Hashing – Separate Chaining

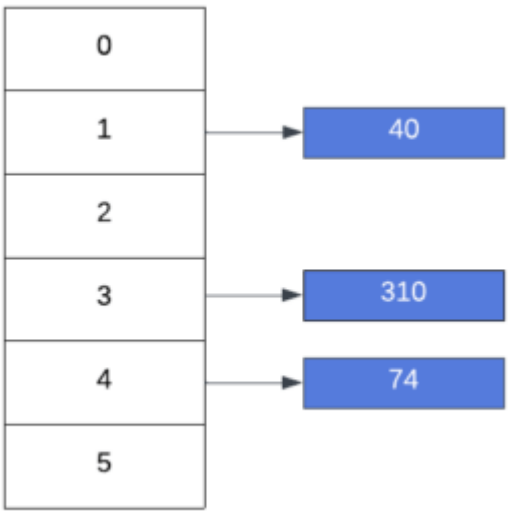
Separate chaining is a collision resolution technique to store elements in a hash table, which is represented as an array of linked lists. Each index in the table is a chain of elements mapping to the same hash value. When inserting keys into a hash table, **we generate an index and mitigate collisions by adding a new element to the list at that particular index.** Nonetheless, preventing duplicate keys in our table will slightly change the insert algorithm since we must ensure the key is not in the list.

Searching for a key, on the other hand, **requires traversing through the list at the generated index** until we find the element or reach the end of the list.

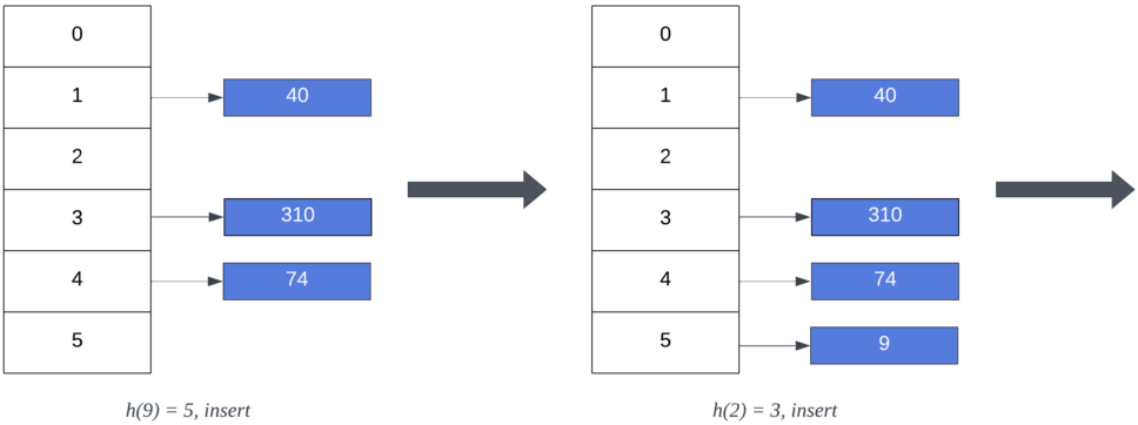
Because each index of the table is a list, we can store elements in the same index that results from the hash function. This is a unique characteristic of separate chaining, since other algorithms, such as linear or quadratic probing, search for an alternative index when finding the position of a key after a collision.

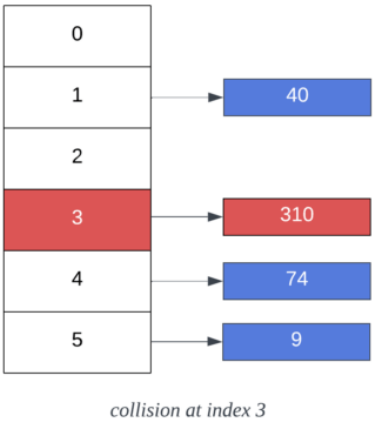
Before diving into the algorithm, let's assume we have the following set of keys and an arbitrary hash function that yields the following:

Now, suppose our hash table has an arbitrary length of 6, and we want to insert the remaining elements of :

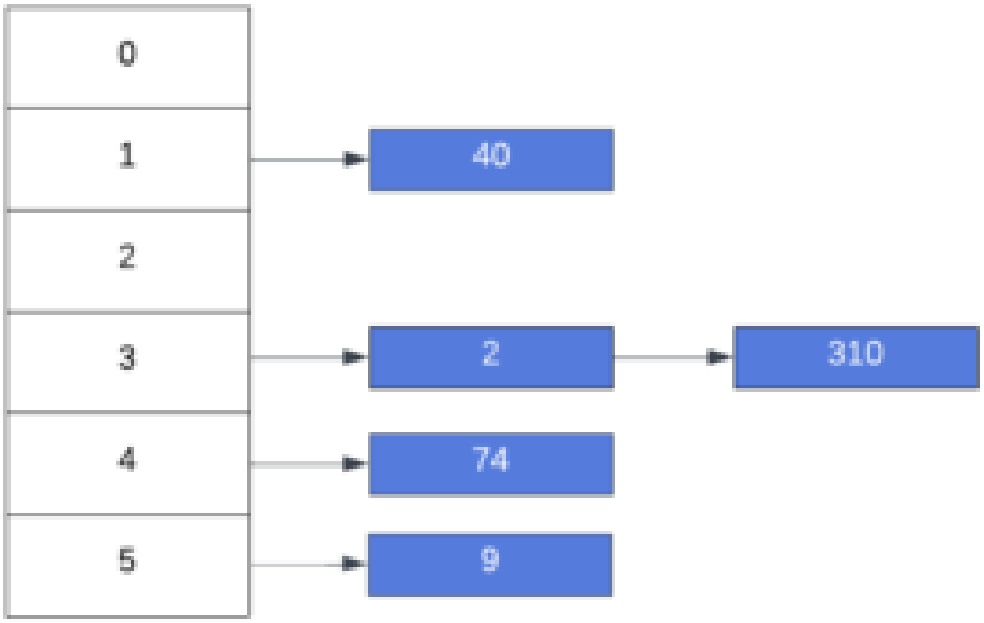


According to our function, , 9 will be inserted at index 5, whereas the last item, 2 will be inserted at index 3:





Once we try to insert 2, we encounter a collision with key *310*. However, because we're using separate chaining as our collision resolution algorithm, our hash table results in the following state after inserting all elements in :



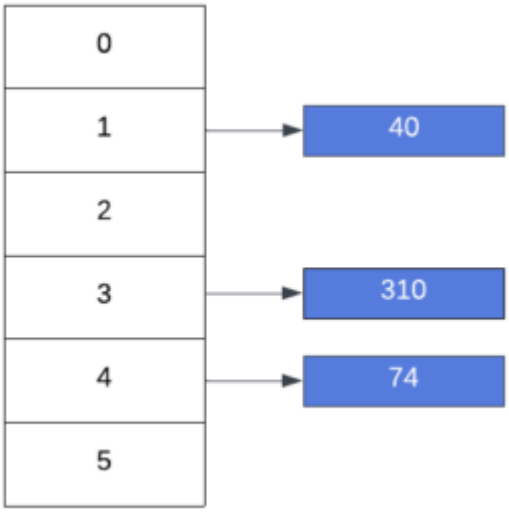
Let’s dive into the separate chaining algorithm and learn how we obtained the previous state in our hash table.

3. Algorithm

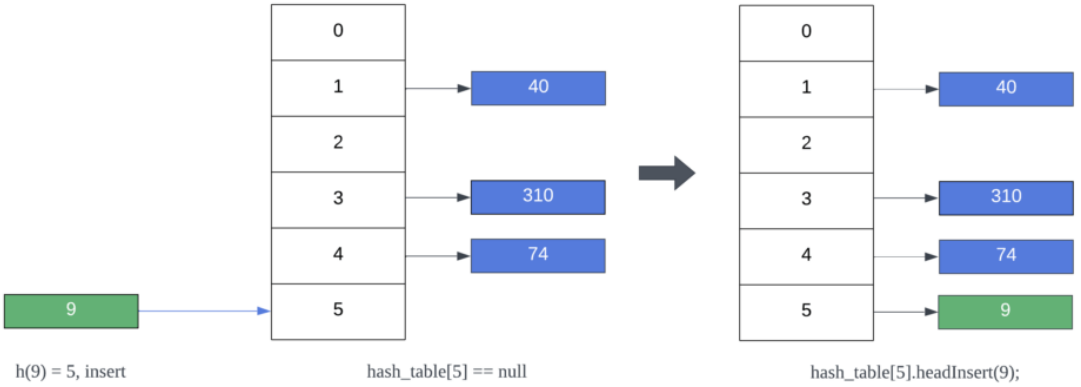
To use the separate chaining technique, we represent hash tables as a list of linked lists. In other words, every index at the hash table is a hash value containing a chain of elements.

Once a collision happens, inserting a key into a list at a particular index could be costly. That is, **if we disallow duplicate keys, we must search throughout the entire list first**. Yet, **if we allow duplicate elements, we optimize the process by inserting them at the head of the list**. On the other hand, searching for an element involves traversing through the entire list until the key is found.

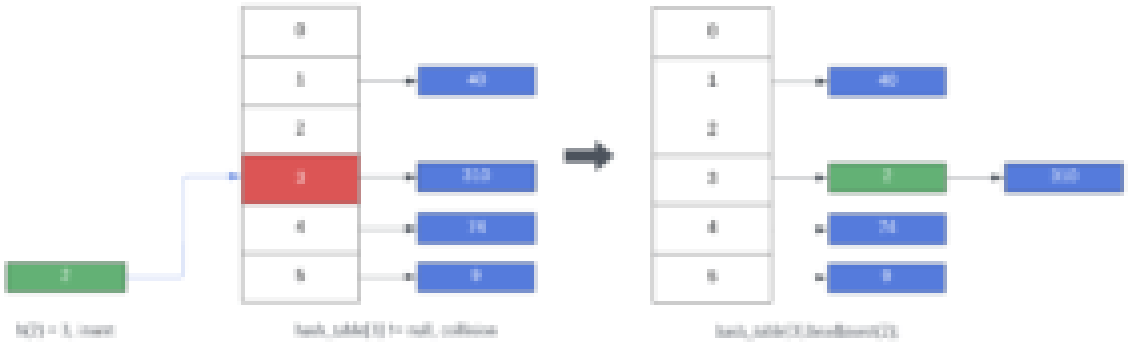
Let’s walk through the algorithm using as the input set and a hash table with the following initial state:



Currently, there are no elements at index 5. Therefore, inserting 9 involves creating a new linked list with a single element. Subsequently, new keys mapping to the same index will be added to the list:

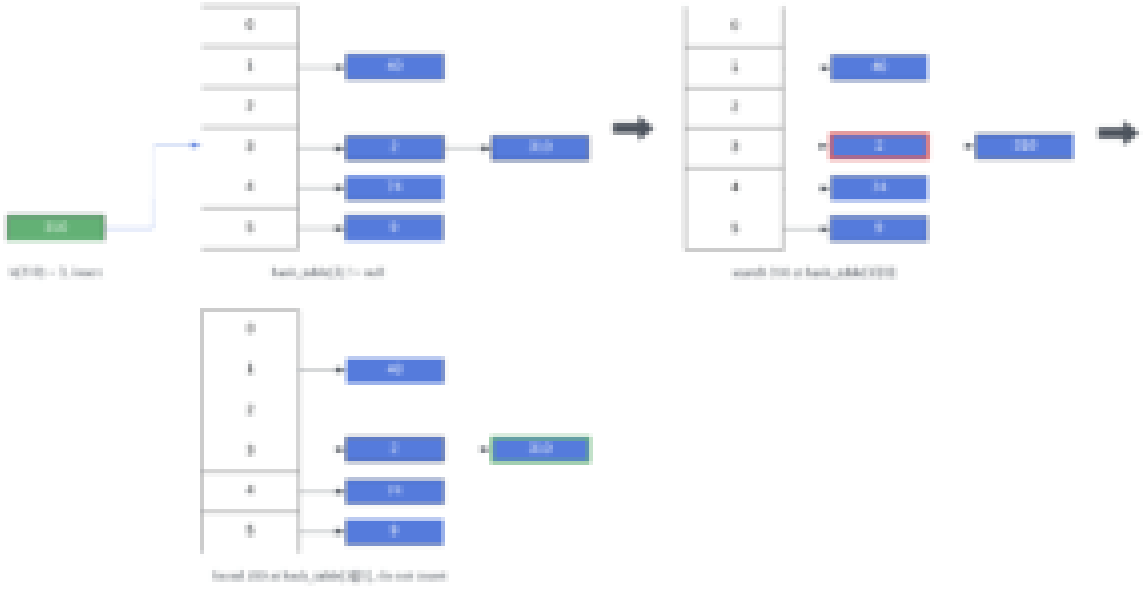


Next, inserting 2 at index 3 results in a collision. In other words, there is an existing list at index 3 storing other values. In this case, **separate chaining simply inserts the key at the beginning of the list.** As a result, the list at index 5 now contains 2 keys and each element still maps to its original hash index:



Note that we never checked for duplicate elements in the previous insert operations. Both previously inserted keys could have been in the hash table already. Nonetheless, **to disallow duplicate elements, we must search for the key before inserting it.**

Let’s look at an example where we try to insert 40, and we disallow duplicate keys in our hash table. For simplicity’s sake, we’ll use the final state of our previous example:



In our previous example, we searched and tried to insert an existing element into our hash table. As a result, no insert operation was performed.

Overall, in separate chaining, keys will use the same index they originally generated from the hash function. This advantage allows searching for a key more performant if we have a good hash function that distributes the elements fairly across the has table.

4. Pseudocode

Let’s look at the pseudocode for separate chaining For simplicity’s sake, we’ll use two different functions to determine whether a key can be inserted or found in the hash table. Let’s start with the insert operation.

4.1. Insert

The *insert* implementation is a void function that adds the key to their corresponding list. The following function creates a new linked list if one does not exist at the specified index.

However, if the list already exists, we simply add the new element: