Data Structures

Trie Homework 1

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Problem #1: Iterative version

- **def** insert(self, str):
- **def** word_exist(self, str):
- **def** prefix_exist(self, str):
- The recursive version of the lecture is slow when we have a lot of queries
- Rewrite an iterative version of these Trie functions
- *Tip: for all exercises, assume the trie is based on 26 lowercase letters, unless otherwise stated*

Problem #2: Is suffix

- Write a trie that contains the following main 2 methods:
  - **def** insert(self, str)
  - **def** suffix_exist(self, str)
- suffix_exist returns true if any inserted word has such a suffix

```
root = Trie()

root.insert("abcd")
root.insert("xyz")

print(root.suffix_exist("abc"))     # False
print(root.suffix_exist("bcd"))     # True
print(root.suffix_exist("xyz"))     # True
print(root.suffix_exist("xy"))      # False
print(root.suffix_exist("yz"))      # True
print(root.suffix_exist("xyzxyz"))  # False
```

Problem #3: minimal prefix

- **def** first_word_prefix(self, str):
- Given a string, find the smallest **full word** in a trie that is a prefix for the given str. If there is no trie word, return None
- This is a sub-problem from the linked Leetcode problem

```
root.insert("xyz")
root.insert("xyzwfe")

print(root.first_word_prefix("xy"))     # None
print(root.first_word_prefix("xyz"))    # xyz
print(root.first_word_prefix("xyzw"))   # xyz
print(root.first_word_prefix("xyzH"))   # xyz

root.insert("x")
print(root.first_word_prefix("xy"))     # x
print(root.first_word_prefix("xyz"))    # x
```

Problem #4: OS Paths

- Assume for simplicity a system path represented as a list of strings
  - /home/software/eclipse/bin ⇒ ["home", "software", "eclipse", "bin"]
- Design a trie
  - def insert(self, path_lst) ⇒ adds a path to the trie
  - def subpath_exist(path_lst)
    - Return True if such a sub-path exists
    - If a path is [W1 W2 W3] then the following are sub-paths
      - W1
      - W1 W2
      - W1 W2 W3
      - Similar to a prefix but on complete words

Problem #4: OS Paths

```
root = Trie()

root.insert(["home", "software", "eclipse"])
root.insert(["home", "software", "eclipse", "bin"])
root.insert(["home", "installed", "gnu"])
root.insert(["user", "mostafa", "tmp"])

print(root.subpath_exist(["user", "mostafa", "tmp"]))   # True
print(root.subpath_exist(["user", "mostafa"]))          # True
print(root.subpath_exist(["user"]))                     # True
print(root.subpath_exist(["user", "most"]))             # False
print(root.subpath_exist(["user", "NOT"]))              # False
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."