Let us look at an example for linear probing to avoid the collision. Let us consider the following set of keys [56, 1072, 97, 84, 60] and the hash table size of 5. When we apply mod based hash function and start placing the keys in the appropriate slots we get the placement as depicted in Figure 4.

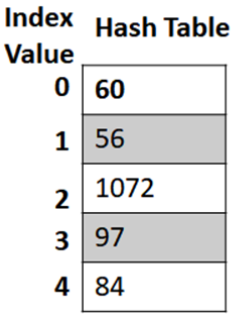

Figure 4 Linear Probing

The value 56 goes to position 1 due to the mod value of (56 mod 5) operation. Similarly, 1072 assumes position 2. However, when we try to place the next element 97 we end up with a collision at slot 2. So, we find the next empty slot at slot 3 and place 97 there. Rest of the elements 84 and 60 go to the positions 4 and 0 respectively based on their mod values.

10.4.3 Double Hashing

We can avoid the challenges with primary clustering and secondary clustering using the double hashing strategy. Double hashing uses a second hash function to resolve the collisions. The second hash function is different from the primary hash function and uses the key to yield non-zero value.

The first hash function in the double hashing finds the initial slot for the key and the second hash function determines the size of jumps for the probe. The i^{th} probe is defined as follows

$h(x,i) = (h_1(x) + i * h_2(x)) \bmod m$ where m is the hash table size

Let us look at an example for the double hashing to understand it better. Let us consider a hash table of size 5 and we use the below given hash functions:
H1(x) = x mod 5
H2(x) = x mod 7

Let us try to insert two elements 60 and 80 into the hash table. We can place the first element 60 at slot 0 based on the hash function. When we try to insert the second element 80, we face a collision at slot 0. For the first iteration we apply the double hashing as follows:
H(80,1) = (0+1*3) mod 5 = 3

Hence, we now place the element 80 in slot 3 to avoid collision as depicted in figure 5.

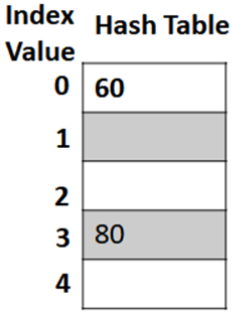

Figure 5 Double Hashing Example

10.5 COMPARISON OF COLLISION RESOLUTION METHODS

Comparison of linear probing, quadratic probing and double hashing is given below:

Collision Resolution Technique	Separate Chaining	Open Addressing - Linear Probing	Open Addressing - Quadratic Probing	Double Hashing
Primary clustering	No	Yes	No	No
Secondary clustering	No	No	Yes	No
Key storage in Hash table	Inside & Outside Hashtable	Inside Hash table	Inside Hash table	Inside Hash table

10.6 LOAD FACTOR AND REHASHING

The hash table provides constant time complexity for operations such as retrieval, insertion and deletion with lesser keys. As the key size grows, we run out of vacant spots in the hash table leading to collision that impacts the time complexity. When the collision happens, we need to re-adjust the hash table size so that we can accommodate additional keys. Load factor defines the threshold when we should re-size the hash table to main the constant time complexity.

Load factor is the ratio of the elements in the hash table to the total size of the hash table. We define load factor as follows:

Load factor = (Number of keys stored in the hash table)/Total size of the hash table.

In open addressing as all the keys are stored within the hash table, the load factor is <=1. In separate chaining method as the keys can be stored outside the hash table, there is a possibility of load factor exceeding the value of 1.

If the load factor is 0.75 then as soon as the hash table reaches 75% of its size, we increase its capacity. For instance, lets consider the hash table of size 10 with load factor of 0.75. We can insert seven hash keys into this hash table without triggering the re-size. As soon as we add the eighth key, the load factor becomes 0.80 that

exceeds the configured threshold triggering the hash table resize. We normally double the hash table size during the resize operation.

10.6.1 Rehashing

when the load factor exceeds the configured value, we increase the size of hash table. Once we do it we should also re-compute the hash values for the existing keys as the size of the hash table has changed. This process is called "rehashing". Rehashing is a costly exercise especially if the key size is huge. Hence it is necessary to carefully select the optimal initial size of the hash table to start with.

Given below are the high-level steps for rehashing:

1. For each new key insert into the hash table, compute the load factor
2. If the load factor exceeds the pre-defined value then increase the hash table size (normally we double the hash table size)
3. Recompute the hash value (rehash) for each of the existing elements in the hash table.

Let us look at the rehashing with an example. We have a hash table of size 4 with load factor of 0.60. Let's start by inserting these elements – 30, 31 and 32. We can insert 30 at slot 2 and 31 at slot 3 and 32 at slot 0. Insertion of 32 triggers the hash table resize as the load factor has breached the threshold of 0.60. As a result, we double the hash table size to 8.

With the new hash table size, we need to recalculate the hash values of the already inserted keys. Key 30 will now be placed in slot 6, key 31 will be placed in slot 7 and key 32 in slot 0.

☞ Check Your Progress – 2

1. The main techniques of open addressing are _____
2. Load factor triggers _____
3. Linear probing leads to _____ clustering
4. _____ probing make large jumps leading to secondary clustering
5. Second hash function in double hashing can result in 0. True/False
6. _____ should be done for the existing keys of the hash table post resizing.

10.7 SUMMARY

In this unit, we started discussing the main motivations for the hashing. Hashing allows us to store and retrieve large data efficiently.
Index mapping uses the input values as direct index into the hash table. Index mapping requires huge hash table size leading to inefficiencies. When we handle large size input values we encounter collision where multiple input values compete for the same spot in the hash table. The main collision resolution techniques are separate chaining and open addressing. In separate chaining we chain the values that get mapped to a spot. We use linear probing and quadratic probing as part of open addressing technique to find the next available spot. We use two hash functions as part of double hashing. Load factor determines the trigger for the hash table resizing and once the hash table is resized, we re-compute the hash values of the existing keys using rehashing.

10.8 SOLUTIONS/ANSWERS

☞ Check Your Progress – 1

1. Index mapping
2. handling non integer keys and large hash table size
3. computing efficiency, uniform distribution, deterministic and minimal collisions
4. O(1)
5. Linked List
6. True

☞ Check Your Progress – 2

1. linear probing, quadratic probing and double hashing
2. resizing of hash table
3. primary
4. quadratic
5. False
6. Rehashing

10.9 FURTHER READINGS

Horowitz, Ellis, Sartaj Sahni, and Susan Anderson-Freed. *Fundamentals of data structures*. Vol. 20. Potomac, MD: Computer science press, 1976.

Cormen, Thomas H., et al. *Introduction to algorithms*. MIT press, 2022.

Lafore, Robert. *Data structures and algorithms in Java*. Sams publishing, 2017.

Karumanchi, Narasimha. *Data structures and algorithms made easy: data structure and algorithmic puzzles*. Narasimha Karumanchi, 2011.

West, Douglas Brent. *Introduction to graph theory*. Vol. 2. Upper Saddle River: Prentice hall, 2001.

https://en.wikipedia.org/wiki/Hash_function#Trivial_hash_function

https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_data_structures/Hash_tables_and_hashing

https://ieeexplore.ieee.org/book/8039591