CSC207.01 2014S, Class 54: Patterns of Design

Overview

- Preliminaries.
  - Admin.
  - Questions.
- Algorithm design.
- ADT design.
- Data-structure design.
- Object design.
- Code design.

Preliminaries

Admin

- Distributed: Draft of take-home final
- Earnest will go over sample problems from the in-class final at tonight's mentor session.
- I admit that my record keeping is not perfect. When you get grades from me and they are missing something, let me know.
- I have not been pushing most of the comments on code (which are normally on comments).

Upcoming Work

- Continue to work on the exam.
- Decide which exam you are taking (and when, for the in-class exam)
- No more readings.
- Today's writeup: No writeup.

Extra Credit

- College budget talk, today at noon or 7:30 p.m.
- CS table Friday: Casual conversation.
- Conference track meet Friday and Saturday. NBB runs at 4:05 and 5:10.
- Listen to EB's radio show on KDIC Friday at 5pm.
- Listen to DNP guest star on some radio show Friday at 11pm.

Questions (Exam and Otherwise)

Is there a way to compare two trees?

I have not written a comparator for trees. You could write one. It will probably look something like the following.

```
public static boolean equals(BSTNode one, BSTNode two)
{
  // Base case one: Both are the same node.  Obviously the same tree.
  // Also covers the both are null case.
  if (one == two)
    return true;

  // Base case two: One, but not the other is null.  Different trees.
  else if ((one == null) || (two == null))
    return false;

  // Recursive case: Both are nodes
  else return (one.key.equals(two.key)) &&
      (one.value.equals(two.value)) &&
      (equals(one.smaller, two.smaller)) &&
      (equals(one.larger, two.larger));
} // equals(BSTNode, BSTNode)
```

Should we copy values or move nodes in rearranging trees?

Move nodes. It ends up working better in the long run, at least if I trust my experience and intuition.

Algorithm design

When given an algorithm design problem, how do you get started?

- Get donuts?
- Ask a professor or other professional.
- Draw a picture of the problem.
- Make sure that we understand the problem well.
  - Specify input, types, preconditions
  - Specify output, types, postconditions, goals
  - Write unit tests
- Brainstorm about how to get from preconditions to postconditions
- Identify data structures that may be useful.
- Fiddle - Try to solve a sample problem by hand.
- See if it's been solved already - there's no reason to rebuild something that others have already built, unless you think you can do it better.

How Sam tends to approach algorithm design.

- Solve a few examlpes by hand to develop intuition.
- Formalize (informally) - Demonstrate understanding of problem, perhaps check with "client"
- Consider whether I've solved similar problems before, and see if I can adapt those algorithms.
  - May help to classify the problem
    - Optimization: Best/smallest/etc.
    - A collection of operations
    - Arrangement of data
    - Searching
    - ...
- Consider some common algorithm design strategies
  - Divide and conquer
  - Dynamic programming / caching
  - Greed, particularly for optimization
  - Learn more in 301
- Sketch
- Attempt
- Refine
- Think about edge cases
- Run tests

Much later

- If we know that we're writing a loop, write/sketch a loop invariant

ADT design

Sam's basic questions on ADT design:

- What is the overall *purpose* or *philosophy* of the ADT?
- What are the *use cases* that will guide your design?
- What *methods* will support those use cases.

Data-structure design

When given a data-structure design problem, how do you get started?

- See above.
- Main approaches to organizing data
  - Array (contiguous indexed memory)
  - One-dimensional linked structure
  - Two-dimensional linked structurs (e.g., trees)
  - Hybrid

What are Sam’s basic questions on data structure design?

Object design

"Design Patterns"

Ways of thinking about common problems.

A language for expressiong common solutions.

I expect you to know (and have seen)

- Factory
- Model/View/Controller
- Adapter
- Singleton
- Iterator
- Observer
- Decorator

Code design