Due to the print book page limit, we cannot inlcude all good CheckPoint questions in the physical book. The CheckPoint on this Website may contain extra questions not printed in the book. The questions in some sections may have been reordered as a result. Nevertheless, it is easy to find the CheckPoint questions in the book on this Website. Please send suggestions and errata to Dr. Liang at y.daniel.liang@gmail.com. Indicate the book, edition, and question number in your email. Thanks!

# Chapter 27 Check Point Questions

## Section 27.2

**▼27.2.1**

What is a hash function? What is a perfect hash function? What is a collision?

If you know the index of an element in the array, you can retrieve the element using the index in O(1) time. So, can we store the values in an array and use the key as the index to find the value? The answer is yes if you can map a key to an index. The array that stores the values is called a hash table. The function that maps a key to an index in the hash table is called a hash function.

How do you design a hash function that produces an index from a key? Ideally, we would like to design a function that maps each search key to a different index in the hash table. Such a function is called a perfect hash function. However, it is difficult to find a perfect hash function. When two or more keys are mapped to the same hash value, we say that a collision has occurred.

Hide Answer

## Section 27.3

**▼27.3.1**

What is a hash code? What is the hash code for Byte, Short, Integer, and Character?

A typical hash function first converts a search key to an integer value called a hash code, and then compresses the hash code into an index to the hash table.

For a search key of the type byte, short, int, and char, simply cast it to int. So two different search keys of any one of these types will have different hash codes.

Hide Answer

**▼27.3.2**

How is the hash code for a Float object computed?

For a search key of the type float, use Float.floatToIntBits(key) as the hash code. Note that floatToIntBits(float f) returns an int value whose bit representation is the same as the bit representation for the floating number f. So, two different search keys of the float type will have different hash codes.

Hide Answer

**▼27.3.3**

How is the hash code for a Long object computed?

For a search key of the type long, simply casting it to int would not be a good choice, because all keys that differ in only the first 32 bits will have the hash code. To take the first 32 bits into consideration, divide the 64 bits into two halves and perform the exclusive or operator to combine the two halves. This process is called folding. So, the hashing code is

```
int hashCode = (int)(key ^ (key >> 32));
```

Note that >> is the right-shift operator that shifts the bits 32 position to the right.

Hide Answer

**27.3.4**

How is the hash code for a Double object computed?

For a search key of the type double, first convert it to a long value using doubleToLongBits, then perform a folding as follows:

```
long bits = Double.doubleToLongBits(key);
int hashCode = (int)(bits ^ (bits >> 32));
```

Hide Answer

**27.3.5**

How is the hash code for a String object computed?

The hashcode for a string in Java is

$(...((s_0*b + s_1)b + s_2)b+...+s_{n-2})b+ s_{n-1}$

where b is 2.

Hide Answer

**27.3.6**

How is a hash code compressed to an integer representing the index in a hash table?

The hash code for a key can be a large integer that is out of the range for the hash table index. You need to scale it down to fit in the range of the index. Assume the index for a hash table is between 0 and N-1. The most common way to scale an integer to between 0 and N-1 is to use

```
h(hashCode) = hashCode % N
```

To ensure that the indices are spread evenly, choose N to be a prime number greater than 2.

Hide Answer

**27.3.7**

If N is a value of the power of 2, is N / 2 same as N >> 1?

Yes.

Hide Answer

**27.3.8**

If N is a value of the power of 2, is m % N same as m & (N - 1) for any integer m?

Yes.Open addressing is to find an open location in the hash table in

Hide Answer

**27.3.9**

What is new Integer("-98").hashCode() and what is "ABCDEFGHIJK".hashCode()?

new Integer("-98").hashCode() is -98 and"ABCDEFGHIJK".hashCode() is -331017146.

Hide Answer

## Section 27.4

**27.4.1**

What is open addressing? What is linear probing? What is quadratic probing? What is double hashing?

Open addressing is to find an open location in the hash table in the event of collision. Open addressing has several variations: linear probing, quadratic probing, and double hashing. When a collision occurs during the insertion of an entry to a hash table, linear probing finds the next available location sequentially.

Quadratic probing can avoid the clustering problem in linear probing. Linear probing looks at the consecutive cells beginning at index k. Quadratic probing, on the other hand, looks at the cells at indices $(k + j^2) \% n$, for $j \geq 0$, i.e., k, (k + 1) % n, (k + 4) % n, (k + 9) % n, ..., and so on.

Another open addressing scheme that avoids the clustering problem is known as double hashing. Starting from the initial index k, both linear probing and quadratic probing add an increment to k to define a search sequence. The increment is 1 for linear probing and j2 for quadratic probing. These increments are independent of the keys. Double hashing uses a secondary hash function on the keys to determine the increments to avoid the clustering problem.

Hide Answer

**27.4.2**

Describe the clustering problem for linear probing.

Linear probing tends to cause groups of consecutive cells in the hash table to be occupied. Each group is called a cluster. Each cluster is actually a probe sequence that you must search when retrieving, adding, or removing an entry. As clusters grow in size, they may merge into even larger clusters, further slowing down the search time. This is a big disadvantage of linear probing.

Hide Answer

**27.4.3**

What is secondary clustering?

Quadratic probing works in the same way as linear probing except for the change of search sequence. Quadratic probing avoids the clustering problem in linear probing, but it has its own clustering problem, called secondary clustering, i.e., the entries that collide with an occupied entry use the same probe sequence.

Hide Answer

**27.4.4**

Show the hash table of size 11 after inserting entries with keys 34, 29, 53, 44, 120, 39, 45, and 40, using linear probing.

Use the animation https://liveexample.pearsoncmg.com/dsanimation/LinearProbingeBook.html to verify your anwser. To start, click the Remove All button to empty the hash table and set the load factor threshhold to 0.99.

Hide Answer

**27.4.5**

Show the hash table of size 11 after inserting entries with keys 34, 29, 53, 44, 120, 39, 45, and 40, using quadratic probing.

Use the animation https://liveexample.pearsoncmg.com/dsanimation/QuadraticProbingeBook.html to verify your anwser. To start, click the Remove All button to empty the hash table and set the load factor threshhold to 0.99.

Hide Answer

**27.4.6**

Show the hash table of size 11 after inserting entries with keys 34, 29, 53, 44, 120, 39, 45, and 40, using double hashing with the following functions:

```
h(k) = k % 11;
h'(k) = 7 - k % 7;
```

See the text.

Hide Answer

## Section 27.5

**27.5.1**

Show the hash table of size 11 after inserting entries with the keys 34, 29, 53, 44, 120, 39, 45, and 40, using separate chaining.

Use the animation https://liveexample.pearsoncmg.com/dsanimation/SeparateChainingeBook.html to verify your anwser. To start, click the Remove All button to empty the hash table and set the load factor threshhold to 0.99.

Hide Answer

## Section 27.6

**27.6.1**

What is load factor? Assume the hash table has the initial size 4 and its load factor is 0.5; show the hash table after inserting entries with the keys 34, 29, 53, 44, 120, 39, 45, and 40, using linear probing.

Load factor is the ratio between the number of elements in the hash table and the hash table size. It measures how full in the hash table. The other part is omitted. Use the animation https://liveexample.pearsoncmg.com/dsanimation/LinearProbingeBook.html to verify your anwser. To start, click the Remove All button to empty the hash table and set the load factor threshhold to 0.5.

Hide Answer

**27.6.2**
Assume the hash table has the initial size 4 and its load factor is 0.5; show the hash table after inserting entries with the keys 34, 29, 53, 44, 120, 39, 45, and 40, using quadratic probing.

See the text

Hide Answer

**27.6.3**
Assume the hash table has the initial size 4 and its load factor is 0.5; show the hash table after inserting entries with the keys 34, 29, 53, 44, 120, 39, 45, and 40, using separate chaining.

See the text

Hide Answer

## Section 27.7

**27.7.1**
What is 1 << 30 in line 8 in Listing 27.2? What are the integers resulted from 1 << 1, 1 << 2, and 1 << 3?

$2^{30}$
$2^{1}$
$2^{2}$
$2^{3}$

Hide Answer

**27.7.2**
What are the integers resulted from 32 >> 1, 32 >> 2, 32 >> 3, and 32 >> 4?

32 >> 1 is 16
32 >> 2 is 8
32 >> 3 is 4
32 >> 4 is 2

Hide Answer

**27.7.3**
In Listing 27.2, will the program work if LinkedList is replaced by ArrayList? In Listing 27.2, how do you replace the code in lines 56-59 using one line of code?

Yes.
The second part of this question: return get(key) != null

Hide Answer

**27.7.4**
Describe how the put(key, value) method is implemented in the MyHashMap class.

See the text.

Hide Answer

**27.7.5**

In Listing 27.2, the supplementalHash method is declared static, can the hash method be declared static?

No. The reason: The hash method implementation uses capacity. capacity is an instance data field in MyHashMap.

Hide Answer

**27.7.6**

Show the output of the following code.

```
MyMap<String, String> map = new MyHashMap<>();
map.put("Texas", "Dallas");
map.put("Oklahoma", "Norman");
map.put("Texas", "Austin");
map.put("Oklahoma", "Tulsa");
System.out.println(map.get("Texas"));
System.out.println(map.size());
```

Austin
2

Hide Answer

**27.7.7**

If x is a negative int value, will x & (capacity - 1) be negative?

capacity is a 32-bit int value. Since the maximum capacity for the hash table is $2^{30}$ (line 8 in MyHashMap.java), the first bit in capacity is 0. Thus, the first bit in the result of x & (capacity - 1) is 0. Therefore, x & (capacity - 1) is positive, even though x is negative.

Hide Answer

## Section 27.8

**27.8.1**

Why can you use a foreach loop to traverse the elements in a set?

Because MySet implements Iterable.

Hide Answer

**27.8.2**

Describe how the add(e) method is implemented in the MyHashSet class.

See the text

Hide Answer

**27.8.3**

Can lines 100-103 in Listing 27.4 be removed?

Yes. It is redundant, because contains(e) is true after lines 94-95 and table[bucketIndex] is not null.

Hide Answer

**27.8.4**

Implement the remove() method in lines 150-152?

```
@Override /** Remove the element returned by the last next() */
public void remove() {
    if (current == 0) // next() has not been called yet
    throw new IllegalStateException();
    set.remove(list.get(--current));
  list.remove(current); // Remove the element from the list
}
```

Hide Answer