Lab 04 report

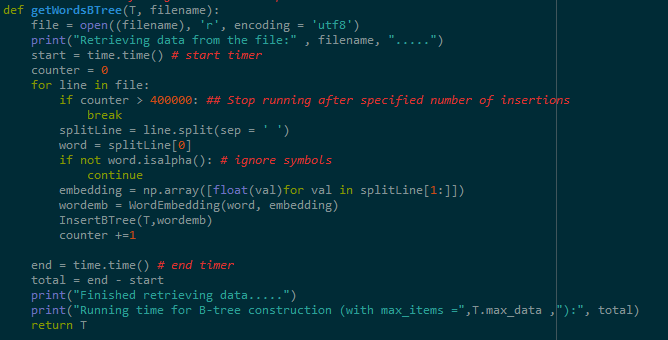
In this lab we are asked to compare running times of two data structures used to derive word embeddings to enable the comparison of two given words. The two data structures we are asked to use are a b-tree and a binary search tree. The first step in the implementation is to ask the user to choose between a binary search tree and a b-tree.

To implement the first step we must first read the ‘glove.6B.50d.txt’ file and store each word along with its embedding in both tree data structures. To do this we must implement the word embedding object which will help with the storage of each structure in the tree. To insert into each tree I implemented the insert function for both trees using the insert functions and the get words functions which read the files and store all of the words along with the embeddings using the word embedding object

The following is the function used to read the words from the large word file with the embedding for a binary search treeA screenshot of a cell phone

Description automatically generated

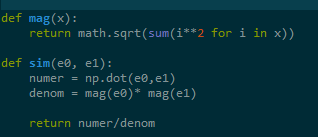
The next function is very similar to the previous but it is implemented for a b-tree



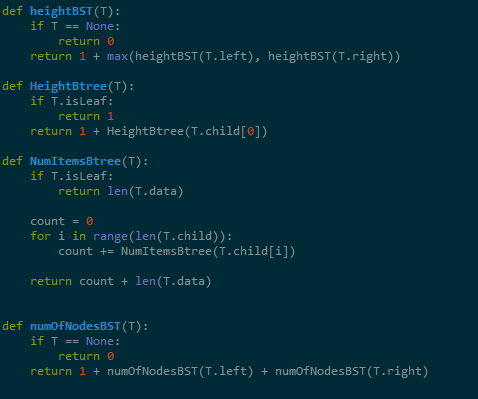
Notice that both are very similar but they both need to be inserted differently using each one of the data structures insert function

The next step is to create another function which will read a different file and will compare the words and their embeddings using the similarity function.

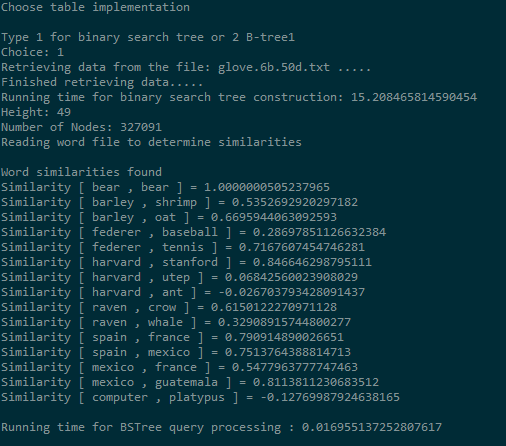
The similarity function was implemented using the equation that was given to us and had to be done using a different mag function which iterates through the list to find the denominator of the function given to us



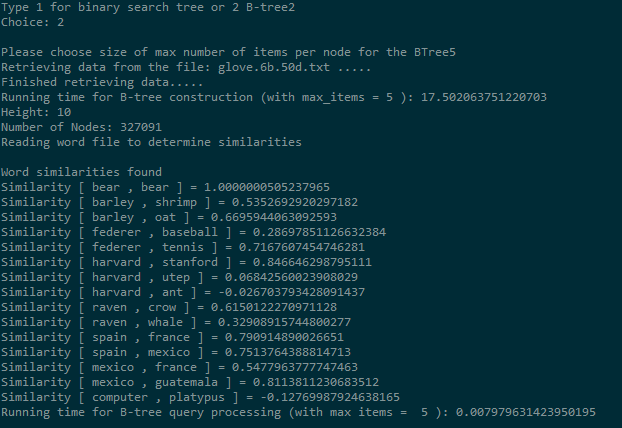
After the comparisons are done for each of the data structures and all the words, we are asked to implement the tree status for both implementations. We are asked to find the height of the trees, the number of nodes and the running time for binary search tree construction. The following image is the implementation of the tree status implementations. They are done recursively and are useful because they are used with the Word Embedding object and use integers



The following is an implantation of the two data structures being implemented with almost all the words in the file



The previous is a screenshot of the bst implementation

****

The previous image is the implantation of the b-tree with a maximum of 5 items per node

**Source Code**

#Lab04

#Daniel Solorzano

#Data Structures fall 2019

import numpy as np

import time

import math

class BTree(object):

# Constructor

def \_\_init\_\_(self, data, child=[], isLeaf=True, max\_data=5):

self.data = data #array

self.child = child

self.isLeaf = isLeaf

if max\_data <3: #max\_data must be odd and greater or equal to 3

max\_data = 3

if max\_data%2 == 0: #max\_data must be odd and greater or equal to 3

max\_data +=1

self.max\_data = max\_data

def InsertBTree(T, wordemb):

if not IsFull(T):

InsertInternal(T, wordemb)

else: #if full, split

m, l, r = Split(T)

T.data =[m]

T.child = [l,r]

T.isLeaf = False

k = FindChild(T,wordemb)

InsertInternal(T.child[k],wordemb)

def InsertInternal(T,wordemb):

# T cannot be Full

if T.isLeaf:

InsertLeaf(T,wordemb)

else:

k = FindChild(T,wordemb)

if IsFull(T.child[k]):

m, l, r = Split(T.child[k])

T.data.insert(k,m)

T.child[k] = l

T.child.insert(k+1,r)

k = FindChild(T,wordemb)

InsertInternal(T.child[k],wordemb)

def Split(T):

#print('Splitting')

#PrintNode(T)

mid = T.max\_data//2

if T.isLeaf:

leftChild = BTree(T.data[:mid],max\_data=T.max\_data)

rightChild = BTree(T.data[mid+1:],max\_data=T.max\_data)

else:

leftChild = BTree(T.data[:mid],T.child[:mid+1],T.isLeaf,max\_data=T.max\_data)

rightChild = BTree(T.data[mid+1:],T.child[mid+1:],T.isLeaf,max\_data=T.max\_data)

return T.data[mid], leftChild, rightChild

def InsertLeaf(T,i):

T.data.append(i)

T.data.sort(key=lambda obj: obj.word)

def IsFull(T):

return len(T.data) >= T.max\_data

def FindChild(T, k):

# Determines value of c, such that k must be in subtree T.child[c], if k is in the BTree

for i in range(len(T.data)):

if k.word < T.data[i].word:

return i

return len(T.data)

def FindChild2(T, k):

# Determines value of c, such that k must be in subtree T.child[c], if k is in the BTree

for i in range(len(T.data)):

if k < T.data[i].word:

return i

return len(T.data)

## Retrieve words from glove.txt and imput them into a BST

def getWordsBTree(T, filename):

file = open((filename), 'r', encoding = 'utf8')

print("Retrieving data from the file:" , filename, ".....")

start = time.time() # start timer

counter = 0

for line in file:

if counter > 400000: ## Stop running after specified number of insertions

break

splitLine = line.split(sep = ' ')

word = splitLine[0]

if not word.isalpha(): # ignore symbols

continue

embedding = np.array([float(val)for val in splitLine[1:]])

wordemb = WordEmbedding(word, embedding)

InsertBTree(T,wordemb)

counter +=1

end = time.time() # end timer

total = end - start

print("Finished retrieving data.....")

print("Running time for B-tree construction (with max\_items =",T.max\_data ,"):", total)

return T

def get2WordsBTree(root, file):

print("Reading word file to determine similarities")

print("\nWord similarities found")

start = time.time()

file = open(file, 'r', encoding = 'utf8')

for line in file:

splitLine = line.strip().split(sep = ' ')

word1 = splitLine[0]

word2 = splitLine[1]

r1 = retrieveBTree(root, word1)

r2 = retrieveBTree(root, word2)

#print(r1,r2)

print("Similarity [", word1,",", word2,"] =", sim(r1, r2))

end = time.time() # end timer

total = end - start

print("Running time for B-tree query processing (with max items = ", root.max\_data, "):", total)

def retrieveBTree(T, key):

# Returns node where k is, or None if k is not in the tree

for i in T.data:

#print("key:",key,"i:", i.word)

if key == i.word:

#print("here--------------------")

return i.emb

#if key in T.data:

#return T.word

if T.isLeaf:

#print("NONE--------------------------")

return None

return retrieveBTree(T.child[FindChild2(T,key)], key)

def Print(T):

# Prints data in tree in ascending order

if T.isLeaf:

for t in T.data:

print(t.word,end=' ')

else:

for i in range(len(T.data)):

Print(T.child[i])

print(T.data[i].word,end=' ')

Print(T.child[len(T.data)])

def PrintD(T,space):

# Prints data and structure of B-tree

if T.isLeaf:

for i in range(len(T.data)-1,-1,-1):

print(space,T.data[i].word)

else:

PrintD(T.child[len(T.data)],space+' ')

for i in range(len(T.data)-1,-1,-1):

print(space,T.data[i].word)

PrintD(T.child[i],space+' ')

###################################################################################################

## BST Implementation

class WordEmbedding(object):

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

# word must be a string, embedding can be a list or and array of ints or floats

self.word = word

self.emb = np.array(embedding, dtype=np.float32) # For Lab 4, len(embedding=50)

self.left = left

self.right = right

def InsertBST(T, word, embedding):

if T == None:

T = WordEmbedding(word, embedding)

elif T.word > word:

T.left = InsertBST(T.left, word, embedding)

else:

T.right = InsertBST(T.right, word, embedding)

return T

def get2WordsBST(root, file):

print("Reading word file to determine similarities")

print("\nWord similarities found")

start = time.time()

file = open(file, 'r', encoding = 'utf8')

for line in file:

splitLine = line.strip().split(sep = ' ')

word1 = splitLine[0]

word2 = splitLine[1]

r1 = retrieveBST(root, word1)

r2 = retrieveBST(root, word2)

print("Similarity [", word1,",", word2,"] =", sim(r1, r2))

end = time.time() # end timer

total = end - start

print()

print("Running time for BSTree query processing :", total)

def mag(x):

return math.sqrt(sum(i\*\*2 for i in x))

def sim(e0, e1):

numer = np.dot(e0,e1)

denom = mag(e0)\* mag(e1)

return numer/denom

## Retrieve words from glove.txt and imput them into a BST

def getWordsBST(filename):

file = open((filename), 'r', encoding = 'utf8')

print("Retrieving data from the file:" , filename, ".....")

start = time.time() # start timer

T = None

counter = 0

for line in file:

if counter > 400000: ## Stop running after specified number of insertions

break

splitLine = line.split(sep = ' ')

word = splitLine[0]

if not word.isalpha(): # ignore symbols

continue

embedding = np.array([float(val)for val in splitLine[1:]])

T = InsertBST(T, word, embedding)

counter +=1

end = time.time() # end timer

total = end - start

print("Finished retrieving data.....")

print("Running time for binary search tree construction:", total)

return T

def ShowBST(T,ind):

# Display rotated BST in text form

if T is not None:

ShowBST(T.right,ind+' ')

print(ind, T.word)

ShowBST(T.left,ind+' ')

def retrieveBST(root, word):

if root.word == word:

#print(root.word, root.emb)

return root.emb

else:

if root.word > word:

return retrieveBST(root.left, word)

else:

return retrieveBST(root.right, word)

def heightBST(T):

if T == None:

return 0

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

def HeightBtree(T):

if T.isLeaf:

return 1

return 1 + HeightBtree(T.child[0])

def NumItemsBtree(T):

if T.isLeaf:

return len(T.data)

count = 0

for i in range(len(T.child)):

count += NumItemsBtree(T.child[i])

return count + len(T.data)

def numOfNodesBST(T):

if T == None:

return 0

return 1 + numOfNodesBST(T.left) + numOfNodesBST(T.right)

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

print("Choose table implementation")

choice = input("Type 1 for binary search tree or 2 B-tree")

print("Choice:", choice)

if choice == '1':

bst = getWordsBST("glove.6b.50d.txt")

print("Height:", heightBST(bst))

print("Number of Nodes:", numOfNodesBST(bst))

get2WordsBST(bst, "words.txt")

else:

maxsize = input("Please choose size of max number of items per node for the BTree")

btree = BTree([],max\_data=int(maxsize))

getWordsBTree(btree, "glove.6b.50d.txt")

print("Height:", HeightBtree(btree))

print("Number of Nodes:", NumItemsBtree(btree))

#Print(btree)

get2WordsBTree(btree, "words.txt")

“I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.”

X\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_