Lab 4 - Tree ADT

Trie structure is used for **part1** and radix tree structure is used as modified data structure for **part2**.

1. Memory Space used

Case1 – wordlist1000.txt,

	Trie (Bytes)	Radix tree (Bytes)
Node size	216	224
Total number of nodes	3027	1314
Used Memory space	653832	294336

Case2 – wordlist10000.txt,

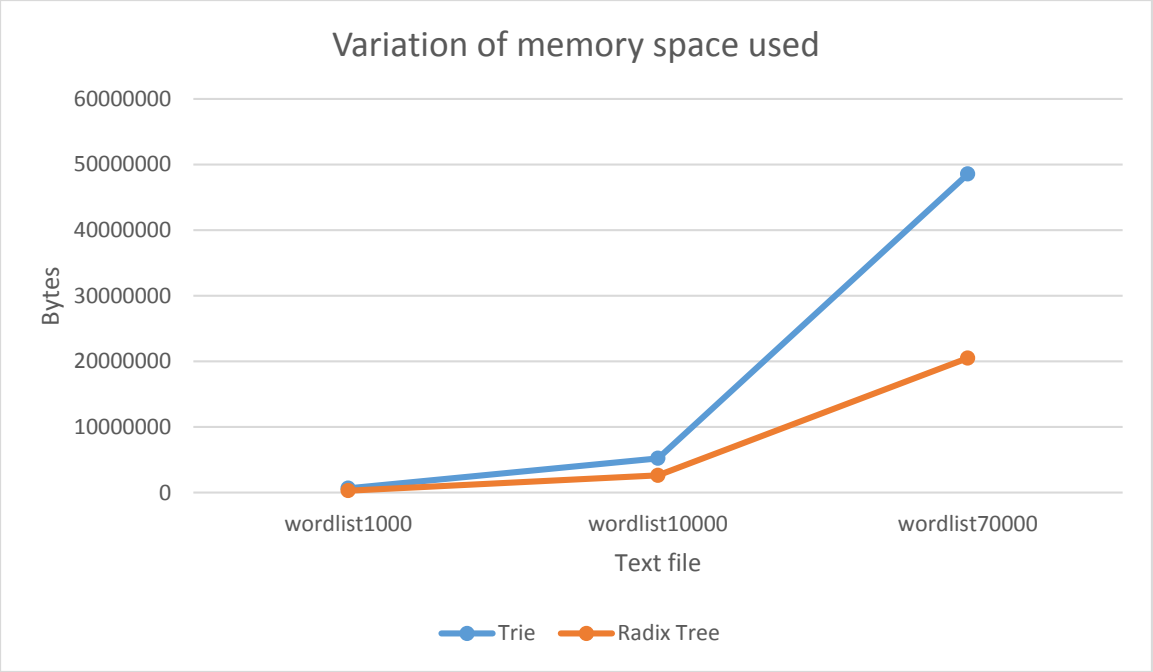
	Trie (Bytes)	Radix tree (Bytes)
Node size	216	224
Total number of nodes	24179	11704
Used Memory space	5222664	2621696

Case3 – wordlist70000.txt,

	Trie (Bytes)	Radix tree (Bytes)
Node size	216	224
Total number of nodes	224753	91503
Used Memory space	48546648	20496672

Trie has an empty root node, with links to other nodes one for each possible alphabetic value. The shape and the structure of a trie is always a set of linked nodes, connecting back to an empty root node. An important thing to note is that the number of child nodes in a trie depends completely upon the total number of values possible. For example, if we are representing the English alphabet, then the total number of child nodes is directly connected to the total number of letters possible. In the English alphabet, there are 26 letters, so the total number of child nodes will be 26. The main disadvantage of tries is that they need a lot of memory for storing the strings. For each node we have too many node pointers (equal to number of characters of the alphabet).

Radix tree is a data structure that represents a space optimized trie (prefix tree) in which each node that is the only child is merged with its parent. In this structure number of nodes used for storing words is very low compared to trie structure.



As an example memory space taken to store set of words in wordlist1000.txt in trie is 653832 bytes and radix tree only takes 294336 bytes.

2. Time taken to store the dictionary

Case1 – wordlist1000.txt,

	Trie (seconds)	Radix tree (seconds)
Time taken to store	0.003462	0.001448

Case2 – wordlist10000.txt,

	Trie (seconds)	Radix tree (seconds)
Time taken to store	0.006751	0.005351

Case3 – wordlist70000.txt,

	Trie (seconds)	Radix tree (seconds)
Time taken to store	0.057120	0.043908

If string is of length n, then using trie worst case time complexity for searching the string is O(n). Insertion also takes O(n) time in worst case.

According to the above data, time taken to store the same set of words is low in radix tree structure when compared to trie node structure.

As an example for storing wordlist1000.txt trie structure takes 0.003462 seconds and radix tree takes only 0.001448 seconds.

3. Time taken to print a list of suggestions for chosen word prefix.

Case1 – wordlist1000.txt,

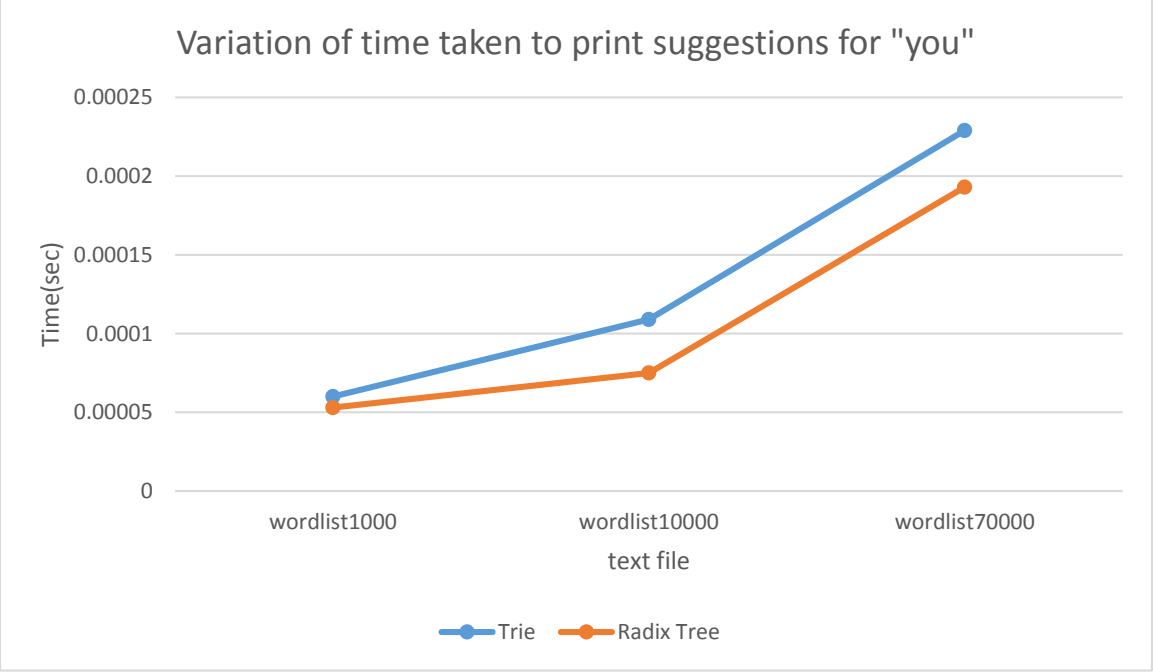
Prefix	Trie (seconds)	Radix tree (seconds)
you	0.000060	0.000053
a	0.001046	0.001014
he	0.000203	0.000150
res	0.000095	0.000081

Case2 – wordlist10000.txt,

Prefix	Trie (seconds)	Radix tree (seconds)
you	0.000109	0.000075
a	0.005393	0.004171
ros	0.000068	0.000055
ka	0.000195	0.000139

Case3 – wordlist70000.txt,

Prefix	Trie (seconds)	Radix tree (seconds)
you	0.000229	0.000193
a	0.019054	0.018826
ros	0.000675	0.000389
ka	0.001746	0.001209



As we can see in the above tables time taken to print a suggestion list for a given prefix is always high in trie structure and low in radix tree structure. As an example for prefix “you” to print the suggestion list in wordlist1000.txt trie takes 60 micro seconds and radix tree structure takes only 53 micro seconds.