Matthaus Litteken
CS 543 Homework #1
January 27, 2012

1 Totally Nuts Writeup

1.1 Instructions to Run

This was written using ruby. It uses features specific to ruby 1.9.x and was tested on ruby 1.9.2 and 1.9.3. To run, execute the following commands from the project root. (ruby may need to be replaced by ruby19 on debian based systems).

```
export RUBYLIB=${RUBYLIB}:./lib
ruby hw1.rb
```

1.2 What I Did & How I Did It

I wrote my Totally Nuts solver in ruby. I basically followed the simple recursive method we had discussed in class to efficiently prune the large search tree. For each of the cogs, it is used as the center piece and the remaining pieces are tested in each rotation to see if they fit against the center. If they match, the puzzle as assembled so far is passed into the same function to see if any of the remaining cogs, in some rotation fit with the puzzle. If the puzzle ever reaches size 6 and the last cog is a match, it is printed as a solution and added to the results set.

I modeled the puzzle as an array of 7 arrays of 6 integers. Ruby 1.9.3 has a handy rotate method that pushes the first element of the array onto the end. I used that to rotate the individual cogs for various combinations. The actual assembly of the puzzle was all handled in one function, that switched on the current puzzle size. Each puzzle size corresponded to particular elements in the puzzle array matching against the current cog.

The main search method iterated over each of the cogs and used it as the first piece of the puzzle. It then called down into the assemble_puzzle method, which tried all of the rotations of the remaining pieces to see if they fit. If a fit were found, it was passed into the same assemble_puzzle method until it was solved or stopped fitting. In that way bad paths were never explored very far and the solutions were found fairly rapidly. Solutions were printed out and added to a results array that was an instance variable for the class and easily accessible to the calling program.

1.3 Problems Along the Way

I ran into a few interesting problems mostly relating to how objects are handled in ruby.

The first was how to handle outer iterations continuing with the puzzle objects. The push operator (¡¡) operates on the array being pushed upon, so I was experiencing side effects of that behavior manifesting in having the same cog used more than once in a solved puzzle. The obvious solution and the one I employed was to copy the array before calling the recursive function. That way, the recursion could proceed on a copy, and the unchanged puzzle object could be passed along through the iterations.

Another problem I ran into was that I was using the difference operator to remove the current cog from the remaining cogs. However this caused problems if there were multiple cogs with the same order. The difference operation would remove all of them instead of just one of them. Because of this I changed the difference operation to a delete operation of the first index of the desired cog.

The last problem I fixed was one of ordering. The last few solutions had the first cog rotated, which shouldn't have been happening. After digging in, I realized that when I rotate the remaining cogs, they are also rotating the cogs in the results collection, because ruby was created reference copies of the original cog. I solved this by using Marshal to explicitly make deep copies for the results array.

1.4 How It Worked Out

The solution I coded up worked out very well. It found 15 solutions to the posted problem (there are more if symmetry is ignored). It found those solutions in 300ms, even when using an interpreted language like ruby. Ruby was a little tricky in certain aspects, like making shallow copies of what would usually be a primitive, like integers. But once those issues were worked around it was very nice.

Using ruby also meant that I was able to use rspec to do some basic unit testing to make sure that the puzzle was being parsed correctly and that it got more than one solution, one solution or no solutions for puzzles that have more than one solution, exactly one solution or no solution.