Review for Final

Victor Milenkovic

Department of Computer Science
University of Miami

CSC220 Programming II – Spring 2018

Tides Foundation (Google Education and University Relations Fund)

- Funding from Google to make the course more accessible to non-CS majors.
  - Visual Debugger
  - Tools for final project
    - BetterBrowser
    - trie
    - external storage
- We will be posting a survey.
- In addition to usual course evaluation.
- Did you use the Visual Debugger?

What to do

- Reread notes.
- Go over the Collections Framework tutorial.
- Retake quizzes.
- If you didn't get full credit for an assignment, find out what you did wrong.
- Go over review for midterm.
- Go over the midterm.

Remember the first lecture?

- A computer has perhaps 2 billion bytes of RAM and runs at about 2 billion instructions per second. So it takes a second to find something.
- Google finds things faster, even though...
- Powers of two:
  - $2^{10} = 1024$ which is about 1000
  - log2(1000) is about 10
  - log2(1000000000) is about 30
- To find the bad coin in 1000:
  - weigh 500
  - weigh 250
  - weigh 125
  - weigh 63
  - weigh 32
  - weigh 16
  - weigh 8
  - weigh 4
  - weigh 2
  - weigh 1
- 10 weighings instead of 1000.
- $log_2 n$ instead of n.

prog01

- Object oriented programming:
- What the compiler allows.
- Casting.
- What will cause a run time error.
- Which method will be invoked.

prog02

- Double array size when reallocating (that's what ArrayList does).
- System.arraycopy
- inserting into a sorted array
- binary search
- $O()$ for all ArrayBasedPD and SortedPD methods

prog03

- Timing using repetition to get accuracy.
- Divide System.nanoTime() by 1000.0 (not 1000) to get microseconds.
- Average of $1000000/t$ calls to keep experiment less that one second.
- How fast is your computer?
- Use of $\mathrm{O}()$ to estimate running time.
- ExponentialFib, LinearFib, PowerFib, LogFib, ConstantFib.

prog04

- doubly linked list
- inserting, removing, finding in a sorted DLL
- List interface
- $O()$ for List.get()
- Why not use binary search?

prog05

- Stack operations.
- Stack implemented using array or singly linked list.
- Using a number stack and operator stack to implement expression evaluation.
- How is operator precedence implemented?

prog06

- Queue interface.
- Implementation using singly linked list.
- Implementation using circular array.
- Use of AbstractQueue to implement Queue.
- Iterator interface.
- Implementation of Iterator.
- New type of for loop thanks to Iterator.
- Word Puzzle
- Use of queue to search for connections
- Breadth first search (who else uses this?)

prog07

- Map interface.
- Jumble dictionary.
- AbstractMap
- Inserting, finding, removing from sorted doubly linked list.
- Generating coin flips in the computer.
- Idea of skip list: keys on multiple levels.
- How many levels does a key appear? What's the average?
- How many nodes on each level?
- How many levels?
- How far does find go on each level?
- $O()$ for find?
- $O()$ for all operations?

prog08

- binary tree: root, leaf, height, left, right
- linked implementation using left and right pointer
- array implementation using 2i+1 and 2i+2 (and (j-1)/2)
- binary search tree: search order
- binary search tree implements Set or Map: TreeSet or TreeMap
- heap: heap order
- PriorityQueue uses heap to implement Queue
- finding, inserting, and removing from linked binary search tree
- adding to array heap: swapping up
- removing root from array heap: swapping down
- binary tree $O(n)$ per operation if input is bad: sorted
- heap is guaranteed $O(\log n)$ per operation
- Comparator interface: tell the PriorityQueue how to order things.
- Also: Comparable interface. Requires compareTo method.

prog09

- sorting methods
- how does each one work
- what are its properties?
- insertion sort: compare and move in a loop.
- heap sort: only swap down!
- quick sort: really like the binary search tree, only it's possible to make it random
- merge sort
  - useful for sorting "big data"
  - merge idea is useful for Google

prog10

- hash function
- Don't use pow or ^.
- hash index
- chained hash table
  - review of linked list
  - Singly-linked so you can't go back. How do you remove?
  - It's o.k. if it gets full.
- open addressing
  - find method
  - needs to be no more than half full of DELETED plus used

prog11

- (Radix) Trie
- Guaranteed $\mathrm{O}(L)$ per operation
- Good for storing maps externally.
- Same $\mathrm{O}()$ as a hash table but sorted.

Interfaces and Implementations

- Interface: List
  - Methods: size(), add(x), add(i, x), remove(i), get(i), set(i, x)
  - Implementations: ArrayList, LinkedList
- Interface: Map
  - Methods: size(), get(k), put(k, v), keySet()
  - Implementations: TreeMap, HashMap
- Interface: Deque [StackInt]
  - Methods: empty(), peek(), pop(), push(x)
  - Implementations: ArrayDeque [ArrayStack], LinkedList [LinkedStack]
- Interface: Queue
  - Methods: size(), offer(x), peek(), poll(), [add(x), element(), remove()]
  - Implementations: ArrayDeque, LinkedList, PriorityQueue [Heap]
- Interface: Comparable
  - Methods: compareTo(that)
- Interface: Comparator
  - Methods: compare(x, y)
- Interface: Iterator
  - Methods: hasNext(), next()
- Interface: Iterable
  - Methods: iterator
  - Implementations: everything
- Interface: Set
  - Methods: add(), contains(), remove()
  - Implementations: TreeSet, HashSet

Running Times

- List implemented as ArrayList or LinkedList
  - add is $\mathrm{O}(1)$ (sort of)
  - get, set are $\mathrm{O}(1)$ for ArrayList and $\mathrm{O}(n)$ for LinkedList
  - indexOf or contains are $\mathrm{O}(n)$
  - remove $\mathrm{O}(n)$ (for different reasons)
- Stack implemented as List: push and pop at the end, all $\mathrm{O}(1)$.
- Queue implemented as linked list or circular array (ArrayDeque)
  - peek, offer, and poll are all $\mathrm{O}(1)$ for LinkedList and ArrayDeque
  - If you needed to iterate through the Queue and remove some people,
  - ArrayDeque Iterator remove would be $\mathrm{O}(n)$ but LinkedList Iterator remove would be $\mathrm{O}(1)$
- Queue implemented as PriorityQueue using heap data structure
  - offer and poll are $\mathrm{O}(\log n)$
  - peek is still $\mathrm{O}(1)$
- Set or Map
  - skip list: $\mathrm{O}(\log n)$ expected time for contains, add, remove
  - binary search tree: $\mathrm{O}(n)$ worst, $\mathrm{O}(\log n)$ average, but that can be fixed (CSC317)
  - hash table: $\mathrm{O}(1)$ expected time (really $\mathrm{O}(L)$ where L is length string)
  - external trie: $\mathrm{O}(L)$ time, but constant is a millisecond.

Using Collections Framework

- To use TreeSet or TreeMap
  - implement Comparable
  - provide compareTo
- To use HashSet or HashMap
  - implement (actually override) equals
  - implement hashCode

prog12

- How and why does Google use each interface and implementation?
- Queue: searching for new pages
- (external) Map implemented using Trie: ids of pages. Sorted good?
- Map implemented using hash table: ids of words. Unsorted o.k.?
- Hard disk acts as Map from Long to PageFile (URL and reference count).
- Hard disk acts as Map from Long to List<Long>, the page ids for each word.
- (temporary) Set of page ids on a given page (to foil Google bombers)
- array of Iterator
- PriorityQueue
- Comparator interface
- merge operation

sorting

- Know sorting algorithms:
  - Insertion Sort
  - Quick Sort
  - Heap Sort
  - Merge Sort
- Know properties:
  - worst case
  - best case
  - expected running time
  - stable
  - in place

Don't forget!

- When is the running n times $\log n$ and when is it n plus $\log n$?
- What is $\mathrm{O}(n + \log n)$?
- How many additions can your computer do per second?
- How big is RAM?
- How big is your hard disk?
  - 1ms to seek
  - Read 512 bytes at at time.
  - Read a large file very quickly.
- If I ask for an interface, please write down an interface.

Extra Credit

- Implementation of List
  - get(x) and set(i, x) are $O(1)$ *guaranteed*
  - add(x) is *always* $O(1)$
  - the space used should be $O(n)$ where n is size (no fair allocating an ginormous array!)
  - add(x) should not return false
- If you figure it out, keep it to yourself.
  - Email the design to me.
  - One shot.
  - Add 50 one prog below 50.