Edit Distance

to calculate edit distance we need a large dataset, contains correct tokens.Also implementation of levenshtein function to calculate the distance between the terms.

Terms Dataset

Smaller Dataset

There is a large enough collection of correct terms in this link which is tokenized using my own **tokenizer** implemented in previous projects. The output contains 21895 tokens which is stored in `pltk/WordProcesses/WordsDataset` (each tokens is separated using *newLine character*)

Extra Large Dataset

(abadis dictionary)[https://dictionary.abadis.ir/fatofa/] has thousands of words which we can scrap them easily the code is implemented in `04-MakeADictionary/ScrapAbadisDictionary/runme.py`. this code is using `selenium` to send requests.This code has some perfect features such as: - because of rapid requests simultaneously the server will block the script's IP but the script will not terminate and will wait for the user to enter the capcha or change the IP address - Under any situation if code reruns it will remember the scraped pages and will continue scraping from the last page

> NOTE: Since the dataset extracted from abadis dictionary is too large for our purpose, we will use smaller dataset.

Levenshtein function

The main levenshtein function is described completely in the class. I made two more ideas hopefully to increase the algorithm's speed (which unfortunately was not successful!!!)

First Idea

The first idea is to have a distance limiter.which terminates the process if the distance goes over 2. To achieve this goal we must change the order of iterations over the matrix cells. For instance to iterate a **4x5** matrix we have:
The initial matrix is:

0	1	2	3	4	5
1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1

The new iteration contains two nested loops in the table below number of loops are separated using dash character:

0	1	2	3	4	5
1	1-0	1-1	1-2	1-3	1-4
2	2-0	3-0	3-1	3-2	3-3
3	2-1	4-0	5-0	5-1	5-2
4	2-2	4-1	6-0	7-1	7-2

The inner loop will terminated whenever if calculated distance is bigger than 2. i.e. for two words of `abcdef` and `bdeg`we have:

		a	b	c	d	e	f
	0	1	2	3	4	5	6
b	1	1	1	2	T	T	T
d	2	2	2	2	T	T	T
e	3	T	T	T	T	T	T
g	4	T	T	T	T	T	T

> NOTE: all `T` marks are known as `distanceLimitValue +1`, which means the answer of this table is 3

As you can see most of table is not calculated. but the extra calculation which this method affords to the algorithm to keep track of the order of cells to calculate is much more, which leads to slower algorithm(I will prove it practically)

Second Idea

To describe second idea, we can recursively find the necessary cells and calculate only those values: As a solid example lets calculate the the edit distance between `abcdef` and `abbdwf`.The goal is to find the bottom left cell in the table(colored in green).since the last characters are equal there is no need to calculate the upper and left neighbour(colored in red) we should only calculate the diagonal cell.

		a	b	c	d	e	f
	0	1	2	3	4	5	6
a	1	O	O	O	O	O	
b	2	O	O	O	O	O	
b	3	O	O	O	O	O	
d	4	O	O	O	O	O	
w	5	O	O	O	O	O	🟥 X
f	6					🟥 X	🟩 O

> as you can see approximately all of the cells are calculated

Third Idea (greedy method)

In this method each cell represents a node in a tree structure and the answer is in the root, Also each edge has the cost of **1** or **0** also we use greedy approach to extend the nodes while the edges which has **zero** cost or has less numbers as coordinates are in high priority to choose and extend.

i.e. for `abcdef` and `bcwgf`

		a	b	c	d	e	f
	0	1	2	3	4	5	6
b	1		leve4				
c	2			level3	level3		
w	3			level3	level2	level2	
g	4				level2	level1	
f	5						root

> value of `level4`=1 (bucause `b` is simillar)
> value of `level3`=`level4`=1 (bucause `c` is simillar)
> value of `level2`=`level3`+1=2 (bucause `w!=d`)
> value of `level1`=`level2`+1=3 (bucause `e!=g`)
> value of `root`=`level1`=3 (bucause `f` is simillar)

> NOTE: this algorithm is obviously acting greedy which may make mistakes some times, to avoid wrong answers from this algorithm the answers from this algorithm must validate by main `levenshtein` algorithm.

outputs and time calculations

simply run the `runme.py` code:

```
 python3 runme.py
21896 tokens loaded...
Please write your word: سبر
your words seems to be wrong, did u mean: سپر or بر or سبک or سر or جبر or سبق or سبب



calculating the time for other algorithms:
levenshtein calcilation time : 0.77s
words with one distance: ['سپر', 'بر', 'سبک', 'سر', 'جبر', 'سبق', 'سبب']
   and number of words whith 2 distance: 250

levenshtein_distanceLimiter calcilation time : 0.80s
words with one distance: ['سپر', 'بر', 'سبک', 'سر', 'جبر', 'سبق', 'سبب']
   and number of words whith 2 distance: 250

levenshtein_recursion calcilation time : 1.33s
words with one distance: ['سپر', 'بر', 'سبک', 'سر', 'جبر', 'سبق', 'سبب']
   and number of words whith 2 distance: 250

levenshtein_calculatingNecessaryCells calcilation time : 1.31s
words with one distance: ['سپر', 'بر', 'سبک', 'سر', 'جبر', 'سبق', 'سبب']
   and number of words whith 2 distance: 250

levenshtein_GreedyBFS calcilation time : 0.21s
words with one distance: ['سپر', 'بر', 'سبک', 'سر', 'جبر', 'سبق', 'سبب']
   and number of words whith 2 distance: 0
```