CS 367 - Introduction to Data Structures
Thursday, April 21, 2016

Program 5 due 10 pm **Friday**, May 6th

Homework 9 due 10 pm tomorrow, April 22nd
Homework h10 assigned Monday

Last Time

Graphs

- topological ordering
- Dijkstra's algorithm

Exam 2 returned

Today

Hashing

- terminology
- designing a good hash function
- choosing table size

Next Time

Read: finish *Hashing*, start *Sorting*

Hashing

- expanding a hash table
- handling collisions

Java Support for Hashing
Tree Map vs. Hash Map
Sorting Intro

Hashing

Goal:

Concept:

hash table

table size (TS)

load factor

key

hash function

hash index

Ideal Hashing

Assume

- store 150 students records
- table is an array of student records
- null is sentinel value meaning element is unused
- key is the student id number, a 5 digit integer

  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 Hash Function Steps:

1.

2.

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 hashtable.
Also assume the hash function is simply %tablesize.

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

Table Size and Distribution

Assume 50 items are stored in a hashtable.
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?