichever skips more*



Boyer-Moore: Putting it together

11 characters of *T* ignored completely!



Skipped 15 alignments



Boyer-Moore



Learn from character comparisons to skip pointless alignments

1. When we hit a mismatch c , move P along until c becomes a match (or P moves past c) “Bad character rule”
2. Try alignments in one direction, but do character comparisons in *opposite* direction “Right-to-left scanning”
3. When we move P along, make sure characters that matched in the last alignment also match in the next alignment “Good suffix rule”