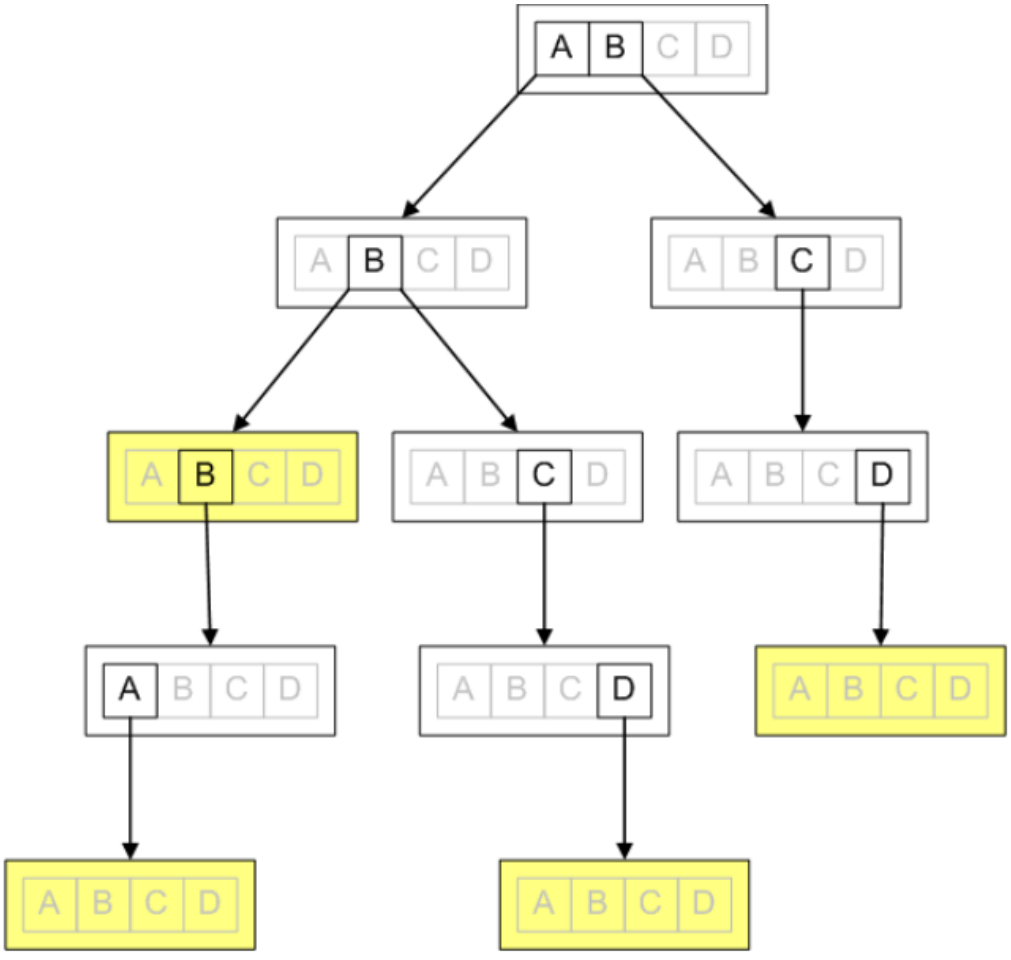
E/14/049

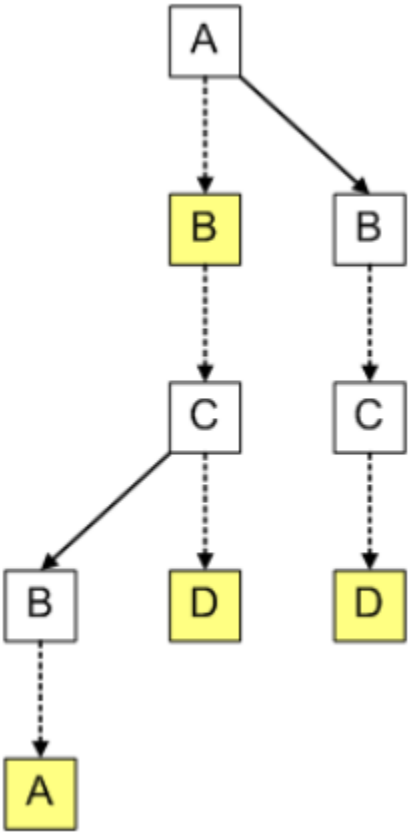
Report

Before discussing about the efficient algorithm for autocomplete the search word , let's take a look at a simple data structure that supports a fast auto-complete lookup but needs too much memory: a *trie*. A trie is a tree-like data structure in which each node contains an array of pointers, one pointer for each character in the alphabet. Starting at the root node, we can trace a word by following pointers corresponding to the letters in the target word.



As we can see in the figure it wasting lot of memory. Problem has to do with the memory taken up by all the null pointers stored in the node arrays. We could consider using a different data structure in each node, such as a hash map.

However, managing thousands and thousands of hash maps is generally not a good idea, so let's take a look at a better solution which is the ternary tree.



Test case 1 : search the word "som"

	Time taken to store the dictionary (in micro sec)	time taken to retrieve the data (in micro sec)
Q1	1000	2000
Q2	1000	1000

Test case 2 : search the word “sister”

	Time taken to store the dictionary (in micro sec)	time taken to retrieve the data (in micro sec)
Q1	1000	1000
Q2	1000	1000

Test case 3 : search the word “k”

	Time taken to store the dictionary (in micro sec)	time taken to retrieve the data (in micro sec)
Q1	1000	4000
Q2	1000	5000

Therefor Even though the 2nd method is lengthy it is more efficient than the first one.. if we try large dictionaries the result would be much favorable for the 2nd method.