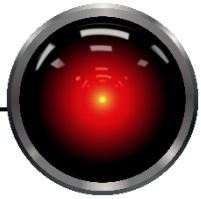
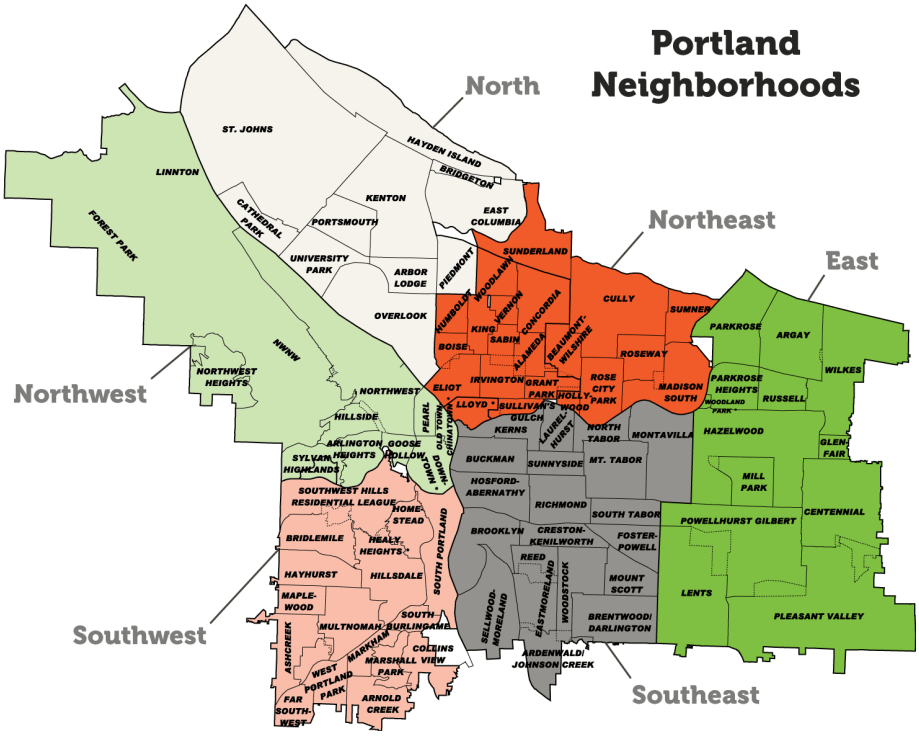


A3

Part I: 20 points each (Part II will be released later)

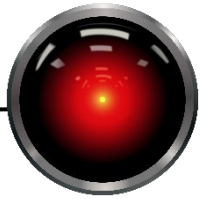


Portland has the following neighborhood regions: North (N), Northwest (NW) , Southwest (SW), Southeast (SE), East (E), and Northeast (NE) that we want to color with a 3 color map with colors R, G, and B.

Questions 1-4 are related to the Portland map and are to be considered independently. For algorithm simulation questions, you must show the steps of the algorithm, not just the final result.

1. Suppose we assign NW=R and E=B. If we wanted to make all arcs consistent, we could use the AC3 algorithm. What would our initial queue be?
2. Simulate the forward checking algorithm, showing the result of assigning B to E (east neighborhood).
3. Simulate the maintaining arc consistency algorithm, showing the result of assigning B to E.
4. Show the conflict sets that arise when forward checking is used and we assign G to NW followed by R to SE.

5. Consider a problem with three variables X, Y, and Z all with a domain of {False, True}. Show how the relationship X xor Y = Z can be modeled as a set of binary constraints between X, Y, Z, and an encapsulated variable.



Part II – Sudoku (120 points)

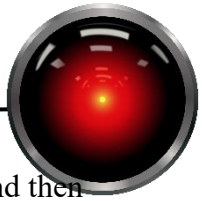
The code package for this assignment on Blackboard contains a partially implemented constraint satisfaction problem for solving Sudoku puzzles. You will need to implement functionality in three modules:

- driver – Solves Sudoku puzzles. Two puzzles are provided for you. When solving these puzzles, you should first run the AC3 algorithm to propagate any constraints from the values that have been set. Print out the puzzle before starting inference and then after inference is completed. If the puzzle is not solved, a backtracking search should be run. The easier of the two problems can be solved entirely by inference without any searching.
- backtrack – Implement a backtracking search. Variable restriction should be via the minimum remaining value (MRV) heuristic and inferences should be via the maintaining arc consistency (MAC) algorithm. Your algorithm should return a dictionary of variable assignments or None if no solution exists and the number of variables that were assigned by the backtracking search not counting any assignments due to inference.
- constraint_prop – implement the AC3 algorithm

Several modules are provided in package csp_lib. You do not need to modify code in these modules, but you will need to read them in order to understand how to interact with the Sudoku CSP representation.
The sudoku module implements the class Sudoku that is derived from a CSP class (both in module csp_lib). The code and comments explain the data structures that are used and contain a sample instantiations of two Sudoku puzzles. In general, the CSP class methods will be very useful for this problem. Some of the ones that you might want to pay attention to are (nonexhaustive list):

- goal_test – Tests whether or not the problem is sovled.
- suppose – Given a variable and value, restrict the domain to the value and return a list of removed values; e.g., if we assign val3 to var, we would get back [(var,val1), (var,val2), (var, val4), …] and the csp object would be modified to have val3 the sole member of the domain. Note that this is different from assign which does not return information that will let you restore the state later.
- prune – Given a var name and a value, remove it from the domain.
- infer_assigment – Return variables that can be currently assigned based on their domains.
- restore – Given a removal list (see prune), restore the values to domains

As with your last assignment, setting breakpoints and stepping through functions can be very helpful for understanding. Module csp.backtrack_util provides a set of utility functions for backtracking. You will not need most of them, but you will need one of the variable ordering routines and the maintaining arc consistency (mac) routine. If you are confused about how the AC3 interface works, you might want to pay attention to mac as it calls AC3 with a smaller queue.



Your driver program should create both the easy and hard sudoku problems and then solve them.

Output: Your code should show for each puzzle:

- Initial sudoku state
- The state of the puzzle after running AC3.
- If the puzzle is not in a goal state (the Sudoku class provides a goal test that can be used on the instances current domains), then run a backtracking search and show the solution.

Part II Submission:

- Modules: backtrack, constraint_prop, and driver (with affidavit)
- Text file with output from your program run. Use a text file as opposed to a screen shot.