Data Structures and Algorithms

Session 3. January 28, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

Announcements

- Homework 1 up on website (slight change)
  - Due Feb. 9th before class
- Office Hour reminder
  - My OH today after class (this week only)

Review

- Mathematical Review
  - Exponential, log identities
  - Proofs: straightforward proof, by induction, contradiction
- Big-Oh: upper bound growth rate, invariant to constants, limit as N grows to infinity
- Maximum Subsequence example (unfinished)

Today's Plan

- Comments on homework
- Finish Big-Oh examples from previous slides
- Abstract Data Types
- Lists
  - Array List Implementation
  - Linked List Implementation
- Brief Intro to Java Collections API

Homework Notes: Written Problems

- Hint for proofs (1.7 and 1.12): some are easier to prove via induction
- Coming up with proofs and new algorithms takes creativity. Sometimes it just won't come to you:
  - Take a break, come back to it later
  - Try thinking of alternate but equivalent ways of posing the problem

Homework Notes: Programming

- All code in the book is fair game to re-use
- Selection problem:
  - select(k) finds the kth largest number in an array
- Graphing class: need to run locally to see graphics
  - I will post detailed instructions and an example soon (today or tomorrow)

Abstract Data Types

- Defined by:
  - What information it stores
  - How the information is organized
  - How the information can be accessed
- Doesn’t specify **implementation**

Vs. Implementation

- What information it stores
  - `What classes/types of variables`
- How the information is organized
  - `How it is stored in memory`
- How the information can be accessed
  - `What methods (and algorithms)`

Abstract Data Type: Lists

- An ordered series of objects
- Each object has a previous and next
  - Except *first* has no previous, *last* has no next
- We can insert an object to a list (at location *k*)
- We can remove an object from a list
- We can read an object from a list (location *k*)

Applications for Lists

- To Do: insert tasks, remove when done
- Word Processor:
  - typing text inserts to list,
  - deleting text removes (simple array won’t work)
- Shopping: insert needed items, remove when bought

Array Implementation of Lists

- 1st Hurdle: arrays have sizes
  - Create bigger array when we run out of space, copy old array to big array
- 2nd Hurdle: Inserting object anywhere but the end
  - Shift all entries forward one. O(N)
- Get kth and insertion to end constant time O(1)

Linked List Implementation

- Store list objects anywhere in memory
- Each object has a reference to its next object
- Insert at *k* requires T(get kth) + constant
  - Insert to front is constant time
- T(get kth) = O(N), if naïve

Linked Lists vs. Array Lists

- Linked Lists
  - No additional penalty on size
  - Insert/remove O(1)*
  - get kth costs O(N)*
  - Need some extra memory for links
- Array Lists
  - Need to estimate size/grow array
  - Insert/remove O(N)*
  - get kth costs O(1)
  - Arrays are compact in memory

Lists in Java

- **Collection** Interface extends **Iterable**
- A Collection stores a group of objects
- We can add and remove from a Collection
- **Iterator** objects let us iterate over objects in a Collection (also enhanced **for** loop)
- Built in **LinkedList** and **ArrayList** implementations of Collection

Assignments

- Previous Reading: Ch. 1, Ch. 2, Sections 3.1-3.5
- Start homework