At this lab section, we will learn how to implement hash tables in Java using OOP principles.

Hash Tables

Separate Chaining

Asst. Prof. Dr. Feriştah DALKILIÇ
Res. Asst. Fatih DİCLE

PART 1 – Separate Chaining

Separate Chaining approach to collision resolution alters the structure of the hash table so that each location can represent more than one value. Such a location is called a bucket. Anytime a new search key maps into a particular location, you simply place the key and its associated value in the bucket, much as we did with open addressing. To find a value, you hash the search key, locate the bucket, and look through the key-value pairs in it. In all likelihood, the bucket contains few values, so this mini search will be fast. When you remove an entry, you find it in its bucket and delete it. Thus, the entry no longer exists in the hash table.

What can you use to represent a bucket? A list, a sorted list, a chain of linked nodes, an array, or a vector are some possibilities with which you are familiar. Anything that involves an array or vector will cause a substantial memory overhead, since each location in the hash table will have a fixed amount of memory allocated to it. Much of this memory will be unused. Either a linked implementation of a list or a chain of linked nodes is a reasonable choice for a bucket, since memory is allocated to the bucket only as needed.

A hash table for use with separate chaining; each bucket is a chain of linked nodes

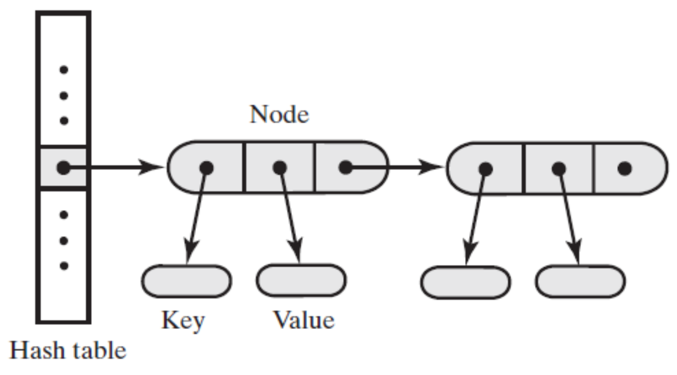

Figure 1. A hash table with linked chains

Figure illustrates a hash table with linked chains as buckets. In this arrangement, each location in the hash table is a head reference to a chain of linked nodes that make up the bucket. Each node contains references to a search key, to the key's associated value, and to the next node in the chain. Notice that a node must reference the search key so that you can locate it later when you search the chain. Resolving collisions by using buckets that are linked chains is called separate chaining.

Exercise – 1

In this exercise, you are expected to experiment with the Separate Chaining approach to collision resolution.

Step – 1

Create a new Java Project. Add the DictionaryInterface.java and HashedDictionary.java classes given to you in *src* folder.

Step – 2

Fill in the blanks in add and `getValue` methods of HashedDictionary.java according to the scheme given in Figure 1.

Your Code

Step – 3

Add a new class with the name of "Test.java". Create an instance of HashedDictionary and add the given *contact_name – phone_number* pairs into the dictionary.

contact_name	phone_number
"Dirk"	"555-1234"
"Abel"	"555-5678"
"Miguel"	"555-9012"
"Tabbie"	"555-3456"
"Tom"	"555-5555"
"Sam"	"555-7890"
"Reiss"	"555-2345"
"Bette"	"555-7891"
"Carole"	"555-7892"
"Derek"	"555-7893"
"Nancy"	"555-7894"

Step – 4

Add the following method into "Test.java" and display the current content of the dictionary.

```
public static void display(DictionaryInterface<String, String> dictionary)
{
    Iterator<String> keyIterator   = dictionary.getKeyIterator();
    Iterator<String> valueIterator = dictionary.getValueIterator();

    while (keyIterator.hasNext() && valueIterator.hasNext())
        System.out.println(keyIterator.next() + " : " + valueIterator.next());
    System.out.println();
} // end display
```

Step – 5

In Test.java, perform the operations given below:

- Display the phone book.
- Show the contact count in your phone book.

- Retrieve the Sam’s phone number.
- Query whether Bo in your contact list.
- Update the Miguel’s phone number as "555-9015".
- Remove Reiss from your contacts.
- Display your current phone book.
- Delete your all contacts.

Your `Test.java`

Your Output