String Algorithms and Data Structures

Boyer-Moore

CS 199-225
Brad Solomon

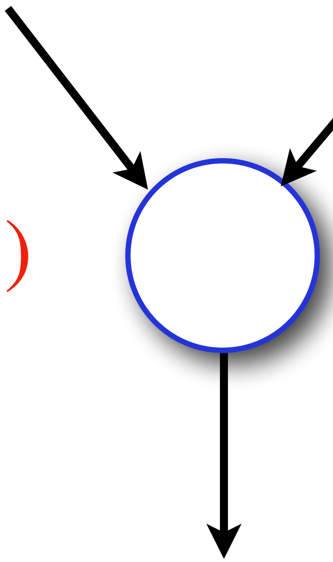
February 20, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

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

Why continue?

The Z-algorithm is:

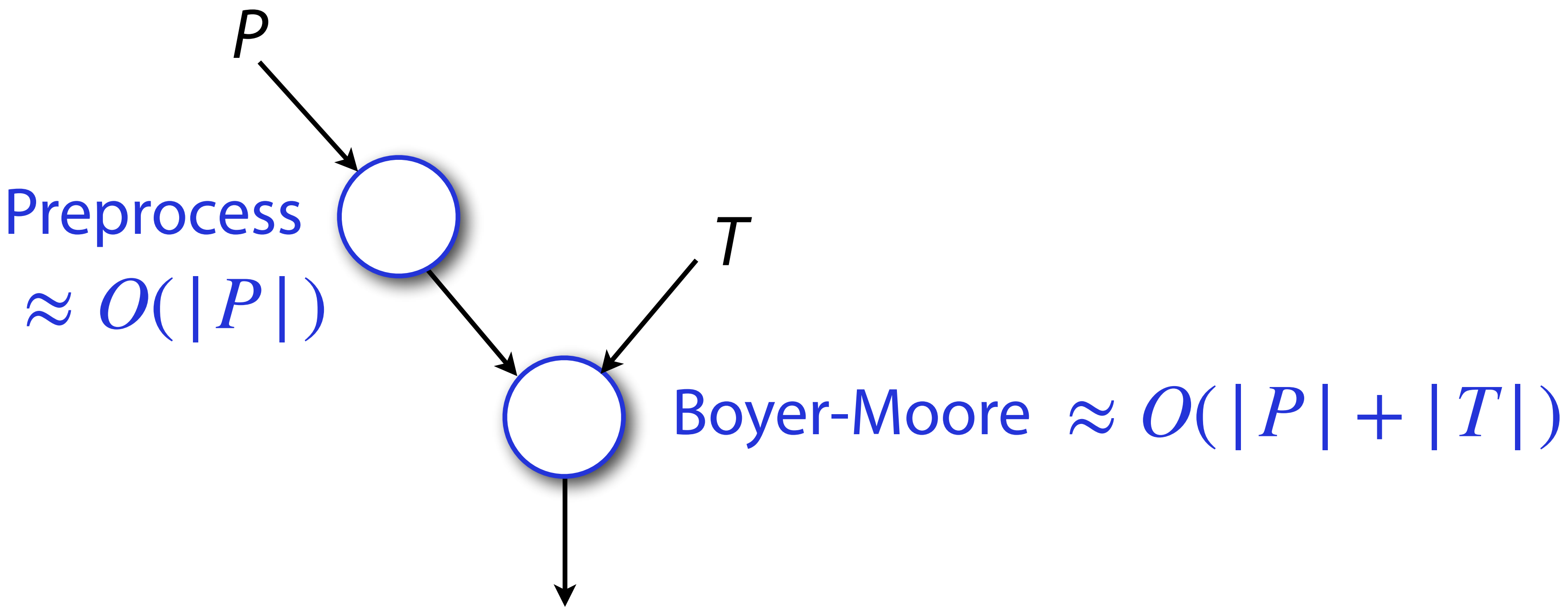
- The Z-algorithm is: $O(|P| + |T|)$ time
- An alphabet-independent solution

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



Find instances of P in T

'instances': An exact, full length copy

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?

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 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?

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

`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 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?

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

`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 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
cat
```

There is no 'r' in '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 …`

1) Our pattern doesn't match at this alignment

T: `woul`

P: `word`

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) 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: `word`

```
T: There would have been a …
           word ------------------->
   0123456789…
```

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: `word`

T: `There would have been a …`

```
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
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: 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

A A B B B

How many alignments can we skip? 1

AABAB

AABBB

There IS an A in 'AAABB'!

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

A A B B B

A A B B B skip!

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

How many alignments can we skip? 1

AABAB

AABBB

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. (c) If there was 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 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 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 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 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!

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 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 C C T T T T G C

skip!

We skipped three alignments

Can we do anything to make this better?

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 improvement

Continue to test alignment from left-to-right

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

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

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`

T: `woul`

P: `word`

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
```

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 there was 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 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 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 G A A
         P:                      C C T T T T G C                      Case (c)

Step 4:  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 G A A
         P:                        C C T T T T G C                    Case (a)

(etc)
```

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

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

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

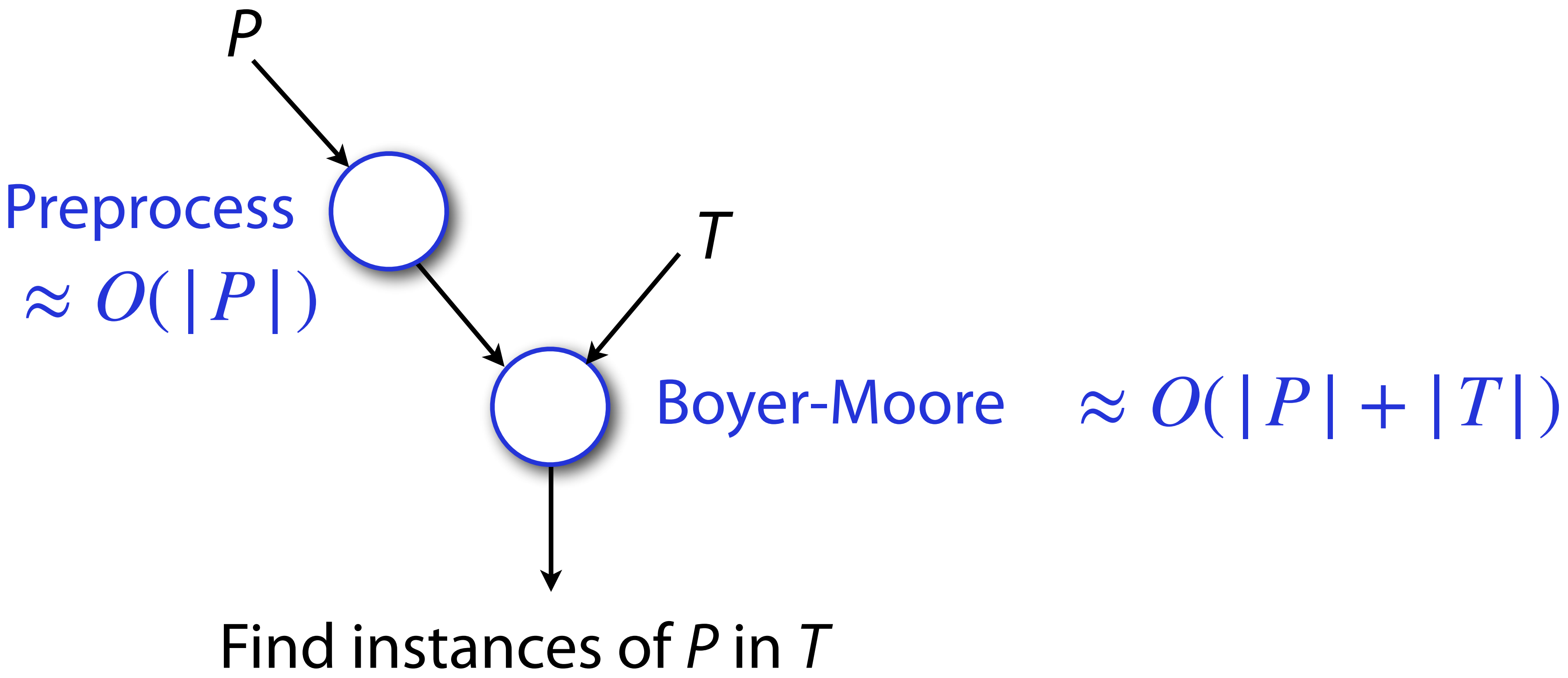
“Right-to-left scanning”

How do we put the first two rules in practice?

Boyer, RS and Moore, JS. "A fast string searching algorithm." *Communications of the ACM* 20.10 (1977): 762-772.

Exact pattern matching w/ Boyer-Moore

Boyer Moore **preprocesses** the pattern



'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

Σ

	T	C	G	C
A				
C				
G				
T				

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

Σ

	T	C	G	C
A				
C				
G				
T				

T: ? ? ? T ? ? ? ? ? ?
P: 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*

P

Σ	T	C	G	C
A				
C				
G				
T				2

T: ? ? ? T ? ? ? ? ? ?
P: 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*

P

Σ

	T	C	G	C
A				
C				
G				
T				2

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

P

Σ		T	C	G	C
	A				3
	C				
	G				
	T				2

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

P (columns), *Σ* (rows)

	T	C	G	C
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

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

Σ		B	A	B	A	A	A	B
	A	0	1	0	1	0	0	0
	B	0	0	1	0	1	2	3

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

P

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

T: T T T T T T T T T T

P: T C G C

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

P

Σ		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

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

P

Σ	T	C	G	C
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

Boyer-Moore: Tracking total skips

P

		A	A
Σ	A	0	0
	B	0	1

T: B B B B

T: B B B B B

T: B B B B B B

Boyer-Moore: Tracking total skips

P

		A	A	A
Σ	A	0	0	0
	B	0	1	2

T: B 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 $O(|P| + |T|)$?

Boyer, RS and Moore, JS. "A fast string searching algorithm." *Communications of the ACM* 20.10 (1977): 762-772.

Worst-Case Bad Character rule

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

Step 1:

```
T: A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
P: A A A
```

Case (c)

Step 2:

```
T: A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
P:   A A A
```

Case (c)

Step 3:

```
T: A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
P:     A A A
```

Case (c)

Step 4:

```
T: A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
P:       A A A
```

Case (c)

(etc)

Using just bad character, $O(|P||T|)$

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 G $\boxed{\text{A C}}$ 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
         A C A T A C
           A C A T A C
             A C A T A C
```

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

```
      A T C
        A T C
          A T C
            A T C
```

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?

"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

A T C

A T C

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 ________ somewhere in the pattern!

How many alignments do we skip?

"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
             G C A G C
               G C A G C
                 G C A G C
                   G C A G C
```

This is a full length match!

How many alignments do we skip?

"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*

Step 1:

t

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

P: C T T A C T T A C

t occurs *in its entirety* to the left within *P*

Step 2:

t

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

P: C T T A C T T A C

prefix of *P* matches a *suffix* of *t*

Step 3:

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

P: C T T A C T T A C

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*

Step 1:
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   bc: **6**, gs: 0 *bad character*

Step 2:
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   bc: 0, gs: **2** *good suffix*

Step 3:
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   bc: 2, gs: **7** *good suffix*

Step 4:
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

Boyer-Moore: Putting it together

11 characters of *T* ignored completely!

Step 1:
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

Step 2:
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

Step 3:
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

Step 4:
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

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"

Boyer, RS and Moore, JS. "A fast string searching algorithm." *Communications of the ACM* 20.10 (1977): 762-772.