Fast String Searching

J Strother Moore
Department of Computer Sciences
University of Texas at Austin

The Problem

One of the classic problems in computing is *string searching*: find the first occurrence of one character string ("the *pattern*") in another ("the *text*").

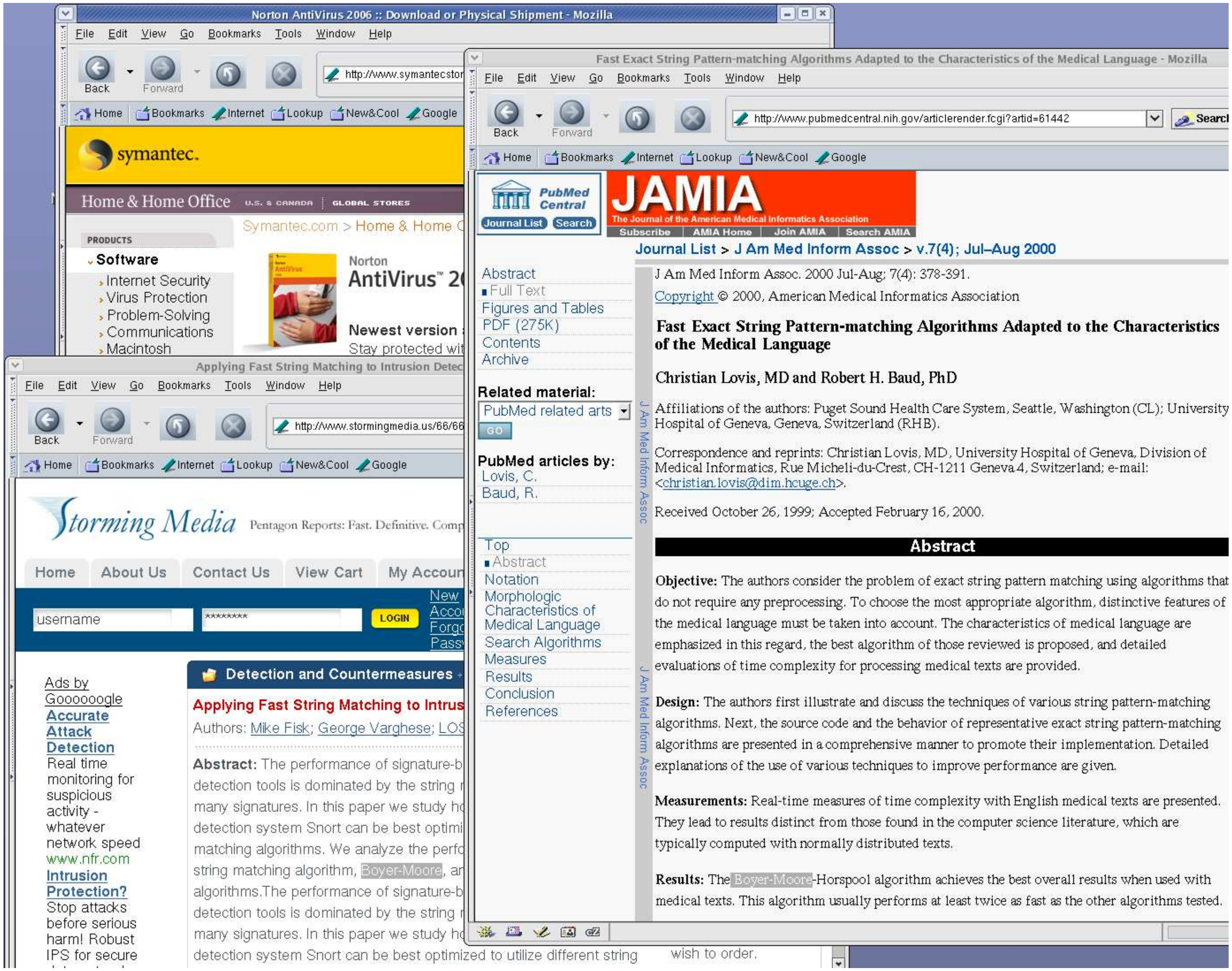
Generally, the text is *very* large (e.g., gigabytes) but the patterns are relatively small.

Examples

Find the word “comedy” in this *NY Times* article:

> Fred Armisen’s office at “Saturday Night Live” is deceptively small, barely big enough to fit a desk, a couch, and an iPod. The glorified closet, the subject of a running joke on the comedy show, now in its 31st season, can simultaneously house a wisecracking . . .

```
AAAAAAAAAAAAACAAAGACAGGGGCAACAAAGTGAGACCCTAAAAAAAAAAAAACCCCA
AAACGGAGAACTTGGAATCCTGTGTCCAAAAAAAAAAGCAGGAAGAGAGCGTGTAGAAAC
TGAAGCTGAAGTGGAAAAAAAAAAGTCGCCAGCACCTACTGTGGAGACCAGAAAGGAAAA
AAAAAATTGGCAGTCTCGTAGCATACCAAAACTAGGCTTGAAAAAAAAAACACACAAAAA
AACACAGGCTACCCAGTATTTTATCGTCCAAAAAAAAAGAGGGAAGAAGGACATTTATAT
TTGCCTTCTGCCAAAAAAAAAAGTACCTCCCGCCTAGAAGAGAGTTTAGAAATCACCAAA
AAAAAATAGAGAGTCCCAAAATGTTCGGAATACTCAGAAAAAAAAATCTTAGTCAGTGCT
CACTCAGAGGGACCGGGTATTTAAAAAAAACCTAGACCAGATGCAGCAGGTACAAATTAA
TCAATCCCAAAAAAAAGACCTTCTACCCTTCCAAAAAATGATAGTTGTCTGCAATCCAAA
AAAAAGACTCTCCGGAAGGTGGACATGCAGAACCTACCAAAAAAAAAGAGAAGAAAGAAT
TGCCGGGCAAAAAGTTCCACGTAAAAAAAAAAGGAAATGGGAATGGAGTGTTGTTCTCCT
TCCTACCTAGTTTTGAAAAAAAAGGATGGATGTGGGTCACCTGCTCACGTTCTCCAAAAA
AAAGTGGGTGCTCTCTCACAATATTCTTAGAGGTGGCAAAAAAAATAAAGTTGATGGAAA
CAGTACTGTGTGGGCCAAACAAAAAAAAAATGGCACCACCTTTTCATTGGCTGAAAAAAA
AATTCAACTGAAAAACACAAGTCATACCTTCCTGTTTTATTTGCAAAAAAAATTTTTCAA
ACCCCACGGCAACAAACGACAGTATCAAAAAAACAACTTCATTTGACATTCTGCTATATT
AATGCTCTATGTGGAAAAAAAAACCATCAAGTTGTGCCTTTTTTCAAAGAAATCCATGCA
AAAAAAAGACCCATGAAATAATTTTCTGGATCATCCATACAGAACCAAAAAAAAGAGGTG
```


Norton AntiVirus 2006 :: Download or Physical Shipment - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward
http://www.symantecstor
Home Bookmarks Internet Lookup New&Cool Google
symantec.
Home & Home Office U.S. & CANADA GLOBAL STORES
Symantec.com > Home & Home O
PRODUCTS
Software
Internet Security
Virus Protection
Problem-Solving
Communications
Macintosh
Norton
AntiVirus™ 20
Newest version
Stay protected wit
Fast Exact String Pattern-matching Algorithms Adapted to the Characteristics of the Medical Language - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward
http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=61442
Search
Home Bookmarks Internet Lookup New&Cool Google
PubMed Central
Journal List Search
JAMIA
The Journal of the American Medical Informatics Association
Subscribe AMIA Home Join AMIA Search AMIA
Abstract
Full Text
Figures and Tables
PDF (275K)
Contents
Archive
Related material:
PubMed related arts
GO
PubMed articles by:
Lovis, C.
Baud, R.
Top
Abstract
Notation
Morphologic Characteristics of Medical Language
Search Algorithms
Measures
Results
Conclusion
References
J Am Med Inform Assoc
J Am Med Inform Assoc
Applying Fast String Matching to Intrusion Detec
File Edit View Go Bookmarks Tools Window Help
Back Forward
http://www.stormingmedia.us/66/66
Home Bookmarks Internet Lookup New&Cool Google
Storming Media
Pentagon Reports: Fast. Definitive. Comp
Home About Us Contact Us View Cart My Accoun
username
****
LOGIN
New
Acco
Forg
Pass
Ads by Goooooogle
Accurate Attack Detection
Real time monitoring for suspicious activity - whatever network speed
www.nfr.com
Intrusion Protection?
Stop attacks before serious harm! Robust IPS for secure
wish to order.

Journal List > J Am Med Inform Assoc > v.7(4); Jul–Aug 2000

J Am Med Inform Assoc. 2000 Jul-Aug; 7(4): 378-391.

Copyright © 2000, American Medical Informatics Association

Fast Exact String Pattern-matching Algorithms Adapted to the Characteristics of the Medical Language

Christian Lovis, MD and Robert H. Baud, PhD

Affiliations of the authors: Puget Sound Health Care System, Seattle, Washington (CL); University Hospital of Geneva, Geneva, Switzerland (RHB).

Correspondence and reprints: Christian Lovis, MD, University Hospital of Geneva, Division of Medical Informatics, Rue Micheli-du-Crest, CH-1211 Geneva 4, Switzerland; e-mail: <christian.lovis@dim.hcuge.ch>.

Received October 26, 1999; Accepted February 16, 2000.

Abstract

Objective: The authors consider the problem of exact string pattern matching using algorithms that do not require any preprocessing. To choose the most appropriate algorithm, distinctive features of the medical language must be taken into account. The characteristics of medical language are emphasized in this regard, the best algorithm of those reviewed is proposed, and detailed evaluations of time complexity for processing medical texts are provided.

Design: The authors first illustrate and discuss the techniques of various string pattern-matching algorithms. Next, the source code and the behavior of representative exact string pattern-matching algorithms are presented in a comprehensive manner to promote their implementation. Detailed explanations of the use of various techniques to improve performance are given.

Measurements: Real-time measures of time complexity with English medical texts are presented. They lead to results distinct from those found in the computer science literature, which are typically computed with normally distributed texts.

Results: The Boyer-Moore-Horspool algorithm achieves the best overall results when used with medical texts. This algorithm usually performs at least twice as fast as the other algorithms tested.

Detection and Countermeasures

Applying Fast String Matching to Intrus

Authors: Mike Fisk; George Varghese; LOS

Abstract: The performance of signature-b
detection tools is dominated by the string m
many signatures. In this paper we study ho
detection system Snort can be best optimi
matching algorithms. We analyze the perfo
string matching algorithm, Boyer-Moore, ar
algorithms.The performance of signature-b
detection tools is dominated by the string m
many signatures. In this paper we study ho
detection system Snort can be best optimized to utilize different string

Variants of the problem allow *wildcards* in the pattern and/or the text. *Exact* matching is when no wildcards are allowed.

We describe the fastest sequential algorithm for solving the exact string searching problem. The algorithm is called the *Boyer-Moore fast string searching algorithm*.

Example

Find the word “comedy” in this *NY Times* article:

> Fred Armisen’s office at “Saturday Night Live” is deceptively small, barely big enough to fit a desk, a couch, and an iPod. The glorified closet, the subject of a running joke on the comedy show, now in its 31st season, can simultaneously house a wisecracking . . .

COMEDY

JOKE ON THE COMEDY

COMEDY

🟥🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦

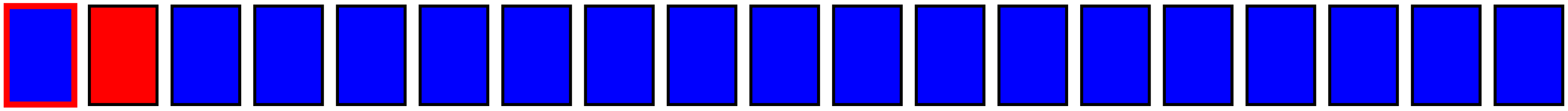
JOKE ON THE COMEDY

COMEDY

J	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

JOKE ON THE COMEDY

COMEDY

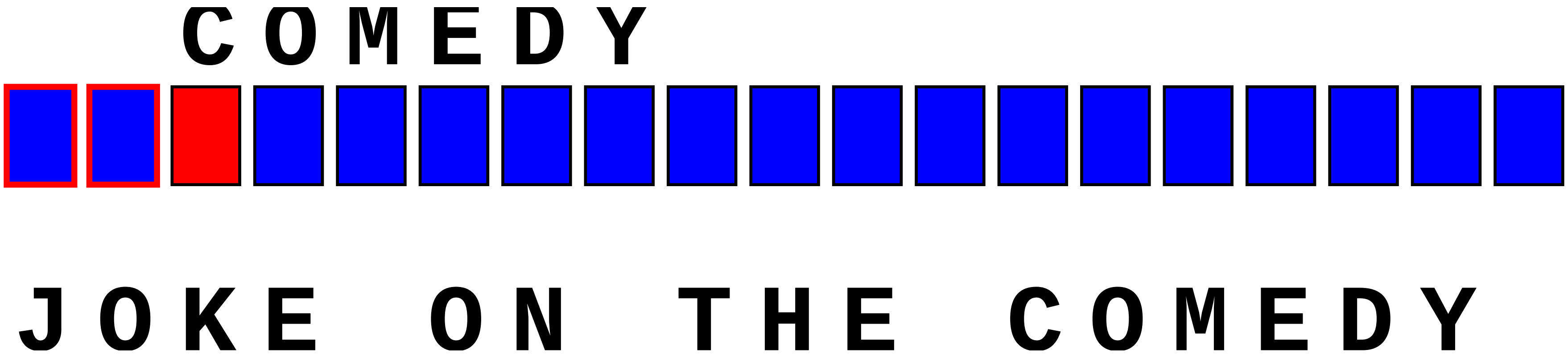


JOKE ON THE COMEDY

COMEDY

🟦	O	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

JOKE ON THE COMEDY



COMEDY

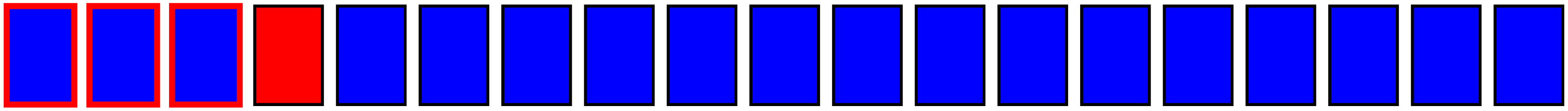
JOKE ON THE COMEDY

COMEDY

🟦	🟦	K	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

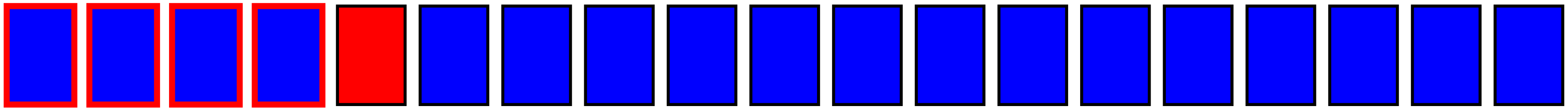
JOKE ON THE COMEDY

COMEDY



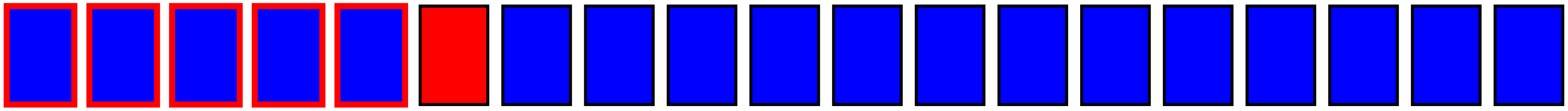
JOKE ON THE COMEDY

COMEDY



JOKE ON THE COMEDY

COMEDY



JOKE ON THE COMEDY

COMEDY

🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦COMEDY🟦

JOKE ON THE COMEDY

COMEDY

🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦

JOKE ON THE COMEDY

COMEDY

🟦🟦🟦🟦🟦🟥🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦

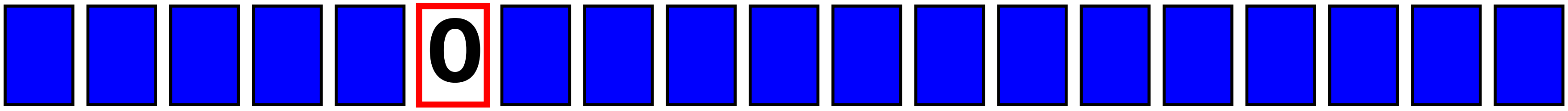
JOKE ON THE COMEDY

COMEDY

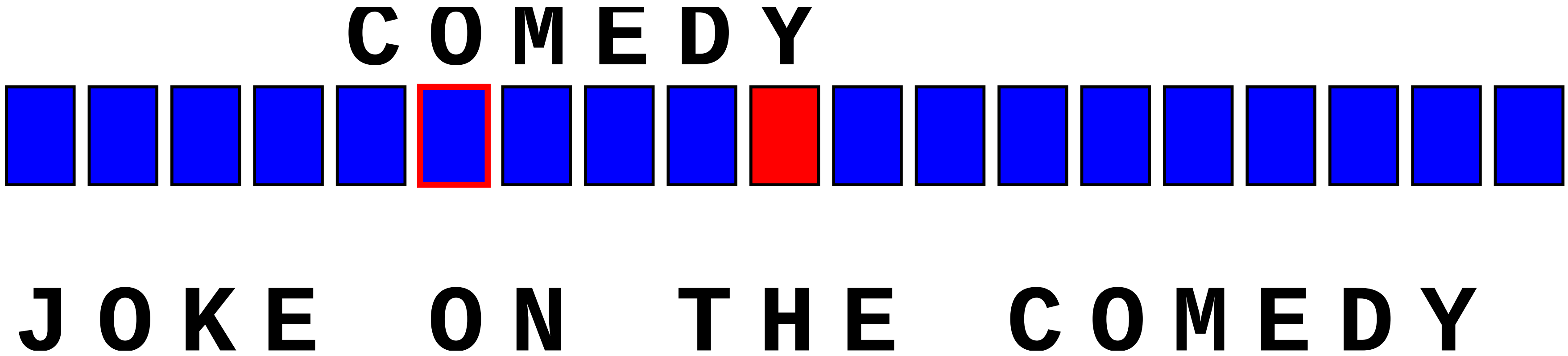
					O													

JOKE ON THE COMEDY

COMEDY



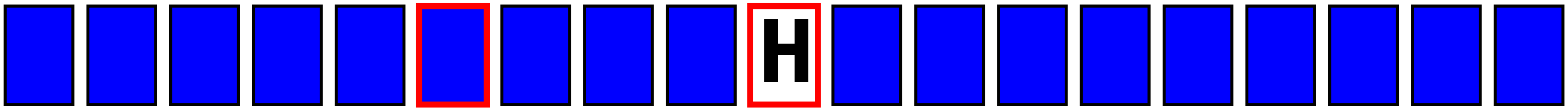
JOKE ON THE COMEDY



COMEDY

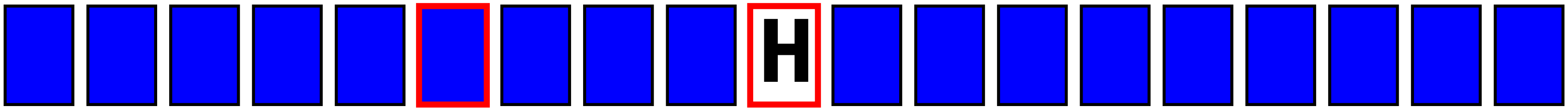
JOKE ON THE COMEDY

COMEDY



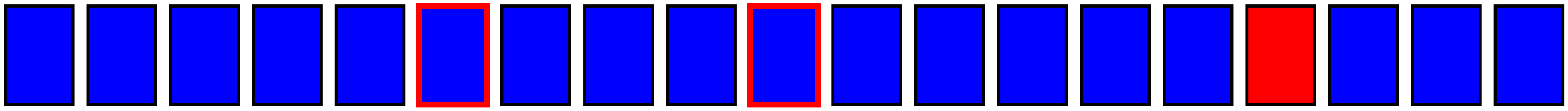
JOKE ON THE COMEDY

COMEDY



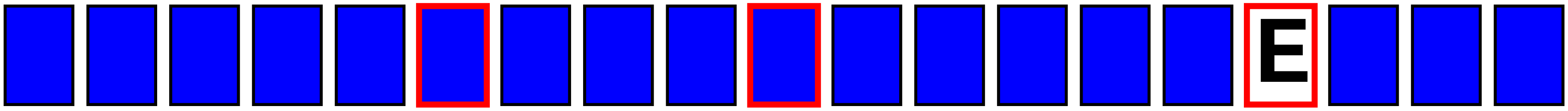
JOKE ON THE COMEDY

COMEDY



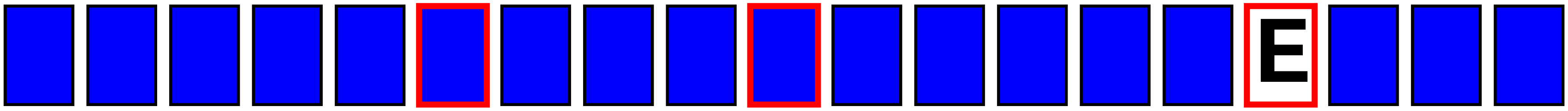
JOKE ON THE COMEDY

COMEDY

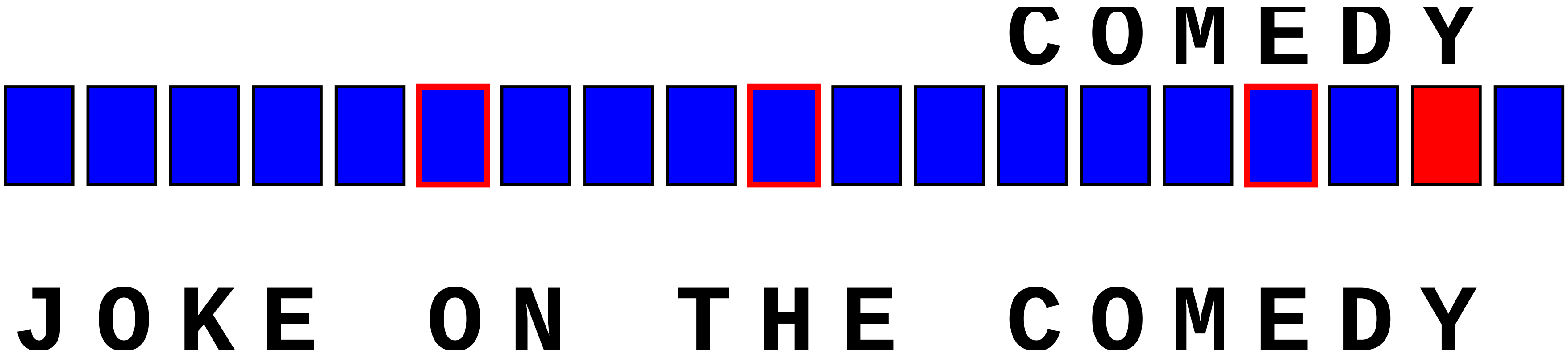


JOKE ON THE COMEDY

COMEDY

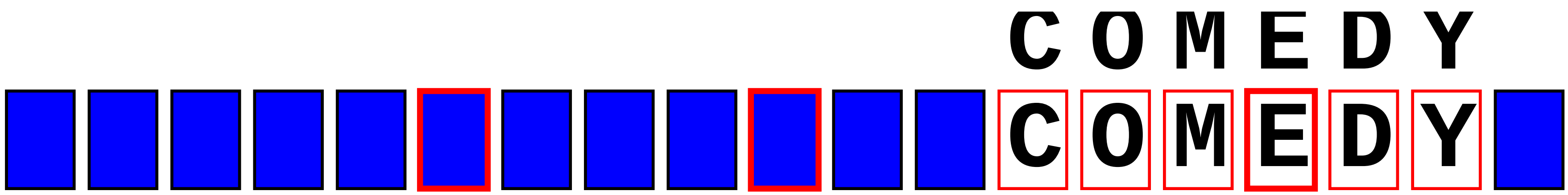


JOKE ON THE COMEDY



COMEDY

JOKE ON THE COMEDY



JOKE ON THE COMEDY

Key Property: The longer the pattern, the faster the search!

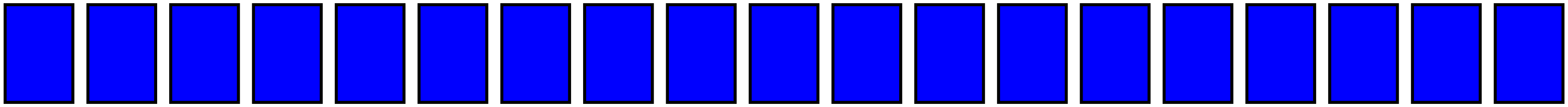
Pre-Computing the Skip Distance

```
pat: 543210
     COMEDY
txt: xxxxxOxxxxxxxxxxx...
          ↑
```

A	6
B	6
C	5
D	1
E	2
F	6
G	6
H	6
I	6
J	6
K	6
L	6
M	3
N	6
O	4
P	6
Q	6
R	6
S	6
T	6
U	6
V	6
W	6
X	6
Y	0
Z	6

This is a 1-dimensional array, `skip`[c], as big as the alphabet.

COMEDY

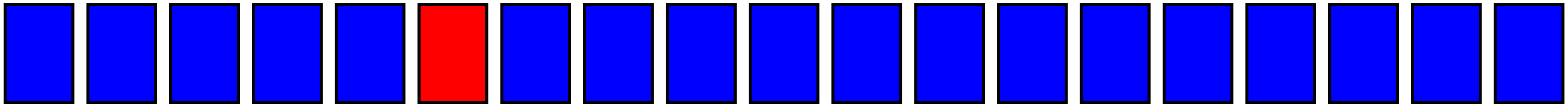


JOKE ON THE COMEDY

skip[c]:

A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6

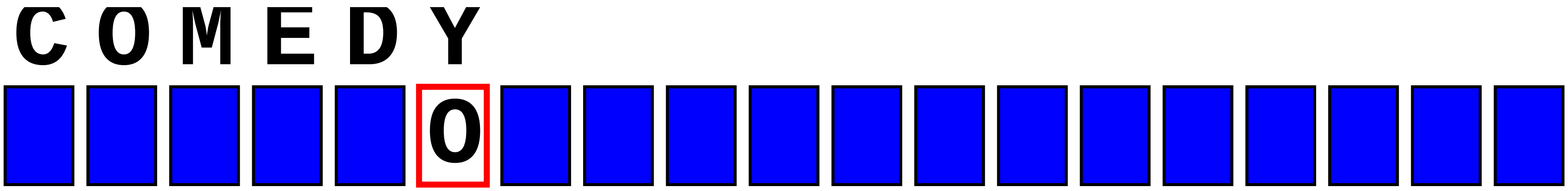
COMEDY



JOKE ON THE COMEDY

skip[c]:

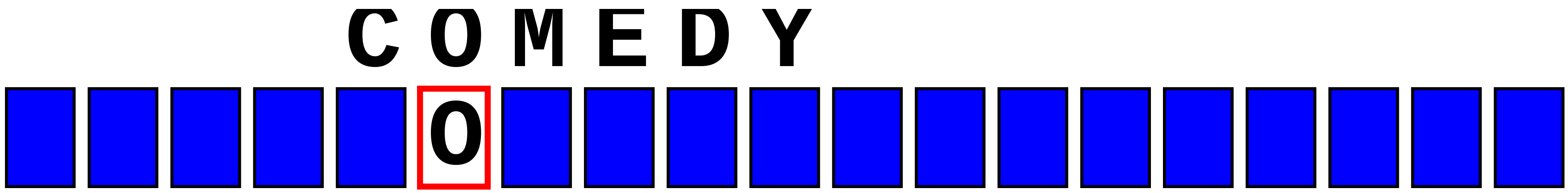
A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6



JOKE ON THE COMEDY

skip[c]:

A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6



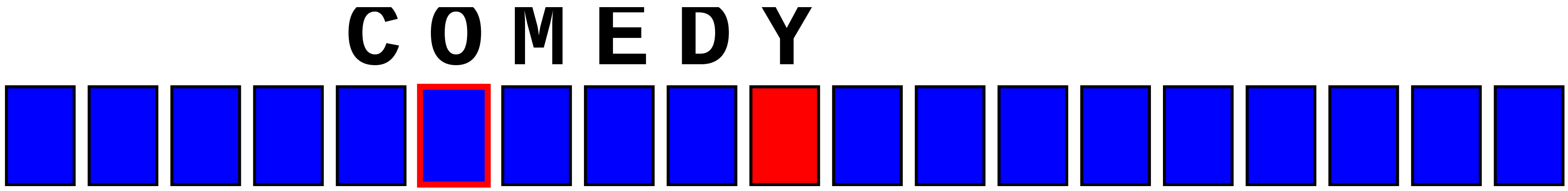
COMEDY

O

JOKE ON THE COMEDY

skip[c]:

A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6

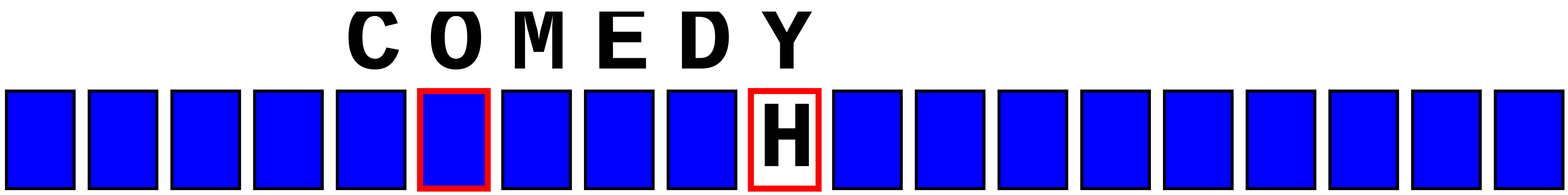


COMEDY

JOKE ON THE COMEDY

skip[c]:

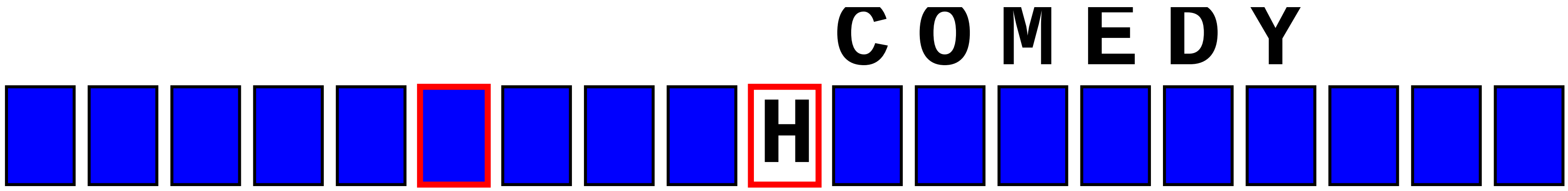
A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6



JOKE ON THE COMEDY

skip[c]:

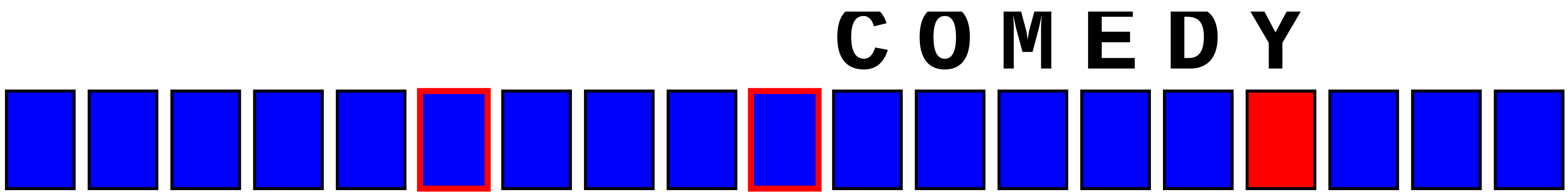
A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6



JOKE ON THE COMEDY

skip[c]:

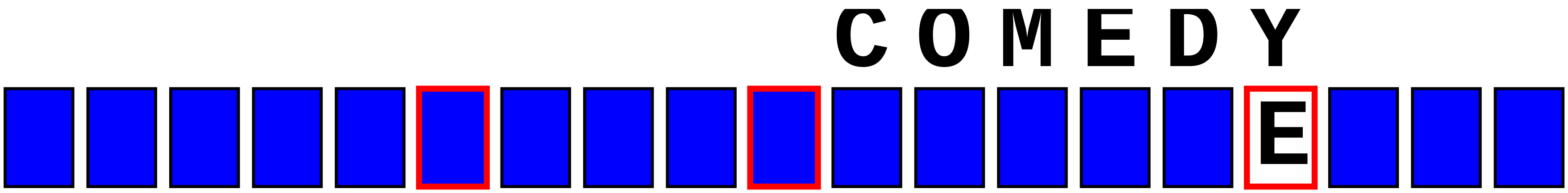
A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6



JOKE ON THE COMEDY

skip[c]:

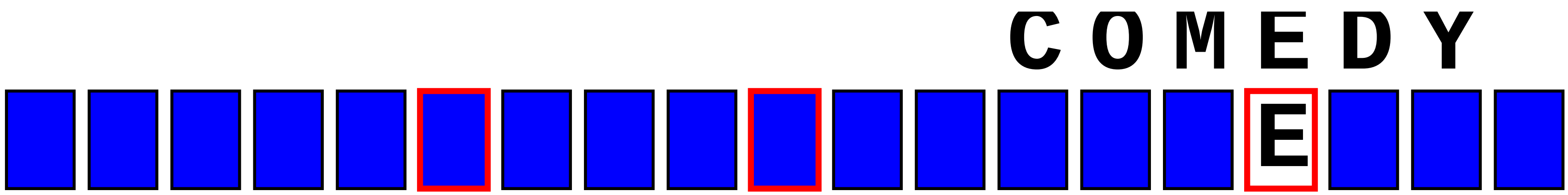
A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6



JOKE ON THE COMEDY

skip[c]:

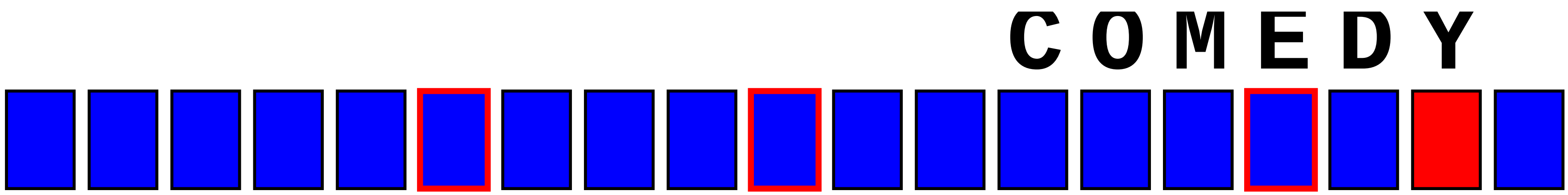
A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6



JOKE ON THE COMEDY

skip[c]:

A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6

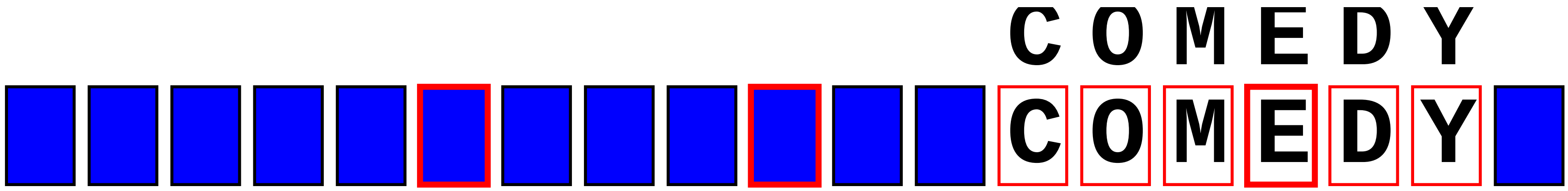


COMEDY

JOKE ON THE COMEDY

skip[c]:

A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6



JOKE ON THE COMEDY

skip[c]:

A	6	F	6	K	6	P	6	U	6
B	6	G	6	L	6	Q	6	V	6
C	5	H	6	M	3	R	6	W	6
D	1	I	6	N	6	S	6	X	6
E	2	J	6	O	4	T	6	Y	0
								Z	6

But Wait! There’s More!

```
pat: NONPARTIPULAR
txt: -----------------------
                 |
```

But Wait! There's More!

```
pat: NONPARTIPULAR
txt: ------------R----------
                 |
```

But Wait! There's More!

```
pat: NONPARTIPULAR
txt: -----------A-----------
                |
```

But Wait! There’s More!

```
pat: NONPARTIPULAR
txt: ----------P------------
               |
```

But Wait! There's More!

```
pat:   NONPARTIPULAR
txt: ----------P------------
               |
```

Slide 2 to match the discovered character.

But Wait! There's More!

```
pat:   NONPARTIPULAR
txt: ----------P??----------
               |
```

But Wait! There's More!

```
pat:   NONPARTIPULAR
txt: ----------PAR----------
               |
```

But Wait! There's More!

```
pat: NONPARTIPULAR
txt: -----------------------
                 |
```

But Wait! There's More!

```
pat: NONPARTIPULAR
txt: ------------R----------
                 |
```

But Wait! There's More!

```
pat: NONPARTIPULAR
txt: -----------AR----------
                |
```

But Wait! There's More!

```
pat: NONPARTIPULAR
txt: ----------PAR----------
               |
```

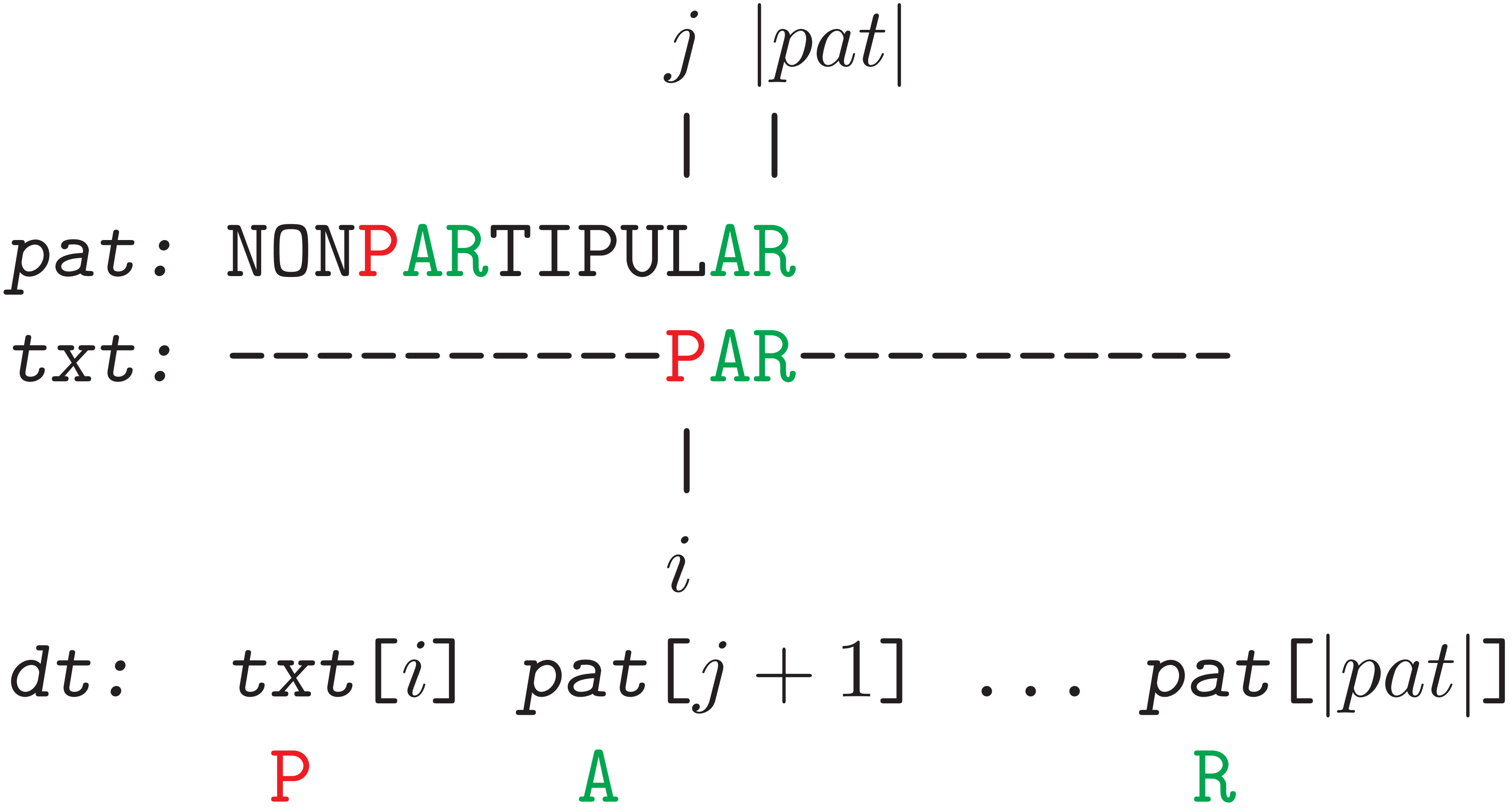
But Wait! There's More!

```
pat: NONPARTIPULAR
txt: ----------PAR----------
               |
```

But Wait! There's More!

```
pat:        NONPARTIPULAR
txt: ----------PAR----------
               |
```

Slide 7 to match the *discovered substring*!



j $|pat|$

pat: NONPARTIPULAR

txt: ----------PAR----------

i

dt: txt$[i]$ pat$[j+1]$ . . . pat$[|pat|]$

P A R

`dt:` $\quad$ `txt`$[i]$ `pat`$[j+1]$ $\ldots$ `pat`$[|pat|]$

dt can be computed given *txt*$[i]$ and index j in *pat*!

There are only $|\alpha| \times |pat|$ combinations, where $|\alpha|$ is the alphabet size.

The Skip Distance – Delta

Given *pat*, the skip can be pre-computed for every combination of character read, c, and pattern index, j, by finding how far we must slide to find the *last* occurrence of *dt* in *pat*.

```
pat: NONPARTIPULAR
txt: ----------PAR----------
               |
pat:        NONPARTIPULAR
txt: ----------PAR----------
               |
pat: BC-ABC-BBC-CBC
txt: -----------BBC----------
                |
pat:     BC-ABC-BBC-CBC
txt: -----------BBC----------
                |
pat: BC-ABC-BBC-CBC
txt: -----------ABC----------
                |
pat:         BC-ABC-BBC-CBC
txt: -----------ABC----------
                |
pat: BC-ABC-BBC-CBC
txt: -----------DBC----------
                |
pat:             BC-ABC-BBC-CBC
txt: -----------DBC----------
                |
pat: EE-ABC-BBC-CBC
txt: -----------DBC----------
                |
pat:               EE-ABC-BBC-CBC
txt: -----------DBC----------
                |
```

The Delta Array

`delta`$[c,j]$ is an array of size $|\alpha| \times |pat|$ that gives the skip distance when a mismatch occurs after comparing c from *txt* to *pat*$[j]$.

The Algorithm

```
fast($pat$, $txt$)

If $pat$ = ""
    then
    If $txt$ = ""
        then return Not-Found;
        else return 0; end;
    end;
```

```
preprocess $pat$ to produce $delta$;

$j := |pat| - 1$;
$i := j$;
```

```
while $(0 \leq j \wedge i < |txt|)$
 do
 If $pat[j] = txt[i]$
     then
     $i := i - 1$;
     $j := j - 1$;
     else
     $i := i + delta[txt[i], j]$;
     $j := |pat| - 1$;
     end;
```

```
If  $(j < 0)$
    then  return  $i + 1$;
    else  return  $\mathit{Not\text{-}Found}$;  end;

end;
```

Performance

How does the algorithm perform?

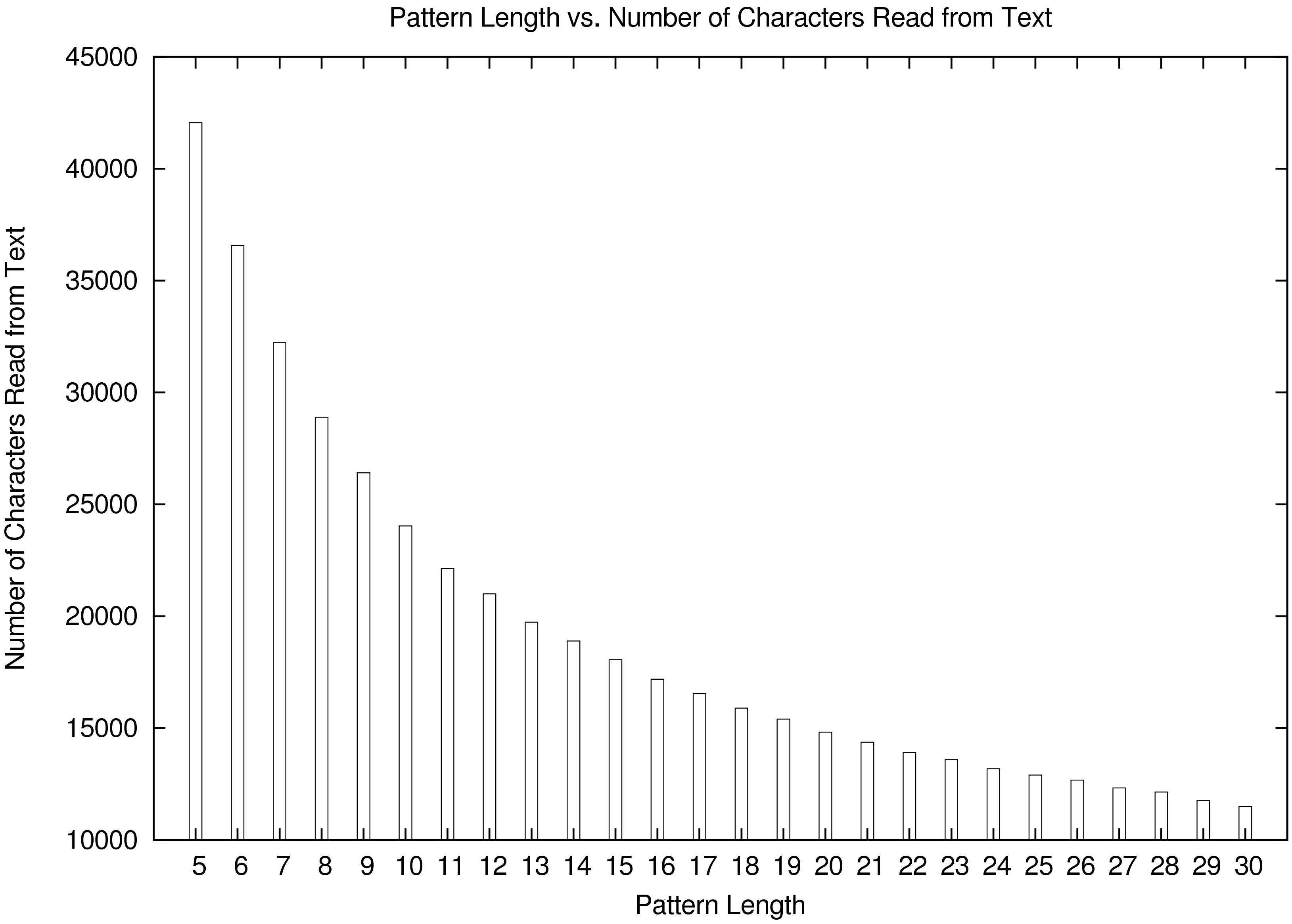
This depends on the size of the alphabet. We only have data on English text right now.

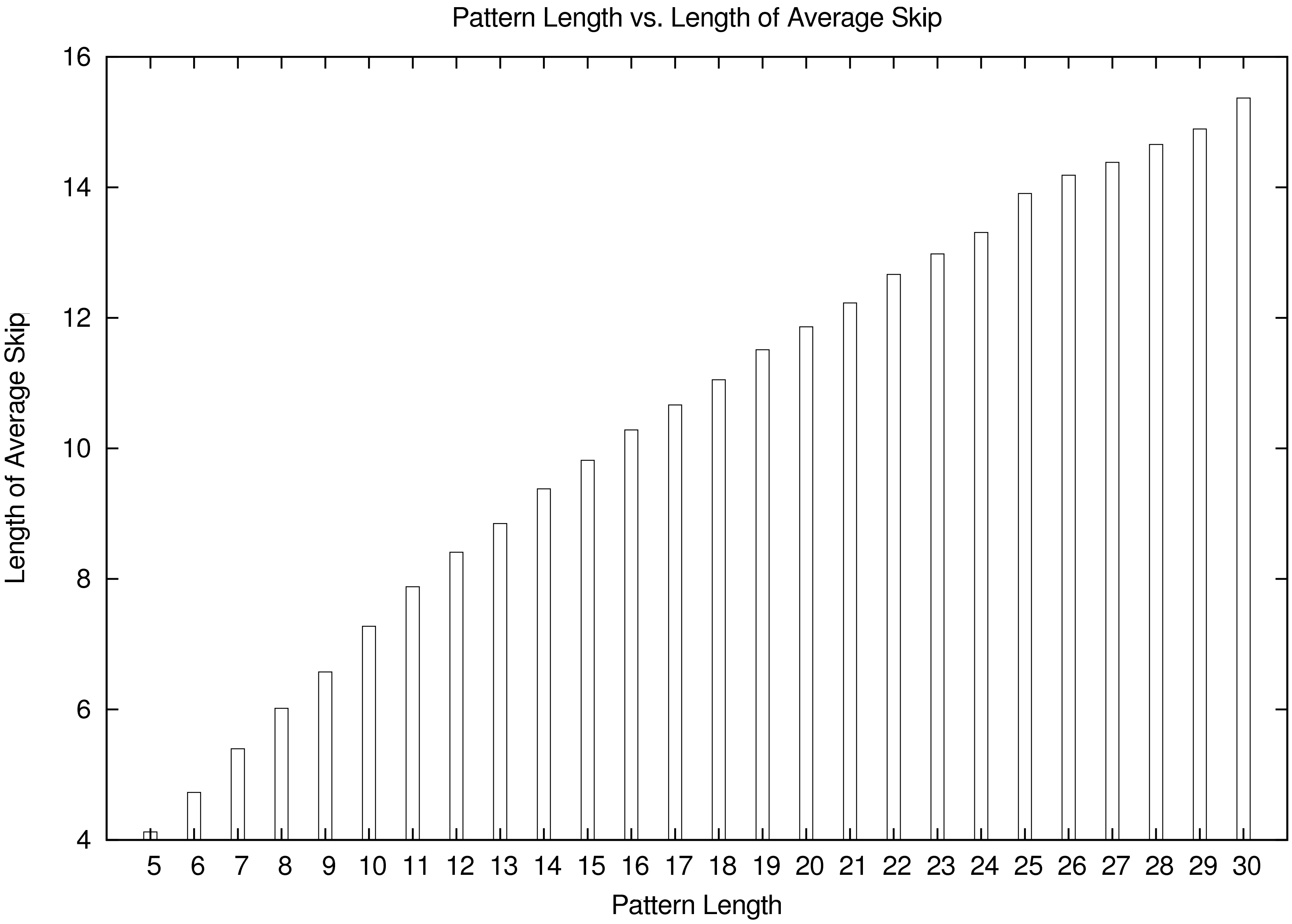
In our test:

`txt`: English text of length 177,985.

`pat`: 100 randomly chosen patterns of length 5 – 30, chosen from another English text and filtered so they do not occur in the search text.

The naive string searching algorithm would look at all 177,985 characters of the search text. In fact, it would look at some characters more than once.


Pattern Length vs. Number of Characters Read from Text
Number of Characters Read from Text
45000
40000
35000
30000
25000
20000
15000
10000
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Pattern Length


Pattern Length vs. Length of Average Skip
Length of Average Skip
16
14
12
10
8
6
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Pattern Length