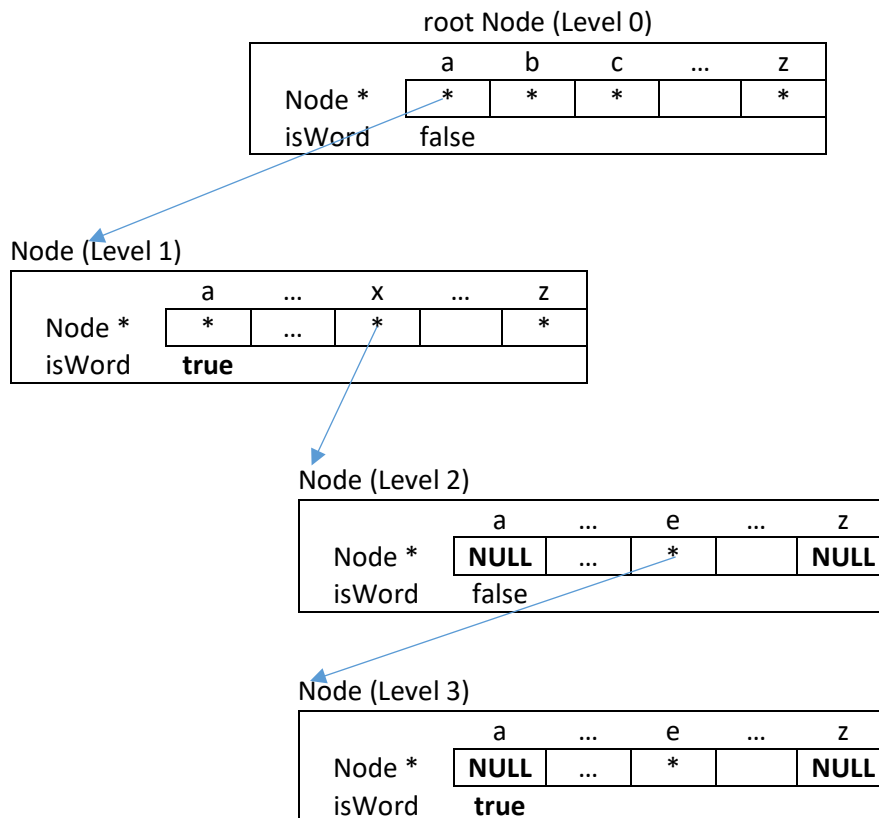


Part 4a: Dictionary

Prefix Data Structure

To store the words in this assignment, you will be creating dictionary using a data structure called a prefix tree. This data structure allows for the quick lookup of whether or not a word is valid. It will also allow you to find all words with a specific prefix.

Rather than store each word individually, we instead store the words using a tree:



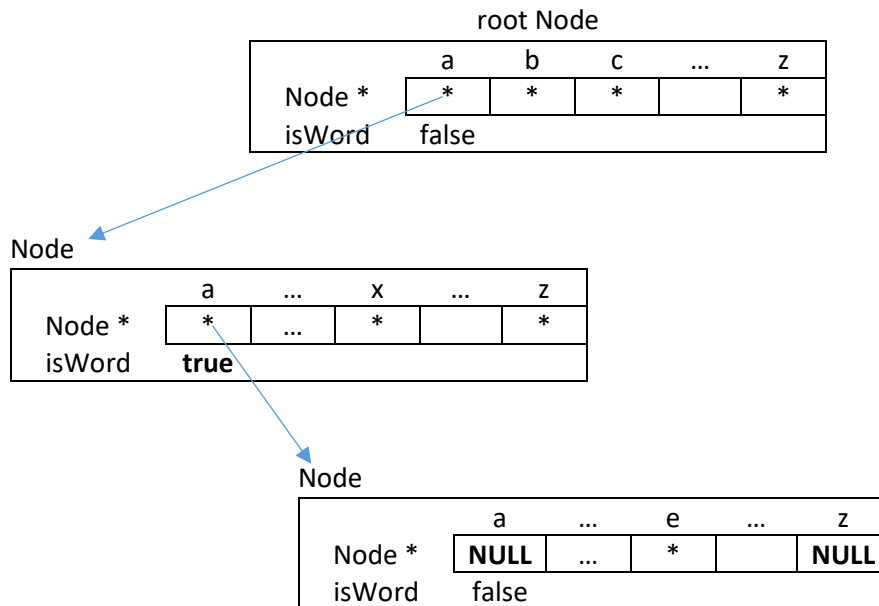
The example above shows two how the words “aa” (abbreviation for administrative assistant) and “axe” are represented using a tree. Each node holds 26 pointers to other nodes; each of these nodes corresponds to a specific letter. Each node also holds a single boolean flag.

Essentially, each word is represented by a path down the tree. If the path ends with a “false”, then the path does not represent a valid word. For example, the root node has a path from the first Node * pointer (i.e. the pointer representing ‘a’) to another Node. Notice that this first level node has a “true” flag. This indicates that “a” is a valid word. If we examine the pointer for the second level “x” position, we find that a path exists for “ax” to another node. When we examine this second level node, we find that the pointer for the ‘a’ position is NULL. This indicates that “aa” is not a valid word or prefix. If we examine the pointer for the ‘e’ position, we see that another node exists. When we look at this 3rd level node, we find that the flag for this position is true; this make sense since “axe” is a valid word. Notice

that the 'a' Node pointer at the third level is NULL; this means that there is no word with a prefix of "axea". Similarly, since the node for 'z' at the third level is NULL, this means there is no word with the prefix "axez".

Prefixes

A prefix is a valid path that may or may not be a valid word. For example, "a" and "ax" are valid prefixes since there are words starting with these letters. However, "aaa" is not a valid prefix; this is represented by a NULL at the end of the path.



Milestone 1 – Implement and test the Dictionary

Your first job is to implement the Dictionary class. The following is the header file:

```
struct Node {
    // Your data structure goes here.
};

class Dictionary
{
public:
    Dictionary();
    Dictionary(string file);
    void addWord(string word);
    bool isWord(string word);
    bool isPrefix(string word);
    void PrintWords(string prefix);
    int wordCount();

private:
    Node * root;
    int numWords;
    // Your private helper methods go here
};
```

Constructor

There are two versions of the constructor. The first just establishes an empty node. The second constructor takes in a string file and reads the file line by line and uses the addWord method to add words to the dictionary.

addWord

This method adds a word to the tree. This is accomplished by reading the word character by character and creating a path for the word if it doesn't exist. Below is the pseudocode for this method:

```
currNode = root;

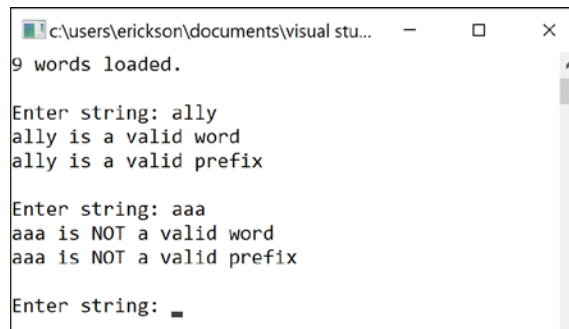
for (each character c of the word) {
    if (the branch for character c is NULL) {
        set the branch of character c = new Node.
        set flag of this new Node to false
    }
    currNode = the pointer index of character c
}
Set the flag at the currNode to true
```

isPrefix

This method returns true if a path down the branches exists for a word. This method will have an implementation very similar to the addWord method. It should return false if it runs into a NULL. If it manages to get through the entire loop without hitting a NULL, it should return true.

isWord

This method is very similar to the addWord and isPrefix method. If you have already implemented the isPrefix method, then it should be fairly trivial to implement isWord. The main difference is you should return the flag found at the end of the path.



```
c:\users\erickson\documents\visual stu...
9 words loaded.

Enter string: ally
ally is a valid word
ally is a valid prefix

Enter string: aaa
aaa is NOT a valid word
aaa is NOT a valid prefix

Enter string: 
```

Random Hints for the Dictionary Class

Be sure to test using the small dictionary before using the bigger dictionary. Here are some random hints that you may find useful.

Figuring out the index of a character

It will be useful to be able to index each character:

0	1	2	3	4	5	6	...	25
a	b	c	d	e	f	g	...	z

We can take advantage of the fact that characters can easily be cast into integers and use the following to figure out the index of a particular letter. For example, the following could be used to figure out the index of the character e:

```
(int)'e' - (int)'a'
```

The above would evaluate to the integer 4.