APPLICATION ANALYSIS

HASH FUNCTIONS

PROBLEM DESCRIPTION

Hash tables use a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. It is possible that different elements, say different strings consist of exactly the same hash value. This phenomenon is called collision.

The problem with hash functions is when at least elements that were hashed from a particular data structure, say array hash or rather reference the same value in the hash table. This could cause conflict as well whenever one tries to index the element, in order to return it hash value. A hash function is created which does all the work. There different types of hash functions. Some hash functions just hash values regardless of whether there might be some duplicates, in terms of hash values. Some hash functions solve collisions, ensuring that each element is hashed to a different place. The limitation of hash tables is that it size is fixed, unlike other related data structures like array lists.

APPLICATION DESIGN AND IMPLEMENTATION

Hash function 1

```
public static int hash1(String key){
    int i =0;
    if(key.equals("")){
      return i;
    }
    else{
      i = 1;
      return i;
    }
}
```

- Hash1 method is of type integer, it takes in a string value, from a certain array. Checks if the string is an empty string and returns a 0 if true, if not it returns 1. Basically this hash function will hash all string values to 1 if they are not empty.

Hash function 2

```
public static int hash2(String key){
  int hashVal = 0;
  for(int i=0;i<key.length();i++){
    hashVal += key.charAt(i);}
  hashVal %= tablesize;
  if(hashVal < 0){
    hashVal+=tablesize;
  }
  return hashVal;
}
```

- This hash function returns an integer value. It takes in a string from a certain array and returns its hashval. It progresses in this by creating a variable hashval which will store the hash value of that string. The function adds the Unicode of all characters present in the string, then takes that Unicode values and apply a modulus of fixed table size, 20011. If the hashval is negative, it is incremented by table size.

Hash function 3

```
public static int hash3(String key){
  int hashVal = 0;
  for(int i=0;i<key.length();i++){
    hashVal = (37*hashVal) + key.charAt(i);
  }
  hashVal%=tablesize;
  if(hashVal<0){
    hashVal+=tablesize;
  }
  return hashVal;
}
```

- Hash3 is a hash function of type integer. The hash function basically calculates hash values by shifting the hash table by a factor of 37, creating some holes or rather empty spaces between each entered element. This is a sufficient hash function in comparison to the first hash function and the second hash fuction. Although collisions may occur, but it reduces the number of collisions in the hash table. The hashVal is multiplied by a factor of 37 and then add Unicode values. A modulus is applied to the hashVal by fixed table size of 20011. The hashVal is returned.

Hash function 4

```
public static int hash4(String key){
  int prime = 31;
  int hashVal = 1;
 hashVal = prime * hashVal + ((key == null) ? 0 : key.hashCode());
  return Math.abs(hashVal);
}
```

- hash4 is a hash function of type integer. The hash function takes in string as a parameter, multiply hashvalue by 31, and extract hashcode using hashCode(). It then returns the absolute value of the hashvaue.

Experimental Design and Results

Below is the main method that calculates the expected value or rather the entropy of hashvalues for all hash functions.

```
 public static void main(String[] args) throws IOException {
  Hashtable<String,Integer> table = new Hashtable<String,Integer>();
  ArrayList<String> mlist = new ArrayList<String>();
  ArrayList<Integer> mlist2 = new ArrayList<Integer>();
  ArrayList<Integer> hash_values1 = new ArrayList<Integer>();
  ArrayList<Integer> entropy_values2 = new ArrayList<Integer>();
  int index = 0;
  Scanner file = new Scanner(new FileInputStream("testdata"));
  while(file.hasNextLine()){
    String line = file.nextLine();
      String fullname = line.substring((line.lastIndexOf("|"))+1);
      table.put(fullname,index);
      mlist.add(fullname);
      index++;

    }

for(int i =0;i<mlist.size();i++){
  hash_values1.add(hash1(mlist.get(i)));
  //entropy_values1.add(hash1(mlist.get(i)));
  //entropy_values2.add(hash1(mlist.get(i)));
  //System.out.println(entropy_values.get(i));
}

//insert number of occurences of each duplicate
Map<Integer,Integer> count = new HashMap<Integer,Integer>();
  for (int ind : hash_values1) {
    if (count.containsKey(ind)) {
        count.put(ind, count.get(ind) + 1);
    } else {
        count.put(ind, 1);
    }
}

for (Map.Entry<Integer, Integer> entry : count.entrySet()) {
    mlist2.add(entry.getValue());    }

  //entropy calculation
  double partial_entropy = 0;
  for(int j=0;j<mlist2.size();j++){
    double value = (double)mlist2.get(j);
    double probability = value/10000;
    BigDecimal dec_val = BigDecimal.valueOf(probability);
    double formula = -probability*(Math.log(probability));;
    BigDecimal final_value2 = BigDecimal.valueOf(formula);
    partial_entropy =Math.abs(partial_entropy+formula);
  }
  BigDecimal final_entropy = BigDecimal.valueOf(partial_entropy);
  if(partial_entropy==0){
      System.out.println("Expected value: "+final_entropy.toPlainString()+" => Certainty");
  }
```

```
  else{
    DecimalFormat formater = new DecimalFormat("0.00000");

    if(partial_entropy==0){
    System.out.println("Expected value: "+formater.format(final_entropy)+" => Csertainty");

}

Else{  System.out.println("Expected value: "+formater.format(final_entropy)+" => Uncertainty");}
  }
  }
}
```

- This is the calculation of the entropy from 10 000 entries of data, specifically 10 000 hashvalues stored in the array list.

Calculation:

```
//entropy calculation
double partial_entropy = 0;
for(int j=0;j<mlist2.size();j++){
  double value = (double)mlist2.get(j);
  double probability = value/10000;
  BigDecimal dec_val = BigDecimal.valueOf(probability);
  double formula = -probability*(Math.log(probability));
  BigDecimal final_value2 = BigDecimal.valueOf(formula);
  partial_entropy =Math.abs(partial_entropy+formula);
}
BigDecimal final_entropy = BigDecimal.valueOf(partial_entropy);
if(partial_entropy==0){
    System.out.println("Expected value: "+final_entropy.toPlainString()+" => Certainty");
}
```

Output:

Hash function 1: entropy = 0.0 => Certainty

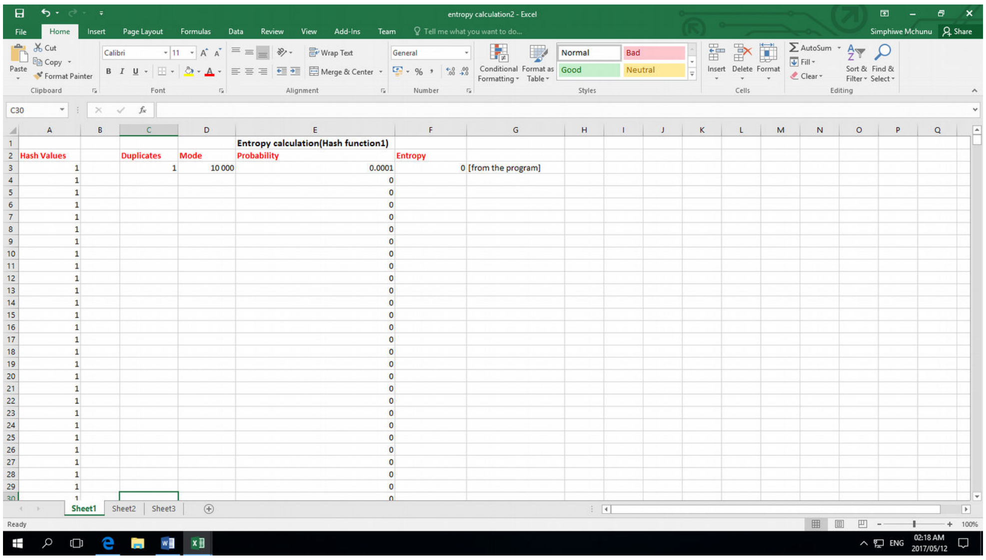
Hash function 2: entropy = 6.63181 => Uncertainty

Hash function 3: entropy = 8.89434 => Uncertainty

Hash function 4: entropy = 9.20618 => Uncertainty

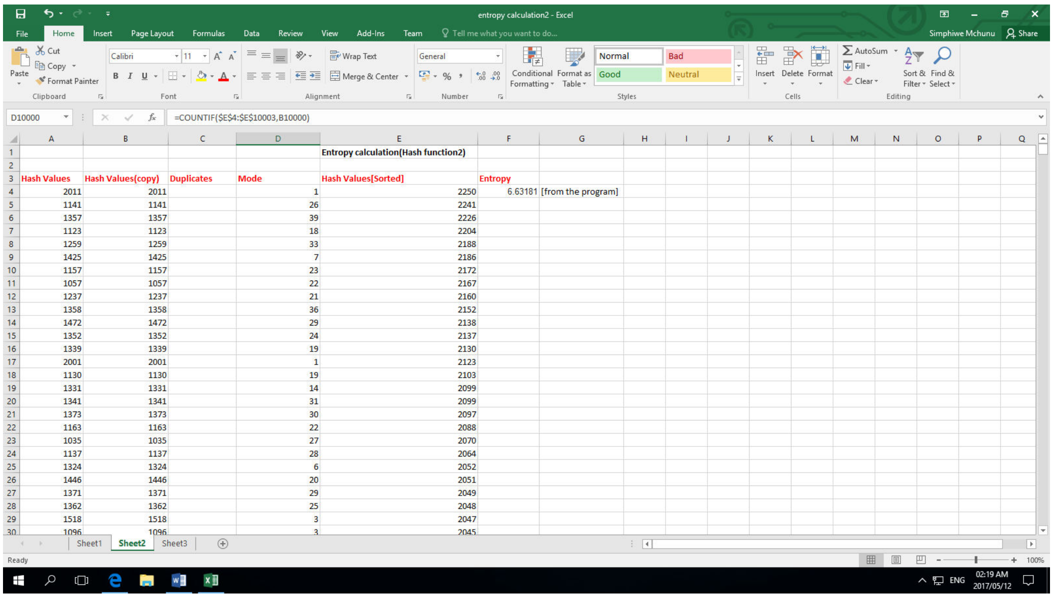
Tables

Hash function 1

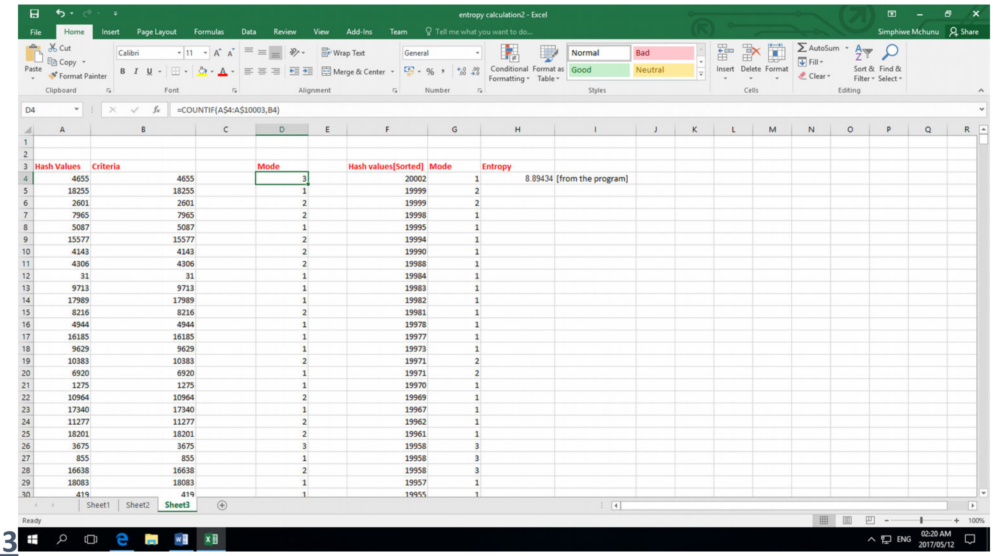


	A	B	C	D	E	F	G
1					Entropy calculation(Hash function1)		
2	Hash Values		Duplicates	Mode	Probability	Entropy	
3	1		1	10 000	0.0001	0	[from the program]
4	1				0		
5	1				0		
6	1				0		
7	1				0		
8	1				0		
9	1				0		
10	1				0		
11	1				0		
12	1				0		
13	1				0		
14	1				0		
15	1				0		
16	1				0		
17	1				0		
18	1				0		
19	1				0		
20	1				0		
21	1				0		
22	1				0		
23	1				0		
24	1				0		
25	1				0		
26	1				0		
27	1				0		
28	1				0		
29	1				0		
30	1				0		

Hash function 2



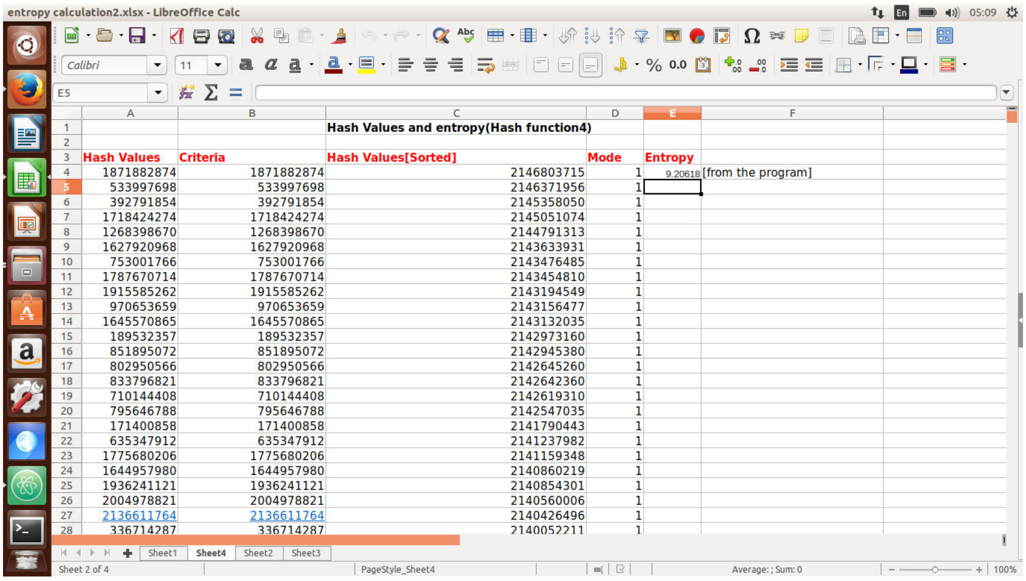
	A	B	C	D	E	F	G
1					Entropy calculation(Hash function2)		
2							
3	Hash Values	Hash Values(copy)	Duplicates	Mode	Hash Values[Sorted]	Entropy	
4	2011	2011		1	2250	6.63181	[from the program]
5	1141	1141		26	2241		
6	1357	1357		39	2226		
7	1123	1123		18	2204		
8	1259	1259		33	2188		
9	1425	1425		7	2186		
10	1157	1157		23	2172		
11	1057	1057		22	2167		
12	1237	1237		21	2160		
13	1358	1358		36	2152		
14	1472	1472		29	2138		
15	1352	1352		24	2137		
16	1339	1339		19	2130		
17	2001	2001		1	2123		
18	1130	1130		19	2103		
19	1331	1331		14	2099		
20	1341	1341		31	2099		
21	1373	1373		30	2097		
22	1163	1163		22	2088		
23	1035	1035		27	2070		
24	1137	1137		28	2064		
25	1324	1324		6	2052		
26	1446	1446		20	2051		
27	1371	1371		29	2049		
28	1362	1362		25	2048		
29	1518	1518		3	2047		
30	1096	1096		3	2045		



Hash Values	Criteria	Mode	Hash values[Sorted]	Mode	Entropy	
4655	4655	3	20002	1	8.89434	[from the program]
18255	18255	1	19999	2		
2601	2601	2	19999	2		
7965	7965	2	19998	1		
5087	5087	1	19995	1		
15577	15577	2	19994	1		
4143	4143	2	19990	1		
4306	4306	2	19988	1		
31	31	1	19984	1		
9713	9713	1	19983	1		
17989	17989	1	19982	1		
8216	8216	2	19981	1		
4944	4944	1	19978	1		
16185	16185	1	19977	1		
9629	9629	1	19973	1		
10383	10383	2	19971	2		
6920	6920	1	19971	2		
1275	1275	1	19970	1		
10964	10964	2	19969	1		
17340	17340	1	19967	1		
11277	11277	2	19962	1		
18201	18201	2	19961	1		
3675	3675	3	19958	3		
855	855	1	19958	3		
16638	16638	2	19958	3		
18083	18083	1	19957	1		
419	419	1	19955	1		

Hash function 3

Hash function 4



Hash Values and entropy(Hash function4)

Hash Values	Criteria	Hash Values[Sorted]	Mode	Entropy	
1871882874	1871882874	2146803715	1	9.20618	[from the program]
533997698	533997698	2146371956	1		
392791854	392791854	2145358050	1		
1718424274	1718424274	2145051074	1		
1268398670	1268398670	2144791313	1		
1627920968	1627920968	2143633931	1		
753001766	753001766	2143476485	1		
1787670714	1787670714	2143454810	1		
1915585262	1915585262	2143194549	1		
970653659	970653659	2143156477	1		
1645570865	1645570865	2143132035	1		
189532357	189532357	2142973160	1		
851895072	851895072	2142945380	1		
802950566	802950566	2142645260	1		
833796821	833796821	2142642360	1		
710144408	710144408	2142619310	1		
795646788	795646788	2142547035	1		
171400858	171400858	2141790443	1		
635347912	635347912	2141237982	1		
1775680206	1775680206	2141159348	1		
1644957980	1644957980	2140860219	1		
1936241121	1936241121	2140854301	1		
2004978821	2004978821	2140560006	1		
2136611764	2136611764	2140426496	1		
336714287	336714287	2140052211	1		

Discussion and Conclusion:

The tables above indicate the mode or number of occurences a hash value of different stings appears. The tables are sorted out in order from smallest to largest. The entropy value was calculated using from a java program. The number of occurences is thus what we call collision. As the tables show, there are many values that hash to the same place, even though the strings might differ. This is a huge problem because whenever a certain value is indexed from the hashtable it will point to different elements of String and not a unique string.

The entropy as defined before is the expected value of the probability of occurences. Output for hash function 1 is 0, this is certainty of the outcome while other functions give entropy greater than 1, this represents uncertainty.

In concluding, when the entropy is 0, this means that we are certain about the expected value. From table 1, the hash table consists of only 1s, from the initial index up to 10 000, hence the data won't be random. In general, the entropy is the indication of the randomness collected by the operating system.