Lecture 8: Practice Design

Solving a larger problem: practice with wish lists, decomposing problems into subproblems, recursive methods on trees, methods with accumulator parameters

Problem 1

Suppose you are working on a research paper, and you have gathered a set of documents together for your bibliography: books and Wikipedia articles. Every document has an author, a title, and a bibliography of documents; additionally, books have publishers, and wiki articles have URLs.

- Since you know that wiki articles are not necessarily authoritative sources *[citation needed]*, you want to produce a bibliography containing just the authors and titles of the books you've found, either directly or transitively through the bibliographies of other documents. Format the entries as "Last name, First name. "Title"."
- Since bibliographies must be alphabetized, sort the bibliography by the authors' last names.
- Documents may be referenced more than once, but should only appear in the bibliography once. Remove any duplicates (defined as the same author name and the same title)

> ***Do Now!***
>
> Design your data definitions to represent this information. Design a method to produce a properly formatted bibliography. Design whatever helper methods you need to solve the problem well.

Problem 2

Variant A

Suppose you are given a list of integers. Your task is to determine if this list contains:

- A number that is even
- A number that is positive and odd
- A number between 5 and 10, inclusive

The order in which you find numbers in the list satisfying these requirements does not matter. The list could have many more numbers than you need. Any number in the list may satisfy multiple requirements. For example, the list (in Racket notation) `(list 6 5)` satisfies all three requirements, while the list `(list 4 3)` does not.

> ***Do Now!***
>
> Design a method on lists of integers to check whether the list satisfies these criteria. Hint: what information do you need to propagate down the recursive calls as you process the list?

Variant B

Again, the list must contain numbers satisfying the three requirements above. Again, order does not matter. This time, a given number in the list may only be used to satisfy a single requirement; however, duplicate numbers are permitted to satisfy multiple requirements. So, `(list 6 5)` does *not* meet all the criteria for this variant, but `(list 6 5 6)` *does*.

> ***Do Now!***
>
> Design a new method on lists of integers to check for this stricter property. How does your design differ from Variant A?

Variant C

Again, the list must contain numbers satisfying the three requirements above. Again, order does not matter. Again, a given number in the list may only be used to satisfy a single requirement. This time, however, the list may not contain any extraneous numbers. So, `(list 6 5 6)` satisfies all our criteria for this variant, but `(list 6 5 42 6)` does *not*.

> ***Do Now!***

Design a third method on lists of integers to check whether the list meets this new property.

Variant Z

Exercise

(Very open-ended) There is something distinctly unsatisfying about designing these solutions to work for precisely three requirements: why not four, or five, or an arbitrary number? Design *functions in DrRacket* to solve the preceding three problems again, this time supporting an arbitrary number of requirements to be checked on the elements of the list. What essential features of Racket are you using, that we do not yet have in Java?