# **Separate Chaining**

* **Separate Chaining** is a collision resolution strategy that keeps a **list** of all elements that **hash to the same value**.
* We can now think of the hash table as an **array of linked chains**.
* We can use Java’s standard library List implementations to accomplish this.
* Each entry table[i] is the head pointer to a list of linked nodes containing the items that the hash function has mapped into location i.

**Diagram

Description automatically generated**

* If space is tight, it might be preferable to avoid their use (since those lists are doubly linked and waste space).
* With separate chaining, the size of the dictionary is dynamic and can exceed the size of the hash table, because each linked chain can be as long as necessary.
* However, the length of these chains affects the efficiency of retrievals and removals.
* If all keys were to map to the same index in the array, the add, remove and retrieval operations could all take up to O(N) time.
* Even so, separate chaining is the most time-efficient collision-resolution scheme.

**Example**

* We assume, for this section, that the keys are the first 10 perfect squares and that the hashing function is simply *hash*(*x*) = *x* mod 10. (The table size is not prime but is used here for simplicity.) Figure 5.5 should make this clear.
* To perform a search, we use the hash function to determine which list to traverse. We then search the appropriate list. To perform an insert, we check the appropriate list to see whether the element is already in place (if duplicates are expected, an extra field is usually kept, and this field would be incremented in the event of a match). If the element turns out to be new, it is inserted at the front of the list, since it is convenient and also because frequently it happens that recently inserted elements are the most likely to be accessed in the near future.
* The class skeleton required to implement separate chaining is shown in Figure 5.6. The hash table stores an array of linked lists, which are allocated in the constructor.