Trie

@K3NSAH

NOTE: numbers are inclusive

You will write an application to build a tree structure called Trie for a dictionary of English words, and use the Trie to generate completion lists for string searches.

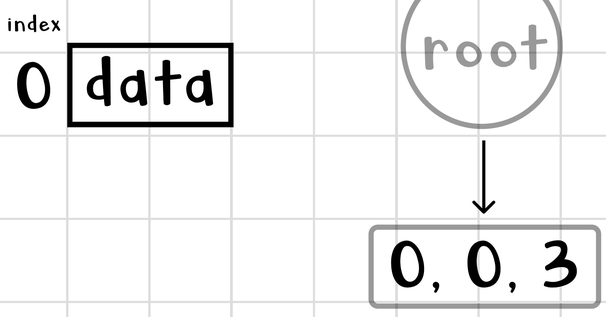
index of word

position of first common char

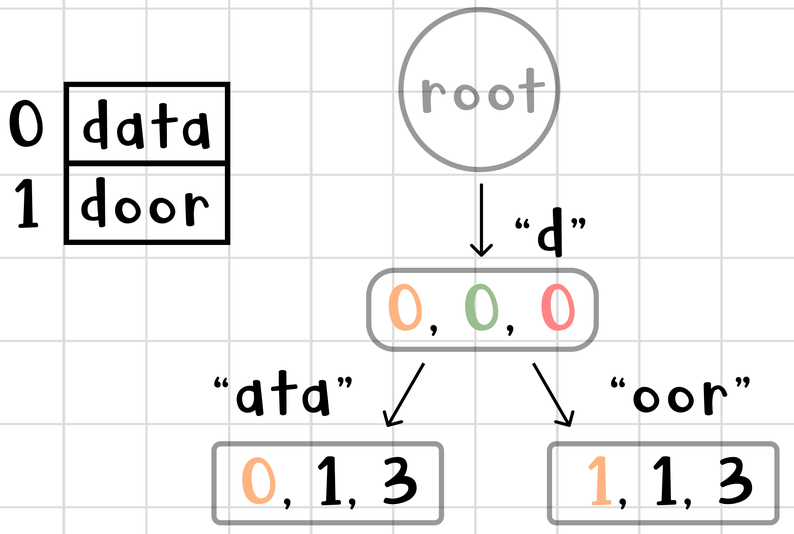
position of last common char

* word list is originally stored in an array that the trie is built off of.

Trie #1

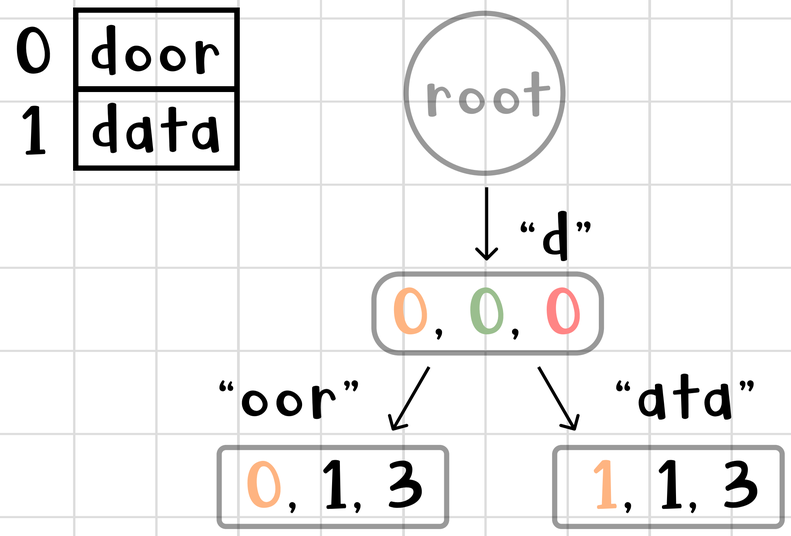


Trie #2



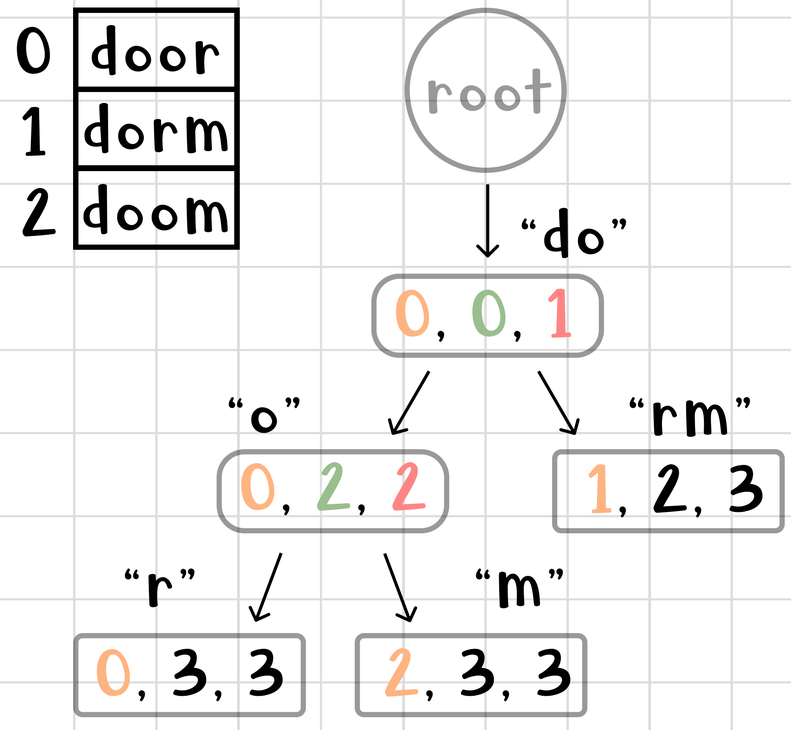
* internal nodes represent prefixes, leaf nodes represent complete words.

Trie #3: if "door" came before "data" in the array

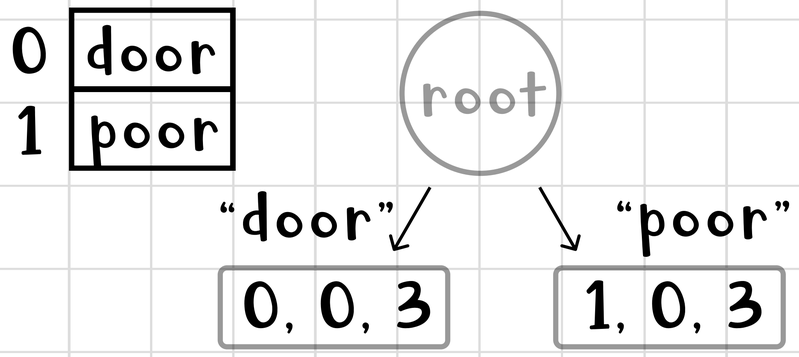


* array fills tree by inserting left child then right child.

Trie #4



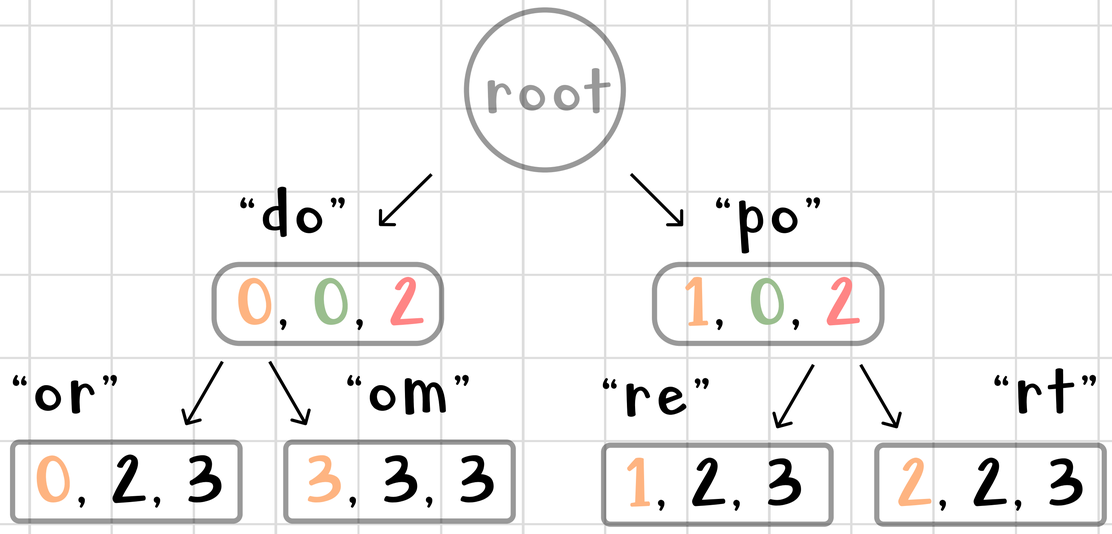
Trie #5: no common elements



* we do not care about common suffixes

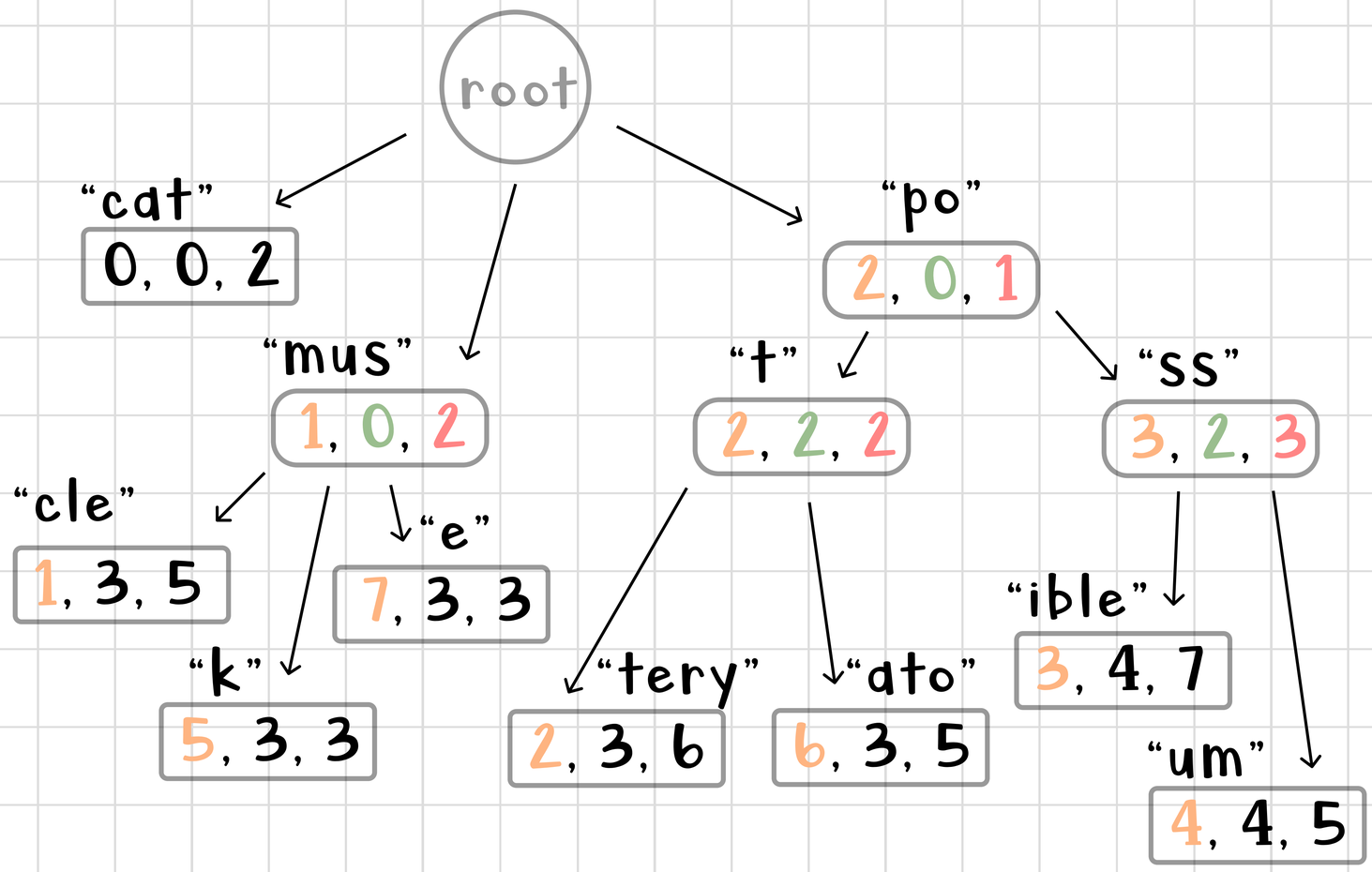
Trie #6

0	door
1	pore
2	port
3	doom



Trie #7: nodes can have any number of children

0	cat
1	muscle
2	pottery
3	possible
4	possum
5	musk
6	potato
7	muse



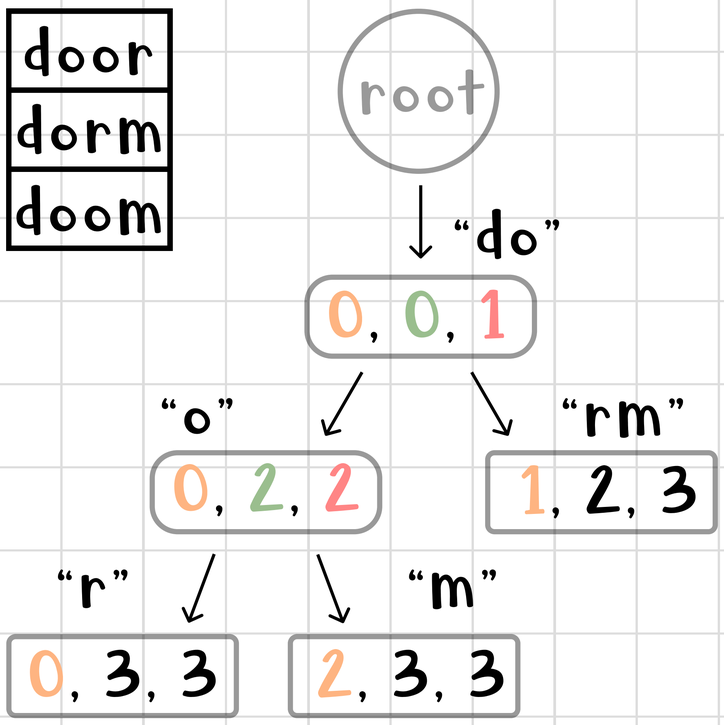
Trie Data Structure

Since the nodes in a trie have varying numbers of children, the structure is built using linked lists in which each node has three fields:

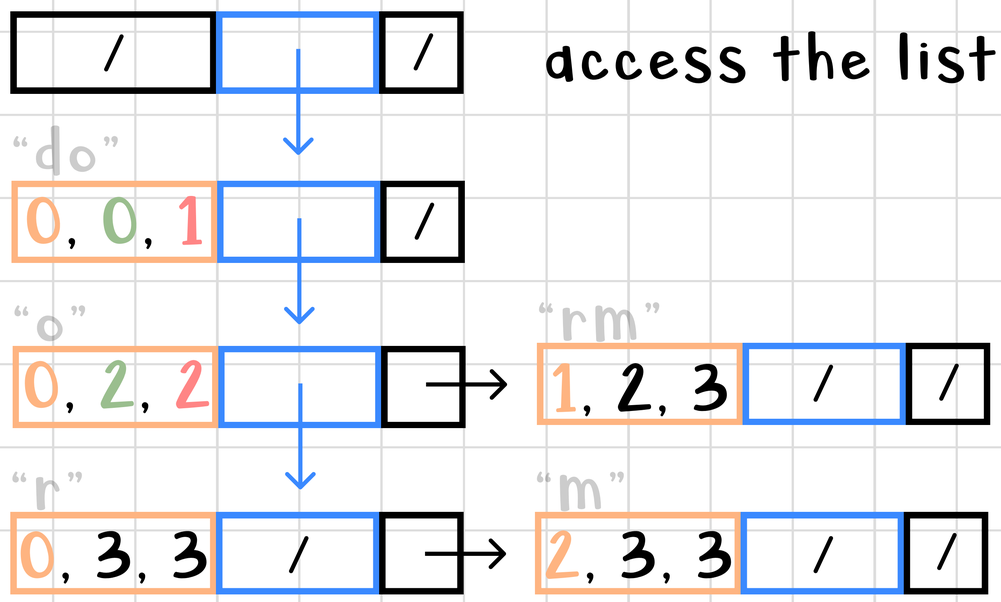
- substring (triplet of indecies)
- {first} child
- sibling (ptr to next sibling)

example trie

0	door
1	dorm
2	doom

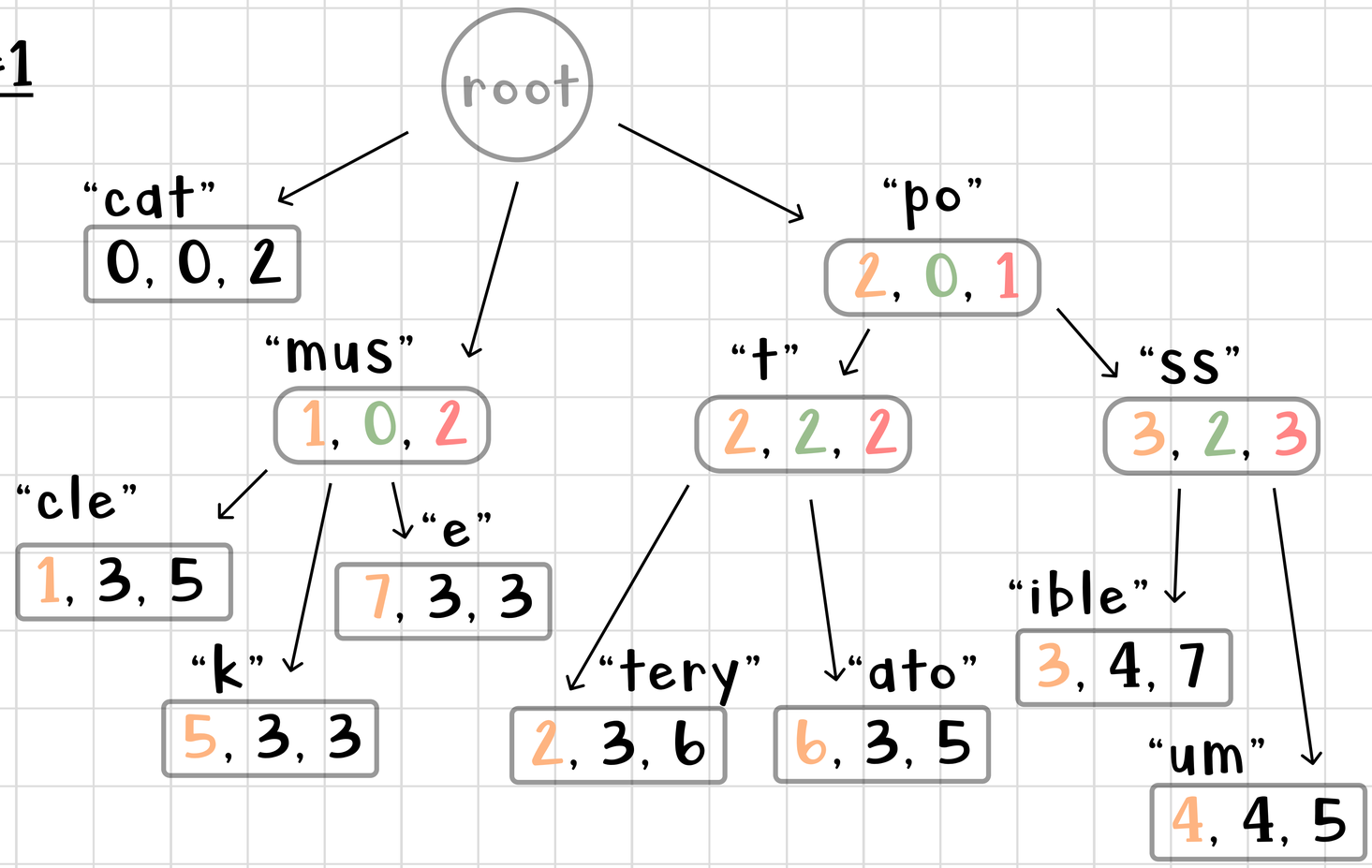


example data structure



Completion List

Example #1



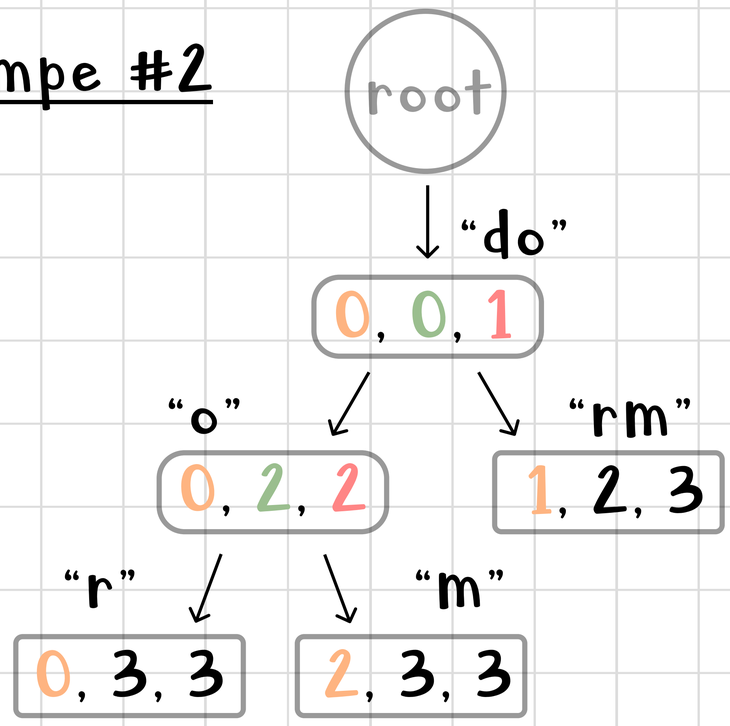
completion list: mus
↳ muscle, musk, muse

completion list: po
↳ pottery, potato, possible, possum

completion list: pa
↳ no match

completion list: possums
↳ no match

Exampe #2



completion list: dom
↳ no match

completion list: d
↳ door, doom, dorm