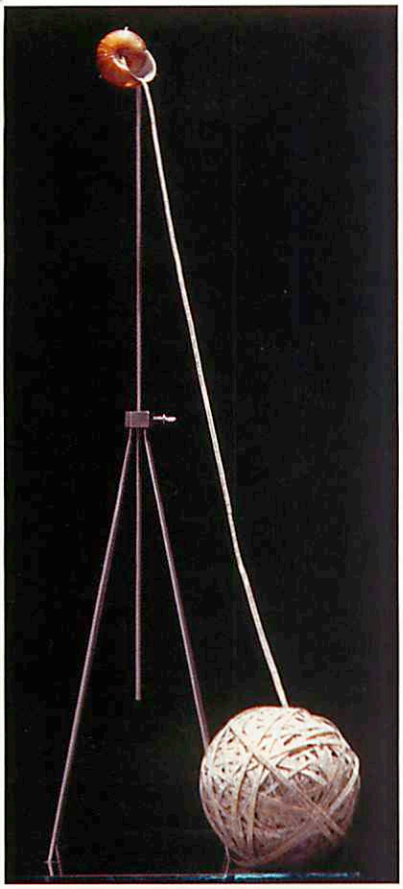


Catherine Widgery, Word Ball with Sea Shell, *1996*

Eureka!

Forget the needle in the haystack. Modern searchers often don't know what they're looking for until they have found it

BY BRIAN HAYES

EVERYBODY'S LOOKING FOR SOMEthing—the Holy Grail, the Promised Land, the Northwest Passage, the source of the Nile, the great white whale, *temps perdu,* Shangri-la, the end of the rainbow. The detective searches for clues, the shopper for bargains, the plumber for leaks, the programmer for bugs. On the back page of the local newspaper, M's and F's of various description are in search of one another, in various permutations. Much of science is framed as a search: for new particles, new planets, new theorems. As for me, I spend a lot of time searching for my keys and eyeglasses.

Searching, it seems, is just one of those things we do; it's a part of human culture, if not human nature. We're the species that walks, talks and can't remember where the car is parked at the airport. How intriguing, then, that after so many centuries of unaided groping and rummaging we finally have some power tools for searching. The Internet has brought us "search engines." Machines for finding things—what a concept! A search engine may not help you retrieve that earring lost in the sofa cushions, but when it comes to anything represented as bits and bytes, searching will never be the same. When the next generation of epic poets writes of the great quest, the heroes will surely begin their adventures by consulting the oracle of AltaVista or Google.

I FIND IT MILDLY CURIOUS THAT TECHnological aids to searching have come along only in the modern era. For other intellectual labors—measuring, counting, reckoning time—people have relied on mechanical crutches from the outset. But searchers have mostly been on their own, with no instruments comparable to the ruler, the abacus or the clock. The sole exception—the one ancient invention that unquestionably had a profound effect on searching—was the alphabet.

It's not the letterforms themselves that aid the search but, rather, the idea of assigning the letters a canonical sequence. Strictly speaking, alphabetizing is a device to aid sorting, not searching; it's a way of putting things in order, from first to last. But sorting is a crucial adjunct to searching. Dictionaries and telephone directories would not be of much use without the alphabetic principle.

Suppose you're trying to find a specific entry in a long list of words or numbers. If the list is unordered, the best you can do is scan from top to bottom. On average you'll have to look at half the entries before you come to the target, and if you're unlucky, all of them. By contrast, a list that is sorted offers a much quicker method. Instead of plodding through, item by item, you start at the midpoint and check to see whether the item you are looking for is above or below. Discard the half that doesn't include the target and divide the remainder of the list in half again. Continuing by halves, you soon zero in on the target. The number of comparisons needed is proportional to the logarithm of the list length—a number much smaller than the length itself. For a list of a million words, the direct and unordered search could take a million steps; the logarithmic method no more than twenty.

The logarithmic advantage is so dramatic that for most of human history searching a large archive was simply not feasible without some kind of sorted index or concordance. But times have changed. That "find" or "search" command in the menu bar will zip through megabytes of unsorted text so fast that you can't measure the time between clicking the mouse and seeing the result. This newfound facility for searching without sorting raises the question of whether the *abc* sequence, which has been the hallmark of literacy for millennia—the first academic accomplishment of almost every child—can endure much longer as a universal cultural artifact. The fact is, you needn't know anything about the *sequence* of letters in order to read and write. That sequence is useful only when you look up a name in a phone book or a word in a dictionary—skills that may rarely be exercised in the coming computopia.

So will we keep drumming the alphabet into the minds of preschoolers? I expect we will, at least for a few more generations. Children who are compelled to master addition and subtraction in case they are ever marooned on a desert island and have to make change for a dollar after the batteries in their calculators run down—those children are not going to escape learning their *abc*'s.

AS FAR AS I KNOW, THE FIRST MECHanized devices that could sort and search were the punch-card machines invented in the 1880s. The cards with the famous warning not to fold, spindle or mutilate were designed by Herman Hollerith, who founded the little company that grew up to be IBM. The Hollerith card represented information by the presence or absence of holes punched in an array of columns and rows.

I have a sentimental fondness for anoth-

er style of punch card, which had holes only near the perimeter. To record data you converted specific holes into notches by snipping away the bit of paper that separated the hole from the edge of the card. I can remember experimenting with such a scheme in my nerdy boyhood, using three-by-five cards filched from my mother's recipe box. Of course I didn't invent the idea. Hollerith himself considered an edge-notched design before settling on his column-and-row format. Moreover, I have recently learned—through the wonders of search engines—that several companies manufactured edge-notched cards from the 1920s into the 1970s. One brand, called the McBee card, was popular in libraries for keeping track of books and borrowers.

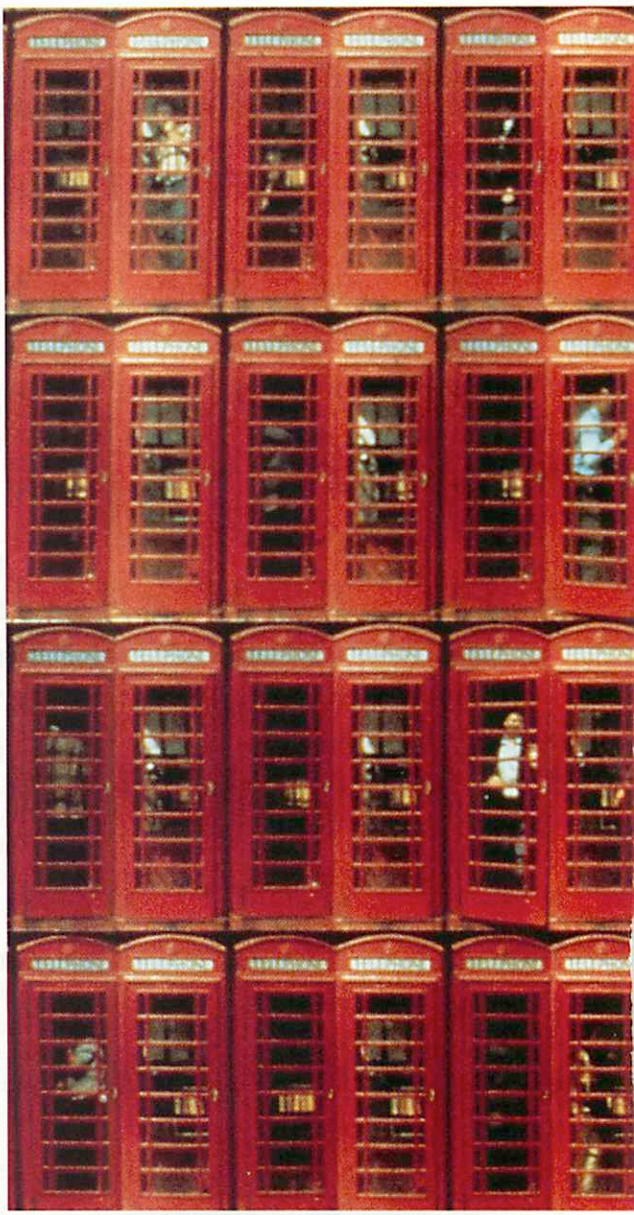
WHAT IS MOST CHARMING ABOUT edge-notched cards is that they require none of the motorized machinery of the Hollerith system. The only equipment needed for sorting and searching is a knitting needle. Consider a deck of thirty-two cards numbered from 0 through 31. Along one edge, each card has five holes, which are assigned the values 1, 2, 4, 8 and 16. Notches are cut at the holes whose values add up to the card number. For example, card five has notches at holes 1 and 4, whereas card thirteen is notched at 1, 4 and 8.

Now suppose you want to retrieve card thirteen from an unsorted stack. You begin by inserting the needle into hole 1 and letting the cards with a notch at that position fall away; card thirteen should be among the dropped cards, so you gather them up for further processing, putting the other cards aside. Now, with the reduced deck, move the needle to hole 2, but this time keep the cards that remain on the skewer and set aside the ones that drop off. At holes 4 and 8 you select the cards that fall, and finally at hole 16 you retain the un-notched card that stays on the needle—there should be only one, and it should be card thirteen.

Finding one McBee card out of thirty-two takes five skewering operations. It's no coincidence that 5 is the logarithm to the base 2 of 32. (In other words, $2^5 = 32$.) As that relation suggests, the card-search process is a logarithmic one. Admittedly, with only thirty-two cards all this rigmarole hardly seems worth the bother, but the system begins to pay off with larger decks. Logarithms grow slowly. To find one card in a thousand would take just ten operations; one in a million, just twenty. That performance is impressive by any standard.

And there's more. If you're adroit enough to knit with several needles at once, it's possible to search an edge-notched deck in a single step. To make the trick work, each card has to have a second set of holes and notches complementary to the first set; that is, wherever a hole appears in the first set, the second set has a notch, and vice versa. With such a double-punched deck, you merely insert needles at the right positions, give a good shake, and the unique selected card falls out. The performance is even better than logarithmic; it is a "constant-time" algorithm, which runs just as fast no matter how large the deck is.

If edge-notched cards are such hot stuff, why haven't they replaced all those clunky old computers on our desks? Well, for one thing, it would take a knitting needle 800 feet long to handle a million cards. Then there's the job of punching all those notches. And, finally, the simple search described above—picking one number out of a finite sequence of numbers—is probably not the best benchmark for evaluating the merits of cardboard versus silicon.



THE CLASSIC TESTING ground for search algorithms is a procedure known as exact string matching. In computer science a string is a sequence of characters—letters, numbers, spaces, punctuation—with no higher-level structures such as paragraphs or pages. Think of a string as a novel written on ticker tape. The search task is to find all the occurrences of one string (called the *pattern*) inside another string (the *text*). Usually the pattern string ("great white whale") is much smaller than the text (*Moby Dick*).

A moment's thought suggests an algorithm. Write the pattern and the text on separate strips of paper, and line them up on their leftmost characters. Now, working from left to right, compare each character in the pattern with the corresponding character in the text; if they are all identical, note that you have found a match. Now slide the pattern one place to the right and compare all the characters again. Continue until the rightmost character in the pattern slides off the end of the text. For a pattern of p characters and a text of t characters, the number of comparisons needed is approximately $p \times t$.

Another moment's thought suggests an easy speedup: you don't always have to compare all the characters. Since a match has to be exact, you can stop comparing and slide the pattern along to its next position as soon as you come to the first mismatch. To what extent does this shortcut improve performance? If you happen to be searching for the pattern *xxx* in the text *xxxxxxxxxx*, the shortcut will never be taken, so it's not helpful at all. But that's worst-case behavior. More typically, the shortcut can reduce the number of comparisons from $p \times t$ to something closer to $p + t$.

James Crable, Telephone Boxes, London, England, *1986*

MAKING FURTHER IMPROVEMENTS requires more than a moment's thought. Indeed, the quest to improve exact string search algorithms occupied an entire community of computer scientists, off and on, throughout the 1960s and 1970s. The techniques that came out of that effort are ingenious, but simple and obvious they are not.

And yet the basic principles are not so mysterious. Suppose you are searching for the pattern *algorithm* in the text *algebraic algorithms*. Starting from the left, the first three characters are confirmed to match, but at the fourth position you find an *o* opposite an *e*. At this point the naive algorithm would slide the pattern one place to the right and start the character-by-character comparison again. Certainly, though, a human searcher could see at a glance that all those comparisons are a waste of time, and that no alignment of the first four characters can possibly work. Although computers are not as good at glancing, with a little preliminary work—by building a table of the character positions within the pattern—the computer will recognize that it can safely ratchet the pattern along to the fifth position.

That idea underlies a search procedure worked out in the 1970s by James H. Morris of Carnegie Mellon University in Pittsburgh, Pennsylvania, and Donald E. Knuth and Vaughan R. Pratt, both of Stanford University. The Knuth-Morris-Pratt algorithm never examines any character in the text more than once, so it is guaranteed to require no more than $p + t$ comparisons.

Another technique can often do even better. Again, line up the pattern at the start of the text:

```
algorithm
algebraic algorithms
```

As in the earlier algorithms, the pattern moves from left to right along the text. But in this algorithm, you read the pattern backward, beginning with the *last* letter, as you compare it with the text. That change of direction might seem insignificant—after all, the letters are the same no matter which way you read them—but, in fact, swimming "upstream" has an important advantage. In this example, the advantage is apparent as soon as you find the first mismatch, between *m* in the pattern and *c* in the text. Because the letter *c* does not appear *anywhere* in the pattern, you can safely shift the pattern forward by its entire length, skipping nine positions to the right in the text. Through such leapfrog tactics, the algorithm searches a text without ever looking at most of its characters. Although the worst-case performance is still about $p + t$ comparisons, typical results are much better.

The upstream algorithm was invented around 1975 by Robert S. Boyer and J Strother Moore, both of the University of Texas at Austin, and independently by R. William Gosper. It is widely admired for its sheer cleverness and efficiency, but writing a computer program that puts the algorithm into action is not as easy as I have made it seem. Note that my chosen pattern word, *algorithm*, has no repeated letters. Figuring out how far ahead you can safely skip is trickier when you're searching for *banana* or *Mississippi*.

AFTER ALMOST TWENTY-FIVE YEARS, the Boyer-Moore algorithm remains the fastest known method for many kinds of searches, and yet it is not quite the last word. One major drawback is that it works only for *exact* string matches. In many a quest, you don't really know what you're looking for, though you hope you'll recognize it when you see it. Such approximate pattern matching turns out to be a much harder problem than exact searching.

It's possible to leave some slack in the search process by including "wild cards" in the pattern—a practice known variously as globbing or grepping, depending on how it's done. Under glob rules, typing in *fo** will bring up *fo, foo, fog, fork* and much else; according to grep syntax, the matches for *fo** would be *f, fo, foo, fooo, foooo* and so on. The two processes look much the same, differing only enough to ensure the occasional disastrous mistake.

In recent years the strongest impetus for the development of approximate search algorithms has come from molecular biologists. The decoding of genomes has given rise to a corpus of text that runs to several billion characters, all written in a four-letter alphabet that ribosomes read fluently but that people find opaque. Approximate matching is the only way to find anything interesting in this archive. For example, *Homo sapiens* has thousands of genes in common with the fruit fly *Drosophila melanogaster*, but because of subtle variations none of those shared genes would be recognized as the same by an exact string search.

Thus genome searches attempt to measure the *similarity* between sequences, usually in terms of an "edit distance": the minimum number of changes needed to convert one string into the other. Three kinds of editorial change must be taken into account: substitutions, insertions and deletions. Of the three, substitutions are easy enough to cope with because they are strictly local, affecting only one letter at a time. Insertions and deletions complicate matters because they alter the alignment of the string over a large region. Yet no adequate genetic search can be

Continued on Page 46

Continued from Page 11

conducted without considering them; as genes evolve, bits of DNA are constantly being snipped out or spliced in.

Approximate genome searching is computationally expensive. Given two strings of length n, finding the edit distance between them takes roughly n^2 steps—which makes the process far slower than checking for an exact match. The job turns out to be so laborious that it simply cannot be done routinely when screening a new DNA sequence against a large genome database. Instead, the screening programs do an *approximate* approximate string match: they estimate the edit distance without actually finding the optimum sequence of editing operations.

THE INTERNET WOULD APPEAR TO BE the ideal application for search algorithms—the perfect test case for methods that have continued to be refined over the years. Ironically, though, when searching the Net one encounters difficulties that make classic search algorithms ill-suited to the task. Part of the problem is the monstrous size of the Internet, which would overwhelm any algorithm that attempted to search sequentially, character by character. Another complication is that people searching the Net are seldom trying to retrieve a specific document. Instead, their searches are a form of exploration—an attempt to find "what's out there" on some topic.

The need for Internet search tools had already been recognized a decade ago, before the World-Wide Web made the Internet a mass medium. In that quaint, pre-Web era, which now seems as remote as the age of Morse-code telegraphy, people searched the Net with Archie or WAIS, the Wide-Area Information Service, and what resulted from the search was an unadorned list of file names. When the Web was born, the first search aids were manually compiled directories, the ancestors of such services as Yahoo! and the volunteer Open Directory Project. Such directories have the important virtue of organizing the content of the Web by subject, providing a taxonomy of knowledge that is similar to a library's subject catalogue. Unfortunately, though, maintaining the hierarchical structure requires so much effort that such handcrafted directories take in only a small fraction of the total Web.

Search engines, which emerged in the early 1990s, have broader coverage; some claim to index more than a thousand million pages. But because of that vast scale, the search process bears little resemblance to what happens when you click the "find" command in a word-processing program. Instead, search engines rely on a three-phase process. The first phase is carried out long before you log on and type in your search term. A program called the crawler, or spider, surveys the Web, systematically visiting pages and retrieving text and other content. Crawling the Web takes weeks or months, which is why a search sometimes sends you to pages that no longer exist.

The crawler brings back gigabytes or terabytes of data—too much for any sequential search algorithm to handle. Thus to make fast searches possible, the collection of retrieved documents has to be organized in some way, and that is the second phase of the engine's operation. In principle, the text could be sorted alphabetically to enable a logarithmic search, but there are better schemes. Most search engines rely on a method called the vector-space model. At the heart of the model is a gigantic matrix with a column for every document and a row for every anticipated search term. Thus the matrix could have a million columns and 100,000 rows. The entries in the matrix record how many times each term appears in each document.

OFTEN YOU DON'T *know what you're looking for, though you hope you'll recognize it when you see it.*

The giant matrix is put to use in the third and final phase of a search engine's cycle—the response to your query. The process is straightforward: for each word in the query, the engine goes to the appropriate row of the matrix and chooses all columns that have a non-zero entry. The process takes only milliseconds—and it's a good thing, too. Large search engines get millions of queries a day, which works out to some hundreds a second; they've got to pump out answers at the same rate or the input hopper will overflow.

AS IT HAPPENS, FINDING DOCUments that match a given query is the easiest part of the search engine's job. The real challenge is figuring out the best way to dump the majority of those matches. With at least a thousand million Web pages out there, queries routinely yield 10,000 or 100,000 hits. The search engine must somehow divine which of those documents best meet the user's needs and list them first.

The enormous matrix built into the vector-space model is the key to the winnowing process. The matrix can be given a geometric interpretation. The rows (corresponding to the potential search terms) define the dimensions of an imaginary space, and the entries in each column (representing a document) give the coordinates of a point in that space. For a small matrix—say, three rows and five columns—it's easy to see how this plan works. The rows correspond to length, width and height dimensions, and the columns define five points that lie somewhere inside this volume. Adding another 99,997 spatial dimensions makes the model harder to visualize, but mathematically it works the same way.

How can such a bristling, multidimensional abstraction help you find your way around the Web? It's remarkably straightforward. The search engine simply treats the query as if it were another Web document brought back by the crawler. It tallies all the terms in the query and plots the position of the corresponding point in the vector-space model. The best answers then become the documents that lie nearest the query point.

THE ALGORITHMS THAT RANK THE hits are becoming more sophisticated. The innovation that I find most interesting was pioneered by the Google search engine, which was developed by Sergey Brin and Lawrence Page when they were still doctoral students at Stanford University. In its rankings, Google considers not only the content of a page but also the number of other pages that have links to it. Presumably, people who create such links think the page is worth visiting, so an abundance of links can be taken as a strong recommendation.

Not only do Web search engines have to find and rank pages, but they also have to cope with pages designed to manipulate or deceive; for example, some Web designers rig the system by adding invisible extraneous keywords. Such tactics would be a bizarre aberration in other contexts—you don't expect the books in a library catalogue to disguise their content and vie for the notice of readers—but on the Web, the searcher and the searchee are often adversaries. Moreover, the search engines themselves can turn traitor: some

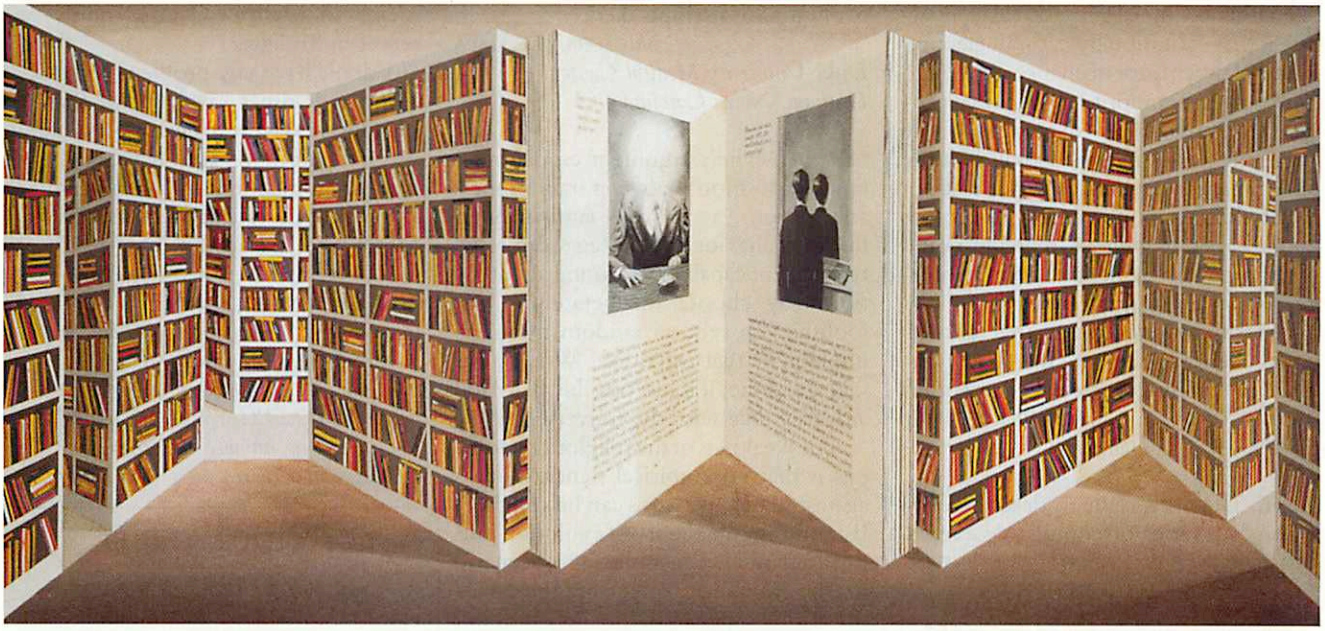
accept payment from the owners of Web pages for preferential placement.

NEW TECHNOLOGIES THAT TRANSform daily life tend to dazzle and annoy us in equal measure. My cellular telephone is a wonderful convenience when I'm caught in traffic, but yours is an abomination when it rings in the theater. The automated teller machine is a marvel of the age when it hands me money in a foreign city, but not when it refuses my card at the ATM around the corner. Search engines seem to fall into the same category, innovations we love to hate. They are windows into a world of riches, except when the windows are broken or closed or dirty or distorted.

Indeed, searching for the term "windows" is all it takes to illustrate the frustrations of Web searching. Google returns "about 20,700,000" documents; needless to say, I didn't look at all of them, but the first few dozen had nothing to do with panes of glass—and quite a lot to do with a certain software manufacturer.

For now, though, I remain far more enchanted than frustrated. Access to those thousand million documents is priceless, even if the path to them is sometimes indirect. Consulting Google has become my first reflex for answering every question. What was the first Internet search engine? Who said, "*Cherchez la femme*"? What is the Echelon program? And it's not just the answers that delight, if you ever find them. The very process of searching has charms of its own, and more often than not the best outcome is finding what you weren't looking for. Isn't that always the moral of the quest genre—that the search for meaning ends by finding meaning in the search itself?

Some weeks ago, curious about those edge-notched cards that fascinated me in childhood, I launched a Web search. Soon I was led to the Dead Media Project, which turned out to be a celebration of bygone information technologies. Among the dead media was the Indecks Information Retrieval System, a set of edge-notched cards that Stewart Brand reviewed in the *Last Whole Earth Catalog* in 1971. There was also a reference to McBee cards, which was how I first learned of their existence.



Patrick Hughes, Book Look, *1997*

Searching for "indecks" turned up dozens of Web pages, but they were a mixed and mysterious lot. One bore the title "And once devoured, through eternal night one lollipop glows steady like a . . ." Not much help there. It took a while to deduce the unifying thread: nearly all of the hits were pages in which the file name "index.html" had been misspelled "indecks.html."

Searching for "McBee card" took me first to the crash of a B-25 bomber in 1943 (I still don't understand why). More useful was a work of fiction by Charles Brownson titled "The Altair Lizard," in which a character explains the edge-notched mechanism, speculates on the fate of the McBee Company and complains of difficulty in ordering supplies. Following further leads, I discovered a dozen scattered copies of a nerd-humor posting called "The Legend of Mel, the Real Programmer." Mel worked for the Royal-McBee Corporation, in the period after the punch-card company merged with the Royal Typewriter Company. I learned that the Royal-McBee LGP-30 was the computer on which Edward Lorenz discovered deterministic chaos. I even found the Web site of the company in its current incarnation, McBee Systems, Inc., which is now a purveyor of office supplies.

THIS LONG CHAIN OF LINKS WAS leading me in fascinating directions—but not where I wanted to go. And, despite several hours of diligent and diverting searches, I never did find anything more about McBee cards on the Web. Instead, I found what I was seeking in the library, where I searched by running my finger slowly along the spines of books until I came to a fifty-year-old volume by Howard F. McGaw titled *Marginal Punched Cards in College and Research Libraries*. Thumbing through its pages, I learned much of what I wanted to know about McBee cards—for instance, that the first patents were issued in 1896, and that the main users were libraries and chemists. So here was a case in which gigabytes and terabytes and 100,000-dimensional vector spaces were outdone by the Dewey decimal system. Evidently, there are still places the Net can't take us.

But don't gloat, Luddites. The next person who asks a Web search engine to find McBee cards will very likely be referred to *this* document. •

***BRIAN HAYES** is a freelance writer and a former editor of* AMERICAN SCIENTIST.