**CS 2302 Data Structures**

**Spring 2019**

**Lab Report #5**

Due: November 1, 2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

For this lab we were asked to implement the same functionality as Lab 4 but this time we were to use Hash Tables. These hash tables were of the type of chaining or linear probing. The type of hash table is determined by the user. The difference in this lab is that the hash function is also determined by the user at the beginning of the program. These hash functions were given to us in the lab assignment. Aside from these differences all other functions remained the same since we needed to achieve the same outputs.

**Proposed Solution Design and Implementation**

**Part #1**

For Part 1, I was asked to implement and compare the following hash functions. All the functions have a modulus with n in which n represents the length of the table. The first function was to have the length of the string mod n. The second function takes the ascii value of the first character in the string mod n. The third function takes the product of the ascii values of the first and last characters in the string with mod n. The fourth function takes the sum of the ascii values of the characters in the string with mod n. The fifth function has a recursive formulation h(”,n) = 1; h(S,n) = (ord(s[0]) + 255\*h(s[1:],n)) with mod n and h being the recursive call to the fifth function. The ascii value is found using ord(c). The sixth function is of my choice and I decided to take the recursive formulation but slightly modify it. Instead the ascii value of the first character in the string is multiplied by 5 then added to the ascii value of the first character as well as added to the value of the recursive call to function 1 on the rest of the characters in the string. This value is then modded with n as well. A complication with these hash functions was implementing into the HashTable classes and having the desired output reached. The ability to complete this task was achieved by storing the call to the hash function in an array. In the distinctive classes for the hash tables there was an added attribute to their constructors which accounted for these hash functions. When the hash table is declared there are two attributes associated with one being the length of the file which was just read and the other being the hash function that was previously chosen by the user. The hash function is reached by calling the array which holds the different hash functions and then by calling on the index where the desired hash function is found. From this point Lab 5 becomes extremely similar with Lab 4 with the differences laying in the runtime for the type of hash function as well as the type of hash table.

**Experimental Results**

**﻿﻿﻿Building hashtable with chaining using option 1**

**Chaining hashtable stats:**

**Number of nodes: 20000**

**Running time for chaining 10.696313858032227**

**Word similarities found:**

**Similarity between bear and bear = 1.0000001**

**None and shrimp**

**None and None**

**Similarity between federer and baseball = 0.28697854**

**Similarity between federer and tennis = 0.71676075**

**Similarity between harvard and stanford = 0.8466463**

**harvard and None**

**Similarity between harvard and ant = -0.026703795**

**Similarity between raven and crow = 0.6150122**

**Similarity between raven and whale = 0.32908916**

**Similarity between spain and france = 0.7909149**

**Similarity between spain and mexico = 0.75137645**

**Similarity between mexico and france = 0.54779637**

**Similarity between mexico and guatemala = 0.81138104**

**computer and None**

**Running time for chaining of embedded words: 0.019846439361572266**

**﻿Building hashtable with chaining using option 2**

**Chaining hashtable stats:**

**Number of nodes: 20000**

**Running time for chaining 5.322410821914673**

**Word similarities found:**

**Similarity between bear and bear = 1.0000001**

**None and shrimp**

**None and None**

**Similarity between federer and baseball = 0.28697854**

**Similarity between federer and tennis = 0.71676075**

**Similarity between harvard and stanford = 0.8466463**

**harvard and None**

**Similarity between harvard and ant = -0.026703795**

**Similarity between raven and crow = 0.6150122**

**Similarity between raven and whale = 0.32908916**

**Similarity between spain and france = 0.7909149**

**Similarity between spain and mexico = 0.75137645**

**Similarity between mexico and france = 0.54779637**

**Similarity between mexico and guatemala = 0.81138104**

**computer and None**

**Running time for chaining of embedded words: 0.008975028991699219**

**﻿Building hashtable with chaining using option 3**

**Chaining hashtable stats:**

**Number of nodes: 20000**

**Running time for chaining 0.8532369136810303**

**Word similarities found:**

**Similarity between bear and bear = 1.0000001**

**None and shrimp**

**None and None**

**Similarity between federer and baseball = 0.28697854**

**Similarity between federer and tennis = 0.71676075**

**Similarity between harvard and stanford = 0.8466463**

**harvard and None**

**Similarity between harvard and ant = -0.026703795**

**Similarity between raven and crow = 0.6150122**

**Similarity between raven and whale = 0.32908916**

**Similarity between spain and france = 0.7909149**

**Similarity between spain and mexico = 0.75137645**

**Similarity between mexico and france = 0.54779637**

**Similarity between mexico and guatemala = 0.81138104**

**computer and None**

**Running time for chaining of embedded words: 0.0015034675598144531**

**﻿Building hashtable with chaining using option 4**

**Chaining hashtable stats:**

**Number of nodes: 20000**

**Running time for chaining 0.29982423782348633**

**Word similarities found:**

**Similarity between bear and bear = 1.0000001**

**None and shrimp**

**None and None**

**Similarity between federer and baseball = 0.28697854**

**Similarity between federer and tennis = 0.71676075**

**Similarity between harvard and stanford = 0.8466463**

**harvard and None**

**Similarity between harvard and ant = -0.026703795**

**Similarity between raven and crow = 0.6150122**

**Similarity between raven and whale = 0.32908916**

**Similarity between spain and france = 0.7909149**

**Similarity between spain and mexico = 0.75137645**

**Similarity between mexico and france = 0.54779637**

**Similarity between mexico and guatemala = 0.81138104**

**computer and None**

**Running time for chaining of embedded words: 0.0008137226104736328**

**﻿Building hashtable with chaining using option 5**

**Chaining hashtable stats:**

**Number of nodes: 20000**

**Running time for chaining 0.1716461181640625**

**Word similarities found:**

**Similarity between bear and bear = 1.0000001**

**None and shrimp**

**None and None**

**Similarity between federer and baseball = 0.28697854**

**Similarity between federer and tennis = 0.71676075**

**Similarity between harvard and stanford = 0.8466463**

**harvard and None**

**Similarity between harvard and ant = -0.026703795**

**Similarity between raven and crow = 0.6150122**

**Similarity between raven and whale = 0.32908916**

**Similarity between spain and france = 0.7909149**

**Similarity between spain and mexico = 0.75137645**

**Similarity between mexico and france = 0.54779637**

**Similarity between mexico and guatemala = 0.81138104**

**computer and None**

**Running time for chaining of embedded words: 0.0003974437713623047**

**﻿Building hashtable with chaining using option 6**

**Chaining hashtable stats:**

**Number of nodes: 20000**

**Running time for chaining 1.0165669918060303**

**Word similarities found:**

**Similarity between bear and bear = 1.0000001**

**None and shrimp**

**None and None**

**Similarity between federer and baseball = 0.28697854**

**Similarity between federer and tennis = 0.71676075**

**Similarity between harvard and stanford = 0.8466463**

**harvard and None**

**Similarity between harvard and ant = -0.026703795**

**Similarity between raven and crow = 0.6150122**

**Similarity between raven and whale = 0.32908916**

**Similarity between spain and france = 0.7909149**

**Similarity between spain and mexico = 0.75137645**

**Similarity between mexico and france = 0.54779637**

**Similarity between mexico and guatemala = 0.81138104**

**computer and None**

**Running time for chaining of embedded words: 0.002105712890625**

Running Time

HashTable using Chaining

**Conclusion**

In conclusion this lab compared the functionality of different data structures. By tying in Lab 4 to this Lab it is clear to see the differences that are present in each data structure in computer science. It is also apparent that there are advantages and disadvantages to each data structure. Comparing the results obtained from Lab 4 compared to Lab 5 it is clear to see that a hash table with an efficient hash function is better than a b-tree or a binary search tree. When using a simple hash function like function 1 the hash table takes longer to construct. When being able to produce a value for the string and modifying it as to prevent collisions the construction of the hash table becomes much quicker and efficient. Overall this lab showcased a real-world example of deciding which data structure is most efficient to use when encountering a problem.

**Appendix**

**﻿**

**﻿**

**﻿import codecs**

**import time**

**import numpy as np**

**from HashTable\_Chaining import HashTable\_Chain**

**from HashTable\_LinearProbing import HashTable\_LP**

**from WordEmbedding import WordEmbedding**

**def readFile(filename):**

**empty = []**

**with open(filename) as txt:**

**for line in txt:**

**piece = line.split()**

**word = piece[0]**

**embedding = np.array([float(value) for value in piece[1:]])**

**empty.append(WordEmbedding(word, embedding))**

**return empty**

**def readPairs(filename):**

**empty = []**

**with open(filename) as txt:**

**for line in txt:**

**empty.append(list(map(lambda x: x.strip(), line.split(","))))**

**return empty**

**def CalSimilarity(e1, e2):**

**return np.dot(e1, e2) / (np.linalg.norm(e1) \* np.linalg.norm(e2))**

**def function1(k, n):**

**return len(k) % n**

**def function2(k, n):**

**return ord(k[0]) % n**

**def function3(k, n):**

**return (ord(k[0]) \* ord(k[-1]) ) % n**

**def function4(k, n):**

**return sum(list(map(lambda x: ord(x), k))) % n**

**def function5(k, n):**

**return 1 if (k == "") or (len(k) == 0) else (ord(k[0]) + 255 \* function5(k[1:], n)) % n**

**def function6(k, n):**

**return 1 if (k == "") or (len(k) == 0) else (ord(k[0])\*5 + ord(k[0]) + function1(k[1:], n)) % n**

**modelFile = "glove.6B.50d.txt"**

**pairsFile = "pairs.txt"**

**ArrOfHashFunc = [function1, function2, function3, function4, function5, function6]**

**print("Choose table implementation")**

**print("Type 1 for chaining or 2 linear probing")**

**choice = int(input("Choice: "))**

**TypeTable = None**

**Hash\_Function = None**

**elapsed = 0**

**print("Choose hash function: \n")**

**print("1. The length of the string % n")**

**print("2. The ascii value (ord(c)) of the first character in the string % n")**

**print("3. The product of the ascii values of the first and last characters in the string % n")**

**print("4. The sum of the ascii values of the characters in the string % n")**

**print("5. The recursive formulation h("",n) = 1; h(S,n) = (ord(s[0]) + 255\*h(s[1:],n))% n")**

**print("6. Another function of your choice")**

**Hash\_Function = int(input("Hash Function: ")) - 1**

**if choice == 1:**

**TypeTable = "chaining"**

**print("\nBuilding hashtable with chaining using option",Hash\_Function+1,"\n")**

**temp = readFile(modelFile)**

**table = HashTable\_Chain(len(temp), ArrOfHashFunc[Hash\_Function])**

**start = time.time()**

**for x in temp:table.insert(x)**

**end = time.time()**

**elapsed = (end - start)**

**print("Chaining hashtable stats: ")**

**elif choice == 2:**

**TypeTable = "linear probing"**

**print("\nBuilding hashtable with linear probing\n")**

**temp = readFile(modelFile)**

**table = HashTable\_LP(len(temp), ArrOfHashFunc[Hash\_Function])**

**start = time.time()**

**for x in temp:table.insert(x)**

**end = time.time()**

**elapsed = (end - start)**

**print("Linear probing hashtable stats: ")**

**print("Number of nodes:",(len(table)))**

**print("Running time for",TypeTable,elapsed,"\n")**

**pairs = readPairs(pairsFile)**

**print("Word similarities found: ")**

**total = 0**

**for (x1, x2) in pairs:**

**start = time.time()**

**e1 = table.find(x1)**

**e2 = table.find(x2)**

**end = time.time()**

**total += (end - start)**

**if e1 == None or e2 == None:**

**print(table.find(x1),"and",table.find(x2))**

**continue**

**emb= CalSimilarity(e1.emb, e2.emb)**

**print("Similarity between",x1,"and",x2,"=",emb)**

**﻿class HashTable\_Chain(object):**

**# Constructor**

**def \_\_init\_\_(self, size, hash\_form):**

**self.bucket = [[] for i in range(size)]**

**self.n = 0**

**self.h = hash\_form**

**def insert(self,k):**

**b = self.h(str(k), len(self.bucket))**

**if k not in self.bucket[b]:**

**self.bucket[b].append(k)**

**self.n += 1**

**def find(self,k):**

**b = self.h(str(k), len(self.bucket))**

**try:**

**i = self.bucket[b].index(str(k))**

**except:**

**i = -1**

**return None if i == -1 else self.bucket[b][i]**

**def \_\_len\_\_(self):**

**return self.n**

**﻿class HashTable\_LP(object):**

**# Constructor**

**def \_\_init\_\_(self,size, hash\_form):**

**self.item = [-1 for i in range(size)]**

**self.n = 0**

**self.h = hash\_form**

**def insert(self,k):**

**pos = self.h(str(k), len(self.item))**

**if self.item[pos] == k: return None**

**if self.item[pos] < 0:**

**self.item[pos] = k**

**self.n += 1**

**return pos**

**for i in range(1, len(self.item) + 1):**

**pos = (pos + i) % len(self.item)**

**# Found empty bucket**

**if self.item[pos] < 0:**

**self.item[pos] = k**

**self.n += 1**

**return pos**

**return -1**

**def find(self,k):**

**pos = self.h(str(k), len(self.item))**

**if self.item[pos] == -1: return None**

**if self.item[pos] == k: return self.item[pos]**

**for i in range(1, len(self.item) + 1):**

**pos = (pos + 1) % len(self.item)**

**if self.item[pos] == k: return self.item[pos]**

**if self.item[pos] == -1: return None**

**return None**

**def \_\_len\_\_(self):**

**return self.n**

**﻿class WordEmbedding(object):**

**def \_\_init\_\_(self,word,embedding):**

**# 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)**

**def \_\_gt\_\_(self, other):**

**if isinstance(other, WordEmbedding): return self.word > other.word**

**if isinstance(other, int): return False**

**return self.word > other**

**def \_\_lt\_\_(self, other):**

**if isinstance(other, WordEmbedding): return self.word < other.word**

**if isinstance(other, int): return False**

**return self.word < other**

**def \_\_eq\_\_(self, other):**

**if isinstance(other, WordEmbedding): return self.word == other.word**

**if isinstance(other, int): return False**

**return self.word == other**

**def \_\_str\_\_(self):**

**return self.word**

**def \_\_repr\_\_(self):**

**return self.word**

**def \_\_len\_\_(self):**

**return len(self.word)**

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