

E04 Futoshiki Puzzle (Forward Checking)

19335016 HaoRan Chen

2021 年 10 月 18 日

目录

1	Futoshiki	2
2	Tasks	2
3	Codes	3
4	Results	10

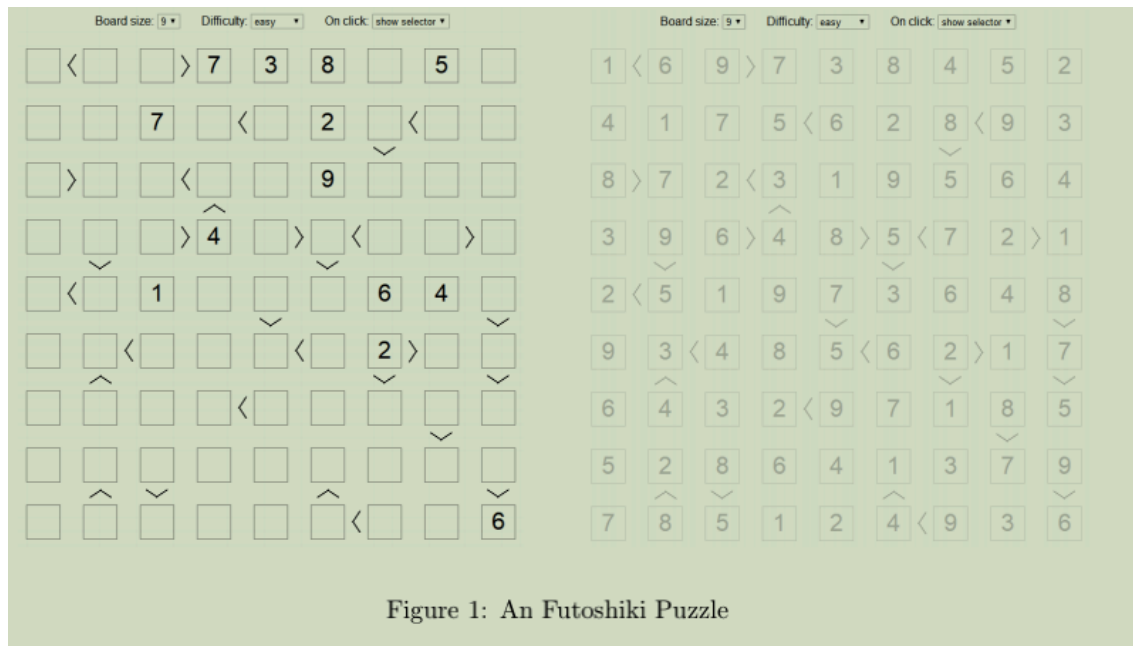


Figure 1: An Futoshiki Puzzle

图 1: Futoshiki

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/>.

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_course2021@163.com.

3 Codes

```

1 #include <fstream>
2 #include <iostream>
3 #include <map>
4 #include <set>
5 #include <string>
6 #include <vector>
7 #include <ctime>
8 #define PII pair<int, int>
9 #define VVI vector<vector<int> >
10 #define VVSI vector<vector<set<int> > >
11 using namespace std;
12 const int SIZE=9;
13 //DWO: Domain Wipe Out, 表示该节点（变量）的值域Domain已经为空。
14 class Futoshiki{
15     public:
16     VVI puzzle;
17     //int puzzle[9][9];
18     vector<pair<PII, PII>> less_constraints;
19     Futoshiki(){
20         puzzle = {{0, 0, 0, 7, 3, 8, 0, 5, 0},
21                 {0, 0, 7, 0, 0, 2, 0, 0, 0},
22                 {0, 0, 0, 0, 0, 9, 0, 0, 0},
23                 {0, 0, 0, 4, 0, 0, 0, 0, 0},
24                 {0, 0, 1, 0, 0, 0, 6, 4, 0},
25                 {0, 0, 0, 0, 0, 0, 2, 0, 0},
26                 {0, 0, 0, 0, 0, 0, 0, 0, 0},
27                 {0, 0, 0, 0, 0, 0, 0, 0, 0},
28                 {0, 0, 0, 0, 0, 0, 0, 0, 6}};
29
30         add(0, 0, 0, 1);        add(0, 3, 0, 2);
31         add(1, 3, 1, 4);        add(1, 6, 1, 7);

```

```
32     add(2, 6, 1, 6);           add(2, 1, 2, 0);
33     add(2, 2, 2, 3);           add(2, 3, 3, 3);
34     add(3, 3, 3, 2);           add(3, 5, 3, 4);
35     add(3, 5, 3, 6);           add(3, 8, 3, 7);
36     add(4, 1, 3, 1);           add(4, 5, 3, 5);
37     add(4, 0, 4, 1);           add(5, 4, 4, 4);
38     add(5, 8, 4, 8);           add(5, 1, 5, 2);
39     add(5, 4, 5, 5);           add(5, 7, 5, 6);
40     add(5, 1, 6, 1);           add(6, 6, 5, 6);
41     add(6, 8, 5, 8);           add(6, 3, 6, 4);
42     add(7, 7, 6, 7);           add(7, 1, 8, 1);
43     add(8, 2, 7, 2);           add(7, 5, 8, 5);
44     add(8, 8, 7, 8);           add(8, 5, 8, 6);
45 }
46 // 计算总可行数
47 int domainCount(const VVSI& domains) {
48     int count = 0;
49     for(int i = 0; i < SIZE; i++) for(int j = 0; j < SIZE; j++)
50         count += domains[i][j].size();
51     return count;
52 }
53
54 void add(int x, int y, int x1, int y1){
55     less_constraints.push_back({{x, y}, {x1, y1}});
56 }
57
58 bool isSolved(){
59     for(int i=0; i<SIZE; i++) for(int j=0; j<SIZE; j++)
60         if(puzzle[i][j]==0) return false;
61     return true;
62 }
63
64 // 初始化每个格子的可行域
```

```

65 VVSI makeDomains(){
66     VVSI domains(SIZE, vector<set<int> >(SIZE, set<int>()));
67     for(int i = 0; i < SIZE; i++) for(int j = 0; j < SIZE; j++){
68         if(puzzle[i][j]==0) for(int k = 0; k < SIZE; k++)
69             domains[i][j].insert(k + 1);
70         else domains[i][j].insert(puzzle[i][j]);
71     }
72
73     for(int i = 0; i < SIZE; i++) for(int j = 0; j < SIZE; j++){
74         if(puzzle[i][j]!=0){
75             for(int ii = 0; ii < SIZE; ii++) if(ii != i)
76                 domains[ii][j].erase(puzzle[i][j]);
77             for(int jj = 0; jj < SIZE; jj++) if(jj != j)
78                 domains[i][jj].erase(puzzle[i][j]);
79         }
80     }
81     // 清除不符合约束条件的数
82     for (int i = 0; i < less_constraints.size(); i++) {
83         PII sp = less_constraints[i].first;
84         PII lp = less_constraints[i].second;
85         if (puzzle[lp.first][lp.second] != 0) {
86             for (int k = puzzle[lp.first][lp.second]; k <= SIZE; k++)
87                 domains[sp.first][sp.second].erase(k);
88         }
89         else {
90             int minimum = *domains[sp.first][sp.second].begin();
91             domains[lp.first][lp.second].erase(minimum);
92         }
93         if (puzzle[sp.first][sp.second] != 0) {
94             for (int k = 1; k <= puzzle[sp.first][sp.second]; k++) {
95                 domains[lp.first][lp.second].erase(k);
96             }
97         }

```

```

98         else {
99             int minimum = *domains[lp.first][lp.second].rbegin(); // 取最后元素
100             domains[sp.first][sp.second].erase(minimum);
101         }
102     }
103     return domains;
104 }
105
106 // 在每次迭代中使用MRV函数选择一个位置，并在其域中分配一个值。
107 // 然后通过删除一些与赋值冲突的值来更新一些格子的域。
108 VVSI updateDomains(VVSI domains, const PII& pos) {
109     // 检查列
110     for (int i = 0; i < SIZE; i++) {
111         if (i == pos.first) continue;
112         else if (puzzle[i][pos.second] == puzzle[pos.first][pos.second])
113             return VVSI(); // DWO
114         else {
115             domains[i][pos.second].erase(puzzle[pos.first][pos.second]);
116             if (domains[i][pos.second].size() == 0) return VVSI(); // DWO
117         }
118     }
119
120     // 检查行
121     for (int j = 0; j < SIZE; j++) {
122         if (j == pos.second) continue;
123         else if (puzzle[pos.first][j] == puzzle[pos.first][pos.second]) {
124             return VVSI(); // DWO
125         }
126         else {
127             domains[pos.first][j].erase(puzzle[pos.first][pos.second]);
128             if (domains[pos.first][j].size() == 0) {
129                 return VVSI(); // DWO
130             }

```

```

131     }
132 }
133
134 // 检查约束条件
135 for (int i = 0; i < less_constraints.size(); i++) {
136     PII sp = less_constraints[i].first;
137     PII lp = less_constraints[i].second;
138     if (pos == lp) {
139         for (int k = puzzle[pos.first][pos.second]; k <= SIZE; k++) {
140             domains[sp.first][sp.second].erase(k);
141             if (puzzle[sp.first][sp.second] == 0 &&
142                 domains[sp.first][sp.second].size() == 0)
143                 return VVSI(); // DWO
144         }
145     }
146 }
147 else if (pos == sp) {
148     for (int k = 1; k <= puzzle[pos.first][pos.second]; k++) {
149         domains[lp.first][lp.second].erase(k);
150         if (puzzle[lp.first][lp.second] == 0 &&
151             domains[lp.first][lp.second].size() == 0)
152             return VVSI(); // DWO
153     }
154 }
155 }
156 }
157 return domains;
158 }
159
160 // 选择可行域最小的格子并返回其位置，minimum remaining values (MRV)
161 PII MRV(const VVSI& domains) {
162     int val = 114514;
163     PII pos = make_pair(-1, -1);

```

```
164     for (int i = 0; i < SIZE; i++) {
165         for (int j = 0; j < SIZE; j++) {
166             if (puzzle[i][j] == 0 && domains[i][j].size() < val) {
167                 val = domains[i][j].size();
168                 pos = make_pair(i, j);
169             }
170         }
171     }
172     return pos;
173 }
174
175 VVI ForwardChecking(const VVSI& domains) {
176     if (isSolved()) return puzzle;
177     PII pos = MRV(domains);
178     for (auto p = domains[pos.first][pos.second].begin();
179          p != domains[pos.first][pos.second].end(); p++) {
180         puzzle[pos.first][pos.second] = *p;
181         auto temp_domains = updateDomains(domains, pos);
182         if (temp_domains.size() != 0) {
183             VVI res = ForwardChecking(temp_domains);
184             if (res.size() != 0) return res;
185         }
186     }
187
188     puzzle[pos.first][pos.second] = 0;
189     return VVI();
190 }
191
192 void print(){
193     for(int i = 0; i < SIZE; i++){
194         for(int j = 0; j < SIZE; j++) printf("%d ",puzzle[i][j]);
195         printf("\n");
196     }
```



```
197         puts("=====");
198     }
199 };
200 int main(){
201     clock_t start, end;
202     Futoshiki game;
203     game.print();
204     VVSI domains = game.makeDomains();
205
206     start=clock();
207     game.ForwardChecking(domains);
208     end=clock();
209
210     game.print();
211     printf("\ntime is %lf s\n",((double)end-start)/CLOCKS_PER_SEC);
212     return 0;
213 }
```

4 Results

The image shows two side-by-side command prompt windows. The left window displays the results of a Backtracking algorithm, showing a 10x10 grid of numbers and a completion time of 3.793000 s. The right window displays the results of a Forward Checking algorithm, showing the same 10x10 grid of numbers and a completion time of 0.998000 s. Both windows show the same sequence of numbers, indicating that both algorithms found the same solution.

0	0	0	7	3	8	0	5	0	
0	0	7	0	0	2	0	0	0	
0	0	0	0	0	9	0	0	0	
0	0	0	4	0	0	0	0	0	
0	0	1	0	0	0	6	4	0	
0	0	0	0	0	0	2	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	6	
=====									
1	6	9	7	3	8	4	5	2	
4	1	7	5	6	2	8	9	3	
8	7	2	3	1	9	5	6	4	
3	9	6	4	8	5	7	2	1	
2	5	1	9	7	3	6	4	8	
9	3	4	8	5	6	2	1	7	
6	4	3	2	9	7	1	8	5	
5	2	8	6	4	1	3	7	9	
7	8	5	1	2	4	9	3	6	
=====									
time is 3.793000 s									
请按任意键继续. . .									

0	0	0	7	3	8	0	5	0	
0	0	7	0	0	2	0	0	0	
0	0	0	0	0	9	0	0	0	
0	0	0	4	0	0	0	0	0	
0	0	1	0	0	0	6	4	0	
0	0	0	0	0	0	2	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	6	
=====									
1	6	9	7	3	8	4	5	2	
4	1	7	5	6	2	8	9	3	
8	7	2	3	1	9	5	6	4	
3	9	6	4	8	5	7	2	1	
2	5	1	9	7	3	6	4	8	
9	3	4	8	5	6	2	1	7	
6	4	3	2	9	7	1	8	5	
5	2	8	6	4	1	3	7	9	
7	8	5	1	2	4	9	3	6	
=====									
time is 0.998000 s									
请按任意键继续. . .									

其中, 左边为原 back tracking 算法的 check 函数运行结果, 右边为本函数运行结果, 可见相对于传统 BackTracking 算法, ForwardChecking 运行速度有较大提升.