

CS-7611

COMPILER DESIGN LABORATORY

PROJECT ABSTRACT

**TOPIC: CONTENT SIMILARITY CHECKER FOR
SENTENCES**

TEAM MEMBERS:

TEJAS RAMESH RAMESH-2017103070

PAVITHRA KARTHY-2017103059

OVERVIEW / PROBLEM STATEMENT

The measurement of sentence similarity plays a very important role in text-related research and applications such as web page retrieval, text mining and dialogue systems. The methods currently existing for computing sentence similarity have been adopted from approaches used for long text documents, wherein they process sentences in a very high-dimensional space; these consequently inefficient, require human input, hence are not adaptable to some domains.

The aim of this project is to compute the similarity between very short texts of sentence length. It takes account of semantic information and word order information from the sentences.

Here in this project, we aim to develop a base for determining the measure of semantic similarity between 2 words, phrases or sentences and generate graph structure which displays their relationship.

Existing techniques for detecting similarity between long texts such as documents, have centered on analyzing shared words. These methods are usually effective while dealing with long texts because similar long texts will usually contain a degree of co-occurring words. Whereas in short texts, word co-occurrence may be rare or sometimes even null. This is primarily due to the inherent flexibility of natural language enabling people to express similar meanings using quite different sentences in terms of structure and word content.

Since such information in short texts is limited, this problem poses a difficult computational challenge. The focus of this project is on computing the similarity between very short texts, primarily of sentence length.

For a pair of sentences, T_1 and T_2 , that contain exactly the same words in the same order with the exception of two words from T_1 which occur in the reverse order in T_2 . For example:

. T_1 : A quick brown dog jumps over the lazy fox.

. T_2 : A quick brown fox jumps over the lazy dog.

As both the sentences contain the same words, any method based on a “bag of words” will give a decision that T_1 and T_2 are exactly the same. However, it is clear for a human that T_1 and T_2 are only similar to a certain extent. The dissimilarity between T_1 and T_2 is the result of the different word order. Therefore, a computational method for sentence similarity should take into account the impact of word order.

For the example pair of sentences T_1 and T_2 , the joint word set is:

$T = \{\text{A quick brown dog jumps over the lazy fox}\}$

While semantic similarity represents the lexical similarity, word order similarity provides information about the relationship between the words: which words appear in the sentence and which words come before or after other words. Both semantic and syntactic information plays an important role in conveying the meaning of sentences.

Thus the overall sentence similarity is defined as a combination of word order similarity and semantic similarity.

TOOLS USED TO DEVELOP CODE:

LINUX/TERMINAL IMPLEMENTATION

Operating system: LINUX (Ubuntu v.18.04)

Ubuntu is a free and open-source Linux distribution based on Debian. Ubuntu is officially released in three editions: Desktop, Server, and Core for the internet of things devices and robots. All the editions can run on the computer alone, or in a virtual machine.

Lex tool:

Lex is a computer program that generates lexical analyzers ("scanners" or "lexers").

Lex is commonly used with the yacc parser generator. Lex, Lex reads an input stream specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language. In addition to C, some old versions of Lex could also generate a lexer in Ratfor.

Yacc Tool:Yacc (Yet Another Compiler-Compiler)

It is a computer program for the Unix operating system developed by Stephen C. Johnson. It is a Look Ahead Left-to-Right (LALR) parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to Backus–Naur Form (BNF).Yacc is supplied as a standard utility on BSD and AT&T Unix. GNU-based Linux distributions include Bison, a forward-compatible Yacc replacement.

PYTHON IMPLEMENTATION

Jupyter Notebook:Project Jupyter is a nonprofit organization created to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". Spun-off from IPython in 2014 by Fernando Pérez, Project Jupyter supports execution environments in several dozen languages.

Notepad:To give text file inputs in the format (.txt)

NLTK(Natural language Toolkit):The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

IMPLEMENTATION DETAILS AND SAMPLE OUTPUTS.

FLOW OF THE PROJECT IN LINUX/TERMINAL IMPLEMENTATION:

1.Compile the lex code (project30703059.l) using the command:

```
lex project30703059.l
```

2.Compile the yacc code (project30703059.y) Using the command:

```
yacc project30703059.y
```

```
gcc y.tab.c -ll -ly
```

You will receive some warnings, but that is okay as it won't hinder the working of the code.

3. Run the program using the command:

```
./a.out
```

4. Upon entering the program you will receive instructions on how to execute the code.

```
WELCOME TO THE SENTENCE SIMILARITY CHECKER
```

```
PLEASE NOTE: ENTER THE STRINGS IN THE FOLLOWING MANNER:
```

```
EXAMPLE:The quick brown fox .
```

```
END1(###to view entered 1st string and its length###)
```

```
The quick brown dog .
```

```
END2(###to view entered 2nd string and its length###)
```

```
ENTER THE SIZE OF THE FIRST STRING
```

```
ENTER THE SIZE OF THE SECOND STRING
```

Enter the details as per user's wish

(PLEASE NOTE: FOR DISSIMILAR SENTENCE LENGTHS THE PROGRAMS OUTPUT IS:)

```
"STRINGS ARE DIFFERENT
```

```
DISSIMILAR CONTENT!!!!
```

```
EXAMPLE IF USER'S INPUT IS:
```

```
ENTER THE SIZE OF THE FIRST STRING 10
```

ENTER THE SIZE OF THE SECOND STRING 9

Whatever the input after this step the output is:

"STRINGS ARE DIFFERENT

DISSIMILAR CONTENT!!!!

After entering the sizes of the sentences and hitting enter:

User must enter the first sentence:

Example: The quick brown fox .

Followed by:

END1-(“END1”:DISPLAYS THE FIRST SENTENCE WHICH IS STORED IN A LINKED LIST)

SIMILARLY,

User must enter the second sentence:

Example: The quick brown dog .

Followed by:

END2-(“END2”:DISPLAYS THE SECOND SENTENCE WHICH IS STORED IN A LINKED LIST)

After entering both sentences:

User must type:

COMPARE-(“COMPARE”:runs the main logical part of the program to give the user the results-1.Subset of words in 1st sentence,2.Subset of words in the 2nd sentence,3.Total set of words in both sentences.)

After this Output is obtained:

- 1.Directly if content is similar.
- 2.By pressing Ctrl+D if the content is different.

FLOW OF THE PROJECT IN PYTHON (JUPYTER NOTEBOOK) IMPLEMENTATION:

- 1.Just download the .ipynb file and run it on jupyter notebook

SAMPLE OUTPUT SNAPSHOTS NEXT PAGE ONWARDS

(PLEASE NOTE:THE LEX AND YACC FILES IN THE SNAPSHOT HAVE BEEN RENAMED FOR PROJECT SUBMISSION AND EVALUATION PURPOSES .)

GITHUB REPOSITORY LINK FOR THIS PROJECT.

<https://github.com/tejas3070/CompilerLabProject>

SAMPLE OUTPUT SNAPSHOTS (LINUX/TERMINAL IMPLEMENTATION)

```
tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070$ lex project.l
tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070$ yacc project.y
tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070$ gcc y.tab.c -ll -ly
y.tab.c: In function 'yyparse':
y.tab.c:1111:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
    yychar = yylex ();
                ^~~~~~
y.tab.c:1246:7: warning: implicit declaration of function 'yyerror'; did you mean 'yyerrok'? [-Wimplicit-function-declaration]
    yyerror (YY_("syntax error"));
    ^~~~~~
    yyerrok
In file included from project.y:13:0:
project.l: In function 'insert1':
project.l:95:21: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
    temp1->next = temp;
                    ^
project.l:96:20: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
    temp->prev = temp1;
                    ^
project.l: In function 'insert2':
project.l:113:21: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
    temp6->next = temp5;
                    ^
project.l:114:21: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
    temp5->prev = temp6;
                    ^
project.l: In function 'traverse1':
project.l:136:15: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
    temp2 = temp2->next;
                ^
project.l: In function 'traverse2':
project.l:157:15: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
    temp7 = temp7->next;
                ^
project.l: In function 'compare':
project.l:175:7: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
```

COMPILING

```
tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070$ ./a.out
```

WELCOME TO THE SENTENCE SIMILARITY CHECKER

PLEASE NOTE: ENTER THE STRINGS IN THE FOLLOWING MANNER:

EXAMPLE:The quick brown fox .

END1(###to view entered 1st string and its length###)
The quick brown dog .

END2(###to view entered 2nd string and its length###)

ENTER THE SIZE OF THE FIRST STRING6

ENTER THE SIZE OF THE SECOND STRING6
they us they us them him

END1

SENTENCE 1
they us they us them him
THE LENGTH OF THE FIRST SENTENCE IS

6 we us them we us they

END2

SENTENCE 2
we us them we us they
THE LENGTH OF THE SECOND SENTENCE IS

6 COMPARE

RUNNING THE CODE FOR DISSIMILAR STRINGS

```
THE LENGTH OF THE SECOND SENTENCE IS
```

```
6
```

```
COMPARE
```

```
THE SUBSET OF WORDS IN SENTENCE 1
```

```
{they,us,them,him}
```

```
THE SUBSET OF WORDS IN SENTENCE 2
```

```
{we,us,them,they}
```

```
THE TOTAL SET OF WORDS IN BOTH SENTENCES
```

```
{they,us,them,him,we}
```

```
COMPARE
```

```
THE SUBSET OF WORDS IN SENTENCE 1
```

```
{they,us,them,him}
```

```
THE SUBSET OF WORDS IN SENTENCE 2
```

```
{we,us,them,they}
```

```
THE TOTAL SET OF WORDS IN BOTH SENTENCES
```

```
{they,us,them,him,we}
```

```
STRINGS ARE DIFFERENT
```

```
DISSIMILAR CONTENT!!!!tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070$ lex project.l
```

```
tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070$ yacc project.y
```

OUTPUT 1-(FOR DIFFERENT STRINGS)

WELCOME TO THE SENTENCE SIMILARITY CHECKER

PLEASE NOTE: ENTER THE STRINGS IN THE FOLLOWING MANNER:

EXAMPLE:The quick brown fox .

END1(###to view entered 1st string and its length###)
The quick brown dog .

END2(###to view entered 2nd string and its length###)

ENTER THE SIZE OF THE FIRST STRING10

ENTER THE SIZE OF THE SECOND STRING10
The quick brown dog jumps over the lazy fox .

END1

SENTENCE 1
The quick brown dog jumps over the lazy fox .
THE LENGTH OF THE FIRST SENTENCE IS

10 The quick brown fox jumps over the lazy fox .

END2

SENTENCE 2
The quick brown fox jumps over the lazy fox .
THE LENGTH OF THE SECOND SENTENCE IS

10 COMPARE

RUNNING THE CODE FOR SIMILAR STRINGS.

SENTENCE 2

The quick brown fox jumps over the lazy fox .

THE LENGTH OF THE SECOND SENTENCE IS

10

COMPARE

THE SUBSET OF WORDS IN SENTENCE 1

{The,quick,brown,dog,jumps,over,lazy,fox,.}

THE SUBSET OF WORDS IN SENTENCE 2

{The,quick,brown,fox,jumps,over,lazy,.}

THE TOTAL SET OF WORDS IN BOTH SENTENCES

{The,quick,brown,dog,jumps,over,lazy,fox,.}

STRINGS ARE THE SAME

SIMILAR CONTENT!!!!tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070\$ lex project.l

tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070\$ yacc project.y

tejasramesh@tejasramesh-GP63-Leopard-8RE:~/r070\$ gcc -y tab.c -ll -ly

OUTPUT 2-(FOR DISSIMILAR STRINGS)

```

void wordset()
{
    struct words W[count1];
    struct words1 W1[count2];
    struct words2 W2[count1];
    struct words3 W3[count2];
    struct words4 W4[count1];
    struct words5 W5[count1+count2];
    int i=0,j=0,l=0,count4;
    int co1=0,co2=0,co3=0;
    temp10=h1;
    temp11=h2;
    printf("\n");
    for(i=0;i<count1;i++)
    {strcpy(W[i].wo,temp10->w);
    temp10=temp10->next;
    }
    printf("\n");
    for(i=0;i<count2;i++)
    {strcpy(W1[i].wo1,temp11->w);
    temp11=temp11->next;
    }
    for(i=0;i<count1;i++)
    {
        for(j=0;j<co1;j++)
        {
            if(strcasecmp(W[i].wo,W2[j].wo2)==0)
            break;
        }
        if(j==co1)
        {strcpy(W2[co1].wo2,W[i].wo);
        co1++;
        }
    }
    printf("\n\nTHE SUBSET OF WORDS IN SENTENCE 1");
    printf("\n");
    printf("{");

```

SAMPLE CODE SNIPPET 1

```

int compare();
void create1(char *input1)
{
    temp =(struct node1 *)malloc(1*sizeof(struct node1));
    temp->prev = NULL;
    temp->next = NULL;
    strcpy(temp->w,input1);
    count1++;
}
void create2(char *input2)
{
    temp5 =(struct node2 *)malloc(1*sizeof(struct node2));
    temp5->prev = NULL;
    temp5->next = NULL;
    strcpy(temp5->w,input2);
    count2++;
}

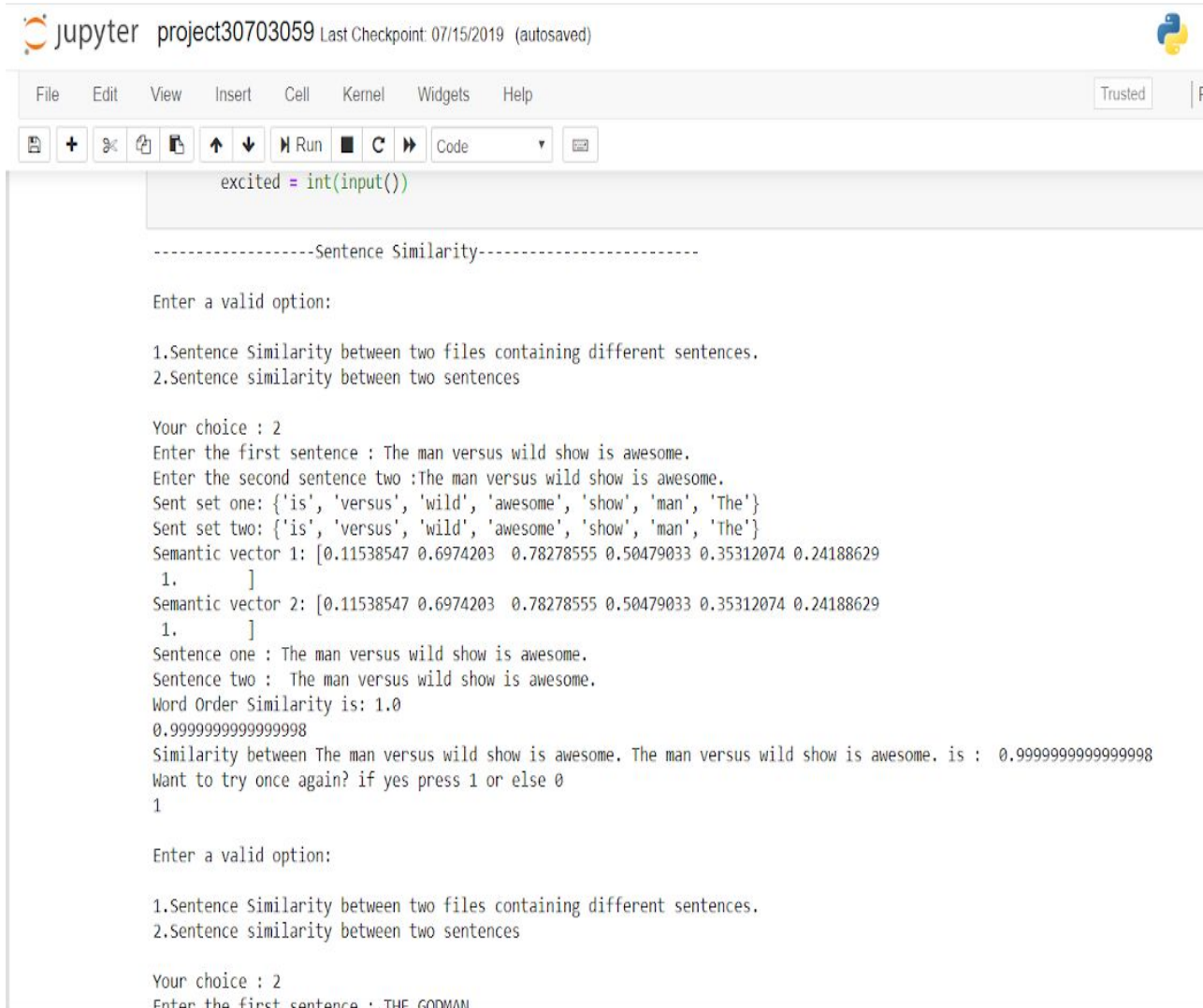
void insert1(char *input1)
{
    if (h1 == NULL)
    {
        create1(input1);
        h1 = temp;
        temp1 = h1;
    }
    else
    {
        if((count1==len1)&&(count2<=len2))
        {insert2(input1);}
        else{
            if(count1<=len1){
                create1(input1);
                temp1->next = temp;
            }
        }
    }
}

```

SAMPLE CODE SNIPPET 2

SAMPLE OUTPUT SNAPSHOTS

(PYTHON-JUPYTER NOTEBOOK IMPLEMENTATION)



The image shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook contains a Python script that calculates sentence similarity using semantic vectors and word order similarity. The script prompts the user for input and displays the results of the similarity calculations.

```
excited = int(input())

-----Sentence Similarity-----

Enter a valid option:

1.Sentence Similarity between two files containing different sentences.
2.Sentence similarity between two sentences

Your choice : 2
Enter the first sentence : The man versus wild show is awesome.
Enter the second sentence two :The man versus wild show is awesome.
Sent set one: {'is', 'versus', 'wild', 'awesome', 'show', 'man', 'The'}
Sent set two: {'is', 'versus', 'wild', 'awesome', 'show', 'man', 'The'}
Semantic vector 1: [0.11538547 0.6974203 0.78278555 0.50479033 0.35312074 0.24188629
1.      ]
Semantic vector 2: [0.11538547 0.6974203 0.78278555 0.50479033 0.35312074 0.24188629
1.      ]
Sentence one : The man versus wild show is awesome.
Sentence two : The man versus wild show is awesome.
Word Order Similarity is: 1.0
0.9999999999999998
Similarity between The man versus wild show is awesome. The man versus wild show is awesome. is : 0.9999999999999998
Want to try once again? if yes press 1 or else 0
1

Enter a valid option:

1.Sentence Similarity between two files containing different sentences.
2.Sentence similarity between two sentences

Your choice : 2
Enter the first sentence : THE GODMAN
```

OUTPUT FOR TYPED OUT SIMILAR SENTENCES.

```
jupyter project30703059 Last Checkpoint: 07/15/2019 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Your choice : 2
Enter the first sentence : THE GODMAN
Enter the second sentence two :THE MANGOD
Sent set one: {'THE', 'GODMAN'}
Sent set two: {'MANGOD', 'THE'}
first word : THE
second word MANGOD
[]
[]
first word : GODMAN
second word MANGOD
[]
[]
first word : MANGOD
second word GODMAN
[]
[]
first word : THE
second word GODMAN
[]
[]
Semantic vector 1: [0. 1. 1.]
Semantic vector 2: [1. 1. 0.]
Sentence one : THE GODMAN
Sentence two : THE MANGOD
first word : THE
second word MANGOD
[]
[]
first word : GODMAN
second word MANGOD
[]
[]
```

OUTPUT FOR TYPED OUT DIFFERENT SENTENCES.

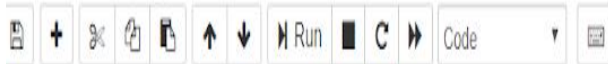


```

[]
[]
first word : THE
second word GODMAN
[]
[]
Semantic vector 1: [0. 1. 1.]
Semantic vector 2: [1. 1. 0.]
Sentence one : THE GODMAN
Sentence two : THE MANGOD
first word : THE
second word MANGOD
[]
[]
first word : GODMAN
second word MANGOD
[]
[]
first word : MANGOD
second word GODMAN
[]
[]
first word : THE
second word GODMAN
[]
[]
Word Order Similarity is: 0.0
0.4374999999999999
Similarity between THE GODMAN THE MANGOD is : 0.4374999999999999
Want to try once again?

```

OUTPUT FOR TYPED OUT DIFFERENT SENTENCES.



```
excited = int(input())
```

2.Sentence similarity between two sentences

Your choice : 1

Enter the path of the file :D:\input1.txt

Enter the path of the second fileD:\input1.txt

Sent set one: {'lazy', 'is', 'A', 'dog.Its', 'over', 'the', 'ferocious', 'attitude', 'brown', 'fox', 'quick', 'jumps'}

Sent set two: {'lazy', 'is', 'A', 'dog.Its', 'over', 'the', 'ferocious', 'attitude', 'brown', 'fox', 'quick', 'jumps'}

Semantic vector 1: [0.11538547 0.39607801 0.48553712 0.66639403 0.6974203 1.

1. 0.24021822 0.04046295 0.84885042 0.44185649 0.84885042]

Semantic vector 2: [0.11538547 0.39607801 0.48553712 0.66639403 0.6974203 1.

1. 0.24021822 0.04046295 0.84885042 0.44185649 0.84885042]

Sentence one : A quick brown fox jumps over the lazy dog.Its attitude is ferocious.

Sentence two : A quick brown fox jumps over the lazy dog.Its attitude is ferocious.

Word Order Similarity is: 1.0

Similarity between the sentences between the 2 file is :

1.0000000000000002

Want to try once again?

1

OUTPUT FOR SAME INPUT FILE GIVEN TWICE (SIMILAR INPUT)



```
excited = int(input())
```

-----Sentence Similarity-----

Enter a valid option:

- 1.Sentence Similarity between two files containing different sentences.
- 2.Sentence similarity between two sentences

Your choice : 1

Enter the path of the file :D:\input1.txt

Enter the path of the second fileD:\input2.txt

Sent set one: {'lazy', 'is', 'A', 'dog.Its', 'over', 'the', 'ferocious', 'attitude', 'brown', 'fox', 'quick', 'jumps'}

Sent set two: {'lazy', 'fox.Its', 'is', 'calm', 'A', 'nature', 'over', 'the', 'very', 'brown', 'in', 'and', 'composure', 'tim id', 'dog', 'quick', 'jumps'}

first word : lazy

second word calm

[Synset('lazy.s.01'), Synset('faineant.s.01')]

[Synset('composure.n.01'), Synset('calm_air.n.01'), Synset('calm.v.01'), Synset('steady.v.01'), Synset('calm.v.03'), Synset('sedate.v.01'), Synset('calm.s.01'), Synset('calm.a.02')]

first word : is

second word calm

OUTPUT FOR DIFFERENT INPUT FILES.



```
excited = int(input())
```

```

first word : quick
second word attitude
[Synset('quick.n.01'), Synset('quick.s.01'), Synset('flying.s.02'), Synset('agile.s.01'), Synset('quick.s.04'), Synset('immediate.s.05'), Synset('quick.s.06'), Synset('promptly.r.01')]
[Synset('attitude.n.01'), Synset('position.n.04'), Synset('attitude.n.03'), Synset('attitude.n.04')]
first word : jumps
second word attitude
[Synset('jump.n.01'), Synset('leap.n.02'), Synset('jump.n.03'), Synset('startle.n.01'), Synset('jump.n.05'), Synset('jump.n.06'), Synset('jump.v.01'), Synset('startle.v.02'), Synset('jump.v.03'), Synset('jump.v.04'), Synset('leap_out.v.01'), Synset('jump.v.06'), Synset('rise.v.11'), Synset('jump.v.08'), Synset('derail.v.02'), Synset('chute.v.01'), Synset('jump.v.11'), Synset('jumpstart.v.01'), Synset('jump.v.13'), Synset('leap.v.02'), Synset('alternate.v.01')]
[Synset('attitude.n.01'), Synset('position.n.04'), Synset('attitude.n.03'), Synset('attitude.n.04')]
Word Order Similarity is: 0.32135890469923556
Similarity between the sentences between the 2 file is :

0.5538874669795613
Want to try once again? if yes press 1 or else 0

```

OUTPUT FOR DIFFERENT INPUT FILES.



```
In [*]: from nltk import word_tokenize
from nltk import sent_tokenize
from nltk.corpus import wordnet as wn
import numpy as np
from nltk.corpus import brown
import math
import nltk

CONST_PHI = 0.2
CONST_BETA = 0.45
CONST_ALPHA = 0.2
CONST_PHI = 0.2
CONST_DELTA = 0.875
CONST_ETA = 0.4
total_words = 0
word_freq_brown = {}

def proper_synset(word_one, word_two):
    pair = (None, None)
    maximum_similarity = -1
    synsets_one = wn.synsets(word_one)
    synsets_two = wn.synsets(word_two)
    print("first word :", word_one)
    print("second word", word_two)
    print(synsets_one)
    print(synsets_two)
    if (len(synsets_one) != 0 and len(synsets_two) != 0):
        for synset_one in synsets_one:
            for synset_two in synsets_two:
                similarity = wn.path_similarity(synset_one, synset_two)
                if (similarityv == None):
```

CODE SNIPPET 1



```
def depth_common_subsumer(synset_one, synset_two):
    height = 100000000
    if synset_one is None or synset_two is None:
        return 0
    elif synset_one == synset_two:
        height = max([hypernym[1] for hypernym in synset_one.hypernym_distances()])
    else:
        hypernym_one = {hypernym_word[0]: hypernym_word[1] for hypernym_word in synset_one.hypernym_distances()}
        hypernym_two = {hypernym_word[0]: hypernym_word[1] for hypernym_word in synset_two.hypernym_distances()}
        common_subsumer = set(hypernym_one.keys()).intersection(set(hypernym_two.keys()))
        if (len(common_subsumer) == 0):
            height = 0
        else:
            height = 0
            for cs in common_subsumer:
                val = [hypernym_word[1] for hypernym_word in cs.hypernym_distances()]
                val = max(val)
                if val > height: height = val

    return (math.exp(CONST_BETA * height) - math.exp(-CONST_BETA * height)) / (
        math.exp(CONST_BETA * height) + math.exp(-CONST_BETA * height))

def word_similarity(word1, word2):
    synset_wordone, synset_wordtwo = proper_synset(word1, word2)
    return length_between_words(synset_wordone, synset_wordtwo) * depth_common_subsumer(synset_wordone, synset_wordtwo)

def I(search_word):
    global total_words
```

CODE SNIPPET 2