CS 4613 **E. K. Wong**
Project 2: Futoshiki **Spring 2021**

Total number of points = 100.

Project Description: Design and implement a program to solve 6 × 6 Futoshiki puzzles. The rules of the game are as follows:

- The game board consists of 6 × 6 cells. Some of the cells already have numbers (1 to 6) assigned to them and there are inequality signs (< or >) placed horizontally or vertically between some of the cells (See Figure 1.)
- The goal is to find assignments (1 to 6) 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.)

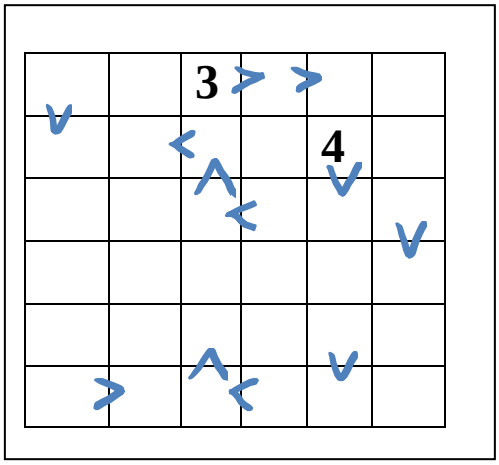

Figure 1. Initial game board

6	5	3	2	1	4
5	1	2	3	4	6
1	6	4	5	2	3
3	4	6	1	5	2
2	3	1	4	6	5
4	2	5	6	3	1

Figure 2. Solution

As a first step in your program, go through the 6 × 6 grid and for each cell with a number, use *Forward Checking* at that cell location to reduce the domain of its unassigned neighbors[1]. Next, use the *Backtracking Algorithm* in Figure 5 to solve the puzzle. Implement the function *SELECT-UNASSIGNED-VARIABLE* in the algorithm by using the *minimum remaining values* heuristic and, in case of a tie, use the *degree* heuristic as tie breaker. If there are more than one variables left after applying the *degree* heuristic, the algorithm can arbitrarily choose the next 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 6.) Also, you do not need 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 format of the input file (representing the initial game board in Figure 1) is as shown in Figure 3 below. The first six rows contain the initial cell values of the game board, with each row containing six integers, ranging from 0 to 6. Digit "0" indicates a blank cell. This is followed by a blank line. The next six rows contain the inequalities between horizontally-adjacent

[1] Two or more cells are neighbors if they belong to the same constraint. They do not have to be physical neighbors. Apply *forward checking* (one time) only at cell locations that initially have a number assigned to them.

cells. A value of 0 indicates no inequality between two cells. This is followed by a blank line. The next six rows contain the inequalities between vertically-adjacent cells. Again, a value of 0 indicates no inequality between two cells. The format of the output file (representing the solution in Figure 2 above) is shown in Figure 4 below. The output file contains six rows of integers, with each row containing six integers ranging from 1 to 6, separated by blanks.

Team: You can work on the project alone or you can work in a team of two. You can discuss with your classmates on how to do the project but every team is expected to write their own code and submit their own program and report.

Testing your program: Three input test files will be provided 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 the 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 are required to submit separately (as described in 1 and 2 above.)

```
0 0 3 0 0 0
0 0 0 0 4 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 0
0 0 < 0 0
0 0 0 0 0
0 0 0 0 0
> 0 < 0 0

v 0 0 0 0 0
0 0 ^ 0 v 0
0 0 0 0 0 v
0 0 0 0 0 0
0 0 ^ 0 v 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 the upper case 6 character and v is the lower case v on the keyboard.

```
6 5 3 2 1 4
5 1 2 3 4 6
1 6 4 5 2 3
3 4 6 1 5 2
2 3 1 4 6 5
4 2 5 6 3 1
```

Figure 4. Output file containing the solution.

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

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

Figure 5. The Backtracking Algorithm for CSPs.