Q7. Implement a trie to accept all three letter strings. What is the cost to add an element in this structure? Estimate it.

We are assuming here that the strings would be made up of only small english alphabets. The structure used for representing a node of the trie data structure is as follows.

```
Node:
    isWordEnd: A boolean variable
    arr: an array of Node type
```

The isWordEnd field will contain True if a word(the 3 letter string) ends at that node otherwise False. And the arr field will contain an array of size 26 such that each cell is also a Node of the trie structure.

The node initialization, insertion, search procedures are as follows.

Node initialization procedure:

```
CREATE_NODE():
    node = Create a new Node
    node.isWordEnd = False
    FOR i = 1 to 26:
        node.arr[i] = NULL

    RETURN node
```

Node insertion procedure:

```
INSERT(root, word):
    temp = root

    FOR i = 1 to word.size():
        index = word[i] - 'a'
        IF temp.arr[index] == NULL:
            temp.arr[index] = CREATE_NODE()

        temp = temp.arr[index]

    temp.isWordEnd = True
```

Node search procedure:

```
SEARCH(root, word):
    temp = root

    FOR i = 1 to word.size():
        index = word[i] - 'a'
        IF temp.arr[index] == NULL:
            RETURN False

        temp = temp.arr[index]

    RETURN temp.isWordEnd
```

The pseudo code to insert all three letter string is as follows.

```
root = CREATE_NODE()

FOR i = 'a' to 'z':
    FOR j = 'a' to 'z':
        FOR z = 'a' to 'z':
            insert(root, i + j + z)
```

Time Complexity:

Since, we are only inserting strings of length 3, the depth of trie will only be 3. Hence, at most we need to do 3 comparisons always. Therefore, the time complexity will be constant.

Its C++ implementation is as follows.

```
#include<bits/stdc++.h>
using namespace std;

const int SIZE = 26;

struct node{
    bool endOfWord;
    node* ar[SIZE];
};

node* getNode()
{
    node* n = new node;
    n->endOfWord = false;

    for(int i=0;i<SIZE;i++)
    n->ar[i] = NULL;

    return n;
}

void insert(node *root , string st)
```

```
{
    node *tempRoot = root;

    for(int i=0;i<st.size();i++)
    {
        int index = st[i] - 'a';

        if(tempRoot->ar[index] == NULL)
        tempRoot->ar[index] = getNode();

        tempRoot = tempRoot->ar[index];
    }

    tempRoot->endOfWord = true;
}

bool search(node *root , string st)
{
    node *tempRoot = root;

    for(int i=0;i<st.size();i++)
    {
        int index = st[i] - 'a';

        if(tempRoot->ar[index] == NULL)
        return false;

        tempRoot = tempRoot->ar[index];
    }

    return tempRoot->endOfWord;
}

int main()
{
    node *root = getNode();

    for (int i = 0; i < 26; i++) {
        for (int j = 0; j < 26; j++) {
            for (int z = 0; z < 26; z++) {
                string s = string(1, char(i + 'a')) + string(1, char(j + 'a')) +
string(1, char(z + 'a'));
                insert(root, s);
            }
        }
    }
}
```