E04 Futoshiki Puzzle (Forward Checking)

Suixin Ou

School of Computer Science
Sun Yat-sen University

October 12, 2021

Task

Problem

- Futoshiki is a board-based puzzle game. It is playable on a square board having a given fixed size.
- The purpose of the game is to discover the digits hidden inside the board's cells; each cell is filled with a digit between 1 and the board's size. On each row and column each digit appears exactly once; therefore, when revealed, the digits of the board form a so-called Latin square.
- At the beginning of the game some digits might be revealed. The board might also contain some inequalities between the board cells; these inequalities must be respected and can be used as clues in order to discover the remaining hidden digits.
- Each puzzle is guaranteed to have a solution and only one.
- You can play this game online: http://www.futoshiki.org/.

Task

Input-output

- Input: a n x n matrix of initial state, and a list of inequal constraints.

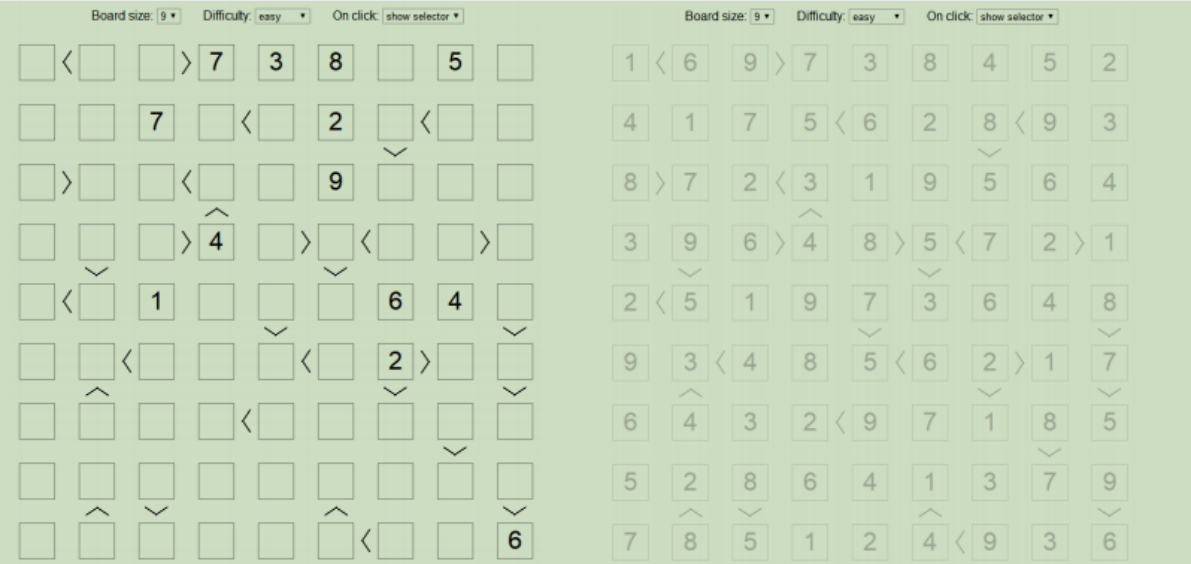

Figure 1: An Futoshiki Puzzle

- Output: the n x n matrix of the terminate state that satisfys all constraints (including inequal constraints, row and column constraints).

Task

Submission

pack your report `E04_YourNumber.pdf` and source code into zip file `E04_YourNumber.zip`, then send it to `ai_course2021@163.com`.

Solution

Algorithm procedure

```
FCCheck(C,x)
  // C is a constraint with all its variables already
  // assigned, except for variable x.
  for d := each member of CurDom[x]
     IF making x = d together with previous assignments
        to variables in scope C falsifies C
     THEN remove d from CurDom[x]
 IF CurDom[x] = {} then return DWO (Domain Wipe Out)
 ELSE return ok
```

Solution

Algorithm procedure

```
FC(Level) /*Forward Checking Algorithm */
   If all variables are assigned
        PRINT Value of each Variable
        RETURN or EXIT (RETURN for more solutions)
                       (EXIT for only one solution)
   V := PickAnUnassignedVariable()
   Assigned[V] := TRUE
   for d := each member of CurDom(V)
        Value[V] := d
        DWOoccured:= False
        for each constraint C over V such that
              a) C has only one unassigned variable X in its scope
           if(FCCheck(C,X) == DWO)     /* X domain becomes empty*/
                DWOoccurred:= True
                break /* stop checking constraints */
        if(not DWOoccured) /*all constraints were ok*/
           FC(Level+1)
        RestoreAllValuesPrunedByFCCheck()
   Assigned[V] := FALSE //undo since we have tried all of V's values
   return;
```

Solution

Read input

```
24          //初始地图
25          maps = {{0, 0, 0, 7, 3, 8, 0, 5, 0},
26                  {0, 0, 7, 0, 0, 2, 0, 0, 0},
27                  {0, 0, 0, 0, 0, 9, 0, 0, 0},
28                  {0, 0, 0, 4, 0, 0, 0, 0, 0},
29                  {0, 0, 1, 0, 0, 0, 6, 4, 0},
30                  {0, 0, 0, 0, 0, 0, 2, 0, 0},
31                  {0, 0, 0, 0, 0, 0, 0, 0, 0},
32                  {0, 0, 0, 0, 0, 0, 0, 0, 0},
33                  {0, 0, 0, 0, 0, 0, 0, 0, 6}};
```

```
37          //初始化行列约束
38          for (int i = 0; i < 9; i++) {
39              for (int j = 0; j < 9; j++) {
40                  if (maps[i][j] != 0) {
41                      Count_RowNumbers[i][maps[i][j]]++;
42                      Count_ColumnNumbers[j][maps[i][j]]++;
43                  }
44              }
45          }
46          //添加比较符号约束
47          addConstraints(0, 0, 0, 1);
48          addConstraints(0, 3, 0, 2);
49          addConstraints(1, 3, 1, 4);
50          addConstraints(1, 6, 1, 7);
51          addConstraints(2, 6, 1, 6);
```

Visualize output

```
124     //显示结果
125     void show() {
126         for (int i = 0; i < nRow; i++) {
127             for (int j = 0; j < nColumn; j++) {
128                 cout << maps[i][j] << " ";
129             }
130             cout << endl;
131         }
132         cout << "======================" << endl;
133     }
```

Solution

Check whether conditions are all satisfied. **You should finish a check function for the Forward Checking algorithm.**

```
85      //check函数检查当前位置是否可行，
86      //以下注释掉的内容是back tracking算法的check函数
87      //你们需要自行实现forward checking算法的check部分
88  >   bool check(int x, int y) {···
122     }
```

Search for correct solution.

```
134     //搜索流程，可以不用修改这部分
135 >   bool search(int x, int y) {···
183     }
```

Task

Some discussion

You are encouraged to explore(**not necessary**):

- Differences between back tracking and forward checking algorithm;
- Influences introduced by the case difficulty;
- Tradeoff between search expenses caused by the number of searched nodes and checking expenses caused by the constraint checking in every single node;

The End