E/16/388

Weerasundara W.M.T.M.P.B.

CO-322 Lab 04 - Tree ADT

Report

Assumptions

- When storing the words, words with non alphabetical characters were neglected.
  Ex :- Words like a-horizon ,a-ok was not stored.
- Words are stored in simple letters as in the files.
  Ex :- word like AMERICA will be stored as america
- When search for suggestions for a word, any user input will be converted to lowercase form 1st and then look for suggestions.
  Ex :- suggestions for AM, Am, am are the same.

First, 2 Tree data structures were created to store the words in text files and then for user input string, program will provide suggestions from the stored words.

Part 1 Code – Trie.c

Part 2 Code – Radix.c

Trie data structure was implemented 1st and then it was modified to reduce the space usage by removing unnecessary nodes without losing any information.

Then

(a) memory space usage

(b) time taken to store the dictionary

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

Were compared between 2 data structures.

(a) memory space usage

for wordlist1000.txt

	Trie	Radix Tree
Number of nodes	3026	686
Memory consumption	653616 Bytes	153664 Bytes

for wordlist10000.txt

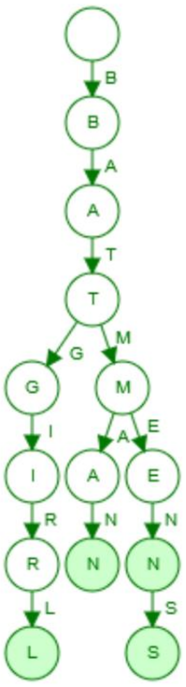
	Trie	Radix Tree
Number of nodes	24179	8076
Memory consumption	5222664 Bytes	1809024 Bytes

for wordlist70000.txt

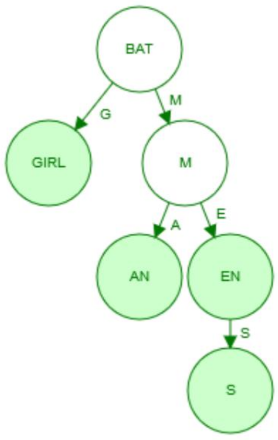
	Trie	Radix Tree
Number of nodes	215916	46817
Memory consumption	46637856 Bytes	10487008 Bytes

We can clearly observe the number of nodes and memory consumption of the radix tree is much lower than the trie data structure. In radix data structure, unnecessary nodes were removed and strings with the same prefix is stored with utilizing common nodes. Therefore redundancy of nodes is removed. Therefore less space is utilized in radix tree structure.

Ex: How words BATMAN, BATMEN, BATGIRL, BATMENS are stored in 2 data structurs



Trie



Radix tree

Therefore by this example we can see the better memory utilization of radix tree over trie data structure. In tries, characters have their own nodes but in radix common prefixes have common nodes and unmatching suffixes have their own nodes. Hence lesser nodes are used.

(b) time taken to store the dictionary

for wordlist1000.txt

	Trie	Radix Trie
Time consumption for store	0.00113500 s	0.00105900 s

for wordlist10000.txt

	Trie	Radix Trie
Time consumption for store	0.00945800 s	0.00863900 s

for wordlist70000.txt

	Trie	Radix Trie
Time consumption for store	0.08206300 s	0.07463900 s

When the dictionary size increases, time to store increases.
Also in here, we can see some what of a better performances by radix tree regarding the storing time. Radix tree takes some less time than trie data structure. The reason is since radix tree is a compressed version of trie structure, it has less number of nodes. So when storing a new entry, entry's insertion will be optimized if there is already stored strings with the same prefix like it. Only the unmatching suffix part will require a new node. Alse if there is no matching prefix, string will be stored in a brand new node there fore no multiple nodes will be made redundantly like in a trie. Therefore storing takes less time in a radix tree.

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

for wordlist1000.txt

	Trie	Radix Trie
add	0.000064 s	0.000037 s

the	0.000085 s	0.000071 s
your	0.000073 s	0.000013 s

for wordlist10000.txt

	Trie	Radix Trie
add	0.000236 s	0.000108 s
the	0.000398 s	0.000101 s
your	0.000096 s	0.000062 s

for wordlist70000.txt

	Trie	Radix Trie
add	0.000437 s	0.000187 s
the	0.001836 s	0.000636 s
your	0.000122 s	0.000073 s

By comparing these time for print suggestions, we can see when the dictionary size increase, time to print increases. Also for both data structures individually, when comparing the time, time increase if there are more matches for the prefix since it has to traverse through many nodes to get all the matches.
When length of prefix increase, number of matches decrease, therefore longer inputs takes less time.

When comparing time between 2 data structures, we can see radix tree takes less time than trie data structure, also because it has less number of nodes so there are less number of traversals.
Radix tree stores common prefixes in a common node and only make more nodes for unmatching nodes unlike trie structure. Therefore when traverse to find suggestions on a radix tree, the time taken mostly comes from traversal of the unmatched suffixes.
Since unmatched suffixes have own nodes, it takes less time. In trie each letter in a prefix and suffix associated with a node. Therefore it takes more time to print suggestions in trie structure.

Conclusion

So in conclusion,

- Radix tree uses less memory than trie structure.
- Radix tree takes less time to perform same operation than trie structure
- Therefore radix tree is more efficernt and saves memory.
- Downside of radix tree is it is much harder to implement than a trie.