Prof. G. Hahn
FEV 2009
CPT 2105

By Jeff Prosise

High-Performance String Searching

How a smart algorithm can make a big difference.

Why are some programs faster than others? Why, for example, would one word processor require 30 seconds to spell-check a document when another may take only 15 seconds?

Sometimes the speed difference lies in the efficiency of the code. If one programmer can do a job in 100 clock cycles, another might be able to do it in 95. But very often speed isn't attributable so much to minute differences in code as it is to the problem-solving technique that the code embodies. I'm talking about *algorithms*—the step-by-step procedures that tell computers how to do the things they don't do naturally, such as finding the cube root of a number or sorting a list of names. An algorithm describes an approach to solving a problem. The choice of algorithm can have a dramatic impact on a program's performance.

I've been fascinated by algorithms ever since I caught the programming bug more than a decade ago. My first encounter with algorithms came when I wrote a line-drawing routine that relied on sines and cosines to compute pixel positions. I was astounded to find that the code I had so carefully crafted took 10 seconds to draw a line connecting two corners of the screen. This experience led me to learn about Bresenham's line-drawing algorithm, which converts lines into pixels using simple integer arithmetic. This allows it to run many times faster than algorithms that rely on classic geometrical equations.

Among the traditional types of algorithms discussed in computer science textbooks are methods for string searching. The object of a string search is to find occurrences of a specified text string within a larger body of text—for example, to identify occurrences of the string "pain" in the phrase "The rain in Spain." Windows 95's Find utility uses a string search when you type text into the box labeled "Containing text" on the Advanced tab. The Find command in Microsoft Word's Edit menu performs a string search, too. But not all string-search algorithms are created equal. Some are substantially faster than others, and one in particular has become to string searching what Bresenham is to line drawing. The algorithm is called Boyer-Moore, and it's simple enough for anyone to understand. Let's look at it more closely to see how it works. Along the way, you'll see what a difference an algorithm can make.

BRUTE-FORCE STRING SEARCHING

Suppose someone asked you to devise an algorithm for searching a buffer full of text for a specified string. The simplest approach to string searching is the brute-force method, which involves comparing each and every substring in the text that's being searched against the text that's being searched for. Figure 1 shows the steps involved in a brute-force search. Initially, the leftmost characters in the two strings are aligned, so that the "p" in "pain" lines up with the "T" in "The." Then the two characters are compared. If the characters are not the same, the search string "pain" and the arrow specifying the current location in the text that's being searched are shifted right one character position, and the process is repeated. If the characters *are* the same, the arrow is shifted right one position, but the search string is not. If four consecutive characters compare identically, we've found a substring that equals the search text. Otherwise, the search proceeds until the right end of the search string lines up with the right end of the string that's being searched, and a final comparison fails.

> You'd be surprised at the number of applications whose Search commands use the brute-force technique instead of the faster Boyer-Moore method.

You'd be surprised at the number of application programs whose Search or Find commands use the brute-force technique depicted in Figure 1. If the text being searched contains no more than a few thousand characters, brute force is fine, because users aren't likely to complain—and probably won't even notice—if a search requires 0.8 seconds instead of 0.4. But what happens if the target of the search contains millions of characters, or perhaps billions? There's a big difference between waiting 20 minutes and waiting just 10. And that's exactly the kind of difference that Boyer-Moore can make.

THE BOYER-MOORE ALGORITHM

Brute force is the simplest approach to string searching, but it's hardly the best. Given a buffer M whose length is m and a search string N whose length is n, a brute-force search through all of M for occurrences of N requires at least m-n+1 individual comparisons. Greater efficiency may be obtained by minimizing the number of comparisons performed.

In 1977, R. S. Boyer and J. S. Moore published a paper entitled "A Fast String Searching Algorithm" that described a more efficient approach to string searching. The method outlined in the paper came to be known as the Boyer-Moore string-search algorithm and has since been the subject of countless papers. The Boyer-Moore technique applies a bit of intelligence—some would say common sense—to the search process that reduces the number of comparisons required. In many cases, a Boyer-Moore search can outperform a brute-force search by a factor of 2:1.

The key to the Boyer-Moore algorithm is the observation that some comparisons are unnecessary, because they can't possibly yield a match between the two strings. Consider the comparison between the strings "pain" and "The rain in Spain," for example. Comparing the rightmost character in "pain" to the corresponding character in "The rain in Spain" (a space) makes it clear that the strings don't match:

```
pain
The rain in Spain
   ^
```

A brute-force string-comparison method would shift one place to the right and try again, as shown here:

```
 pain
The rain in Spain
   ^
```


(1)

But in reality, this comparison is unnecessary. The fact that there's no space character in "pain" means the strings can't possibly match as long as "pain" is lined up anywhere over the space between "The" and "rain." Thus, after comparing "n" against " " in the first step and seeing that they are different, we could save time by shifting a full four places to the right, as shown here:

```
    pain
The rain in Spain
     ^
```

This simple modification allows us to do in one step what formerly required four. That's the essence of Boyer-Moore: working from the right end of the character strings instead of the left (notice that the arrow starts out lined up with the "n" in "pain" rather than the "p"), and, after a mismatch occurs, shifting right as many as *n* places instead of just one. (Once again, *n* equals the number of characters in the search string.) When two characters are the same, the arrow is moved one character to the left and the previous two characters are compared. If the number of successful comparisons equals the length of the search text, then a match has been identified.

The trick to implementing the Boyer-Moore algorithm is in knowing how many places to shift after a failed comparison. If the character in the text that you're searching doesn't appear in the search string, it's easy: just shift right a full *n* places. But what if the "n" in "pain" lined up over the "a" in "Spain"? In that case, shifting right more than two places could result in a missed match:

```
          pain
The rain in Spain
            ^
             pain
The rain in Spain
               ^
```

```
      pain
      The rain in Spain
 1      ^                  No match
.....................................
       pain
      The rain in Spain
 2      ^                  No match
.....................................
        pain
      The rain in Spain
 3      ^                  No match
.....................................
         pain
      The rain in Spain
 4       ^                 No match
.....................................
            .
            .
            .

                 pain
      The rain in Spain
13                ^         No match
.....................................
                 pain
      The rain in Spain
14                ^         One character matches
.....................................
                 pain
      The rain in Spain
15                ^         Two characters match
.....................................
                 pain
      The rain in Spain
16                ^         Three characters match
.....................................
                 pain
      The rain in Spain
17                ^         Got it!
```

FIGURE 1: *Brute-force string searching involves comparing each and every substring in the text that's being searched against the text that's being searched for. While effective, it's hardly efficient. The search shown here requires 17 separate comparisons to find the string "pain" in the phrase "The rain in Spain."*

The solution? If the character in the target text appears in the string being searched for, shift right a number of places that equals the distance from the right end of the search string to the character in question. Since "a" is two characters to the left of the rightmost character in "pain," proper procedure would be to shift two places instead of four:

```
          pain
The rain in Spain
            ^
             pain
The rain in Spain
               ^
```

Now the "n" in "pain" lines up with the "n" in "Spain," and the matching substring will be detected.

In practice, it's quite easy for a search program to know how many places to shift for any character in the target text. The secret is to build a table that contains an element for every possible character. For conventional 8-bit character sets, the table contains 256 elements, requiring just 256 bytes of storage. Each entry in the table holds a shift count for the corresponding character. For characters that don't appear in the search text, the shift count equals the number of characters in the search text. For characters that do appear in the search text, the shift count equals the distance from the right end of the search text to the rightmost occurrence of the character. If the search text is "pain," the shift counts for "p," "a," "i," and "n" are 3, 2, 1, and 0, respectively. For all other characters, the shift count is 4. Given a character, a simple table lookup (something that computers can do very fast) then tells you how many places to shift.

Figure 2 shows how Boyer-Moore improves upon the brute-force technique that is diagrammed in Figure 1. In step 1, the "n" in "pain" is compared with the space between "The" and "rain." Because the characters don't match, and because the shift count for the space character is 4, the search text is moved four places to the right, and the arrow is moved with it. In steps 2 through 4, the arrow is moved backward as three successive characters match, and in step 5 the search text is shifted right four places, because of the mismatch between "p" and "r." Note that when a shift occurs, the arrow *always* moves to the end of the search text. In the end, the number of steps required to perform the search is reduced from 17 to 11—a savings of more than 35 percent.

There are many variations on the basic Boyer-Moore algorithm that lend added efficiency to the search process. Even the original Boyer-Moore algorithm is somewhat more complicated than the one discussed here, because it uses an additional heuristic that takes advantage of repeating patterns in the search string. Nonetheless, Figure 2 illustrates the gist of Boyer-Moore string searching in all of its various forms.

THE FASTSEARCH PROGRAM

In order to see for myself just how much of a difference Boyer-Moore can make, I took an old DOS text-searching utility I wrote some years ago and modified it to use a Boyer-Moore string search. The original version used a brute-force algorithm similar to the one shown in Figure 1.

The results were telling. In an informal test performed by searching a 40MB document file for a text string that was 11 characters long, the original version of the program needed 65 seconds to search the entire file. The Boyer-Moore version, which I named FastSearch, took only 41 seconds. In some cases, FastSearch required as little as 33 seconds to scan the entire file for longer strings. (One of the interesting char-

(2)

```
Shift Table for "pain"
Characters                    Shift Count
==========                    ===========
   "p"                        3
   "a"                        2
   "i"                        1
   "n"                        0
All others                    4

      pain
1     The rain in Spain       No match; shift right 4 places
        ^                     (shift count for " " = 4)

       pain
2      The rain in Spain      One character matches;
         ^                    compare the previous character

       pain
3      The rain in Spain      Two characters match;
        ^                     compare the previous character

       pain
4      The rain in Spain      Three characters match;
       ^                      compare the previous character

       pain
5      The rain in Spain      No match; shift right 4 places
       ^                      (shift count for "r" = 4)

          pain
6      The rain in Spain      No match; shift right 4 places
           ^                 (shift count for " " = 4)

             pain
7      The rain in Spain      No match; shift right 1 place
              ^                (shift count for "i" = 1)

              pain
8      The rain in Spain      One character matches;
               ^             compare the previous character

              pain
9      The rain in Spain      Two characters match;
              ^               compare the previous character

             pain
10     The rain in Spain      Three characters match;
             ^                compare the previous character

            pain
11     The rain in Spain      Got it!
            ^
```

FIGURE 2: *The Boyer-Moore string-search algorithm speeds string searches by skipping unnecessary comparisons. When a character in the search string and a character in the text being searched do not match, the arrow is moved up to n places to the right, where n equals the length of the search text. The exact shift count is retrieved from a table built from characters in the search string.*

acteristics of the Boyer-Moore algorithm is that longer search strings will often result in speedier searches.) By comparison, the Find utility that comes with Windows 95 required 34 seconds to perform the same test search. I don't know what search algorithm that Find utility uses, but I'd be willing to bet that it's not a brute-force technique, and that it's probably some form of Boyer-Moore. Find is also written with 32-bit code that's optimized for 32-bit processors, while FastSearch uses generic 16-bit *x*86 code that runs on all Intel CPUs.

In computer programming, there is often a trade-off between memory requirements and performance. Speedier algorithms require more logic, and more logic means more code and more memory. Yet Boyer-Moore added only 100 bytes to the size of the FastSearch executable. It says a lot about the algorithm when a 30 percent or greater increase in performance can be achieved with just 100 additional bytes of code.

In case you'd like to try out FastSearch for yourself, you can download it from PC Magazine Online (see the sidebar "Guide to Our Utilities" in this issue's Utilities column for downloading instructions). You can get syntactical help for FastSearch by typing

```
fs /?
```

at the command prompt. The command

```
fs \docfiles\*.doc "Hello, world"
```

scans all the .DOC files in the \DOCFILES directory and prints the names of those that contain the text string "Hello, world." FastSearch also supports the following command line switches:

- /S for searching subdirectories, too;
- /C for performing case-sensitive searches (by default, searches are not case-sensitive); and
- /A for searching hidden and system files as well as normal files (by default, files marked with hidden and system attributes are not included in the search).

Thus, the command

```
fs \ "Microsoft" /s /c /a
```

searches every file on your hard disk for the word *Microsoft*. (Try it—you may be surprised!) Note that FastSearch may be interrupted by sharing violations if called upon to search an entire hard disk while Windows is running. Just press the F key in response to the "Abort, Retry, Fail" message and FastSearch will skip the offending file and resume the search.

FURTHER READING

Boyer-Moore is one of a larger family of algorithms designed for performing exact string searches, approximate string searches, "sounds-like" string searches, and other types of textual comparisons. You can read more about it in almost any book on classic computer algorithms. One of my favorite such books is *Practical Algorithms for Programmers*, by Andrew Binstock and John Rex (1995, Addison-Wesley). In it, you'll find discussions of all sorts of algorithms (including Boyer-Moore), plus generous amounts of sample code showing how the algorithms are implemented—a key ingredient that's missing from many books of this genre. □

Jeff Prosise is a contributing editor of PC Magazine. *His new book,* Programming Windows 95 with MFC, *was recently published by Microsoft Press.*

3