CS 310: Hash Table Collision Resolution

Chris Kauffman

Week 6-2
Logistics

Reminders

- Career fair today and tomorrow at the JC
- No class next Monday (Columbus Day)
- We meet next Tuesday and Wednesday

Midterm Exam

- Two weeks from today
- Covers material through next week (Hash tables)
- Open resource

HW 2

- Due in 7 days
- Test Cases now available: 256 initially
- May be a revision which adds a few

Goals Today

- Hash Functions
- Collision Resolution
- Separate Chaining In Hash Tables
Practice: Hash Codes for these Fine Fellows?

public int hashCode()

Ideas for hashCode() implementation of the following things

Fundamental Types

- Integer
- Long
- Character
- Boolean
- Float
- Double

Container Types

- Integer []
- Double []
- String []
- ArrayList<T>
- LinkedList<T>

class Flurb{
    int x;
    double y;
    String s;
}
Basic `hashCode()` Strategy

Poor man’s strategy: `x.toString().hashCode()`
More thoroughly …

Fundamental Types

- All have a fixed size in bytes
- `int` has 4 bytes
- Convert bytes of intrinsic to 4 bytes
- If shorter than 4 bytes like `Character`, done
- If 8 bytes like `Long`, `Double`, use XOR to reduce 8 to 4 bytes

Container Types

- Use String approach
- Polynomial hash code of elements
- For each element compute its hash code
- Update polynomial hash code
- Treat fields as part of the sequence

Trivia

Did anyone find two different strings with the same hash code?
Trivia Answers

Two different strings with the same hash code

> "Aa".hashCode()
2112
> "BB".hashCode()
2112
> 'A'+0
65
> 'a'+0
97
> 'B'+0
66
> 'A'*31+'a'
2112
> 'B'*31+'B'
2112
Write a static hash function that will take any Object and compute a valid hash code that follows the hash rule.

```
public static int hashAny(Object o)
```

**Hint:** this is possible but really hard in java, will involve recursion, and will likely have pitiful runtime performance. You’ll need to use the mysterious [Reflection API](https://docs.oracle.com/javase/8/docs/api/).

*To inspire jealousy:* Other programming languages kindly define suitable hash functions automatically for new data types

- Clojure: yes!
- OCaml: yes!
- Java: nope…
- Python: nope…
- Standard ML: nope…
- Julia: nope…
- C/C++: well, what do you think…
Hash Table Class So Far...

class MyHashSet<T>{
    T hta[]; int size;
    boolean contains(T x){
        int xhc = x.hashCode();
        // If xhc out of bounds?
        xhc = ???;
        // Is this okay?
        return
            x.equals(this.hta[xhc]);
    }
    void add(T x){
        int xhc = x.hashCode();
        // If xhc out of bounds?
        xhc = ???;
        // Is this okay?
        this.hta[xhc] = x;
        this.size++;
    }
}

▶ Simple Hash Set with add(x)/contains(x)
▶ What if xhc is out of bounds in hta?
▶ Unconditionally set hta[xhc] to x in add(x)?
Pragmatic Collision Resolution: Separate Chaining

Motivation

- Put x in table at hta[xhc]
- **Problem:** What if hta[xhc] is occupied?

Separate Chaining

Most of you recognize this problem can be solved simply
- Internal array contains lists
- Add x to the list at hta[xhc]

```java
public class HashTable<T>{
    private List<T> hta[];
    ...
```
Separate Chaining: Example

Code

```java
String [] sa1 = new String[]{
   "Chris","Sam","Beth","Dan"
};

SeparateChainHS<String> h =
   new SeparateChainHS<String>(11);

for(String s : sa1){
   h.add(s);
}
print(h.load());
// load = 4 / 11
// 0.36363636363636365
```

Load = 0.36

\[
load = \frac{\text{item count}}{\text{array length}}
\]
Separate Chaining: Example

Code

String [] sa2 = new String[]{
   "Chris","Sam","Beth","Dan",
   "George","Kevin","Nikil",
   "Mark","Dana","Amy","Foo",
   "Spike","Jet","Ed"
};

SeparateChainHS<String> h =
   new SeparateChainHS<String>(11);

for(String s : sa2){
   h.add(s);
}

h.load();
// load = 14 / 11
// 1.2727272727272727
Implement Separate Chaining

- Write add/remove/contains for SeparateChainingHS
- What are the time complexities of each method?

```java
public class SeparateChainingHS<T>{
    private List<T> hta[];
    private int itemCount;

    // Constructor, n is initial size of hta[]
    public SeparateChainingHS(int n){
        this.itemCount = 0;
        this.hta = new List<T>[n];
        for(int i=0; i<n; i++){
            this.hta[i]=new LinkedList<T>();
        }
    }

    public void add(T x); // Add x if not alread present
    public void remove(T x); // Remove x if present
    public boolean contains(T x); // Return true if x present, false o/w
}
```
Separate Chaining Viable in Practice

Java’s built-in hash tables use it

- Simple to code
- Reasonably efficient
- `java.util.HashSet / HashMap / Hashtable` all use separate chaining

**Code** shown in Weiss pg 799

- Rolled own linked list
- No remove (write it yourself)
- Part of code distribution

Analyses of methods are influenced by **Load**

\[
load = \frac{\text{item count}}{\text{array length}}
\]
**Analysis**

**add()**
add(x) is $O(1)$ assuming adding to a list is $O(1)$

```java
int xhc = x.hashCode();
List l = hta[ abs(xhc) % hta.length ];
l.add(x);
```

**remove() / contains()**
> Assume fair hash function (distributes well)
> **Load** is the average number of things in each list in the array.
> **remove(x) / contains(x)** must potentially look through **Load** elements to see if x is present
> Therefore complexity $O(\text{Load}) = O\left(\frac{\text{itemCount}}{\text{arraySize}}\right)$