With the advent of computer-aided word processing, the question remains: what is the best way to find a specific piece of text for which you are looking? For my word processing software, I decided to implement a method for searching just published this year by Robert S. Boyer and J Strother Moore. The use of the Boyer-Moore approach to string searching allows my word processor to find what you’re looking for quickly, and with more natural search strings.

This searching technique only concerns itself with letters that are in the string for which you are looking1 and, unlike other searching techniques, it starts looking for its match at the *end* of the search string2. These two attributes combine to allow Boyer-Moore to skip sections of text without ever needing to check them! When this mode of searching finds a letter that is not in the phrase for which you are looking, it can skip ahead by the length of that phrase3. Other searching methods start at the beginning of the string and would need to process nearly every character in the entire text at least once4. Also, Boyer-Moore works particularly well when dealing with texts of natural human language5. Most users will be searching for words or phrases that only contain a handful of the letters in the alphabet6, increasing the likelihood that the search will be able to jump ahead.

Another strength is that, because it starts looking at the end of the string, Boyer-Moore allows for more natural search patterns. Looking for a phrase that begins with “the” with other search methods would slow down search times7 simply because of how often the word “the” is present (in English of course). You won’t need to “computerize” your language to find what you’re looking for.

The Boyer-Moore method of searching is a great fit for my word processor. It returns search results quicker than other methods8 and allows for the use of natural language. It also uses less memory that the competing Knuth-Morris-Pratt method9, saving your precious RAM.

Justifications:

Let N = the length of the text, M = the length of the search string, and R = the size of the alphabet.

1. Boyer-Moore uses an array of size R (the alphabet consisting of all possible characters that may be searched for). It then preprocesses the search string to determine the position of the right most occurrence of each character in the search string and stores those results in the array, using -1 to represent characters that are not in the search string. This array is used to tell the algorithm how many positions to the right to move when a mismatch occurs.
2. Boyer-Moore starts at position M-1 of the search string and then moves to M-2 when a match is found, and so on.
3. When a mismatching character is found, Boyer-Moore determines how far to skip ahead based on the values in the array built above. Should a mismatch occur on the first character, this value would be (M-1) + (-1) = M. It will skip ahead by M characters.
4. In the worst case, a brute force approach’s outer loop would operate N-M times. The KMP algorithm’s loop has a cap of N iterations.
5. Human languages, as opposed to languages like binary or that used for genetic code sequencing, have comparatively large alphabets. This increases the likelihood of a mismatch on the first comparison. Likelihood of a mismatch is roughly (R-1) / R. With binary, a mismatch would only occur on the first comparison ~50% of the time (1 / 2 = .5). With English, considering just letters and spaces, a mismatch would occur nearly 96.3% of the time (26 / 27 = .96296). While not realistic to give every character equal weight, this demonstrates the appeal of large alphabets.
6. Typical search patterns for typical users are unlikely to contain every possible character of the alphabet. Even with a relatively small alphabet, say English letters, numbers and punctuation, a search string of length 10 could only possibly contain a fraction of the possible characters in the alphabet.
7. Take the search string “the birdhouse” for example. By starting at the beginning of the search string, every time the string “the ” appears in the text, both brute force and KMP would need to make at least 5 comparisons to determine a mismatch has occurred – one for each of the letters ‘t’, ‘h’, ‘e’, the space, and the first character of the subsequent word. Boyer-Moore would only mimic this when the string “ouse” was found, much less likely than “the ”.
8. Analysis of realistic runtimes of brute force, KMP, and Boyer-Moore searching show that both brute force and KMP require time proportional to ~1.1N. Boyer-Moore requires time proportional to N/M. This result is a factor of how often Boyer-Moore is able to skip ahead M characters by mismatching the character at M-1 with a character not contained in the search string. With a search string of size 10 and a search field of 1,000,000 characters, Boyer-Moore would produce a result after 1,000,000 / 10 = 100,000 processor cycles. KMP or brute force would be expected to take 1.1 x 1,000,000 = 1,100,000 processor cycles. With a new Apple II with a 1MHz processor, it would take Boyer-Moore 1/10th of a second, instead of over a second for the other two methods.
9. KMP requires an array of size RxM to build its deterministic finite state automaton. Boyer-Moore requires an array of size R, requiring less space by a factor of M.