▾ Lab#2, NLP@CGU Spring 2023

This is due on 2023/03/13 15:30, commit to your github as a PDF (lab2.pdf) (File>Print>Save as PDF).

IMPORTANT: After copying this notebook to your Google Drive, please paste a link to it below. To get a publicly-accessible link, hit the *Share* button at the top right, then click "Get shareable link" and copy over the result. If you fail to do this, you will receive no credit for this lab!

LINK: paste your link here

https://colab.research.google.com/drive/1javTrABQkeIldC3vuZ6VcTBCHaF0Q4VN?usp=sharing

---

Student ID:B0944104

Name:林冠霆

▾ Question 1 (100 points)

Implementing Trie in Python.

Trie is a very useful data structure. It is commonly used to represent a dictionary for looking up words in a vocabulary.

For example, consider the task of implementing a search bar with auto-completion or query suggestion. When the user enters a query, the search bar will automatically suggests common queries starting with the characters input by the user.

按兩下 (或按 Enter 鍵) 即可編輯

```
#  YOUR  CODE  HERE!
#  IMPLEMENTIG  TRIE  IN  PYTHON

class  TrieNode:

    def  __init__(self,  char:str):
        #  the  character  stored  in  this  node
        self.char  =  char

        #  whether  this  can  be  the  end  of  a  word
        self.is_end  =  False

        #  a  counter  indicating  how  many  times  a  word  is  inserted
        #  (if  this  node's  is_end  is  True)
        self.counter  =  0

        #  a  dictionary  of  child  nodes
        #  keys  are  characters,  values  are  nodes
        self.children  =  {}
class  Trie(object):

    def  __init__(self):
        self.root  =  TrieNode("")

    #  def  insert(self,  word):
    def  insert(self,  word):
        node  =  self.root
        for  char  in  word:
            if  char  in  node.children:
                node  =  node.children[char]
            else:
                #  If  a  character  is  not  found
```

```python
                    # If a character is not found,
                    # create a new node in the trie
                    new_node = TrieNode(char)
                    node.children[char] = new_node
                    node = new_node

            # Mark the end of a word
            node.is_end = True

            # Increment the counter to indicate that we see this word once more
            node.counter += 1

    # def dfs(self, node, prefix):
    def dfs(self, node, prefix):
            if node.is_end:
                    self.output.append((prefix + node.char, node.counter))

            for child in node.children.values():
                    self.dfs(child, prefix + node.char)
    # def query(self, x):
    def query(self, word):
            self.output = []
            node = self.root

            # Check if the prefix is in the trie
            for char in word:
                    if char in node.children:
                            node = node.children[char]
                    else:
                            # cannot found the prefix, return empty list
                            return []

            # Traverse the trie to get all candidates
            self.dfs(node, word[:-1])

            # Sort the results in reverse order and return
            return sorted(self.output, key=lambda word: word[1], reverse=True)
# # DO NOT MODIFY THE VARIABLES
obj = Trie()
obj.insert("長庚資工")
obj.insert("長大")
obj.insert("長庚")
obj.insert("長庚")
obj.insert("長庚大學")
obj.insert("長庚科技大學")

# # DO NOT MODIFY THE BELOW LINE!
# # THE RESULTS : [(words, count), (words, count)]
print(obj.query("長"))
# [('長庚', 2), ('長庚資工', 1), ('長庚大學', 1), ('長庚科技大學', 1), ('長大', 1)]

print(obj.query("長庚"))
# [('長庚', 2), ('長庚資工', 1), ('長庚大學', 1), ('長庚科技大學', 1)]
```

```
[('長庚', 2), ('長庚資工', 1), ('長庚大學', 1), ('長庚科技大學', 1), ('長大', 1)]
[('長庚', 2), ('長庚資工', 1), ('長庚大學', 1), ('長庚科技大學', 1)]
```