# Trie (Prefix Tree)

```
class TrieNode:
    def __init__(self):
        self.children = collections.defaultdict(TrieNode)
        self.is_word = False
        self.word = ''

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word):
        p = self.root
        for c in word:
            p = p.children[c]
        p.is_word = True
        p.word = word

    def search(self, word):
        p = self.root
        for c in word:
            if c in p.children: p = p.children[c]
            else: return False
        return p.is_word

    def startsWith(self, prefix):
        p = self.root
        for c in prefix:
            if c in p.children: p = p.children[c]
            else: return False
        return True
    # def bfs(self):
    # def dfs(self):
```

Implement Trie (Prefix Tree) `trie`
Add and Search Word - Data structure design `trie`
Map Sum Pairs (insert("apple", 3); insert("app", 2); sum("ap")) `delta update`
Implement Magic Dictionary `trie` `mismatch:=1`

Replace Words `trie` `first word in search`
Longest Word in Dictionary `self.is_word, self.word` `bfs` `word_path`
Longest Word in Dictionary through Deleting `heap(word), isSubsequence?`

## hard，还没做

**Word Search II**
**Concatenated Words**
**Palindrome Pairs**
**Word Squares**
**Prefix and Suffix Search**
**Design Search Autocomplete System**
**Maximum XOR of Two Numbers in an Array**