CSC 1052 – Algorithms & Data Structures II:

Exam Review II

Professor Henry Carter
Spring 2017

Exam Format

- Short answer questions
  - A few words to a sentence answers
  - Testing basic concepts and terminology
- Long answer
  - A few sentences to a paragraph
  - Testing deeper concepts
- Thought exercise
  - Possibly multiple subsections
  - Requires you to assemble multiple concepts into a complete solution

Recursion

- Three components of a recursive algorithm
- Three checks for correctness
- Recursive array processing
  - Use start and end indices to cut down the size
  - Example: binary search

Problem

- Given the following method:

```
int exer(int num){
    if(num == 0)
      return 0;
    else
      return num + exer(num + 1);
}
```

  - Is there a constraint on the input num?
  - Is exer(7) valid? exer(0)? exer(5)?
  - If yes, what is the output?

Recursion on Linked Lists

- Data structures sometimes have a recursive structure
  - Ex., linked lists
- Recursion makes traversing and printing linked lists simple in terms of code length
- Complications arise from modifying the linked list
  - Modify the existing list or return a pointer?
- Applications:
  - Towers of Hanoi
  - T-square fractal

Find the errors in the following implementation

- A method to return the sum of the squares of integers in a linked list:

```
int sumSquares(LLNode<Integer> list){
 return 0;
 if(list != null)
    return (list.getInfo() *
      list.getInfo() +
      sumSquares(list.getLink()));
}
```

Pitfalls/avoiding recursion

- Static vs dynamic storage allocation
- Tail recursion
  - Loop conversion
- Direct use of a stack
  - Avoid using the call frame stack
- Combinatorial blow-up
- Hidden cost of recursion?

Problem

- In your own words, define:
  - Run-time stack
  - Static storage allocation
  - Dynamic storage allocation
  - Activation record
  - Tail recursion

Queues

- FIFO data structure
- 2 basic operations
- Array-based implementation:
  - Fixed vs floating front
  - Fixed vs expanding size
- Tradeoffs?

Problem

- There are two ways a developer could check for an empty queue using the interface methods. What are they? Are they necessary?
- Draw the internal array representation for each step in the following code snippet (using wrapping):

```
ArrayBoundedQueue<String> q = new
    ArrayBoundedQueue<>(5);
q.enqueue(“X”);
q.enqueue(“M”);
q.dequeue();
q.enqueue(“T”);
```

Linked Queue

- Add a pointer: front and rear of the queue
- Eliminating the head pointer
- Comparing memory usage:
  - Pointer counts

Problem

- Consider swapping the front and rear references in our linked list representation. How would this complicate implementation?

Variations

- Removing exceptions
- Glass queue
- Deque
  - Requiring doubly linked lists
- Application: average wait time simulation

Code Trace

```
int e1 =1; int e2 = 0; int e3 = 4;
gq.enqueue(e2);
gq.enqueue(e1);
gq.enqueue(e1+e3);
e2 = gq.peekRear();
gq.dequeue(); gq.enqueue(e3 * e3);
gq.enqueue(e2);
gq.enqueue(3);
e3 = gq.peekFront();
gq.dequeue();

while(! gq.isEmpty()) {
    e1 = gq.dequeue();
    System.out.println(e1);}
```

Collections

- Data is accessed by content
  - Ignorant of index or ordering
- 4 Operations
- Collection requires that elements be equal or not equal
- Array-based implementation
  - Unsorted array (OOG?)
- Vocabulary density

Problems

- Identify the observers and transformers in the Collection interface.
- What would be the result of using the ArrayCollection class to hold objects of a class that has not overridden the equals() method from the Object class?

Comparables and sorting

- Equality must be defined per class
- Elements are compared based on “key” information
- The Comparable interface allows comparing elements with compareTo()
  - Defines ordering
- Sorting the array implementing the collection exchanged efficiency between add/remove and contains

Problem

- Recall the three criteria for an equivalence relation:
  - Reflexive
  - Symmetric
  - Transitive
- Which of the following definitions of equals meet these criteria?
  - Two circles are equal if they have the same area
  - Two circles are equal if their radii are within 10% of each other
  - Two integers are equal if they have the same remainder when divided by a specific integer (e.g., 3)
  - Two integers are equal if the second integer is a multiple of the first.

Next Time...

- Exam!
- NO electronic devices
- Peer Tutors
  - http://www.csc.villanova.edu/help/