CSCE 625
Programing Assignment #3
due: Tues, October 20 (by start of class)

Objective

The goal of this assignment is to implement and solve two example Constraint Satisfaction Problems (CSPs) using back-tracking, and to implement and evaluate the effect of the MRV heuristic.

Problem 1 (the 'jobs' puzzle):

- There are four people: Roberta, Thelma, Steve, and Pete.
- Among them, they hold eight different jobs.
- Each holds exactly two jobs.
- The jobs are chef, guard, nurse, clerk, police officer (gender not implied), teacher, actor, and boxer.
- The job of nurse is held by a male.
- The husband of the chef is the clerk.
- Roberta is not a boxer.
- Pete has no education past the ninth grade.
- Roberta, the chef, and the police officer went golfing together.

Who holds which jobs?

In case it is ambiguous...

- the only implicit gender restrictions are that the *actor* has to be male (all other jobs can be held by male or female)
- since the husband of the chef is the clerk, you can assume this implies the chef is female
- assume that an education beyond the ninth grade is required for the following jobs: nurse, teacher, police officer
- in the last constraint, read it as 3 distinct people playing golf (not that Roberta is the chef)

Problems like this can easily be encoded as a CSP. You will have to choose what your variables and domains are. Then you will have to encode the constraints described above in a consistency_check() function that, given a partial variable assignment, checks whether any constraints are violated (more details below).

Problem 2 (the 'houses' puzzle):

Variants of this problem are also known as the 'zebra' puzzle or Einstein's puzzle:

- The Englishman lives in the red house
- The Spaniard owns the dog
- The Norwegian lives in the first house on the left
- The green house is immediately to the right of the ivory house
- The man who eats Hershey bars lives in the house next to the man with the fox
- Kits Kats are eaten in the yellow house
- The Norwegian lives next to the blue house
- The Smarties eater owns snails
- The Snickers eater drinks orange juice
- The Ukranian drinks tea
- The Japanese person eats Milky Ways
- Kit Kats are eaten in a house next to the house where the horse is kept
- Coffee is drunk in the green house
- Milk is drunk in the middle house
- A zebra is kept in one of the houses.
- One of the persons prefers water.

This problem is more complex than the jobs puzzle. You will have to carefully choose the representation by deciding what you want to use as variables and their domains. Also, you will have to make sure your consistency_check() function adequately captures all the constraints mentioned above.

Implementation

You can use any programming language you like. Your code should print out the first solution it finds, by displaying the bindings for each variable. (Since the state spaces of these problems are relatively small, you might also continue the search to try to find other solutions, or exhaustively search the space to find them all. But this is optional.)

Your implementation should follow the basic 'backtracking' procedure shown in the textbook in Figure 6.5 (ignore the lines referring to 'inference', which are irrelevant for this project). This algorithm is general and should work for arbitrary CSP problems. The only domain- or problem-specific part is the function that checks for consistency. Assume you will be calling a function with a signature like this:

```
bool consistency_check(PartialAssignment)
```

The idea is that you pass it an assignment (a list/array/hash of variable-to-value bindings; remember, a partial assignment might not have values for some variables yet, and could even

be empty), and it goes through the constraints associated with a particular problem and (for those constraints involving variables that are bound to values), determines whether any of these is violated. This is all the CSP() search routine needs to know, and it will use the output of this function to search the state space systematically to look for solutions (which are complete and consistent assignments). Note that you will have two different consistency_check() functions, one for each puzzle, e.g. consistency_jobs() and consistency_houses().

It is relatively easy to find solutions to the jobs puzzle. The houses puzzle is harder and has a larger state space to search. The backtracking procedure can still be used to search this space, but it is slower. To improve the speed (i.e. reduce the number of nodes explored), you will implement the **MRV heuristic**. This can be done by intelligently adapting choice of unassigned variable to process in each iteration by choosing the one that has the least remaining values. You will have to figure out how to evaluate this in your code, but a suggestion is to loop through all the variables and their domains, add them one at a time to the current assignment in a trial fashion, call the consistency function, and keep track of how many values are still feasible for each unassigned variable. While this might seem inefficient, in the end it is a win because it cuts down the total number of nodes that need to be searched. You should run your code with and without the MRV heuristic on each of the puzzles and *determine how much MRV improves the efficiency of search in terms of reduction in number of states explored*.

What to Turn in:

- You will submit your code for testing using the web-based CSCE *turnin* facility, which is described here: https://wiki.cse.tamu.edu/index.php/Turning_in_Assignments_on_CSNet (you might have to be inside the TAMU firewall to access this)
- Include a brief document that describes how to compile and run your code.
- Include a short write-up that describes your choice of representation, your consistency function, and the solution (variable assignment) for each problem. Also give a count of the number of state searched with and without the MRV heuristic.