605.202: Introduction to Data Structures

W.T. Door

Project 0 Analysis

Due Date: June 20, 2021

Dated Turned In: June 21, 2021

Project 0 Analysis

The stack data structure seems to be a very good way to get things back in reverse order. In this problem, there are three main components: user input/output through the file system, conversion of the decimal into a binary number, and use of the stack to hold intermediate results.

The user input and output probably takes longer than the actual application would because the numbers have to be converted into output text. In a real compiler, the results would stay in the primitive type domain.

Conversion of the number is very simple, and there are many ways to store the intermediate results. An array could have been used and the results just read back in reverse order. This would be more efficient than the work used to set up the stack. However, this would be a very basic algorithm that just applied to this project. The stack object allows reuse of the stack for various other project types.

The stack itself is implemented as an array. The Java programming language itself provides a Stack class (java.util.Stack) that handles any type of element. In the array version of the stack that was used for this assignment, there is a limit to the number of items that can be stored in the stack. The Java Stack class uses an underlying Vector class that can grow with the size of the problem so that only the available memory becomes a restriction.

The overall problem runs in O(log[2]) time. This is because the conversion algorithm repeatedly cuts the problem in half. The O(1) access time of the stack does not change this because the stack is used only to hold the remainder from each division. Figure 1 shows the problem growth with problem size.

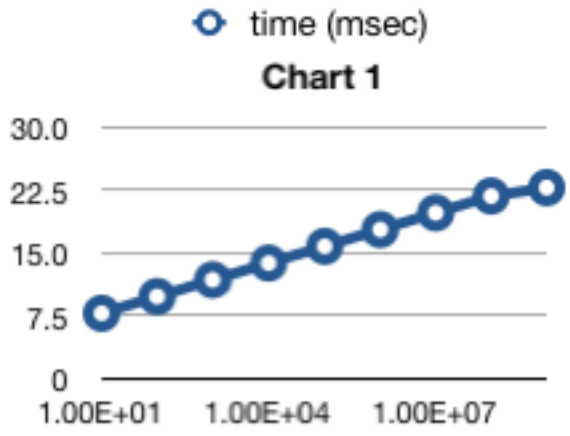

Figure 1: Graph of Results

What I Learned

There is a lot of startup overhead in the Java application. The first couple of conversions take a very long time and the numbers actually go down with problem size. In order to fix this, I ran five single-digit conversions before starting data gathering for my metrics.

The values of O() can actually be very accurate predictors of how an algorithm can perform. I was not expecting such a close correlation to curve fitting for these outputs.

Java already has an existing Stack class that could have been used in this implementation. Also, the underlying implementation allows the computing machine to be the limiting factor in how tall the stack can grow.

What I Might do differently Next Time

Write the program in C++. I had a really hard time figuring out how to do the input and output with Java and writing it in a language I know would have saved a lot of time.

I would compare my algorithm with the standard Java one to see if there is any performance difference between what I consider a clean and efficient implementation and the more general one offered by the language designers.

Justification for Design Decisions

There are four classes in this problem. The main entry point for the application (Project 0) is used to outline the solution algorithm without actually implementing the details. This provides a clean separation between the specifics of the project assignment and the other classes that could possibly be reused in other problems.

The BaseConverter class is the algorithm that is used to convert between decimal and binary. Having this as a separate class allows other base conversions to be added later.

The IntegerStack class is the core of this data structure assignment. For the use of this problem, I made it an integer stack. It could have been a more general stack that held any type of object.

The RuntimeMetric class is specific to the metrics that were being collected for this project. It could be extended in the future with other constructors and methods that allow different types of metrics and reports.

Issues of Efficiency

A stack is a straightforward way to store the results of these calculations. The access time to the most recent remainder on the stack is O(1), so the overall problem solution is driven by the problem itself, not the data structure.

One way to improve this algorithm's runtime is not to do two separate actions to store and retrieve the remainders of the division. Because the number of binary digits is limited to 32, the numbers could just be stored in an array, starting at the end. Then the string conversion would be a single step instead of having to pop the stack. This makes the solution more efficient at the expense of being less general.