0208. 实现 Trie (前缀树)

ITCharge 大约 3 分钟

- 标签：设计、字典树、哈希表、字符串
- 难度：中等

题目链接

- 0208. 实现 Trie (前缀树) - 力扣

题目大意

要求：实现前缀树数据结构的相关类 `Trie` 类。

`Trie` 类：

- `Trie()` 初始化前缀树对象。
- `void insert(String word)` 向前缀树中插入字符串 `word` 。
- `boolean search(String word)` 如果字符串 `word` 在前缀树中，返回 `True` （即，在检索之前已经插入）；否则，返回 `False` 。
- `boolean startsWith(String prefix)` 如果之前已经插入的字符串 `word` 的前缀之一为 `prefix` ，返回 `True` ；否则，返回 `False` 。

说明：

- $1 \leq word.length, prefix.length \leq 2000$。
- `word` 和 `prefix` 仅由小写英文字母组成。
- `insert` 、 `search` 和 `startsWith` 调用次数 **总计** 不超过 $3 * 10^4$ 次。

示例：

- 示例 1：

```py
输入：
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
输出：
[null, null, true, false, true, null, true]
```

```
解释：
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 True
trie.search("app");     // 返回 False
trie.startsWith("app"); // 返回 True
trie.insert("app");
trie.search("app");     // 返回 True
```

解题思路

思路 1：前缀树（字典树）

前缀树（字典树）是一棵多叉树，其中每个节点包含指向子节点的指针数组 `children` ，以及布尔变量 `isEnd` 。 `children` 用于存储当前字符节点，一般长度为所含字符种类个数，也可以使用哈希表代替指针数组。 `isEnd` 用于判断该节点是否为字符串的结尾。

下面依次讲解插入、查找前缀的具体步骤：

插入字符串：

- 从根节点开始插入字符串。对于待插入的字符，有两种情况：
  - 如果该字符对应的节点存在，则沿着指针移动到子节点，继续处理下一个字符。
  - 如果该字符对应的节点不存在，则创建一个新的节点，保存在 `children` 中对应位置上，然后沿着指针移动到子节点，继续处理下一个字符。
- 重复上述步骤，直到最后一个字符，然后将该节点标记为字符串的结尾。

查找前缀：

- 从根节点开始查找前缀，对于待查找的字符，有两种情况：
  - 如果该字符对应的节点存在，则沿着指针移动到子节点，继续查找下一个字符。
  - 如果该字符对应的节点不存在，则说明字典树中不包含该前缀，直接返回空指针。
- 重复上述步骤，直到最后一个字符搜索完毕，则说明字典树中存在该前缀。

思路 1：代码

```py
class Node:
    def __init__(self):
        self.children = dict()
        self.isEnd = False

class Trie:

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

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

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

        return cur is not None and cur.isEnd

    def startsWith(self, prefix: str) -> bool:
        cur = self.root
        for ch in prefix:
            if ch not in cur.children:
                return False
            cur = cur.children[ch]
        return cur is not None
```

思路 1：复杂度分析

- **时间复杂度**：初始化为 $O(1)$。插入操作、查找操作的时间复杂度为 $O(|S|)$。其中 $|S|$ 是每次插入或查找字符串的长度。
- **空间复杂度**：$O(|T| \times \sum)$。其中 $|T|$ 是所有插入字符串的长度之和，$\sum$ 是字符集的大小。