**QUESTION**: Implement Hash table. Pick 20 random words.

Each word must be of different lengths, maximum length 8 and minimum length 3.

The words will be of letters a-zA-Z0-9 and the space character.

Insert them into a hash table.  The hashing algorithm uses ASCII table values for each character and adds up the values. Initially the table size is 31.  The program should increase the table size and rehash.

The collision resolution is done using quadratic probing.

At the end print the total number of collisions you get.

If the table size is more than 20,000 and the words to insert are 8000, is the above hashing algorithm adequate ? why ?

**ANSWER**

Load Factor and rehashing handling: Before adding new element, load factor is checked. While checking load factor we assume the new value is already added and if this load factor is greater than 0.5, rehashing is performed and then the new key is added.

Handling made for collision count: Flag has been added to consider collisions during rehashing in the entire collision count. If set as true, collisions detected during rehashing would be also be counted. If set as false, the collisions detected during rehashing would be ignored in the final collision count.

Efficient calculation of quadratic probing: Since quadratic probing requires \* and % operations. It is avoided by calculation new hash value from old hash value. Hi = Hi-1 + 2i – 1 (mod N)

Our key is a string of maximum length 8. Ascii value of each character would be less than 127. Sum of ascii value of all 8 characters will not exceed 127\*8 which is 1016, this means the sum would never exceed 1016 and our hash function of finding sum of all characters can assume values less than 1016. Since the function does not distribute the keys well, there will be many collisions. Use of quadratic probing will eliminate the primary clustering of elements but there is not much control over the secondary clustering. Even if we add 8000 elements in a table of size 20000 using existing algorithm, there will be performance issues observed in case of finding, adding or deleting keys.

Current algorithm can be changed by multiplying the weight of 63 (52 alphabets, 10 numbers, 1 space) to the hash value before adding ascii value. Also, for collision resolutions, Double hashing can be implemented to avoid secondary clustering problem.

**Output screenshot:**

Input keys: "Hey",

"There",

"WhatsUp",

"Numbers",

"One1",

"Two2",

"Three3",

"4Four",

"Hi5",

"6Sticks",

"7Eleven",

"8Byte",

"9One1",

"OhGOD",

"Graphs8",

"Trees3",

"Linked12",

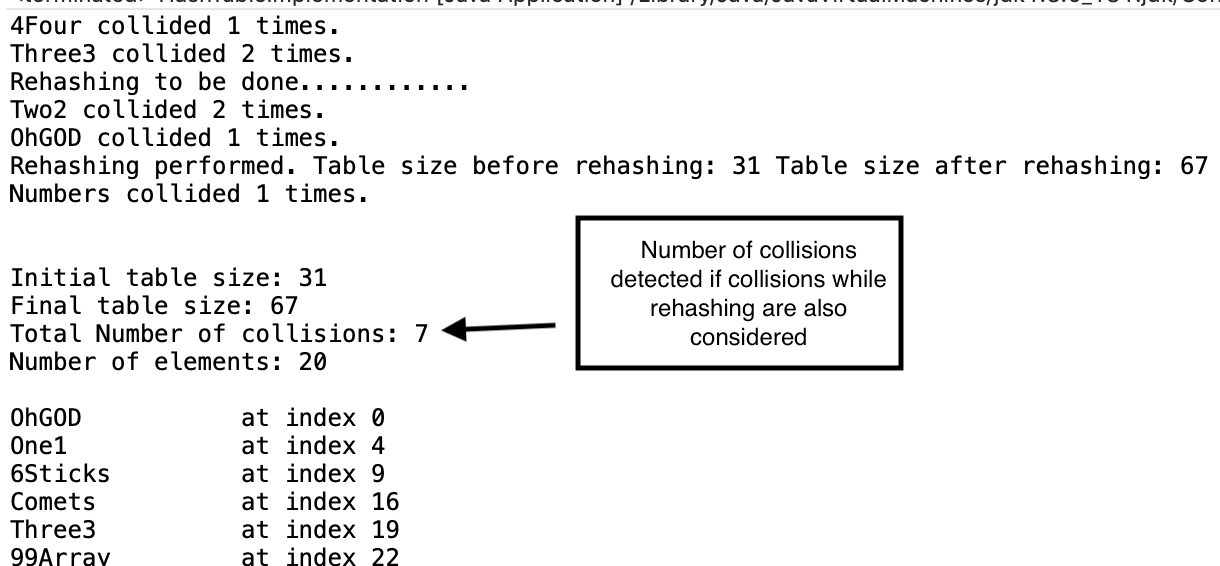
"99Array",

"Comets",

"UTD"

Screenshot:

Case1:



Case2:

