CECS 277 – Lecture 9 – Hash Tables

Hashing is used to find and store elements to a data structure quickly. It is comprised of two parts: the Hash Table, which is where the elements are stored, and a Hash Function, which computes the location of where in the Hash Table an element will be stored, based on the element's value.

This Hash Function hashCode() is overridden from the Object class. Several other commonly used classes already override it so that you do not have to: String, Integer, Point, Rectangle, Color, along with all of the collection classes, such as: ArrayList, LinkedList, HashSet and TreeSet. If, however, you need to use your own class, you can easily override the hashCode method to store your own objects in a Hash Table.

Using a hashCode() Method – A hashCode method returns an integer location given the value of the element. It must return the same location for that same value, every single time the method is run.

```
String s = "Hello";
int hashVal = s.hashCode();
```

The value of hashVal could return a negative integer, so it may need to be modified to use as a location. It will also need to be modulated to make sure that it can be placed in a storage location that you created, (ex. An array of 20 elements).

```
if(hashVal<0){
     hashVal *= -1;
}
hashVal %= array.length;
```

Once the hash value has been modulated, it becomes more likely that two different values may hash to the same location. These are called collisions, and they need to be handled otherwise data may be lost. By storing the elements into an array of LinkedLists, we can deal with the collision by storing multiple elements in each location, but it is still better to have a hashCode() method that avoids any collisions. To also help avoid collisions you should allocate more space for the array than is needed. About 30% more works well, and using a prime number of spaces helps.

Creating a hashCode() Method – You may need to create a hashCode() method for one of your own classes. When faced with this, you should examine the data for your object. Figure out a way to use the object's data and combine it to create a single (and hopefully unique) value.

Example: Hash code for String.

```
final int HASH_MUL = 31;
int h = 0;
for(int i = 0; i < s.length; i++){
     h = HASH_MUL * h + s.charAt(i);
}
return h;
```

Example: A possible hashCode() method for a class Student.

```
public class Student{
     private String name;
     private char grade;

     public Student(String n, char g){
          name = n;
          grade = g;
     }
     public int hashCode(){
          final int HASH = 13; //<-prime #
          int strHash = name.hashCode();
          int charHash = HASH * (int)grade;
          int h = (strHash + charHash) * HASH;
          return h;
     }
     ...
}
public class TestStudent{
     public static void main(String[]args){
          Student [] students = new Student[41];
          Student s = new Student("Jenny", 'B');
          int h = s.hashCode();
          h %= students.length;
          students[h] = s;
     }
}
```