Class Notes, CS 3137

1 Tries

(Reference: D. Knuth, The Art of Computer Programming, Vol 3., Sorting and Searching, pages 481-505)

- A *Trie* (short for re*Trie*val) is a multiway tree that is used for efficient searching. The idea is much like a thumb index dictionary: if you place your thumb in the cutout of the dictionary for a particular letter, you go to directly to that section of the dictionary, and can then locate all words beginning with that letter.
- Formally, a trie is an M-ary tree whose nodes are M place vectors (arrays) with components corresponding to digits or characters. Each node of the trie on level l represents the set of all keys that begin with a certain sequence of l characters; the node specifies an M-way branch, depending on the $(l+1)$st character.
- We will use the dictionary example to analyze tries. A dictionary trie consists of nodes that have a branching factor of 27 (26 letters in the alphabet plus 1 more which we will explain below).
- The worst case access time for a trie search is simply the number of levels of the tree. In a dictionary example, the worst case number of accesses is equal to the number of letters we are searching for in the word. For example, a 7 letter word may take up 7 trie accesses to find it.
- If an N letter word is unique after K letters, we can then store that word at level K in the trie, thus reducing the overall number of levels in the trie.
- Tries can be implemented in a number of ways. The figure below shows a trie of the 31 most common words in English organized as a forest of trees. The table below it is a trie of the same data organized in tabular form. Each letter of the alphabet is along the left side column of the table, and if we are searching for the word HER we go to the letter H entry in the first column of the trie, which tells us to go to column (5) to find the branch for the second letter in the word we are looking for which is E. Column 5 is also a 27-way branch node, and it tells us that the branch for E is column 11, where the entry for R - the third letter of the word we are looking for - is the node we are looking for. For this 3 letter word, we took 3 branches.
- The extra entry in each node (27th entry for a 26 letter alphabet) is kind of a NULL pointer that is used to signify that the path from the root to that node forms a legal entry in the trie. In the table, the word HE is found by having an entry in the null field of column 11. This is needed since HE is an entry we want to find, but the trie needs to keep indexing words that continue beyond the 2 letters HE such as HER or $HEALTH$ or $HEAT$ etc.

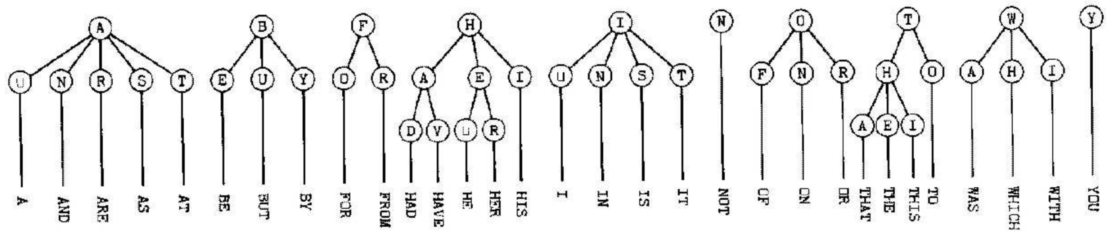


Figure 1: TRIE as a forest of trees for the 31 most common English words

	1	2	3	4	5	6	7	8	9	10	11	12
$\emptyset$	—	A	—	—	—	I	—	—	—	—	HE	—
A	(2)	—	—	—	(10)	—	—	—	WAS	—	—	THAT
B	(3)	—	—	—	—	—	—	—	—	—	—	—
C	—	—	—	—	—	—	—	—	—	—	—	—
D	—	—	—	—	—	—	—	—	—	HAD	—	—
E	—	—	BE	—	(11)	—	—	—	—	—	—	THE
F	(4)	—	—	—	—	—	OF	—	—	—	—	—
G	—	—	—	—	—	—	—	—	—	—	—	—
H	(5)	—	—	—	—	—	—	(12)	WHICH	—	—	—
I	(6)	—	—	—	HIS	—	—	—	WITH	—	—	THIS
J	—	—	—	—	—	—	—	—	—	—	—	—
K	–	—	—	—	—	—	—	—	—	—	—	—
L	—	—	—	—	—	—	—	—	—	—	—	—
M	—	—	—	—	—	—	—	—	—	—	—	—
N	NOT	AND	—	—	—	IN	ON	—	—	—	—	—
O	(7)	—	—	FOR	—	—	—	TO	—	—	—	—
P	—	—	—	—	—	—	—	—	—	—	—	—
Q	—	—	—	—	—	—	—	—	—	—	—	—
R	—	ARE	—	FROM	—	—	OR	—	—	—	HER	—
S	—	AS	—	—	—	IS	—	—	—	—	—	—
T	(8)	AT	—	—	—	IT	—	—	—	—	—	—
U	—	—	BUT	—	—	—	—	—	—	—	—	—
V	—	—	—	—	—	—	—	—	—	HAVE	—	—
W	(9)	—	—	—	—	—	—	—	—	—	—	—
X	—	—	—	—	—	—	—	—	—	—	—	—
Y	YOU	—	BY	—	—	—	—	—	—	—	—	—
Z	—	—	—	—	—	—	—	—	—	—	—	—

- Lets compare an English language dictionary trie with a binary search tree for English words. A fully balanced BST on a dictionary of the roughly 25,000 words found in the unix spelling file /usr/dict/words would need about 15 levels (i.e. $2^{15} = 32K$). But since the tree will probably not be fully balanced, it will require more levels. The trie for this file needs 21 levels, although most words are found at a much higher level. The word **electroencephalography** needs 21 levels to disambiguate it from the words electroencephalogram and electroencephalograph. This points to a possible strategy using root words and leaving prefixes and suffixes off. If we reverse the characters to create the trie the word immunoelectrophoresis needs 15 levels to disambiguate it from electrophoresis (remember we are stripping off characters backwards!).

- The number of accesses in a trie is equal to the length of the key being used. If this is much less than log_2 of the number of entries, then a trie is better than a Binary Search Tree. For example, a trie can access all $26^5 = 11,881,376$ 5 letter words in 5 accesses, where a BST will take about $log_2 26^5$ or about 23.5 comparisons. Keep in mind that we will have to allocate $27^0 + 27^1 + 27^2 + 27^3 + 27^4 = 551,881$ nodes (each containing a 27-vector) for this trie.

- **Example:** Suppose we want to implement an English language Dictionary as a trie. A typical college dictionary contains over 100,000 words. The naive approach is just to start filling up the pointers of the trie for every word. But analysis suggests that the trie will be very sparse - most pointers won't even be needed, since many letter combinations do not exist in English, and we don't have to bother searching or indexing them.

- If we leave out proper names and abbreviatons from our dictionary, we can see that given a particular first letter, there are only a few choices of the second letter. If the first letter is B, then the second letter will NEVER be one of the 17 letters b,c,d,f,g,j,k,m,n,p,q,s,t,v,w,x, or z. Many other letters also don't have these same 17 letters as second letters (i.e. c,d,f,g,h,j,k,l,m,n,p,q,r,t,v,w,x,z with a few exceptions). A way to take advantage of this is to have a trie node that encodes the 9 actual letters that could occur, plus a special extra node that simply points to a linear list of these obscure words. This linear list then can be scanned if needed, but probably won't be accessed very often. Of the $26^2 = 676$ possible combinations of the first 2 letters of a word, only 309 actually are legal words. In addition, 88 of these 309 pairs are combinations that you would probably not recognize as legal word prefixes (...,bd,bh,...,xr,yc,yp,....).

- Indeed, the trie entries may be sparse and unneeded after a certain point. An excellent strategy is to use a trie for an initial pruning of the search tree, and then a simpler and perhaps less eficient search for the final accesses. In a real dictionary, that is what you do: go to the thumb index for the word (trie-like), then a back and forth for the page the word is on (much like a BST comparison: word we want is less than 1st word on page or greater than last word on page), and finally a linear sequential search on the page for the actual word.

- **More examples:** Fig. 2 is a trie for the words: cat car cot cod cop code coder cope. Fig3 is a trie for the words: electro electrocardiogram electrocardiograph electrode electroencephalogram electroencephalograph electroencephalography electrolysis electrolyte electrolytic electron electronic electrophoresis electrophorus. Fig. 4 is a trie for the same words only using the reverse of each word. Note how the number of levels is reduced. So indexing on a different sequence of the keys may be more efficient and produce a better distribution of the data. The moral of this story is: know your data to choose the right data structure.

- **Deletion in Tries:** Deletion in tries is tricky. A simple strategy that works well (and is useful in many other search data structures) is *lazy deletion*. We simply leave the data in the trie, but mark it as no good. It takes up space but we will never access it as good data.

- If we are really going to implement a delete, we need to analyze the different cases in a trie. If it is a wordnode on a trienode that contains other entries we simply delete the wordnode and mark it as open. If the deletion leaves only a single pointer to the nodes below in the trienode, then we can move the lower level up one level (see example to follow).

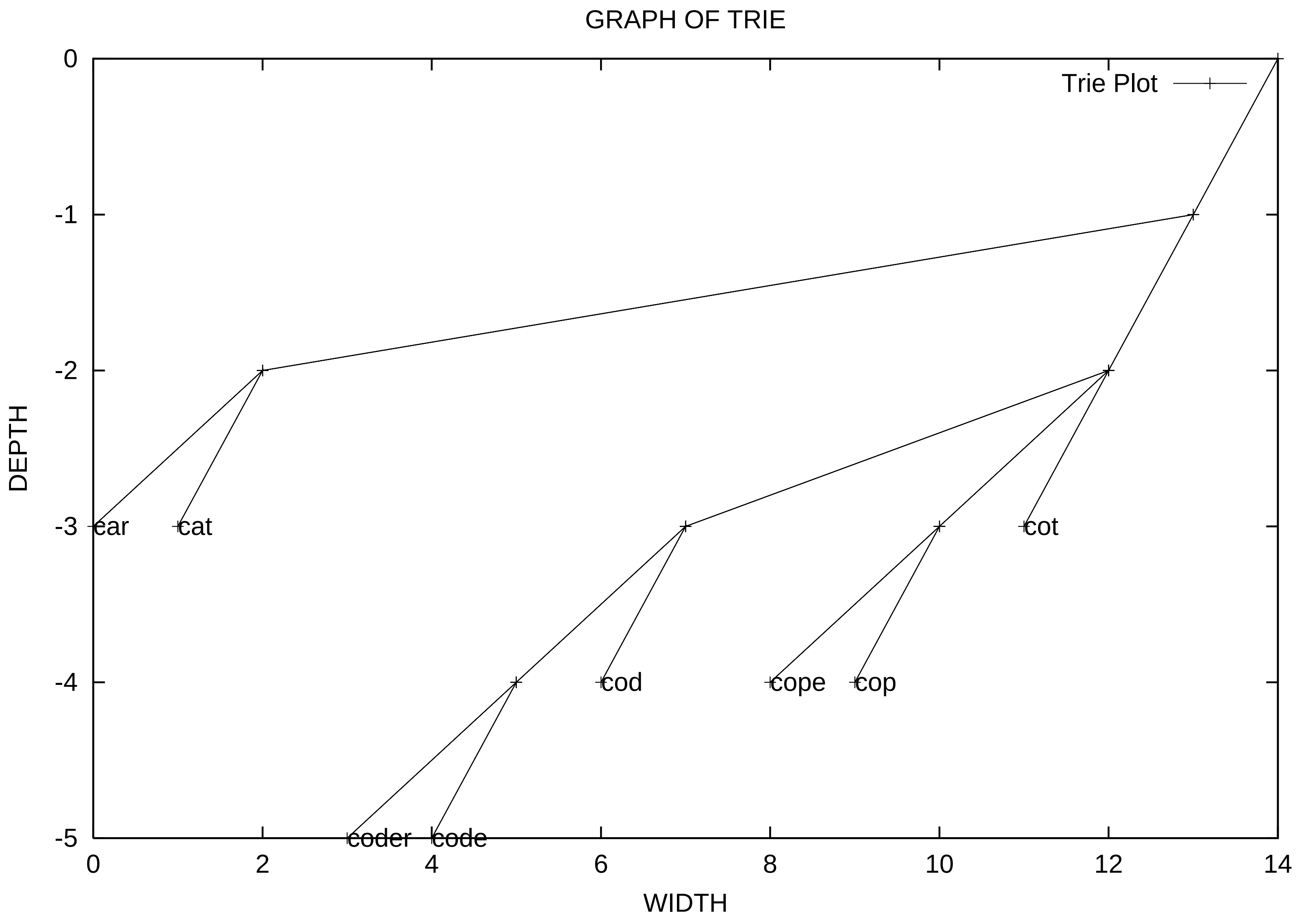


Figure 2: TRIE for the words: cat, car, cot, cod, cop, code, coder, cope

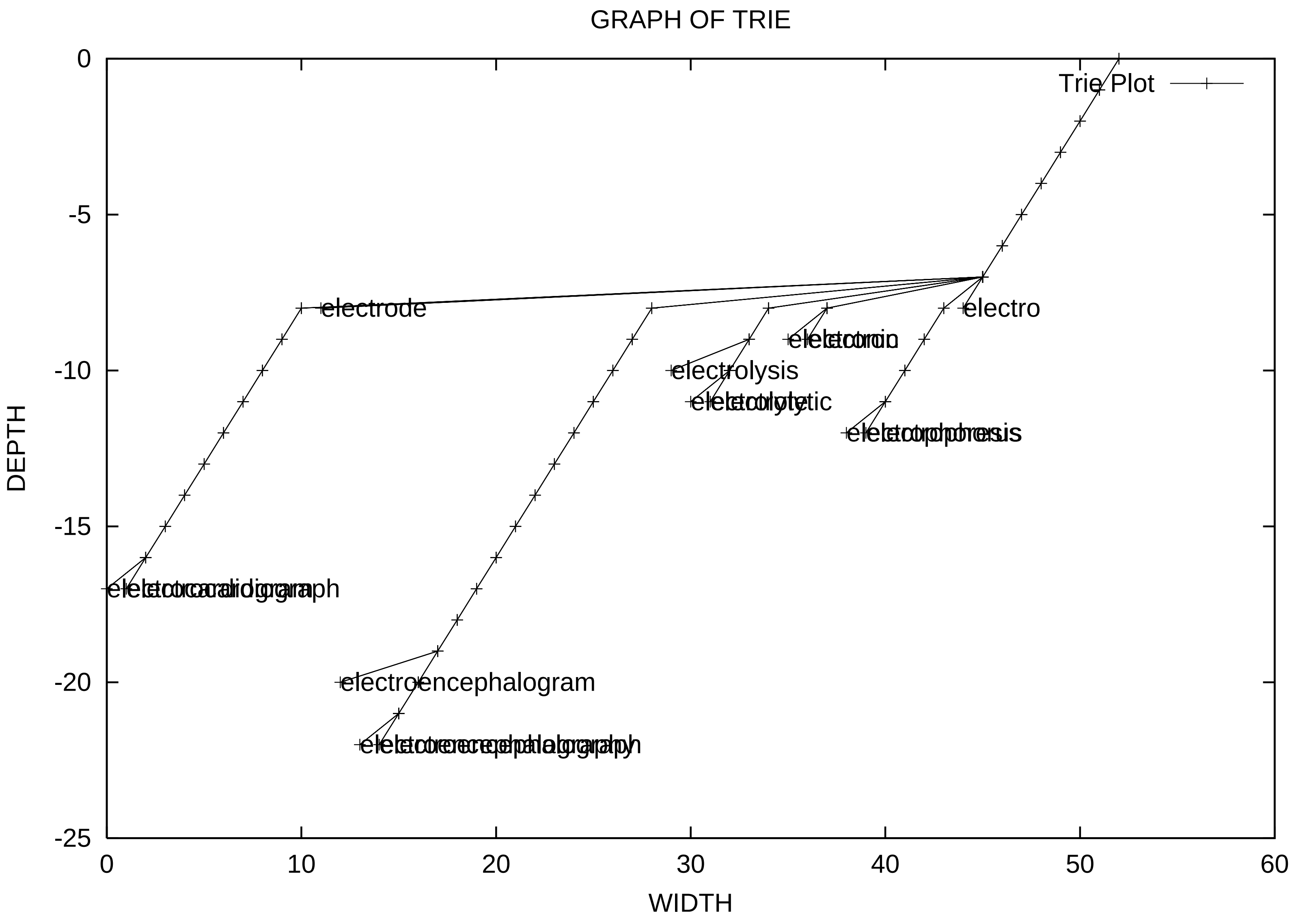


Figure 3: TRIE for the words: electro electrocardiogram electrocardiograph electrode electroencephalogram electroencephalograph electroencephalography electrolysis electrolyte electrolytic electron electronic electrophoresis electrophorus.

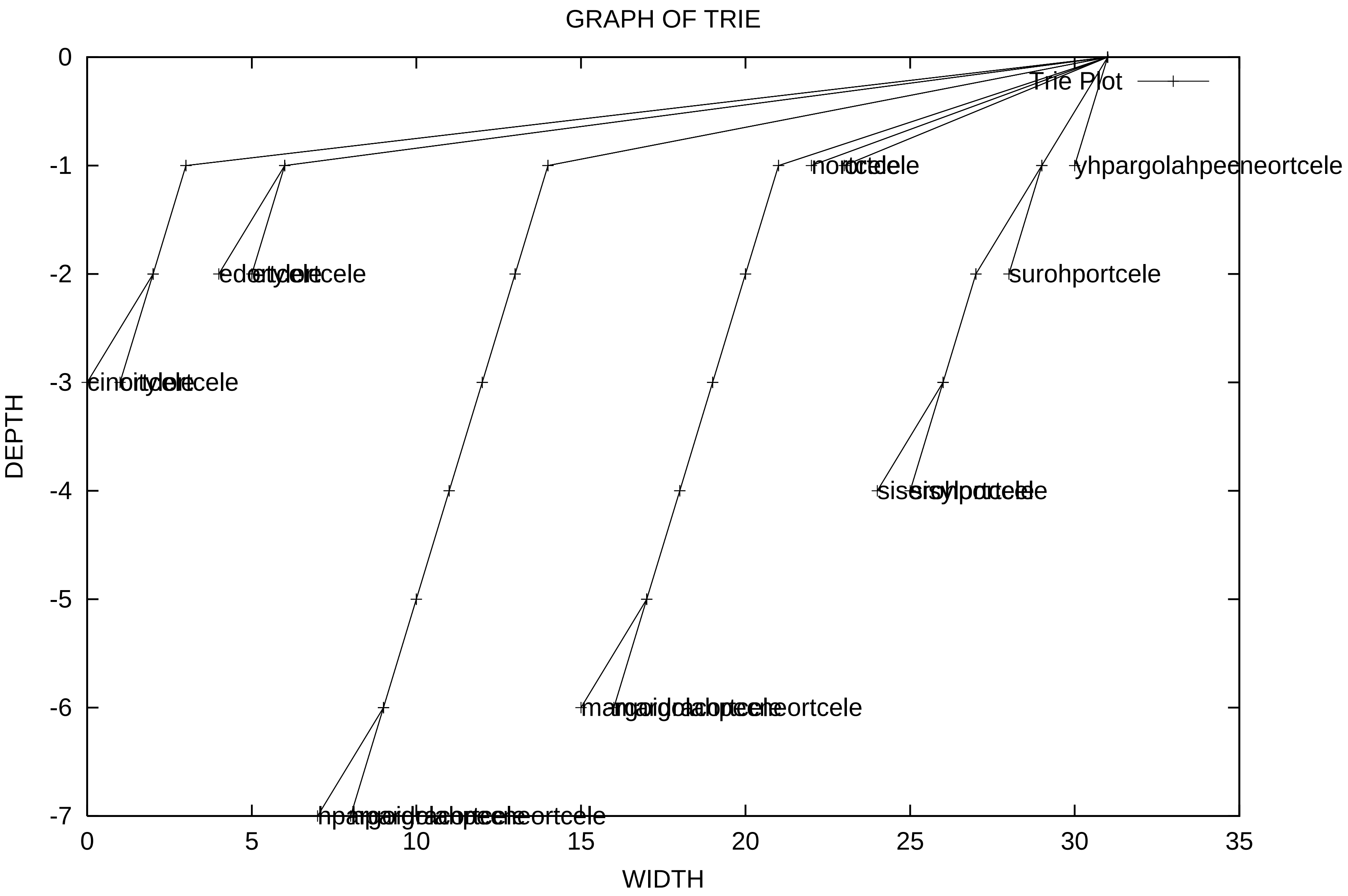
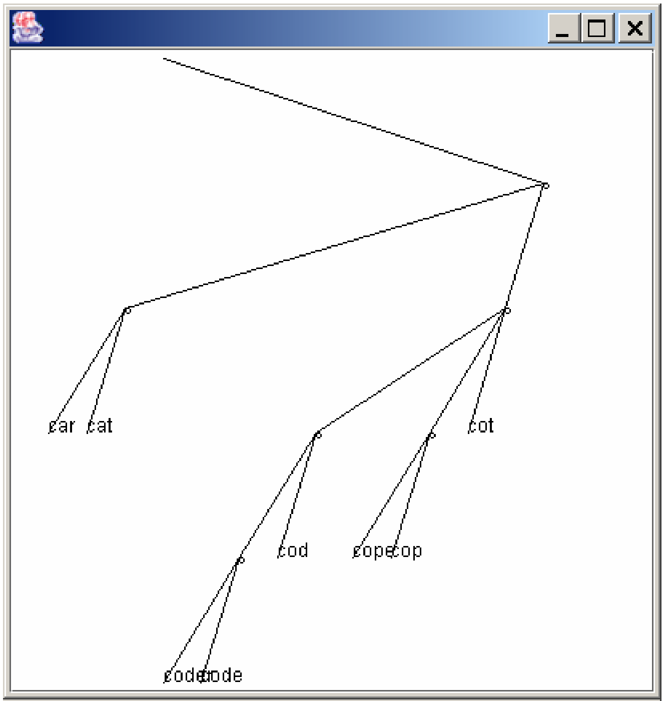


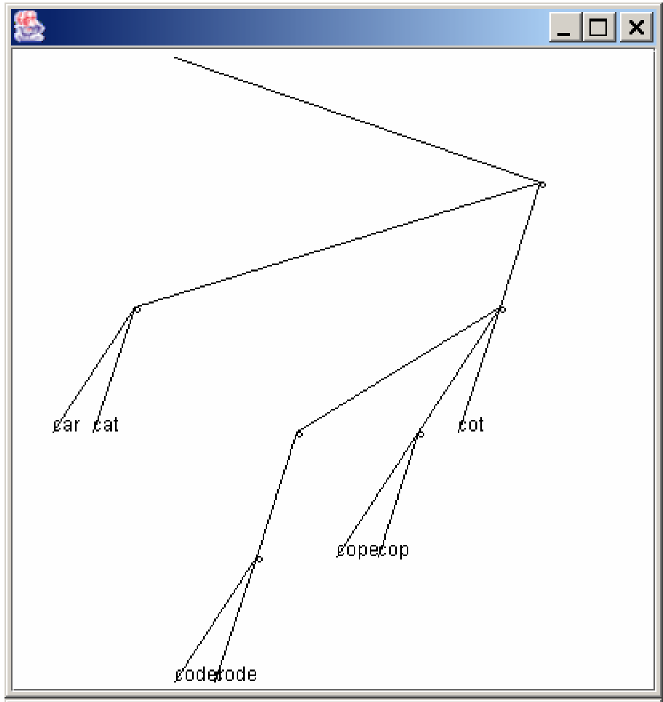
Figure 4: TRIE for the words in figure 3 indexed in reverse order

Deletion In Tries

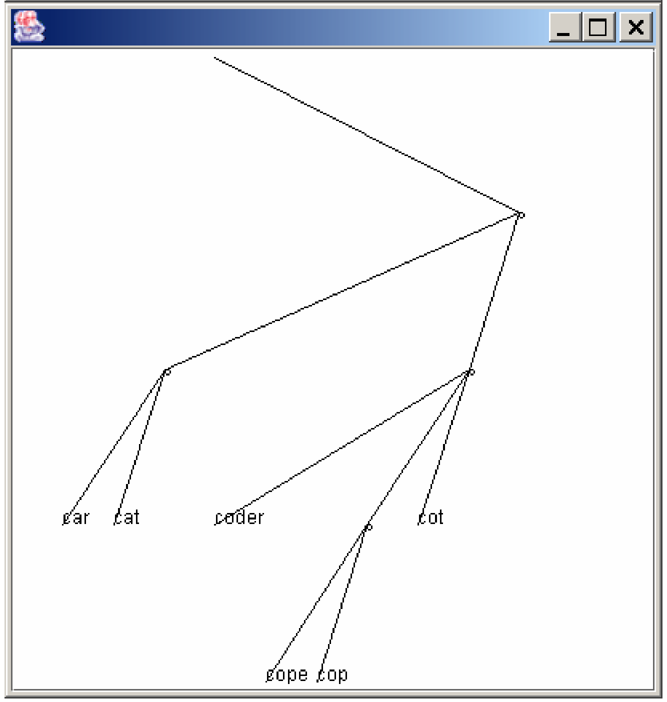
Original Trie -→



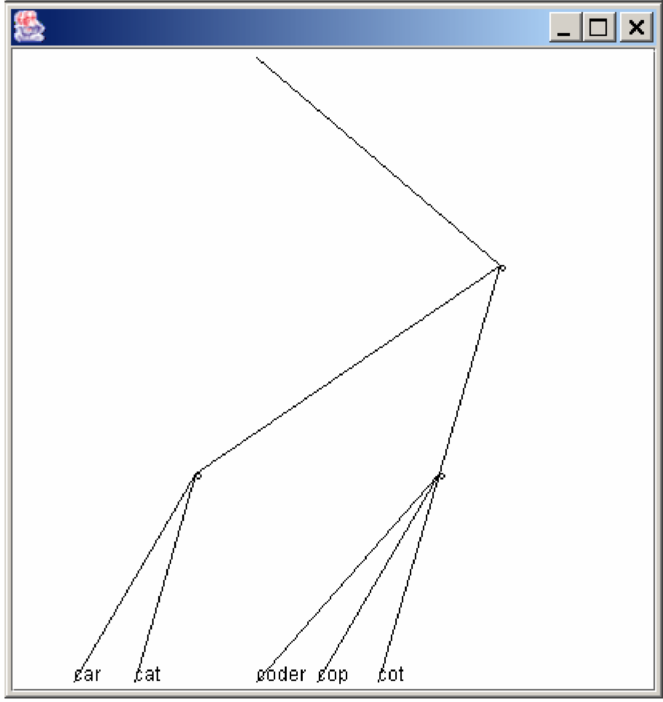
Delete cod -→



Delete code-→



Delete cope-→


car
cat
coder
cop
cot