



BU-ALI SINA UNIVERSITY

REPORT

Document Similarity (Using LCS)

Mehran Shahidi
Mehrdad Shahidi

supervised by
Dr. Moharam Mansorizade

August 3, 2020

Summary

In this project, we are going to implement a similar document finder that will try to find similar document on the web based on the given document, and it's going to give a score how much the documents are similar to each other.

https://github.com/m3hransh/similar__document__finder



Contents

1	Introduction	1
2	Implementing Similarity Rater Using LCS	1
2.1	Python Implementatoin	1
2.1.1	Runtime and Memory Complexity	2
3	Similarity Score	2
3.1	Implementing Similarity Score in Python	3
4	Dependency Score	3
4.1	Partial Dependency	3
4.2	Implementing Dependency Score in Python	4
5	Documents Similarity Program	5
6	Find The Most Similar Code in Geeksforgeeks	8
6.1	Web Scraping	8
6.2	Implementing Web Scraping in Python	8
7	Command-Line Applicatoin	9
7.1	Execution samples	13

1 Introduction

Similarity is a complex concept. In this project we are going to use LCS (Longest Common Sequence) to find some scores to decide how similar two documents are to each other. Then we improve the functionality and try to solve a harder problem of searching similar documents based on a document at hand on the Internet to see if there exists contents that are close or they contain in the document. The second part of the project needs web scraping that is a complex problem by itself. At first, to simplify the task we only look on some specified websites that we know the format of their contents and can extract those. Then we make the problem harder and try to find some optimized method to solve the problem in a general way.

2 Implementing Similarity Rater Using LCS

The **longest common subsequence**¹ (LCS) problem is the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two sequences). It differs from the longest common substring problem: unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences. For example, consider the sequences (ABCD) and (ACBAD). They have 5 length-2 common subsequences: (AB), (AC), (AD), (BD), and (CD); 2 length-3 common subsequences: (ABD) and (ACD); and no longer common subsequences. So (ABD) and (ACD) are their longest common subsequences.

Let two sequences be defined as follows: $X = (x_1x_2 \cdots x_m)$ and $Y = (y_1y_2 \cdots y_n)$. The prefixes of X are $X_{1,2,\dots,m}$; the prefixes of Y are $Y_{1,2,\dots,n}$. Let $LCS(X_i, Y_j)$ represent the set of longest common subsequence of prefixes X_i and Y_j . This set of sequences is given by the following.

$$LCS(X_i, Y_j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ LCS(x_{i-1}, y_{j-1}) \cup x_i & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max\{LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases} \quad (1)$$

2.1 Python Implementatoin

Using the LCS algorithm we can find the longest common sequence in the two or more strings. But applying this method to long documents can be irrelevant. Because in document similarity problems we mostly care how semantically those are the same and finding a common sequence of characters has little to do with that. Instead, we can split the documents into words and find the longest common sequence of words.

Now, lets implement this in Python. The LCS function is going to take two lists of strings and find LCS among those. Using a dynamic programming approach and with use of tabulation we solve the LCS problem.

As an example, lets say, we have two following lists of strings.

$x = [Hey, Mehran, how, are, you, doing, today]$

$y = [Hey, Mehrdad, how, are, doing?]$

Using recursive formula in the equation 1 and taking a bottom up approach we can fill a table similar to what shown in the 1 and find the LCS by returning the bottom right cell in the table.

Table 1: LCS of two lists. Blue colored list is the answer.

	\emptyset	Hey	Mehran,	how	are	you	doning	today?
\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
Hey	\emptyset	$\leftarrow ['hey']$	$\leftarrow ['hey']$	$\leftarrow ['hey']$	$\leftarrow ['hey']$	$\leftarrow ['hey']$	$\leftarrow ['hey']$	$\leftarrow ['hey']$
Mehrdad,	\emptyset	$\uparrow ['hey']$	$\uparrow ['hey']$	$\uparrow ['hey']$	$\uparrow ['hey']$	$\uparrow ['hey']$	$\uparrow ['hey']$	$\uparrow ['hey']$
how	\emptyset	$\uparrow ['hey']$	$\uparrow ['hey']$	$\leftarrow ['hey', 'how']$	$\leftarrow ['hey', 'how']$	$\leftarrow ['hey', 'how']$	$\leftarrow ['hey', 'how']$	$\leftarrow ['hey', 'how']$
are	\emptyset	$\uparrow ['hey']$	$\uparrow ['hey']$	$\uparrow ['hey', 'how']$	$\leftarrow ['hey', 'how', 'are']$	$\leftarrow ['hey', 'how', 'are']$	$\leftarrow ['hey', 'how', 'are']$	$\leftarrow ['hey', 'how', 'are']$
doing?	\emptyset	$\uparrow ['hey']$	$\uparrow ['hey']$	$\uparrow ['hey', 'how']$	$\uparrow ['hey', 'how', 'are']$	$\uparrow ['hey', 'how', 'are']$	$\uparrow ['hey', 'how', 'are']$	$\uparrow ['hey', 'how', 'are']$

¹https://en.wikipedia.org/wiki/Longest_common_subsequence_problem

The code implemented in Python is shown in Listing 1. First, the table is filled with empty lists. Looping through the table, strting with second row and second column (first row and first column initialaized with empty before) using similar formula as equation 1 fill the table with appropriate values. *indx* function is used to map each index of texts with right index in table. Because the talble have one extra row and column to find the right index for the table we need to add 1 to the index of each character in the text. For convenience a *lambda* function is written for that.

```

1  def LCS(text1, text2):
2      '''
3      text1, text2 : two string or list of strings.
4      return: list of similar characters or strings.
5      '''
6      n = len(text1)
7      m = len(text2)
8
9      # create (n+1)*(m+1) table
10     table = [[]]*(m+1) for i in range(n+1))
11
12     # map index of texts to table
13     indx = lambda i: i+1
14
15     for i in range(n):
16         for j in range(m):
17             if text1[i] == text2[j]:
18                 # LCS(i,j) = LCS(i-1, j-1) ^ text1[i]
19                 table[indx(i)][indx(j)] = table[indx(i-1)][indx(j-1)] + [text1[i]]
20             else:
21                 # LCS(i, j) = max(LCS(i, j-1), LCS(i-1, j))
22                 table[indx(i)][indx(j)] = max(table[indx(i)][indx(j-1)], table[indx(i-1)][indx(j)], key= lambda x:len(x))
23
24     return table[n][m]
```

Listing 1: Implementatoin of LCS in python

2.1.1 Runtime and Memory Complexity

The runtime complexity of function is shown in Listing 1 is $O(m \times n)$, because it needs to loop through the table of size $m \times n$. There is subtlty in calculating Memory Complexity. In Python, lists only have references to their elements and don't store whole elements. And in total there is $m \times n$ references in the table with number of LCS distinct strings references. That in the worst case gives $O(m \times n \times \min\{n, m\})$.

3 Similarity Score

Now that we can find the LCS, how we can say how much the documents are similar? One of the methods is used in statistics to estimate similarity of two sets is **Jaccard similarity coefficient**.² The Jaccard index, also known as Intersection over Union and the Jaccard similarity coefficient (originally given the French name coefficient de communauté by Paul Jaccard), is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (2)$$

There is only a subtlety here, in LCS we don't find the intersections. So we can amend the formula this way:

$$J'(A, B) = \frac{|LCS(A, B)|}{|A| + |B| - |LCS(A, B)|} \quad (3)$$

²https://en.wikipedia.org/wiki/Jaccard_index

3.1 Implementing Similarity Score in Python

Python implementation is shown in the Listing 2.

```

1  def sim_rate(t1, t2, lcs):
2      '''
3      t1: list of words
4      t2: list of words
5      lcs: longest common sequence of t1 and t2
6      return: similarity rate and including rate.'''
7      rate = len(lcs)/(len(t1)+len(t2)-len(lcs))
8
9      return rate

```

Listing 2: jaccard similarity rate

4 Dependency Score

Other way to look at similarity is to see how much a set includes elements of the other set compare to all the elements in the set. Let's say we have two documents and one of those is subset of the other. if the second document has lots of strings compare to the first one. This causes low smilarity score. So we introduce another score to recognize how much one document depends on other. So the score is defined as following:

$$S(A, B) = \frac{|LCS(A, B)|}{|A|} \quad (4)$$

shows the dependency of set A on set B.

4.1 Partial Dependency

What if the documents are not the subset of each other nor similar but still there is some parts in them that are exactly the same. In LCS, we don't care about if the similar words occurs in dense or in a sparse space. As an example considier two following random texts. The colored parts shows the parts that are similar. As you can see if the uncolored text increase the Similarity and Dependency score is going to decrease, although they have exactly similar parts in them.

Text 1

Both rest of know draw fond post as. It agreement defective to excellent. Feebly do engage of narrow. Extensive repulsive belonging depending if promotion be zealously as. Preference inquietude ask now are dispatched led appearance. Small meant in so doubt hopes. Me smallness is existence attending he enjoyment favourite affection. Delivered is to ye belonging enjoyment preferred. Astonished and acceptance men two discretion. Law education recommend did objection how old. **To sure calm much most long me mean. Able rent long in do we. Uncommonly no it announcing melancholy an in. Mirth learn it he given. Secure shy favour length all twenty denote. He felicity no an at packages answered opinions juvenile.**

Text 2

Both rest of know draw fond post as. It agreement defective to excellent. Feebly do engage of narrow. Extensive repulsive belonging depending if promotion be zealously as. Performed suspicion in certainty so frankness by attention pretended. Newspaper or in tolerably education enjoyment. Extremity excellent certainty discourse sincerity no he so resembled. Joy house worse arise total boy but. Elderly up chicken do at feeling is. Like seen drew no make fond at on rent. Behaviour extremely her explained situation yet september gentleman are who. Is thought or pointed hearing he. **To sure calm much most long me mean. Able rent long in do we. Uncommonly no it announcing melancholy an in. Mirth learn it he given. Secure shy favour length all twenty denote. He felicity no an at packages answered opinions juvenile.**

Table 2: Two random texts. Colored parts are similar

For the complexity of recognizing this part we are not going to define a score, and only use diagrams to gain the insight about the existence of this. You can see the occurrence distribution graph of LCS in Figure 1.

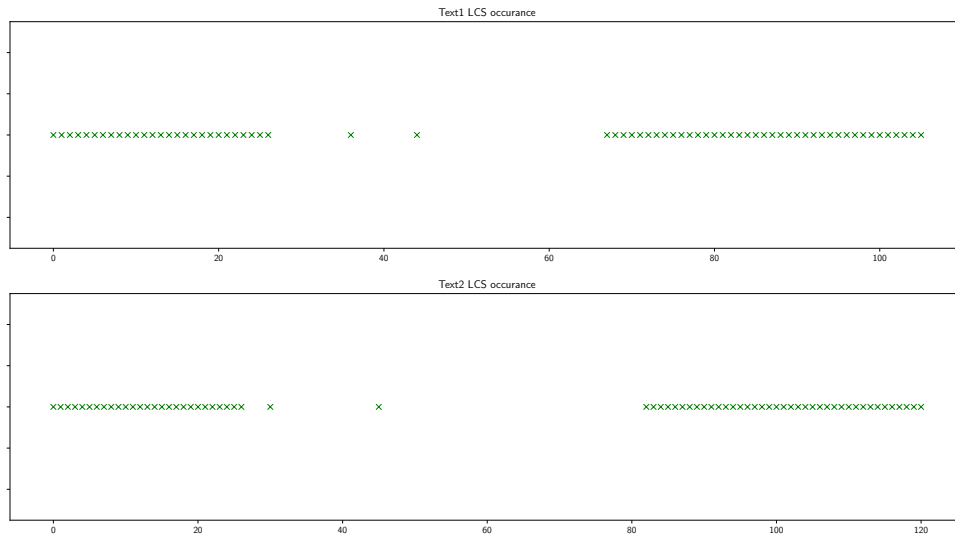


Figure 1: LCS occurrence distribution of texts in Table 2

4.2 Implementing Dependency Score in Python

Now let's implement the the Dependency Score using formula in the Equation 4. For the partial inclusion, the occurrence indexes of LCS will be found to draw the the graph later in another part of the program. The code is shown in the Listing 3.

```

1  def depend_score(t1, t2, lcs):
2      '''
3      t1: list of words
4      t2: list of words
5      lcs: longest common sequence of t1 and t2
6      return: return dependency rate of t1 on t2'''
7      return len(lcs)/len(t1)
8
9
10 def dist_finder(text, words):
11     '''
12     text: list of words
13     words: List of word occurrences in text
14     return: index of words occurrences in text'''
15     counter = 0
16     dist_list = []
17
18     for i, word in enumerate(text):
19         if len(words) == counter:
20             break
21         if words[counter] == word:
22             dist_list.append(i)
23             counter += 1
24
25     return dist_list

```

Listing 3: Inclusion Score

5 Documents Similarity Program

In this section, we write a Python program that uses previous implementations. The program is going to read two texts from files that are given by system arguments through the execution terminal and is going to print the Similarity and Dependency score with the LCS strings. Afterward, will plot the distribution of LCS occurrence in both text1 and text2. Before that, for making life easier, we create Class is called **Docs_Sim** that have all the functionality we need for comparing two documents. The code is shown in the Listing 4

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  import re
4
5
6  class Docs_Sim:
7      '''
8      takes two list of strings.
9      finds similarity between them by scoring them.
10     '''
11
12     def __init__(self, text1='', text2=''):
13         '''
14         text1, text2: list of string words
15         '''
16
17         self.text1 = re.findall(r'(\w+)', text1)
18         self.text2 = re.findall(r'(\w+)', text2)
19         self.lcs = []
20
21     def LCS(self):
22         '''
23         Implementing longest common sequence using Dynamic programming.
24         Take two file as an input and will find longest common sequence of
25         words in that.
26         text1, text2 : two string or list of strings.
27         return: list of similar characters of strings.
28         '''
29         n = len(self.text1)
30         m = len(self.text2)

```



```

31
32     # create (n+1)*(m+1) table
33     table = [[[]]*(m+1) for i in range(n+1)]
34
35     # map index of texts to table
36     indx = lambda i: i+1
37
38     for i in range(n):
39         for j in range(m):
40             if self.text1[i] == self.text2[j]:
41                 # LCS(i,j) = LCS(i-1, j-1) + 1
42                 table[indx(i)][indx(j)] = table[indx(i-1)][indx(j-1)] + [self.
text1[i]]
43             else:
44                 # LCS(i, j) = max(LCS(i, j-1), LCS(i-1, j))
45                 table[indx(i)][indx(j)] = max(table[indx(i)][indx(j-1)], table[
indx(i-1)][indx(j)], key= lambda x:len(x))
46
47         self.lcs = table[n][m]
48         return self.lcs
49
50
51     def dist_finder(self):
52         ''' Returns: (dist_list1, dist_list2) list occurrence distributao in
53         of lcs in the text1 and text2
54         '''
55         counter = 0
56         dist_list1 = []
57
58         for i, word in enumerate(self.text1):
59             if len(self.lcs) == counter:
60                 break
61             if self.lcs[counter] == word:
62                 dist_list1.append(i)
63                 counter += 1
64
65         counter = 0
66         dist_list2 = []
67
68         for i, word in enumerate(self.text2):
69             if len(self.lcs) == counter:
70                 break
71             if self.lcs[counter] == word:
72                 dist_list2.append(i)
73                 counter += 1
74
75         return dist_list1, dist_list2
76
77     def sim_score(self):
78         '''
79         return: similarity rate'''
80         rate = len(self.lcs)/(len(self.text1)+len(self.text2)-len(self.lcs))
81
82         return rate
83
84
85     def depend_score(self):
86         '''
87         return: (dependency score of t1 on t2, dependency score of t2 on t1)'''
88         dep_t1 = len(self.lcs)/len(self.text1)
89         dep_t2 = len(self.lcs)/len(self.text2)
90         return (dep_t1, dep_t2)
91
92
93     def draw_dist(self):
94         '''Plotting lcs occurrence distribution of each text'''
95         fig, (ax1, ax2) = plt.subplots(2)
96         dist1, dist2 = self.dist_finder()
97
98         ax1.set_title("Text1 LCS occurrence")
99         ax1.plot(dist1, np.zeros_like(dist1) + 0, 'x', color='green')
100        ax1.set_yticklabels([])

```

```

101
102     ax2.set_title("Text2 LCS occurrence")
103     ax2.plot(dist2, np.zeros_like(dist2) + 0, 'x', color='green')
104     ax2.set_yticklabels([])
105
106     plt.show()

```

Listing 4: *Docs_Sim* class Implementation

Using class *Docs_Sim* Listing 5 shows a program that takes two files as input. And print resulted score on Console with plot that shows the distribution of LCS occurrences.

```

1  #!/usr/bin/python3
2  import sys
3  from docs_sim import Docs_Sim
4
5
6  if __name__ == "__main__":
7      # getting file names from Terminal args
8      if len(sys.argv) >= 3:
9          f_name1 = sys.argv[1]
10         f_name2 = sys.argv[2]
11     else:
12         # default values of filenames
13         f_name1 = 'text1.txt'
14         f_name2 = 'text2.txt'
15
16     #opening files
17     with open(f_name1, 'r') as f1:
18         with open(f_name2, 'r') as f2:
19             # Read and split text to words
20             t1 = f1.read()
21             t2 = f2.read()
22             docs = Docs_Sim(t1,t2)
23             docs.LCS()
24             sim = docs.sim_score()
25             dep1,dep2 = docs.depend_score()
26
27             print("Similarity Score: {:.2%} \n"
28                   "Dependency Score of text1 on text2: {:.2%} \n"
29                   "Dependency Score of text2 on text1: {:.2%} \n"
30                   "\nLCS Words: \n{}".format(sim, dep1, dep2, ' '.join(docs.lcs)))
31
32     docs.draw_dist()

```

Listing 5: Python program using previous Implementation

The execution of the program is shown in Figure 4. *text1.txt* and *text2.txt* contains Text1 and Text2 that were shown in the Table 1. The plot is going to be similar to Figure 1. As you can see, the Similarity score of texts is 42.77% and Inclusion score (the rate the smaller text depends on the bigger text) is 64.15%.

Figure 2: Execution of program in Listing 5

```
$ ./main.py text1.txt text2.txt
Similarity Score: 42.77%
Dependency Score of text1 on text2: 64.15%
Dependency Score of text2 on text1: 56.20%

LCS Words:
Both rest of know draw fond post as.
It agreement defective to excellent. Feebly do engage of narrow.
Extensive repulsive belonging depending if promotion be zealously
as. so he To sure calm much most long me mean. Able rent long in
do we. Uncommonly no it announcingmelancholy an in. Mirth learn
it he given. Secureshy favour length all twenty denote.
He felicity noan at packages answered opinions juvenile.
```

If you have cloned the applications Git repository on GitHub, you can run `git checkout phase#1` to check out this version of the application.

6 Find The Most Similar Code in Geeksforgeeks

In this section, we write an application that will search geeksforgeeks on the Google by some specified keywords, and try to find the most similar code to the code that is given by the user. So first, we need to find the URLs by google search and then scrape code sections of each URL to compare it with the given document. For the comparison and Similarity score we will use the previous implementations.

6.1 Web Scraping

Web scraping is the porcess of gathering information from the Internet. The word usually refer to a process that involves automation. There challenges to web scraping. One of those are the websites varies in the structure. So we need unique way of treating those and gathering information we want. another challenge about web scraping is that through time the structure website can change and the program you wrote may become irrelevant in the future.

6.2 Implementing Web Scraping in Python

For implementation the searching part we will use **google** python package. For scraping part we will use **request** and **BeautifulSoup**. In this section, we only consider, gathering codes from Geeksforgeeks.org website to not deal with the variety structure of websites and can gather exactly the data we want. So first, with the use of google search we find relevant geeksforgeeks URLs and then scrape top list URLs to get relevant codes against some keywords given by the user. The following Listing 6 is the implementation of two functions that do these tasks.

```
1 import requests
2 from bs4 import BeautifulSoup
3 from googlesearch import search
4 from docs_sim import Docs_Sim
5 import re
6
7 def geeks_lcs(url, t1):
8     '''
9     url: geeksforgeeks url
10    t1: document string
11    return: docs_sim object of t1 and most similar code section in the URL
12    '''
13
```

```

14     r = requests.get(url)
15
16     # parse the html
17     soup = BeautifulSoup(r.content, 'html5lib')
18
19     # parse tree of codes sections
20     code_sections = soup.findAll('div', attrs={'class': 'container'})
21
22     max_score= 0
23     max_docs = Docs_Sim(t1)
24     # get each sectoin parse tree
25     for code_sec in code_sections:
26         code = []
27         # add words of each line of the code sectoin
28         for line in code_sec.findAll('code'):
29             if line['class'][0] != 'comments':
30                 code += re.findall(r'(\w+)', line.text)
31         docs = Docs_Sim(t1)
32         docs.text2 = code
33         # find lcs between t1 and the code
34         docs.LCS()
35         score = docs.sim_score()
36         # check if the section is more similar to t1 or not
37         if score > max_score:
38             max_score = score
39             max_docs = docs
40
41     return max_docs
42
43
44 def google_lcs(query,t1, num=5):
45     '''
46     query: search on google
47     t1: document to compare against
48     num: number of search on the google
49     return: (url that had the most similar code sec,
50             docs_sim object of t1 and most similar code section in the url)
51     '''
52     max_docs = Docs_Sim(t1)
53     url = ''
54     for j in search(query, tld='co.in', num=num, stop=5, pause=2):
55         docs= geeks_lcs(j, t1)
56         if docs > max_docs:
57             max_docs = docs
58             url = j
59
60     return url, max_docs

```

Listing 6: Implementing of Web Scrapping and Google search in Python

7 Command-Line Applicatoin

In this section, a CLI will be created that works based on the previous implementations. Description of the applicaton is shown in the Figure 3.

Figure 3: CLI Description

```

Usage: docsim [OPTIONS] COMMAND [ARGS]...

Simple CLI for finding similarity between codes

Options:
  --help  Show this message and exit.

Commands:
  check-dir      Find most similar pairs for each file in the directory.
  check-files    Check the similarity of two code files.
  clean-comments Find most similar pairs for each file in the directory.
  query          Search files on the google.

#####
Usage: docsim check-dir [OPTIONS] DIR

Find most similar pairs for each file in the directory.

Options:
  --plot / --no-plot  Draw plot of the lcs occurrence distribution

#####
Usage: docsim check-files [OPTIONS] FILENAME1 FILENAME2

Check the similarity of two code files.

Options:
  --plot / --no-plot  Draw plot of the lcs occurrence distribution

#####
Usage: docsim query [OPTIONS]

Search files on the google.

Options:
  -f, --file PATH      File path
  -d, --dir TEXT        Directory of code files
  -n, --num INTEGER     Number websites will check on Google
  -k, --keywords TEXT   Keywords for searching on google
  -w, --website TEXT    Website to search on google, default value:
                        www.geeksforgeeks.org

  --plot / --no-plot  Draw plot of the lcs occurrence distribution of two top
                        documents

```

For implementation **Click** package is used. The code is shown in the Listing 7.

```

1  #! /home/mehran/.virtualenvs/lcs/bin/python3
2  import click
3  import os
4  import sys
5  import re
6  from scraping import google_lcs
7  from docs_sim import Docs_Sim
8
9  @click.group()
10 def cli():
11     '''
12     Simple CLI for finding similarity between codes
13     '''
14     pass
15
16 @cli.command()
17 @click.option('--file', '-f', help='File path', type=click.Path(exists=True))
18 @click.option('--dir', '-d', help='Directory of code files')

```



```
19 @click.option('--num', '-n', default=3, help='Number websites will check on Google')
20 @click.option('--keywords', '-k', help='Keywords for searching on google' )
21 @click.option('--website', '-w', default='site:www.geeksforgeeks.org ', help='Website
    to search on google, default value: www.geeksforgeeks.org')
22 @click.option('--plot/--no-plot', default=False, help='Draw plot of the lcs occurrence
    distribution of two top documents')
23 def query(file, dir, num, keywords, website, plot):
24     '''
25     Search files on the google.
26     '''
27     if file:
28         if keywords != None:
29             with open(file, 'r') as f:
30                 t1 = f.read()
31
32             # geeksforgeeks query
33             query = website
34             query += keywords
35             url, docs = google_lcs(query, t1, num)
36             print('url: {} \nSimilarity Score: {:.2%}\n'
37                   'dependency Score: ({:.2%}, {:.2%})\n LCS:\n {}'.format(url,
38                                     docs.sim_score(), *docs.depend_score(), ' '.join(docs.lcs)))
39             if plot:
40                 docs.draw_dist()
41         else:
42             click.echo("--keywords option is not given!")
43     elif dir:
44         if keywords != None:
45             files = os.listdir(dir)
46
47             docs_list = []
48             for f in files:
49                 with open(os.path.join(dir, f), 'r') as file:
50                     t1 = file.read()
51
52                     query = website
53                     query += keywords
54                     url, docs_sim = google_lcs(query, t1, 5)
55                     docs_list.append((f, url, docs_sim))
56
57             docs_list.sort(key = lambda x : x[2], reverse=True)
58             for d in docs_list:
59                 print('file:{}\nurl: {} \nSimilarity Score: {:.2%}\n'
60                       'dependency Score: ({:.2%}, {:.2%})\n'.format(d[0], d[1],
61                             d[2].sim_score(), *d[2].depend_score()))
62
63             if plot:
64                 docs_list[0][2].draw_dist()
65         else:
66             click.echo("--keywords option is not given!")
67
68 @cli.command()
69 @click.argument('filename1', type=click.Path(exists=True))
70 @click.argument('filename2', type=click.Path(exists=True))
71 @click.option('--plot/--no-plot', default=False, help='Draw plot of the lcs occurrence
    distribution')
72 def check_files(filename1, filename2, plot):
73     '''Check the similarity of two code files.'''
74     with open(filename1, 'r') as f1:
75         t1 = f1.read()
76
77     with open(filename2, 'r') as f2:
78         t2 = f2.read()
79
80     docs = Docs_Sim(t1, t2)
81     docs.LCS()
82     sim = docs.sim_score()
83     dep1, dep2 = docs.depend_score()
84
85     print("Similarity Score: {:.2%} \n"
86           "Dependency Score of text1 on text2: {:.2%} \n"
87           "Dependency Score of text2 on text1: {:.2%} \n")
```

```

88         "\nLCS Words: \n{}".format(sim, dep1, dep2, ' '.join(docs.lcs)))
89     if plot:
90         docs.draw_dist()
91
92
93     @cli.command()
94     @click.argument('dir', type=click.Path(exists=True))
95     @click.option('--plot/--no-plot', default=False, help='Draw plot of the lcs occurrence
96         distribution')
97     def check_dir(dir, plot):
98         '''Find most similar pairs for each file in the directory.'''
99         files = os.listdir(dir)
100
101         docs_list = []
102         for f1 in files:
103             with open(os.path.join(dir, f1), 'r') as f:
104                 t1 = f.read()
105
106             max_docs_sim = Docs_Sim(t1)
107             f2_max = ''
108             for f2 in files:
109                 if f1 != f2:
110                     with open(os.path.join(dir, f2), 'r') as f:
111                         t2 = f.read()
112
113                     docs_sim = Docs_Sim(t1, t2)
114                     if docs_sim > max_docs_sim:
115                         max_docs_sim = docs_sim
116                         f2_max = f2
117
118             docs_list.append((f1, f2_max, max_docs_sim))
119
120         docs_list.sort(key = lambda x : x[2], reverse=True)
121         for d in docs_list:
122             print('file1: {} \nfile2: {} \nSimilarity Score: {:.2%} \n'
123                   'dependency Score: ({:.2%}, {:.2%}) \n'.format(d[0], d[1],
124                           d[2].sim_score(), *d[2].depend_score()))
125
126         if plot:
127             docs_list[0][2].draw_dist()
128
129     def remove_comment(string, file_ex='py'):
130         if file_ex == 'py':
131             string = re.sub(r'#![.*?]\n', '', string)
132         elif file_ex == 'cpp' or file_ex == 'c':
133             # remove all occurrences streamed comments (/*COMMENT */) from string
134             string = re.sub(re.compile("/\*.*?*/", re.DOTALL), "", string)
135             # remove all occurrence single-line comments (//COMMENT\n ) from string
136             string = re.sub(r"//.*?\n", "", string)
137
138         return string
139
140
141     @cli.command()
142     @click.argument('dir', type=click.Path(exists=True))
143     def clean_comments(dir):
144         '''Remove C-style and python-style comments of files in directory.'''
145         file_list = os.listdir(dir)
146
147         for file in file_list:
148             with open(os.path.join(dir, file), 'r') as f:
149                 code = f.read()
150
151             file_extention = re.search(r'\.(.*)$', file).group(1)
152             code = remove_comment(code, file_extention)
153
154             with open(os.path.join(dir, file), 'w') as f:
155                 f.write(code)
156
157
158     if __name__ == "__main__":

```

159 cli()

Listing 7: Python CLI using previous Implementations

7.1 Execution samples

Let's run the program on samples codes that is gathered from real project codes of DS students. The codes are implementation of n-queen or magic square. There is a *sample-codes* folder in the project that consist of 6 codes.

Figure 4: Execution of command **docsim** check-dir code-samples

```
$ docsim check-dir code-samples
file1: code06.cpp
file2: code01.cpp
Similarity Score: 98.90%
dependency Score: (98.90%, 100.00%)

file1: code01.cpp
file2: code06.cpp
Similarity Score: 98.90%
dependency Score: (100.00%, 98.90%)

file1: code05.cpp
file2: code02.cpp
Similarity Score: 74.19%
dependency Score: (81.66%, 89.03%)

file1: code02.cpp
file2: code05.cpp
Similarity Score: 74.19%
dependency Score: (89.03%, 81.66%)

file1: code04.py
file2: code03.py
Similarity Score: 26.11%
dependency Score: (42.16%, 40.69%)

file1: code03.py
file2: code04.py
Similarity Score: 26.11%
dependency Score: (40.69%, 42.16%)
```


Figure 5: Execution of command **docsim** query --file code-samples/code01.cpp -k 'n queen' --plot

```
$ docsim query --file code-samples/code01.cpp -k 'n queen' --plot
url:https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/
Similarity Score: 65.30%
dependency Score: (79.44%, 78.57%)
LCS:
define N void int board N N for int i 0 i N i for int j 0 j N j
board i j bool int board N N int row int col int i 0 i col i
if board row i return false for i row j col i 0 j 0 i j if board
i j return false for i row j col j 0 i N i j if board i j return
false return true bool int board N N int col if col N return true
for int i 0 i N i if board i col board i col 1 if board col 1
return true board i col 0 return false bool int board N N 0 0 0
if board 0 false Solution does not exist return false board
return true int main
```

