Hashing

CS 240

Alex Vondrak

ajvondrak@csupomona.edu

Winter 2012

Hashing

Definition (Hash Function)

A function $h\colon K \to H$ that maps a large set of keys to a smaller set of hash codes (or simply hashes)

- For programming purposes, typically $H = \mathbb{N}$—integers suitable for array indices
- In Java, the `hashCode` method of every `Object` returns an **`int`**

Example

In Java, the hash code of a `String` `s` with a length `n` is computed by

$$h(\texttt{s}) = \sum_{i=0}^{\texttt{n}-1} \left(\texttt{s.charAt(i)} \times 31^{\texttt{n}-1-\texttt{i}}\right)$$

Perfect vs Imperfect Hashing

Definition (Collision)

A collision occurs when keys k_1 and k_2 hash to the same value, v

Definition (Perfect Hash Function)

A hash function that produces no collisions (i.e., a 1-1 function)

Example

A trivially perfect hash function maps the i^{th} element of K to just i:

$$\begin{aligned}\texttt{""} &\mapsto 0\\ \texttt{"a"} &\mapsto 1\\ \texttt{"aa"} &\mapsto 2\\ &\vdots\end{aligned}$$

Open-Address Hashing

Let's populate the following array using the hash function $h(n) = n \% 10$ to generate our indices

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

314 159 265 358 97 9323 84692

After populating, what happens if we search for, say, 217?

(A) Insert element at empty index $i =$ __________

(B) Hash collision

(C) Try at the next index, $i =$ __________

(D) We've tried every index, grow the array

Open-Address Hashing

Let's populate the following array using the hash function $h(n) = n \% 10$ to generate our indices

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

31 41 59 26 53 58 97 932

After populating, what happens if we search for, say, 27183?

(A) Insert element at empty index $i =$ __________

(B) Hash collision

(C) Try at the next index, $i =$ __________

(D) We've tried every index, grow the array

Open-Address Hashing

Let's populate the following array using the hash function $h(n) = n \% 10$ to generate our indices

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

10 20 30 40 50 60 70 80 91

After populating, what happens if we search for, say, 271?

(A) Insert element at empty index $i =$ ____________

(B) Hash collision

(C) Try at the next index, $i =$ ____________

(D) We've tried every index, grow the array

Multiple Choice Question

In the best case, what is the complexity of inserting a value using open-address hashing?

(A) $\Omega(1)$

(B) $\Omega(\log n)$

(C) $\Omega(n)$

(D) $\Omega(n^2)$

Multiple Choice Question

In the worst case, what is the complexity of inserting a value using open-address hashing?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) $O(n^2)$

Multiple Choice Question

How could an open-address hash implementation in Java compute the index of an arbitrary `Object o`?

(A) `o.hashCode()`

(B) `o.hashCode() % array.length`

(C) `Math.abs(o.hashCode()) % array.length`

(D) None of the above

Multiple Choice Question

Suppose we want a hash table that works much like our `AssociationList<K, V>`.
How should we modify this storage scheme to keep track of the two items—the key and the value?

(A) Use two arrays—one for the keys, one for the values

(B) Hash the key for the index, store the value in the array

(C) Both of the above

(D) None of the above

Open-Address Hash Table

$h(k) = k \ \% \ 10$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

$38 \mapsto 31 \quad 98 \mapsto 14 \quad 48 \mapsto 15 \quad 15 \mapsto 92 \quad 53 \mapsto 65 \quad 90 \mapsto 35 \quad 29 \mapsto 89$

After populating, what happens if we look up the key 48? 58?

(A) Insert element at empty index $i =$ ________________

(B) Hash collision

(C) Try at the next index, $i =$ ________________

(D) We've tried every index, grow the array

Reducing Collisions

There are many ways to design a hash function & table structure...

Division Hash Function

- What we've used so far (modular arithmetic)
- Certain table sizes work better for this: prime numbers of the form $4k + 3$ (like $1231 = 4 \times 307 + 3$)

Mid-Square Hash Function

Return some middle digits of k^2

Multiplicative Hash Function

Pick a c such that $0 < c < 1$; return the first few fractional digits after the decimal point in $c \times k$

Reducing Collisions From Linear Probing

Definition (Linear Probing)

The demonstrated process of searching ahead for vacant spots in the array one index at a time

Definition (Clustering)

When several different keys hash to the same location, elements tend to cluster around each other, which is a problem (values aren't well-distributed across the hash table)

Definition (Double Hashing)

Instead of look at the index `(i + 1) % data.length` for each failed index `i`, we have a second hash function, and look at

```
(i + hash2(key)) % data.length
```

Multiple Choice Question

Suppose we use double-hashing to start at index 0, but instead of linear probing, our particular key has us "hop forward" by 2.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

What's the problem with this?

(A) It's inefficient

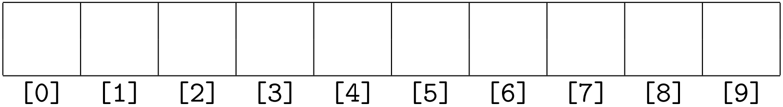
(B) If we search for the key later, we have to use linear probing

(C) We only probe half of the array, in this case

(D) None of the above

Multiple Choice Question

Suppose we use double-hashing to start at index 0, but instead of linear probing, our particular key has us “hop forward” by 2.



What’s the problem with this?

(A) It’s inefficient

(B) If we search for the key later, we have to use linear probing

(C) We only probe half of the array, in this case—since `hash2` returns a value that’s not relatively prime to `data.length`

(D) None of the above

Double Hashing

So sayeth Knuth:

- For the `data` array, both the `data.length` and `data.length - 2` must be prime (i.e., they're twin primes)
- `hash1(k) = Math.abs(k.hashCode()) % data.length`
- `hash2(k) = 1 + (Math.abs(k.hashCode()) % (data.length - 2))`

Then the double-hashing scheme returns a `hash2` that's relatively prime to `data.length`

Chained Hashing

Definition

- Like open-addressing, store data in an array
- Use hash function to generate index into array
- Use an array of linked lists
- When a hash collision occurs, simply add element to linked list

Note

To have an array of instances of a generic class, you need to have a cast like

```
(Node<K, V>[]) new Node[10];
```

Chained Hashing

Use the hash function $h(n) = n \% 10$

314 159 265 358 97 9323 84692

After populating, what happens if we search for, say, 217?

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Chained Hashing

Use the hash function $h(n) = n \% 10$

31 41 59 26 53 58 97 932

After populating, what happens if we search for, say, 27183?

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Chained Hashing

Use the hash function $h(n) = n \% 10$

10 20 30 40 50 60 70 80 91

After populating, what happens if we search for, say, 271?

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Multiple Choice Question

In the best case, what is the complexity of inserting a value using chained hashing? Looking up a value?

(A) $\Omega(1)$

(B) $\Omega(\log n)$

(C) $\Omega(n)$

(D) $\Omega(n^2)$

Multiple Choice Question

In the worst case, what is the complexity of inserting a value using chained hashing? Looking up a value?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) $O(n^2)$

Efficiency

Definition (Load Factor)

$$\alpha = \frac{\text{Number of elements in the table}}{\text{The size of the table's array}}$$

Open Addressing With Linear Probing

With a non-full array, no removals, and $\alpha < 1$ the average number of elements examined in a successful search is approximately

$$\frac{1}{2}\left(1 + \frac{1}{1 - \alpha}\right)$$

Efficiency

Definition (Load Factor)

$$\alpha = \frac{\text{Number of elements in the table}}{\text{The size of the table's array}}$$

Open Addressing With Double Hashing

With a non-full array, no removals, and $\alpha < 1$ the average number of elements examined in a successful search is approximately

$$\frac{-\ln(1-\alpha)}{\alpha}$$

Efficiency

Definition (Load Factor)

$$\alpha = \frac{\text{Number of elements in the table}}{\text{The size of the table's array}}$$

Chained Hashing

The average number of elements examined in a successful search is approximately

$$1 + \frac{\alpha}{2}$$

Multiple Choice Question

Definition (Load Factor)

$$\alpha = \frac{\text{Number of elements in the table}}{\text{The size of the table's array}}$$

What does it mean if $\alpha \geq 1$?

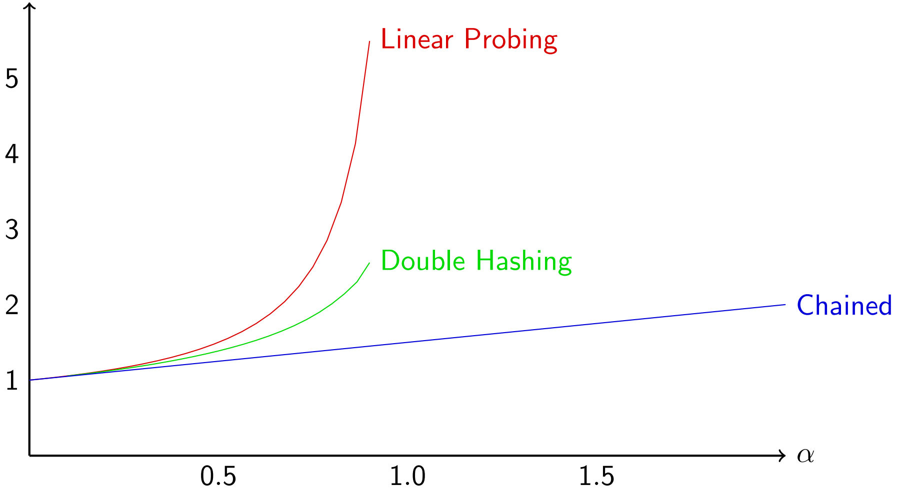
(A) The array is full of elements

(B) It's impossible for $\alpha > 1$

(C) The array needs to grow

(D) Donald Knuth is angry

Efficiency



Using Java's Hash Tables

docs.oracle.com/javase/6/docs/api/java/util/Hashtable.html

Example

```
import java.util.Hashtable;
...
Hashtable<String, Integer> env =
    new Hashtable<String, Integer>();
env.put(null, null); // ERROR: null not allowed!
env.put("one", 1);
env.put("two", 2);
env.put("one", 100); // overwrites old "one"

Integer one = env.get("one");
if (one != null) {
    System.out.println("one = " + one);
}
```

Using Java's Hash Tables

docs.oracle.com/javase/6/docs/api/java/util/HashMap.html

Example

```
import java.util.Map;
import java.util.HashMap;
...
Map<String, Integer> env =
   new HashMap<String, Integer>();
env.put(null, null); // OKAY: null allowed
env.put("one", 1);
env.put("two", 2);
env.put("one", 100); // overwrites old "one"

Integer one = env.get("one");
if (one != null) {
   System.out.println("one = " + one);
}
```