Day-1

==========

1. Submit a single python file which contains implementation of all the task functionalities — <Roll_No>_D1.py
2. The input to the program will come from a file containing source document — input.txt, and the final output has to be written in 'Summary_PR.txt' for T3 and 'Summary_MMR.txt' for T4.

==========

T1. Preprocessing and obtaining sentences using NLTK:

1. Segment the document into sentences (split on newline character), and store the sequence of sentences in a list. The first sentence in the document should be the first element of the list, the second sentence should be the second element of the list, and so on.
2. Preprocess each sentence (list element).
    a. Remove punctuation symbols (the method *translate* can be used).
    b. Convert all English letters in the sentence to lowercase.
    c. Tokenize the sentence into words.
    d. Remove a set of very common words (called stopwords). You can use the inbuilt list of stopwords provided by NLTK, that can be accessed by:
        from nltk.corpus import stopwords
        stop_words = set(stopwords.words('english'))
    e. Lemmatize, that is, reduce the different inflectional variants of a word to the dictionary form of the word (use *WordNetLemmatizer*)

At the end of this task, each element of the List should be a pre-processed sentence, in the same order as in the input document.

For example, consider a toy document containing two sentences:

I was walking down the street.
I met my friends Rayan, George and Seni.

After the task, the contents of the list should be:

List[0] = i walk down street
List[1] = i meet my friend rayan george seni

T2. Sentence Representation:

Represent a sentence as a vector of the TF-IDF weights of its constituent words. TF denotes term frequency, that is, how frequently a word appears in a document. IDF denotes inverse document frequency, which measures how important a word is.

1. Compute TF-IDF of each word w in a given sentence s.
    a. TF(w,s) = Number of times the word w occurs in the given sentence s.
    b. IDF(w) = log(Total number of sentences/Number of sentences with w in it).
    c. TF-IDF of the word w in s is TF(w,s)×IDF(w).

2. Construct a TF-IDF matrix for the whole document, where the rows are the words, and the columns are the sentences, and the (i, j)-th entry in the matrix denotes the TF-IDF value of Word i in Sentence j.

Thus, each column of the matrix is the representation of a sentence. This representation is called the TF-IDF vector of the sentence.

T3. Summarization — Using PageRank

Create an undirected, weighted graph where:

1. The vertices will be the sentences (represented by sentence ids / indices of the List you constructed in Part 1).
2. There will be an edge between all vertex pairs (complete graph).
3. The weight of the edge (u, v) between the nodes u and v will be the **cosine similarity** between the TF-IDF vectors of the sentences u and v (as computed in T2).

Compute PageRank of each node (sentence) in the graph. PageRank is a measure of the importance of a node in a graph. The underlying assumption is that more important nodes are likely to receive links from other important nodes.

Display the top-n nodes (sentences) having the highest PageRank values as the summary. Note that in the final output summary, the sentences must be arranged in the same order in which they appear in the input document.