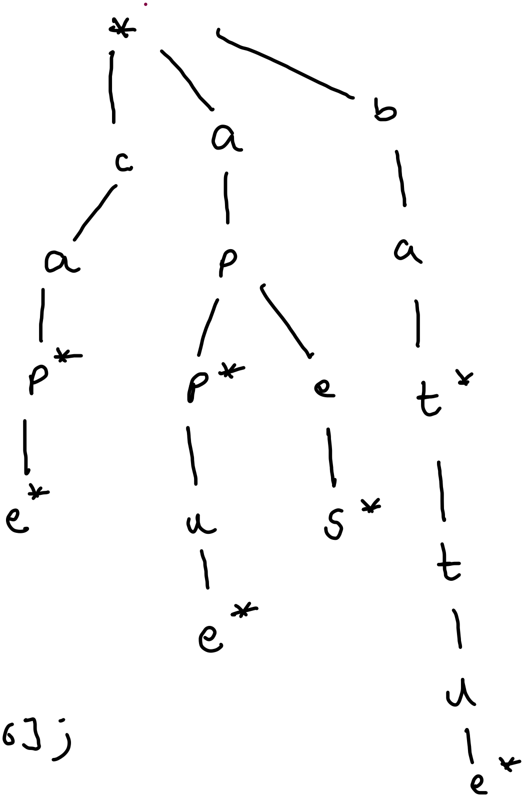
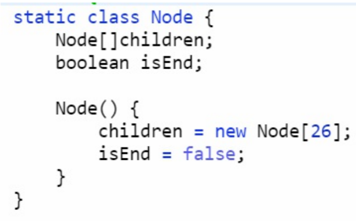
Use ?

cape, cap
apple , bat
app , battle
apes

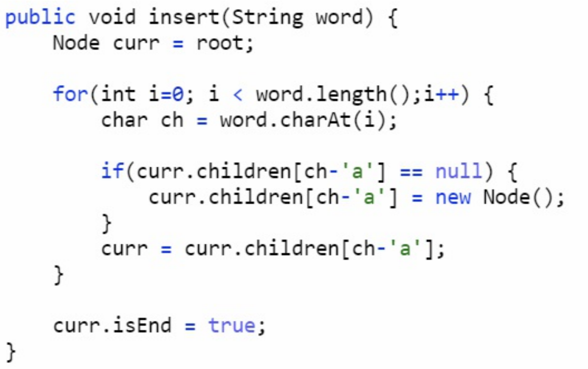


add (" bat ")
add (" battle")
add (" cape ")
add (" cap ")
add ("apple")
add ("app")
add (" apes")
search (" bat") -> false
search ("apple") -> true
startswith (" app") -> true

```
Node {

  Node [] children = new Node[26];
  boolean isEnd = false;

}
```



```java
static class Node {
    Node[]children;
    boolean isEnd;

    Node() {
        children = new Node[26];
        isEnd = false;
    }
}
```

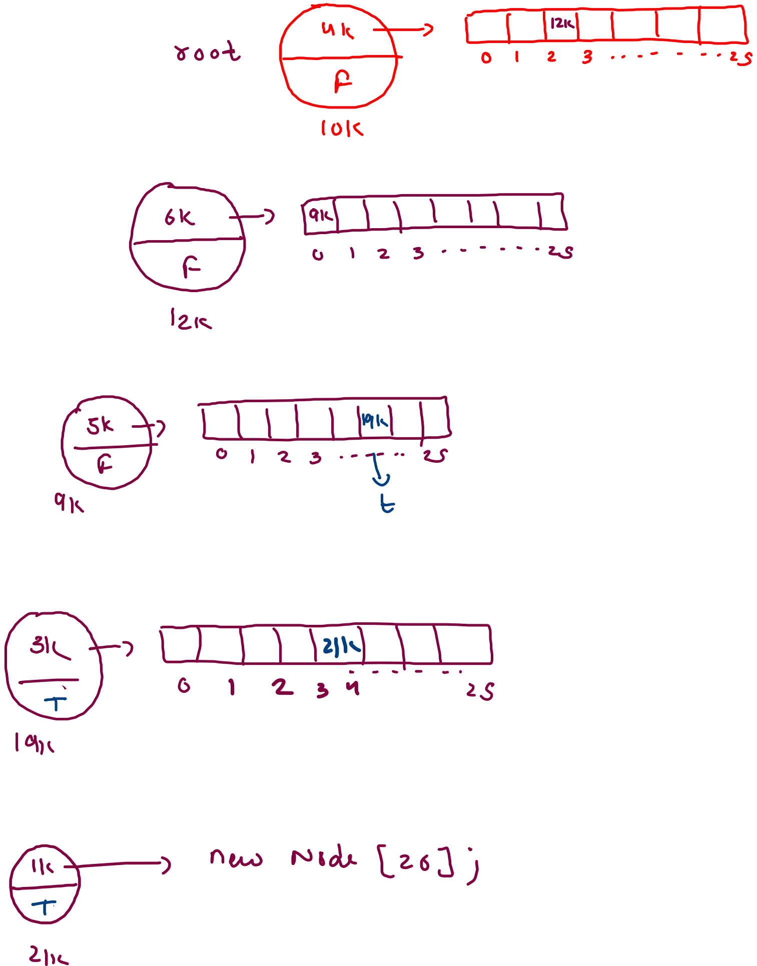


```java
public void insert(String word) {
    Node curr = root;

    for(int i=0; i < word.length();i++) {
        char ch = word.charAt(i);

        if(curr.children[ch-'a'] == null) {
            curr.children[ch-'a'] = new Node();
        }
        curr = curr.children[ch-'a'];
    }

    curr.isEnd = true;
}
```

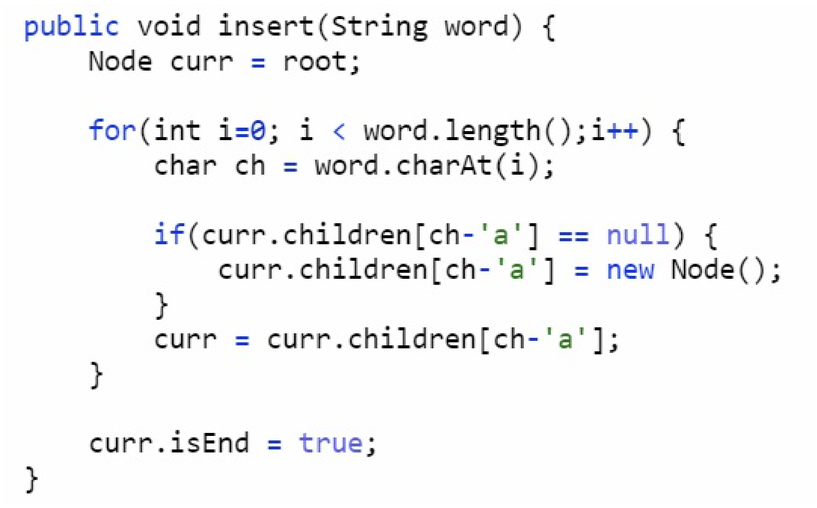
memory ,

insert (cat)

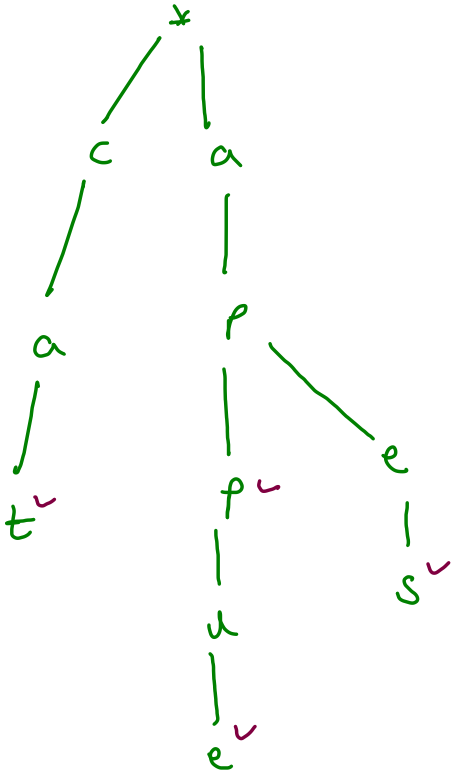
insert (cate)

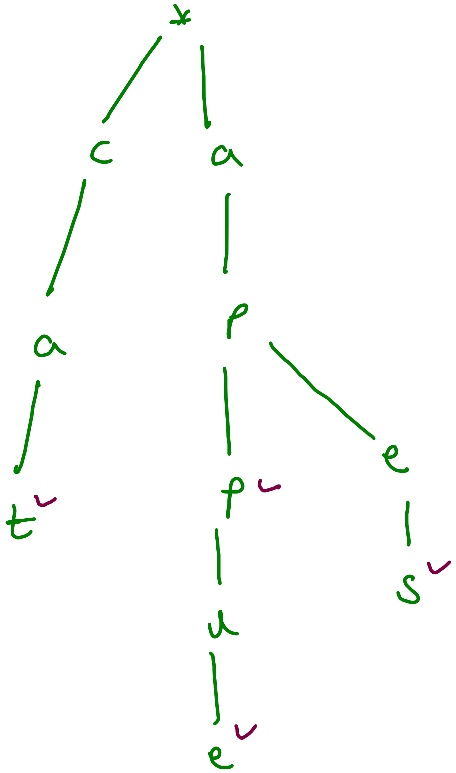


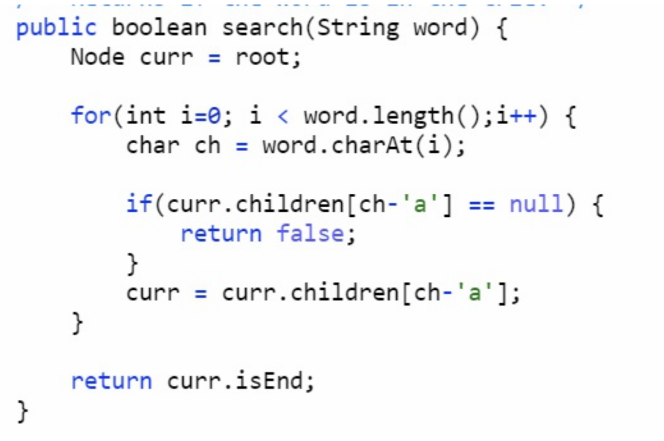
cat, apple, apes, app



```
public void insert(String word) {
    Node curr = root;

    for(int i=0; i < word.length();i++) {
        char ch = word.charAt(i);

        if(curr.children[ch-'a'] == null) {
            curr.children[ch-'a'] = new Node();
        }
        curr = curr.children[ch-'a'];
    }

    curr.isEnd = true;
}
```

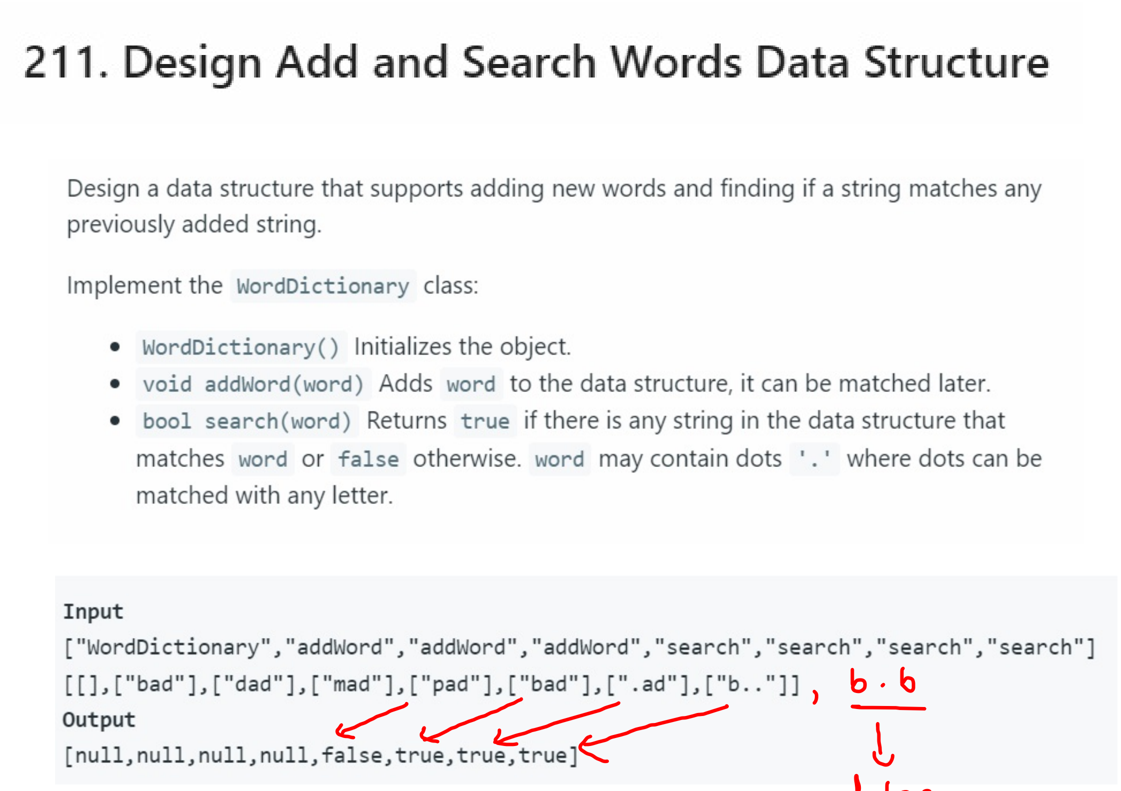






```
public boolean search(String word) {
    Node curr = root;

    for(int i=0; i < word.length();i++) {
        char ch = word.charAt(i);

        if(curr.children[ch-'a'] == null) {
            return false;
        }
        curr = curr.children[ch-'a'];
    }

    return curr.isEnd;
}
```

word: app, ape, cap



211. Design Add and Search Words Data Structure

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the `WordDictionary` class:

- `WordDictionary()` Initializes the object.
- `void addWord(word)` Adds `word` to the data structure, it can be matched later.
- `bool search(word)` Returns `true` if there is any string in the data structure that matches `word` or `false` otherwise. `word` may contain dots `'.'` where dots can be matched with any letter.

```
Input
["WordDictionary","addWord","addWord","addWord","search","search","search","search"]
[[],["bad"],["dad"],["mad"],["pad"],["bad"],[".ad"],["b.."]]
Output
[null,null,null,null,false,true,true,true]
```

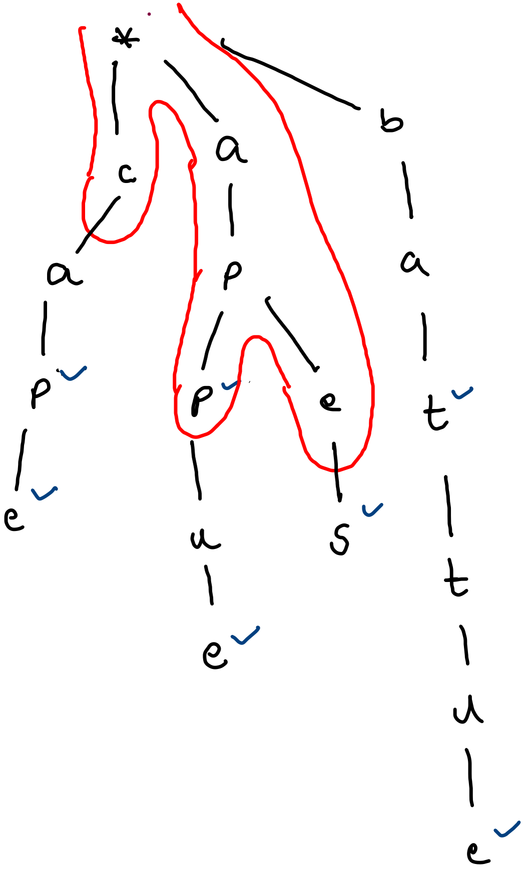
, b.b

↓

false

bad, mad

dad



cap, cape, app, apple, apes, bat, battle.

bat → true

a.p.e → true

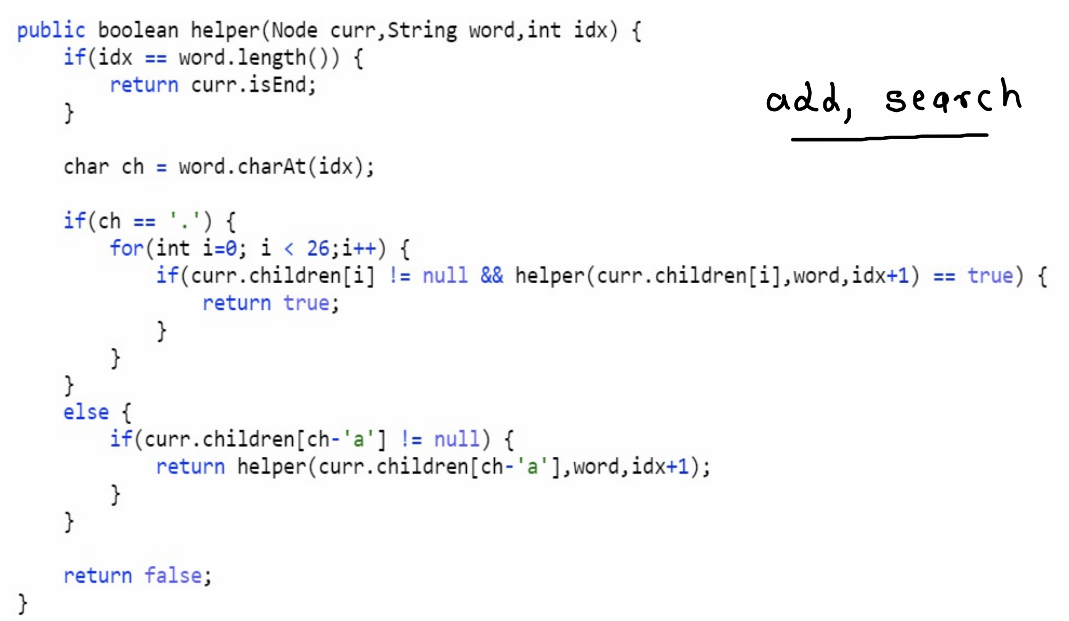
a.p. → false

c.p. → true

c.. → true

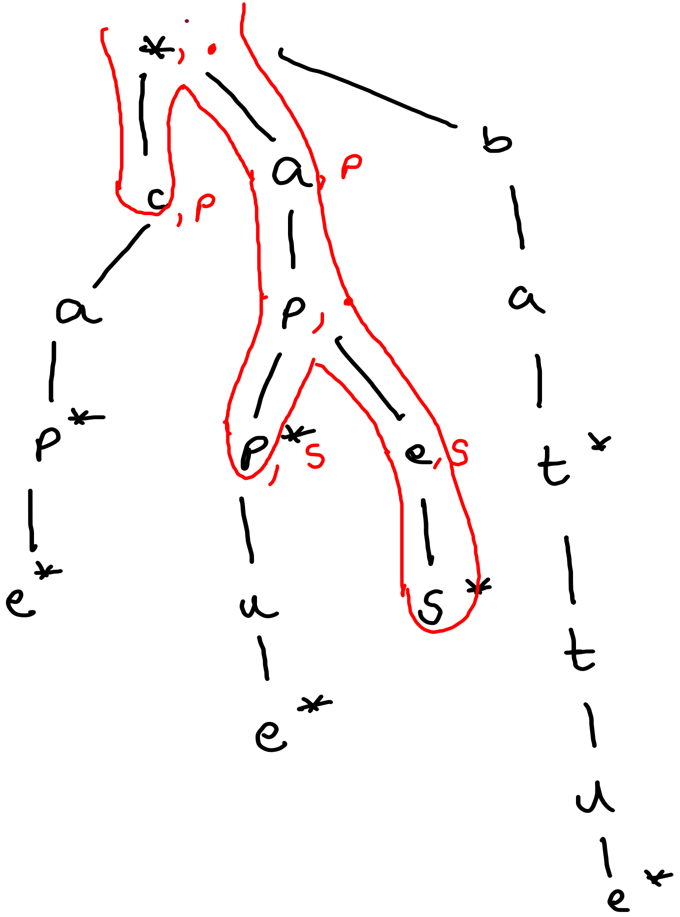
... → true

.. → false



```
public boolean helper(Node curr,String word,int idx) {
    if(idx == word.length()) {
        return curr.isEnd;
    }

    char ch = word.charAt(idx);

    if(ch == '.') {
        for(int i=0; i < 26;i++) {
            if(curr.children[i] != null && helper(curr.children[i],word,idx+1) == true) {
                return true;
            }
        }
    }
    else {
        if(curr.children[ch-'a'] != null) {
            return helper(curr.children[ch-'a'],word,idx+1);
        }
    }

    return false;
}
```

add, search



.p.s

h.w → print all words in trie