DSA SERIES

- Learn Coding

Topic to be Covered today

Trie

208. Implement Trie (Prefix Tree)

```
struct Node {
    Node* links[26];

    bool flag = false;

    bool containsKey(char ch) { return links[ch - 'a'] != NULL; }

    void put(char ch, Node* node) { links[ch - 'a'] = node; }

    Node* get(char ch) { return links[ch - 'a']; }

    void setEnd() { flag = true; }

    bool isEnd() { return flag; }
```

```
class Trie {
public:
    Node* root;
    Trie() {
        root = new Node();
    }


    void insert(string word) {
        Node* node = root;


        for (char ch : word) {
            if (!node->containsKey(ch)) {
                node->put(ch, new Node());
            }
            node = node->get(ch);
        }
        node->setEnd();
    }
```

```
    bool search(string word) {
        Node* node = root;

        for (char ch : word) {
            if (!node->containsKey(ch)) {
                return false;
            }
            node = node->get(ch);
        }
        return node->isEnd();
    }

    bool startsWith(string prefix) {
        Node* node = root;

        for (char ch : prefix) {
            if (!node->containsKey(ch)) {
                return false;
            }
            node = node->get(ch);
        }
        return true;
    }
};
```

211. Design Add and Search Words Data Structure

```
struct Node {
    Node* links[26];

    bool isEnd = false;

    bool containsKey(char ch) { return links[ch - 'a']!=NULL; }

    void put(char ch, Node* node) { links[ch - 'a'] = node; }

    Node* get(char ch) { return links[ch - 'a']; }
};

bool dfs(int index, string& word, Node* node) {

    if (index == word.size()) {
        return node->isEnd;
    }

    char ch = word[index];

    if (ch != '.') {
        if (!node->containsKey(ch))
            return false;
```

```
        return dfs(index + 1, word, node->get(ch));
    }


    for (int i = 0; i < 26; i++) {
        if (node->links[i] != NULL) {
            if (dfs(index + 1, word, node->links[i])) {
                return true;
            }
        }
    }
    return false;
}


class WordDictionary {
public:
    Node* root;
    WordDictionary() { root = new Node(); }


    void addWord(string word) {
        Node* node = root;
```

```
    for (char ch : word) {
            if (!node->containsKey(ch)) {
                node->put(ch, new Node());
            }


            node = node->get(ch);
        }
        node->isEnd = true;
    }


    bool search(string word) {
        return dfs(0,word,root);
    }
};
```

Learn coding

THANK YOU