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

E04 Futoshiki Puzzle (Forward Checking)

17341068 Yangfan Jiang

September 21, 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/.

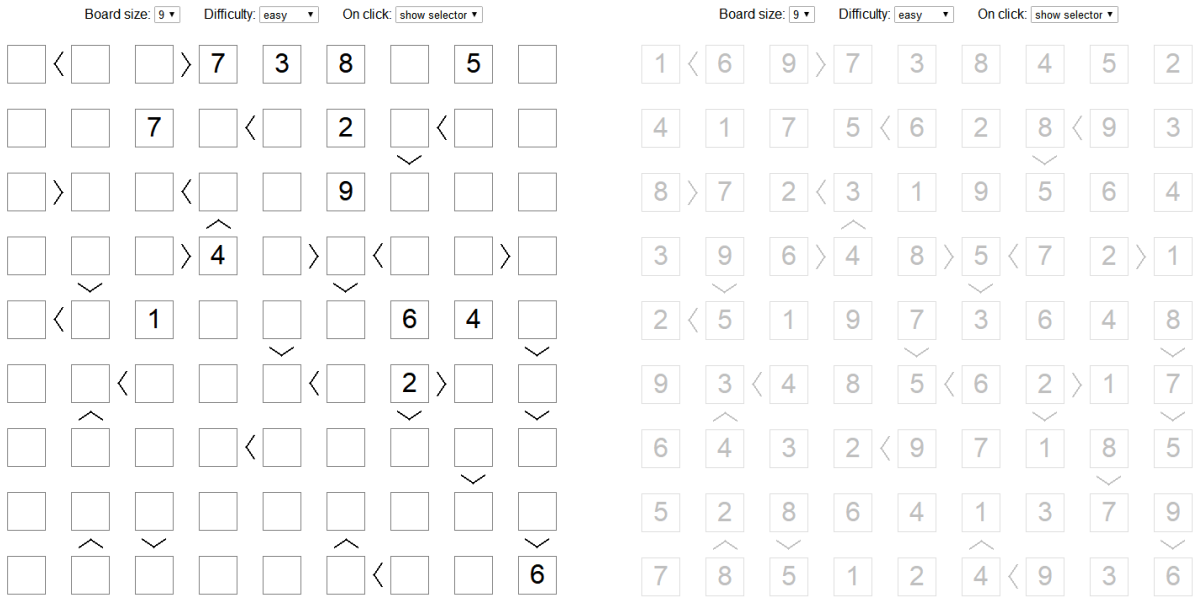

Figure 1: An Futoshiki Puzzle

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<ctime>
#include<algorithm>
#include<string>
#include<vector>
#include<tuple>

```

```
8  #define N 9
9
10 using namespace std;
11
12 int board[N][N];
13
14 // constrains
15 vector< tuple<int, int> > c;
16 vector<bool*> avaliable;
17 bool isfind = false;
18
19 int x = 0;
20
21 void initialize() {
22         int init_num, tmp, cons_num;
23         int x, y, x2, y2;
24
25         for (int i = 0; i < N * N; i++) {
26                 bool *tmp = new bool[N + 1];
27                 for (int j = 0; j < N + 1; j++)
28                         tmp[j] = false;
29                 avaliable.push_back(tmp);
30         }
31
32         // initialize borad, 0 stands for empty element
33         for (int i = 0; i < N; i++)
34                 for (int j = 0; j < N; j++)
35                         board[i][j] = 0;
36
37         cout << "# of initial numbers: ";
38         cin >> init_num;
39         for (int i = 0; i < init_num; i++) {
40                 cout << "position (x y): ";
41                 cin >> x >> y;
42                 cout << "value: ";
43                 cin >> tmp;
44                 board[x - 1][y - 1] = tmp;
45         }
46
47         cout << "# of constrains: ";
48         cin >> cons_num;
49         for (int i = 0; i < cons_num; i++) {
50                 cout << "positions (x1 y1) (x2 y2) constrain (>)";
51                 cin >> x >> y >> x2 >> y2;
52                 c.push_back(make_tuple((x - 1) * N + (y - 1), (x2 - 1) * N + (y2
53         }
54 }
55
56
```

```
57 void free_mem() {
58         for (auto iter = avaliable.begin(); iter != avaliable.end(); ++iter) {
59                 delete[] * iter;
60         }
61 }
62
63 void unassigned(vector<int> &v) {
64         for (int i = N - 1; i >= 0; i--) {
65                 for (int j = N - 1; j >= 0; j--) {
66                         if (board[i][j] == 0)
67                                 v.push_back(i*N + j);
68                 }
69         }
70 }
71
72 bool getCurDom(int pos, bool avaliable[]) {
73         int x = pos / N;
74         int y = pos % N;
75
76         for (int i = 0; i < N + 1; i++)
77                 avaliable[i] = true;
78
79         for (int i = 0; i < N; i++) {
80                 if (board[x][i] != 0) {
81                         avaliable[board[x][i]] = false;
82                 }
83                 if (board[i][y] != 0) {
84                         avaliable[board[i][y]] = false;
85                 }
86         }
87
88         // check all inequality constrains
89         for (auto it = c.begin(); it != c.end(); ++it) {
90                 int p1 = get<0>(*it);
91                 int p2 = get<1>(*it);
92                 int x1 = p1 / N;
93                 int y1 = p1 % N;
94                 int x2 = p2 / N;
95                 int y2 = p2 % N;
96
97                 if (pos == p1) {
98                         avaliable[1] = false;
99                         if (board[x2][y2]) {
100                                 for (int i = 1; i <= board[x2][y2]; i++)
101                                         avaliable[i] = false;
102                         }
103                 }
104                 else if (pos == p2) {
105                         avaliable[N] = false;
```

```
106                         if (board[x1][y1]) {
107                                 for (int i = N; i >= board[x1][y1]; i--)
108                                         avaliable[i] = false;
109                         }
110                 }
111         }
112
113         for (int i = 1; i <= N; i++)
114                 if (avaliable[i]) {
115                         return false;
116                 }
117         return true;
118 }
119
120
121 // find the index of minimum remaining values
122 int MRV(const vector<int>& v) {
123         int min = 0x7fffffff;
124         //int min = -1;
125         int index = -1;
126         for (int i = 0; i < v.size() ; i++) {
127                 int cnt = 0;
128                 for (int j = 1; j < N + 1; j++) {
129                         if (avaliable[v[i]][j])
130                                 cnt++;
131                 }
132                 if (min > cnt) {
133                         min = cnt;
134                         index = i;
135                 }
136         }
137         return index;
138 }
139
140
141
142 void FC() {
143         ++x;
144         // v (vector<pos>): unassigned element
145         vector<int> v;
146         unassigned(v);
147         if (!v.size()) {
148                 isfind = true;
149                 return;
150         }
151         // get current dom of each unassigned element
152         for (auto iter = v.begin(); iter != v.end(); ++iter) {
153                 if (getCurDom(*iter, avaliable[*iter]))
154                         return;
```

```
155         }
156
157         int min_index = MRV(v);
158         int curr_mem = v[min_index];
159
160         for (int i = N; i >= 1; i--) {
161                 if (avaliable[curr_mem][i]) {
162                         board[curr_mem / N][curr_mem % N] = i;
163                         FC();
164                 }
165                 if (isfind)
166                         return;
167         }
168         v.erase(v.begin() + min_index);
169         // assigned = false
170         board[curr_mem / N][curr_mem % N] = 0;
171         return;
172 }
173
174 void printans() {
175         for (int i = 0; i < N; i++) {
176                 for (int j = 0; j < N; j++) {
177                         cout << board[i][j] << " ";
178                 }
179                 cout << endl;
180         }
181 }
182
183 int main()
184 {
185         initialize();
186         clock_t startTime, endTime;
187         startTime = clock();
188         FC();
189         endTime = clock();
190         cout << "Time used: " << (double)(endTime - startTime) / (CLOCKS_PER_SEC)
191                 << "s" << endl;
192         printans();
193         free_mem();
194         cout << x << " steps" << endl;
195         return 0;
196 }
```

4 Results

Figure 2: Result