School of Data and Computer Science, Sun Yat-sen University

E04 Futoshiki Puzzle (Forward Checking)

17341137 Zhenpeng

September 20, 2019

Contents

1 Futoshiki

Futoshiki is a board-based puzzle game, also known under the name Unequal. It is playable on a square board having a given fixed size (4×4 for example).

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/`.

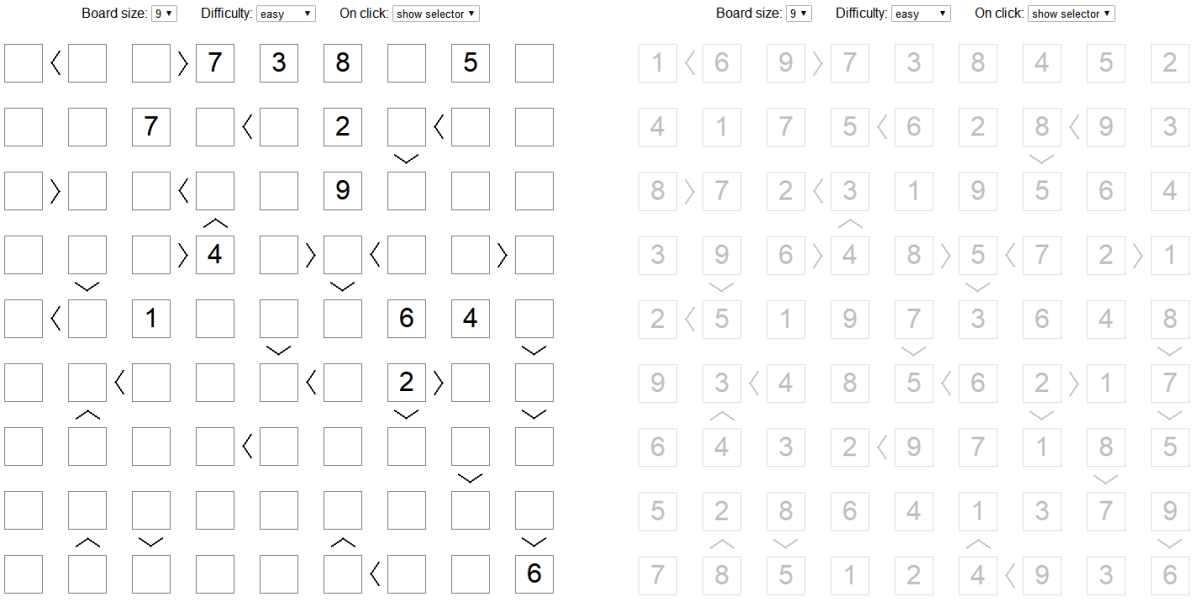

Figure 1: An Futoshiki Puzzle

2 Tasks

1. Please solve the above Futoshiki puzzle (Figure 1) with forward checking algorithm.
2. Write the related codes and take a screenshot of the running results in the file named E04_YourNumber.pdf, and send it to ai_201901@foxmail.com.

3 Codes

```
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;

const int SIZE = 9;

struct Do {
    int val; // value
    int row, col; // position
    int l, r, u, d; // [-1:<] | [0:\] | [1:>]
    bool curdom[SIZE]; // 1~SIZE's availablity
    bool assigned;
};

struct futoshiki {
    Do board[SIZE][SIZE];
    bool RowCheck(futoshiki* board, Do* m);
    bool ColCheck(futoshiki* board, Do* m);
    bool NeiCheck(futoshiki* board, Do* m);
    int CDcount(Do* m);
    bool Goal(futoshiki* board);
    bool FCCheck(futoshiki* board, int c, Do* m);
    void Copyboard(futoshiki* dest, const futoshiki* src);
    Do* heuristicpick(futoshiki* board);
    void propagete(futoshiki* board, Do* m);
};

bool FC(futoshiki* board, int level);
void propagate(futoshiki* board, Do* m);
void display(futoshiki* board);
```

```
32
33
34 int main() {
35     futoshiki FTSK;
36     futoshiki* ptr = &FTSK;
37     for (int i = 0;i<SIZE;i++) {
38         for (int j = 0;j<SIZE;j++) {
39             FTSK.board[i][j].val = 0;
40             FTSK.board[i][j].row = i;
41             FTSK.board[i][j].col = j;
42             FTSK.board[i][j].u = 0;
43             FTSK.board[i][j].d = 0;
44             FTSK.board[i][j].l = 0;
45             FTSK.board[i][j].r = 0;
46             FTSK.board[i][j].assigned = 0;
47             memset(FTSK.board[i][j].curdom, 0,
48                     sizeof(FTSK.board[i][j].curdom));
49         }
50     }
51     cout << "Input as a b < c d,
52             mind to make sure a <= c and b <= d !!" << endl;
53     while(1) {
54         int a, b, d, e;
55         char c;
56         cin >> a >> b >> c >> d >> e;
57         if (a == d) {
58             FTSK.board[a-1][b-1].r = (c == '>') ? 1 : -1;
59             FTSK.board[d-1][e-1].l = (c == '>') ? -1 : 1;
60         }
61         else {
62             FTSK.board[a-1][b-1].u = (c == '>') ? 1 : -1;
63             FTSK.board[d-1][e-1].d = (c == '>') ? -1 : 1;
64         }
```

```
65          if (a+b+d+e == 0) break;
66      }
67      cout << "Input Presets: " << endl;
68      while(1) {
69          int a, b, v;
70          cin >> a >> b >> v;
71          if (a+b+v == 0) break;
72          FTSK.board[a-1][b-1].val = v;
73          FTSK.board[a-1][b-1].assigned = 1;
74          propagate(ptr, &FTSK.board[a-1][b-1]);
75      }
76      display(ptr);
77      FC(ptr, 0);
78      return 0;
79  }
80
81  /* ################################################## */
82
83  bool RowCheck(futoshiki* board, Do* m) {
84      // return false: constraint falsified;
85      //          true: NO falsification.
86      int Row[SIZE];
87      for(int i = 0;i<SIZE;i++) {
88          Row[i] = board->board[m->row][i].val;
89      }
90      // Check Constraints
91      sort(Row, Row + SIZE);
92      for(int i = 0;i<SIZE-1;i++) {
93          if (!Row[i]) continue;
94          if (Row[i] == Row[i+1]) return false;
95      }
96      return true;
97  }
```

```
98
99  bool ColCheck(futoshiki* board, Do* m) {
100     // return false: constraint falsified;
101     //          true: NO falsification.
102     int Col[SIZE];
103     for(int i = 0;i<SIZE;i++) {
104         Col[i] = board->board[i][m->col].val;
105     }
106     // Check Constraints
107     sort(Col, Col + SIZE);
108     for(int i = 0;i<SIZE-1;i++) {
109         if (!Col[i]) continue;
110         if (Col[i] == Col[i+1]) return false;
111     }
112     return true;
113 }
114
115 bool NeiCheck(futoshiki* board, Do* m) {
116     int v = m->val;
117     // Check Constraints
118     // UP
119     if (m->u && board->board[m->row - 1][m->col].assigned) {
120         if (m->u == -1 && v > board->board[m->row - 1][m->col].val)
121             return false;
122         else if (m->u == 1 && v < board->board[m->row - 1][m->col].val)
123             return false;
124     }
125     // DOWN
126     if (m->d && board->board[m->row + 1][m->col].assigned) {
127         if (m->d == -1 && v > board->board[m->row + 1][m->col].val)
128             return false;
129         else if (m->d == 1 && v < board->board[m->row + 1][m->col].val)
130             return false;
```

```
131     }
132     // LEFT
133     if (m->l && board->board[m->row - 1][m->col].assigned) {
134         if (m->l == -1 && v > board->board[m->row][m->col - 1].val)
135             return false;
136         else if (m->l == 1 && v < board->board[m->row][m->col - 1].val)
137             return false;
138     }
139     // RIGHT
140     if (m->r && board->board[m->row - 1][m->col].assigned) {
141         if (m->r == -1 && v > board->board[m->row][m->col + 1].val)
142             return false;
143         else if (m->r == 1 && v < board->board[m->row][m->col + 1].val)
144             return false;
145     }
146     return true;
147 }
148
149 int CDcount(Do* m) {
150     int res = 0;
151     for (int i = 0;i<SIZE;i++) {
152         res += m->curdom[i];
153     }
154     return res;
155 }
156
157 bool CheckCons(futoshiki* board, int c, Do* m) {
158     int R = m->row, C = m->col, tot = 0;
159     // c == 0 >>> row
160     if (c == 0) {
161         for (int i = 0;i<SIZE;i++) {
162             tot += board->board[R][i].assigned;
163         }
```

```
164         return (tot == SIZE-1);
165     }
166     // c == 1 >>> col
167     else if (c == 0) {
168         for (int i = 0;i<SIZE;i++) {
169             tot += board->board[i][C].assigned;
170         }
171         return (tot == SIZE-1);
172     }
173     // c == 2 >>> neighbour
174     else if (c == 0) {
175         tot += (R == 0)      ? 1 : board->board[R - 1][C].assigned;
176         tot += (R == SIZE-1) ? 1 : board->board[R + 1][C].assigned;
177         tot += (C == 0)      ? 1 : board->board[R][C - 1].assigned;
178         tot += (C == SIZE-1) ? 1 : board->board[R][C + 1].assigned;
179         return (tot == SIZE-1);
180     }
181     return false;
182 }
183
184 bool FCCheck(futoshiki* board, int c, Do* m) {
185     // c == 0 >>> row
186     if (c == 0) for (int i = 0;i<SIZE;i++) {
187         if (!m->curdom[i]) {
188             m->curdom[i] = 1;
189             // No falsification
190             if(RowCheck(board, m)) m->curdom[i] = 0;
191         }
192     }
193     // c == 1 >>> col
194     else if (c == 1) for (int i = 0;i<SIZE;i++) {
195         if (!m->curdom[i]) {
196             m->curdom[i] = 1;
```

```
197                 // No falsification
198                 if(ColCheck(board, m)) m->curdom[i] = 0;
199             }
200         }
201         // c == 2 >>> neighbour
202         else if (c == 2) for (int i = 0;i<SIZE;i++) {
203             if (!m->curdom[i]) {
204                 m->curdom[i] = 1;
205                 // No falsification
206                 if(NeiCheck(board, m)) m->curdom[i] = 0;
207             }
208         }
209         if (CDcount(m) == SIZE) {
210             // cout << "DWO!! At c = " << c <<endl;
211             // displaydo(m);
212             return false;
213         }
214         else return true;
215 }
216
217 bool Goal(futoshiki* board) {
218     for (int i = 0;i<SIZE;i++) {
219         for (int j = 0;j<SIZE;j++) {
220             if (!board->board[i][j].assigned) return false;
221         }
222     }
223     return true;
224 }
225
226 Do* heuristicpick(futoshiki* board) {
227     // MRV
228     Do* maxi = &board->board[0][0];
229     for (int i = 0;i<SIZE;i++) {
```

```
230         for (int j = 0;j<SIZE;j++) {
231             if(board->board[i][j].assigned) continue;
232             if(CDcount(maxi) < CDcount(&board->board[i][j]) ||
233                 maxi->assigned) {
234                 maxi = &board->board[i][j];
235                 if (CDcount(maxi) == SIZE-1) return maxi;
236             }
237         }
238     }
239     return maxi;
240 }
241
242 void propagate(futoshiki* board, Do* m) {
243         // cout<<"P: "<<m->val<<endl;
244     for (int i = 0;i<SIZE;i++) {
245         board->board[m->row][i].curdom[m->val-1] = 1;
246         board->board[i][m->col].curdom[m->val-1] = 1;
247     }
248     if (m->r == -1) {
249         for (int i = 0;i<m->val - 1;i++) {
250             board->board[m->row][m->col + 1].curdom[i] = 1;
251         }
252     }
253     else if (m->r == 1) {
254         for (int i = m->val;i<SIZE;i++) {
255             board->board[m->row][m->col + 1].curdom[i] = 1;
256         }
257     }
258     if (m->u == -1) {
259         for (int i = 0;i<m->val - 1;i++) {
260             board->board[m->row - 1][m->col].curdom[i] = 1;
261         }
262     }
```

```
263     else if (m->u == 1) {
264         for (int i = m->val;i<SIZE;i++) {
265             board->board[m->row - 1][m->col].curdom[i] = 1;
266         }
267     }
268     if (m->l == -1) {
269         for (int i = 0;i<m->val - 1;i++) {
270             board->board[m->row][m->col - 1].curdom[i] = 1;
271         }
272     }
273     else if (m->l == 1) {
274         for (int i = m->val;i<SIZE;i++) {
275             board->board[m->row][m->col - 1].curdom[i] = 1;
276         }
277     }
278     if (m->d == -1) {
279         for (int i = 0;i<m->val - 1;i++) {
280             board->board[m->row + 1][m->col].curdom[i] = 1;
281         }
282     }
283     else if (m->d == 1) {
284         for (int i = m->val;i<SIZE;i++) {
285             board->board[m->row + 1][m->col].curdom[i] = 1;
286         }
287     }
288 }
289
290 void Copyboard(futoshiki* dest, const futoshiki* src) {
291         memcpy(dest, src, sizeof(futoshiki));
292 }
293
294 bool FC(futoshiki* board, int level) {
295     if (Goal(board)) {
```

```
296         // Return when all cells are assigned.
297         cout << "Goal!" << endl;
298         display(board);
299         return true;
300     }
301     Do* v = heuristicpick(board); // Pick with MRV
302     v->assigned = true;
303     bool dwo = false;
304     int pos = 0;
305     for (int i = 0;i<SIZE;i++) if (!v->curdom[i]) {
306         futoshiki boardcopy;
307         Copyboard(&boardcopy, board);
308         v->val = i+1;
309         propagate(board, v);
310         dwo = false;
311         // row constraint
312         if (!dwo && CheckCons(board, 0, v)) {
313             for (int i = 0;i<SIZE;i++)
314             if (!board->board[v->row][i].assigned) {
315                 dwo = !FCCheck(board, 0, &board->board[v->row][i]);
316             }
317         }
318         // col constraint
319         if (!dwo && CheckCons(board, 1, v)) {
320             for (int i = 0;i<SIZE;i++)
321             if (!board->board[i][v->col].assigned) {
322                 dwo = !FCCheck(board, 1, &board->board[i][v->col]);
323             }
324         }
325         // neighbour constraint
326         if (!dwo && CheckCons(board, 2, v)) {
327             if (v->row &&
328                 board->board[v->row - 1][v->col].assigned) {
```

```
329                 dwo = !FCCheck(board, 2,
330                         &board->board[v->row - 1][v->col]);
331             }
332             else if (v->row!=SIZE-1 &&
333                 board->board[v->row + 1][v->col].assigned) {
334                 dwo = !FCCheck(board, 2,
335                         &board->board[v->row + 1][v->col]);
336             }
337             else if (v->col &&
338                 board->board[v->row][v->col - 1].assigned) {
339                 dwo = !FCCheck(board, 2,
340                         &board->board[v->row][v->col - 1]);
341             }
342             else if (v->col!=SIZE-1 &&
343                 board->board[v->row][v->col + 1].assigned) {
344                 dwo = !FCCheck(board, 2,
345                         &board->board[v->row][v->col + 1]);
346             }
347         }
348         if(!dwo && FC(board, level + 1)) return true;
349         Copyboard(board, &boardcopy);
350     }
351     v->assigned = false;
352     return false;
353 }
354
355 void display(futoshiki* board) {
356     for (int i = 0;i<SIZE;i++) {
357         cout << "\t";
358         for (int j = 0;j<SIZE;j++) {
359             if (board->board[i][j].r == 1)
360                 cout << " " << board->board[i][j].val << " >";
361             else if (board->board[i][j].r == -1)
```

```
362                 cout << " " << board->board[i][j].val << " <";
363             else cout << " " << board->board[i][j].val << "  ";
364         }
365         cout << endl << "\t";
366         if (i!=SIZE-1) {
367             for (int j = 0;j<SIZE;j++) {
368                 if (board->board[i+1][j].u == 1) {
369                     cout << " ^  ";
370                 }
371                 else if (board->board[i+1][j].u == -1) {
372                     cout << " v  ";
373                 }
374                 else cout << "    ";
375             }
376         }
377         cout << endl;
378     }
379 }
```

4 Results

- Solve 7 × 7 board

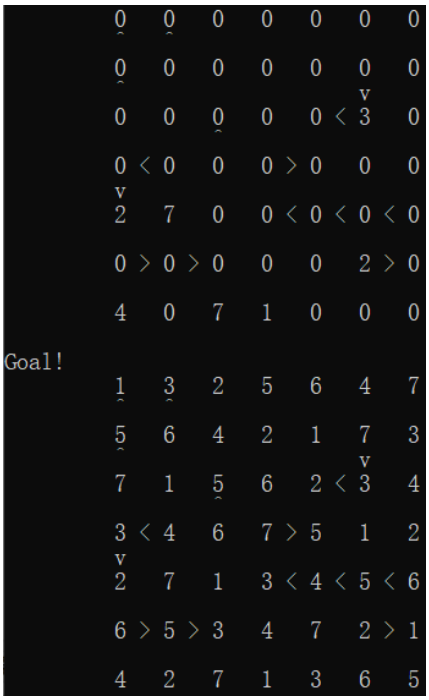

Figure 2: 7x7 board - 1

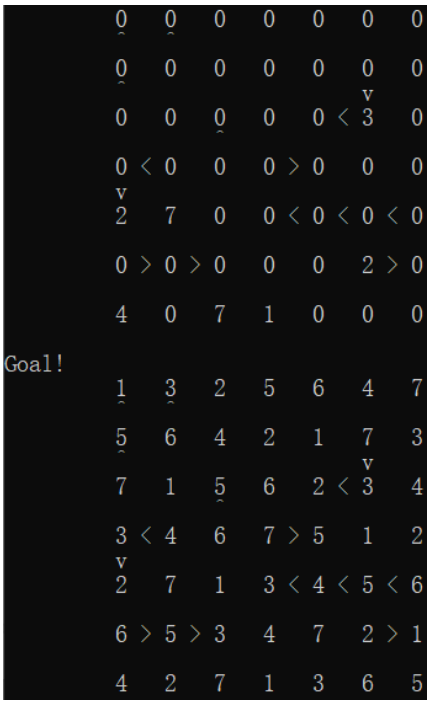

Figure 3: 7x7 board - 2

- Solve 9 × 9 board

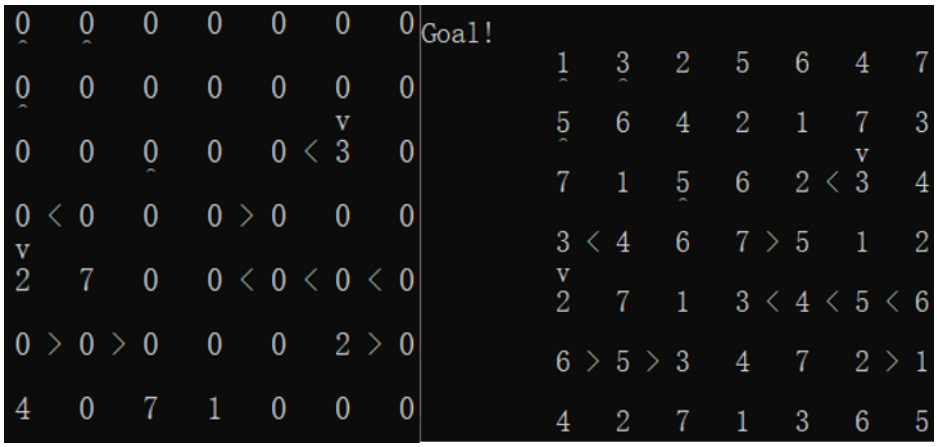

Figure 4: 9x9 board