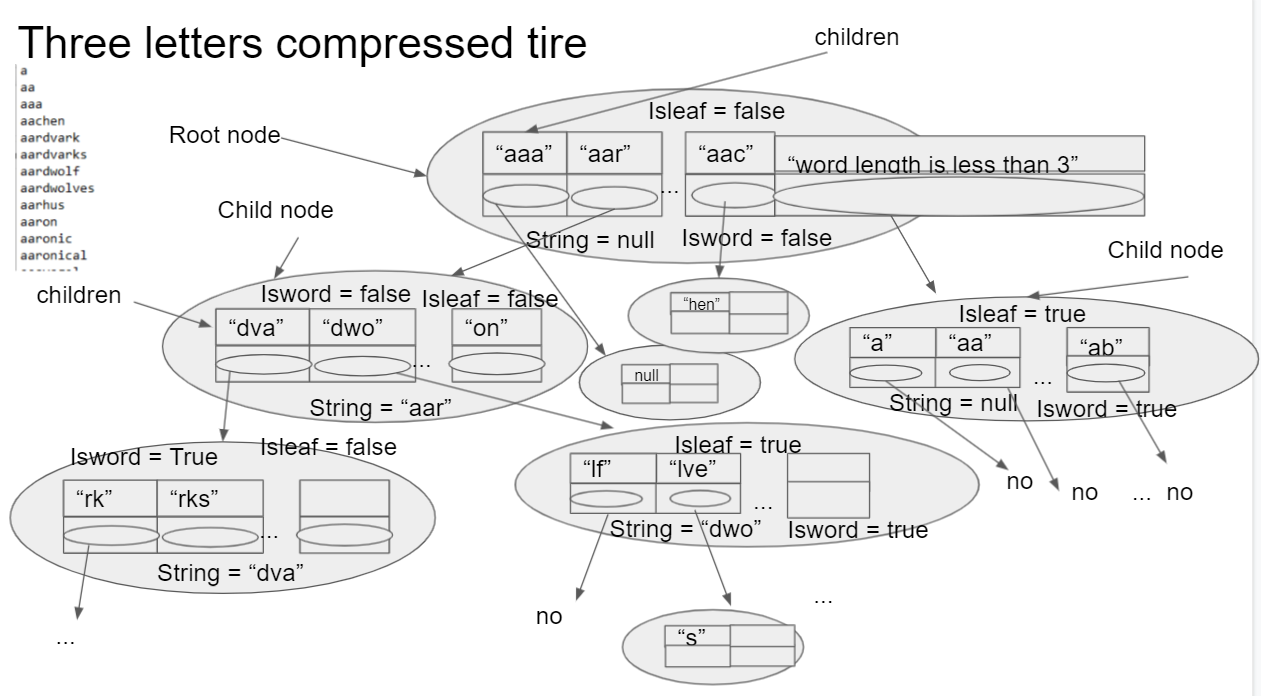
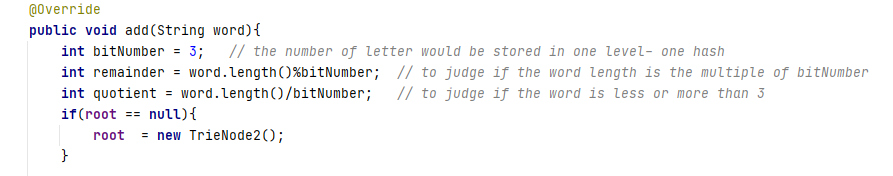
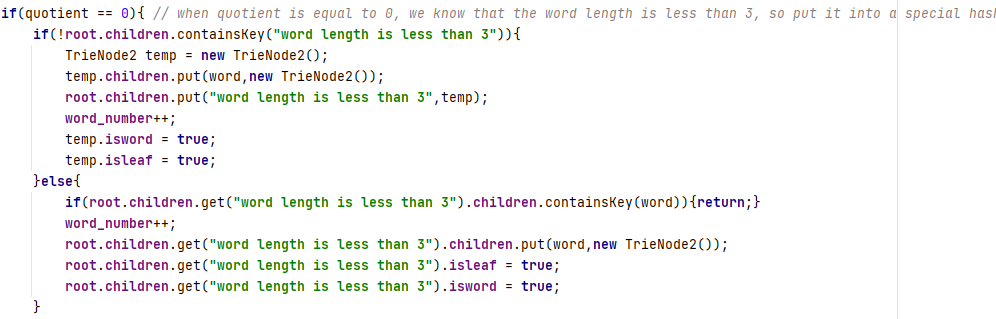
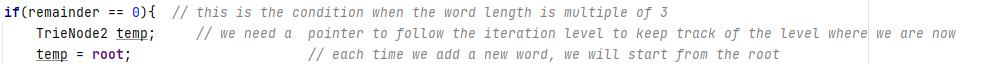
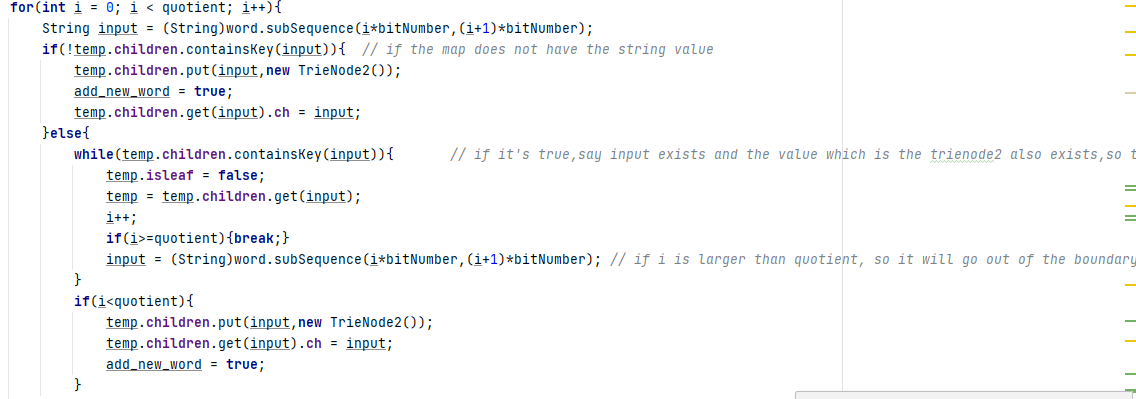
This is Three Character Compressed Trie which is different from the single Character Compressed Trie but to some degree they have the same basic rule. As what can be seen from the figure below, it has a clear introduction about what a Three Character Compressed Trie looks like:

 Each node of the trie stores 3 letters for one time rather than only one character each time which will save the storage space more. Each node of the trie has exact five objects. First each node has a linkedhashmap which helps to store key of the string word and the value of next level child nodes. Also, each node has 4 flags (one has not been drawn in the figure). String flag is used to depict which this trie node belongs to and the root node of each trie must have a null string flag because the root node will not belong to any string. For example, like what can be seen in the figure, the node in the first position of the second level is from String ”aaa” in the root node, so this node just belongs to the string “aaa” and the string flag will be assigned value as “aaa”. Isword flag has the function of showing if all the string of all the child node from the root node to current node can compose a whole word like “aap”,”end” can compose a whole word. So, the isword flag in the “end” node will have value of true. Apparently, isleaf flag is used to depict whrther this node is a lead node, if it is the flag value will be true otherwise false. The last but not least is the counter which can show the exact order when this word is added into the trie(e.g. the word ”append” is the first word added into the trie, it has word\_number = 1)and even though when the duplicated word is added into the trie, the counter will not plus 1. So counter can also count how many words has been added into this trie almost like a dictionary.

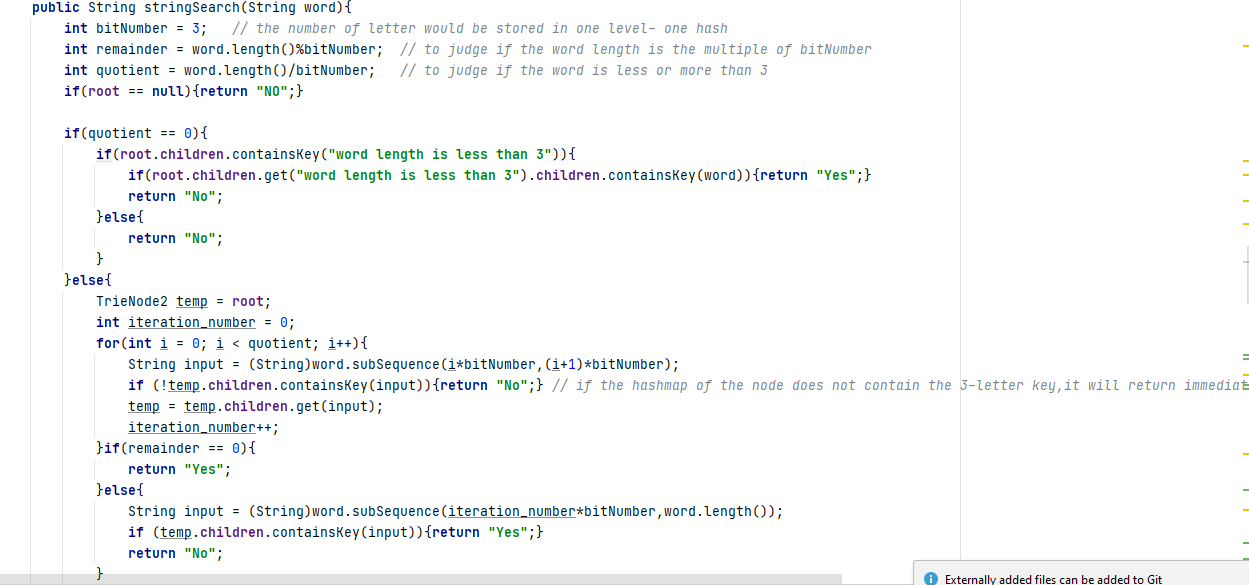
Meanwhile, Dividing and Storing words when we add or search the exact word will be 3 letters for one time. For instance, if a word which is “append” wants to be added into this trie, first, divide this word in the way of three char by three char. So the dividing result is “app”, ”end”. For those which word length is less than 3, the root hashmap has a special key ”word length is less than 3” and it has a child trie node. In this trie node, there is a special hashmap stores these special strings.

 Meanwhile, from the code above, we need to judge which kind the string word belongs to. And then do the following operations. The more detailed code is all in GitHub.

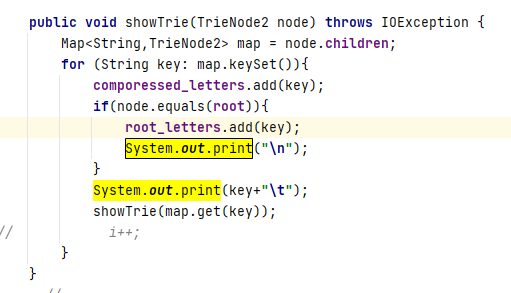
In the second step, String “app” will be added into the root node by using the linkedhashmap function “put(key,value)”, and then the value is just the child trie node. Using a pointer temp to follow the exact level where the current level is, so after adding the “app” in the root it can go into the second level. In this child trie node, do the same operation as what did in the root node and the third level and so on so forth till the whole word has been added into the trie at the last step.

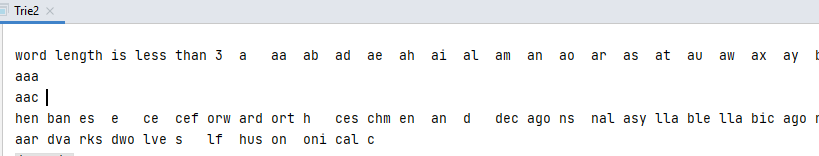
However, if the word is has been added into the trie in the past, and it wants to be added for one more time, it will start the check operation. For example, like the code above, after dividing the word into several segments, use the hashmap function “containsKey” to judge if this string has already existed in the hashmap, if so, the pointer “temp” will go the next level to check the next 3-letter string and so on so forth. However, if the string is not in the hashmap, so a new key will be added into the hashmap then create a new branch (a new child node). In conclusion, when all word have been added into the whole trie, the trie resembles a tree structure. So the name of the trie is almost from the tree.

As it is like a tree, the adding and searching speed is fast, every time when traversing the trie, the complexity would be O(log n) ,at the same time, this kind of trie also compress the whole dictionary. So from the time complexity and space saving, this kind of structure has both advantages.

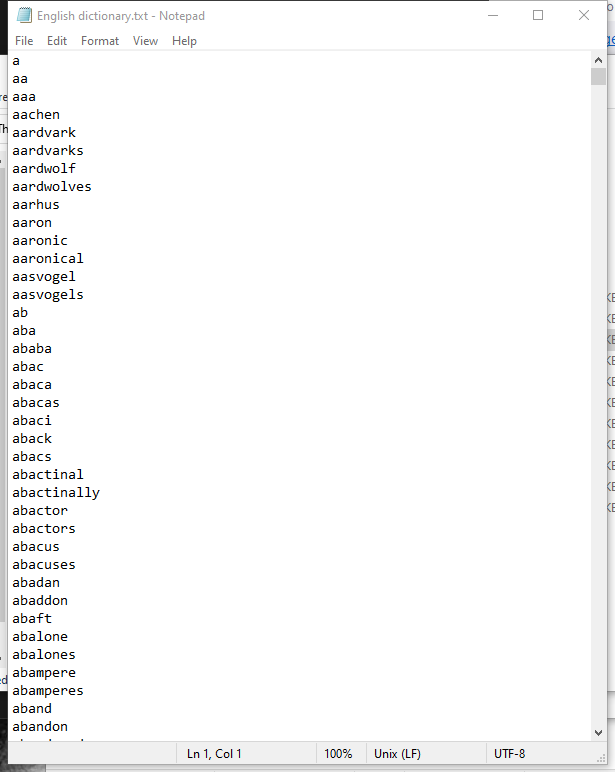
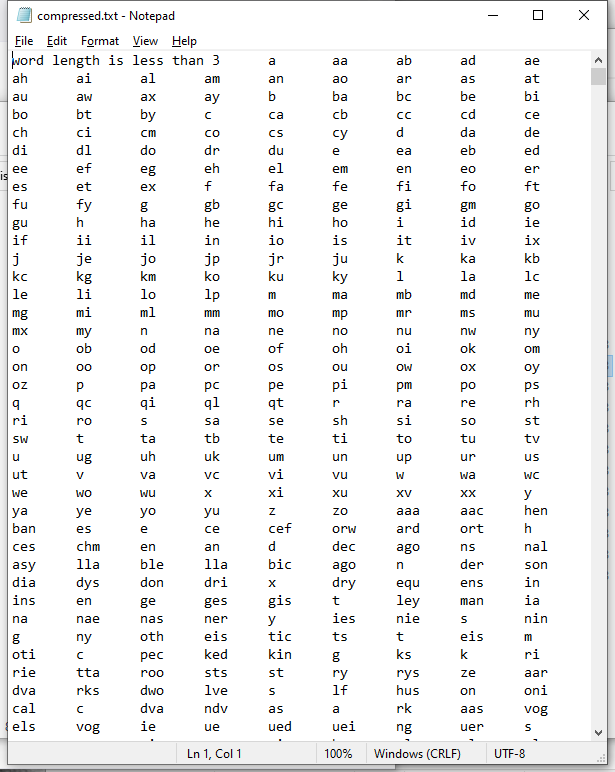


Above is the searching part of the code, the structure of the code is quite like the add part, so no more explanation here. More detail could be seen in code and the commentary in GitHub.





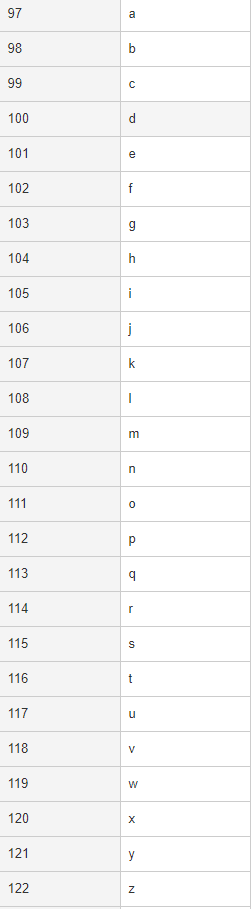
Then the system.out.print shows that the string word are printed branch by branch, which means that it will from the first key of the root node to the bottom(i.e. the leaf node of this branch) and then return to the root node and next branch, so on so forth. So, it is clear for users to check the word added into the trie.

Above is the result of compressed into txt file, so input the word also branch by branch, which means that after writing into the “word length is less than 3” and the following word, then it will write into the txt file next branch.



From the screenshot above, the first step compressed efficiency is quite good which is more than fifty percent of compressed rate.

*01100001*

*01100010*

*01100011*

*01100100*

*01100101*

*01100110*

*01100111*

*01101000*

*01101001*

*01101010*

*01101011*

*01101100*

*01101101*

*01101110*

*01101111*

*01110000*

*01110001*

*01110010*

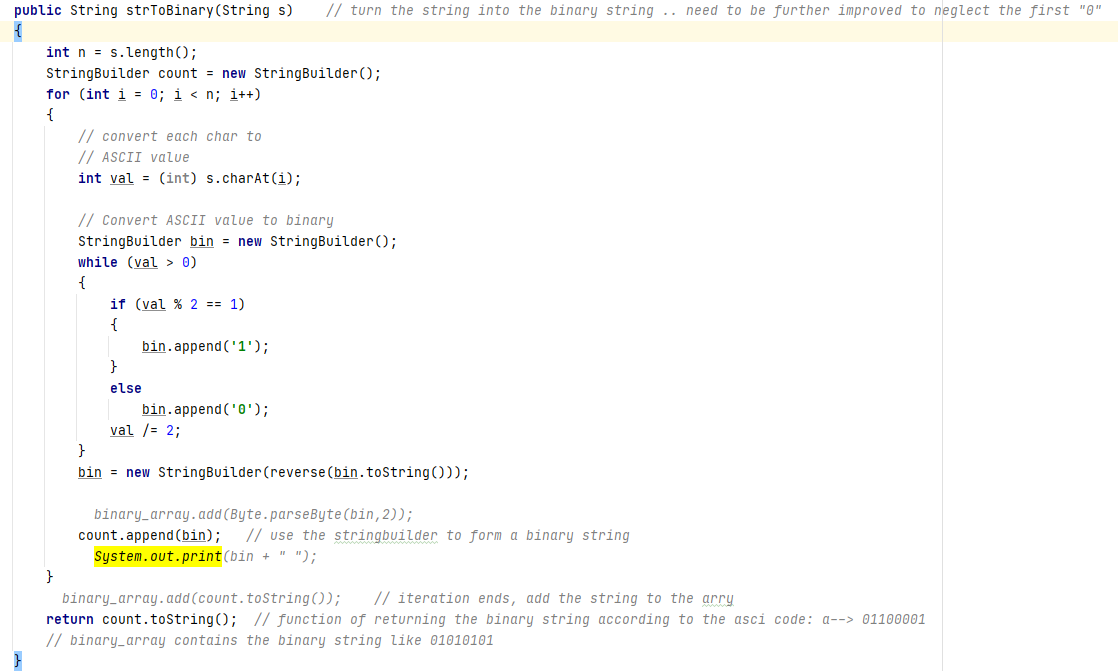
*01110011*

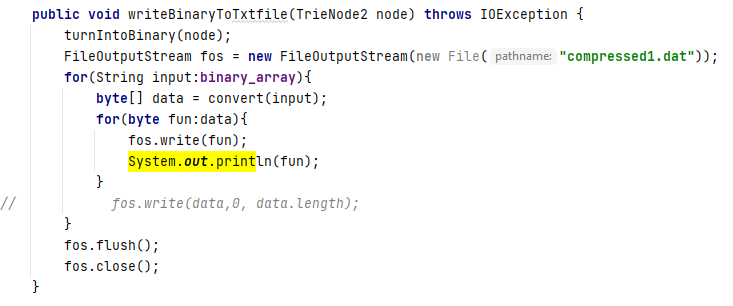
*01110100*

*01110101 01111000*

*01111001 01110110*

*01111010 01110111*

Then because a byte word is just one byte, and it can represent an English word regardless of the uppercase or the lowercase and even some special signature only using from -128~127 value range which means that only using the 7 bits (the first digit is always 0 which can be ignored). Basing on this, one method of using arithmetic encoding way can be implemented. First, the string in each node should be transformed into this kind of binary string word. The process can be seen in the screenshot above. The method input is each string and the output is the binary string like from “aaa” to “01100001”, actually there is a mistake here. This function needs to be further improved to return a 7 bits string like “110001” rather than 8 bits string. Or in some other place of the method, this mistake needs to fixed and the method needs to be further finished. Because the imminent presentation, basically this need has not been met well. In the future work, this problem will be solved.



Then the returned binary string is stored in the array of “binary\_array”, this binary array will be written into a .dat file and which will do a further compression.

Because there is a mistake here needed to be fixed in the future work, the original compression result maybe wrong, but I still put it here just for reference:



So from the result, the more compression rate is achieved for about 10% more.

This is the three letters compression trie. Meanwhile all the code is written by myself only a little part of citation which has been written in the citation txt file in the GitHub.