CSC 440 Assignment 3: Rubik's Cube Solver

Out: Thursday, February 26nd

Due: Friday, March 8nd, by 11:59PM

Introduction

In this assignment, you will develop an algorithms for solving the 2Ö2Ö2 Rubik's Cube, known as the Pocket Cube. Call a configuration of the cube "k levels from the solved position" if it can reach the solved configuration in exactly k twists, but cannot reach the solved configuration in any fewer twists. The rubik directory in the problem set package contains the Rubik's Cube library, as well as a graphical user interface to visualize your algorithm. We will solve the Rubik's Cube puzzle by finding the shortest path between two configurations (the start and goal) using BFS. A BFS that goes as deep as 14 levels will require a great deal of memory. However, the number of nodes at depth 7 is much smaller than half the total number of nodes. So, we can do a two-way BFS, starting from each end (the starting configuration and the solution) and meet in the middle.

You will be provided with a zip file on Sakai, rubik.zip, which contains a test framework, a library that gives you a representation of a Rubik's cube, and even a tool for visualizing a cube (however, this tool may not work properly in all Python versions, but you do not need the visualization tool to do the assignment).

Write a function `shortest_path(start, end)` in `solver.py` which is provided to you on Sakai. This function takes two positions, and returns a list of moves that is a shortest path between the two positions.

For this assignment, your solution must be in Python.

Your solution should be simple enough that it works in both Python 2.7 and 3.x We will evaluate your solutions in Python 2.7.

Pair Programming

You will work with a partner for this entire assignment. Please refer to the pair programming policy in the syllabus.

Lateness

Submissions will not be accepted after the due date, except with the intervention of the Dean's Office in the case of serious medical, family, or other personal crises.

Grading Rubric

Your grade will be based on three components:

- Functional Correctness (25%)
- Design and Representation (50%)
- Invariant Documentation (25%)

Design and Representation is our evaluation of the structure of your program, the choice of representations (how do you store moves? how do you efficiently search a graph?), and the use of appropriate (not excessive) comments. We have given you some obvious hints in the file `rubik.py`

Invariant Documentation is to force you to reason about the running time of the algorithm, as well as its correctness. Whenever you have a loop in the body of your algorithm, you should state whatever invariants hold.

- Initialization: how is the problem set up
- Maintenance: how do I know I'm making progress?
- Termination: how do I know I'm done
- Usually, these should all be closely related.