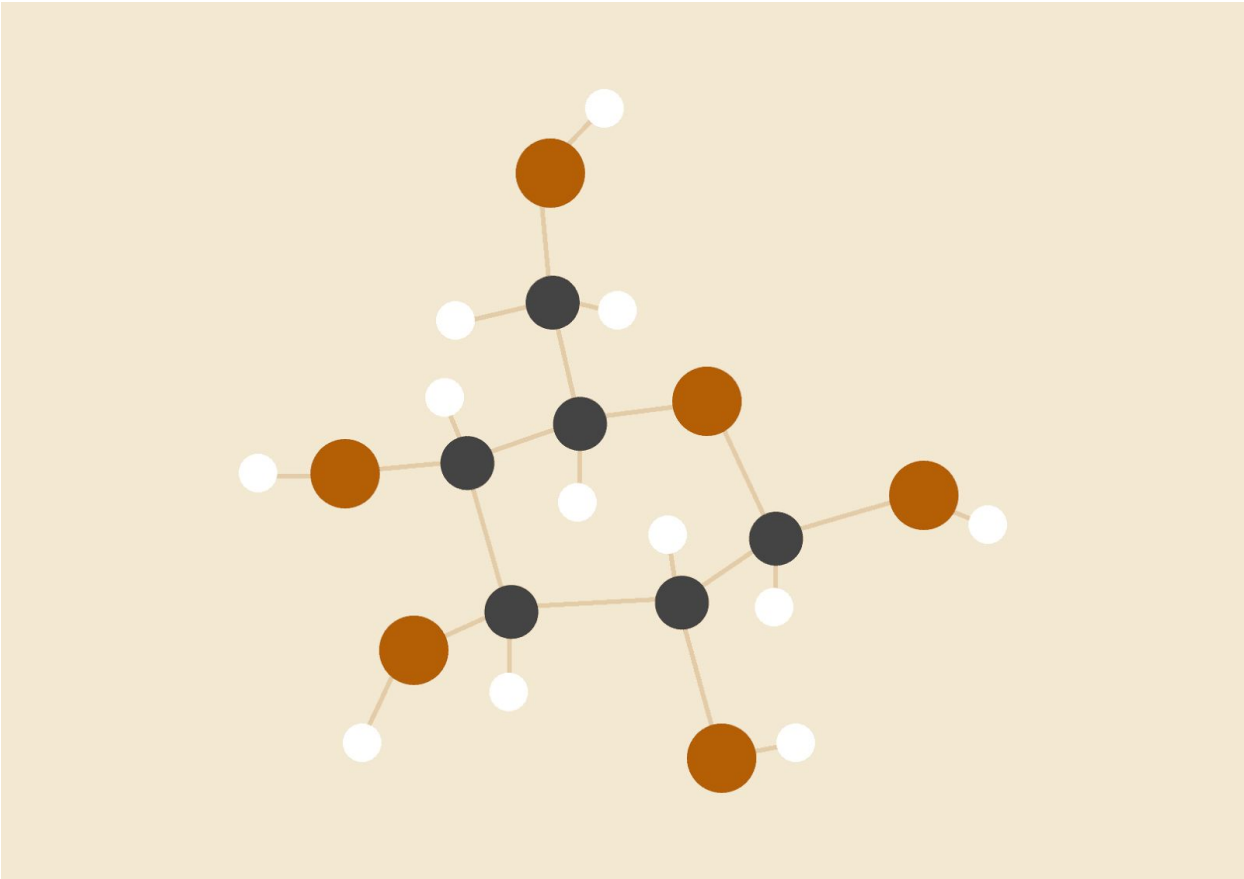
Computer Science

Report for Assignment 3



Brandon Gower-Winter

10.05.2017

CSC2001F

Problem

For assignment 3 one was tasked with calculating how well specific hash functions work using the Telephone Directory Data from previous assignments.

The hash functions to test were:

- Worst Case hash function h(x)=1
- Hash function 1(From our notes) to add unicode values
- Hash function 2(From our notest) to add shifted unicode values
- Another reasonable function of one's choice

A hash function takes a key(of any data type) and returns an integer value representation of that object. Calculating that integer value must be repeatable using the same constraints in order for a hash function to be considered viable.

Hash functions are used with HashTables which are used to map the key values into an array(usually) at the position of the returned hash value.

The hashing of Strings(Data type used in this assignment) can be done in many different ways. The most popular is some sort of summation of each character's Unicode value. This is extremely popular due to the String data type being a modified character array and thus the easiness of extracting each letter to modify is as simple as getting the next item in the array.

Application Design

For Assignment 3 I wanted to make everything as modular as possible and as such I came up with idea of representing a Hash Function as an object with a hash method. This hash function would return an integer. Each hash function would be it's own unique object and they would all inherit from a base HashFunction Class(that would just return the HashCode value provided by java). This allowed for the easy implementation of an Entropy Calculator and would make making a generic hash table possible.

Each hash function would inherit from the the HashFunction base class and override the ‘hash’ method which took in an object that could be type casted for each specific function and returned an integer value using the processes discussed below. This meant that any hash function would be of type HashFunction and could be stored in the entropy calculator for testing and still have the desired result.

The worst case hash function always returns the same value and as such was easy to implement. My implementation just returned 0(First element in the array) no matter what was put into it.

Hash Function 1 is a String specific hash and goes through every value in a string and adds up their Unicode values. Example: (“abc” would return 97 + 98 + 99 = 294) .

Hash Function 2 is also a String specific hash and it goes through every value in a string and add up their Unicode values whilst shifting each addition by some specified seed constant. Example: (“abc” with a seed of 37 would return ((97)*37 + 98)*37 + 99) = 136 518

As can be seen by the previous example the hash values can be large integer values that this results in two problems. The first being that you would need an array of 136 518 to be able to insert that into the correct place. That value is way to large and a quick fix is to take the modulus of the hash value = hash value % tablesize where table size is the size of the array used to store these values. The second problem is that sometimes the integer values get so big that they return a negative number. This is a result of the integer reaching it’s maximum possible value(int.max) and as such the integer ,once reaching it’s maximum threshold resets to its minimum threshold(int.min) this means that the integer can return a negative. To fix this I just made sure to take the absolute value of the hash value and return that.

The hash function of my own choosing is a modified version of the universal hash that I found on the University of Washington’s website. (https://courses.cs.washington.edu/courses/cse326/00wi/handouts/lecture15/sld016.htm)The object takes in a seed of integers and the size of the table. When hashing the each character in the string is visited by its corresponding position in the seed array. If that position doesn’t exist in the array then the seed value of the array at position 0 is used as the multiplicative value. The seed value can’t ever be a number larger than the table size and if a number is found to be bigger then it is changed to a value of 1. Example(“abc” with a seed of <3,5,8,7> and a table size of 10 returns (97*3) + (98*5) + (99 *8) = 1573)

The entropy calculator is uses a trivial counter hash table to count the occurrences of each possible hash value in the array. The trivial hash table is then used to calculate the probability of each hash value occurring then is used to calculate the entropy bit of each probability using the formula -P(x)*log(P(x)). Finally each entropy bit is added and the entropy of a hash function using a specified amount of data values(10 000 in this case) is returned.

Worst Case Hash Function h(x)=1

First 10 entries:

Hash value for Lueilwitz Candelario is: 0

Hash value for Wolf Mariane is: 0

Hash value for Gislason Kenna is: 0

Hash value for Hickle Leone is: 0

Hash value for Moore Gilbert is: 0

Hash value for Eichmann Eliane is: 0

Hash value for Wolff Jaylin is: 0

Hash value for Herzog Ally is: 0

Hash value for Lesch Ephraim is: 0

Hash value for Schaden Vernon is: 0

Summation of Unicode Values

First 10 entries:

Hash value for Lueilwitz Candelario is: 2011

Hash value for Wolf Mariane is: 1141

Hash value for Gislason Kenna is: 1357

Hash value for Hickle Leone is: 1123

Hash value for Moore Gilbert is: 1259

Hash value for Eichmann Eliane is: 1425

Hash value for Wolff Jaylin is: 1157

Hash value for Herzog Ally is: 1057

Hash value for Lesch Ephraim is: 1237

Hash value for Schaden Vernon is: 1358

Summation of Shifted Unicode Values

First 10 entries:

Hash value for Lueilwitz Candelario is: 9950

Hash value for Wolf Mariane is: 13410

Hash value for Gislason Kenna is: 19572

Hash value for Hickle Leone is: 19000

Hash value for Moore Gilbert is: 8410

Hash value for Eichmann Eliane is: 10293

Hash value for Wolff Jaylin is: 9527

Hash value for Herzog Ally is: 15886

Hash value for Lesch Ephraim is: 17169

Hash value for Schaden Vernon is: 15681

Universal Hash Function

First 10 entries:

Hash value for Lueilwitz Candelario is: 13225

Hash value for Wolf Mariane is: 5998

Hash value for Gislason Kenna is: 7874

Hash value for Hickle Leone is: 5902

Hash value for Moore Gilbert is: 7258

Hash value for Eichmann Eliane is: 8738

Hash value for Wolff Jaylin is: 6418

Hash value for Herzog Ally is: 5191

Hash value for Lesch Ephraim is: 7145

Hash value for Schaden Vernon is: 8523

Experimental Design

The point of this assignment was to test the effectiveness of different hash functions. This can be done by calculating the entropy of each of these functions and comparing the results. The entropy is the case of hash functions describes the likelihood of a random key inserted into a hash table using that specified hash function encountering a collision.

The entropy is calculated by taking the occurrences of each value in the array and calculating the probability of them by taking probability[pos] = array[pos]/numberOfEntires.

Then each probability is turned into an entropy bit by entropyBit[pos] = -1*proabability[pos]*log(probability[pos])

Then the entropy is calculated by summing up the entropy bits by entropy = entropyBit[0] + entropyBit[1] + … + entropyBit[tablesize - 1]

For this experiment 10 000 String values were tested where m(max value obtainable) = 20011. This was all done through code at runtime and each hash function was tested on newly instantiated HashEntropy class.

A seed of 7 was used for the Shifted Unicode Hash (To lessen the chance of integer overflow).

A seed of 912471914 was used for the Universal Hash (Chosen at random).

Results:

Because 10 000 entries were tested only the first 10 are shown. Because the the entropy calculation is all done with code relevant code snippets have been included in this section to illustrate where calculations were done.

Table:

Entry Name	Constant Hash	Hash Function 1	Hash Function 2	Universal Hash Function
Lueilwitz Candelario	0	2011	9950	13225
Wolf Mariane	0	1141	13410	5998
Gislason Kenna	0	1357	19572	7874
Hickle Leone	0	1123	19000	5902
Moore Gilbert	0	1259	8410	7528
Eichmann Eliane	0	1425	10293	8738
Wolff Jaylin	0	1157	9527	6418
Herzog Ally	0	1057	15886	5191

Lesch Ephraim	0	1237	17169	7145
Schaden Vernon	0	1358	15681	8523

To calculate the probability of each number occurring this code was used:

```
for(int i = 0; i < counter.length; i++)
        {
        probabilityArray[i] = counter[i]/(double)testCounter;
        }
        return probabilityArray;
```

To calculate each entropy bit:

```
for(int i = 0; i < counter.length; i++)
        {
        if(probabilityArray[i] == 0)
        computeArray[i] = 0;
        else
        computeArray[i] = -1*probabilityArray[i]*Math.log(probabilityArray[i]);
        }
        return computeArray;
```

To calculate the entropy:

```
for(int i = 0; i < counter.length; i++)
```

```
{
entropy += computeArray[i];
}
return entropy;
```

Entropy:

	Constant Hash	Hash 1	Hash 2	Universal Hash
Entropy	0	6.632	8.877	8.473

Conclusion

After analysis of the data collected there are some points to consider.

The Constant Hash function has the lowest entropy and is the worst Hash Function. This was to be expected though as the Constant Hash was used for the very purpose of generating an Entropy of 0. This means that for every key added to a hash table using the constant hash it will yield a collision(Except on the first key entry).

Hash 1 yielded an entropy of 6.632 which is much higher than that of the Constant Hash. This however does not compete with Hash 2 which yielded and entropy of 8.877(The highest of all the hashes). This means that it is preferable to use Hash 2 over Hash 1 in your implementation of Hash Table as it will result in less collisions.

The universal hash yielded and entropy of 8.473 which is close in value to Hash 2. It must be noted though that the entropy of the universal hash is dependent on the seed provided to it. If you give it a bad seed(EG: 9) it will return a bad entropy thus it is to my belief that finding the most appropriate seed will result in a Higher Entropy than that of Hash 2. This statement is unsubstantiated and is just a theory. More seed testing must be done to validate it.

Looking at the data the best Hash function to use is Hash 2 as it provided the highest entropy and as a result is the least likely to suffer from collisions. It must be noted however that supplying a different and more effective seed could result in the universal

hash being more effective.