**Project 7: Incremental Search**

**1. Introduction**

In this project, you will learn how to design and implement an incremental search feature like Google Instant. Incremental search or real-time suggestions is a user interface interaction method to progressively search for and filter through text. As the user type text, one or more possible matches for the text are found and immediately presented to the user. This immediate feedback often allows the user to stop short of typing the entire word or phrase they were looking for. The user may also choose a closely related option from the presented list.

**2. Your task**

- Build an application with incremental search feature (e.g. web browsers, search instant, code completion)

**3. Optional**

For extra mark, you may consider some other effective datastructure and algorithm design support this feature for searching in large dataset.

Report project 7: Incremental Search

**Problem:** When the user wants to search something on internet but just know a part of sentence they want to search. From there, they want the program to suggest lines of sentences that best match with user’s input for easy searching. And I build a program to suggest information to the users.

**Relate works:**

* Collect the simple english sentences to use for program.

Table

Description automatically generated

* Choose the algorithm to solve the problem of this project.
* Longest Common Subsequence
* Get the five sentences have longest common subsequence with user input

|  |
| --- |
| **def** lcs**(**X**,** Y**):**  X**,** Y **=** X**.**lower**(),** Y**.**lower**()**  m **=** **len(**X**)**  n **=** **len(**Y**)**  bX **=** **[**0**]\*len(**X**)**  bY **=** **[**0**]\*len(**Y**)**  L **=** **[[None]\*(**n **+** 1**)** **for** i **in** **range(**m **+** 1**)]**  **for** i **in** **range(**m **+** 1**):**  **for** j **in** **range(**n **+** 1**):**  **if** i **==** 0 **or** j **==** 0 **:**  L**[**i**][**j**]** **=** 0  **elif** X**[**i**-**1**]** **==** Y**[**j**-**1**]:**  L**[**i**][**j**]** **=** L**[**i**-**1**][**j**-**1**]+**1  bX**[**i**-**1**]** **=** **max(**bX**)** **+** 1  bY**[**j**-**1**]** **=** **max(**bY**)** **+** 1  **else:**  L**[**i**][**j**]** **=** **max(**L**[**i**-**1**][**j**],** L**[**i**][**j**-**1**])**  **return** L**[**m**][**n**]** |
| **def** DictionaryLcs**(**UserInput**,** sentences**):**  DictLcs **=** **{}**  **for** sen **in** sentences**:**  DictLcs**[**sen**]** **=** lcs**(**sen**,** UserInput**)**  **return** **dict(sorted(**DictLcs**.**items**(),** key**=lambda** item**:** item**[**1**],** reverse**=True))** |
| **def** take**(**n**,** iterable**):**  "Return first n items of the iterable as a list"  **return** **dict(**islice**(**iterable**,** n**))** |
| five\_sens **=** take**(**5**,** DictionaryLcs**(**UserInput**,** sentences**).**items**())** |

* Selection sort
* Sort five sentences by longest common subsequence, length, first common index

|  |
| --- |
| **def** selectionSort**(**df**,** size**):**    **for** step **in** **range(**size**):**  idx **=** step  **for** i **in** **range(**step **+** 1**,** size**):**    # to sort in descending order, change > to < in this line  # select the minimum element in each loop  **if** df**.**iloc**[**i**][**'longest'**]** **>** df**.**iloc**[**idx**][**'longest'**]:**  idx **=** i  **elif** **(**df**.**iloc**[**i**][**'longest'**]** **==** df**.**iloc**[**idx**][**'longest'**]** **and**  df**.**iloc**[**i**][**'firstSameIndex'**]** **<** df**.**iloc**[**idx**][**'firstSameIndex'**]):**  idx **=** i  **elif** **(**df**.**iloc**[**i**][**'longest'**]** **==** df**.**iloc**[**idx**][**'longest'**]** **and**  df**.**iloc**[**i**][**'firstSameIndex'**]** **==** df**.**iloc**[**idx**][**'firstSameIndex'**]** **and**  df**.**iloc**[**i**][**'len'**]** **<** df**.**iloc**[**idx**][**'len'**]):**  idx **=** i    # put min at the correct position  **(**df**.**iloc**[**step**],** df**.**iloc**[**idx**])** **=** **(**df**.**iloc**[**idx**],** df**.**iloc**[**step**])**  **return** df |
| **def** DictionarySimilarSort**(**UserInput**,** sentences**):**  dfSimilar **=** pd**.**DataFrame**({**'sen'**:[],** 'longest'**:[],** 'len'**:[],** 'firstSameIndex'**:[]})**  **for** sen **in** sentences**:**  dfSimilar**.**loc**[len(**dfSimilar**.**index**)]** **=** **[**sen**,**  **len(**longestSubstringFinder**(**UserInput**,** sen**)),**  **len(**sen**),**  sen**.**lower**().**index**(**UserInput**.**lower**())** **if** UserInput**.**lower**()** **in** sen**.**lower**()** **else** NaN**]**  dfSimilar **=** selectionSort**(**dfSimilar**,** 5**)**  **return** dfSimilar**.**sen |

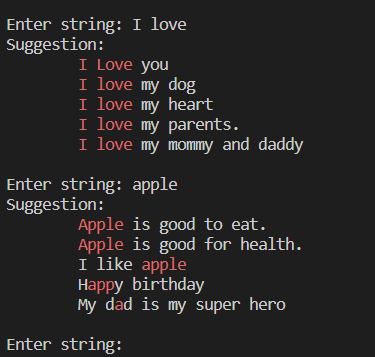
* Build the program by programming language. Apply the algorithm and other computation to find the sentences match with user’s input.
* Create Graphical User Interface

**Description of your solution to the problem and the architecture of your system:**

* Using Longest Common Subsequence to find five sentences within the sentences collected containing longest subsequence with user’s input.
* Caculate the score of five similar strings found and sort it and show suggestions to users.

|  |
| --- |
| **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  **while** **True:**  UserInput **=** **input(**'\nEnter string: '**)**  **if** **not** UserInput**:** **break**  five\_sens **=** take**(**5**,** DictionaryLcs**(**UserInput**,** sentences**).**items**())**  SimilarSens **=** DictionarySimilarSort**(**UserInput**,** **list(**five\_sens**))**  **print(**'Suggestion:'**)**  **for** sen **in** setColor**(**UserInput**,** SimilarSens**):**  **print(**'\t'**+**sen**)** |

**Run Program**



Using GUI

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated