Auto complete

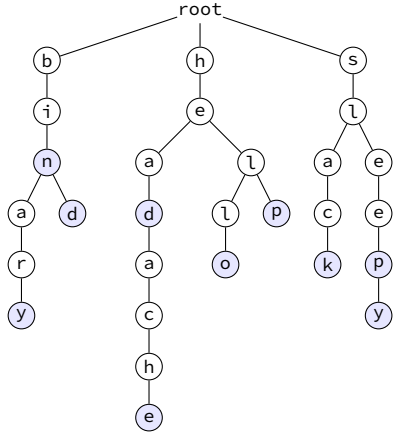
Most text editors have functionality for "auto completion", where a small box appears with possible ways to complete a started word. Below we provide two possible data structures that can be used to implement a simple mechanism for completing a started word.

We show examples of how the data structures look if they store the words: `bin`, `binary`, `bind`, `head`, `headache`, `hello`, `help`, `slack`, `sleep`, `sleepy`.

Strategy 1

The first strategy is to use a *prefix-tree* that holds strings. A prefix tree has a root. If a string s is inserted into the tree, the root node has a child with the first letter of s. That node in turn has a pointer to the second letter of s, and so on. The last letter of s is marked as a *terminal* node (the terminal nodes are colored blue in the example). That is, if you follow a path from the root node to a terminal node, then you have spelled a word that is in the tree.

From a prefix tree it is easy to find all ways to complete a started word. It is done by returning all paths from the root node that begin with the letters from a given word and ends in a terminal node.



Strategy 2

The second strategy is to use a hashmap, where we let each prefix of a word map to a list of all possible ways to complete the word. If `H` is a hashmap, and s is a string to be added, we consider each prefix p of s and add s to the list `H[`p`]`. In the example to the right, you see how the hashmap might look if all the example words have been added (note that some of the lists are shortened, indicated by "…").

From such a hashmap it is very easy to find all ways to complete a beginning words. This is done by looking up the started word in the hash map and return the list.

Key	Value
`""`	`"bin", ..., "sleepy"`
`"b"`	`"bin", "binary", "bind"`
`"bi"`	`"bin", "binary", "bind"`
`"bin"`	`"bin", "binary", "bind"`
`"bina"`	`"binary"`
`"binar"`	`"binary"`
`"binary"`	`"binary"`
`"bind"`	`"bind"`
`"h"`	`"head", ... "help"`
`"he"`	`"head", ... "help"`
`"hea"`	`"head", "headache"`
`"head"`	`"head", "headache"`
`"headа"`	`"headache"`
`"headac"`	`"headache"`
`"headach"`	`"headache"`
`"headache"`	`"headache"`
`"hel"`	`"hello", "help"`
`"hell"`	`"hello"`
`"hello"`	`"hello"`
`"help"`	`"help"`
`"s"`	`"slack", "sleep", "sleepy"`
`"sl"`	`"slack", "sleep", "sleepy"`
`"sla"`	`"slack"`
`"slac"`	`"slack"`
`"slack"`	`"slack"`
`"sle"`	`"sleep", "sleepy"`
`"slee"`	`"sleep", "sleepy"`
`"sleep"`	`"sleep", "sleepy"`
`"sleepy"`	`"sleepy"`

Discuss the advantages and disadvantages of the two different strategies both with regards to runtime and memory usage