Department of Computer Engineering

University of Peradeniya

CO 322 Data Structures and Algorithms

Lab 04 - Tree ADT

Registration Number : E/16/086

Name : A.L.V.H.Dharmathilaka

In this lab a Trie data structure is implemented using C language to store a dictionary of English words, and use it to quickly retrieve words for an text auto-complete application. Different number of word sets are used to test the implementation.

wordlist1000.txt : 1000 words

wordlist10000.txt : 10000 words

wordlist70000.txt : 70000 words

Part 1

Design and implement a Trie node structure and associated operations and algorithms to store and retrieve a collection of English words. It has the following functions.

- typedef struct trienode{

  ...

  }TrieNode;

- TrieNode* createNode(...);
- TrieNode* insertWord(...);
- int printSuggestions(...);
- int getIndex(…);
- Main function

Compile the code : gcc part1.c –o part1

Run the code : part1 < txt file name that going to use >

Example : part2 wordlist1000.txt

Disadvantages in trie node structure

- Tries containing few long strings perform worse than BSTs .
- Many nodes have one child.
- Long chains of nodes without any branching

Part 2

Design a Radix Tree (Trie) Structure for Text Auto-complete. This structure reduce the space usage of the Trie structure by removing unnecessary nodes without losing any information because a radix tree is a compressed version of a trie. In a trie, on each edge single letter is written, while in a PATRICIA tree (or radix tree) whole word can be stored.

Part 2 has following functions

- typedef struct trienode{

  ...

  }TrieNode;
- typedef struct linkedStr{

  …

  }linkedStr;
- TrieNode* createNode(...);
- TrieNode* insertWord(...);
- int printSuggestions(...);
- int getIndex(…);
- linkedStr* linkedStrAppend(..);
- linkedStr* createNewString(..);
- Main function

Compile the code : gcc part2.c –o part2

Run the code : part2 < txt file name that going to use >

Example : part2 wordlist1000.txt

Important notes

Only the alphabetical characters are considered and other symbols and digits are neglected. All letters are considered as lower case letters.

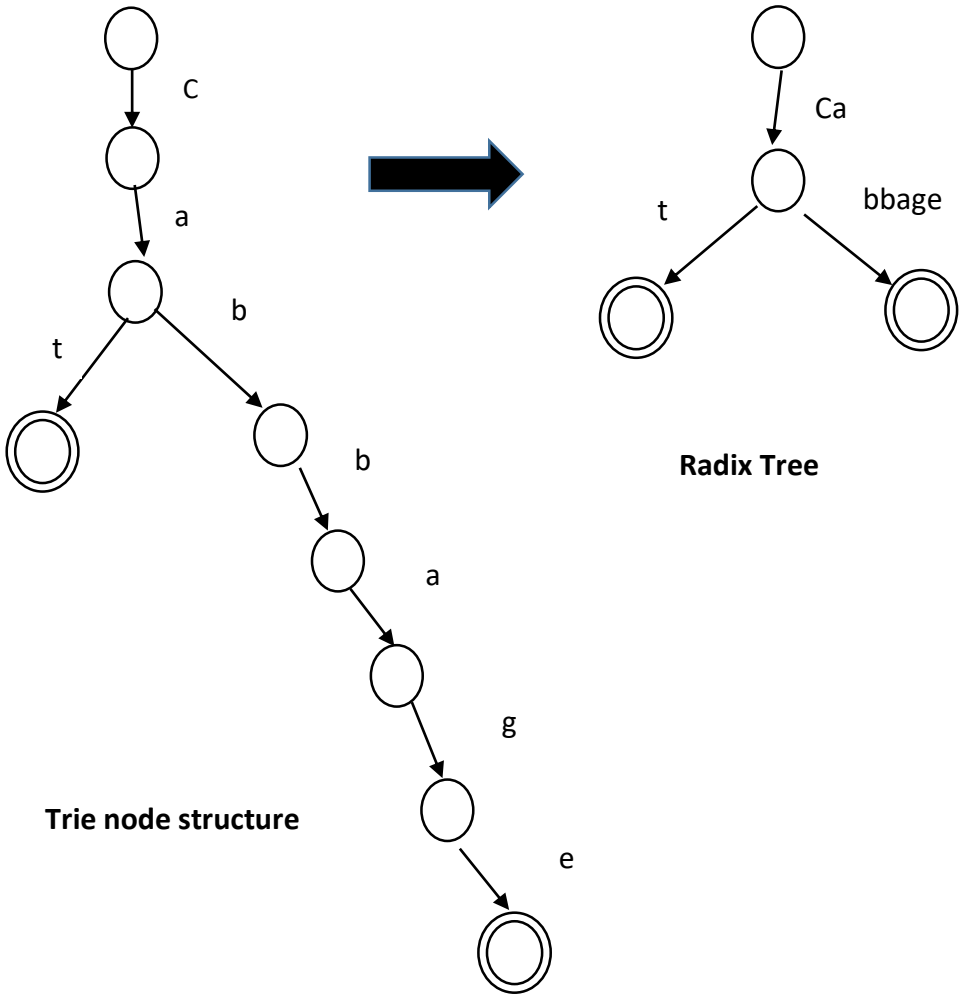
Input text file must be given as command line input.

(a) memory space usage

Table1.0

	Memory space for part1 (bytes)	Memory space for part2 (bytes)
wordlist1000.txt	108000	76944
wordlist10000.txt	1080108	904512
wordlist70000.txt	7549848	5399184

Most prominent idea of the implementing a Radix tree is to reduce the memory space. According to the Table 1.0 it is clear that part2 radix tree structure allocates less memory than part1 trie structure. Unlike Radix tree structure normal trie structure may contain long chains of nodes without any branching. It wastes the storage. In Radix tree remaining characters will be allocated to one node. As example,



Trie node structure

Radix Tree

(b) time taken to store the dictionary

Table 2.0

	Time taken to store the dictionary in part1 (seconds)	Time taken to store the dictionary in part2(seconds)
wordlist1000.txt	0.001000	0.001000
wordlist10000.txt	0.004000	0.004000
wordlist70000.txt	0.039000	0.033000

Insertion works like in a trie, except that sometimes have to split an edge into two

E.g. to insert “cabbie”, we have to split “bbage” into “bb” and “age”:

So there is no significant amout of difference of the time durations that spend to store the data in both tree structures. It is clearly shown in the Table 2.0. The difference in time durations in wordlist70000.txt file can be neglected.

(c) time taken to print a list of suggestions for chosen word prefixes

Search prefix : administration

Table 3.1

	Time taken to search in the dictionary in part1 (seconds)	Time taken to search in the dictionary in part2(seconds)
wordlist1000.txt	0.002000	0.003000
wordlist10000.txt	0.001000	0.005000
wordlist70000.txt	0.002000	0.005000

Search prefix : act

Table 3.2

	Time taken to search in the dictionary in part1 (seconds)	Time taken to search in the dictionary in part2(seconds)
wordlist1000.txt	0.014000	0.015000
wordlist10000.txt	0.068000	0.067000
wordlist70000.txt	0.029500	0.019900

Search prefix : church

Table 3.3

	Time taken to search in the dictionary in part1 (seconds)	Time taken to search in the dictionary in part2(seconds)
wordlist1000.txt	0.005000	0.005000vv
wordlist10000.txt	0.008000	0.005000v
wordlist70000.txt	0.059000	0.044000

To print values , first they have to be found. Finding values in a radix tree works the same as in a trie. In above three tables table 3.1 , table 3.2 , table 3.3 both time durations take to search in the dictionary in part1 and part2 have close to each other. Those values are not exactly same but they are closer enough to say that finding values in a radix tree works the same as in a trie. But time taken to search, is increased with the increase of word size.