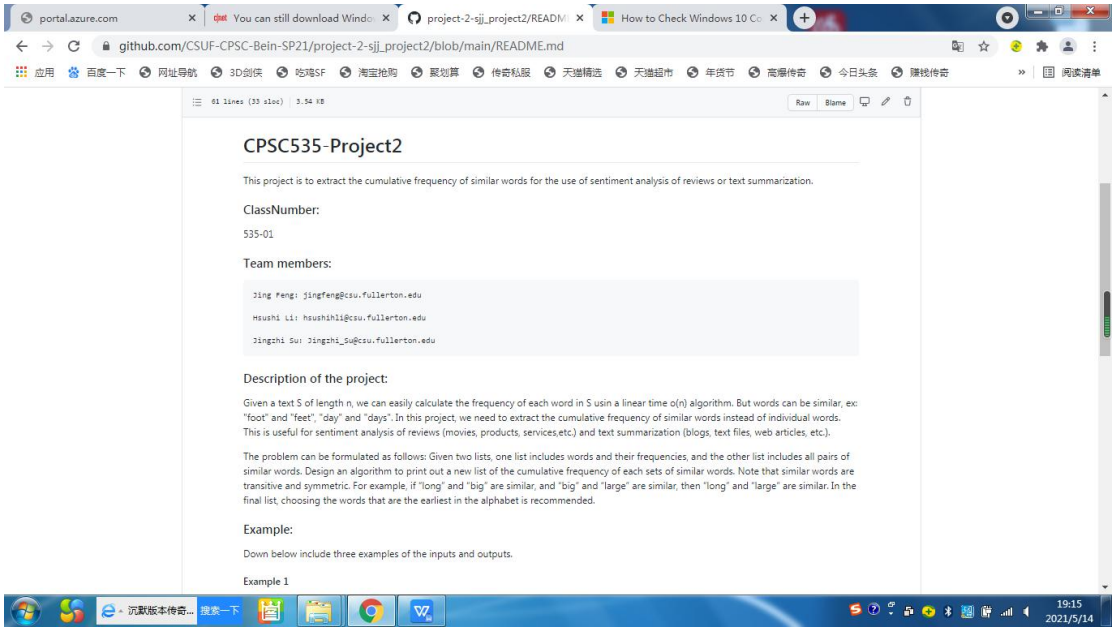
CPSC535-SP21-Project2

Team members:

Jing Feng: jingfeng@csu.fullerton.edu

Hsushi Li: hsushihli@csu.fullerton.edu

Jingzhi Su: Jingzhi_Su@csu.fullerton.edu



Pseudocode

```
def set_id(pair_list, id_dict, id):
    """
    This method is to assign id numbers to synonyms/similar words;
    Each set of synonyms will have the same id number.
    """
    synonym_leftpart = pair_list[0]
    synonym_rightpart = pair_list[1]

    # case 1: synonym_leftpart or synonym_rightpart is not in id_dict, which means the synonym word is not assigned an ID yet
    # (1) if synonym_leftpart is not in id_dict, then assign an ID to it
    if synonym_leftpart not in id_dict:
        id_dict[synonym_leftpart] = [id, synonym_rightpart]
        id += 1

    # (2) if synonym_rightpart is not in id_dict, since synonym_leftpart must be in id_dict, so assign the same ID to it
    if synonym_rightpart not in id_dict:
        id_dict[synonym_rightpart] = [id_dict[synonym_leftpart][0], synonym_leftpart]

    # case 2: synonym_leftpart and synonym_rightpart are both in id_dict
    # check if synonym_leftpart and synonym_rightpart have the same ID
    # if not, update the larger ID to the smaller one
    id_leftpart = id_dict[synonym_leftpart][0]
    id_rightpart = id_dict[synonym_rightpart][0]
    pair_leftpart = id_dict[synonym_leftpart][1]
    pair_rightpart = id_dict[synonym_rightpart][1]

    if id_leftpart > id_rightpart:
        id_dict[synonym_leftpart][0] = id_rightpart
        id_dict[pair_leftpart][0] = id_rightpart
    elif id_leftpart < id_rightpart:
        id_dict[synonym_rightpart][0] = id_leftpart
        id_dict[pair_rightpart][0] = id_leftpart

    return id_dict, id


def format_word(word):
    """
    This method is to format the user inputs WF and SYN.
    """
```

```
        new_word = re.sub(r'{|}|\"|”  |  “ |\(|\)', '', word).strip()
        return new_word


    if __name__ == "__main__":
        # Read the filenames of the user inputs
        filename_WF = the user input WF
    filename_SYN = the user input SYN

        id_dict = {} # Using a dictionary id_dict to record the id assignment for all synonyms pairs
        id = 0          # id starts from 0, increased by 1 when meeting a different word

        result_dict = {}      # Initial the result_dict dictionary which will be the final output result

        SYN_string = ""    # Used to get the entire string SYN for final output display
        WF_string = ""      # Used to get the entire string WF for final output display

        try:
            with open(filename_SYN, "r") as f_SYN:
                Read SYN file line by line
                For each pair of synonyms in SYN:
    Call set_id() function to assign/update an id for this pair of synonyms to id_dict

            with open(filename_WF, "r") as f_WF:
                Read WF file line by line
                Convert the string WF to a dictionary WF, and each element in WF is a word:frequency pair

                for each_pair in WF:
                    word = each_pair[0]
                    frequency = int(each_pair[1])
                    if word not in id_dict: # which means this word has no synonym
                        result_dict[id] = [word, frequency]
                        id += 1
                    else: # which means the word is in id_dict and has some synonym
                        syn_id = id_dict[word][0]
                        if syn_id not in result_dict:
                            result_dict[syn_id] = [word, frequency]
                        else:
                            if word < result_dict[syn_id][0]: #compare and store the word that is the earliest in the alphabet in result_dict
                                result_dict[syn_id][0] = word
                            result_dict[syn_id][1] += frequency # calculate the cumulative
```

```
frequency of its synonyms in result_dict

        print the result to screen

            except Exception as reason: # if there is no such file or the file cannot be opened, an error is raised
                    print(reason)
```

Edge Cases:

Here, three kinds of edge cases are specifically explained. We hope the approach we take could meet the project requirements.

(1) Edge case 1: There are repeated elements in WF [] or SYN [], including the following three cases:

a. There are repeated elements in WF[] only;

b. There are repeated elements in SYN[] only;

c. There are repeated elements in both WF[] and SYN[].

For example: (Example 3)

Input:

WF[] = { (“tons of”, 2), (“large number of ”, 12), (“mystical”, 13), (“magical”, 28), (“magic”, 5), (“unexplained”, 11), (“huge”, 2), (“large”, 51), (“horses”, 25), (“horse”, 24), (“large mammal”, 24), (“herbivore”, 5), (“large number of”,12)} of size 13

SYN[] = { (“herbivore”, “horses”), (“horse”, “large mammal”), (“horses”, “large mammal”), (“large number of”, “huge”), (“tons of”, “large”), (“huge”, “large”), (“mystical”, “magical”) , (“magical”,” unexplained”), (“magic”, “magical”), (“horse”,” large mammal”)} of size 10

Output:

CF[] = {("huge", 67),("magic", 57),("herbivore", 78)} of size 3

In the above example, the element (“large number of”, 12) appears twice in WF[], and the element (“horse”, “large mammal”) appears twice in SYN[]. The strategy of our algorithm is to treat such repeated elements as occurring only once.

(2) Edge case 2: there is some element in WF[], but not in SYN[].

For example:

Input:

WF[] = { (“tons of”, 2), (“large number of ”, 12), (“mystical”, 13), (“magical”, 28), (“magic”, 5), (“unexplained”, 11), (“huge”, 2), (“large”, 51), (“horses”, 25), (“horse”, 24), (“large mammal”, 24), (“herbivore”, 5), (“large number of”,12), (“dog”, 100)} of size 14

SYN[] = { (“herbivore”, “horses”), (“horse”, “large mammal”), (“horses”, “large mammal”), (“large number of”, “huge”), (“tons of”, “large”), (“huge”, “large”), (“mystical”, “magical”) , (“magical”,” unexplained”), (“magic”, “magical”), (“horse”,” large mammal”)} of size 10

Output:

CF[] = {("huge", 67),("magic", 57),("herbivore", 78),("dog", 100)} of size 4

In the above example, the element (“dog”, 100) appears in WF[], but the word “dog” has no similar words in SYN[]. Since there are no special requirements for this situation in the project document, the strategy of our algorithm is that if we encounter such a word, we will add this word and its frequency in the final list CF[].

(3) Edge case 3: there is some element in SYN[], but not in WF[].

For example:

Input:

WF[] = { (“tons of”, 2), (“large number of ”, 12), (“mystical”, 13), (“magical”, 28), (“magic”, 5), (“unexplained”, 11), (“huge”, 2), (“large”, 51), (“horses”, 25), (“horse”, 24), (“large mammal”, 24), (“herbivore”, 5), (“large number of”,12)} of size 13

SYN[] = { (“herbivore”, “horses”), (“horse”, “large mammal”), (“horses”, “large mammal”), (“large number of”, “huge”), (“tons of”, “large”), (“huge”, “large”), (“large”, “big”), (“mystical”, “magical”) , (“magical”,” unexplained”), (“magic”, “magical”), (“horse”,” large mammal”)} of size 11

Output:

CF[] = {("huge", 67),("magic", 57),("herbivore", 78)} of size 3

In the above example, there is a pair of similar words (“large”, “big”) in SYN[], but the right part of this pair, “big”, is not in WF[]. Since there are no special requirements for this situation in the project document, the strategy of our algorithm is that if we encounter such synonyms, we will still assign a related ID to “big”, which is the same ID as “large”. But because there is no “big” in WF[], this word does not affect the final cumulative frequency of its similar words, and the final result in CF[] is still correct.

Description of how to run the code:

1. Download the program cumulative_frequencies.py to your computer
2. Open the terminal and change the current working directory
3. Run the executable file in the terminal (Please use python3 to run the program)

   python3 cumulative_frequencies.py
4. Enter two .txt files names you wish to apply synonyms cumulative frequencies.

   If your test code is in the current working directory, you can type the filename directly.

   For example:

   WF.txt

   SYN.txt

   If not, you need to type the absolute pathname of the file into the terminal and return.

   For example:

   /Users/Jim/Desktop/WF.txt

   /Users/Jim/Desktop/SYN.txt
5. After running the program, it will print the output to the screen.

*** About the format of user input files:**

(1) The content in the **WF.txt** should look like as follows (for example 1). There are {} at the beginning and end of the entire list, and each pair inside is surrounded by (). In each pair, the word needs to be surrounded by “”, and the frequency is just an integer without “”.

{(“foot”, 5), (“feet”, 12), (“day”, 3), (“days”, 8), (“fear”, 2), (“scared”, 1), (“long”, 12), (“large”, 5), (“big”,5), (“was”, 4), (“is”, 4), (“are”, 15)}

(2) The content in the **SYN.txt** should look like as follows (for example 1). There are {} at the beginning and end of the entire list, and each pair inside is surrounded by (). Two similar words in each pair need to be surrounded by “”.

{(“foot”, “feet”), (“day”,“days”), (“fear”, “scared”), (“long” ,“big”), (“big” , “large”), (“is”, “are”), (“is”, “was”)}

*** About the order of the elements in CF[]**

In our algorithm, because we scan from left to right of WF[], **the order of the elements in the final result CF[] is based on the order of the elements in WF[],** which is a little different from the examples given in the project document. We also choose the words that are the earliest in the alphabet as recommended in the project document.

Three snapshots of code executing

Example1

```
Jings-MBP:p2 yichen$ python3 cumulative_frequencies.py
Input:
        Please enter the file name of WF: wf1.txt
        Please enter the file name of SYN: syn1.txt

Output:
        The WF : {(“foot”, 5), (“feet”, 12), (“day”, 3), (“days”, 8), (“fear”, 2), (“scared”, 1), (“long”, 12), (“large”, 5), (“big”,5), (“was”, 4), (“is”, 4), (“are”, 15)}
        The SYN: {(“foot”, “feet”), (“day”,“days”), (“fear”, “scared”), (“long” ,“big”), (“big” , “large”), (“is”, “are”), (“is”, “was”)}
        The CF : {(“feet”, 17),(“day”, 11),(“fear”, 3),(“big”, 22),(“are”, 23)}
```

Example2

```
Jings-MBP:p2 yichen$ python3 cumulative_frequencies.py
Input:
        Please enter the file name of WF: wf2.txt
        Please enter the file name of SYN: syn2.txt

Output:
        The WF : {(“tons of”, 2), (“large number of ”, 12), (“mystical”, 13), (“magical”, 28), (“magic”, 5), (“unexplained”, 11), (“huge”, 2), (“large”, 51), (“horses”, 25), (“hor
se, 24), (“large mammal”, 24), (“herbivore”, 5)}
        The SYN: {(“herbivore”, “horses”), (“horse”,“large mammal”), (“horses”, “large mammal”), (“large number of” ,“huge”), (“tons of” , “large”), (“huge”, “large”), (“mystical”
, “magical”) , (“magical”,“unexplained”), (“magic”, “magical”)}
        The CF : {(“huge”, 67),(“magic”, 57),(“herbivore”, 78)}
```

Example3

```
Jings-MBP:p2 yichen$ python3 cumulative_frequencies.py
Input:
        Please enter the file name of WF: wf3.txt
        Please enter the file name of SYN: syn3.txt

Output:
        The WF : { (“tons of”, 2), (“large number of ”, 12), (“mystical”, 13), (“magical”, 28), (“magic”, 5), (“unexplained”, 11), (“huge”, 2), (“large”, 51), (“horses”, 25), (“ho
rse, 24), (“large mammal”, 24), (“herbivore”, 5), (“large number of”,12)}
        The SYN: { (“herbivore”, “horses”), (“horse”,“large mammal”), (“horses”, “large mammal”), (“large number of” ,“huge”), (“tons of” , “large”), (“huge”, “large”), (“mystical
”, “magical”) , (“magical”,“unexplained”), (“magic”, “magical”), (“horse”,“large mammal”)}
        The CF : {(“huge”, 67),(“magic”, 57),(“herbivore”, 78)}
```