Maria F. Corona Ortega
ID: 80560734
E-Mail: mfcoronaortega@miners.utep.edu
CS 2302 | MW 1:30-2:50 am
Instructor: Dr. Olac Fuaentes
TA: Anindita Nath

CS 2302 Lab 5 Report

Introduction

In this lab we explore Natural Language Processing using two structures which are binary search trees and hash tables with chaining. Using the website provided by the lab prompt we are to retrieve a file containing words and embeddings and effectively sort these using BST and hash tables while ignoring special characters. Lastly we are to use the newly found sorted of these words to quickly compare them and their embeddings.

Proposed Solution

Both implementations require a prompt for the user to choose the desired implementation. Both implementations read a text file line by line and populate both the read word and embedding into a list with the first element being the word and the second element is a numpy array containing the embedding. For both of these splits we traverse the text file line by line, and using a temporary array we split and parse all values. Using an if statement we check for special characters those are skipped. Into the list we append the word and bumpy array.

- **Task 1 (Binary Search Tree implementation):** In the BST implementation we use the original code and classes provided on the class website and insert in the form of a sorted binary search tree.
- **Task 2 (Hash Table with chaining implementation):** In the hash table implementation we use the original code and classes provided on the class website and insert in the form of a hash table

Methods

In this potion of the report we will only cover methods added to any provided code on the class website. Those methods will be listed below but will mention if they use any of the methods provided.

MyBST: In this method we begin by opening and reading the given file. For testing purposes we did use a reduced version of the original file. We also initialize a variable for the node count and a timer to check for runtime. Using a for loop we read the file line by line split and parse as described on the proposed solution. After this we check the first letter or character in each word and if it is not a

special character we append into list *L.* We then use the *insert* method from the BST code, raise the node count. Once the loop is over and we are done traversing all lines on the file we stop the timer and print stats.

FindHeight: I implemented this method in order to calculate the height of the BST. This method traverses the tree's left branches and return a counter of the levels traversed in the tree.

MyHT: In this method we begin by opening and reading the given file. For testing purposes we did use a reduced version of the original file. We also initialize a variable for the node count and a timer to check for runtime. Using a for loop we read the file line by line split and parse as described on the proposed solution. After this we check the first letter or character in each word and if it is not a special character we append into list *L.* We then use the *InsertC* method from the Hash table with chaining code. In this same loop we add to the count of the total number of items from the *H* or hash table class. Once the loop is over and we are done traversing all lines on the file we stop the timer and print stats.

load_factor: I implemented this method in order to calculate the load factor of the hash table. This method receives the table, pulls *H.num_items* and length of *H.item* returning the load factor.

Experimental Results

Binary Search Tree

	Running Rime	# of Nodes	Tree Height
10 items	0.0003895739937433973 s	6	4
20 items	0.0011876650096382946 s	14	4
50 items	0.008922490989789367 s	41	5
100 items	0.012382077999063767 s	87	6

Hash Table

	Running Time	Size	Load Facor
10 items	0.0003118650056421757 s	10	0.6

	Running Time	Size	Load Facor
20 items	0.00040882299072109163 s	20	0.75
50 items	0.00299809699936304 24 s	50	0.82
100 items	0.00274875000468455 26 s	100	0.87

Conclusion

In conclusion on this lab we learned to problem solve and edit data in order to fit into an efficient data structure and with this improve efficiency for large data files. Something that I would like to learn and implement in the future is the comparison of strings. Due to time I was unable to decipher the prompted string comparison implementation but I plan to discuss this with my TA during our demo session.

Appendix

```
#Course: CS 2302 Data Structures | Spring 2019
#Author: Maria Fernanda Corona Ortega
#Assignment: Lab 5
#Instructor: Olac Fuentes
#Purpose of Code: The purpose of this code is to explore the
implementation
#of binary search trees and Hash tables in order ot explore what is
#Natural Language Processing
#Last Modification: 4/1/2019

# Implementation of hash tables with chaining using strings
import numpy as np
import timeit

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 Delete(T,del_item):
    if T is not None:
        if del_item < T.item:
            T.left = Delete(T.left,del_item)
        elif del_item > T.item:
            T.right = Delete(T.right,del_item)
        else:  # del_item == T.item
            if T.left is None and T.right is None: # T is a leaf, just
remove it
                T = None
            elif T.left is None: # T has one child, replace it by
existing child
                T = T.right
            elif T.right is None:
                T = T.left
            else: # T has two chldren. Replace T by its successor,
delete successor
                m = Smallest(T.right)
                T.item = m.item
                T.right = Delete(T.right,m.item)
    return T

def InOrder(T):
```

```
    # Prints items in BST in ascending order
    if T is not None:
        InOrder(T.left)
        print(T.item,end = ' ')
        print()
        InOrder(T.right)

def InOrderD(T,space):
    # Prints items and structure of BST
    if T is not None:
        InOrderD(T.right,space+'        ')
        print(space,T.item)
        InOrderD(T.left,space+'        ')

def SmallestL(T):
    # Returns smallest item in BST. Returns None if T is None
    if T is None:
        return None
    while T.left is not None:
        T = T.left
    return T

def Smallest(T):
    # Returns smallest item in BST. Error if T is None
    if T.left is None:
        return T
    else:
        return Smallest(T.left)

def Largest(T):
    if T.right is None:
        return T
    else:
        return Largest(T.right)

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 FindHeight(T):
    ct = 1
    if T.left is None:
        return ct
    else:
        ct += FindHeight(T.left)
    return ct
```

```
#########################################################################

def MyBST():
    file = open("test_text.txt",'r')
    T = None
    NodeCT = 0
    start = timeit.default_timer()
    for line in file:
        L = []
        a = np.array([])
        tmp = []
        tmp = line.split(' ',1)
        a = tmp[1].split(' ')
        for i in a:
            float(i)
        if ord(tmp[0][0]) > 65:
            L.append(tmp[0])
            L.append(a)
#            print(L[:])
            T = Insert(T,L)
            NodeCT += 1
    stop = timeit.default_timer()

    print('Binary Search Tree stats:')
    print('Number of nodes:', NodeCT)
    print('Height:', FindHeight(T))
    print('Running Time of Binary search tree construction: ', stop -
start,'s\n')
#    print('Reading word file to determine similarities\n')

#    file2 = open("simword_test.txt",'r')
##    print(file2.read())
#    L = []
#
#    print('Word similarities found:\n')
#    start2 = timeit.default_timer()
#    for line in file2:
#        tmp = []
#        tmp = line.split(' ')
#        sim_val = sim(tmp[0],tmp[1])
#
#        print('- Similarity',tmp[0],tmp[1], sim_val)
#        L.append(tmp)
#    stop2 = timeit.default_timer()
#    print(L[:])
#    print(InOrder(T))
#    print(InOrderD(T,' '))
#    print('Running time for binary search tree query processing:',
stop2 - start2,'s\n')


#def sim(w1, w2):
#        return 0

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

class HashTableC(object):
    # Builds a hash table of size 'size'
```

```
    # Item is a list of (initially empty) lists
    # Constructor
    def __init__(self,size, num_items = 0 ):
        self.item = []
        self.num_items = num_items
        for i in range(size):
            self.item.append([])

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*255 + ord(c))% n
    return r

def load_factor(H):
    return H.num_items / len(H.item)

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

def MyHT():
    file = open("test_text.txt",'r')
#    print("Output of Readline function is :")
    size = 13
    H = HashTableC(size)
    start = timeit.default_timer()
    for line in file:
        L = []
        a = np.array([])
        tmp = []
        tmp = line.split(' ',1)
        a = tmp[1].split(' ')
        for i in a:
            float(i)
        if ord(tmp[0][0]) > 65:
            L.append(tmp[0])
            L.append(a)
#            print(L[:])
            InsertC(H,L[0],len(L))
            H.num_items += 1
#            print(H.item)
#            print(L[0],FindC(H,L[0]))
    stop = timeit.default_timer()
```

```
    print('Hash table stats:')
#    print('Initial table size:')
    print('Final table size:', len(H.item))
    print('Load factor:', load_factor(H))
#    print('Percentage of empty lists:')
#    print('Standard deviation of the lengths of the lists:')
    print('Running Time of Hash Table construction: ', stop -
start,'s\n')

#    print('Reading word file to determine similarities\n')

#    file2 = open("simword_test.txt",'r')
##    print(file2.read())
#    L = []
#
#    print('Word similarities found:\n')
#    start2 = timeit.default_timer()
#    for line in file2:
#        tmp = []
#        tmp = line.split(' ')
#        sim_val = sim(tmp[0],tmp[1])
#
#        print('- Similarity',tmp[0],tmp[1], sim_val)
#        L.append(tmp)
#    stop2 = timeit.default_timer()
#    print(L[:])
#    print(InOrder(T))
#    print(InOrderD(T,' '))
#    print('Running time for binary search tree query processing:',
stop2 - start2,'s\n')

######################PROGRAM IMPLEMENTATION###################

user = input('Choose table implementation \nType 1 for binary search
tree or 2 for hash table with chaining\n')

if user is '1':
    print('Choice:', user, '\n\nBuilding binary search tree\n\n')
    MyBST()
if user is '2':
    print('Choice:', user, '\n\nBuilding hash table with
chaining\n\nHash table stats:\n')
    MyHT()
```

Academic Honesty Certification

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.

Maria F. Corona Ortega

09/09/2017