Text Analyzer of Long Documents Mentioning People and their Stories

In this project, you will create a tool that analyzes how people are mentioned in a very long document. Imagine you have thousands of pages of diaries, interview transcripts, historical archives or other reports, and you need a clear and quick way to see how the people mentioned in those documents are connected and in what contexts they are discussed. You could be a journalist or analyst going through official reports to uncover how frequently certain names appear in connection with specific events. A text analysis tool that can quickly find, count, and connect mentions of different people could be very valuable.

With this project, you will build a basic text analysis tool. You will parse through long documents, identify recurring names, and surface possible direct and indirect relationships among the individuals mentioned. In future courses in your degree, you will learn more sophisticated ways to perform such analyses, but what you will learn in this project is already a useful start for text processing and basic analysis.

Base Inputs: (additional relevant inputs appear in each question)

The following inputs will be received as command line arguments:

1. A sentence file path. This file includes many sentences taken from a very long document mentioning people and their stories. This file will be in CSV format (you can read about the format here) with the following columns:
    a. First column with the header “sentence”, where each row contains a sentence.
2. A people file path. This file contains a list of main people to analyze. This file will be in CSV format with the following columns:
    a. First column with the header “Name” will be the main name of the character. For example - Harry Potter.
    b. Second column with header “Other Names” will contain nicknames or other ways the character is referred to, each separated by a comma “,”. For example - Boy Who lived,Dear boy,Pottah
3. A file path with a list of common words to remove. Words such as “the”, “it”, etc, this file will be in CSV format:
    a. First (and only) column will be words, each row will contain a single common word that you would need to remove during the text preprocessing.
4. A number indicating which output your program should produce, for example if the number 2 is received, we expect the output produced in task 2.
5. Other parameters will be supplied, as described in the following tasks.
6. **In case of invalid input, the message “invalid input” should be printed to the user.**

Note:

- In each step, the requested output should be **printed** in a JSON format. There is no requirement to create files or return specific values.
- For each task, we have provided sample input files and correct result json files for debugging purposes.
- While some tasks can be solved using the output of previous tasks, this is not necessarily true for all of them.
- Please note that the printed output should be sorted.
- You are **not** permitted to work with the following packages: networkx, scikit-learn, scipy, pygraphviz.
- The argparse module is the default recommended standard library module for implementing basic command line applications See documentation here

Tasks:

Task 1. Initial Text Preprocessing

1. First, start with preprocessing the sentences and people files. This means you will create a cleaner version of the text and make it ready for analysis.
   You will need to perform the following steps.
   a. **Convert to Lowercase:** Make all person names, all sentences, and all entries in the "other names" column lowercase.
   b. **Remove Punctuation:** Remove **all** punctuation from the data (sentences,names and other names) by replacing punctuation with a **single whitespace** (e.g., “Mr.” becomes “mr ”). Punctuation marks will include anything that is **not** numbers or letters.
   c. **Remove Unwanted Words:** Remove all words that appear in the supplied remove file from all data.
   d. **Remove Consecutive Whitespaces:** Remove multiple consecutive whitespaces (>1 whitespace appearing one after the other). E.g., (“mr   dunder”) becomes (“mr dunder”).
   e. **Remove whitespace suffix and prefix:** Remove whitespaces at the end or beginning of the data. E.g., (“ Mr. Potter ”) becomes (“Mr. Potter”)
   f. **For the ‘other names’ column:** A list of lists should be created, where each sub-list contains a separate name. In the supplied data
   g. there will be no overlap between names or other names of different people (meaning, no two people will have a shared name or other name).
   h. **Split the data into Words:** Split sentences into words - each sentence becomes a list of words.
   i. **Remove any empty sentences**: any sentence with 0 words should be removed
   j. **Remove people with duplicate names**: if after cleaning two people have the same full name, keep the one that appeared first in the file.

Example, for the sentences:

- “Harry rode on a silver chariot”
- “It shined!   like a star of platinum”
- He was known as the red magician

And for a remove file containing the words:
“on”, “a”, “he”, “as”, “of”, “it”, “was”, “the”

The cleaned sentences would be:

- “harry rode silver chariot”
- “shined like star platinum”
- “known red magician“

This task requires the use of **4 arguments**:

- **Question Number**: A reference number for the task (e.g., 1).
- **Sentence Input Path**: The path for sentences file
- **People Input Path**: The path for the people file
- **Remove Input Path**: The path for a file with a list of common words to remove

The output should be printed in JSON format. Specifically, the output should be formatted as follows:

```
{

        f"Question {args.question_number}": {

        "Processed Sentences": processed_sentences

        "Processed Names": list of lists, each containing a person’s clean name, and nicknames
        (see example below)

        }

}
```

For the rest of the project you will be required to preprocess the data you will receive in the same way. It is good practice (and easier in the long run) to write generic functions and reuse them over the exercise.
A list of example inputs and preprocessed outputs is given to you to validate your preprocessing - make sure that when you run your preprocessing code on the given inputs, you get exactly the same outputs as we provided. Otherwise errors will propagate to the remainder of the project.

Input arguments:

- **Question Number**: Task number (in this case, 1).
- **Sentence Input Path**: The path for sentences file
- **People Input Path**:The path for People file
- **Remove Input Path**: The path for a file with a list of common words to remove

Examples :

# Q1_example_1

- Command: python main.py -t 1 -s Q1_examples/example_1/sentences_small_1.csv Q1_examples/example_1/people_small_1.csv -r REMOVEWORDS.csv
- Output: Q1_examples/example_1/Q1_result1.json

# Q1_example_2

- Command: python main.py -t 1 -s Q1_examples/example_2/sentences_small_2.csv Q1_examples/example_2/people_small_2.csv -r REMOVEWORDS.csv
- Output: Q1_examples/example_2/Q1_result2.json

# Q1_example_3

- Command: python main.py -t 1 -s Q1_examples/example_3/sentences_small_3.csv Q1_examples/example_3/people_small_3.csv -r REMOVEWORDS.csv
- Output: Q1_examples/example_3/Q1_result3.json

Task 2.Counting Sequences of Words

The goal in this task is to count common sequences of words in a sentence file. A **K-seq** represents a sequence of **K** consecutive words appearing one after another.
For example, a **2-seq** is a pair of consecutive words, while a **3-seq** is a triplet of consecutive words.

For a received **N** you will be required to create a k-seq key value pair data structure (see detailed examples below) for each 1 =< k <= N.
In each k-seq structure the key will be the string representation of a tuple for a given k-seq, and the value will be the number of times this k-seq appears in all sentences.
Notice, each sentence is processed independently, without minding sentences that come before or after it.

For example, for **N=3** The task is to compute the following:

- `1-seq`: Counts how many times each individual word (1-seq) appears in the text.
- `2-seq`: Counts how many times each pair of consecutive words (2-seq) appears.
- `3-seq`: Counts how many times each sequence of three consecutive words (3-seq) appears.

If the input sentences are: *Sentence 1 : Harry caught the Snitch*

Sentence 2 : *The Snitch flew away*

Sentence 3 : Hermione saw the Snitch.

After preprocessing the sentences become:

Sentence 1 : harry caught snitch

Sentence 2 : *snitch flew away*

Sentence 3 : *hermione snitch.*

For N=3, compute **1-seq**, **2-seq**, and **3-seq** as follows:

```
{
  "Question 1": {
        "3-Seq Counts": [
        ["1_seq", [["away", 1], ["caught", 1], ["flew", 1], ["harry", 1], ["hermione", 1], ["snitch", 3]]],
        ["2_seq", [["caught snitch", 1], ["flew away", 1], ["harry caught", 1], ["hermione snitch", 1], ["snitch flew", 1]]],
        ["3_seq", [["harry caught snitch", 1], ["snitch flew away", 1]]]
        ]
  }
}
```

This task requires the use of the following **arguments**:

- **Question Number**: Number of task performed (in this case, 2).
- **General data input**
  - If given --preprocessed flag, will receive path to a json file with output as received in task 1
  - **Otherwise, can receive regular output:**
    - **Sentence Input Path**: The path for sentences file
    - **Remove Input Path**: The path for a file with a list of common words to remove
- **N**: Maximal kseq length to create.

The expected output is that for each K a key-value pair list of each k-seq and its count be printed, sorted lexicographically by the key.

The output should be printed in JSON format. Specifically, the output should be formatted as follows:

```
{
  f"Question {args.question_number}": {
        f"{args.N}-Seq Counts": [
                ["1_seq”, [your_1_seq_key_pair_list_result]]
                …
                ["N_seq”, [your_N_key_pair_list_result]]
        ]
  }
}
```

Examples :

# Q2_example_1

- Command: python main.py -t 2 --maxk 3 -s Q2_examples/example_1/sentences_small_1.csv Q2_examples/example_1/people_small_1.csv -r REMOVEWORDS.csv
- Output: Q2_examples/example_1/Q2_result1.json

# Q2_example_2

- Command: python main.py -t 2 --maxk 4 -s Q2_examples/example_2/sentences_small_2.csv Q2_examples/example_2/people_small_2.csv -r REMOVEWORDS.csv
- Output: Q2_examples/example_2/Q2_result2.json

# Q2_example_3

- Command: python main.py -t 2 --maxk 5 -s Q2_examples/example_3/sentences_small_3.csv Q2_examples/example_3/people_small_3.csv -r REMOVEWORDS.csv

- Output: Q2_examples/example_3/Q2_result3.json

Task 3.Counting Person Mentions

In this task, the goal is to count how many times each person's name, including any alternate names (nicknames), appears in the text. The count will take into account both the main name and any additional names listed in the "Other Names" column from the People file.

The output should be a list of key value pairs where:

- **Keys**: Only the **main name** of the person (as listed in the People file).
- **Values**: The total count of how many times the main name or any of the alternate names (nicknames) appear in the text.

Clarification :

People Names and Alternate Names

Both the main name and any alternate names (nicknames) associated with each person will be considered.

- When counting occurrences, any references to alternate names (e.g., nicknames) should be included under the entry for the primary name.
- If only the nickname appears in the text but the primary name does not, it should still be counted under the original name.

For example:

- If the People file contains the name "Dudley Dursley" in the "Name" column and the alternate names "Dudders", "Big D", "Popkin", etc., in the "Other Names" column, all of these names will be counted when they appear in the text.

Pay attention to not count the same person twice due to a nickname being part of a characters name

You can assume that a character's name will not be wholly contained within a nickname (e.g if the characters name is "Harry Potter" it won't have a nickname like "The great Harry Potter") and that one nickname won't be wholly contained within another nickname (if Potter is a nickname, Mr. Potter won't be a nickname).

For example, If the input sentences are: *Sentence 1 : Harry caught the Snitch*

Sentence 2 : *The Snitch flew away*

Sentence 3 : Hermione saw the Snitch.

After preprocessing the sentences become:

Sentence 1 : harry caught snitch

Sentence 2 : *snitch flew away*

Sentence 3 : *hermione snitch.*

This task requires the use of the following **arguments**:

- **Question Number**: Number of task performed (in this case, 3).
- General data input
  - If given --preprocessed flag, will receive path to a json file with output as received in task 1
  - **Otherwise, can receive regular output:**
    - **Sentence Input Path**: The path for sentences file
    - **People Input Path**: The path for People file
    - **Remove Input Path**: The path for a file with a list of common words to remove

The output should be printed in JSON format. Specifically, the output should be formatted as follows:

```
{

        "Question 3": {

        "Name Mentions": {

                [
                    [“Name1” , Number],
                    [“Name2”,Number],
                    …
                ]
        }
}
```

Examples :

# *Q3_example_1*

- Command: python main.py -t 3 -s Q3_examples/example_1/sentences_small_1.csv Q3_examples/example_1/people_small_1.csv -r REMOVEWORDS.csv
- Output: Q3_examples/example_1/Q3_result1.json

# *Q3_example_2*

- Command: python main.py -t 3 -s Q3_examples/example_2/sentences_small_2.csv Q3_examples/example_2/people_small_2.csv -r REMOVEWORDS.csv
- Output: Q3_examples/example_2/Q3_result2.json

# *Q3_example_3*

- Command: python main.py -t 3 -s Q3_examples/example_3/sentences_small_3.csv Q3_examples/example_3/people_small_3.csv -r REMOVEWORDS.csv
- Output: Q3_examples/example_3/Q3_result3.json

# *Q3_example_4*

- Command: python main.py -t 3 -s Q3_examples/example_4/sentences_small_4.csv Q3_examples/example_4/people_small_4.csv -r REMOVEWORDS.csv
- Output: Q3_examples/example_4/Q3_result4.json

Task 4. Basic Search Engine Functionality

In this task, you need to build a basic search engine. Given a K-seq list file (a JSON file path representing a list of tuples of K consecutive words, such as those created in task two) and a sentences csv file, it will print a list of key value pairs where the keys are the supplied k-seqs and the values sentences in which that K-seq appear, where keys are the k-seqs and the value is the list of cleaned sentences in which they appear. Additionally, the search for a given K-seq should run in **O(1) time** (only each search, not including construction of data structure used for performing the search). In your video, discuss what data structure you used to enable this complexity.

This question refers only to the sentences, there is no need to account for names and other names.

This task requires the use of the following **arguments**:

- **Question Number**: A reference number for the task .
- **General data input**
  - If given --preprocessed flag, will receive path to a json file with output as received in task 1
  - **Otherwise, can receive regular output:**
    - **Sentence Input Path**: The path for sentences file
    - **Remove Input Path**: The path for a file with a list of common words to remove
- **Kseq JSON Path**: The path to the Kseq list file.

The output should be printed in JSON format. Specifically, the output should be formatted as follows:

```
{
        "Question 4": {
        "K-Seq Matches": [
        [
        "word1",
        [
        ["Sentence", "containing", "word1"],
        ["Another", "sentence", "with", "word1"]
        ]
    ],
    [
        "word1 word2",
        [
        ["Sentence", "containing", "word1", "word2", "next", "to", "each"]
        ]
    ],
    [
        "word3",
        [
        ["Sentence", "containing", "word3"]
        ]
    ]
    ]
    }
}
```

Notice that the result should be printed by the key’s alphabetical order.

For example, If the input sentences are:

Sentence 1 : Harry caught the Snitch!z

Sentence 2 : *The Snitch flew away*

Sentence 3 : Hermione saw the Snitch.

After preprocessing the sentences become:

Sentence 1 : harry caught snitch

Sentence 2 : *snitch flew away*

Sentence 3 : *hermione snitch*

Let's assume our K-seq list file contains the following tuples:

- “snitch”
- “harry caught”

For these K-seqs, the output would look like this:

```
{
    "Question 4": {
    "K-Seq Matches": [
    [
        "snitch",
        [
        ["snitch", "flew", "away"],
        ["hermione", "snitch"]
        ]
    ],
    [
        "harry caught",
        [
        ["harry", "caught", "snitch"]
        ]
    ]
    ]
    }
}
```

The K-sequence “snitch” appears in the sentences "snitch flew away" and "hermione snitch". The K-sequence 'harry', 'caught') appears in the sentence "harry caught snitch".

Examples :

# Q4_example_1

- Command: python main.py -t 4 -s Q4_examples/example_1/sentences_small_1.csv --qsek_query_path Q4_examples/example_1/kseq_query_keys_1.json -r REMOVEWORDS.csv
- Output: Q4_examples/example_1/Q4_result1.json

# Q4_example_2

- Command: python main.py -t 4 -s Q4_examples/example_2/sentences_small_2.csv --qsek_query_path Q4_examples/example_2/kseq_query_keys_2.json -r REMOVEWORDS.csv
- Output: Q4_examples/example_2/Q4_result2.json

# Q4_example_3

- Command: python main.py -t 4 -s Q4_examples/example_3/sentences_small_3.csv --qsek_query_path Q4_examples/example_3/kseq_query_keys_3.json -r REMOVEWORDS.csv
- Output: Q4_examples/example_3/Q4_result3.json

# Q4_example_4

- Command: python main.py -t 4 -s Q4_examples/example_4/sentences_small_4.csv --qsek_query_path Q4_examples/example_4/kseq_query_keys_4.json -r REMOVEWORDS.csv
- Output: Q4_examples/example_4/Q4_result4.json

Task 5.Contexts of People and Associated K-seqs

In this task, you will find the context in which people are mentioned. For each person you will need to find the k-seqs (for example, for N = 3: 1-seq, 2-seq, and 3-seq) that appear in the same sentence as him.

For a received input N, and for each person in the People file identify the sentences in which they are mentioned, and for each 1 =< K<= N that, find the k-seqs that make up these sentences, along with the k-seq's number of appearances in these sentences.

A person may be mentioned either by his name or any of their other names.

The goal is to print a key value pair list where:

1. **Keys** are the names of the persons.
2. **Values** are lists of k-seqs that appear in the same sentence as the person (regardless which of his name was used).
3. **Sorting**: the results should be sorted lexicographically by the keys

This task requires the use of the following **arguments**:

- **Question Number**: A reference number for the task .
- General data input
  - If given --preprocessed flag, will receive path to a json file with output as received in task 1
  - **Otherwise, can receive regular output:**
    - **Sentence Input Path**: The path for sentences file
    - **People Input Path**: The path for People file
    - **Remove Input Path**: The path for a file with a list of common words to remove
- **N**: length of maximal Kseq.

The output should be printed in JSON format. Specifically, the output should be formatted as follows:

```
  "Question 5": {
          “Person Contexts and K-Seqs”:
         { [“person name” ,[list_of_kseq_for_person]] }
}
```

- Notice that for each person, each associated k-seq should only appear once
- Notice that the persons and k-seqs for each person should be sorted in alphabetical order

Examples :

# Q5_example_1

Command: python main.py -t 5 -s Q5_examples/example_1/sentences_small_1.csv Q5_examples/example_1/people_small_1.csv -r REMOVEWORDS.csv --maxk 3

Output: Q5_examples/example_1/Q5_result1.json

# Q5_example_2

Command: python main.py -t 5 -s Q5_examples/example_2/sentences_small_2.csv Q5_examples/example_2/people_small_2.csv -r REMOVEWORDS.csv --maxk 4

Output: Q5_examples/example_2/Q5_result2.json

# Q5_example_3

Command: python main.py -t 5 -s Q5_examples/example_3/sentences_small_3.csv Q5_examples/example_3/people_small_3.csv -r REMOVEWORDS.csv --maxk 5

Output: Q5_examples/example_3/Q5_result3.json

# Q5_example_4

Command: python main.py -t 5 -s Q5_examples/example_4/sentences_small_4.csv Q5_examples/example_4/people_small_4.csv -r REMOVEWORDS.csv --maxk 6

Output: Q5_examples/example_4/Q5_result4.json

Task 6. Finding Direct Connections Between People

In this task, you need to analyze how people are connected based on their proximity in the document by constructing a graph.

People are considered connected if they are mentioned within the same window of sentences. A sentence window is a term for a sequence of consecutive sentences. A Window of size k will consist of k consecutive sentences.

This task involves constructing a graph (more explanation here), where:

1. **Nodes** represent the persons from the **People** list.
2. **Edges:** edge i,j between person i and person j exists if $C_{ij} \geq t$, where C_{ij} is their mutual appearance count (number of windows in which they are connected), and t is provided as the threshold parameter.

In this task you will print the list of edges making up the graph.

Requirements:

- The program should consider matches for **a person's name or part of it**, or any of their **nicknames** when identifying mentions of a person.
- If a person appears multiple times in the same window, they should only be counted **once** for that window.
- For any pair of persons appearing within the same window, their count should only reflect **one occurrence** per window.

For the parameter **K** the program should compute for any window of size K, and any pair persons, if they both appear in that window.

The parameter t is the threshold for connection between two people. Two people should appear together in at least t different windows to be considered connected.

The results should be a list of tuples, where each tuple contains the pair of connected persons (meaning for each pair of person i and person j such that $C_{ij} \geq t$**)**.

This task requires the use of the following **arguments**:

- **Question Number**: A reference number for the task .
- General data input
  - If given --preprocessed flag, will receive path to a json file with output as received in task 1
  - **Otherwise, can receive regular output:**
    - **Sentence Input Path**: The path for sentences file
    - **People Input Path**: The path for People file
    - **Remove Input Path**: The path for a file with a list of common words to remove
- **Window size :** The window size (K).
- **Threshold**: The threshold (t).

The output should be printed in JSON format. Specifically, the output should be formatted as follows:

```
{
   "Question 6": {
      "Pair Matches": [
                  [personA, personB ],
                  [personC, personD ],
                  [personB, personD ]
          ]


         }
}
```

- **The order of people within the pair should be by lexicographic order.**

Examples :

# Q6_example_1

- Command: python main.py -t 6 -s Q6_examples/example_1/sentences_small_1.csv Q6_examples/example_1/people_small_1.csv -r REMOVEWORDS.csv --windowsize 4 --threshold 4
- Output: Q6_examples/example_1/Q6_result1_w4_t4.json

# Q6_exmaple_2

- Command: python main.py -t 6 --s Q6_examples/exmaple_2/sentences_small_2.csv Q6_examples/exmaple_2/people_small_2.csv -r REMOVEWORDS.csv --windowsize 3 --threshold 2
- Output: Q6_examples/exmaple_2/Q6_result2_w3_t2.json

# Q6_exmaple_3

- Command: python main.py -t 6 -s Q6_examples/exmaple_3/sentences_small_3.csv Q6_examples/exmaple_3/people_small_3.csv -r REMOVEWORDS.csv --windowsize 5 --threshold 2
- Output: Q6_examples/exmaple_3/Q6_result2_w5_t2.json

# Q6_exmaple_4

- Command: python main.py -t 6 -s Q6_examples/exmaple_4/sentences_small_4.csv Q6_examples/exmaple_4/people_small_4.csv -r REMOVEWORDS.csv --windowsize 5 --threshold 1
- Output: Q6_examples/exmaple_4/Q6_result2_w5_t1.json

Task 7. Determining Indirect Connections Through a Graph

In this task, you need to identify whether two people are connected by **any path** in the previously constructed graph G. Even if they are not directly connected, they could still be linked through one or more intermediate people—so-called "friends" or "friends of friends." This can help reveal hidden relationships that may not be evident from direct mentions alone.
You will receive a list of person-pairs (Name1, Name2) taken from the People file and will use the graph G to check for indirect connections.The input list of person-pairs (Name1,Name2) can include names from either the **Name** column or the **"other names"** column.

print a list of lists where each sub list has the following values:

- **Name 1**: name of the first person.
- **Name 2**: name of the second person.
- **Value**: **True** if Name1 and Name2 are connected by any path in G; otherwise, **False**.

Notice:

- The names should be ordered in alphabetical order (Name 1 and Name 2, and the pairs in general)
- Each Pair can only appear once

This task requires the use of the following **arguments**:

- **Question Number**: A reference number for the task .
- **General data input**
  - If given --preprocessed flag, will receive path to a json file with output as received in task 6
  - **Otherwise, can receive regular output:**
    - **Sentence Input Path**: The path for sentences file
    - **People Input Path**: The path for People file
    - **Remove Input Path**: The path for remove file
    - **Window size :** The window size.
    - **Threshold**: The threshold for connection.
- **People_connections JSON Path**: The path to the people_connections list file.

- **Maximal_distance**: the maximal possible distance between two people.

The output should be printed in JSON format. Specifically, the output should be formatted as follows:

```
{

  "Question 7": {
            [Name1, Name2,  true],
            [Name1, Name3,  false],
            [Name5, Name2, false]
          }
}
```

Examples :

# *Q7_example_1*

- : python main.py -t 7 -s Q7_examples/example_1/sentences_small_1.csv Q7_examples/example_1/people_small_1.csv --pairs Q7_examples/example_1/people_connections_1.json -r REMOVEWORDS.csv --windowsize 5 --threshold 2 --maximal_distance 1000
- Output: Q7_examples/example_1/Q7_result1_w5_t2.json

# *Q7_exmaple_2*

- Command: python main.py -t 7 -s Q7_examples/exmaple_2/sentences_small_2.csv Q7_examples/exmaple_2/people_small_2.csv --pairs Q7_examples/exmaple_2/people_connections_2.json -r REMOVEWORDS.csv --windowsize 3 --threshold 2 --maximal_distance 1000
- Output: Q7_examples/exmaple_2/Q7_result1_w3_t2.json

# Q7_exmaple_3

- Command: python main.py -t 7 -s Q7_examples/exmaple_3/sentences_small_3.csv Q7_examples/exmaple_3/people_small_3.csv --pairs Q7_examples/exmaple_3/people_connections_3.json -r REMOVEWORDS.csv --windowsize 5 --threshold 1 --maximal_distance 1000
- Output: Q7_examples/exmaple_3/Q7_result1_w5_t1.json

# Q7_exmaple_4

- Command: python main.py -t 7 -s Q7_examples/exmaple_4/sentences_small_4.csv Q7_examples/exmaple_4/people_small_4.csv --pairs Q7_examples/exmaple_4/people_connections_4.json -r REMOVEWORDS.csv --windowsize 5 --threshold 2
- Output: Q7_examples/exmaple_4/Q7_result1_w5_t2.json

Task 8. Extension: Checking Fixed-Length Paths Between People

In this extension, you need to extend the functionality to identify whether two people are connected by a path of a fixed length. A path between two nodes cannot contain two occurrences of the same node (loops in nodes are not allowed).

Input:

1. **People connections file**: A json file containing a list of lists, where each sub list contains a pair of names representing two individuals in the graph.
   For example:
   ```
   {
      "keys": [
         ["harry potter", "aurelius dumbledore"],
         ["hermione granger", "draco malfoy"],
         ["hermione granger", "harry potter"]
      ]
   }
   ```
2. **K**: An integer representing the exact length of the path to check between Name1 and Name2.

Output:

A list of lists where each sub list has the following values:

- **Name 1**: name of the first person.
- **Name 2**: name of the second person.
- **Value**: **True**if there exists a path of exactly length K between Name1 and Name2 in the graph G; otherwise, **False**.

Analysis of Runtime:

In the accompanying video, report how the value of `K` influenced the runtime. Provide actual runtime measurements for each `K` in the range `[0, 5]`.

This task requires the use of the following **arguments**:

- **Question Number**: A reference number for the task .
- **General data input**
    - If given --preprocessed flag, will receive path to a json file with output as received in task 6
    - **Otherwise, can receive regular output:**
        - **Sentence Input Path**: The path for sentences file
        - **People Input Path**: The path for People file
        - **Remove Input Path**: The path for remove file
        - **Window size :** The window size.
        - **Threshold**: The threshold.
- **People_connections JSON Path**: The path to the people_connections list file.
- **K :** The path length

The output should be printed in JSON format. Specifically, the output should be formatted as follows:

```
{
        "Question 8": {
        "Pair Matches": [
        [
                "Name1",
                "Name2",
                false
        ],
        [
                "Name2",
                "Name3",
                true
        ]
        ]
        }
}
```

Examples :

# Q8_example_1

- Command: python main.py -t 8 -s Q8_examples/Q8_example/sentences_small.csv Q8_examples/Q8_example/people_small2.csv -r REMOVEWORDS.csv --windowsize 3 --threshold 2 --fixed_length 2
- Output: Q8_examples/exmaple_1/Q8_example_1_w_3_threshold_2__fixed_length_2.json

# Q8_example_2

- Command: python main.py -t 8 -s Q8_examples/Q8_example/sentences_small.csv Q8_examples/Q8_example/people_small2.csv -r REMOVEWORDS.csv --windowsize 3 --threshold 2 --fixed_length 3
- Output: Q8_examples/exmaple_1/Q8_example_1_w_3_threshold_2__fixed_length_3.json

# Q8_example_3

- Command: python main.py -t 8 -s Q8_examples/Q8_example/sentences_small.csv Q8_examples/Q8_example/people_small2.csv -r REMOVEWORDS.csv --windowsize 3 --threshold 2 --fixed_length 8
- Output: Q8_examples/exmaple_1/Q8_example_1_w_3_threshold_2__fixed_length_8.json

Task 9. Extension: Grouping Sentences by Shared Words

In this extension, you need to implement a basic text analysis algorithm to group sentences based on their shared word content. The goal is to identify groups of sentences that are related through a network of shared words. Here are the specific requirements:

1. **Graph Representation of Sentences**:
    - Treat each sentence as a "node" in a graph.
    - Create an edge between two nodes (sentences) if they share a certain number of words (determined by a threshold T).
2. **Threshold Parameter T**:
    - A positive integer that specifies the minimum number of shared words required for two sentences to be considered connected.

      For example:
        - If T = 2, two sentences are connected by an edge if they share **two or more words** in common (not necessarily consecutive).
3. **Grouping Sentences**:
    - Identify groups of interconnected nodes (sentences) in the graph.
        - groups of interconnected nodes - Meaning a group where all nodes can reach the other.
        - For each group, print all nodes that could be part of that group.
        - A node that has no connections is considered to be a group by itself.
    - Print a List of key value pair where:
        - **Keys**: "Group 1," "Group 2," etc.
        - **Values**: Lists of sentences that belong to each group.
        - **Sorting**:
            1. Group 1 should be the smallest, the last group the largest.
            2. Each group list of sentences should be sorted.
4. **Analysis of Runtime**:
    - In your video, discuss how you implemented this efficiently, and report briefly how the value of T impacted the number of clusters and runtime. (report actual numbers).

This task requires the use of the following **arguments**:

- **Question Number**: A reference number for the task .
- General data input
  - If given --preprocessed flag, will receive path to a json file with output as received in task 1
  - **Otherwise, can receive regular output:**
    - **Sentence Input Path**: The path for sentences file
    - **Remove Input Path**: The path for a file with a list of common words to remove
- **Threshold**: The threshold.

formatted as follows:

```
{
        "Question 9": {
        "group Matches": {
                ["Group 1", list of sentences],
                ["Group 2", list of sentences]
                }
        }
}
```

Examples :

# Q9_example_1

- Command: python main.py -t 9 -s Q9_examples/example_1/sentences_small_1.csv -r REMOVEWORDS.csv --threshold 1
- Output: Q9_examples/example_1/Q9_result1.json

# Q9_exmaple_2

- Command: python main.py -t 9 -s Q9_examples/exmaple_2/sentences_small_2.csv -r REMOVEWORDS.csv --threshold 3
- Output: Q9_examples/exmaple_2/Q9_result2.json

Q9_exmaple_3

- Command: python main.py -t 9 -s Q9_examples/exmaple_3/sentences_small_3.csv -r REMOVEWORDS.csv --threshold 6
- Output: Q9_examples/exmaple_3/Q9_result3.json