Separate Chaining

4 min

A hash map with a linked list separate chaining strategy follows a similar flow to the hash maps that have been described so far. The user wants to assign a value to a key in the map. The hash map takes the key and transforms it into a hash code. The hash code is then converted into an index to an array using the modulus operation. If the value of the array at the hash function's returned index is empty, a new linked list is created with the value as the first element of the linked list. If a linked list already exists at the address, append the value to the linked list given.

This is effective for hash functions that are particularly good at giving unique indices, so the linked lists never get very long. But in the worst-case scenario, where the hash function gives all keys the same index, lookup performance is only as good as it would be on a linked list. Hash maps are frequently employed because looking up a value (for a given key) is quick. Looking up a value in a linked list is much slower than a perfect, collision-free hash map of the same size. A hash map that uses separate chaining with linked lists but experiences frequent collisions loses one of its most essential features.

Instructions

In the application, we've implemented an array with separate chaining. Notice when two keys hash to the same value they get added to the chained linked list.

Separate chaining doesn't require linked lists to be the underlying data structure for each array index. What would a separate chaining algorithm with a different data structure look like? What would the benefits of using a tree at each index? What would be the drawbacks?

What if at every index we had another hash table?