CSC2001F

ASSIGNMENT 3

May 9, 2017

Student ID: HLLYAS001
University of Cape Town
Department of Computer Sciences

Contents

INTRODUCTION

Problem

A hash function can be described as a conversion or mapping of data arbitrary in size to data of a fixed sized. These output data values, also known as hash values, can be used in data structures such as a hash table to increase search time operations and overall operation efficiency of a data structure. The functions performance can be evaluated by analyzing the spread and collision of data. A collision of data is when a function produces duplicate values from two or more distinct input data where as the spread describes the distribution of values in a given range or table size.To measure the quality of a hash function, the method of calculating the functions entropy for a given set of input and table size is employed. The hypothesis of this experiment follows that a 'good' hash function would have less collisions, more spread and higher entropy of data, which is the lack of predictability or order of data. The report investigates four hash functions using telephonic phone book entry names as input for hashing.

HASH FUNCTIONS

The worst case hash function

The worse case hash function describes the maximum number of collisions with poor spread. It is hence the case of $h(x) = 1$, where $h(x)$ is the hash function and its output the hash value. This function is simply implemented by returning a hash value of 1 for every telephonic name entry which is inserted into the function.

Hash function 1

Hash function 1 is a simple function for converting String data into hash values. The hash value is gained by finding the sum of the ASCII values from each character in the string and then applying the modulus operator using the table size m to this summation. For a small m we achieve a good distribution of values in a hash table because the function gives equal weight to all the letters in a string used as input. It can also be said that if the summation of letters is not large enough, the distribution would be poor due the range of ASCII values A to Z. For example, the ASCII values of A is 65 and Z is 90. For a 10 lettered string of upper case letters the sum of the letters is in between 650 - 900. Using a small table size of 100 would

yield a good distrubtion however if this size is increased to 1000 the hash values would only be slotted in the range 650 - 900 resulting in a poor distribution.

Hash function 2

Hash function 2 is essentially a modified hash function 1. Besides extracting the ASCII values for summation the formula for creating a hash value also shifts or multiplies each character by some value. Initially the ASCII value of the first character is set to the hash value.The next iteration sets the the new hash value to the previous hash value multiplied by a constant and then adds this to the ASCII value of the next character in the string. The process increases the hash value and the larger hash values are more efficient for spreading of data in the hash table slots. The function solves some of the uniqueness problems of hash values and reduces the probability of collision.

HashCode java function

Java's hashCode() function extends from the java.lang.String class. It implements a product sum algorithm over the entire text of a string. For a string s from the java.lang.String class the following formula is used to calculate the hash value:

$$h(s) = \sum_{i=0}^{n-1} s[i].31^{n-1-i}$$

where terms are summed using Java 32-bit int addition, $s[i]$ denotes the UTF-16 code unit of the ith character of the string, and n is the length of s.

EXPERIMENT DESIGN

The experiment follows a basic procedure of insertion and calculation. The file "testdata" contains a set of phone book entries from which the name of the entry needs to be extracted and used as the key value in the hash functions. The "Main" class reads in the data from testdata, splits it up based on the piping character and passes all the names into each function in a while loop. Once all the names have been passed the loop ends and entropy is calculated. To calculate the Entropy an "Entropy" class containing two methods and the declaration of a instance of hashmap called frequency was created. By using the hashmap instance methods such as get(), put() and containsKey(), the occurence of each hash value cn found. The void method "Occur" counts the number of times a hash value occurs by using an if statement.

The code says that if a instance of a hashmap "frequency" doesn't contain a input key then by using the method put() we can establish the key and set the occurance to 1 in frequency. Howver if the key already exists (containsKey()) in frequency, replace the old key with new one and increment the occurence by one. The key in this event is the hash value.
The second method calculates the entropy and may be used by each function contained in the Main class. The steps to calculating the entropy are given in the next section.

The following displays basic arguments given to put(), get() and containsKey():

1. put(int key, int occurence)
2. get(int key)
3. containsKey(int key)

METHOD OF ENTROPY

Description

As mentioned before the entropy is a measure of unpredictability or disorder. Hence a higher entropy proves more uniqueness. For our problem it is important to avoid collision of hash values and therefore we can measure the output of the hash functions by computing the entropy.

Steps to calculate entropy

1. Calculate the occurences of hash values
2. Divide each value found in 1. by the number of input to the hash function to calculate the probability $p(x)$
3. For each probability compute $h = -p(x).logp(x)$
4.The entropy will be the sum of all h.

SAMPLE HASH VALUES

Worst case hash function values

0

Hash function 1 values

1254 1474 1374 1026 1127 1156 1672 1034 1037 1337

Hash function 2 values

13768 -12338 -4200 19827 16806 3040 5278 2501 5776 8693

HashCode() values

-796020310 -2012322528 -118158568 2103544288 469348279 224659220 221192044 -1635901140 -1644537457 -773380355

RESULTS AND ANALYSIS

Tabulated entropy results

Hash Function	Entropy	Table size
worst case hash function	0	20011
hash function 1	6.631809	20011
hash function 2	9.045985	20011
hashCode()	9.206181	20011

Discussion

To calculte the entropy successfully the values used in java had to be of type double or float. Using an int data type would reflect in all answers equaling zero as java's integer division always rounds towards zero. By casting the required values (occurences and number of input) to double type a much higher precision could be obtained hence its implementation. An instance of the entropy class (see code) was also created for each function to reduce coding. From the entropy values obtained it is clear that for large table sizes hash function 2 and java's hashCode() worked much better. This is due to the shifting and multiplication of characters in a string which produces much larger hash values. The spread or dispersion is more evenly spread out and the number of collisions have decreased since the range of hash values are large especially for hashCode().
The worst case function represents a hash value of 1 for every key inserted into the function. The collisions are a maximum as 10000 keys have the same hash value. Therefore the entropy

is 0 as no uniqueness is present.
In the hash function 1 the hash values are smaller and the range is limited due the the number of characters used in a text string. As a result the entropy is lower than the more complex hash function 2 and hashCode() since more duplicates are present in a set.

CONCLUSION

The number of collisions and quality of spread is dependent on the quality of hash function and its ability to map distinct keys into unique hash values. If the hash function has a unique outcome every time the dispersion of data is more even, and the entropy is higher.