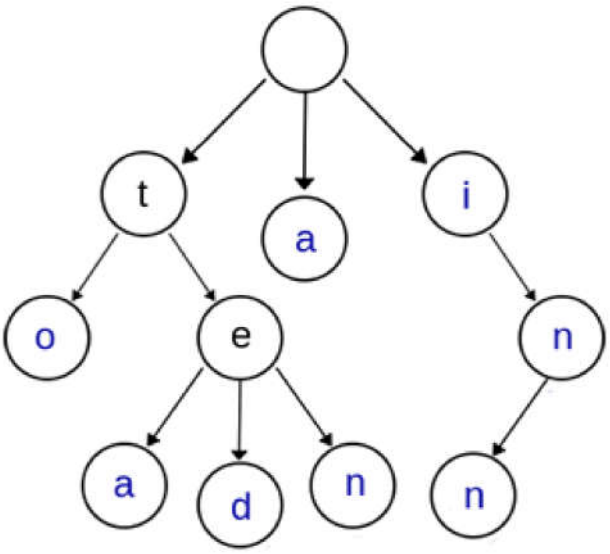
208. 实现 Trie (前缀树)

> 题目：Implement Trie (Prefix Tree)
> 语言: python3
> 英文版链接：https://leetcode.com/problems/implement-trie-prefix-tree/description/
> 中文版链接：https://leetcode-cn.com/problems/implement-trie-prefix-tree/

题目分析

Trie树，又叫字典树、前缀树（Prefix Tree）、单词查找树 或 键树，是一种多叉树结构。如下图：



上图是一棵Trie树，表示了关键字集合{“a”, “to”, “tea”, “ted”, “ten”, “i”, “in”, “inn”} 。从上图可以归纳出Trie树的基本性质：

1. 根节点不包含字符，除根节点外的每一个子节点都包含一个字符。
2. 从根节点到某一个节点，路径上经过的字符连接起来，为该节点对应的字符串。
3. 每个节点的所有子节点包含的字符互不相同。

通常在实现的时候，会在节点结构中设置一个标志，用来标记该结点处是否构成一个单词（关键字）。

可以看出，Trie树的关键字一般都是字符串，而且Trie树把每个关键字保存在一条路径上，而不是一个结点中。另外，两个有公共前缀的关键字，在Trie树中前缀部分的路径相同，所以Trie树又叫做前缀树（Prefix Tree）。

优点

1. 插入和查询的效率很高，都为$O(m)$，其中 m 是待插入/查询的字符串的长度。
2. 关于查询，会有人说 hash 表时间复杂度是$O(1)$不是更快？但是，哈希搜索的效率通常取决于

- hash 函数的好坏，若一个坏的 hash 函数导致很多的冲突，效率并不一定比Trie树高。
- Trie树中不同的关键字不会产生冲突。

3. Trie树只有在允许一个关键字关联多个值的情况下才有类似hash碰撞发生。
4. Trie树不用求 hash 值，对短字符串有更快的速度。通常，求hash值也是需要遍历字符串的。
5. Trie树可以对关键字按字典序排序。

缺点

1. 当 hash 函数很好时，Trie树的查找效率会低于哈希搜索。
2. 空间消耗比较大。

应用

- 字符串检索
- 词频统计
- 字符串排序
- 前缀匹配
- 作为其他数据结构和算法的辅助结构，如后缀树，AC自动机等。

本题解析

作为本体来说，还是很简单的，只是构建一颗简单的Trie树，其中有三个操作，分别是insert、search、startWith，insert很好理解，search是查询输入的单词是不是插入Trie树过，startWith是看看有没有相同的前缀，比如插入了apple，那么app就是它的一个前缀。

我们为了直观展示，申请一颗Node类，作为一个辅助类，这样虽然耗费空间大一点，但是更加直观。

```
class Node:
    def __init__(self):
        self.children = {}
        self.is_leaf = False
```

答案

```python
class Trie:

    def __init__(self):
        self.root = Node()

    def insert(self, word: str) -> None:
        node = self.root
        for c in word:
            if c not in node.children:
                node.children[c] = Node()
            node = node.children[c]
        node.is_leaf = True

    def search(self, word: str) -> bool:
        node = self.root
        for c in word:
            if c not in node.children:
                return False
            node = node.children[c]
        return node.is_leaf

    def startsWith(self, prefix: str) -> bool:
        node = self.root
        for c in prefix:
            if c not in node.children:
                return False
            node = node.children[c]
        return True
```