DEAR COMPUTER, IS THE LETTER-STRING A WORD?

MICHAEL KEITH
Salem, Oregon

In "Is That Letter-String Really a Word?" in the May 1999 Word Ways, Ross Eckler considers the question of determining the plausibility that a given letter string is an English word.

Since I do a lot of word explorations using a computer, this led me to ponder a related, but slightly different question: what is the quickest way for a computer program to determine if a given letter string is a word? In this case, plausibility isn't good enough: no false positives or false negatives are allowed. We need to be able to determine unequivocally if a given letter string is present in a (possibly very large) word list.

FIRST STEPS

The first method one might employ is a naive, brute-force search. This simply compares the input letter string to every word in the dictionary. One dictionary word list I frequently use has about 100,000 words, so this would involve 100,000 comparisons. It might seem that each comparison would involve checking all the letters in the two strings being compared, but since we check each letter one at a time, usually only one letter (the first one) has to be compared. Most of the time the first letters will differ, and so we can immediately conclude that the letter string doesn't match the word in question. Thus, on the average, this method requires a little more than 100,000 comparisons.

We can do much better by grouping all the N-letter words in our word list together. When we are given a letter string, we know (or can easily determine) how many letters it has, and then we need only compare it against the N-letter words in the word list. If the length of the input string is assumed to be random, then on the average this reduces the number of comparisons to about 6,000.

We can carry this idea a step further. Take each length-N word list and divide it into 26x26 = 676 sublists, each one containing all those words that start with a certain bigram. We now need to augment our big word list with two tables: one that says, for each bigram, how many words there are in that sublist, and one that gives the location of each sublist within the big list. These two tables require a total of 26x26x4 (because each entry is four bytes)x2(for two tables) = 5408 bytes. We need tables for each word length (say, from 1 to 15), for a total of about 81K bytes. This is very small and so quite reasonable.

Now, when we are given a letter string, we first check the bigram table for the requisite N. For about two-thirds of the bigrams, the table will say there are no words of length N starting with this bigram, so we can immediately conclude "not a word" with no further work. When we do have to check for a match, only about 32 (instead of 6000, as above) words have to be checked.

GETTING MORE SOPHISTICATED

Although this might seem pretty good, we can do much better. The next step is to eliminate the need for comparing words letter by letter. This can be done by turning each word in the dictionary into a base-27 number, using A=1, B=2, etc., and considering each letter as a digit in place-value notation. For example, ACE becomes the number 27x27x1(for A) + 27x3(for C) + 5(for E) = 815. Actually, we only have to use the last N-2 letters, since the first two are implicitly known due to the partitioning of the dictionary into sections by initial bigram. Suppose our words are 15 letters in length. Then we need a range of 27^(15-2) to represent any word. As luck would have it, 27^13 is slightly less than 2^64, so we can fit the value of any word into a 64-bit binary computer word. Two-letter strings can be compared by simply comparing two 64-bit numbers--an operation well-suited to the way computers work.

Furthermore, if the words in our dictionary are in alphabetical order (as they no doubt will be), our base-27 numerical "words" will be listed in increasing order. Such a list can be searched much faster than by the naive method of scanning from top to bottom: we can use a binary search. We first compare our input number M with the number in the middle of the list. If it is less than or equal to it, we then know that M, if present in the list at all, must be in the first half. (Similarly, if M is greater, we know it must be in the second half.) This procedure is continued by, at each step, comparing M against the number in the middle of whatever range is left, until there is only one candidate left. Then we compare to see if it is exactly equal, which tells us whether M is in the word list.

We said above that about 32 words have to be checked on the average. With a binary search this only requires 5 comparisons instead of the 32 needed by the naive scanning method--a significant reduction.

At this point it might be tempting to stop, but this is precisely where Eckler's article comes into play. Can we use methods similar to the ones he suggests to quickly detect that a letter string is not a word, thus avoiding the whole binary search most of the time? Note that any technique we employ must be very simple or it will not be worthwhile, since the algorithm we have developed thus far is quite fast already.

The answer is yes, we can. The following two enhancements make the overall word-checker significantly faster.

(1) THE SMALL-WORD CHECK

For small sizes we can avoid the binary search altogether by having another table that immediately gives us the answer ("is a word" vs "not a word"). We decided to do this for 1-, 2-, 3- and 4-letter words, since these have 27x27x27x27 = 456,976 possible base-27 values. If the string is four letters or less, we calculate its base-27 value and then check a table of single-bit values (0 for "not a word", 1 for "is a word"). Since each table entry is only one bit, this entire table only takes 456,976/8 = 57122 bytes. The table is calculated once and stored on disk, so it need not be recreated each time.

(2) THE TERMINAL-4-GRAM CHECK

We also precalculate another table, of size 27x27x27x27, which says whether there are any words with five or more letters whose terminal 4-gram has a specific base-27 value. As it turns out, only about 4 per cent of the possible terminal 4-grams appear in our dictionary, so in 96 per cent of all cases we are able to decide immediately that the letter string is not a word.

The key cleverness in this step is its economy. Almost no extra computational effort is required, because calculation of the base-27 value for the last four letters is an intermediate step in calculating the base-27 value for the last N-2 letters (which we have to do in order to begin the binary search). This step requires almost no superfluous calculation.

The complete word-recognition algorithm can be summarized as follows:

1. If the string is 4 letters or less, check the small-word table and return its yes-or-no answer. Steps 2-4 can be skipped.
2. Compute the base-27 value of the initial bigram. If the bigram table says there no words of length N with this bigram, return "not a word" and skip steps 3-4.
3. Compute the base-27 value of the terminal 4-gram. If the 4-gram table says "not possible", return "not a word" and skip step 4.
4. Finish computing the base-27 value of the last N-2 letters (by starting with the base-27 4-gram and adding in the remaining letters). Do the binary search on the portion of the dictionary corresponding to N and the initial bigram. If the final value equals the value of the letter string, return "is a word", else "not a word".

Is this the ultimate? No, it probably could be made even faster, perhaps by enlarging some of the tables or adding further checks. But the technique is very fast yet still quite simple (the computer code fits on a single page), so it seems to strike a good balance between speed and complexity.

SOME RESULTS

I implemented the method described above in the C programming language and ran it on my 350MHz home PC. Although it's a little tricky to quantify, since the run time depends heavily on the statistics of the input letter strings, on the average it takes about 300 nanoseconds (sic!) to determine if a one-to-fifteen-letter string is a word. This corresponds to about 100 CPU clock cycles. At this speed, I can test about three million words a second for validity. Should I be willing to let a program run for four days, it could decide the fate of a trillion strings!

To test and exercise the word-recognition code, I devised a few (hopefully new) logological problems to try it on.

Problem 1: Take a 14-letter word and write its letters in a circle, then count the number of words of three or more letters that occur as consecutive letters in the circle, in either forward or reverse order. Which 4-letter word yields the most words? The answer, for my word list, which only took 2.4 seconds to find, is OPERATIONALIST. The 30 words found in its circle (not counting itself) are: ope, opera, operation, operational, per, era, rat, ratio, ration, rational, rationalist, ion, stop, stope, stoper, top, tope, toper; sil (OED, a kind of ochre), lan (OED, loan), tar, tare, are, rep, lis (OED, fleur-de-lis), list, repot, repots, pot, pots.

Problem 2: Take a 10-letter word, XXXXXXXXXX, and a 15-letter word, YYYYYYYYYYYYYYY, and write them in a square format (left, below):

X X X X X	B R I E F	C H A R M
X X X X X	C A S E S	F U L L Y
Y Y Y Y Y	M I C R O	T R U S T
Y Y Y Y Y	R A D I O	W O R T H
Y Y Y Y Y	M E T E R	I N E S S

Which pair of words maximizes the number of horizontal and vertical 5-letter words in the square? To solve this puzzle, we used our largest word list, which has 36,400 10-letter and 5,900 15-letter words, and exhaustively tried all 214 million combinations, checking the ten five-letter strings each time. The program took about ten minutes to run, and produced a number of solutions containing seven five-letter words. Two of the nicer ones are shown above. The first yields brief, cases, micro, radio, meter, eerie, and raiae (plural of raia, a zoological group); the second yields charm, fully, trust, worth, Huron, myths, and alure (OED, allure). Note that solving this problem using the second-worst search method described above (linear search through all N-letter words) would have required seven days instead of ten minutes.