Kevin Hsieh
2078611
COSC4368
Dr. Eick

Problem Set 1 Task 2:

I implemented a recursive backtracking search for my CSP solver, it utilizes minimum-remaining-values(MRV) heuristic. If some variable X has no legal values left, the MRV heuristic will select X and failure will be detected immediately, avoiding pointless searches through other variables. It also does a generic form of inference called forward checking. Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it: for each unassigned variable Y that is connected to X by a constraint, delete from Y's domain any vlaue that is inconsistent with the value chosen for X.

```
function RECURSIVE-BACKTRACKING (csp) //returns a solution or failure
        If all variables assigned
                PRINT Value of each Variable
                RETURN or EXIT (RETURN for more solutions)
                                        (EXIT for only one solution)
V := PickUnassignedVariable()
Variable[Level] := V
Assigned[V] := TRUE
for d := each member of Domain(V) (the domain values of V)
        Value[V] := d
        for each constraint C such that V is a variable of C and all other variables of C are assigned:
                IF C is not satisfied by the set of current
                        assignments: BREAK;
                ELSE BackTrack(Level+1)          ▷recurse
        return

function ForwardChecking(csp) //returns a new domain for each var
        If all variables are assigned
                PRINT Value of each Variable
                RETURN or EXIT (RETURN for more solutions) (EXIT for only one solution)
V := PickAnUnassignedVariable()
Variable[Level] := V
Assigned[V] := TRUE
for d := each member of CurDom(V)        ▷current domain
        Value[V] := d
        DWOoccured:= False          ▷DWO (Domain wipe out)
        for each constraint C over V that has one unassigned variable in its scope (say X).
                if(FCCheck(C,X) == DWO) /* X domain becomes empty*/
                DWOoccurred:= True /* no point to continue*/
                break
        if(not DWOoccured) /*all constraints were ok*/
                FC(Level+1)
        RestoreAllValuesPrunedByFCCheck()
return;
```

In the recursive-backtracking function(BT) we first do a check to see if all variables have been assigned a value, if it does, we return or exit (if only one solution). Next we choose an unassigned variable and we go into a for loop, and now check for each value in the domain against the constraint, if constraint is not satisfied we break, and recursively calls the BT. In forward checking, we first do a check to see if all variables have been assigned a value, if it does, we return or exit (if only one solution). Next we choose unassigned variable, whenever a variable X is assigned, the forward-checking process establishes arc consistency for it: for each unassigned
variable Y that is connected to X by a constraint, delete from Y's domain any value
that is inconsistent with the value chosen for X.

Strategies that I implemented: MRV heuristic and forward checking. These propagations helps to reduce the runtime of my program by augmenting the backtracking search. It suggests that we pick the variable with the fewest remaining values in its domain to assign next. By implementing forward checking, instead of performing arc consistency to the instantiated variables, it performs restricted form of arc consistency to the not yet instantiated variables.

I did not conduct any mathematical pre-analysis to eliminated variables.

This program also does not take advantage of the hierarchical structure of the three CSP problems, but instead runs them each separately.

This program is developed generically so future CSP problems having the same structure can be implemented. This was done by wrapping everything in the form of classes, subclasses, and taking advantage of abstract methods and inheritance.

Appendix
ProblemA output
([('A', 35), ('B', 1), ('C', 28), ('D', 27), ('E', 1), ('F', 5)])
{'E': 1, 'A': 35, 'F': 5, 'D': 27, 'B': 1, 'C': 28}
nva= 50
ProblemB output
([('A', 35), ('B', 23), ('C', 6), ('D', 27), ('E', 1), ('F', 5), ('G', 5), ('H', 10), ('I', 7), ('J', 8)])
nva= 61
Problem C
([('A', 10), ('B', 1), ('C', 2), ('D', 28), ('E', 3), ('F', 4), ('G', 5), ('H', 6), ('I', 7), ('J', 8), ('K', 15), ('L', 20), ('M', 100), ('N', 30), ('O', 50)])
nva= 15228

How to run Task2.py:

In terminal enter
python3 Task2.py
Follow prompts and only enter a or b or c as inputs