Danny Shaw 110980476

CMSC433 Project 3 Design Document

Task Breakdown

For task breakdown, I break the grid up into n different pieces, where n corresponds to the number of available processes as determined from Runtime.getRuntime().availableProcessors. If n is odd, I use n-1 processes. I decided to use the Fork Join framework for my implementation, and what my project does is create more fork join RecursiveAction threads if the area assigned to the thread is greater than (totalArea of grid) / numProcessors + 1. If more threads need to be created, I split the board vertically in half and then horizontally into (n/2) pieces if numProcesses is even or (n-1)/2 pieces if numProcesses is odd. This will split the grid into n equal sized chunks, with each thread never overlapping as per the design of my program. Because the threads' areas never intersect, my implementation requires absolutely no synchronization between the filled checkers because it is physically impossible for two threads to occupy the same place at the same time. Each thread then individually runs essentially the single threaded algorithm for finding matches and the solutions of all the threads are aggregated into a shared, thread-safe ArrayList which is returned as the final solution.

Testing Plan

Most of my testing is from manual inspection of the scores that I received from tests that I created. I create grids of size 300x300, 500x500, 1000x1000, and 5000x5000 and test these against various times by making slight modifications to the variables that I alter in my implementation such as number of threads to create, which dictionary order to use, etc. I also have a class that creates dictionary of a desired size (the grid creation and dictionary creation is located in GridDictCreator.java) to test dictionaries of very large size. The tests are located in PrivateTests.java and can be run as standard JUnit tests.