Lab 9

HashMaps

CS61B Fall 2023

Announcements

Project 2B Checkpoint due Monday, 10/23 at 11:59 pm.

Project 2B is due Monday, 10/30 at 11:59 pm.

Throwback: Arrays and Maps

Arrays

Question: How fast does it take for an array to lookup an item, given an index?

Arrays

Question: How fast does it take for an array to lookup an item, given an index?

- **O(1)**! (i.e. constant time)

Arrays

Question: How fast does it take for an array to lookup an item, given an index?

- **O(1)**! (i.e. constant time)

It's pretty fast! We should try to take advantage of this characteristic to create another data structure.

Maps

Let's talk **`Maps`** for a minute. Remember, **`Maps`** in Java represent a mapping between a key and value pairing (<key, value>)

We're going to try combining mapping with the constant lookup time of arrays!

Maps

Let's talk **`Maps`** for a minute. Remember, **`Maps`** in Java represent a mapping between a key and value pairing (<key, value>)

We're going to try combining mapping with the constant lookup time of arrays!

So, how do we insert these pairings into an array?

Hash Functions

Hash Functions

What are hash functions?

- They take in a key as their input and returns a hash code!

Hash Functions

What are hash functions?

- They take in a key as their input and returns a hash code!

Hash codes are of type `int`, which means they can take on any value of -2,147,483,648 to 2,147,483,647.

Do we just make… 4,294,967,296 slots in our underlying array for each <key, value> pairing?

Modulo

So here's our solution: %!

- If we use the % operator, we can reduce our hashcode to a value between 0 and N, where N is the number of buckets we have (or the # of array slots).
- Be careful of negative hash codes when using % (or use `Math.floorMod`)

Possible Problem?

However, if we know that our hash code is technically within a certain range, what happens if two keys end up with the same hash code?

This brings us into our next topic: **hash collisions**!

Hash Collisions

Hash Collisions

If two keys end up with the same hash code, this is called a **hash collision** (and they will end up mapping to the same bucket). To deal with them, at least for this lab, we'll use external chaining.

Hash Collisions

If two keys end up with the same hash code, this is called a **hash collision** (and they will end up mapping to the same bucket). To deal with them, at least for this lab, we'll use external chaining.

External Chaining: Store all keys with the same hash code in a collection of their own, such as a linked list (i.e. each bucket will have a collection, and if a key maps to that bucket, we store it in the corresponding collection).

Duplicates

What about duplicate keys? Assuming we're using the same hash function, the key will be mapped to the same bucket (deterministic), so how do we check if a key is already in the hashmap?

Duplicates

What about duplicate keys? Assuming we're using the same hash function, the key will be mapped to the same bucket (deterministic), so how do we check if a key is already in the hashmap?

- Consider using the **equals method** to check if a key is already in the hashmap, specifically, in the bucket.
- Remember, we can't have any duplicate keys in hashmaps, so we need to ensure that it doesn't happen.

Collisions

What happens if we have too many collisions?

- Think about what happens if **every element gets put in the first bucket.** What happens to the runtime of adding or finding an element?

Collisions

What happens if we have too many collisions?

- Think about what happens if **every element gets put in the first bucket.** What happens to the runtime of adding or finding an element?

The less collisions (aka less elements in a single bucket), the better runtime for all HashMap operations!

Back to Hash Functions

So if minimizing collisions is our goal, how does that relate to the hash function that we use, and how do we pick a **good hash function**?

Back to Hash Functions

So if minimizing collisions is our goal, how does that relate to the hash function that we use, and how do we pick a **good hash function**?

- A good hash function is one that minimizes collisions!
- This means it maps all inputs uniformly over the output range, i.e. it should put approximately the same number of elements in each bucket.

Resizing

Resizing

What happens if we have too many elements in our HashMap, i.e. too many elements in all buckets, total?

- We can try to reduce the number of items per bucket by expanding how many buckets we have total!
- To avoid long lookup times per bucket, we will **resize** our bucket array.

Resizing

At some point, when there are too many items in the hashmap, we want to resize our hashmap. Specifically, when our load factor surpasses a certain limit.

Resizing

At some point, when there are too many items in the hashmap, we want to resize our hashmap. Specifically, when our load factor surpasses a certain limit.

$$\textbf{Load Factor} = \frac{\text{Number of Items}}{\text{Number of Buckets}}$$

Rehashing

We resize our HashMap by increasing the number of buckets and **rehashing** all the elements?

- If we don't rehash, we might mistakenly "lose" an element!
- Consider a HashMap with **4 buckets** and an element that hashes to **7**. What would happen if we didn't rehash when we resize to **8 buckets**?

Thus, with a good resizing mechanism and a good hash function, all Hashmap operations run in amortized **O(1)** runtime!

HashMaps

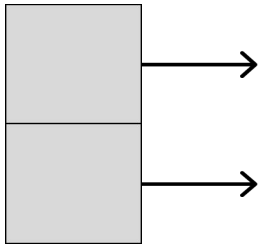
HashMaps

Bringing this all together, we have our data structure: hashmaps!

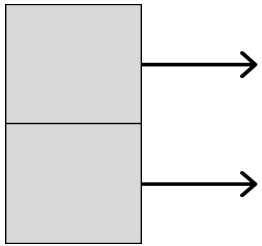
HashMaps

Bringing this all together, we have our data structure: hashmaps!

- We have an array-like structure where each **bucket** has some **<key, value>** pairings stored in it.
- Each of the pairings are put based on their **hash code** or the output of the **hash function**.
- Each bucket ultimately stores a linked list, so we can save multiple entries into a single bucket (**external chaining**)



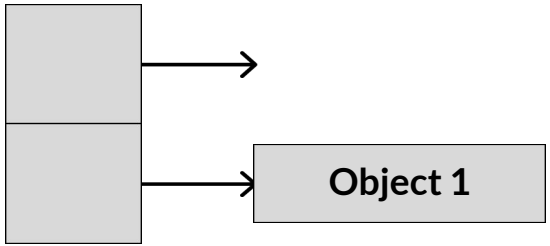
Let's walk through an example! For now, our hashmap starts out with 2 buckets and the load factor limit is set to 0.75. Our hash function will return a randomized integer. **Note: We are only concerned with mapping here, so the value is omitted.**



hashFunction() → returns a random integer

Object 1

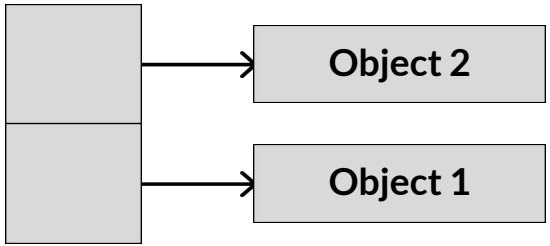
Let's pass our first object into the hash function, Object1, and it returns an **integer of 27.** Which bucket does it go into?



hashFunction() → returns a random integer

Object 2

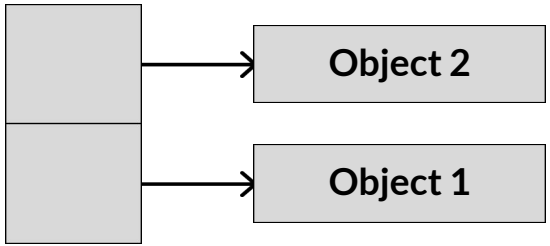
Using the modulus operator, it goes into the bucket corresponding to **index of 1**. How about for Object2, assuming it's **hash code is 22444**?



hashFunction() → returns a random integer

Reminder: our limit for this hash map is a load factor of 0.75 (we check for when it exceeds the load factor)

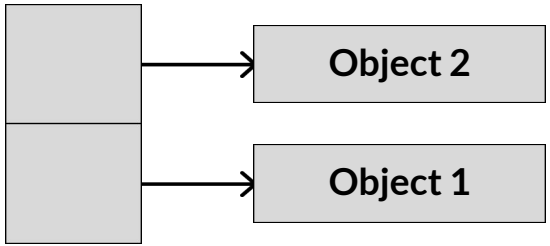
It'll go into the bucket corresponding to index of 0! What do we have to do now?



hashFunction() → returns a random integer

Our current load factor is 1.
Our limit is 0.75.

We need to increase the number of buckets in our hashmap! This means that we have to rehash all the objects that are currently in the hashmap.

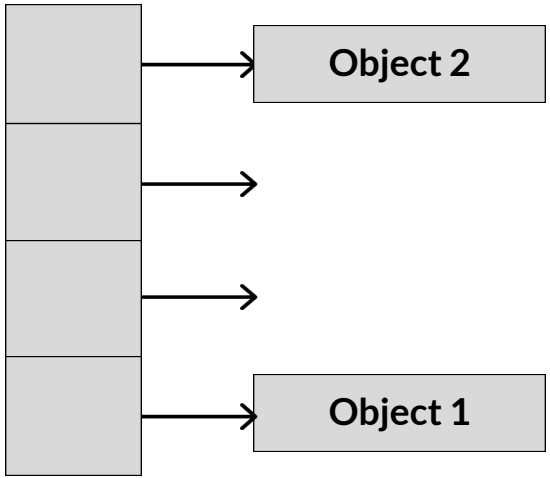


Hash Codes:

- Object 1: **27**
- Object 2: **22444**

Our current load factor is 1.
Our limit is 0.75.

Assuming that our resize factor is 2, what will our hashmap look like? The hash codes of the current objects are provided above.



New Hash Code:

- Object 1: 27 % 4 = **3**
- Object 2: 22444 % 4 = **0**

It'll look something like this! We **rehashed** all our objects based on their original hash codes and the new number of buckets in our hashmap.

Additional Notes

Keep in mind that the **key of our <key, value> pairing** is what is passed into the hash function.

In addition, make sure that you maintain the association of your key with your value (i.e. that the entirety of the <key, value> pairing is placed into the correct bucket).

If a duplicate key is placed into the hashmap, then the old value is replaced with the new value.

Lab Overview

An Overview

Lab 09 is due Friday, 10/27 at 11:59 pm.

Deliverables:

- Complete your implementation of **`HashMap`** and ensure that it implements the interface **`Map61B`**.
- Make sure to fill out **`results.txt`**!

Some tips:

- Take advantage of the provided helper methods and make some of your own!
- **Make sure to read the spec carefully for requirements.**

For help, use the Lab queue: [INSERT YOUR LAB QUEUE HERE]