Hw 7 Idea:

if given dictionary dict[] stored in a file

Index	Word Literal	frequency
0	a	7300
1	aa	6
2	aac	3
3	aach	1
4	aalto	2
5	ab	10
⋮	⋮	⋮

step 1: create an empty prefix Tree, called tr in your AutoCompleteStudent class. That is, tr is an instance variable in your AutoCompleteStudent class.

step 2: Insert each word W from dict[] array into tr. Using method tr.insertStr(dict[i]) defined in the prefix Tree class, where dict[i] means an arbitrary word from the dictionary.

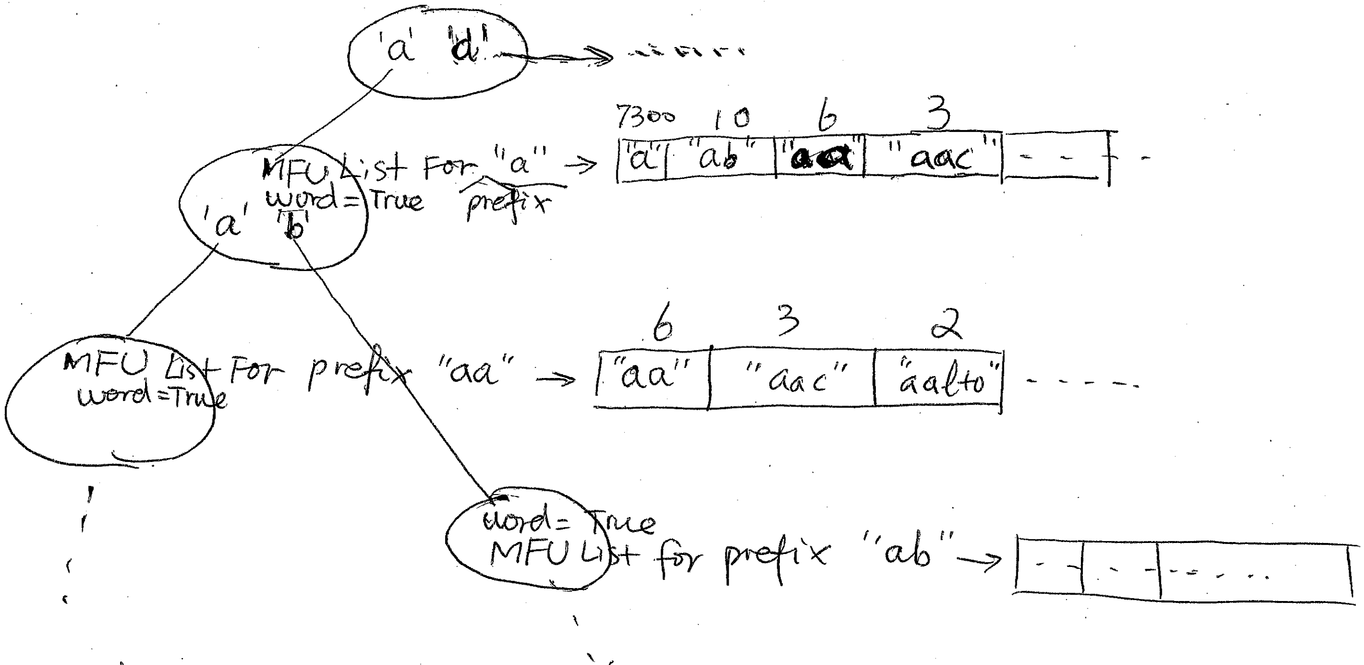
Note: step 1 and step 2 should be performed in the Constructor of your AutoCompleteStudent Class. That is, we have to have the data structure and data ready before we search in the prefix tree tr.

Note: in your Trie2.java class, in Your Trie Node definition, Your need an extra variable, ArrayList<string> mostFreqUsed. We learn why we need this next.

MFU for prefix p is defined as a list of Most frequently used words that has a prefix p.

The purpose of step 1 and step 2 above:

- After the data structure tr is created, The Tree looks likes in below.



We need an instance variable in TrieNode class to store a List of most frequently Used (MFU) words for the prefix string that the TrieNode is associated with, Because each Trie Node is implicitly associated with a prefix of some words defined in the dict[]. See the diagram above, find the TrieNode that is associated with prefix "aa".

- When we typing in the GUI program, we incrementally build many prefix strings, until we complete typing a whole word.

  type → "a" → prefix "a"
  then → type "b" → prefix "ab"
  then → type "s" → get prefix "abs"

- In order to allow user to do auto completion, in the GUI we have to display the MFU List of words for each these prefix Strings you entered. type 'a' → prefix "a" → Show MFU for "a"
Continue type "b" → prefix "ab" → show MFU for prefix "ab" ....

. In the GUI, the above operation (return the pre-stored MFU List for a prefix P in the Trie) is performed by Method tr. search (P)

---

idea of insertStr(s) of prefix Tree Class, where s is a string literal of a word in dict[].

```
TrieNode cur = root of current prefix tree.
prefix = "" ; // empty string
for each character ch in s {
        append ch to prefix ;
        TrieNode next gets the child TrieNode that is
        associated with character ch In cur.
        if ( next == null ) { // if No such Node
                Create a new TrieNode temp ;
                put letter ch into Node cur, associate
                ch with the new Node temp.
                next = temp ;
                next. mostFreqUsed = computeMFU (dict, prefix)
        }
        cur = next ;
} // end for
Cur. word = True ; // meaning this is a word in dict[]
Done the program .
```

idea of method computeMFU (dict, prefix)

if given dict[] as follows:

index	word Literal	frequeney
0	a	7300
1	aa	6
2	aac	3
3	aach	1
4	aalto	2
5	ab	10
⋮	⋮	⋮

when we performing computeMFU(dict, "aa");
We like to compute a list of most frequently Used words Using the information in dict[] array.

Step ①: Do binary search "aa" in dict[] array, because dict[] is sorted in dictionary order.
Note: If "aa" is Not a valid word appearing in dict, You should find the index of first word Wf in dict that has a prefix "aa".
In the example above, this step ① returns index 1.

step ② you do binary search to find the last word in dict[] that has the prefix "aa", returns index 4 in the example.

step③ Sort all words between index 1 and index 4 in dict according to its frequeney number, You get List of MFU for prefix "aa"

aa → aac → aalto → aach
6 3 2 1