### Reducible Words ( Due 30 Oct 2020 )

What are some of the longest English words that remain a valid English words as you remove one letter at a time from those words?

The letters can be removed anywhere from the word one at a time but you may not rearrange the remaining letters to form a valid word. Every time you remove a letter the remaining letters form a valid English word. Eventually you will end up with a single letter and that single letter must also be a valid English word. A valid English word is one that is found in the Oxford English Dictionary or the Webster's Dictionary.

For want of a better term we will call such words reducible words. Here are two examples of reducible words:

**1: sprite.** If you remove the *r* you get *spite*. Remove the *e* and you get *spit*. Remove the *s* and you get *pit*. Remove the *p* and you get *it*. Remove the *t* and you get *i* or *I* which is a valid English word.

**2: string.** Take away the *r* and you have *sting*. Take away the *t* and you have *sing*. Take away the *g* and you have *sin*. Take away the *s* and you have *in*. Take away the *n* and you have *i* or *I* which is a valid English word.

So all reducible words will reduce to one of three letters - *a*, *i*, and *o*. We will not accept any other letter as the final one letter word.

There is no *official* word list in an electronic form that we can use. We will use a curated word list file called [words.txt](https://www.cs.utexas.edu/users/mitra/csFall2020/cs313/assgn/words.txt). All the words are in lower case and are two letters or more in length. This word list will do as our input file.

Your output will be all the words of length 10 that are reducible. You will print each word in alphabetical order on a line by itself. Here is your output of [reducible words](https://www.cs.utexas.edu/users/mitra/csFall2020/cs313/assgn/reducible_words)of length 10.

Here is the outline of the code. The algorithm has been discussed in class. You may always add more helper functions as needed.

# Input: takes as input a positive integer n

# Output: returns True if n is prime and False otherwise

def is\_prime ( n ):

if (n == 1):

return False

limit = int (n \*\* 0.5) + 1

div = 2

while (div < limit):

if (n % div == 0):

return False

div += 1

return True

# Input: takes as input a string in lower case and the size

# of the hash table

# Output: returns the index the string will hash into

def hash\_word (s, size):

hash\_idx = 0

for j in range (len(s)):

letter = ord (s[j]) - 96

hash\_idx = (hash\_idx \* 26 + letter) % size

return hash\_idx

# Input: takes as input a string in lower case and the constant

# for double hashing

# Output: returns the step size for that string

def step\_size (s, const):

# Input: takes as input a string and a hash table

# Output: no output; the function enters the string in the hash table,

# it resolves collisions by double hashing

def insert\_word (s, hash\_table):

# Input: takes as input a string and a hash table

# Output: returns True if the string is in the hash table

# and False otherwise

def find\_word (s, hash\_table):

# Input: string s, a hash table, and a hash\_memo

# recursively finds if the string is reducible

# Output: if the string is reducible it enters it into the hash memo

# and returns True and False otherwise

def is\_reducible (s, hash\_table, hash\_memo):

# Input: string\_list a list of words

# Output: returns a list of words that have the maximum length

def get\_longest\_words (string\_list):

def main():

# create an empty word\_list

# open the file words.txt

# read words from words.txt and append to word\_list

# close file words.txt

# find length of word\_list

# determine prime number N that is greater than twice

# the length of the word\_list

# create an empty hash\_list

# populate the hash\_list with N blank strings

# hash each word in word\_list into hash\_list

# for collisions use double hashing

# create an empty hash\_memo of size M

# we do not know a priori how many words will be reducible

# let us assume it is 10 percent (fairly safe) of the words

# then M is a prime number that is slightly greater than

# 0.2 \* size of word\_list

# populate the hash\_memo with M blank strings

# create an empty list reducible\_words

# for each word in the word\_list recursively determine

# if it is reducible, if it is, add it to reducible\_words

# find words of length 10 in reducible\_words

# print the words of length 10 in alphabetical order

# one word per line

if \_\_name\_\_ == "\_\_main\_\_":

main()

For this assignment you may work with a partner. Both of you must read the paper on [Pair Programming](https://www.cs.utexas.edu/users/mitra/csFall2020/cs313/assgn/PairProg-CACM-1999.pdf) and abide by the ground rules as stated in that paper.

The file that you will be uploading will be called **Reducible.py**. We are looking for clean and structured design. The file will have a header of the following form:

# File: Reducible.py

# Description:

# Student Name:

# Student UT EID:

# Partner Name:

# Partner UT EID:

# Course Name: CS 313E

# Unique Number:

# Date Created:

# Date Last Modified:

If you are working with a partner, you will be submitting only one program but make sure that you have your partner's name and eid in your program. If you are working alone, then remove the two lines that has the partner's name and eid in the header.

Use the [Canvas](http://canvas.utexas.edu/)system to submit your **Reducible.py** file. We should receive your work by 11 PM on Friday, 30 Oct 2020. There will be substantial penalties if you do not adhere to the guidelines. Remember Python is case sensitive. The name of your file must match exactly what we have specified.

* Your Python program should have the proper header.
* Your code must run before submission.
* You should be submitting your file through the web based *Canvas* program. We will not accept files e-mailed to us.