Lab Assignment #4 Deadline Code: 15th, July, Fri 11:59pm

Submitted by: Oscar I. Ricaud Report & Demo: 19th, July, Tues 1:10pm

ID: 80466791

Email Address: [oiricaud@miners.utep.edu](mailto:xyz@miners.utep.edu)

Class Time: Mon- Fri 10:55 am – 12:00 pm

../../../../Downloads/lab4.pdf

­­

**Introduction:**

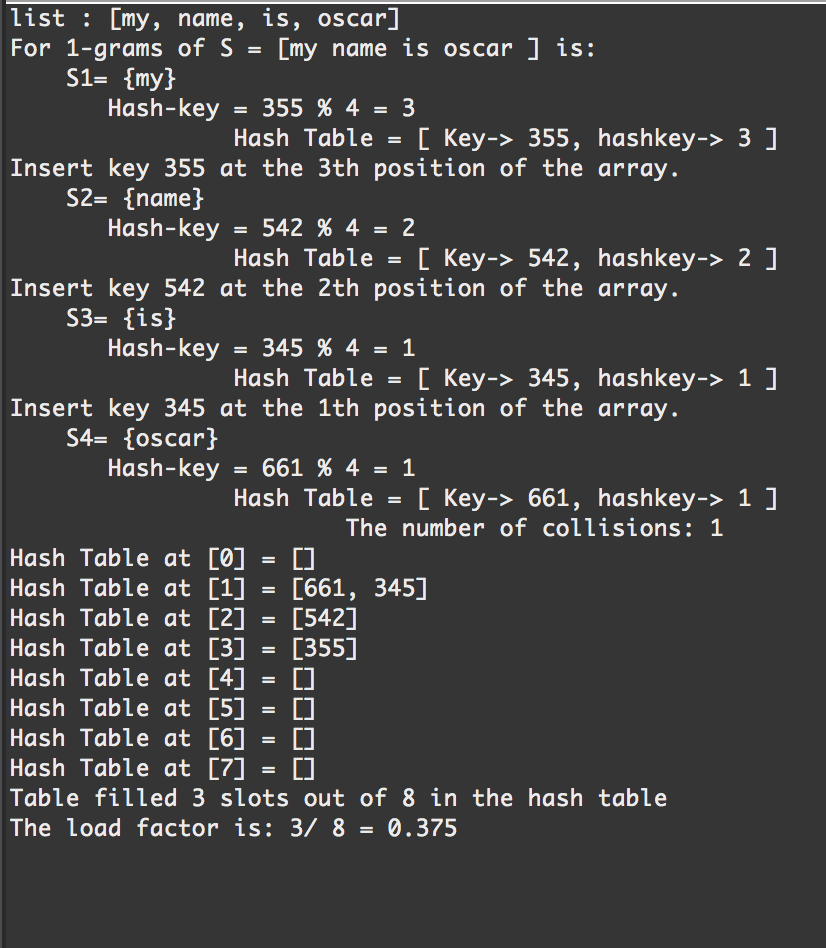
In this lab the main task is to implement a *Hash Table*. A *Hash Table* has a time complexity of O(1) [constant] for insertion, deletion and searching, in other words it is the fastest algorithm we have ever implemented. The hash table has the following properties; an array represented by a single column and a linked lists represented by rows. The program should simply read a text file and retrieve from the user an integer, n. Where n = n-grams and is represented by the maximum number of words a single node can have. Then the program converts the strings into characters using the ASCII format. For each letter in the n-gram it adds every integer/word and takes the % of the table size and inserts the key in the proper place of the hash table.

**Proposed Solution:**

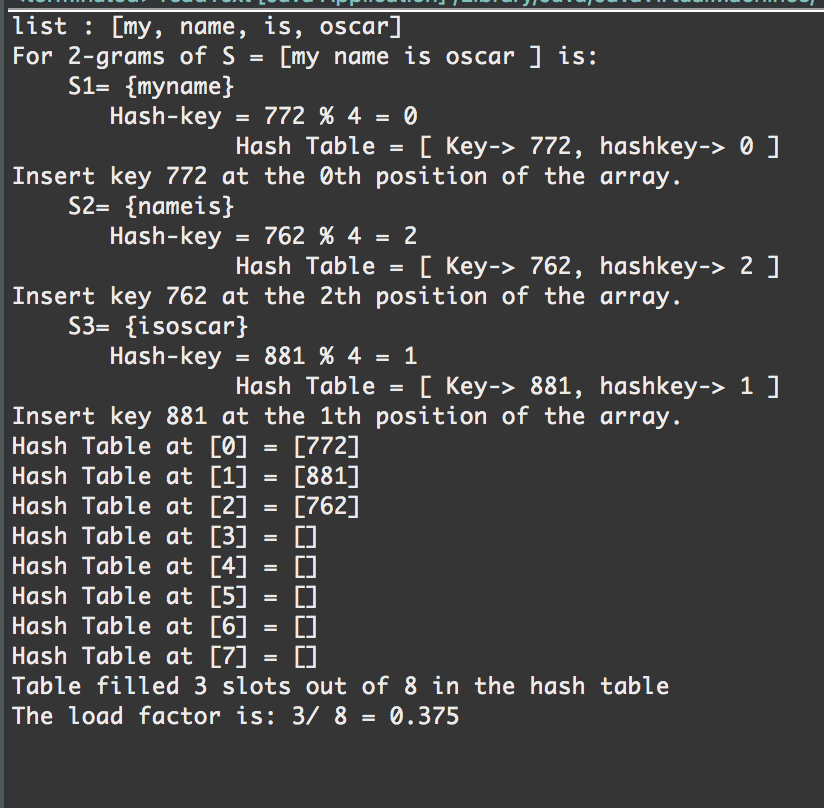
**Define classes**

1. *Main method Class: Reads text file converts to a linked lists, converts to node creates a hash table* 
   1. Read text file.
   2. Convert each text file into one linked list.
   3. For the linked list call the node method to convert the words into nodes.
   4. Initialize the hash table *column*
   5. Create table size and round up to the nearest power of 2.
   6. Build the sub lists respect to the hash table *column*
   7. Once you obtain the proper sub lists
   8. Print the hash table
2. *convertToTxtLinkedList: Method reads fileContents and returns list*
   1. for (i = 0 *TO* the length of the file contents)
   2. Char temp = fileContents.getCharAt(i)
   3. Integer end = 0
   4. Integer begin = 0
   5. If(temp != ‘ ‘)
      1. End++
   6. Else
      1. String temp2 = fileContents.substring(begin end)
      2. List.add(temp2)
      3. Begin = end+1
      4. End = begin
   7. Return list
3. *convertToNode: (node x, list that contains the text words and returns node x)*
   1. For( from the length of the list to the first element of the list)
   2. Create a new node x = new Node (list . get at (i))
   3. End for
   4. Return x
4. *Hashtable – Column(integer n): Initializes the hash table .* 
   1. myTable = new LinkedList(size)
   2. for(i = 0 TO myTable.length)
   3. myTable[i] = new LinkedList
   4. end
5. *buildSublists – Column(iNode sentence, int turtle, int rabbit, int listsize and iNode x).* 
   1. iNode copy = sentence
   2. String temp = {
   3. if(rabbit = listSize+1)
   4. return null
   5. else
   6. for(turtle TO rabbit)
      1. temp = temp + copy.element
      2. copy = copy.next
   7. end for
   8. temp = temp + }
   9. x = new iNode(temp, x)
   10. while(x!= null)
       1. print = The elements
       2. Increase the count++
       3. Key = HashCode(element)
       4. HashKey = key % list size
       5. hashTable(key, hashKey)
       6. x = x.next
   11. end while loop
   12. sentence = sentence.next
   13. buildSublists( sentence, turtle+1, rabbit+1, listsize, x)
   14. return temp
6. *HashCode – Column(Object Cargo) Converts the characters into ASCII Integers.* 
   1. int temp = 0
   2. int finalInt = 0
   3. String s = Cargo
   4. for(int i = 1 TO s.length() i++)
      1. char character = s.CharAt(i)
      2. int ascii = int character;
      3. finalInt = finalInt + ascii
      4. return final Int
   5. end method
7. *HashTable(int key, int hashKey) The magic happens*
   1. Print Key and Hash Key
   2. If myTable[hashKey] is empty
      1. Print Insert key at the haskKey position
      2. myTable[hashKey].add(key)
   3. else
      1. numCollisions++;
      2. int counter = 0;
      3. myTable[hashKey].add(counter, key)
      4. counter++
      5. Print the number of collisions.
   4. end method
8. *HashTable(int key, int hashKey) The magic happens*
   1. Print Key and Hash Key
   2. If myTable[hashKey] is empty
      1. Print Insert key at the haskKey position
      2. myTable[hashKey].add(key)
   3. else
      1. numCollisions++;
      2. int counter = 0;
      3. myTable[hashKey].add(counter, key)
      4. counter++
      5. Print the number of collisions.
   4. end method
9. *Print HashTable() Print table*
   1. Int filled = 0
   2. For( i = 0 TO column.myTable.length; i++)
      1. If(column.myTable[i] is not empty)
         1. Filled++;
      2. End if
      3. Print Hash table at [i]
      4. Print “ Table filled” + filled + “ slots out of “ + column.myTable.length + “ in the hash table”;
      5. Print load factor
   3. end method

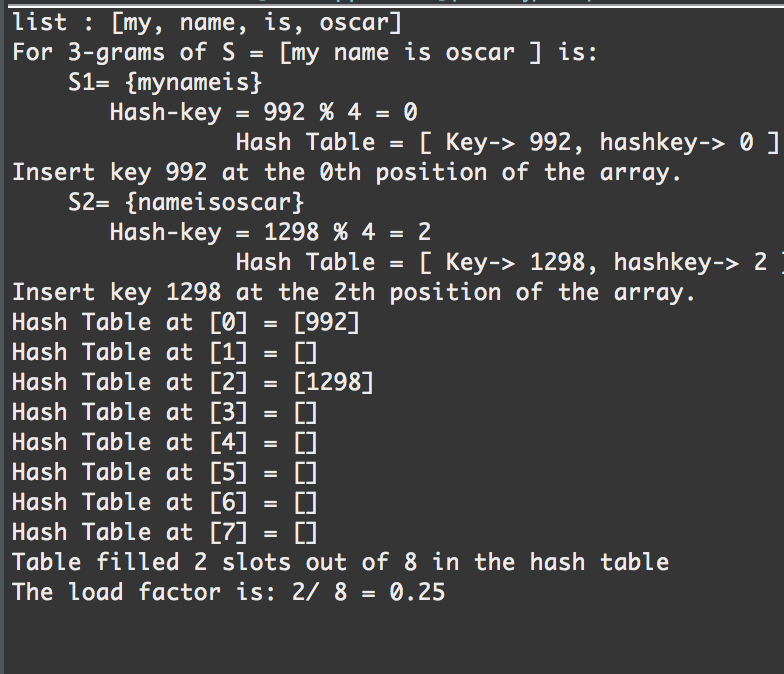
Input1 : “my name is oscar” n =1



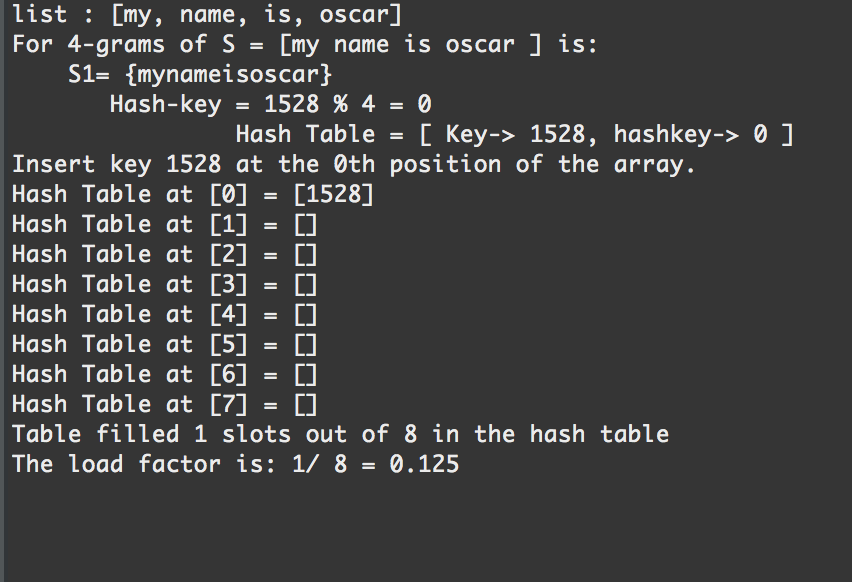
Input2: “my name is oscar” n = 2



Input3 : “my name is oscar” n = 3



Input4 : “my name is oscar” n = 4



Expirement 2:

List: id like to share a revelation that ive had during my time here It came to me when I tried to classify your species and I realized that youre not actually mammal every mammal on this planet instinctively develops a natural equilibrium with the surrounding environment but you humans do not you move to an area and you multiply and multiply until every natural resource is consumed and the only way you can survive is to spread to another area there is another organism on this planet that follows the same pattern. do you know what it is question mark a virus human beings are a disease a cancer of this planet youre a plague and we are the cure

Appendix: ­

package lab4;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.LinkedList;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //

//Author: Oscar Ivan Ricaud.

/\*Assignment: Lab 4

Instructor: Professor Julio Urenda

TA: Saiful Abu

Course 2302

Data of Last modification July 17th, 2016

\*/

//Program purpose:

/\* The purpose of this program is to implement a hash table to get a better understanding how this algorithm is fast.

\* It first reads a text file and converts each word into a node and then using the ASCII it converts the word

\* into an integer. I then created a hash table size on powers of 2 based on the size of the text file. So if the text

\* file is size 100 it will round up to the nearest power, 2^7 = 128 table size.

\*/

//How to operate lab4 readText.java:

/\* Click play button and read console for output

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //

/\*

\* for n = 1 and file= "my name is oscar", n-grams would be "my", "name", "is", "oscar",

\* for n = 2 n-grams would be "my name", "name is", "is oscar".

\* for n = 3, n-grams would be "my name is", "name is oscar".

\* for n = 4 n-grams would be "my name is oscar".

\*/

public class readText {

static int counter = 0;

static String emptySpace = "";

static int n = 5;

public static void main(String[] args) throws IOException {

/\* Begin to read a text file, change path if necessery \*/

File file = new File("/Users/oscarricaud/Desktop/name.txt");

BufferedReader br = new BufferedReader(new FileReader(file));

StringBuffer fileContents = new StringBuffer();

String line = br.readLine();

while (line != null) {

fileContents.append(line);

line = br.readLine();

}

br.close();

LinkedList <String> list = new LinkedList<String>();

list = convertoTxtLinkedList(fileContents.toString());

System.out.println("list : " + list);

/\* End to read a text file \*/

/\* Begin to convert the words into nodes \*/

iNode x = null;

x = convertToNode(x, list);

System.out.print("For " + n + "-grams of S = [" );

iNode.printList(x);

System.out.println("] is:");

/\* End to convert the words into nodes\*/

/\* Create the hash table respect to the table length by rounding up the nearest powers \*/

int log = (int) (Math.log(list.size()) / Math.log(2));

int tableSize = (int) Math.pow(2, log+1 );

Hashtable column = new Hashtable(tableSize);

int min = 0;

int max = n;

int ListSize = list.size();

iNode x2 = null;

/\* Build the n-Gram sublists \*/

column.buildSublists(x, min, max , ListSize, x2);

/\* Print hash table \*/

int filled = 0;

for(int i = 0 ; i < column.myTable.length; i++){

if(!(column.myTable[i].isEmpty())){

filled++;

}

System.out.println("Hash Table at [" + i + "] = " + column.myTable[i].toString());

}

System.out.println("Table filled " + filled + " slots out of " + (column.myTable.length) + " in the hash table");

double loadFactor = (double)(filled)/(column.myTable.length);

System.out.println("The load factor is: " + filled + "/ " + (column.myTable.length) + " = " + loadFactor);

}

// Converts the list into nodes //

private static iNode convertToNode(iNode x, LinkedList<String> list) {

for(int i = list.size()-1; i >= 0; i--){

x = new iNode(list.get(i), x);

}

return x;

}

/\* Converts the text file into a signly linked list \*/

private static LinkedList<String> convertoTxtLinkedList(String string2) {

LinkedList <String> list = new LinkedList<String>();

int end = 0;

int begin = 0;

for(int i = 0 ; i < string2.length(); i++){

char temp = string2.charAt(i);

if(temp !=(' ')){

end++;

}

else{

String temp2 = string2.substring(begin, end);

list.add(temp2);

begin = end+1;

end = begin;

}

}

return list;

}

}

**package** lab4;

**import** java.util.LinkedList;

**public** **class** **Hashtable** {

**int** count = 1;

**int** numCollisions = 0;

**public** **LinkedList** [] myTable;

**public** **Hashtable**(**int** size){

myTable = **new** **LinkedList** [size];

**for**(**int** **i** = 0; i < myTable.length; i++){

myTable[i] = **new** LinkedList();

}

}

**public** **String** **buildSublists**(**iNode** sentence, **int** turtle, **int** rabbit, **int** listSize, **iNode** x){

/\* Debug Here

System.out.println("In the insert method table length = " + myTable.length);

System.out.println("n = " + n);

System.out.println("min = " + min);

System.out.println("max = " + max);

System.out.println("listSize = " + listSize);

\*/

**String** **temp** = "{";

**iNode** **copy** = sentence;

/\* Rabbit is at the end therefore terminate \*/

**if**(rabbit == listSize+1){

// System.out.println("Max Cap hit: ");

**return** **null**;

}

**else**{

**for**(**int** **i** = turtle; i < rabbit; i++){

temp = temp + copy.element;

copy=copy.next;

}

temp = temp + "}";

x = **new** iNode(temp, x);

**while**(x!= **null**){

**System**.***out***.println(" S" + count + "= " + x.element);

count++;

**int** **key** = hashCode(x.element);

**int** **hashKey** = key % (listSize);

**System**.***out***.println(" Hash-key = " + key + " % " + listSize + " = " + hashKey);

hashTable(key, hashKey);

x = x.next;

}

}

sentence= sentence.next;

buildSublists(sentence, turtle+1, rabbit+1, listSize, x);

**return** temp;

}

**private** **void** **hashTable**(**int** key, **int** hashKey) {

**System**.***out***.println(" Hash Table = [ Key-> " + key + ", hashkey-> " + hashKey + " ]");

**if**(myTable[hashKey].isEmpty()){

**System**.***out***.println( "Insert key " + key + " at the " + hashKey + "th position of the array. ");

myTable[hashKey].add(key);

}

**else**{

numCollisions++;

**int** **counter** = 0;

myTable[hashKey].add(counter, key);

counter++;

**System**.***out***.println(" The number of collisions: " + numCollisions);

}

}

**private** **int** **hashCode**(**Object** cargo) {

**int** **temp** = 0;

**int** **finalInt** = 0;

**String** **s** = (**String**) cargo;

**for**(**int** **i** = 1 ; i < s.length (); i ++){

**char** **character** = s.charAt(i);

**int** **ascii** = (**int**) character; //convert the first character

finalInt = finalInt + ascii;

}

**return** finalInt;

}

**public** **void** **print**(){

**for**(**int** **i** = 0; i < myTable.length; i++){

**System**.***out***.println("hashtable[" + i + "] = " + myTable[i] );

}

**package** lab4;

**public** **class** **iNode** {

**Object** element;

**iNode** next;

**public** **iNode** () {

element = 0;

next = **null**;

}

**public** **iNode**(**String** givenval){

element = givenval;

}

**public** **iNode** (**Object** element, **iNode** next) {

**this**.element = element;

**this**.next = next;

}

**public** **int** **size**(**iNode** x){

**int** **size** = 0;

**while**(x!=**null**){

size++;

x = x.next;

}

**return** size++;

}

**public** **String** **toString** () {

**return** element + "";

}

**public** **Object** **getData**(){

**return** element;

}

**public** **iNode** **getNext**(){

**return** next;

}

**public** **iNode** **prev**(**iNode** head, **iNode** tail){

**iNode** **previous** = **null**;

**while**(head != **null**){

**if**(head.next == tail){

previous = head;

**return** previous;

}

head = head.next;

}

**return** previous;

}

**public** **static** **void** **printList**(**iNode** x){

**if** (x!= **null**){

**System**.***out***.print(x.element+" ");

*printList*(x.next);

}

}

}

Conclusion:

Attempting to build a hash table has given me the opportunity to understand more how n-grams are very much different than finding the permutations or combinations of a given list. Surprisingly I thought I had finished the lab the first day but soon enough I realized I had done everything wrong but I worked on it again and fixed everything. I also realized how fast these hash tables are but one of the biggest disadvantage is building the perfect hash function to equally distributed the data among the array. So a hash table can be very fast but if its implemented poorly the worst case would be having one array position with the whole list.

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.”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_