Logo

Description automatically generated **FACULTY OF COMPUTER SCIENCE AND ENGINEERING**

**Time: 1 hours AI 361 Lab Marks: 60**

**Instructor: Muhammad Muneeb Baig Lab #06**

**Here's a general approach:**

1. **Create a dictionary of all valid words in the language you want to check for spelling.**
2. **Get the text input from the user or load it from a file.**
3. **Split the text into individual words.**
4. **For each word, check if it exists in the dictionary. If it does, move to the next word. If it doesn't, move to the next step.**
5. **For each misspelled word, find the nearest words in the dictionary using a distance metric such as Levenshtein distance or Jaccard similarity. You can use libraries such as NLTK or spaCy for this purpose.**
6. **Display the misspelled word and its nearest word to the user.**

**Task 1 (10 Marks):**

[Lab06.txt](https://pern-my.sharepoint.com/:t:/g/personal/muneeb_baig_giki_edu_pk/EQ7yF4ARQHtJmKpoYgnd2jkBMURatPpamNQ8bRr61RKKGw?e=tdep27)

Write a parser that can read all the lines of the file Lab06.txt and print out for each line the original (misspelled) text, the corrected text, and the indexes of any changes. The indexes refer to the index of the words in the sentence. In the example given, there is only an error in the 10th word and so the list of indexes is [9]. It is not necessary to analyse where the error occurs inside the word.

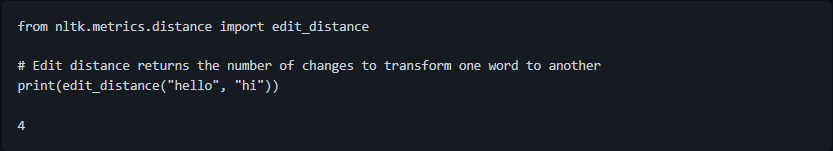
Then split your data into a test set of 100 lines and a training set.

**Task 2 (10 Marks):**

Calculate the frequency (number of occurrences), *ignoring case*, of all words and bigrams (sequences of two words) from the corrected *training* sentences:

**Task 3 (15 Marks):**

[Edit distance](https://en.wikipedia.org/wiki/Edit_distance) is a method that calculates how similar two strings are to one another by counting the minimum number of operations required to transform one string into the other. Also check your results with **Levenshtein distance and Jaccard similarity**. Plot the graphs that shows the difference between results of various approach. At the end also mention the best speller corrector based on your result. There is a built-in implementation in NLTK that works as follows:



Write a function that calculates all words with *minimal* edit distance to the misspelled word. You should do this as follows.

1. Collect the set of all unique tokens in train.
2. Find the minimal edit distance, that is the lowest value for the function edit\_distance between token and a word in train.
3. Output all unique words in train that have this same (minimal) edit\_distance value.

***Do not implement edit distance, use the built-in NLTK function edit\_distance***

**Task 4 (15 Marks):**

Write a function that takes a (misspelled) sentence and returns the corrected version of that sentence. The system should scan the sentence for words that are not in the dictionary (set of unique words in the training set) and for each word that is not in the dictionary choose a word in the dictionary that has minimal edit distance and has the highest *bigram probability*. That is the candidate should be selected using the previous and following word in a bigram language model. Thus, if the sixth word in a sentence is misspelled, we should use the following to rank candidates:

**p(wi+1|wi)p(wi|wi-1)**

For the first and last word of the sentence use only the conditional probabilities that exist.

**Task 5 (10 Marks):**

Using the test corpus evaluate the *accuracy* of your method, i.e., how many words from your system's output match the corrected sentence (you should count words that are already spelled correctly and not changed by the system).

**GOOD LUCK!**