CGU-NLP-LAB2

March 13, 2023

0.1 Lab#2, NLP@CGU Spring 2023

0.1.1 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/16DJb2oxkhdZgvSlK–kz7noZ8S51qo4t?usp=sharing —

Student ID: B0928010

Name:

##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.

[ ]:
```
# YOUR CODE HERE!
# IMPLEMENTIG TRIE IN PYTHON

class TrieNode:

    def __init__(self, char):
        self.char = char
        self.children = []
        self.finished = False
        self.counter = 0

    def __eq__(self, char):
        return self.char == char
```

```
class Trie(object):

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

    def insert(self, word):
        current = self.root
        for char in word:
            if char in current.children:
                current = current.children[current.children.index(char)]
            else:
                current.children.append(TrieNode(char))
                current = current.children[current.children.index(char)]
        current.finished = True
        current.counter += 1

    def dfs(self, node, prefix):
        results = []
        prefix += node.char
        if node.finished:
            results.append((prefix, node.counter))
        for child in node.children:
            results += self.dfs(child, prefix)
        return results

    def query(self, x):
        results = []
        current = self.root
        current_str = ""
        for char in x:
            if char not in current.children:
                return None
            else:
                current = current.children[current.children.index(char)]
                current_str += char
        if current.finished:
            results.append((current_str, current.counter))
        for child in current.children:
            results += self.dfs(child, current_str)
        return results



# # DO NOT MODIFY THE VARIABLES
obj = Trie()
obj.insert("  ")
```

```
obj.insert(" ")
obj.insert(" ")
obj.insert(" ")
obj.insert("  ")
obj.insert("   ")

# # DO NOT MODIFY THE BELOW LINE!
# # THE RESULTS : [(words, count), (words, count)]
print(obj.query(" "))
# [(' ', 2), ('  ', 1), ('  ', 1), ('   ', 1), (' ', 1)]

print(obj.query(" "))
# [(' ', 2), ('  ', 1), ('  ', 1), ('   ', 1)]
```

```
[(' ', 2), ('  ', 1), ('  ', 1), ('   ', 1), (' ', 1)]
[(' ', 2), ('  ', 1), ('  ', 1), ('   ', 1)]
```