COMP 340 - Lecture Notes - 01 - Introduction

Spring 2014

In these notes we discuss bringing HtDP design ideas over to the realm of Algorithms. In doing so we explore some larger themes in the design of computational objects and perhaps design in general.

"An algorithm is a procedure to accomplish a specific task."[1]

[1] Skiena. pg. 3

How to Design Algorithms

Here at Monmouth we start out by looking at *How to Design Programs*[2], which provides you with some recipes [3] for designing and implementing computer programs. Our main text in this course class is titled *The Algorithm Design Manual*. So from page zero, it's clear we're looking at how to construct algorithms and not just planning memorize algorithms' greatest hits. You should quickly notice similarities in the HtDP approach and the algorithm design process advocated by Skiena. Let's go ahead and connect some dots though and let this connection start to fester in our brains.

[2] `http://htdp.org`

[3] Note quite algorithms

HtDP Recipes

Let's take a high-level view of the HtDP recipe[4].

[4] with some modifications

1. Analyze the problem and formulate *Data Definitions* for representations of problem information as computational data[5]. Develop examples of specific information as data. Iteratively refine data representations based on examples.
2. Analyze the problem and for each function[6], formulate a *signature*, *statement of purpose*, and as needed *preconditions*, *postconditions*, and *descriptions of side effects*. Develop examples of specific instances of the functions and formulate them as test cases. Iteratively refine function specifications based on examples.
3. Write templates for each function based on the structure[7] of function input(s).
4. Code using test cases as a guide; Iteratively refine the program and program design based on further examples/tests.
5. Evaluate the completed program and repeat the above steps in order to iteratively improve or generalize the program[8] based on example runs of the program and individual test cases.

[5] objects, variables, etc.

[6] procedure, method, etc.

[7] Atomic,Compound,Itemization,Self-Referencing,Mutually-Referencing, or Cyclic

[8] abstraction

It's crucial to recognize that this process is, at several levels, informed by the examination of *specific, concrete, and actionable examples*. When we're deciding on the best data representation, we use concrete

examples to determine if our *data model* can express all the problem information we need to express. We can then iteratively improve our model based on our experience with our examples. When we're choosing the right set of functions for our program, we use examples to be sure we've covered all the needed functionality and relationships between problem information. Once again, experience earned by examining specific examples allows us to iteratively improve upon our function choices. When our code is written, our examples become the tests that allow us to evaluate our program for some basic correctness[9] and run-time efficiency. Finally, the entire process can then be repeated in order to improve upon or program or generate more general purpose, reusable code via abstraction. In this case, concrete instances of running the program and its tests help to inform the process. To better understand this process, you can find a large number of specific examples of it in action in HtDP1e and HtDP2e.[10]

[9] the only guarantee is that our behaves correctly for our tests!

[10] Key to computing success?:

1. Find concrete examples
2. Profit!

How to Design X

Let's abstract on the HtDP design recipe a little bit.

1. *Modeling:* Identify the real-world problem and iteratively develop a model[11] of the problem through well chosen examples.
2. *Planning:* Develop an outline/plan for the solution[12] to the model of the problem.
3. *Implementing:* Iteratively implement the solution with the help of concrete test cases
4. *Improving:* Iterate the design process to improve[13] upon the overall solution.

[11] {functional, procedural, Object-Oriented} → computational

[12] programmatic

[13] better models, more general solutions, etc.

You could probably capture a lot of design processes with this four step model. I'll leave it to you to think about that. We'll stick to Programs and Algorithms for now. If this is an accurate, generalized model of our program design process, then it should capture our HtDP process. It certainly seems to, and that's not a big surprise as this generalization is based upon the HtDP process! The real test is to see if this model captures a new example. The example we're concerned with is *How to Design Algorithms.* [14]

[14] hopefully the choice of language in this paragraph was not lost on you

How to Design Algorithms

Skiena isn't as direct[15] with his design process as HtDP, but it's clear from the first chapter of the text that his design process goes something like this:

[15] Chapter 10 is pretty darn close though

1. Identify the real-world problem, give it a *name*, and identify the problem's *inputs and outputs*. Choose an abstract model[16] for it's input and output. Check models against concrete instances of the problem.
2. Formulate a high-level strategy for solving the model of the problem.
3. Write a complete pseudo-code algorithm.
4. Evaluate algorithm correctness by finding counter-examples to prove it incorrect or proving correctness mathematically[17]. Use concrete examples to develop proofs or to motivate improvements to algorithm for the sake of correctness.
5. Evaluate algorithm efficiency under the RAM computing model[18] using asymptotic[19] worst-case[20] analysis. Use concrete example to motivate improvements for the sake of efficiency.
6. Generate examples of implementations of the algorithm on a real-world computing system. Evaluate the algorithm for ease of implementation. Evaluate the implementations for correctness and efficiency[21].

[16] combinatorial, recursive, etc.

[17] probably with induction

[18] or a suitably chosen model of computation

[19] big Oh

[20] average-case analysis is a field of research in and of itself

[21] This is the field of Experimental Algorithmics

Step one is clearly the analysis and modeling phase. Step two is the planning phase. Step three is implementation. The remain steps are the improvement phase broken down by the attributes of algorithms that we want to improve upon: *correctness, efficiency, and ease of implementation*. So, it seems that our generalized HtD*X* model is sufficiently expressive for both HtDP and HtDA. However, we really need to put the HtDA recipe through its paces and evaluate it based on some concrete examples.[22]

[22] That should take us, oh, about one semester.

How to Design Proofs

Much of our HtD*X* process applies to designing and writing mathematical proofs. We'll do some of that in this class, and if writing proofs is something you've struggled with[23], then you should really reflect on how to cast your proof writing process within this mold. That's all I'm going to say about proofs at this point, but I do want to point out a really excellent, basic, and free text called *The Book of Proof*[24] by Richard Hammack. It's well worth checking out if you're looking to step up your proof writing game.

[23] not uncommon. don't feel bad about it. do try to get better at it though

[24] `http://www.people.vcu.edu/~rhammack/BookOfProof/`

And so...

If you take away one big picture thing from here[25] it might be this:

[25] other than concrete examples are key

> There is an awful lot of constructive, tangible work to be done before you commit one line of pseudocode to the page.

The process we plan to use doesn't let you get away with just thinking hard about the problem. It requires that you commit your thoughts to paper and write them down. The value of this step cannot be overstated. It will help you check yourself. It will help you communicate your thoughts to others.