Dean Gladish	Data Structures	Professor Eric Alexander	HW 9

HashMaps.

1. Suppose you are writing a $\boxed{\text{HashMap}}$ class that uses open addressing to resolve collisions. You are going to create a $\boxed{\text{TableEntry}}$ class that will be stored at each position in the array. What instance variables should you have in the $\boxed{\text{TableEntry}}$ class and why? How will those instance variables allow you to remove entries from the table?

In order to map keys to values while avoiding duplicate values, we must create instance variables, String key and another variable of type Object where Object is determined by you.

Aside from these we might want instance methods to retrieve and modify these variables.

$\boxed{\text{HashMap}}$ allows key:null pairs, so we must traverse the array. Every time we find the desired value, we should store its key by calling an instance method of $\boxed{\text{TableEntry}}$. We should then be able to use the remove method of ArrayList to get rid of all of these key:value pairs. The use of this removal method has a much better runtime; open addressing is preferable to chaining in this regard. However, probing takes time.

2. Suppose you are using quadratic probing to resolve collisions. Imagine you have an array of length 5 and the first spot you try to add a particular item is 0. If the array isn't full, could you ever fail to find a spot to add the item? Hint: Try generating the sequencing of spots that quadratic probing will look at.

We are using quadratic probing. Say the hash function returns index 1 for the item and the array looks like this:

index:	array
0	value
1	value
2	value
3	
4	

array[1] is full so we begin quadratic probing: We look at indices $(1 + \text{iteration}^2) \% 5$ b/c $5 = \text{TABLE\_SIZE}$.

Say $a \in \mathbb{N}^+$. Then a can be written as $10b + c$ where $b, c \in \mathbb{N}$, so $a^2 = (10b+c)^2 = (10b)^2 + 2(10bc) + c^2 = 100(b^2) + 10(2bc) + c^2$, so $a \% 5$ is $100(b^2) \% 5 + 10(2bc) \% 5 + c^2 \% 5$.

2 continued. So, $a^2\%5 = c^2\%5$ where c is the ones digit of a. This shows that only the ones digit is the sole determinant of the key that the quadratic probing function will calculate. The following are iterators:

iterator i	1	2	3	4	5	6	7	8	9	10	11	12	13
last digit of i^2	2	5	10	7	6	7	10	5	2	1	2	5	10
previous % 5	2	0	0	2	1	2	0	0	2	1	2	0	0

repeating

The final column, which repeats as the value of i increments by 1, shows that we fail to find a spot for the item even though the array is not full. The answer is yes. The following Java code shows that fact up to 9999 increments:

```
ArrayList<String> newList = new ArrayList<String>();
for (int i=0; i < 9999; i++) {
    newList.add(""+ (((1+(i*i)) % 5)));
}
for (int j=0; j<=4; j++) {
    System.out.println(newList.contains(""+j));
}
System.out.println(newList);
```

Here, we never visit the last two spots.

3. Suppose you have keys that are Characters, a hashCode() function where a=1, b=2, ..., z=26$, and you are using the standard modulus operation ($\boxed{\%}$) as a compression function. Your hash table begins as an array of size 7 and you are using linear probing to handle collisions. Draw the array after you perform add on each of the following key-value pairs:

(key → value): (a → 4), (b → 2), (n → 14), (g → 9).

index	0	1	2	3	4	5	6
array	14	4	2	9			

① ② ③

9

hashCode(n) = 14 % 7 → 0

hashCode(g) = 7 % 7 → 0

linear probing (iteration) → collision,

4. Now I call $\boxed{\text{hashMap.insert('g', 10)}}$ on the map you created in question (3). What entries do I look through in the array? What will the array look like after I make this call?

index	0	1	2	3	4	5	6
array	14	4	2	9	10		
	0	1	2	3	4	Iterations.	

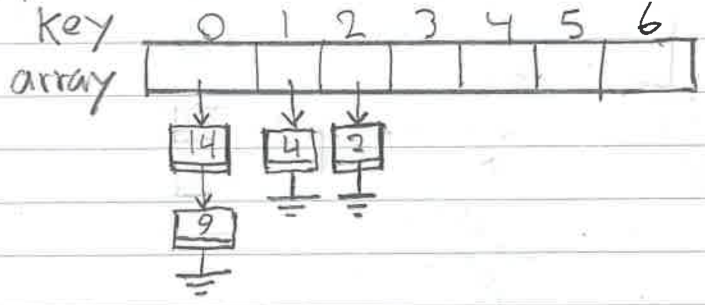
We are using linear probing. We start at hashCode(g) = 7 % 7 → 0. (modulo 7 b/c 7 = TABLE_SIZE)

On Iteration #4 we find an empty spot.

5. After finishing question 4, what is the load factor (lambda) for the array?

$\lambda = \frac{\#\text{keys}}{\text{table size}} = \boxed{\frac{5}{7}}$; the array should be expanded b/c $\frac{5}{7} \not< .5$ and we are using linear probing guidelines.

6. Now imagine that you're using chaining (with a linked list) to resolve collisions. Repeat question (3) with this new collision scheme, drawing the array after all of the key-value pairs have been added.

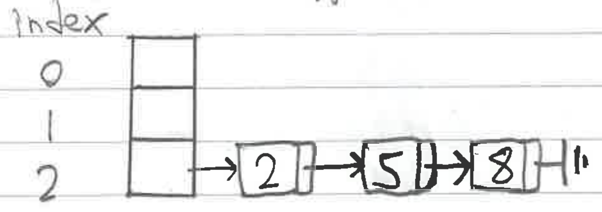


hashcode(n) = 14 % 7 → 0

hashCode(g) = 7 % 7 → 0.

7. Let's say you're using chaining for collision resolution, and you have a spot in the array with a linked list containing 3 items. If you resize the array, would it be correct to compute the new spot for the first item in the list and then copy the linked list to that spot in the new array? Make sure you understand why or why not.

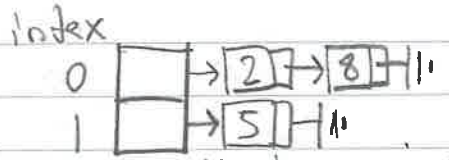
7 continued. Suppose there is an array



ARRAY_SIZE = 3
$2\%3 = 5\%3 = 8\%3 = 2$.
We are using $\boxed{\%}$ as a compression function, so we insert our values 2, 5, and 8 into index 2 of the array. We let our hashCode() function be hashCode(n) = n.

Now, resize to size 2.

Now, $2\%2 = 8\%2 = 0$,
$5\%2 = 1$.



This counterexample shows that such a method is not correct because it has not kept the relationship between values.

This occurs because $n\%m = a\%m \not\Rightarrow n\%(m-1) = a\%(m-1)$ for all n, m, a in $\mathbb{Z}^+$.

As a result, objects in the same linked-list might get separated.

8. Suppose I have a $\boxed{\text{Cat}}$ class that has instance variables for the animal's name, breed, and year of birth. Propose a way to implement $\boxed{\text{hashCode()}}$ for Cats such that two Cats that are equal (have the same values for all instance variables) will return the same hash code and Cats that are not equal will tend not to return the same hash code (i.e., just returning the same code for all Cats is not a correct answer). You may use any methods you want in your description.

Say we look up the ASCII value of each character in the $\boxed{\text{String}}$ name and add each one: sum $= 10^{n-1}\text{char}_0 + 10^{n-2}\text{char}_1 + \cdots + 10^0\text{char}_{n-1}$, n is length(name) and char_k is name.charAt(k). Do the same process for $\boxed{\text{String}}$ breed. Add these + year = result, and do result % TABLE_SIZE = key. Store the original $\boxed{\text{Cat}}$ object at this key.