Boyer-Moore

Ben Langmead

Johns Hopkins
Whiting School of Engineering

Department of Computer Science

Exact matching: slightly less naïve algorithm

```
P: word
T: There would have been a time for such a word
         word
```

We match w and o, then mismatch (r ≠ u)

Mismatched text character (u) doesn't occur in *P*

... since u doesn't occur in *P*, we can skip the next two alignments

```
P: word
T: There would have been a time for such a word
         word
          word   skip!
           word    skip!
            word
```

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Boyer-Moore

Use knowledge gained from character comparisons to skip future alignments that definitely won’t match:

1. If we mismatch, use knowledge of the mismatched text character to skip alignments — “Bad character rule”
2. If we match some characters, use knowledge of the matched characters to skip alignments — “Good suffix rule”
3. Try alignments in one direction, then try character comparisons in *opposite* direction — For longer skips

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

Boyer-Moore: Bad character rule

Upon mismatch, let *b* be the mismatched character in *T*. Skip alignments until (a) *b* matches its opposite in *P*, or (b) P moves past *b*.

```
                      b
Step 1:  T: G 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)

                                        b
Step 2:  T: G 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: G 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
```

(etc)

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Boyer-Moore: Bad character rule

Step 1:
T: G 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: G 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: G 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

We skipped 8 alignments

In fact, there are 5 characters in *T* we never looked at

Boyer-Moore: Bad character rule preprocessing

```
T:  G 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
```

As soon as *P* is known, build a $|\Sigma|$-by-n table. Say *b* is the character in *T* that mismatched and *i* is the mismatch's offset into *P*. The number of skips is given by element in *b*th row and *i*th column.

Gusfield 2.2.2 gives space-efficient alternative.

Boyer-Moore: Good suffix rule

Let *t* be the substring of *T* that matched a suffix of *P*. Skip alignments until (a) *t* matches opposite characters in *P*, or (b) a prefix of *P* matches a suffix of *t*, or (c) *P* moves past *t*, whichever happens first

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

Case (a)

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

Case (b)

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

Boyer-Moore: Good suffix rule

Like with the bad character rule, the number of skips possible using the good suffix rule can be precalculated into a few tables (Gusfield 2.2.4 and 2.2.5)

Rule on previous slide is the *weak* good suffix rule; there is also a *strong* good suffix rule (Gusfield 2.2.3)

```
                    t
               |-------|
T:  C T T G C C T A C T T A C T T A C T
P:  C T T A C T T A C
     Weak:  C T T A C T T A C
                  Strong:  C T T A C T T A C
```

guaranteed mismatch!

With the strong good suffix rule (and other minor modifications), Boyer-Moore is $O(m)$ *worst-case* time. Gusfield discusses proof.

Boyer-Moore: Putting it together

After each alignment, use bad character or good suffix rule, whichever skips more

Bad character rule:
Upon mismatch, let *b* be the mismatched character in *T*. Skip alignments until (a) *b* matches its opposite in *P*, or (b) *P* moves past *b*.

Good suffix rule:
Let t be the substring of *T* that matched a suffix of *P*. Skip alignments until (a) *t* matches opposite characters in *P*, or (b) a prefix of *P* matches a suffix of *t*, or (c) *P* moves past *t*, whichever happens first.

Step 1:
b
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
Part (a) of bad character rule

Step 2:
b *t*
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**
Part (a) of good suffix rule

Step 3:
b *t*
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**
Part (b) of good suffix rule

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

Johns Hopkins
Whiting School of Engineering

Boyer-Moore: Putting it together

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

Up to now: 15 alignments skipped, 11 text characters never examined

Boyer-Moore: Worst and best cases

Boyer-Moore (or a slight variant) is $O(m)$ worst-case time

What's the best case?

> Every character comparison is a mismatch, and bad character rule always slides P fully past the mismatch
>
> How many character comparisons? *floor(m / n)*

Contrast with naive algorithm

Performance comparison

Comparing simple Python implementations of naïve exact matching and Boyer-Moore exact matching:

	Naïve matching		Boyer-Moore		
	# character comparisons	wall clock time	# character comparisons	wall clock time	
P: "tomorrow" **T**: Shakespeare's complete works	5,906,125	2.90 s	785,855	1.54 s	17 matches $\|T\| = 5.59$ M
P: 50 nt string from Alu repeat* **T**: Human reference (hg19) chromosome 1	307,013,905	137 s	32,495,111	55 s	336 matches $\|T\| = 249$ M

* GCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGG

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING