Week 6

Assignments:
Program 2: is being graded
Program 3: available soon and due before 10pm on Thursday 3/14
Homework 5: available soon and due before 10pm on Monday 3/4
X-Team Exercise #2: due before 10pm on Thursday 2/28
X-Team Exercise #3: due before 10pm on Monday 3/4

MIDTERM EXAM
THURSDAY MARCH 7th, 5:00 pm – 7:00 pm

- □ **Lecture 001 students: Room 6210 of Social Sciences Building**
- □ **Lecture 002 students: Room B10 of Ingraham Hall Building**
- □ **Lecture 004 students: Room 125 of Agriculture Building**
- **Bring**
  - **UW ID**
    Note: if you do not have your UW ID, you will be asked to wait until after students with ID have been admitted
  - **#2 Pencils**
  - **good eraser**
- **At Exam:**
  - **arrive early**
  - **get ID scanned**
  - **get scantron form**
  - **find seat directly behind another student**
- **See posted exam information**

Read: Module 6 for this week and 7 for next week

THIS WEEK:

- Hashing
- Ideal Hashing
- Techniques for generating hash codes
- Handling `String` keys
- Handling `double` keys
- Choosing Table Size
- Resizing a hash table
- Collision Handling

NEXT WEEK:

- Graphs
- Exam Review
- Midterm Exam

Hashing

Goal:

Concept:

Terms:

key

hash function

hash index

hash table

table size (TS)

load factor (LF)

Java's **hashCode()** method

Ideal Hashing

Assume

- need to store 150 students records
- table is an array of student records
- null is sentinel value meaning that table element is unused
- key is the student's id number, and it is one of the following 5 digit integers

  11000, 11001, 11002, … 11048, 11049, … 11148, 11149

→ What would be a good hash function to use on the ID number?

```
     int hash(K key) {


     }
```

Trivial Hash Function:

Perfect Hash Function:

```
void insert(K key, D data) {

D    lookup(K key)         {

void delete(K key)         {
```

The UW uses 10 digit ID numbers: 9012345789   9012345432   9023456789

→ Is a perfect hash function possible for these id numbers?

→ Would the last 3 digits of the ID work as above?

Collision:

Key Issues:

-
-
-

Designing a Hash Function

Good Hash Functions:

1.

2.

3.

4.

Java's hashCode function:

What is it?

How can we use it?

Techniques for Generating Hash Codes

Integer Key 90123456789

```
 123 * 11     +     456 * 121     +     789 * 1
```

Extraction

Weighting

Folding

Handling `String` Keys

Handling `Double` Keys

Choosing the Table Size

Table Size and Collisions

Assume 100 items with random keys in the range 0 – 9999 are being stored in a hash table. Also, assume the hash function is: (key % table_size).

→ How likely would a collision occur if the table had room for 10000 elements? 1000? 100?

ITEMS	*T.S.*	*Expected* *# of collisions*	*L.F.*
100	10,000		
100	1,000		
100	100		
100	10		

*

Table Size and Distribution

Now, assume 50 items are stored in a hash table.
Also assume the hashCode function returns multiples of some value x.
For example, if x = 20 then hashCode returns 20, 40, 60, 80, 100, ...

How likely would a collision occur if the table had 60 elements? 50? 37?

N	*TS*	*# collisions*
50	60	
50	50	
50	37	

Resizing the Hash Table

Naïve Expand

30		17	88	

Rehashing

1.

2.

Complexity

Collision Handling using Open Addressing

Open Addressing

Linear Probing

166
359
263

440		266	124	246			337			351

Collision Handling using Open Addressing

Quadratic Probing

166
359
263

440		266	124	246			337			351

Double Hashing

probe sequences assuming H_k is index 0:

Step size	Table size 10	Table size 11
2		
5		

Collision Handling using Buckets

Buckets

Array Buckets

“Chained” Buckets

Tree Buckets

Java API Support for Hashing

`hashCode` method

- method of Object class
- returns an int
- default hash code is BAD - computed from object's memory address

Guidelines for overriding `hashCode`:

`Hashtable<K,V>` and `HashMap<K,V>` class

- in java.util package
- implement `Map<K,V>` interface
  `K` type parameter for the key
  `V` type parameter for the value

  operations:

- constructors allow you to set
  initial capacity (default = 16 for HashMap, 11 for HashTable)
  load factor (default = 0.75)
- handles collisions with chained buckets
- HashMap only:
- Hashtable only:

TreeMap vs HashMap

	TreeMap	HashMap
Underlying d.s.		
Complexity of basic ops.		
Iterating over keys		
Complexity of iterating over values		