605.620: Algorithms for Bioinformatics

William McElhenney

wmcelhe1

Project 1 Analysis

Due: September 24, 2019

Completed: September 25, 2019

Project 1 Analysis

Analysis of Ordinary Matrix Multiplication

The ordinary matrix multiplication (OMM) algorithm can be implemented as a set of three nested loops to iterate through the matrix and add the correct products to an initialized output matrix. For a square matrix of dimension *n*, the time complexity is $O(n^3)$ as we need iterate n^3 times to hit all the values to add to the output. This also means that we must make n^3 multiplications (meaning line 81 will execute exactly n^3 times for a given matrix).

An implementation for OMM can be seen in *matrix_tools.py* at lines 57 – 83. As can be seen, it is as I described it above, a set of three nested loops all going from 0 to *n*.

Analysis of Strassen Matrix Multiplication

Strassen's algorithm cleverly implements a recursive matrix multiplication algorithm by splitting matrices into quarters, defining specific addition/subtraction operations on those partitions, and then recursively multiplying the sums/differences from those operations. The class text states that this algorithm has $\Theta(n^{lg\ 7})$ time complexity.

This algorithm breaks the matrices down until it gets to a single value for each (a 1 by 1 matrix so to speak), at which point it just uses regular multiplication (technically OMM in my implementation as the matrix will still be an iterable). This further means that for the most basic valid matrix (2 by 2) this matrix will only need 7 multiplications, which are defined by the algorithm (as opposed to the eight from OMM and plain recursive matrix multiplication).

We can then use the case above to determine that for any square matrix Strassen's algorithm needs exactly $\frac{n^2}{4} * 7$ multiplications, or the number of possible 2 by 2 sized quadrants in an *n* by *n* matrix times 7, which is quite the savings when we consider an 8 by 8 matrix (512 for OMM vs 112 for Strassen's).

What I Learned

Strassen's is an interesting example of what we can do to reduce the execution times of our programs. By carefully examining the issue and deriving clever solutions we can greatly reduce the computation times of our algorithms.

What Would I Do Differently Next Time

I definitely would start earlier. I have been having some trouble getting my schedule situated this semester and wasn't able to get the work done in the time I had.

Design Decisions

The driver for the program is in the root of the project folder and is called *project1.py*. All this script does is coordinate between the rest of the code in the *./code* directory (that is initiate the read-in, run the multiplications on the data, and write the data out). This script also gives status messages (lets us know when a multiplication completes) and returns encountered exceptions to the user.

The functions for all basic matrix operations (including OMM) are located in *matrix_tools.py* in order to keep down copy-paste type code that happens between similarly acting functions.

All file I/O functions are in *fileIO.py*. The *read()* function makes sure through exceptions that no improperly formatted data is allowed in, and the *write()* function tries to keep the output pretty.

Strassen's functions are stored together in *strassen.py*. The main *strassen()* function is laid out much like the outline in the book on page 79 with steps 1 – 3 broken out into their own separate functions. Copious comments are included to help keep index mismatches from happening and to keep it readable (the multiplications are preassigned by Strassen, so this ends up looking like 'black box' code).

For the sake of ease, I also included two bash scripts to run all the inputs. The first of these, *run_all.sh*, runs all the inputs in the *./inputs* folder (including several test data that are not intended to run properly). The second, *run_valid.sh*, runs only the input provided by class (*classinput.dat*) and my input (*myinput.dat*). *myinput.dat* consists of matrices of all the same values as this produces the cubes of those values in the output, which makes for nice verification that both algorithms are working as intended.

I would have like for my program to be able to end more gracefully, but it mostly just tells the user that it cannot complete if it runs into an error. The only errors it should run into are with improperly formatted data or incorrect paths. I would like to be able to just pick up with the next matrix in the input file, but I have no way of knowing where to start the input again (guess we could see if there are any lines of length 1 before some longer lines and try that).

Efficiency

As shown in its analysis, Strassen's is much more efficient computationally than OMM at the cost of memory usage, programming R&D, and programming time.