CS 4613 **E. K. Wong**
Project 2: Futoshiki **Spring 2020**

Total number of points = 100.

Project Description: Design and implement a program to solve 5 × 5 Futoshiki puzzles. The rules of the game are:

- The game board consists of 5 × 5 cells. Some of the cells already have numbers (1 to 5) assigned to them and there are inequality signs (< or >) between some of the cells (See Figure 1.)
- The goal is to find assignments (1 to 5) for the empty cells such that the inequality relationships between all pairs of adjacent cells with an inequality sign between them are satisfied. In addition, a digit can only appear once in every row and column; that is, the digits in every row and column must be different from each other (See Figure 2.)

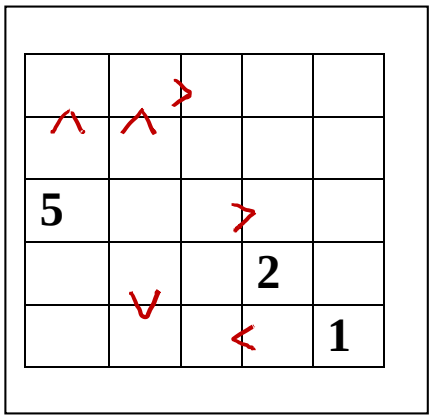

Figure 1. Initial game board

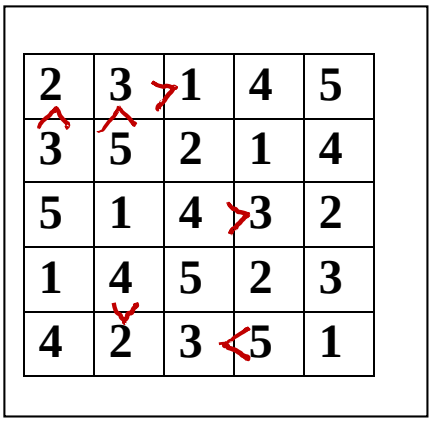

Figure 2. Solution

As the first step in your program, use *Forward Checking* to reduce the domain of empty cells, based on the values of cells that already have a number. If an empty cell has only one value left in its domain after domain reduction, you can repeat Forward Checking on the cell's neighbors, and so on. If any empty cell has an empty domain after applying Forward Checking, then the puzzle does not have a solution and the program can stop here. Next, use the *Backtracking Algorithm* for *CSPs* (in Figure 5 on page 3) to solve the puzzle. Implement the function *SELECT-UNASSIGNED-VARIABLE* in the algorithm by using the *most constrained variable* (or *minimum remaining values*) heuristic, and in case of a tie, use the *most constraining variable* (or *degree*) heuristic as tie breaker. If there are more than one variables left after applying the *most constraining variable* heuristic, you can arbitrarily choose a variable. There is no need to implement the *least constraining value* heuristic in the *ORDER-DOMAIN-VALUES* function; instead, simply order the domain values in increasing order (from 1 to 5.) To make it easier for you to implement the program, you do not have to implement the INFERENCE function inside the *Backtracking Algorithm.*

Your program will read in values from an input text file and produce an output text file that contains the solution. The input file contains 16 rows (or lines) of integers. The first five rows contain the initial cell values of the game board, with each row containing five integers, ranging from 0 to 5. Digit "0" indicates a blank cell. This is followed by a blank line. The next five rows contain the inequalities between horizontally-adjacent cells. A value of 0 indicates no inequality between two cells. This is followed by a blank line. The next four rows contain the inequalities between

vertically-adjacent cells. Again, a value of 0 indicates no inequality between two cells. The input file for the initial game board in Figure 1 is as shown in Figure 3 below. The output file contains five rows (or lines) of integers, representing the solution. Each row contains five integers, ranging from 1 to 5, separated by blanks. The output file (solution) for the initial game board in Figure 1 is as shown in Figure 4 below.

Testing your program: I will provide three input test files on NYU Classes for you to test your program.

Recommended languages: Python, C++/C and Java. If you would like to use a different language, send me an email first.

Submit on NYU Classes by due date:

1. A text file that contains the source code. Put comments in your source code to make it easier for someone else to read your program. Points will be taken off if you do not have comments in your source code.
2. The output text files generated by your program. Name your output files *Output1.txt*, *Output2.txt* and *Output3.txt*.
3. A PDF file that contains instructions on how to run your program. If your program requires compilation, instructions on how to compile your program should also be provided. Also, copy and paste the output text files and your source code onto the PDF file (to make it easier for us to grade your project.) This is in addition to the source code file and output files that you have to submit separately (as described in 1 and 2 above.)

```
0 0 0 0 0
0 0 0 0 0
5 0 0 0 0
0 0 0 2 0
0 0 0 0 1

0 > 0 0
0 0 0 0
0 0 > 0
0 0 0 0
0 0 < 0

^ ^ 0 0 0
0 0 0 0 0
0 0 0 0 0
0 v 0 0 0
```

Figure 3. Input file for the initial game board in Figure 1. Digit 0 indicates a blank cell. > and < are the *greater than* and *less than* characters on the keyboard. ^ is upper case 6 and v is the lower case letter v on the keyboard.

2 3 1 4 5
3 5 2 1 4
5 1 4 3 2
1 4 5 2 3
4 2 3 5 1

Implement forward checking before backtracking

Figure 4. Output file containing the solution for the initial game board in Figure 1.

```
function BACKTRACKING-SEARCH(csp) returns solution or failure
    return BACKTRACK({ }, CSP )
function BACKTRACK(assignment, csp) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
       if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, value)
            if inferences != failure then
                 add inferences to assignment
                 result ← BACKTRACK(assignment, csp)
                 if result != failure then
                    return result
            remove {var=value} and inferences from assignment
  return failure
```

Figure 5. The Backtracking Algorithm for CSPs.