Lab V

Analysis of Text Documents

Using Hash Table

```
bucket 0: [ [female, female] ]
bucket 1: [ ]
bucket 2: [ [factors, factors] [wistar, wistar] [induce, induce] ]
bucket 3: [ [ccc, ccc] [extinction, extinction] ]
bucket 4: [ ]
bucket 5: [ ]
bucket 6: [ [suggest, suggest] ]
bucket 7: [ [development, development] [daily, daily] ]
bucket 8: [ [acpp, acpp] [plus, plus] ]
bucket 9: [ ]
bucket 10: [ [extinguish, extinguish] ]
bucket 11: [ ]
bucket 12: [ ]
bucket 13: [ [moreover, moreover] ]
bucket 14: [ ]
bucket 15: [ ]
bucket 16: [ [postnatal, postnatal] [reminded, reminded] [cage, cage] ]
bucket 17: [ ]
bucket 18: [ [conditioned, conditioned] [randomly, randomly] [exposed, exposed] ]
bucket 19: [ [male, male] ]
bucket 20: [ ]
bucket 21: [ [investigate, investigate] ]
bucket 22: [ ]
bucket 23: [ ]
bucket 24: [ ]
bucket 25: [ ]
bucket 26: [ [results, results] ]
bucket 27: [ [social, social] [conditioning, conditioning] ]
bucket 28: [ [drug, drug] ]
bucket 29: [ ]
bucket 30: [ [day, day] ]
bucket 31: [ ]
bucket 32: [ [relapse, relapse] ]
```

Course: CS 2302

Section: 12:00 p.m. – 1:20 p.m.

Author: Oswaldo Escobedo

Instructor: Dr. Fuentes

TA: Harshavardhini Bagavathyraj

Introduction

The objective of this laboratory is to create a program which reads and analyzes multiple text documents and finds the *word* that appears the most in a specific text using a *Hash Table.*

Therefore, to create this program (python code) we must do the following:

1. Extract all the words from the stop_words.txt file and store them in a list of strings, these words are the most widely used in the English language and therefore do not provide relevant information.
2. Create a hash table which contains the stop words.
3. Display statistics of the Hash Table that was created (No. buckets, No. keys, No. empty buckets, etc).
4. For every text file extract all the words into a list of strings.
5. Remove from the list all the words that are considered a stop word.
6. Create a method which identifies the most repeated word.
7. Display the number of words before and after stop words removal.
8. Display Statistics which describes the hash table that was created.

Proposed Solution Design and Implementation

Observation:

Before starting the laboratory we must understand the code that Dr. Fuentes provided us.

In his code we can see that he created the method 'get_word_list(text)' which receives a text document and returns a list of strings that contains all the words in the file. In this method he makes all the words to be in lowercase, this with the purpose of making the task simple. Next, he creates a for loop that iterates through all the text file, then he has an if statement which has the purpose of identifying if the current element is a word or a space, if it is a word we will form the word, else it will append the word to the list of strings.

Another part of the code that Dr. Fuentes provided us is that he sorted the text files and gave us a for loop which is in charge of reading all the content of the text document and then calls the method mentioned above, in order to create a list of strings which contains all the words of the document.

After having understanding this we must continue with the laboratory.

Note: to make the code more readable, efficient and simple, I will do many methods so that everything is organized.

get_stop_word_list()

First I started by creating the get_stop_word_list() method, which receives nothing, but its purpose is to read the text document stop_words.txt and to call the method get_word_list (), this with the purpose of converting this text file into a list of strings which will be returned at the end of this method.

create_hashtable(L)

Next, I set out to create a method that receives a list and returns a hash table containing all the words in L. I created this method because we will constantly create hash tables made of the words that are in the text file.

After creating this method, I called it to create a hash table containing the stop_words. Next, I started to create methods which have the task of calculating the statistics of the hash tables:

num_of_records(h)

This method receives a hash table and its purpose is to visit each bucket with the purpose of counting each record that exists within that bucket, so that in the end it returns the total number of records in the hashtable.

load_factor(h)

This method receives a hash table and its purpose is to count the total number of records that exist in the hash table, so in the end it will divide it with the number of buckets that exist in the hash table. In other words,

Load Factor = Total No. Records ÷ Total No. Buckets

empty_buckets(h)

This method receives a hash table and its purpose is to visit each bucket and if the bucket does not have a single record we will add one to the counter, else we will do nothing. In the end, we will divide the counter with the total number of buckets. In other words,

Empty Buckets = Total No. of Empty Buckets ÷ Total No. Buckets

long_buckets(h)

This method receives a hash table and its purpose is to visit each bucket and if the bucket has more than one record we will add one to the counter, else we will do nothing. In the end, we will divide the counter with the total number of buckets. In other words,

Long Buckets = Total No. of Long Buckets ÷ Total No. Buckets

In addition, inside the previous for loop I added another if statement which will be true if the length of the current bucket is greater than the current max bucket, therefore, we will update max bucket to be the current buckets length.

remove_stop_words(txt,stop)

This method receives a list of strings and a hash table containing the stop words. The purpose of this method is to visit each element of the list and remove any element that is considered a stop word. Therefore, we create a while loop, and inside of it will be a condition which will be true if the current item in the list is a stop word, if so we will pop that item and add one to the counter (this counter is used to count how many stop words there were in the original list), if the statement if true, it means that the current element is not a stop word, so we will add one to the index. In the end, this method will return a modified list that will no longer contain a stop word and will also return the stop word counter.

Note: the reason I did the while loop is because we are using the pop () function which will remove the item from the list but at the same time it will make the index go one place ahead, so I then did a while loop to manipulate the index more easily and thus avoid skipping words.

most_repeated_word(L)

This method receives a modified list, since it no longer contains stop words. Its purpose is to identify which is the word that is repeated the most times and how many times it is repeated, but also this word should be one of the last to be repeated. For example, let's say we have the following list ['Hello', '' Hello ',' Bye ', Bye], if we call the method it will return (Bye, 2), because despite the fact that both words are repeated the same number of times, we will give priority to the word that is repeated last.

With this in mind, I started by creating an empty hash table, to later create a list of size two which will be used to store the most repeated word will be saved and how many times it is repeated. Then, we will create a for loop to visit all the elements of the list, inside this loop there is an if condition, in which it will be false if the word is not in the hash table inside this condition the word will be inserted as a key and the number one as data (the one represents that the current word has been repeated at least once). On the other hand, this condition will be true if the element is already in the hash table, so we will continue increment one to the record data (representing that the word has been repeated one more time), then we will update the record, after this we find a condition which it will be true if the current word has been repeated more

times than the currently most repeated word, then we will update the current repeated word to be the current word.

print_info(wl,stop)

This method receives a list of strings containing the words of the text document and also receives the hash table containing the stop words. This method has the purpose of printing and introducing the first lines of code, because this method will print the total of words in the document text after the stop words and the total of stop words that were inside this document were eliminated. Next, the create_hashtable () method will be called to create a hash table that will contain the words from the modified list. Lastly, the statistics () method will be called to display the statistics.

statistics(h,L=[])

This method has the purpose of calling all the methods that analyze the hash table (long bucket, empty bucket, load factor, etc.), in addition to which it will print this information and finally it will call the most_repeated_word () method and then print which was the word that was repeated more times and how many times it was repeated.

Time Complexity

Method's name	Time Complexity
get_stop_word_list()	O(n)
create_hashtable(L)	O(n)
remove_stop_words(txt,stop)	O(n)
most_repeated_word(L)	O(n)
num_of_records(h)	O(n)
load_factor(h)	O(n)
empty_buckets(h)	O(n)
long_buckets(h)	O(n)
print_info(wl,stop,abstract)	O(n)
statistics(h,L=[])	O(n)

Experimental Results

Here are two images, a sample of the first inputs of what should be in our code (this output was provided by Dr. Fuentes) and the other image shows the output of my program.

Expected Result

```
Analysis of stop word hash table
Total buckets: 429, total records: 423, load factor 0.986
Empty bucket fraction in table: 0.38
Long bucket fraction in table: 0.263
Length of longest bucket in table: 4

File: abs_00.txt
Total words: 233, total non-stop-words: 134
Analysis of abs_00.txt hash table
Total buckets: 134, total records: 70, load factor 0.522
Empty bucket fraction in table: 0.604
Long bucket fraction in table: 0.104
Length of longest bucket in table: 3
Most common word: rats - occurs 11 times

File: abs_01.txt
Total words: 377, total non-stop-words: 203
Analysis of abs_01.txt hash table
Total buckets: 203, total records: 106, load factor 0.522
Empty bucket fraction in table: 0.655
Long bucket fraction in table: 0.128
Length of longest bucket in table: 4
Most common word: injection - occurs 13 times

File: abs_02.txt
Total words: 217, total non-stop-words: 141
Analysis of abs_02.txt hash table
Total buckets: 141, total records: 86, load factor 0.61
Empty bucket fraction in table: 0.546
Long bucket fraction in table: 0.135
Length of longest bucket in table: 4
Most common word: eeg - occurs 7 times

File: abs_03.txt
Total words: 275, total non-stop-words: 170
Analysis of abs_03.txt hash table
Total buckets: 170, total records: 130, load factor 0.765
Empty bucket fraction in table: 0.441
Long bucket fraction in table: 0.171
Length of longest bucket in table: 4
Most common word: autism - occurs 6 times
```

Actual Result

```
Analysis of stop word hash table
Total buckets: 429, total records: 423, load factor: 0.986
Empty bucket fraction in table: 0.38
Long bucket fraction in table: 0.263
Length of longest bucket in table: 4

File: abs_00.txt
Total Words: 233, total non-stop-words: 134
Analysis of abs_00.txt hash table
Total buckets: 134, total records: 70, load factor: 0.522
Empty bucket fraction in table: 0.604
Long bucket fraction in table: 0.104
Length of longest bucket in table: 3
Most common word: rats - occurs 11 times

File: abs_01.txt
Total Words: 377, total non-stop-words: 203
Analysis of abs_01.txt hash table
Total buckets: 203, total records: 106, load factor: 0.522
Empty bucket fraction in table: 0.655
Long bucket fraction in table: 0.128
Length of longest bucket in table: 4
Most common word: injection - occurs 13 times

File: abs_02.txt
Total Words: 217, total non-stop-words: 141
Analysis of abs_02.txt hash table
Total buckets: 141, total records: 86, load factor: 0.61
Empty bucket fraction in table: 0.546
Long bucket fraction in table: 0.135
Length of longest bucket in table: 4
Most common word: eeg - occurs 7 times

File: abs_03.txt
Total Words: 275, total non-stop-words: 170
Analysis of abs_03.txt hash table
Total buckets: 170, total records: 130, load factor: 0.765
Empty bucket fraction in table: 0.441
Long bucket fraction in table: 0.171
Length of longest bucket in table: 4
Most common word: autism - occurs 6 times
```

Here is a piece of the stop words hash table, this with the purpose of showing the efficiency of the data structure.

```
bucket 0: [ [problem, problem] [whether, whether] ]
bucket 1: [ [you, you] ]
bucket 2: [ ]
bucket 3: [ [across, across] ]
bucket 4: [ [cannot, cannot] ]
bucket 5: [ [backed, backed] [see, see] ]
bucket 6: [ [by, by] [upon, upon] ]
bucket 7: [ ]
bucket 8: [ ]
bucket 9: [ [per, per] ]
bucket 10: [ ]
bucket 11: [ [same, same] ]
bucket 12: [ [fact, fact] ]
bucket 13: [ ]
bucket 14: [ ]
bucket 15: [ [here, here] [use, use] ]
bucket 16: [ [one, one] ]
bucket 17: [ [smaller, smaller] ]
bucket 18: [ ]
bucket 19: [ ]
bucket 20: [ [back, back] [done, done] [showing, showing] ]
bucket 21: [ [great, great] [youngest, youngest] ]
bucket 22: [ ]
bucket 23: [ ]
bucket 24: [ [younger, younger] ]
bucket 25: [ ]
bucket 26: [ ]
bucket 27: [ ]
bucket 28: [ ]
bucket 29: [ ]
bucket 30: [ [me, me] ]
bucket 31: [ [nowhere, nowhere] ]
bucket 32: [ [three, three] ]
bucket 33: [ ]
bucket 34: [ ]
bucket 35: [ [anything, anything] ]
```

```
bucket 36: [ [put, put] ]
bucket 37: [ ]
bucket 38: [ [yet, yet] ]
bucket 39: [ [backs, backs] [given, given] ]
bucket 40: [ [parted, parted] [took, took] ]
bucket 41: [ [someone, someone] ]
bucket 42: [ ]
bucket 43: [ [mr, mr] [necessary, necessary] [used, used] ]
bucket 44: [ [again, again] [gives, gives] ]
bucket 45: [ [have, have] ]
bucket 46: [ ]
bucket 47: [ [numbers, numbers] ]
bucket 48: [ [always, always] [least, least] [their, their] ]
bucket 49: [ [he, he] ]
bucket 50: [ [my, my] ]
bucket 51: [ [interests, interests] ]
bucket 52: [ [enough, enough] ]
bucket 53: [ ]
bucket 54: [ ]
bucket 55: [ ]
bucket 56: [ [been, been] ]
bucket 57: [ ]
bucket 58: [ [uses, uses] ]
bucket 59: [ [right, right] ]
bucket 60: [ ]
bucket 61: [ ]
bucket 62: [ [interest, interest] ]
bucket 63: [ [another, another] ]
bucket 64: [ [puts, puts] [two, two] ]
bucket 65: [ [downing, downing] ]
bucket 66: [ [much, much] [toward, toward] ]
bucket 67: [ ]
bucket 68: [ [places, places] ]
bucket 69: [ ]
bucket 70: [ ]
bucket 71: [ [show, show] ]
```

```
bucket 72: [ [seemed, seemed] ]
bucket 73: [ [needed, needed] ]
bucket 74: [ ]
bucket 75: [ [its, its] [would, would] ]
bucket 76: [ ]
bucket 77: [ [perhaps, perhaps] ]
bucket 78: [ ]
bucket 79: [ [came, came] ]
bucket 80: [ [mostly, mostly] [opening, opening] ]
bucket 81: [ [members, members] ]
bucket 82: [ ]
bucket 83: [ [when, when] ]
bucket 84: [ [for, for] [older, older] ]
bucket 85: [ [most, most] [seconds, seconds] ]
bucket 86: [ [will, will] ]
bucket 87: [ [full, full] [make, make] ]
bucket 88: [ ]
bucket 89: [ ]
bucket 90: [ [anywhere, anywhere] [gets, gets] [seem, seem] ]
bucket 91: [ [want, want] ]
bucket 92: [ ]
bucket 93: [ ]
bucket 94: [ [less, less] ]
bucket 95: [ ]
bucket 96: [ [sees, sees] ]
bucket 97: [ [a, a] [certain, certain] [man, man] [present, present] ]
bucket 98: [ [b, b] [sides, sides] ]
bucket 99: [ [c, c] ]
bucket 100: [ [d, d] [everyone, everyone] ]
bucket 101: [ [downed, downed] [e, e] ]
bucket 102: [ [f, f] [furthered, furthered] ]
bucket 103: [ [all, all] [g, g] [must, must] [so, so] ]
bucket 104: [ [anybody, anybody] [h, h] ]
bucket 105: [ [i, i] ]
bucket 106: [ [j, j] ]
bucket 107: [ [k, k] [this, this] ]
```

```
bucket 108: [ [l, l] [may, may] ]
bucket 109: [ [give, give] [m, m] ]
bucket 110: [ [n, n] ]
bucket 111: [ [o, o] [said, said] [the, the] ]
bucket 112: [ [p, p] ]
bucket 113: [ [anyone, anyone] [q, q] ]
bucket 114: [ [r, r] ]
bucket 115: [ [problems, problems] [s, s] ]
bucket 116: [ [certainly, certainly] [later, later] [t, t] ]
bucket 117: [ [states, states] [u, u] ]
bucket 118: [ [having, having] [large, large] [v, v] ]
bucket 119: [ [away, away] [w, w] ]
bucket 120: [ [x, x] ]
bucket 121: [ [y, y] ]
bucket 122: [ [also, also] [no, no] [z, z] ]
bucket 123: [ [know, know] ]
bucket 124: [ [presenting, presenting] [saw, saw] [shall, shall] ]
bucket 125: [ ]
bucket 126: [ [say, say] ]
bucket 127: [ [few, few] [keep, keep] [part, part] [rather, rather] ]
bucket 128: [ [end, end] ]
bucket 129: [ ]
bucket 130: [ [asking, asking] [even, even] [who, who] ]
bucket 131: [ [interested, interested] ]
bucket 132: [ [if, if] ]
bucket 133: [ ]
bucket 134: [ [becomes, becomes] [ever, ever] [needs, needs] [ways, ways] ]
bucket 135: [ ]
bucket 136: [ ]
bucket 137: [ ]
bucket 138: [ ]
bucket 139: [ [everything, everything] ]
bucket 140: [ [in, in] [why, why] ]
bucket 141: [ [big, big] [noone, noone] ]
bucket 142: [ [case, case] [let, let] [until, until] ]
bucket 143: [ [greatest, greatest] ]
```

Conclusion

In conclusion, this laboratory helped me to practice my knowledge and understanding about this data structure. Also, when I was creating the methods I realized that thanks to the hash tables we can make our code look much easier and more efficient, since it improves the complexity time by a lot. For example, when analyzing which was the largest bucket, note that of all the text documents, the largest was normally between 4 and 5, but it was not more than 5, which says a lot about this data structure. So far, this data structure has been one of my favorites due to its simplicity and because it improves the search, insert, and delete algorithms by O(1).

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.

Oswaldo Escobedo

Appendix

```
#   Course: CS 2302
#   Assignment: Lab V
#   Author: Oswaldo Escobedo
#   Instructor: Dr. Fuentes
#   TA: Harshavardhini Bagavathyraj
#   Date of Last Modification: 04/9/2020
```

```
#   Purpose of the Program: to analyz multiple text documents using hashtable
import numpy as np
import os
import hash_table_chain as htc


def get_word_list(text):
    # Receives a string containing a document
    # Returns a list of strings containing the words in the document
    text = text.lower()
    word_list = []
    curr_wrd = ''
    for c in text:
        if ord(c)>=97 and ord(c)<=122:
            curr_wrd = curr_wrd+c
        else:
            if len(curr_wrd)>0:
                word_list.append(curr_wrd)
                curr_wrd = ''
    return word_list


def get_stop_word_list():
    f = open('stop_words.txt')
    txt = f.read()
```

```
    f.close()
    stop_words = get_word_list(txt)
    return stop_words


def create_hashtable(L):
    h = htc.HashTableChain(len(L))
    for i in L:
        h.insert(i,i)
    return h


def remove_stop_words(txt,stop):
    ctxt = txt.copy()
    c = 0
    i = 0
    while i < len(ctxt):
        if stop.retrieve(ctxt[i]) in ctxt: # stop_word found
            ctxt.pop(i) # Deletes the stop_word from the text
            c += 1 # Counts how many stop_words were in the text
        else: # Because each time we are decreasing the text length, we will only
            i += 1  # change i when there is no stop_word in the current position
    return ctxt,c


def most_repeated_word(L):
```

```
    h = htc.HashTableChain(len(L))
    rep = ['',0] # Stores the most repeated word and how many times appears in the text
    for i in L:
        count = h.retrieve(i)
        if count == None: # Word is not in hashtable
            h.insert(i,1)
        else:
            count += 1
            h.update(i,count) # Updates the times a word appears in the text
            if count > rep[1]: # Updates the most repeated word
                rep[0],rep[1] = i,count
    return rep[0],rep[1]


def num_of_records(h):
    c = 0
    for b in h.bucket: # Visits each bucket
        c += len(b) # Counts how many record a bucket has
    return c


def load_factor(h):
    s = 0
    for b in h.bucket:
        if len(b) >= 1:
```

```
            s += len(b)
    return round(s/len(h.bucket),3) # Rounds the final answer by 3 decimal places


def empty_buckets(h):
    c = 0
    for b in h.bucket:
        if len(b) == 0: # Empty bucket found
            c += 1
    return round(c/len(h.bucket),3)


def long_buckets(h):
    max_buck = 0
    c = 0
    for b in h.bucket:
        if len(b) > 1: # Long bucket found
            c += 1
        if len(b) > max_buck: # Updates the size of the longest bucket
            max_buck = len(b)
    return round(c/len(h.bucket),3),max_buck


def print_info(wl,stop,abstract): # Method to print the introduction of the text
    txt,count = remove_stop_words(wl,stop_table)
    stop_count = len(wl) - count
```

```
    print('Total Words: {}, total non-stop-words: {}'.format(len(wl),stop_count))
    print('Analysis of {} hash table'.format(abstract))
    h = create_hashtable(txt)
    statistics(h,txt)

def statistics(h,L=[]): # Method that displays the statistics from the text
    long,length_long = long_buckets(h)
    total = len(h.bucket)
    rec = num_of_records(h)
    load = load_factor(h)
    empty = empty_buckets(h)
    print('Total buckets: {}'.format(total),end=', ')
    print('total records: {}'.format(rec),end=', ')
    print('load factor: {}'.format(load),end='\n')
    print('Empty bucket fraction in table: {}'.format(empty))
    print('Long bucket fraction in table: {}'.format(long))
    print('Length of longest bucket in table: {}'.format(length_long))
    if len(L) != 0:
        word,times = most_repeated_word(L)
        print('Most common word: {} - occurs {} times'.format(word,times))

if __name__ == "__main__":
```

```
abs_dir = '.\\abstracts\\'  # abstracts folder must be in current folder
abstracts =    sorted(os.listdir(abs_dir)) # Abstract contains a lsit with all abstract file names


stop_words = get_stop_word_list()
stop_table = create_hashtable(stop_words)
print('\nAnalysis of stop word hash table')
statistics(stop_table)


for abstract in abstracts:
    f = open(abs_dir+abstract, 'r', encoding="utf8")
    print('\nFile:',abstract)
    text = f.read()
    f.close()
    wl = get_word_list(text)
    print_info(wl,stop_table,abstract)
```