CS 6613 **E. K. Wong**
Project 2: Kropki Sudoku **Fall 2024**

Total number of points = 100. Optional extra credits = 20. You can work on the project by yourself or you can work in a team of two.

Project Description*:* Implement the *Backtracking Algorithm* in Figure 3 below to solve the Kropki Sudoku puzzle. The rules of the game are:

- The game board consists of 9×9 cells divided into 3×3 non-overlapping blocks as shown below. Some of the cells already have digits (1 to 9) assigned to them initially.
- The goal is to find assignments (1 to 9) for the empty cells so that every row, every column, and every 3x3 block contains the digits from 1 to 9 exactly once.
- A white dot between two adjacent cells requires that the value in one cell is exactly 1 greater than the value of the other cell. This could be satisfied in either direction; e.g., "3 & 4" or "4 & 3" will both satisfy the requirement. A black dot between two adjacent cells requires that the value in one cell is exactly double the value of the other cell. This also could be satisfied in either direction; e.g., "3 & 6" or "6 & 3." If there is no dot between two adjacent cells, then neither condition needs to be fulfilled between the two cells.

Figure 2 shows the solution for the puzzle in Figure 1.

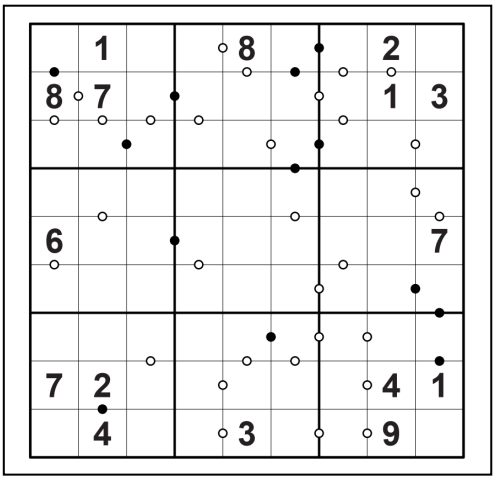

Figure 1. Initial game board

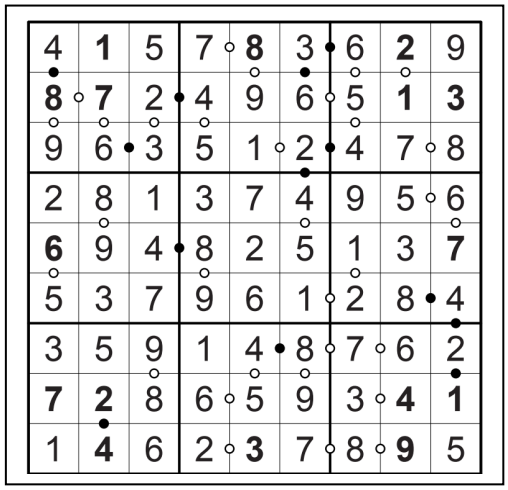

Figure 2. Solution

In the Backtracking Algorithm, use the *minimum remaining value* and *degree* heuristics to implement the *SELECT-UNASSIGNED-VARIABLE* function. For the *ORDER-DOMAIN-VALUES* function, simply order the domain values in increasing order from 1 to 9 instead of using heuristics. You can skip the INFERENCE function inside the Backtracking Algorithm in your implementation. (Reminder: when implementing the *minimum remaining value* and *degree heuristics*, two or more variables are neighbors if they share one or more common constraints.)

Extra Credits (20 points): This part is optional. Implement the INFERENCE function with *Forward Checking* in your implementation of the Backtracking Algorithm. If you do the extra credits part, indicate this in your PDF report.

CS 6613 **E. K. Wong**
Project 2: Kropki Sudoku **Fall 2024**

Input and output files: Your program will read in the initial game board values from an input text file and then produce an output text file that contains the solution. The format of the input file is as shown in Figure 4 below. The first 9 rows contain the initial cell values of the game board, separated by blanks. The cell values range from 0 to 9, with 0 indicating a blank cell. This is followed by a blank line. The next 9 rows contain the locations of dots between *horizontally-adjacent* cells. Row #1 contains the locations of dots for horizontally-adjacent cells in row #1 of the game board, and similarly for the other rows. A value of 0 means there is no dot between two adjacent cells, a value of 1 means there is a white dot between two adjacent cells, and a value of 2 means there is a black dot between two adjacent cells. This is then followed by another blank line. The next 8 rows contain the locations of dots between *vertically-adjacent* cells. Row #1 contains the locations of dots between vertically-adjacent cells in the first and second rows of the game board, and similarly for the other rows. Again, a value of 0 means there is no dot between two adjacent cells, a value of 1 means there is a white dot between two adjacent cells, and a value of 2 means there is a black dot between two adjacent cells. The format of the output file is as shown in Figure 5 below. The output file contains 9 rows of integers ranging from 1 to 9, separated by blanks.

Testing your program: Three input test files will be provided on NYU Brightspace for you to test your program. A sample input file and a sample output file will also be provided.

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

What to submit: Submit the following files on Brightspace by the due date. If you work with a partner, only one partner needs to submit but put both partners' names on the PDF report and the source code file.

1. Your source code file. 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 report that contains the following:

    a. Instructions on how to run your program. If your program requires compilation, instructions on how to compile your program should also be provided.
    b. A description of your formulation of Kropki Sudoku as a constraint satisfaction problem. This includes the set of *variables* you have defined, the *domains* for the variables, the set of *constraints* you have set up, and any other additional information on your formulation.
    c. Also, copy and paste the output files and your source code onto the PDF report (to make it easier for us to grade your project.) This is in addition to the source code and output files that you have to submit separately, as described in (a) and (b) above.

```
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 3. The Backtracking Algorithm for CSPs.

```
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n

d d d d d d d d
d d d d d d d d
d d d d d d d d
d d d d d d d d
d d d d d d d d
d d d d d d d d
d d d d d d d d
d d d d d d d d
d d d d d d d d

d d d d d d d d d
d d d d d d d d d
d d d d d d d d d
d d d d d d d d d
d d d d d d d d d
d d d d d d d d d
d d d d d d d d d
d d d d d d d d d
```

Figure 4. Input file format: *n* is an integer between 0 and 9, *d* is an integer between 0 and 2. The digits are separated by blanks.

```
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
n n n n n n n n n
```

Figure 5. Output file format: *n* is an integer between 1 and 9. The digits are separated by blanks.