Autocorrection

Dillan Zurowski
200431334

Problem

- Words can easily be typed incorrectly
- Some documents are important and spelling mistakes can be an issue

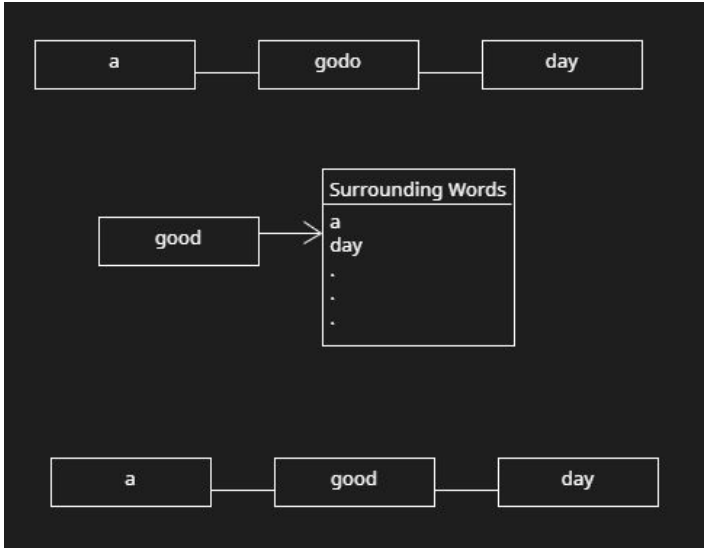
- Obviously autocorrect is already highly used:
  - how to make the program adjust to user preference

Goals

1. Find an incorrect word and replace it with the most likely replacement
    a. Allow the user to choose from a list of potential replacement words
    b. Correct all of the incorrect words without user intervention
2. Allow each word to dynamically change probability as the user uses the system
3. Allow each word to have a list of allowed surrounding words that can be used to increase probability

Knowledge and Data Representation

Input Data: I chose to represent the inputted data as a list of strings, this way I could iterate through the list while also keeping track of parent and child nodes.



Knowledge and Data Representation

Database:

1. Txt file on github - worked for storage but needed to write to the storage and add heuristics to each word.
2. Google Sheets - 2 columns, 1 for the list of words and 1 for the corresponding probability. Can dynamically increase probability to each word but needed more heuristics to increases accuracy. BF-search: sort the words by probability then iterate from highest to lowest.
    a. I wanted to use a different method of correction and increase the accuracy of the correction by checking the surrounding words. I also needed to add a column for the index so I can find the index of a word.
3. Finally, I added to the excel sheet so that each row that contains a word will have a list of words that appear around a word. I can use this list of surrounding words to increase the probability of the word when the incorrect word is surrounded by one of the stored surrounding words.

ex)

Word	Probability	Index	Surrounding words					
is	1	3001	this	a	essay	generally	definition	vague
words	1	3002	of	that				
hopefully	1	3003	that	are				
are	1	3004	hopefully	not	essays	characterized	that	out
i	1	3005	that	already				
reinforcement	1	3006						

Approach

1. Probability
    a. Base probability off of chance of the new word randomly being selected
    b. Base probability off of how many changes needs to be made to get to a solution
    c. Used: Start probability of all words at 1, increase each time the user uses that word as the replacement.
        i. When the words around the incorrect word match that of a temporary word, increase prob by 5 for each time it matches
2. Edit the incorrect word using 4 methods
    a. INSERT - a letter should be added to between each letter in a word.
    b. DELETE - removes each letter.
    c. SWAP - swaps two adjacent letters.
    d. REPLACE - changes one letter to another.

A small tree structure can be created around each word as the program compares the words before and after the current word with the list of surrounding words of each returned potential solution. This technique is used to automatically correct the entire sentence without user intervention. This method also dynamically adds the surrounding words to the list if they are not there already.

Naive Method was used to check if a word is in the database of words. This is a python function that iterates through the list and returns true if found.

Approach - Edit word

The following 4 functions generate a large list of edited words. With this list, we can find words that are part of the stored words.

1. Insert Letter

```
def insert_letter(word):
    split_l = []
    insert_list = []
    for i in range(len(word) + 1):
        split_l.append((word[0:i], word[i:]))
    letters = 'abcdefghijklmnopqrstuvwxyz'
    for l in letters:
      for a, b in split_l:
        insert_list.append(a+l+b)
    return insert_list
```

2. Delete Letter

```
def DeleteLetter(word):
    delete_list = []
    split_list = []
    for i in range(len(word)):
        split_list.append((word[0:i], word[i:]))
    for a, b in split_list:
        delete_list.append(a + b[1:])
    return delete_list
```

Approach - Edit word

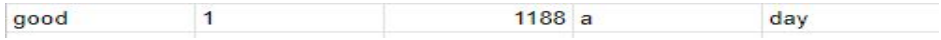
4. Swap Letters

```
# 2. SwitchLetter:swap two adjacent letters
def SwitchLetter(word):
    split_l = []
    switch_l = []
    for i in range(len(word)):
        split_l.append((word[0:i], word[i:]))
        for a,b in split_l:
          if len(b)>=2:
            switch_l.append(a + b[1] + b[0] + b[2:])
    return switch_l
```

3. Replace Letters

```
def replace_letter(word):
    split_l = []
    replace_list = []
    for i in range(len(word)):
        split_l.append((word[0:i], word[i:]))
    alphabets = 'abcdefghijklmnopqrstuvwxyz'
    for l in alphabets:
      for a,b in split_l:
        replace_list.append(a + l + (b[1:] if len(b) > 1 else ''))
    return replace_list
```

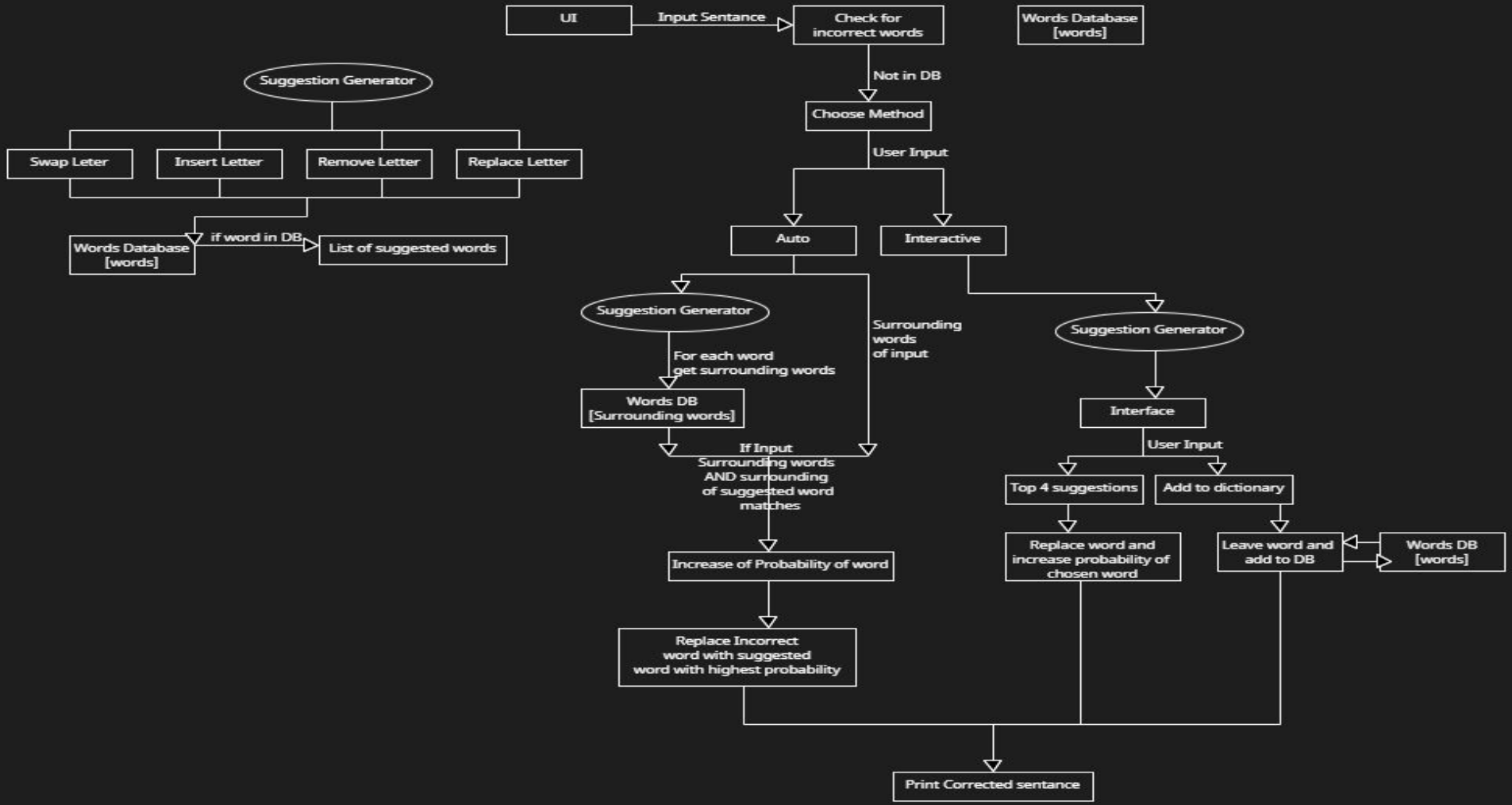
Approach - Option 1 (Autocorrect)

- ex) Input: **a** godo **day**
- The program will take the list generated from the previous 4 functions (temp suggestions and will check if the surrounding words in the input (**a** & **day**) and check if those words are in the list of stored surrounding words for that temporary suggestion.
- 

- Here, the probability of good being the correct word is 1. But “a” and “day” are both in the list of surrounding words so the probability will now be 1+5+5 = 11
- This repeats for all temp suggestions. The word with the highest probability will replace the word
  - **A good day**
- Cons: Takes time to process as it reads from the file more

Approach - Option 2(User preference)

- I wanted to allow the system to learn user preference as the user used the system over time.
- Use the 4 edit functions to generate a list of potential words
- Sort the list by probability
- Give the user the first 4 suggestions
- The user can choose one of the suggested words or add the “incorrect” word to the dictionary
- The system will recognize the selection of the user and increase the probability of that word.
- Not as accurate as option 1 but allows the system to learn from the user and add new words.


UI
Input Sentance
Check for incorrect words
Words Database [words]
Not in DB
Choose Method
User Input
Auto
Interactive
Suggestion Generator
Swap Leter
Insert Letter
Remove Letter
Replace Letter
Words Database [words]
if word in DB
List of suggested words
Suggestion Generator
For each word get surrounding words
Words DB [Surrounding words]
Surrounding words of input
If Input Surrounding words AND surrounding of suggested word matches
Increase of Probability of word
Replace Incorrect word with suggested word with highest probability
Suggestion Generator
Interface
User Input
Top 4 suggestions
Add to dictionary
Replace word and increase probability of chosen word
Leave word and add to DB
Words DB [words]
Print Corrected sentance

Autocomplete

```
Enter Text:this is an examlpe if wors that wll b fixed
Correct all? (y/n)y
Corrected: this is a example in words that will be fixed
```

User Selection

```
Enter Text:this is an examlpe if wors that wll b fixed
Correct all? (y/n)n
Select a Replacement for the word: examlpe
0: Add to dictionary
1: example
1
changing examlpe to example
Select a Replacement for the word: wors
0: Add to dictionary
1: words
2: works
3: word
4: work
1
changing wors to words
Select a Replacement for the word: wll
0: Add to dictionary
1: will
2: all
3: well
4: ill
1
changing wll to will
Select a Replacement for the word: b
0: Add to dictionary
1: be
2: a
3: by
4: i
1
changing b to be
Select a Replacement for the word: fixed
0: Add to dictionary
1: tired
2: feed
3: five
4: field
0
Adding fixed to dictionary
Corrected: this is an example if words that will be fixed
```

Discussion

Pros:

- Very dynamic - learns new words and their heuristics over time
- Learns user preference, more common words
- More accurate results with more use

Cons:

- Slow due to reading from google sheets during run time
  - Possible fix: Use SQL as the database instead
- Not enough training as most words do not have set of surrounding words

Conclusion

The program I created is very user-focused and implements a concept of constant learning and improvement.

In the future, I would like to explore ML concepts and learn more about how to train my dataset in a more efficient ways.

I wanted to avoid using too many libraries as to challenge myself in solving different problems and apply concepts in a new language.

Demo