REPORT

PROBLEM

We have been provided with a telephone directory file that has 10000 entries or items. Faster access to the directory to get information is required, although a big number of 10000 entries could cause the access to be slow, hash tables could be used for an electronic telephone directory which could result in faster and better efficient operations being performed.

The hash table stores the items in a location determined by their content. The issues we might encounter is having more keys than the table size. The keys are not integers and might be the same but have different values which could cause collision.

Hash functions will be used to map items to integers and modulus to solve the integer key size problem.

DESIGN AND IMPLEMENTATION OF HASH FUNCTIONS

1. Hash function worst case h(x)=1

```
public int worstHash(String key){

return 1;
}
```

The code above depicts the design of the worst-case hash function. The function is supplied with a String key from the testdata file that was provided. The function generates or rather, returns a constant hash value of 1 for each key. The hash value of one is returned for all the keys. The insert, delete and find algorithms for this functions have O(1).

2.Hash function 1

```
public static int hash1 (String key) {
  int hashVal = 0;
  for( int i = 0; i < key.length(); i++ )
        hashVal += key.charAt(i);
   return hashVal % tableSize;
 }
```

The code above depicts the design of Hash function 1. The function is supplied with a String key. An int hashVal variable is initialised that is used inside the for loop, to add the Unicode values of the key. The key is looped and each Unicode value is added to the hashVal that was initialised. The final hashVal is mod by the tablesize(assumed to be 20011), this value is then returned as the hash value for the given key.

3.Hash function 2

```
public static int hash2 (String key) {
   int hashVal = 0;
    for( int i = 0; i < key.length(); i++ )
     hashVal = (37 * hashVal) + key.charAt(i);
   return hashVal % tableSize;
   }
```

The code above depicts the design of Hash function 2. The function is supplied with a String key. An int hashVal variable is initialised that is used inside the for loop, to add the shifted values of the key. The key is looped and each Unicode value is added to the hashVal that is multiplied by 37 that was initialised. The final hashVal is mod by the

tablesize(assumed to be 20011), this value is then returned as the hash value for the given key.

4. Hash function 3

```
public static int hash3 (String key) {
    int hashVal = 0;
     for( int i = 0; i < key.length(); i++ )
         hashVal = (17 * hashVal)+ key.charAt(i);
   return hashVal % tableSize;
    }
```

The code above depicts the design of Hash function 3 which is similar to hash function 2. The function is supplied with a String key. An int hashVal variable is initialised that is used inside the for loop, to add the shifted values of the key. The key is looped and each Unicode value is added to the hashVal that is multiplied by 17 that was initialised. The final hashVal is mod by the tablesize(assumed to be 20011), this value is then returned as the hash value for the given key.

HASH VALUE ILLUSTRATIONS FOR THE HASH FUNCTIONS

1. Worst case hash function, h(x)=1

Worst case	unique	Num of occurrence	probability	Entropy cal
1	1	10000	1	0
1	1			0
1				
1				

The table illustrates a sample of hash values, unique hash values, number of occurrences, probability and entropy values of the hash values for the worst case hash function. Since the function returns 1 as the hash value for each

each key, there is only one unique value,1. The number of occurrences of this hash value is then 10000 as is the same for all of them. The probability is also 1.

2. Hash function 1

hashValues1	unique	Num of occurrence	probability	Entropy cal
2011	2011	1	0,0001	0,000921
1141	1141	26	0,0026	0,015476
1357	1357	39	0,0039	0,021632
1123	1123	18	0,0018	0,011376
1259	1259	33	0,0033	0,018856
1425	1425	7	0,0007	0,005085
1157	1157	23	0,0023	0,013972
1057	1057	22	0,0022	0,013462
1237	1237	21	0,0021	0,012948
1358	1358	36	0,0036	0,020257
1472	1472	29	0,0029	0,016945
1352	1352	24	0,0024	0,014477
1339	1339	19	0,0019	0,011905
2001	2001	1	0,0001	0,000921
1130	1130	19	0,0019	0,011905
1331	1331	14	0,0014	0,0092
1341	1341	31	0,0031	0,017907
1373	1373	30	0,003	0,017427
1163	1163	22	0,0022	0,013462

The table illustrates a sample of hash values, unique hash values, number of occurrences, probability and entropy values of the hash values hash function 1. The first column shows the hash value for each key. The values are different because the keys are different, although some might be the same.

3. Hash function 2

hashValues2	Unique	Num of occurrence	probability	Entropy cal
4655	4655	1	0,0001	0,000921
-1756	-1756	1	0,0001	0,000921
2601	2601	1	0,0001	0,000921
7965	7965	1	0,0001	0,000921
5087	5087	1	0,0001	0,000921

15577	15577	1	0,0001	0,000921
-15868	-15868	2	0,0002	0,001703
4306	4306	1	0,0001	0,000921
31	31	1	0,0001	0,000921
9713	9713	1	0,0001	0,000921
-2022	-2022	1	0,0001	0,000921
-11795	-11795	1	0,0001	0,000921
-15067	-15067	1	0,0001	0,000921
-3826	-3826	1	0,0001	0,000921
9629	9629	1	0,0001	0,000921
-9628	-9628	2	0,0002	0,001703
6920	6920	1	0,0001	0,000921
-18736	-18736	1	0,0001	0,000921
10964	10964	2	0,0002	0,001703

4. Hash function 3

hashValues3	Unique	Num of occurrence	probability	Entropy cal
328	328	1	0,0001	0,000921
-269	-269	1	0,0001	0,000921
3856	3856	1	0,0001	0,000921
4370	4370	1	0,0001	0,000921
-2773	-2773	2	0,0002	0,001703
2323	2323	2	0,0002	0,001703
-10393	-10393	1	0,0001	0,000921
-2597	-2597	1	0,0001	0,000921
12242	12242	1	0,0001	0,000921
19677	19677	1	0,0001	0,000921
-9570	-9570	1	0,0001	0,000921
2920	2920	2	0,0002	0,001703
12910	12910	3	0,0003	0,002434
-10678	-10678	1	0,0001	0,000921
-15099	-15099	2	0,0002	0,001703
17554	17554	1	0,0001	0,000921
-11350	-11350	1	0,0001	0,000921
4530	4530	1	0,0001	0,000921
19073	19073	1	0,0001	0,000921

Table 3 and 4 above illustrates a sample of hash values, unique hash values, number of occurrences, probability and entropy values of the hash values of hash function 2 and 3, respectively. The first column shows the hash value for each key. The values are different because the keys are different, although

some might be the same. One noticeable different is that the hash values are positive or negative. This could be due to the hash values being shifted.

DESIGN COMPARISONS AND RESULTS

-Worst case hash function

The worst-case hash function is very simple. It gets a key and returns 1 for each key. This results in the hash values being the same regardless of the keys being different or not. The function does not modulus the constant hash value of 1 by the table size, this differentiates it from the other 3 functions used to generate the hash values.

-Hash function 1,2 &3

The design for these functions is very similar in that the key is looped to access the Unicode value of each char value of the key. The hash values are modulus by the table size in each function which might combat any collisions.

Hash function 1 is different from the other 3 functions in that it adds the Unicode values of the key to generate the hash value and then modulus it by the table size. This results in the hash values being greater than 1 and being positive values.

Hash function 2 and 3 are different from the other functions in that each Unicode value is shifted or multiplied by the values 37 and 17, respectively and then added. Multiplying ensures uniqueness. Thus, more unique values of hash values were observed than for function 1 and worst case function. The hash values of these two functions are either positive or negative which can be seen in table 3 and 4 above.

Functions	Entropy value of each function
h(x)=1	0
Hash function 1	6,63181
Hash function 2	9,04599
Hash function 3	9,04368

The table depicts the sum entropy value of the hash values for each function. The entropy is calculated using the probability of the hash values, h(x). For each hash value, we calculated-P(x) Log (P(x)), which is the entropy value for

each hash value. The entropy is then the sum of the individual entropy values: H(X).

The worst-case hash function as can be seen from the table; has the lowest entropy value. Hash function 2 has the highest entropy value, this could be due that each Unicode of the key was multiplied and added. Thus, function 2 observes the higher entropy of all the other functions. A higher entropy is always better as it could allow for faster operations because the values are unique.

DISCUSSIONS AND CONLUSION

- Returning a constant hash value that is not mod by the table size might cause collision and results in low entropy value. A low entropy value might not be good for operations.
- Adding Unicode values of the key results in positive hash values that are greater that one.
- Shifting Unicode values result in hash values that are positive or negative. And it ensures uniqueness and increase the hash values.
- Shifting the Unicode value by a larger number results in a higher entropy value. A higher entropy value is always better. As seen in hash function 2 entropy number, where 37 was used to shift as opposed to 17 used in function 3.
- Modulus of the hash value by the table size combats collision that may occur.