Week 8 - Friday

COMP 2100

Last time

- What did we talk about last time?
- Open addressing
  - Linear probing
  - Quadratic probing
  - Double hashing
- Chaining
- Started (chaining) hash table implementation

Questions?

Project 3

Assignment 4

Hash Tables

Hash Table Implementation

Recall: Symbol table ADT

- We can define a symbol table ADT with a few essential operations:
  - put(Key key, Value value)
    - Put the key-value pair into the table
  - get(Key key):
    - Retrieve the value associated with key
  - delete(Key key)
    - Remove the value associated with key
  - contains(Key key)
    - See if the table contains a key
  - isEmpty()
  - size()
- It's also useful to be able to iterate over all keys

Chaining hash table

```
public class HashTable {
    private int size = 0;
    private int power = 10;
    private Node[] table = new Node[1 << power];

    private static class Node {
        public int key;
        public Object value;
        public Node next;
    }
    …
}
```

Hashing function

- It's useful to have a function that finds the appropriate hash value
- Take the input integer and swap the low order 16 bits and the high order 16 bits (in case the number is small)
- Square the number
- Use shifting to get the middle `power` bits

```
private int hash(int key)
```

Contains (chaining)

- If the hash table contains the given key, return **`true`**
- Otherwise return **`false`**

```
public boolean contains(int key)
```

Get (chaining)

- Return the object with the given key
- If none found, return **`null`**

```
public Object get(int key)
```

Put (chaining)

- If the load factor is above 0.75, double the capacity of the hash table, rehashing all current elements
- Then, try to add the given key and value
- If the key already exists, update its value and return **`false`**
- Otherwise add the new key and value and return **`true`**

```java
public boolean put(int key, Object value)
```

Maps in the Java Collections Framework

Maps

- Recall that the symbol table ADT is sometimes called a **map**
- Both Java and C++ use the name map for the symbol table classes in their standard libraries
- Python calls it a dictionary (and supports it in the language, not just in libraries)

Concrete example

- We've been working so long on trees and hash tables, we might have forgotten what a symbol table is for:
- Anything you can imagine storing as data with two columns, a key and a value
- In this way you can look up the weight of anyone
- However, the keys **must** be unique
  - Abdul and Carmen might weigh the same, but Abdul cannot weigh two different values
- There are multimaps in which a single key can be mapped to multiple values
  - But they are used much less often

Name (Key)	Weight (Value)
Abdul	210
Bai Li	145
Carmen	105
Deepak	175
Erica	205

JCF Map

- The Java interface for maps is, unsurprisingly, **`Map<K,V>`**
  - **`K`** is the type of the key
  - **`V`** is the type of the value
  - Yes, it's a container with **two** generic types
- Any Java class that implements this interface can do the important things that you need for a map
  - **`get(Object key)`**
  - **`containsKey(Object key)`**
  - **`put(K key, V value)`**

JCF implementation

- Because the Java gods love us, they provided two main implementations of the **`Map`** interface
- **`HashMap<K,V>`**
  - **Hash table** implementation
  - To be useful, type **`K`** must have a meaningful **`hashCode()`** method
- **`TreeMap<K,V>`**
  - **Balanced binary search tree** implementation
  - To work, type **`K`** must implement the **`compareTo()`** method
  - Or you can supply a comparator when you create the **`TreeMap`**

Code example

- Let's see some code to keep track of some people's favorite numbers

```
Map<String,Integer> favorites = new TreeMap<>();

favorites.put("John", 42); // Autoboxes int value
favorites.put("Paul", 101);
favorites.put("George", 13);
favorites.put("Ringo", 7);
if (favorites.containsKey("George")) {
     System.out.println(favorites.get("George"));
}
```

JCF Set

- Java also provides an interface for sets
- A set is like a map without values (only keys)
- All we care about is storing an unordered collection of things
- The Java interface for sets is **`Set<E>`**
  - **`E`** is the type of objects being stored
- Any Java class that implements this interface can do the important things that you need for a set
  - **`add(E element)`**
  - **`contains(Object object)`**

Quiz

Upcoming

Next time…

- Timing comparison of hash tables and trees
- Graphs
- Graph representations

Reminders

- Start Project 3
- Work on Assignment 4
- Read 4.1