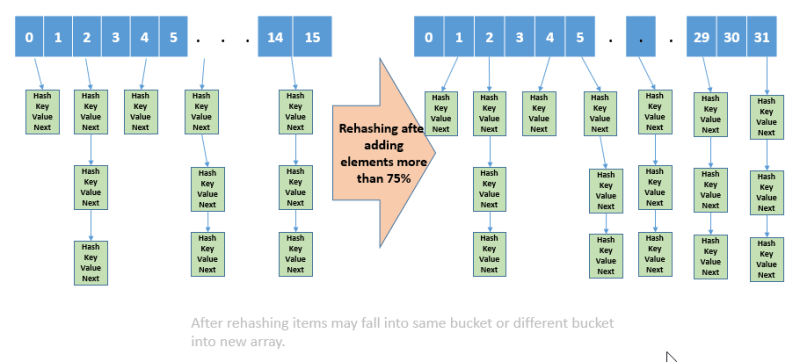
**What is Rehashing?**

Rehashing is the process of re-calculating the hashcode of already stored entries (Key-Value pairs), to move them to another bigger size hashmap when the threshold is reached/crossed.

Rehashing of a hash map is done when the number of elements in the map reaches the maximum threshold value. Java specification suggests that the Good load factor value is .75 and the default initial capacity of HashMap is 16. Once the number of elements reaches or crosses 0.75 times the capacity, the complexity increases, so to overcome this the size of the array is increased by double and all the values are hashed again and stored in the new array of double size. This is done to reduce the complexity and maintain low load factor. In this case, when the number of elements is 12, rehashing occurs. (0.75 \* 16 = 12).

* **Why Rehashing is done?**

Rehashing is done because whenever a new key value pair is inserted into map, the load factor increases and due to which complexity also increases. And if complexity increases our HashMap will not have constant O(1) time complexity. Hence rehashing is done to distribute the items across the hashmap as to reduce both load factor and complexity, So that get() and put() have constant time complexity of O(1). After rehashing is done existing items may fall in the same bucket or different bucket.



* **Rehashing – How it’s done?**

**Steps for Rehashing as follows:**

1. For every new entry into the map, check the load factor.
2. If the load factor is greater than its threshold value (default 0.75 for HashMap), then start Rehash.
3. For Rehashing, initialize a new array of double the size of the previous one.
4. Copy all elements into a new array and make it the new bucket array.

* **What is Load factor in HashMap?**

Load factor in HashMap is basically a measure that decides when exactly to increase the size of the HashMap to maintain the same time complexity of O(1).

Load factor is defined as (m/n) where n is the total size of the hash table and m is the preferred number of entries which can be inserted before an increment in the size of the underlying data structure is required.

This is the traditional dilemma of all array-based data structures:

* Make the table too small, performance degrades and the table may overflow
* Make the table too big, and memory gets wasted.

*Rehashing* or *variable hashing* attempts to circumvent this dilemma by expanding the hash table size whenever it gets too full.

**Note:** If you are going to store a really large no of elements in the hashmap then it is always good to create HashMap with sufficient capacity upfront as rehashing will not be done frequently, this is more efficient than letting it to perform automatic rehashing.

**1. Expanding the hash Table**

For example, using open addressing (linear probing) on a table of integers with hash(k)=k (assume the table does an internal % hSize): We know that performance degrades when λ > 0.5

Solution: rehash when more than half full

https://qphs.fs.quoracdn.net/main-qimg-76bf5a2c171164ec0f1f483e31b416aa

So if we have this table, everything is fine.

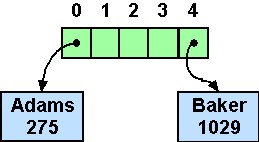
But if we try to add another element (24), then more than half the slots are occupied…

https://qphs.fs.quoracdn.net/main-qimg-80f39523d418b0552f8ecbe51db8bfd2

So we expand the table, and use the hash function to relocate the elements within the larger table…

https://qphs.fs.quoracdn.net/main-qimg-af8c1dc71e05e680ec33163110ec2ec8

**2. Saving the Hash Values**



The rehashing operation can be quite lengthy. Luckily, it doesn't need to be done very often.

We can speed things up somewhat by storing the hash values in the table elements along with the data so that we don't need to recompute the hash values. Also, if we structure the table as a vector of *pointers to* the hash elements, then during the rehashing we will only be copying pointers, not the entire (potentially large) data elements.