# Lab Report 5

# Cesar Lopez

# CS2302 10:30- 12:00 – Olac Fuentes

# Introduction

The problem presented to us is to store information from a text which has the values of a word that is used to create a search function, similar to the one’s that cortana and alex use. The information is located in a text file, and when opened looks to be full with only numbers. Our job is to store the information into either a hash table or a binary search tree, which will organize, and allow us to access the information. Along with getting fast results from our code, we also have to do comparisons of words using the BST or hash table and do comparisons between 2 words.

# Proposed Solution

The solution which I chose was to import both classes of BST and Hash tables into a file and allow the user to choose whichever option they felt would be best. After choosing, we iterate through the file by line and create an array which we use to store information by line. The array position 0 will always hold the word, and the rest of the array holds numbers. The next thing I chose to do was to use the insert function that was provided to us, by inserting the hash, the word, and the numbers, all separate. For BST I just inserted the list that held the list and the numbers and changed return methods to return node[0] which is the string instead of a whole node, that would be a 2d array. After construction, I began the comparison portion which would create an array that held 2 words per line and launch a comparison method. The reason we only checked for 2 words was because we were only doing the similarity between 2 words. After calling similarities with either hash or bst, word1, and word2, we began doing the cosine. First, we used a search function that would search the array for words 1 and 2 and would return the array with the numbers that were a part with the word. After getting those arrays we got the dot product of the numbers. With the dot product, that would give us our top portion of the formula, and next we got our magnitude. For magnitude, we iterated through the array or word 1 and 2, and we would square the numbers and then add them till the end of the list, after that we got the square root. Once we got dot product and magnitude for word 1 and 2, we divided dot product by magnitudes of word 1 \* word 2. Once we finished that, it would return the number which would tell us how similar 2 words were in the Hash or BST.

# Experimental Results

A big problem that I faced with this lab was an issue with word 2 when doing comparisons. The find function would not be returning the proper array, and would instead return a 2d array. The reason this happened was because I was using split when searching the text file, and when I did comparison, word 2 would have a space at the end of it. After debugging for a couple hours, I found the issue was by splitting with a space, and after removing that space, my code worked fine.

# Conclusion

I learned to always be aware of the little things that you are doing because they do cause major issues. I also learned how a simple feature that cortana and alexa use is not too difficult and could probably replicate it in the future. This lab overall taught me something fun and new that I will definitely be messing around with after the semester is over.

# Appendix

1. # Implementation of hash tables with chaining using strings
2. #Lab 5
3. #Cesar Lopez
4. #Professor - Olac Fuentes
5. #TA - Anadita
6. #Class - 10:30 - 12:00
7. #We read information from a text file and did comparisons using cosine to find the similarity. We stored the information using
8. #hash tables or binary search trees to determine speeds.
10. **class** HashTableC(object):
11. # Builds a hash table of size 'size'
12. # Item is a list of (initially empty) lists
13. # Constructor
14. **def** \_\_init\_\_(self,size):
15. self.item = []
16. self.num\_items = 0#used to keep track of number of items, used for load factor
17. **for** i **in** range(size):
18. self.item.append([])
20. **def** InsertC(H,k,l):
21. # Inserts k in appropriate bucket (list)
22. # Does nothing if k is already in the table
23. H.num\_items += 1#used to update and check load factor
24. b = h(k,len(H.item))
25. **if** H.num\_items/len(H.item) >= 1:#checks load factor
26. **for** i **in** range(len(H.item)+1):#updates hash table length with empty nodes
27. H.item.append([])
28. H.item[b].append([k,l])
30. **def** FindC(H,k):
31. # Returns bucket (b) and index (i)
32. # If k is not in table, i == -1
33. b = h(k,len(H.item))
34. **for** i **in** range(len(H.item[b])):
35. **if** H.item[b][i][0] == k:
36. **return** H.item[b][i][1]#updated to return array of numbers instead of extra stuff
37. **return** -1#returns -1 like stated in comments above.
39. **def** h(s,n):
40. r = 0
41. **for** c **in** s:
42. r = (r\*n + ord(c))% n
43. **return** r
45. **def** dotProduct(w1, w2):#uses for loop to iterate the 2 arrays and gets do product
46. hold = 0
47. **for** i **in** range(len(w1)):
48. hold += float(w1[i])\*float(w2[i])
49. **return** hold
51. **def** magni(w):#gets out magnitude, \*\*2 is squared, and sqrt gets our square root
52. hold = 0
53. **for** i **in** range(len(w)):
54. hold += float(w[i]) \*\* 2
55. **return** math.sqrt(hold)
57. **def** sim(H, w1, w2):#here we determine our similarities. First we get do product, next we get magnitudes, and lastly divide dot product my both magnitudes
58. e1 = FindC(H, w1)
59. e2 = FindC(H, w2)
60. dp = dotProduct(e1, e2)
61. m1 = magni(e1)
62. m2 = magni(e2)
63. **return** dp/(m1\*m2)
65. **def** IsEmpty(H):#checks for our empty nodes in our hash table
66. count = 0
67. **for** i **in** range(len(H.item)):
68. **if** len(H.item[i]) <= 0:
69. count += 1
70. **return** count
72. # Code to implement a binary search tree
73. # Programmed by Olac Fuentes
74. # Last modified February 27, 2019
76. **class** BST(object):
77. # Constructor
78. **def** \_\_init\_\_(self, item, left=None, right=None):
79. self.item = item
80. self.left = left
81. self.right = right
83. **def** Insert(T,newItem):
84. **if** T == None:
85. T =  BST(newItem)
86. **elif** T.item > newItem:
87. T.left = Insert(T.left,newItem)
88. **else**:
89. T.right = Insert(T.right,newItem)
90. **return** T
92. **def** Delete(T,del\_item):
93. **if** T **is** **not** None:
94. **if** del\_item < T.item:
95. T.left = Delete(T.left,del\_item)
96. **elif** del\_item > T.item:
97. T.right = Delete(T.right,del\_item)
98. **else**:  # del\_item == T.item
99. **if** T.left **is** None **and** T.right **is** None: # T is a leaf, just remove it
100. T = None
101. **elif** T.left **is** None: # T has one child, replace it by existing child
102. T = T.right
103. **elif** T.right **is** None:
104. T = T.left
105. **else**: # T has two chldren. Replace T by its successor, delete successor
106. m = Smallest(T.right)
107. T.item = m.item
108. T.right = Delete(T.right,m.item)
109. **return** T
111. **def** InOrder(T):
112. # Prints items in BST in ascending order
113. **if** T **is** **not** None:
114. InOrder(T.left)
115. **print**(T.item,end = ' ')
116. InOrder(T.right)
118. **def** InOrderD(T,space):
119. # Prints items and structure of BST
120. **if** T **is** **not** None:
121. InOrderD(T.right,space+'   ')
122. **print**(space,T.item)
123. InOrderD(T.left,space+'   ')
125. **def** SmallestL(T):
126. # Returns smallest item in BST. Returns None if T is None
127. **if** T **is** None:
128. **return** None
129. **while** T.left **is** **not** None:
130. T = T.left
131. **return** T
133. **def** Smallest(T):
134. # Returns smallest item in BST. Error if T is None
135. **if** T.left **is** None:
136. **return** T
137. **else**:
138. **return** Smallest(T.left)
140. **def** Largest(T):
141. **if** T.right **is** None:
142. **return** T
143. **else**:
144. **return** Largest(T.right)
146. **def** Find(T,k):#updated to check T.item[0] instead of T.item, because T.item[0] is where our string is located
147. # Returns the address of k in BST, or None if k is not in the tree
148. **if** T **is** None **or** T.item[0] == k:
149. **return** T.item
150. **if** T.item[0]<k:
151. **return** Find(T.right,k)
152. **return** Find(T.left,k)
154. **def** NumNodes(T):#counts our number of nodes with our return statement, base case checks for empty nodes and returns 0 if found any
155. **if** T **is** None:
156. **return** 0
157. **return** NumNodes(T.left) + NumNodes(T.right)+1
159. **def** FindAndPrint(T,k):
160. f = Find(T,k)
161. **if** f **is** **not** None:
162. **print**(f.item,'found')
163. **else**:
164. **print**(k,'not found')
166. **def** HeightT(T):#we have 2 holds that checks if either is larger than the other, if one is larger, that gives us our larger end of the height, and we add up the larger.
167. hold = 0
168. hold1 = 0
169. **if** T **is** None:
170. **return** 0
171. hold += HeightT(T.left)
172. hold1 += HeightT(T.right)
173. **if**(hold > hold1):
174. **return** 1+hold
175. **else**:
176. **return** 1+hold1
178. **def** BSTDot(w1, w2):#dot product but for our bst, similar to hash one
179. i = 1
180. hold = 0
181. **while** i<len(w1):
182. hold+= float(w1[i])\*float(w2[i])
183. i +=1
184. **return** hold
186. **def** BSTMagni(w):#magnitude but for our bst, similar to hash one
187. i = 1
188. hold = 0
189. **while** i<len(w):
190. hold+=float(w[i])\*\*2
191. i += 1
192. **return** math.sqrt(hold)
194. **def** BSTSim(T, w1, w2):#comparison method for bst's, similar to hash one
195. e1 = Find(T, w1)
196. e2 = Find(T, w2)
197. dp = BSTDot(e1, e2)
198. m1 = BSTMagni(e1)
199. m2 = BSTMagni(e2)
200. **return** dp/(m1\*m2)

203. **import** math
204. **import** time
205. choice = input("Type 1 for Binary Search Tree, or 2 for Hash Table with chaining")
206. **print**("Your choice is: ",choice)
207. **if**(choice == '1'):
208. **print**("Your choice is: Binary Search Tree")
209. **print**()
210. **print**("Building Binary Search Tree..")
211. start = time.time()
212. T = None
213. f = open('glove.6B.50d.txt', encoding='utf-8')#opens file and we read it, store it into an array, and splice it to store into bst
214. **for** line **in** f:
215. a = line.split(' ')
216. T = Insert(T, a)
217. **print**("Binary Search Tree stats:")
218. **print**("Number of nodes: ", NumNodes(T))
219. **print**("Height of Tree: ", HeightT(T))
220. **print**("Running time for Binary Search Tree construction ", time.time()-start)
221. file = open('input.txt', encoding='utf-8')
222. **for** lin **in** file:
223. b = lin.split()
224. **print**("Similarity [",b[0],", ", b[1],"] =",end=" ")#we compare the line with 2 words in it
225. **print**(BSTSim(T, b[0], b[1]))
226. **elif**(choice == '2'):
227. **print**("Your choice is: Hash Table")
228. start = time.time()
229. **print**()
230. **print**("Building Hash Table...")
231. H = HashTableC(13)
232. f = open('glove.6B.50d.txt', encoding='utf-8')#opens file and we read it, store it into an array, and splice it to store into our hash
233. **for** line **in** f:
234. a = line.split(' ')
235. word = a[0]
236. embed = a[1:]
237. InsertC(H, word, embed)
238. **print**("Hash Table Stats:")
239. **print**("Initial Size: 13")
240. **print**("final size: ",len(H.item))
241. **print**("Load Factor: ", H.num\_items/len(H.item))
242. **print**("Percentage of Empty list", (IsEmpty(H)/len(H.item))\*100)
243. **print**("Standard Deviation of Lengths of list: ")
244. file = open('input.txt', encoding='utf-8')
245. **for** lin **in** file:
246. b = lin.split()
247. **print**("Similarity [",b[0],", ", b[1],"] =",end=" ")#we compare the line with 2 words in it
248. **print**(sim(H, b[0], b[1]))
249. **print**()
250. **print**("Time: ",time.time()-start)
251. **else**:
252. **print**("Wrong input, relaunch and try again.")

# Running times

I found it faster to use the hash than it was to use the BST because I felt like it is much easier to search through a hash than it would to search through a BST. The code also ran faster when constructing a Hash and doing comparisons Vs just building the BST. Word search did not differ too much on both BST and Hash, but I preferred working with hash tables.

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. C.L