Ivan Perez
80602918
Lab 5 Report

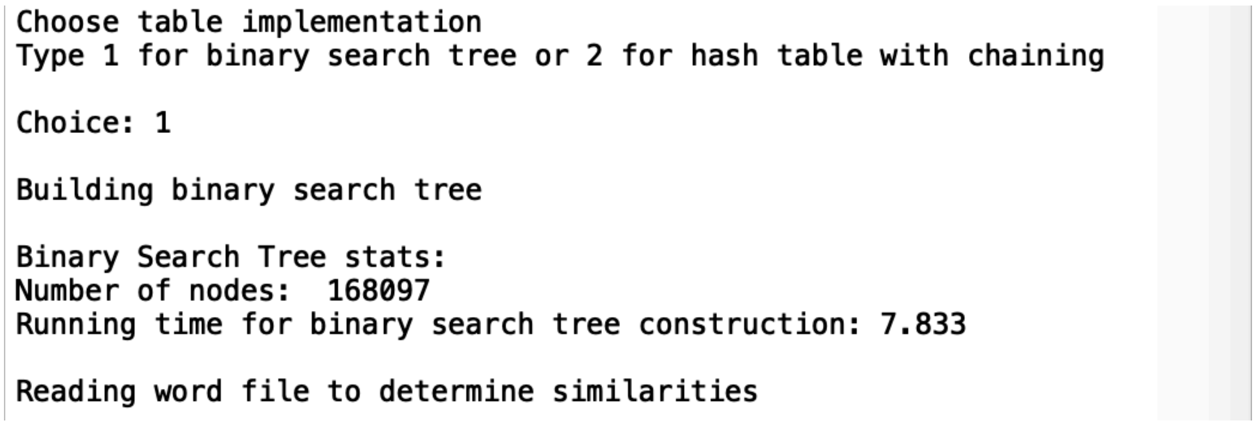
Introduction

This lab requires the implementation of a binary search tree and hash tables. A file is supposed to be read into one of the table options. This file contains a word and its embedding(vector quantities) which we'll later use to compare word similarities. The user is supposed to choose which table implementation they would like to use and its supposed to display the specs of the table implementation chosen.

Proposed solution design and implementation

Python has an alphabetical string comparison so implementing the binary search tree was not too difficult. The insertion tool would place the elements exactly where they needed to go and finding them worked the same way. Implementing the hash table was more difficult. I was not able to do it but what came to mind was to place the elements alphabetically as well.

Experimental results



```
Choose table implementation
Type 1 for binary search tree or 2 for hash table with chaining

Choice: 1

Building binary search tree

Binary Search Tree stats:
Number of nodes:  168097
Running time for binary search tree construction: 7.833

Reading word file to determine similarities
```

I was able to construct the binary search tree but receiving the results to determine similarities was not working me for me after the first time.

Conclusions

This lab's hash table was difficult to implement and difficult to determine how to store the embeddings. It was interesting to see the structure that Alexa and Siri uses to determine how to retrieve information so quickly and it brought a real world element to spark more curiosity and enjoyment in doing this lab.

Appendix

```
################################################################################
#
#
# Course: CS2302 (MW - 1:30pm)
# Author: Ivan Perez
# Assignment: Lab 5
# Instructor: Olac Fuentes
# T.A.: Anindita Nath
# Last Modified: May 5, 2019
# Purpose: Asks user for input to determine which table implementaion they would
# like. Depending on which they choose, embeddings will be stored through
# a binary search tree or hashtable. This code contains methods for the object
# binary search tree. Once words are stored, they determine similarity through
# a secondary file. The runtimes and specs of the table and searching are
# displayed.
#
#######################################################################################
#
import numpy as np
import math
import time

############ BST object methods ####################
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[0] > newItem[0]:
        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[0] == k:
        return T
    if T.item[0]<k:
        return Find(T.right,k)
    return Find(T.left,k)

############### assignment methods #################
```

```
def createEmbedding(array):
    a = np.zeros(50, dtype=float)
    for i in range(len(array)-1):
        a[i] = float(array[i+1])
        # returns list that contains word and array of embeddings
    return [array[0], a]

def readFileToBST():
    print('Building binary search tree')
    # file reading
    file = open('glove.6B.50d.txt','r')
    line = file.readline()
    line = line.split(' ')
    embedding = createEmbedding(line)
    bst = BST(embedding) # bst creation
    nodes = 1 # node counter
    while line:
        line = file.readline()
        line = line.split(' ')
        embedding = createEmbedding(line)
        if embedding[0].isalnum():
            Insert(bst, embedding)
            nodes += 1
        line = file.readline()

    file.close()
    return bst, nodes

def similarity(w0, w1):
    dot = 0
    u = 0
    v = 0
    for i in range(len(w0)):
        dot += (w0[i]*w1[i])
        u += (w0[i]*w0[i])
        v += (w1[i]*w1[i])
    u = math.sqrt(u)
    v = math.sqrt(v)
    return round((dot/(u*v)), 4)

def readPairs(bst):
    print()
    print('Reading word file to determine similarities')
    print()
    w = []
    result = []
    file = open('pairs.txt','r')
    line = file.readline()
    Start = time.time()
    while line:
        line = line.split(' ')
        line[1] = line[1].rstrip('\n')
        w += [line]
        line = file.readline()

    for i in range(len(w)):
        w0 = Find(bst, w[i][0])
        w1 = Find(bst, w[i][1])
        result.append(similarity(w0.item[1],w1.item[1]))
    End = time.time()

    for i in range(len(w)):
        print('Similarity ', w[i],' = ',result[i] )
    return End - Start


############### Main Method ##################
print('Choose table implementation')
print('Type 1 for binary search tree or 2 for hash table with chaining')
i = int(input('Choice: '))

if i == 1:
    #Binary search tree choice
    print()
    Start = time.time()
    bst, nodes = readFileToBST()
    End = time.time()
    print()
    print('Binary Search Tree stats:')
    print('Number of nodes: ', nodes)
    print('Running time for binary search tree construction:', round((End - Start),3))
    run = readPairs(bst)
    print('Running time for binary search tree query processing: ', round(run),3)
elif i == 2:
    # hashtable choice
    print('Currently not available')
else:
    print('invalid response')
```

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.