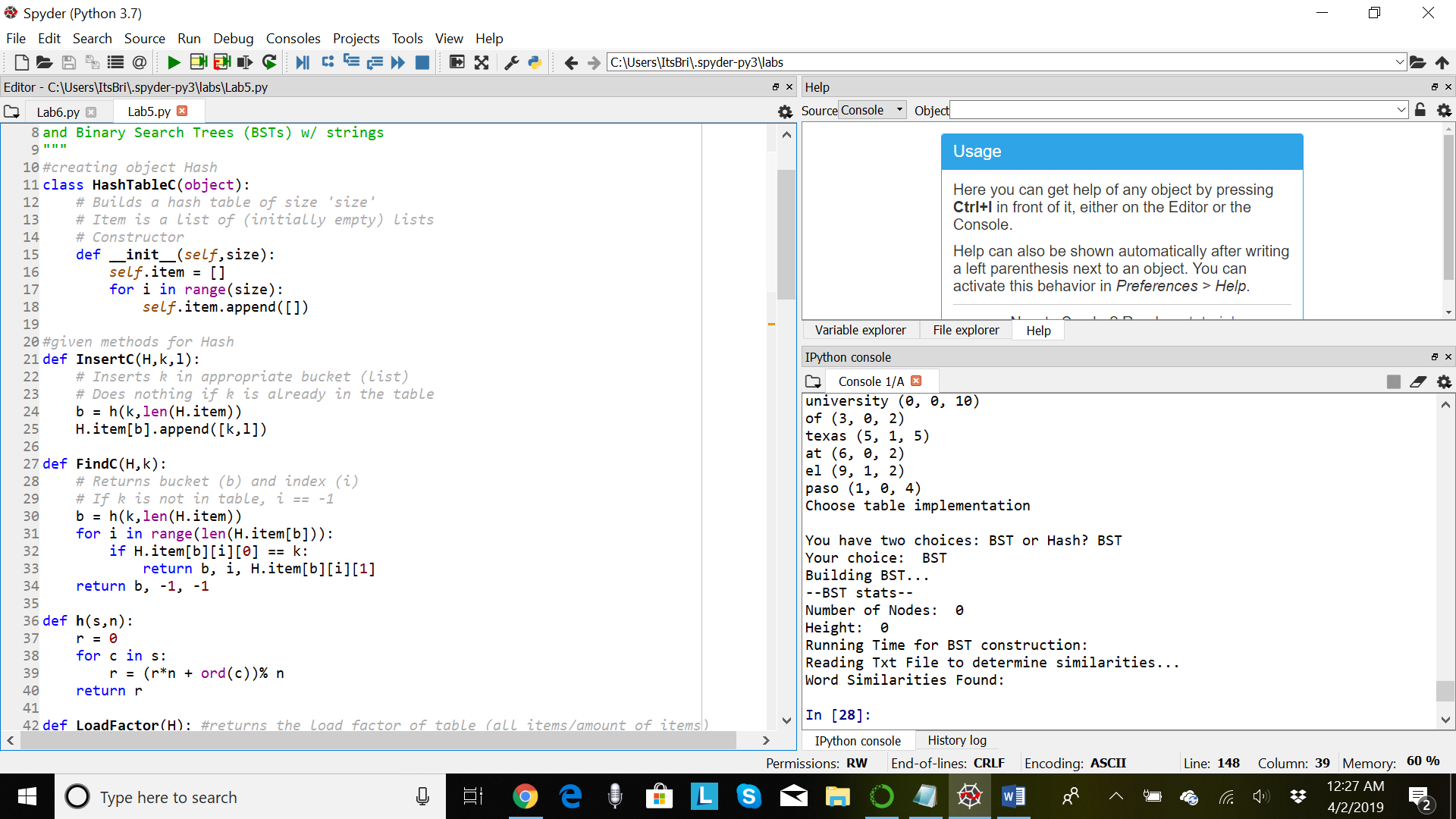
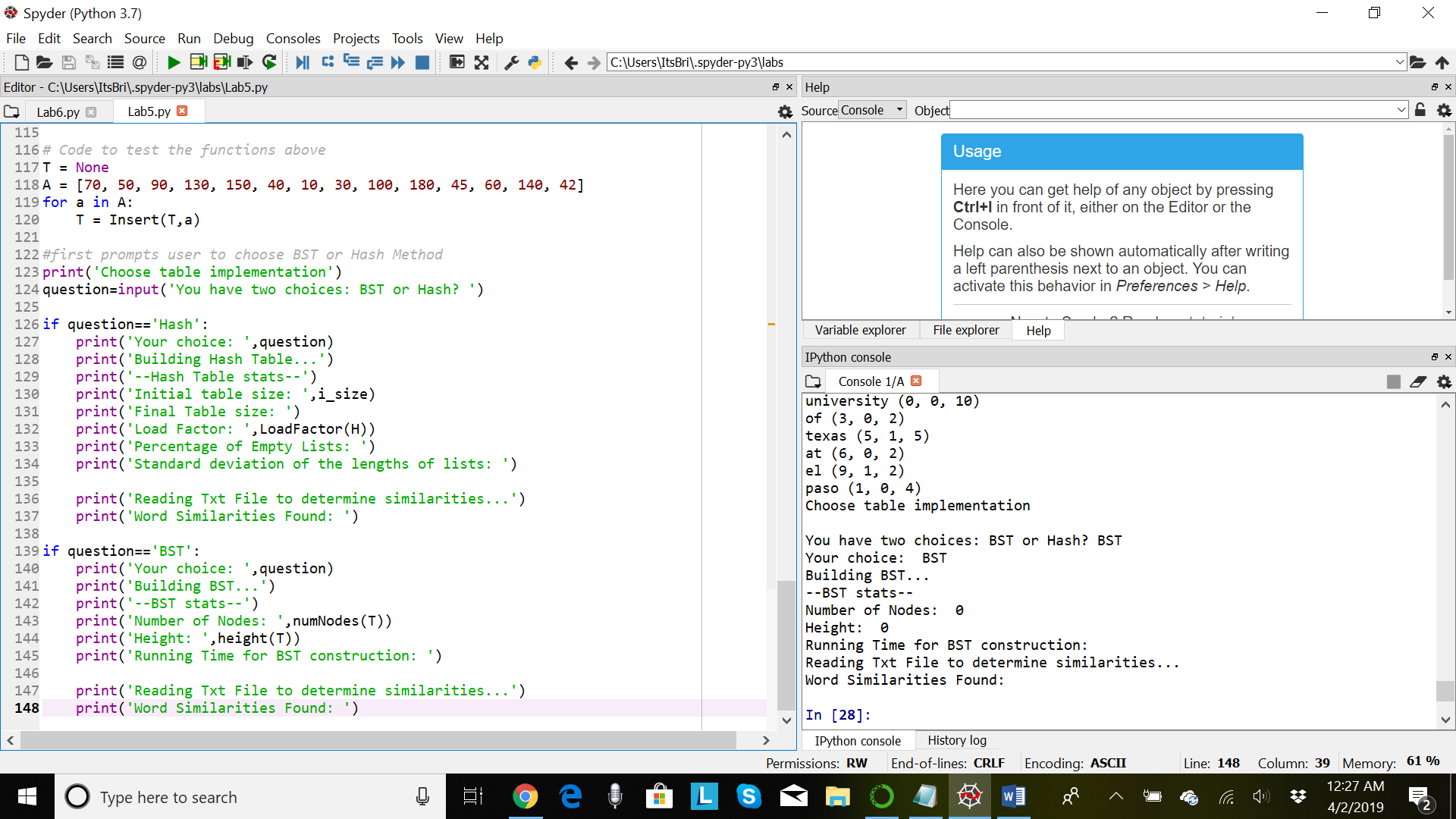
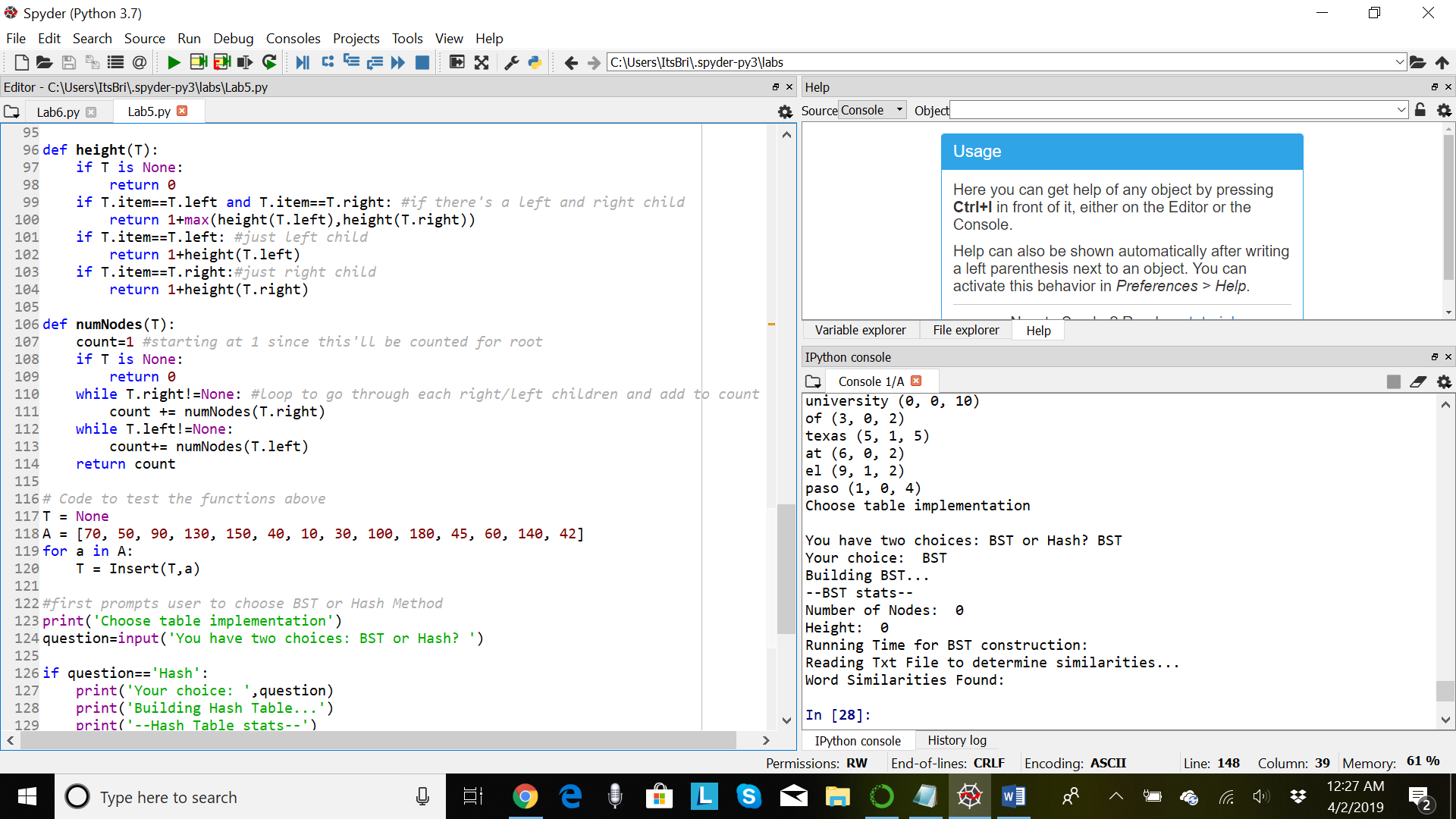
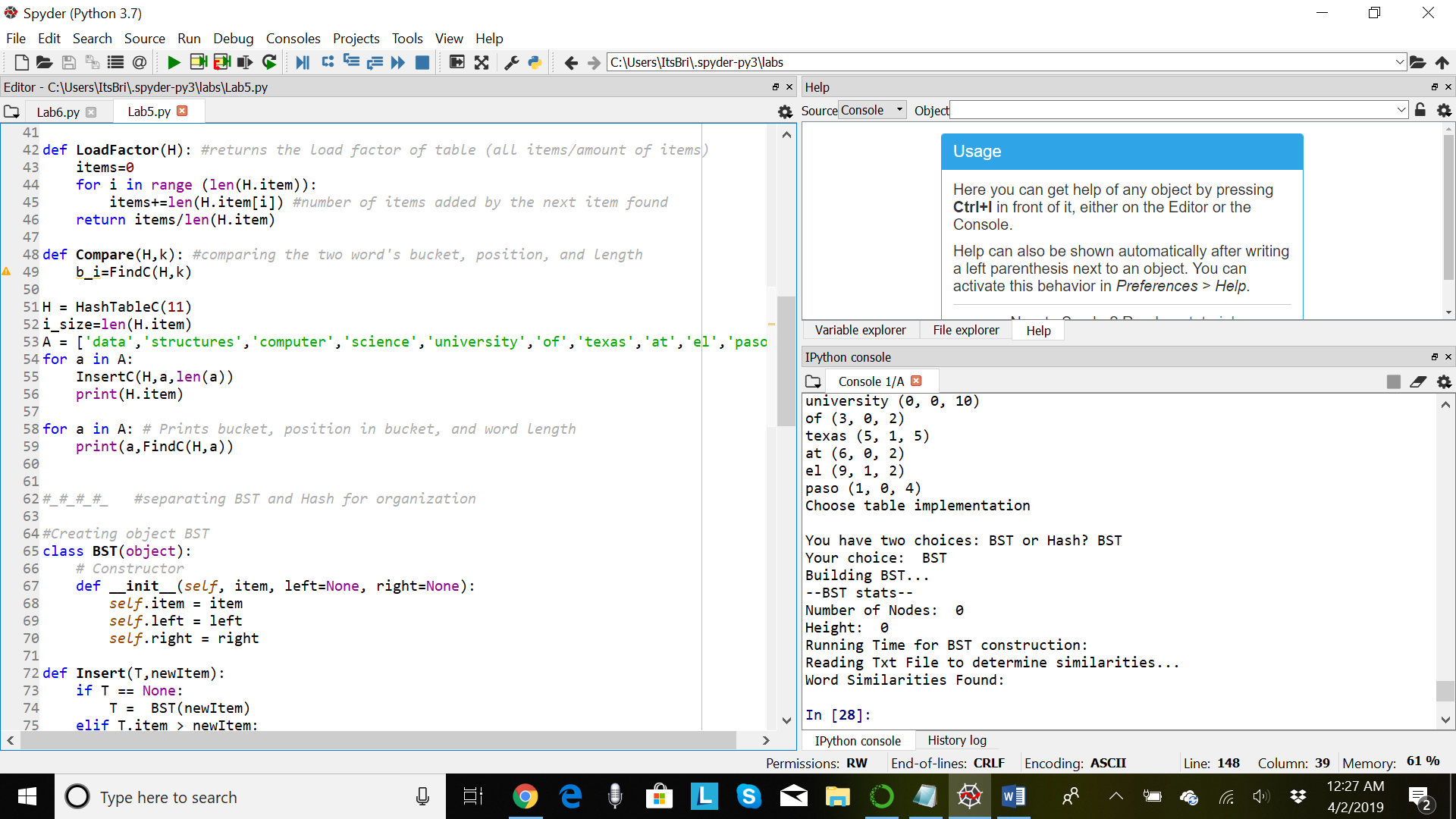
**Lab 5 Report**

This lab required us to use hash tables with chaining and compare that running time towards a Binary Search Tree to see which is the faster method of processing the similarities between two words.

My attempt to solve this problem was to implement both methods of comparing two words and having them each read from the same text file to compare their running times after. Unfortunately, I wasn’t able to finish. I first started with setting up my program with the given code Dr. Fuentes provided on his website. Creating classes for object Hash Tables and BSTs, separated to keep them organized. Then, after those were finished, I moved onto the printing information that would show and prompt the user to give feedback on which method they wanted to see first, as shown as an example in the worksheet given online. I created each line of printing each “stat” for either the hash table or the BST, organizing the print commands under an if-statement if the user chose either “Hash” or “BST” to see implemented first. Then, for each of those prints, I moved onto creating the methods under the corresponding objects of either Hash or BST to print with each line, such as “Height” of the BST and “initial size” of the hash table. This is where I was having issues even understanding where to move on next, as in, how would I create a method to compare the two words given from the text file to be used under BST and Hash. By the time I finished the easier methods to get by, it was time to turn this in to get full credit. I couldn’t afford lost points on this lab (since I didn’t make enough time for it), and external reasons outside of class as to why I didn’t have enough time.

The experiments I chose for testing the methods I had was changing the numbers of how many nodes were input into the BST or changing the size of the Hash Table to see if it would print out the same number. Other experiments weren’t able to be tested since I wasn’t able to finish.

Running times couldn’t be completed either, for this reason.

What I learned from this project is organizing my code better when it is a more complex code for me to learn, and to review method from old labs before (practicing) in order to make future project a bit easier.

**Appendix**:

#creating object Hash

class HashTableC(object):

# Builds a hash table of size 'size'

# Item is a list of (initially empty) lists

# Constructor

def \_\_init\_\_(self,size):

self.item = []

for i in range(size):

self.item.append([])

#given methods for Hash

def InsertC(H,k,l):

# Inserts k in appropriate bucket (list)

# Does nothing if k is already in the table

b = h(k,len(H.item))

H.item[b].append([k,l])

def FindC(H,k):

# Returns bucket (b) and index (i)

# If k is not in table, i == -1

b = h(k,len(H.item))

for i in range(len(H.item[b])):

if H.item[b][i][0] == k:

return b, i, H.item[b][i][1]

return b, -1, -1

def h(s,n):

r = 0

for c in s:

r = (r\*n + ord(c))% n

return r

def LoadFactor(H): #returns the load factor of table (all items/amount of items)

items=0

for i in range (len(H.item)):

items+=len(H.item[i]) #number of items added by the next item found

return items/len(H.item)

def Compare(H,k): #comparing the two word's bucket, position, and length

b\_i=FindC(H,k)

H = HashTableC(11)

i\_size=len(H.item)

A = ['data','structures','computer','science','university','of','texas','at','el','paso']

for a in A:

InsertC(H,a,len(a))

print(H.item)

for a in A: # Prints bucket, position in bucket, and word length

print(a,FindC(H,a))

#\_#\_#\_#\_ #separating BST and Hash for organization

#Creating object BST

class BST(object):

# Constructor

def \_\_init\_\_(self, item, left=None, right=None):

self.item = item

self.left = left

self.right = right

def Insert(T,newItem):

if T == None:

T = BST(newItem)

elif T.item > newItem:

T.left = Insert(T.left,newItem)

else:

T.right = Insert(T.right,newItem)

return T

def Find(T,k):

# Returns the address of k in BST, or None if k is not in the tree

if T is None or T.item == k:

return T

if T.item<k:

return Find(T.right,k)

return Find(T.left,k)

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

def height(T):

if T is None:

return 0

if T.item==T.left and T.item==T.right: #if there's a left and right child

return 1+max(height(T.left),height(T.right))

if T.item==T.left: #just left child

return 1+height(T.left)

if T.item==T.right:#just right child

return 1+height(T.right)

def numNodes(T):

count=1 #starting at 1 since this'll be counted for root

if T is None:

return 0

while T.right!=None: #loop to go through each right/left children and add to count

count += numNodes(T.right)

while T.left!=None:

count+= numNodes(T.left)

return count

# Code to test the functions above

T = None

A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]

for a in A:

T = Insert(T,a)

#first prompts user to choose BST or Hash Method

print('Choose table implementation')

question=input('You have two choices: BST or Hash? ')

if question=='Hash':

print('Your choice: ',question)

print('Building Hash Table...')

print('--Hash Table stats--')

print('Initial table size: ',i\_size)

print('Final Table size: ')

print('Load Factor: ',LoadFactor(H))

print('Percentage of Empty Lists: ')

print('Standard deviation of the lengths of lists: ')

print('Reading Txt File to determine similarities...')

print('Word Similarities Found: ')

if question=='BST':

print('Your choice: ',question)

print('Building BST...')

print('--BST stats--')

print('Number of Nodes: ',numNodes(T))

print('Height: ',height(T))

print('Running Time for BST construction: ')

print('Reading Txt File to determine similarities...')

print('Word Similarities Found: ')

Honesty Statement:

Academic dishonesty includes but is not limited to cheating, plagiarism and collusion. Cheating may involve

copying from or providing information to another student, possessing unauthorized materials during a test, or

falsifying data (for example program outputs) in laboratory reports. Plagiarism occurs when someone

represents the work or ideas of another person as his/her own. Collusion involves collaborating with another

person to commit an academically dishonest act. Professors are required to - and will - report academic

dishonesty and any other violation of the Standards of Conduct to the Dean of Students.

I hereby state hereby this code is mine and mine alone.