Victor Mai || DC127853 || Brief report of Assignment #1
2024/2025 CISC3025 Natural Language Processing

I. Introduction:

- The goal of my project:
  To implement a *sequence comparison algorithm* by using dynamic programming with Time Complexity of $O(n \times m)$. The name of algorithm is ***Levenshtein Distance.
- Explain why the project is both important and interesting in the context of NLP:
  The project is mainly about ***Levenshtein algorithm***, which is one of the Minimum Edit Search algorithm. It provides a way to quantify the ***difference between two strings*** (both words and sentences) by counting the minimum number of operations, that are insertions, deletions, or substitutions, needed to transform one string into another.

II. Background:

- Briefly introduce one or two fundamental NLP concepts that are central to your project:
  1. **Minimum Edit Distance**:
     The metric that is used to measure the distance between two strings by counting the smallest number of operations required to transform one string into the other.
  2. **Alignment**:
     In the output of my project, alignment helps identify which words or letters in a source string correspond to those in a target string with the help of ***Levenshtein algorithm***.

III. Approach & Challenges:

- Summarize your methodological approach in one concise paragraph. Identify one significant challenge you encountered and describe how you addressed it:

```
def sentence_preprocess(sentence:str) -> list[str]:
    words:list = sentence.split()
    no_sign_words:list = [
        re.sub(
            r"(?!^)[^\w\s\$\%\&\*\=\+\-\/'](?!$)",
            '',
            word
        )
        for word in words
    ]
    sentence:list = []
    for word in no_sign_words:
        if word != '':
            sentence.append(word)
    return sentence
```

- When running a project on sentences comparison, the input str of 2 sentences are supposed to be preprocessed before applying method sentence_edit_distance().
- The main Challenge is that there may exist various punctuation marks in the sentences, and some are needed (such as '$', '&') while others are not (such as '!', '?'). What's more, there may be to much space between two words, and there may exist Chinese characters.
- To solve this, I firstly try my best to gather the usage of all possible punctuation marks in daily life. Then I summarize whether to save the certain punctuation mark according to its context:

punctuation mark	save or not?	reason of saving
! or ?	no	
#	yes	# sometimes represents the word "number"
$	yes	$ represents the word "dollar"
%	yes	% represents the phrase "per sent"
&	yes	& always is the word "and"
( ) or { } or [ ]	no	
-	yes	phone number or word "minus"
_	yes	is the bridge of words
+	yes	+ mostly represent the word "plus"
=	yes	the word "equals (to)"
| or \	no	
: or ;	no	
"	no	
'	sometimes	it depends! When ' has at least one alpha beside it, it is saved! (e.g. I'm)
/	yes	/ mostly represent the word "per" (such as km/s, W/s)
< or >	no	
, or .	no	
~	no	

- As you can see, the reasons of saving the certain punctuation is mainly depend on whether it has meaning or it is non-negligible. I apply the regular expression in python to erase not-saving marks.

- In the method, I firstly split the sentence into list of words, then I erase the punctuation mark that is meaningless. After that, the empty words in the list are ignored and return the final list.

IV. Results:

- Summarize the outcomes of your project, highlighting the main findings:
  - The words comparison works stalely as it won't lose any marks. (including "." and " ' ")

```
won't
wont.
The cost is: 2
An possible alignment is:
w o n ' t -
| | | | | |
w o n - t .
```

  - When comparing two sentences, it can also perfectly perform the algorithm without losing the meaning of words and punctuation inside.

```
i'm victor_mai, my phone is 1223-23422-1 & 1-33
victor's name is mai. my phone is 1223-23422-1 and 1-33
The cost is: 8
An possible alignment is:
          -           -         i'm  victor_mai,          my       phone          is 1223-23422-1          &        1-33
          |           |          |           |           |          |           |           |          |           |
    victor's        name          is        mai.          my       phone          is 1223-23422-1        and        1-33
```

  - And my main finding is that the program prefer to use substitution rather than the combination of delete + insert, which is more reasonable if we consider "substitution" as "transformation".
  - Also, the process on word is similar to the process on sentence if we consider the letters in word as words in sentence.

V. Conclusion:

- Reflect briefly on what you learned from the project and what was accomplished:
  The Levenshtein algorithm is a very easy to understand and implement and is effective in measuring the Minimum Edit Distance between two strings!
  I've also found that when we're dealing with punctuation in words, we can use ***regular expressions*** by using "import re" in python to appropriately retain the needed symbols according to logic and presuppositions.