Lab 4 Option B Report

Data Structures

CS 2302

Jesus Maximino Hernandez

November 11, 2018

Introduction

I created a program that sorts English words into Hash Tables using two different hash functions to find words more effectively than trees. This program is also able to calculate load factor and count number of comparisons when looking for a specific word in the table.

Proposed solution

In order to have an optimal hash table I read that it is better to have empty buckets rather than have multiple elements in a single bucket. With that being said, I made the hash table size 1.25 times the number of words in the file. I then made two hash functions to compute where the words are going to go by implementing the module method taught in class and the multiply add and divide method found in a book. After this was done, I made a program to find load factor, count the number of comparisons made when looking for a word and then made a method to calculate the average number of comparisons.

Experimental results

The test cases were not very long and are simply using hash tables using two different hash functions. I tested on a file with 5,000 words. My program is able to put all 400,000 English words into the hash functions in about 20 seconds however, takes a considerable amount of time computing the number of comparisons.

SAMPLE #1: Division/Module function on testw.txt:

```
In [1]: runfile('/Users/maximinohernandez/Desktop/CS/CS3-Python/Lab4-CS3/Lab4Finale.py', wdir='/
Users/maximinohernandez/Desktop/CS/CS3-Python/Lab4-CS3')

Enter 'A' for Divison Method or 'B' for MAD Method: a
Creating Hash Table using Divison Method
Hash Function Completed
Load Factor of HashTable is 0.7999339716077913
Average number of comparisons is 2.097213622291022
```

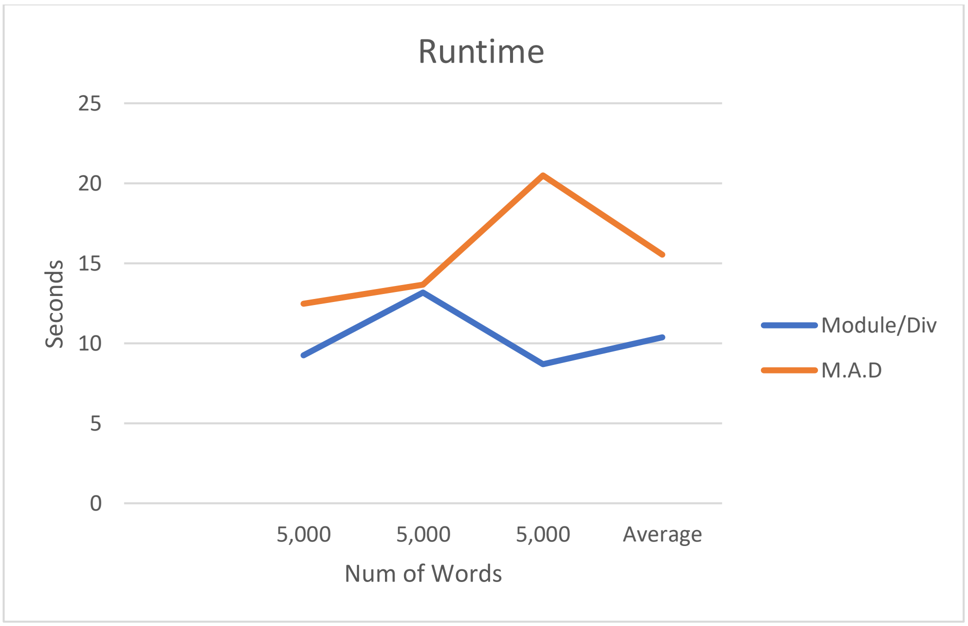
SAMPLE #2: M.A.D. function on testw.txt:

```
In [2]: runfile('/Users/maximinohernandez/Desktop/CS/CS3-Python/Lab4-CS3/Lab4Finale.py', wdir='/
Users/maximinohernandez/Desktop/CS/CS3-Python/Lab4-CS3')
Reloaded modules: HashTableN

Enter 'A' for Divison Method or 'B' for MAD Method: b
Creating Hash Table using MAD Method
Hash Function Completed
Load Factor of HashTable is 0.7999339716077913
Average number of comparisons is 1.0053663570691433
```

Time Complexity

The program is assumed to run in O(nlogn) time since the Hash table is being made from scratch however, it has O(1) access. Here is a graph with the specific times.



Conclusion

As you can see from the chart the M.A.D method took longer to run than the Module method but only has an average of 1.31 comparisons to the Module's 2.12 comparisons. In this lab I learned how to create a hash table using different hash functions while also learning how to calculate load factor and comparisons made.

Appendix

```
"""
Created on Sat Nov 10 22:24:55 2018
Jesus Maximino Hernandez
CS 2302 Data Structures - Diego Aguirre
TA - Manoj Saha
Lab 4 - Option B
Program that sorts english words into a hash table using different hash functions
and has different functions determine comparisons and load factor
@author: JesusMHernandez
"""
from HashTableN import HashTableNode
import math
import random

gl_file_name = "testW.txt"

def string_to_base26(word):
    '''
    Converts a string from Base26 to a positive
    integer.
    '''
    word = word.lower()
    if word == " " or len(word) == 0:
        return 0
    if len(word) == 1:
        return ord(word)-96
    else:
        return string_to_base26(word[1:])+(26**(len(word)-1))*(ord(word[0])-96)

def count_lines(file_name):
    '''
    counts how many words in file
    '''
    count = 0
    with open(gl_file_name) as f: #change file to testFile when you want to test
        for line in f:
            count += 1
    return count


def div_hash_function(word):
    '''
    method to determine hash function using divison method
    '''
    value = string_to_base26(word)
    size = count_lines(gl_file_name)
    #print(str(value % size)+"function")
    return value % size

def mad_hash_function(word):
    '''
    method to determine hash function using Multiply-Add-and-Divide method
    [(ai+b) mod p] mod N,
    '''

    prime = 0
    size = count_lines(gl_file_name)
    prime = find_next_prime(size)
    a = random.randint(0, prime-1)
    b = random.randint(0, prime-1)


    function = ((a+b) % prime) % size
    #print(str(function)+"function")
    return function
def is_prime(a):
    '''
    Function to check if a num is prime
    '''
    for i in range(2, a):
        if a % i == 0:
            return False
    return True

def find_next_prime(n):
  '''
  Function to find the next prime number
  '''
  num = n + 1
  while True:
     if is_prime(num):
        return num
     else:
        num += 1

def num_com(table, word, index):
    '''
    gets num of comparisons when looking for a word
    '''
    counter = 0
    temp = table[index]

    while temp is not None:
        counter += 1
        temp = temp.next

    return counter
def avg_com(table, file_name, method):
    '''
    gets average num of comparisons when looking for words in a file
    '''
    i = 0
    temp = dict()
    with open(file_name) as f:
            for line in f:
                 if "\n" in line:
                     line = line.replace("\n", "")
                 word = (line.lower()) #this line makes every word a lower case
                 if method == 'a':
                     i = div_hash_function(word)
                 if method == 'b':
                     i = mad_hash_function(word)
                 temp[word] = num_com(table, word, i)
    total_comp = 0
    counter = 0
    # Counts comparisons
    for value in temp:
        total_comp += temp[value]
        counter += 1
    return total_comp / counter


def get_load_factor(table):
        '''
        Gets load factor by counting how many elements in table then dividing
        it by size
            '''
        num_elements = 0

        for i in range(len(table)):
            temp = table[i]

            while temp is not None:
                num_elements += 1
                temp = temp.next
        #print(num_elements)
        #print(len(table))
        return num_elements / len(table)

def create_hash(file_name, method):
    '''
    creates the hash table
    '''

    lines = count_lines(file_name)
    #print(lines)
    size = int(round(lines * 1.25))
    #print(size)
    table = [None] * size
#    table = [[] for x in range(size)]
    if(method == 'a'):
        print("Creating Hash Table using Divison Method")
        with open(file_name) as f: #change file to testFile when you want to test
            for line in f:
                 if "\n" in line:
                     line = line.replace("\n", "")
                 word = (line.lower()) #this line makes every word a lower case
                 index = div_hash_function(word)
                 table[index] = HashTableNode(line, table[index])
                 #insert(table, word)
    else:
        print("Creating Hash Table using MAD Method")
        with open(file_name) as f: #change file to testFile when you want to test
            for line in f:
                 if "\n" in line:
                     line = line.replace("\n", "")
                 word = (line.lower()) #this line makes every word a lower case
                 index = mad_hash_function(word)
                 table[index] = HashTableNode(line, table[index])
    return table

def main():
    file_name = gl_file_name
    valid = False
    #method= ''

    while(valid == False):
        answer = input("Enter 'A' for Divison Method or 'B' for MAD Method: ")
        answer = answer.lower()
        if(answer == 'a'or answer == 'b'):
            hash_table = create_hash(file_name, answer)
            print("Hash Function Completed")
            print("Load Factor of HashTable is " +
str(get_load_factor(hash_table)))
            print("Average number of comparisons is " +  str(avg_com(hash_table,
file_name, answer)))
           # num_com("mom")
            break
        if(answer != 'b' or answer != 'a'):
            print("Invalid Input try again")
            valid = False


main(
```

“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.”