**Lab 9 Report**

**CS303L Algorithms and Data Structures**

**Sam Lazrak**

**Spring Semester 2018**

**Objectives:**

∙ Implement a hashing function.

∙ Use the hashing function to create a hash table implementation.

∙ Implement linear and quadratic probing methods and compare their performance with Red Black Trees

**In-class Assignment:**

1. Use the Java source files HashEntry.java and HashMap.java provided in canvas.
2. Create a hash table that is made of elements HashElement(int Key, String Value). The size of hash table will be 100.
3. Implement the following methods for the hash table:
   1. put(int Key, String Value):
   2. Puts the key value pair in the hash table at a certain index.
   3. You need to implement a simple hash function H(key) = key mod mapsize to find the index where you will put the pair.
   4. If collision occurs, i.e., a pair already exists in that index and the key is not the same as the current key, then you will use this function to resolve the collision, H(key)=(7\*H(key)+1) mod mapsize, until you get an empty slot.
   5. If the index is already full and the keys are the same, just replace the old value with the new one.
   6. get(int Key):
   7. Gets the value associated with the Key.
   8. You should not do linear search throughout the hash table for the key, rather you will calculate the index using the hash function stated above, go directly to that index and retrieve the value.
4. Write a driver program to test your implementation of hash table. Allow the user to put or get data.
5. Implement linear probing to put a new value in your hash table. The sequence of probes are H(key) = (H(key) + i) mod mapsize for i = 0, 1, 2, 3, … and so on.
6. Implement quadratic probing to put a new value in your hash table. The sequence of probes are H(key) = (H(key) + i 2 ) mod mapsize for i = 0, 1, 2, 3, … and so on

**Homework Assignment:**

1. Implement linear probing to put a new value in your hash table. The sequence of probes are H(key) = (H(key) + i) mod mapsize for i = 0, 1, 2, 3, … and so on.
2. Implement quadratic probing to put a new value in your hash table. The sequence of probes are H(key) = (H(key) + i 2 ) mod mapsize for i = 0, 1, 2, 3, … and so on.
3. Use the hash table implementation to store the description associated with a given set of UPC keys.
   1. The input file UPC.csv provides the key and corresponding descriptions in a comma separated file and the various search keys are provided in the file input.txt
   2. First test the program by entering couple of keys manually and print the description. Once you are convinced the program is working correctly, test the program for the given search keys and determine the total time taken to complete the search.
4. Compare the times for searching the keys using the first function (given in class), linear probing, quadratic probing with hash table and also with the corresponding performance of the RBTree that was implemented in an earlier lab.

Source code:

package Lab9;

public class HashEntry {

private int index;

private String data;

HashEntry(int index, String data) {

this.index = index;

this.data = data;

}

public int getIndex() {

return index;

}

public String getData() {

return data;

}

public void setData(String val) {

this.data = val;

}

}

package Lab9;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

import Lab8.RBBST;

public class HashMap {

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

// In-Class Assignment

Scanner s = new Scanner(System.in);

int[] keys = new int[100];

String[] values = new String[100];

HashMap table = new HashMap();

for(int i = 0; i < 5; i++) {

System.out.println("Enter the key:");

keys[i] = s.nextInt();

System.out.println("Enter your value:");

values[i] = s.next();

table.put(keys[i], values[i]);

System.out.println("Getting value from key " + keys[i] + "...");

System.out.println("The value is " + table.get(keys[i]) + "\n");

}

// Homework Assignment

File f = new File("input.txt");

Scanner scan = new Scanner(f);

String line = "";

String cvsSplitBy = ",";

String[] c = new String[17];

String[] d = new String[17];

// uniform hashing search

int j = 0;

System.out.println("Result of putting input keys into hash table and getting values: ");

long time = System.nanoTime();

while(scan.hasNextLine()){

line = scan.nextLine();

String[] search = line.split(cvsSplitBy);

c[j] = search[0];

d[j] = search[2];

table.put(Integer.parseInt(c[j]), d[j]);

// System.out.println("Key: " + c[j] + " " + "Value: " + table.get(Integer.parseInt(c[j])));

j++;

}

time = System.nanoTime() - time;

for(int i = 0; i < 17; i++) {

System.out.println("Key: " + c[i] + " " + "Value: " + table.get(Integer.parseInt(c[i])));

}

System.out.println( "\n" + "Time taken to search for values of keys using uniform hashing "

+ "in input.txt: " + time + " milliseconds");

// linear probing search

File fi = new File("input.txt");

Scanner sc = new Scanner(fi);

int k = 0;

long linearTime = System.nanoTime();

while(sc.hasNextLine()){

line = sc.nextLine();

String[] search = line.split(cvsSplitBy);

c[k] = search[0];

d[k] = search[2];

table.linearProbe(Integer.parseInt(c[k]), d[k]);

k++;

}

linearTime = System.nanoTime() - linearTime;

System.out.println( "\n" + "Time taken to search for values of keys using"

+ " linear probe in input.txt: " + linearTime + " milliseconds");

// quadratic probing search

File fil = new File("input.txt");

Scanner sca = new Scanner(fil);

int h = 0;

long quadraticTime = System.nanoTime();

while(sca.hasNextLine()){

line = sca.nextLine();

String[] search = line.split(cvsSplitBy);

c[h] = search[0];

d[h] = search[2];

table.quadraticProbe(Integer.parseInt(c[h]), d[h]);

h++;

}

quadraticTime = System.nanoTime() - quadraticTime;

System.out.println( "\n" + "Time taken to search for values of keys using "

+ "quadratic probe in input.txt: " + quadraticTime + " milliseconds");

// red-black tree search

File file = new File("input.txt");

Scanner scanner = new Scanner(file);

int g = 0;

RBBST<String, String> rb = new RBBST<String, String>();

long RBTime = System.nanoTime();

while(scanner.hasNextLine()){

line = scanner.nextLine();

String[] search = line.split(cvsSplitBy);

c[g] = search[0];

d[g] = search[2];

rb.insert(c[g], d[g]);

rb.search(c[g]);

g++;

}

RBTime = System.nanoTime() - RBTime;

System.out.println( "\n" + "Time taken to search for values of keys using Red-Black Search Tree"

+ " in input.txt: " + RBTime + " milliseconds");

}

private final static int TABLE\_SIZE = 100;

HashEntry[] table;

HashMap() {

table = new HashEntry[TABLE\_SIZE];

}

public String get(int index) {

index = index % TABLE\_SIZE;

String entry = table[index].getData();

return entry;

}

public void put(int index, String data) {

index = index % TABLE\_SIZE;

boolean put = false;

if(table[index] == null) {

table[index] = new HashEntry(index, data);

}

else {

while(put == false) {

if(index == table[index].getIndex()){

index = 7\*(index + 1) % TABLE\_SIZE;

table[index] = new HashEntry(index, data);

put = true;

}

else {

index = 7\*(index + 1) % TABLE\_SIZE;

put = false;

}

}

}

}

public void linearProbe(int index, String data){

int i = 0;

index = index % TABLE\_SIZE;

boolean put = false;

if(table[index] == null) {

table[index] = new HashEntry(index, data);

}

else {

while(put == false) {

if(index == table[index].getIndex()){

index = 7\*(index + i) % TABLE\_SIZE;

table[index] = new HashEntry(index, data);

put = true;

}

else {

index = 7\*(index + i) % TABLE\_SIZE;

put = false;

}

i++;

}

}

}

public void quadraticProbe(int index, String data){

int j = 0;

index = index % TABLE\_SIZE;

boolean put = false;

if(table[index] == null) {

table[index] = new HashEntry(index, data);

}

else {

while(put == false) {

if(index == table[index].getIndex()){

index = 7\*(index + j\*j) % TABLE\_SIZE;

table[index] = new HashEntry(index, data);

put = true;

}

else {

index = 7\*(index + j\*j) % TABLE\_SIZE;

put = false;

}

j++;

}

}

}

}

The output was:

Enter the key:

1

Enter your value:

a

Getting value from key 1...

The value is a

Enter the key:

2

Enter your value:

b

Getting value from key 2...

The value is b

Enter the key:

3

Enter your value:

c

Getting value from key 3...

The value is c

Enter the key:

4

Enter your value:

d

Getting value from key 4...

The value is d

Enter the key:

5

Enter your value:

e

Getting value from key 5...

The value is e

Result of putting input keys into hash table and getting values:

Key: 79 Value: INDIANA LOTTO

Key: 93 Value: treo 700w

Key: 123 Value: Wrsi Riversound cafe cd

Key: 161 Value: Dillons/Kroger Employee Coupon ($1.25 credit)

Key: 214 Value: Rhinestone Watch

Key: 2141 Value: """V"": Breakout/The Deception VHS Tape"

Key: 2144 Value: Tintorera - Tiger Shark

Key: 2143 Value: Taxi : The Collector's Edition VHS

Key: 2147 Value: Toshiba 2805 DVD player

Key: 2158 Value: GREEN SUGAR COOKIES4276

Key: 2155 Value: HOT COCOA W/BKMK

Key: 2158 Value: GREEN SUGAR COOKIES4276

Key: 2160 Value: Dollar Bar Rich Raspberry

Key: 2172 Value: Mixed seasonal flower bouquet

Key: 2177 Value: 4 way 13 AMP Extension Lead (Wilkinson UK)

Key: 2184 Value: Christopher's Assorted Fruit Jellies

Key: 2187 Value: fairway

Time taken to search for values of keys using uniform hashing in input.txt: 75556 nanoseconds

Time taken to search for values of keys using linear probe in input.txt: 1197960 nanoseconds

Time taken to search for values of keys using quadratic probe in input.txt: 1170062 nanoseconds

Time taken to search for values of keys using Red-Black Search Tree in input.txt: 1009624 nanoseconds