Load Factor & Initial Capacity:

* HashMap

1. Iteration over collection views requires time proportional to the "capacity" of the HashMap instance (the number of buckets) plus its size (the number of key-value mappings).
2. It's very IMPORTANT NOT to set the "initial capacity" too high or the "load factor" too low if iteration PERFORMANCE is IMPORTANT. An instance of HashMap has two parameters that affect its performance:
3. Initial Capacity:

* The capacity is the number of buckets in the "hash table", and the initial capacity is simply the capacity at the time the "hash table" is created.
* default initial capacity of HashSet: 16

1. Load Factor:

* The load factor is a Measure of how full the "hash table" is Allowed to Get before its capacity is automatically increased.

When the number of entries in the "hash table" EXCEEDS the product of the load factor and the current capacity, the hash table is "rehashed" (that is, internal data structures are rebuilt) so that the "hash table" has approximately Twice the number of buckets.

* default load factor of HashSet: 0.75

1. As a general rule speaking:
2. The default load factor (.75) offers a Good TradeOff between Time and Space Costs (Hard Disk / SSD).
3. Higher value of load factor decreases the space overhead but increase the lookup cost (reflected in most of the operations of the HashMap class, including get and put).

The expected number of entries in the map and its load factor should be taken into account when setting its initial capacity, so as to minimize the number of rehash operations.

1. If the initial capacity is greater than the maximum number of entries divided by the load factor, no rehash operations will ever occur.
2. If many mappings are to be stored in a HashMap instance, creating it with a sufficiently large capacity will allow the mappings to be stored more efficiently than letting it perform automatic rehashing as needed to grow the table.
3. Note that using many keys with the same hashCode() is a sure way to Slow Down Performance of any hash table. To improve impact, when keys are Comparable, this class may use comparison order among keys to help break ties.

Diagram

Description automatically generated

Let's resized the initial capacity to be 100 and the load factor to be .75 as following example:

1. 100 \* 75% = 75 (Threshold)

That is, while the 75th entry (i.e., key-value-pair) is added, hashMap will increase the capacity from 100 to 200.

1. 200 \* 75% = 150 (Threshold)

While adding the 150th entry, hash map will increase the capacity from 200 to 300.

1. 300 \* 75% = 225 (Threshold)

While adding the 225th entry, hash map will increase the capacity from 300 to 400.

Index lookup costs:

1. The database server incurs additional costs when it finds a row through an index.
2. The index is stored on disk, and its pages must be read into memory with the data pages that contain the desired rows.
3. An index lookup works down from the root page to a leaf page.
4. The root page, because it is used so often, is almost always found in a page buffer.
5. The odds of finding a leaf page in a buffer depend on the size of the index, the form of the query, and the frequency of column-value duplication.
6. If each value occurs only once in the index and the query is a join, each row to be joined requires a non-sequential lookup into the index, followed by a non-sequential access to the associated row in the table.

Please refer to the link:

-> https://www.baeldung.com/java-hashmap-load-factor

-> https://www.baeldung.com/java-hashmap

-> https://www.ibm.com/docs/en/informix-servers/14.10?topic=query-index-lookup-costs