Leetcode May Challenge DAY: 14

1. Python

```
class Trie:


    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.child = {}



    def insert(self, word: str) -> None:
        """
        Inserts a word into the trie.
        """
        current = self.child
        for l in word:
            if l not in current:
                current[l] = {}
            current = current[l]
        current['#']=1



    def search(self, word: str) -> bool:
        """
        Returns if the word is in the trie.
        """
        current = self.child
```

```
        for l in word:
            if l not in current:
                return False
            current = current[l]
        return '#' in current


    def startsWith(self, prefix: str) -> bool:
        """
        Returns if there is any word in the trie that starts with the given prefix.
        """
        current = self.child
        for l in prefix:
            if l not in current:
                return False
            current = current[l]
        return True


# Your Trie object will be instantiated and called as such:
# obj = Trie()
# obj.insert(word)
# param_2 = obj.search(word)
# param_3 = obj.startsWith(prefix)
```

2. C++

```
struct Node{
    char x;
    bool isTerminal;
    map<char, Node*> m;
};

class Trie {
    Node* root;
public:
    /** Initialize your data structure here. */
    Trie() {
        root = new Node;
        root->isTerminal = false;
        // Any Dummy Value is Fine
        root->x = 'X';
    }

    /** Inserts a word into the trie. */
    void insert(string word) {
        Node* t = root;
        int i=0;
        for(; i<word.size(); i++){
            if((t->m).count(word[i])==0){
                break;
            }else{
                t = (t->m)[word[i]];
            }
```

```
        }
        for(; i<word.size();i++){
            Node* temp = new Node;
            temp->x = word[i];
            temp->isTerminal = false;
            (t->m)[word[i]] = temp;
            t = temp;
        }
        t->isTerminal = true;

    }

    /** Returns if the word is in the trie. */
    bool search(string word) {
        Node* temp = root;
        for(int i=0;i<word.size();i++){
            if((temp->m).count(word[i])==0){
                return false;
            }else{
                temp = (temp->m)[word[i]];
            }
        }

        return temp->isTerminal;
    }

    /** Returns if there is any word in the trie that starts with the given prefix. */
    bool startsWith(string word) {
```

```
        Node* temp = root;
        for(int i=0;i<word.size();i++){
            if((temp->m).count(word[i])==0){
                return false;
            }else{
                temp = (temp->m)[word[i]];
            }
        }

        return true;
    }
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```

3. JAVA

```
class Trie {


   class TrieNode{
      public boolean isEnd;
      public TrieNode[] next;

      public TrieNode(){
         this.isEnd = false;
         this.next = new TrieNode[26];
      }
   }

   private TrieNode root;

   /** Initialize your data structure here. */
   public Trie() {
      this.root = new TrieNode();
   }

   /** Inserts a word into the trie. */
   public void insert(String word) {
      if(word != null){
         int i, N = word.length();
         char ch;
         TrieNode current = this.root;
         for(i = 0; i < N; i++){
            ch = word.charAt(i);
```

```
            if(current.next[ch - 'a'] == null){
                current.next[ch - 'a'] = new TrieNode();
            }

            current = current.next[ch - 'a'];
        }

        current.isEnd = true;
    }
}

/** Returns if the word is in the trie. */
public boolean search(String word) {
    if(word == null || word.length() == 0)
        return true;
    else{
        int i, N = word.length();
        char ch;
        TrieNode current = root;

        for(i = 0; i < N; i++){
            ch = word.charAt(i);
            if(current.next[ch - 'a'] == null){
                return false;
            }

            current = current.next[ch - 'a'];
        }
```

```
        return current.isEnd;
    }
  }

  /** Returns if there is any word in the trie that starts with the given prefix. */
  public boolean startsWith(String prefix) {
    if(prefix == null || prefix.length() == 0)
      return true;
    else{
      int i, N = prefix.length();
      char ch;
      TrieNode current = root;

      for(i = 0; i < N; i++){
        ch = prefix.charAt(i);
        if(current.next[ch - 'a'] == null){
          return false;
        }

        current = current.next[ch - 'a'];
      }

      return true;
    }
  }
}
```

```
/**
 * Your Trie object will be instantiated and called as such:
 * Trie obj = new Trie();
 * obj.insert(word);
 * boolean param_2 = obj.search(word);
 * boolean param_3 = obj.startsWith(prefix);
 */
```