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/1wPL3FESx1__JbRM-64dBxQRpAVcCYJSp?usp=sharing

---

Student ID: B0928007

Name: 余明昌

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):
        self.char = char
        self.children = set()
        self.count = 0

class Trie(object):

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

    def insert(self, word):
        current = self.root
```

```
        current = self.root
        for i, w in enumerate(word):
            flag = True
            for node in current.children:
                if (w == node.char):
                    current = node
                    flag = False
                    if (i == len(word) - 1):
                        current.count += 1
                    break

            if (flag):
                for w in word[i: ]:
                    temp = TrieNode(w)
                    current.children.add(temp)
                    current = temp
                current.count += 1
                return

    def dfs(self, node, prefix):
        result = []
        if (node.count != 0):
            result.append((prefix, node.count))

        for n in node.children:
            result += self.dfs(n, prefix + n.char)

        return result

    def query(self, x):
        node = self.root
        temp = ""

        for c in x:
            for n in node.children:
                if (n.char == c):
                    node = n

        result = self.dfs(node, x)
        result = sorted(result, key = lambda x: x[1], reverse = True)

        return result



# # 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)]
```