

# String Searches

Thomas Schwarz, SJ

# Problem

- We are given a long string (*text*)
  - such as a book or a genome
- We are given a short string (*pattern*)
- We want to find where the shorter string is located in the longer one

# Naïve Algorithm

- We slide the pattern successively through the text
- We compare the letters in the pattern with the text
- If two letters differ, go to the next location
- If we reach the end, we have found a match

# Naïve Algorithm

- Example

M A L L S R T A A E E V I A S F T E E Q A V L A L T N V E K D K

A L L A S F T E

- No match on first character: move pattern by one

M A L L S R T A A E E V I A S F T E E Q A V L A L T N V E K D K

A L L A S F T E

- After sliding, three letters coincide, but then we have a mismatch: move pattern by one

# Naïve Algorithm

- Example:

M	A	L	L	S	R	T	A	A	E	E	V	I	A	S	F	T	E	E	Q	A	V	L	A	L	T	N	V	E	K	D	K
		A	L	L	A	S	F	T	E																						

- No match on first, slide

M	A	L	L	S	R	T	A	A	E	E	V	I	A	S	F	T	E	E	Q	A	V	L	A	L	T	N	V	E	K	D	K
			A	L	L	A	S	F	T	E																					

- No match on first, slide

- ...

# Naïve Algorithm

- What are the costs:
  - At best, we compare each letter of the text with a letter in the pattern
  - $n$ : length of pattern
  - $m$ : length of string
- Best time:  $\Theta(n)$
- Worst time:  $\Theta(nm)$

# Naïve Algorithm

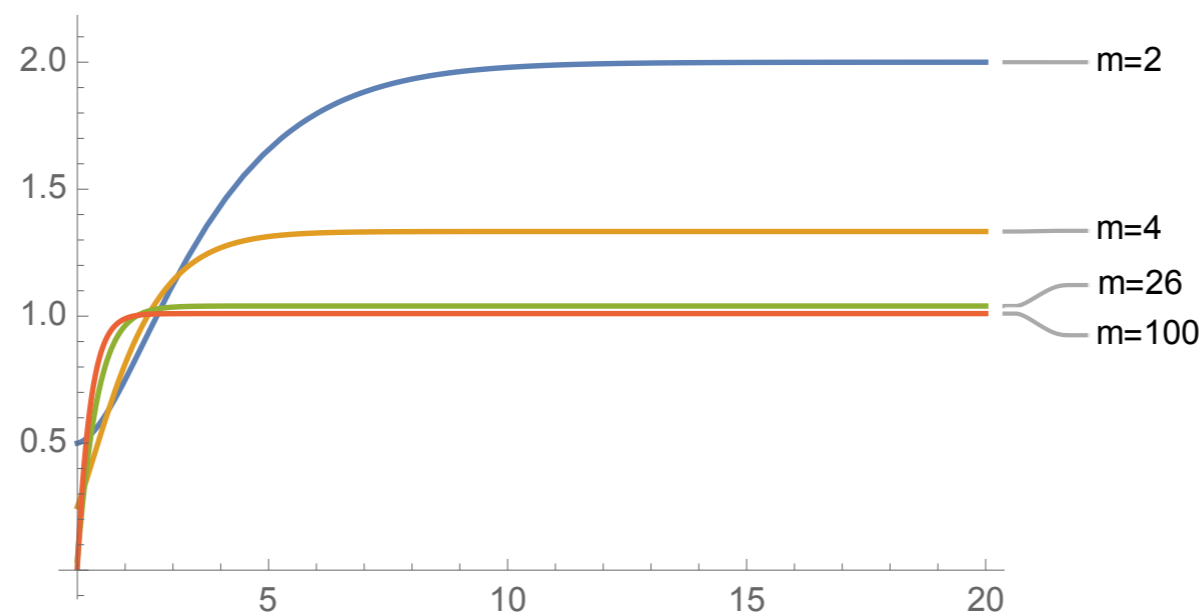
- Average time:
  - Depends on how likely matches between letters are
  - If we assume there are  $c$  characters and all are equally likely and the appearance of a character is independent of its neighbors and ... :
  - Probability of a character matching is  $1/c$
  - Expected number of characters compared is
  - $$\frac{c-1}{c} \cdot 1 + \frac{c-1}{c^2} \cdot 2 + \dots + \frac{c-1}{c^{m-1}} \cdot (m-1) + \frac{1}{c^m} \cdot m$$

# Naïve Algorithm

- Average time:

- $$\frac{c-1}{c} \cdot 1 + \frac{c-1}{c^2} \cdot 2 + \dots + \frac{c-1}{c^{m-1}} \cdot (m-1) + \frac{1}{c^m} \cdot m$$

- Converges quickly to  $\frac{c}{c-1}$





# Naïve Algorithm

- Thus:
  - Average number of comparisons is close to 1

# Boyer Moore Algorithm

- How can we do better?
  - Need to be able to slide the pattern further
  - But for this we need to foresee the text
  - That is why it is better to compare the pattern and the text from the right



# Boyer Moore Algorithm

M A L L S R T A A E E V I A S F T E E Q A V L A L T N V E K D K  
A L L A S F T E

- So we slide four to the right

M A L L S R T A A E E V I A S F T E E Q A V L A L T N V E K D K  
A L L A S F T E

- And then compare at the new location

M A L L S R T A A E E V I A S F T E E Q A V L A L T N V E K D K  
A L L A S F T E

# Boyer Moore Algorithm

M	A	L	L	S	R	T	A	A	E	E	V	I	A	S	F	T	E	E	Q	A	V	L	A	L	T	N	V	E	K	D	K					
											A	L	L	A	S	F	T	E																		

- The 'V' does not appear in the string at all
- So we can slide by the length of the pattern

M	A	L	L	S	R	T	A	A	E	E	V	I	A	S	F	T	E	E	Q	A	V	L	A	L	T	N	V	E	K	D	K					
											A	L	L	A	S	F	T	E																		

# Boyer Moore Algorithm

- To implement the "bad character" at the end, we need to process the string
  - Shift is the smallest distance of the bad character to the end in the pattern or the length of the pattern

A	L	L	A	S	F	T	E
---	---	---	---	---	---	---	---

A:	4
E:	0
F:	2
L:	5
S:	3
T:	1



# Boyer Moore Algorithm

- This is not the only knowledge that we can use
  - Assume we have already matched part of the pattern, but now have a disagreement
  - This means that we know a part of the text

M A L L S R T A A E A L L A S A T E E Q A V L A L T N V E K D K A L L A S F T E Q A L S A L

A L L A S F T E

- Where can 'ATE' be matched in the pattern?
- Answer: not at all



# Boyer Moore Algorithm

- We **preprocess** the pattern
  - For each letter we find the minimum distance to the end
  - For each suffix, we find the minimum distance of another copy of the suffix to the end
  - Or: If the alphabet is small:
    - Where can the suffix preceded by a single letter be found

# Boyer Moore Algorithm

- Example:
  - pattern: 011001001
    - match "1": where can "11" be found: distance 6
    - match "01" where can "101" be found:
      - **01**1001001: shift 7
    - match "001": where is "0001":
      - **01**1001001: shift 7
    - match "01001": where is "101001" :
      - **01**1001001: shift 7

# Boyer Moore Algorithm

- Both rules give usually different safe shift amounts
  - Always use the larger one

# Boyer Moore Algorithm

- Example

```
0 1 0 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 0
0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0
```

- Bad letter: shift 1
- Good suffix "": shift 1

# Boyer Moore Algorithm

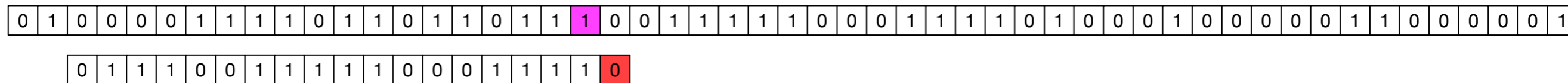
- Example:

0	1	0	0	0	0	1	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0	1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1
0	1	1	1	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1

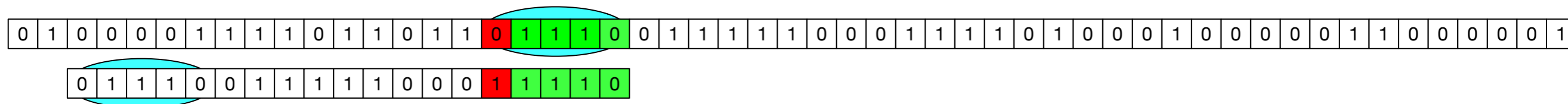
- Bad character: shift 1

# Boyer Moore Algorithm

- Example



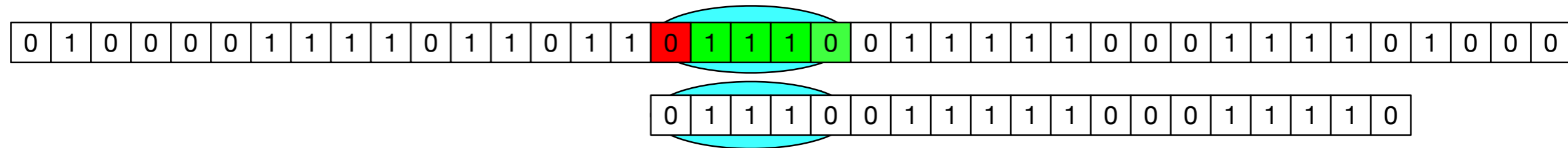
- Compare:



- Bad character rule: shift by one
- Good suffix rule: shift by 14

# Boyer Moore Algorithm

- Example



- After shift, we find a match
- Then we shift by one

# Boyer Moore Algorithm

- Your turn:
  - Preprocess "AGGTAA"
  - Bad character table
  - Bad suffix table



# Boyer Moore

- Analysis is very difficult
  - Worst case:
    - Pattern and text consists of a single letter
      - $\sim n$  comparisons
  - Best case:
    - Pattern and text have completely different letters
    - $\lfloor \frac{n}{m} \rfloor$  comparisons

# Boyer Moore

- Analysis is very difficult
  - Speed-up usually substantial
  - Called a "sub-linear" algorithm

# Variants:

- Only the bad character rule:
  - Boyer-Moore-Horspool:
    - Only bad character rule
  - Apostolico-Giancarlo
    - Uses the pattern preprocessing in order to not compare letters that are known to be good
- Instead of a single bad character:
  - Use pairs of characters

# Evaluation

- Algorithm comparison depends on the model
- Experimental evaluation:
  - Define and find "typical scenarios"
  - Use statistics to compare results