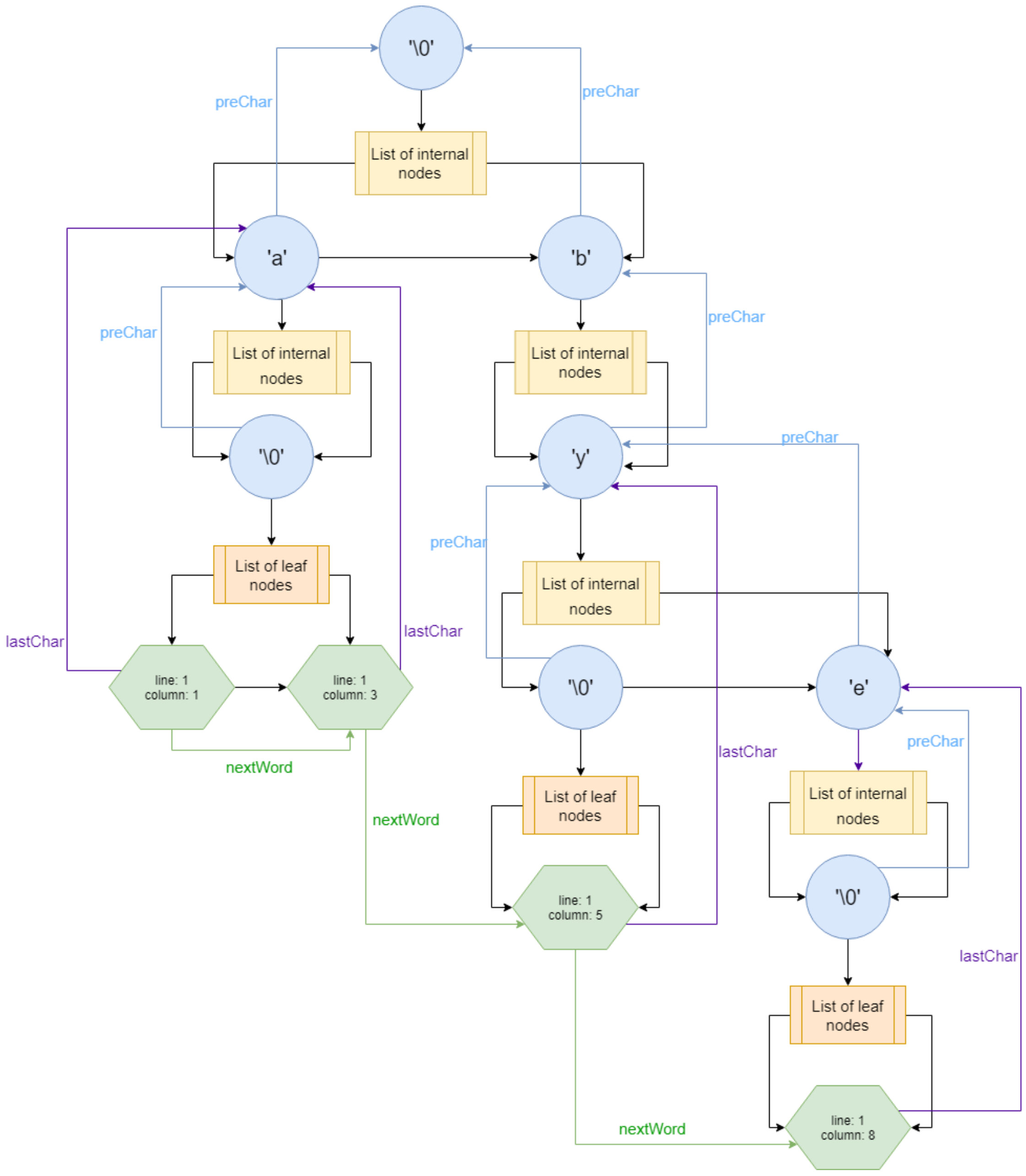
COMP3506 Assignment 2

The main data structure is a standard retrieval tree based on linked lists. The nodes are in two types: internal or leaf. Internal nodes contain the character value, link to the parent node (the previous character), link to children (a linked list of internal nodes representing next characters or a linked list of leaf nodes representing occurrences), and link to the next internal node. Leaf nodes contain the line and column number of the word, link to the internal node representing the last character of the word, link to the next word's leaf node, and link to the next leaf node in the list. A string “a a by bye” will be stored in the trie as illustrated below:



The program preprocesses the original text file to allow faster text searching. Operations such as insertion and searching one word takes O(dm) time where d is the size of the alphabet (27 in this case: 26 English characters plus apostrophe) and m is the size of the searched term. 27 is relatively small and the searched term is usually short. The trie stores each word's occurrences in the leaf nodes, so the position information can be retrieved directly after finding the word in the trie.

Other possible solutions:

1. Compressed trie:

Compressed trie may use less space O(s) where s is the number of strings in the file, while my standard trie uses O(n) where n is the total size of the strings in the file. The drawback of a compressed trie is that the trie needs to be restructured every time when inserting a new word that has the same prefix as some existed words, which could take a lot of extra time for constructing the trie. Also, the runtime of searching in a compressed trie is not faster than a standard trie. Its memory usage advantage is obvious for long strings with a long common prefix, while in a large text file, there are usually many repeated words, short words, and words that do not share a prefix. Therefore, the extra effort and runtime taken for restructuring may offset the memory advantage.

2. Array-based trie

Arrays can be more efficient in terms of searching but take more memory. The runtime will be O(m) where m is the size of the searched item instead of O(dm) where d is the size of the alphabet. Each internal node contains an array in size of 27 linking to each child nodes, so the child representing the next character can be located directly rather than looping through all children as a linked list does. In this case, the memory usage will be O(dn) where n is the total size of the file. Since the text file is large, n will be far bigger than m. Additionally, the list of leaf nodes i.e. occurrences of each word still need to be dynamic because there is no way to know the number of occurrences in advance. Therefore, I chose to use linked lists for both leaf nodes and internal nodes.

3. Without preprocessing

Store a list of all words. Searching will check every single word in the list which can be far inefficient.

When searching a phrase, only the first word is searched in the trie. Then for each occurrence, the method checks whether the following words matches the rest of the phrase.

Logic searches (and, or, not, and compound) share similar codes so I put them together and use the variable "mode" to direct to a specific logic. When adding a new list to another list, if the list is temporarily created by another method instead of what exists in the trie (e.g. list of leaf nodes), it is fine to change its tail node. Hence, the original list can just add the new list's head node and set the tail as the new list's tail. The rest of the nodes are linked in the new list. There is no need to loop through each node in the new list anymore.

The idea of logic searches within lines or sections is to deal with each searched word's occurrence line number instead of really searching words on each line or section so that words are only searched once in the document trie.