Data Structures Used :

Since our text editor is meant to handle large files and perform search operations on them and perform context-based auto-complete recommendations, we need an efficient way to store the words in a tree like data structure to exponentially reduce lookup times as compared to linear searches in the same data. For this we have designed two custom classes, based on multiway trees.

1. **Alphabet:**

This data structure represents a single node of the multiway tree. For out customized purposes, it stores a lot of extra data with itself too. It includes:

**[\_\_\_**  data: The alphabet that is stored in the node.

successors: A hash-map of all {alphabet : instances\_list} where alphabet is the character that succeeds the present character and List is a list of Alphabet objects, denoting every instance.

level: Denotes the position of the alphabet in a word.

ending: Denotes whether the Node is an instance of the last alphabet of a word.

end\_pos: Denotes the ending position of the word in the text. **\_\_\_\_]**

1. **Trie:**

This is the primary handler class of the nodes that is responsible for adding a new word to the structure and displaying the contents and other related data of the word structure.

It has the following functionalities:

add\_word: Adding a word to the existing structure.

get\_all\_subnodes: Does a full traversal from that node and returns a list of all nodes that are found under the current node.

get\_suggestions: Takes in a substring and then traverses the structure to get all possible auto-complete recommendations for the string.

highlight: Takes in a substring, searches for it and returns address of all instances of the substring. This makes the search results visible to the user.