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

E04 Futoshiki Puzzle (Forward Checking)

18308045 Zhengyang Gu

September 21, 2020

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/.

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 2018@foxmail.com.

3 Codes

```
//
// Created by GreenArrow on 2020/9/14.
//

#include <cstring>
#include <ctime>
#include <iostream>
#include <vector>

using namespace std;

struct my_tuple
{
    int x;
    int y;
    int value;
```

```
};

class FutoshikiPuzzle
{
public:
    vector<vector<int>> maps;
    vector<pair<pair<int, int>, pair<int, int>>> less_constraints;
    int nRow, nColumn;
    //表示第行中某个数字是否存在x
    int Count_RowNumbers[9][10];
    //表示第列某个数字是否存在y
    int Count_ColumnNumbers[9][10];
    int total = 0;
    //表示(x,y)点值是否因被剪枝valueFC
    unsigned char is_pruned[9][9][10];
    //表示(x,y)点被剪枝的个数
    unsigned char pruned_num[9][9];
    vector<my_tuple> restore;

    void initial()
    {
        //初始地图
        maps = {{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, 6}};
        nRow = maps.size();
        nColumn = maps[0].size();

        //添加限制
        addConstraints(0, 0, 0, 1);
        addConstraints(0, 3, 0, 2);
        addConstraints(1, 3, 1, 4);
        addConstraints(1, 6, 1, 7);
        addConstraints(2, 6, 1, 6);
        addConstraints(2, 1, 2, 0);
```

```
58          addConstraints(2, 2, 2, 3);
59          addConstraints(2, 3, 3, 3);
60          addConstraints(3, 3, 3, 2);
61          addConstraints(3, 5, 3, 4);
62          addConstraints(3, 5, 3, 6);
63          addConstraints(3, 8, 3, 7);
64          addConstraints(4, 1, 3, 1);
65          addConstraints(4, 5, 3, 5);
66          addConstraints(4, 0, 4, 1);
67          addConstraints(5, 4, 4, 4);
68          addConstraints(5, 8, 4, 8);
69          addConstraints(5, 1, 5, 2);
70          addConstraints(5, 4, 5, 5);
71          addConstraints(5, 7, 5, 6);
72          addConstraints(5, 1, 6, 1);
73          addConstraints(6, 6, 5, 6);
74          addConstraints(6, 8, 5, 8);
75          addConstraints(6, 3, 6, 4);
76          addConstraints(7, 7, 6, 7);
77          addConstraints(7, 1, 8, 1);
78          addConstraints(8, 2, 7, 2);
79          addConstraints(7, 5, 8, 5);
80          addConstraints(8, 8, 7, 8);
81          addConstraints(8, 5, 8, 6);
82
83          //初始化域
84          memset(is_pruned, 0, sizeof(is_pruned));
85          for (int x = 0; x < 9; x++)
86          {
87              for (int y = 0; y < 9; y++)
88              {
89                  int i = maps[x][y];
90                  if (i)
91                  {
92                      Count_RowNumbers[x][i]++;
93                      Count_ColumnNumbers[y][i]++;
94                      for (int row_or_col = 0; row_or_col < 9; row_or_col++)
95                      {
96                          if (row_or_col != x)
97                          {
98                              if (!is_pruned[row_or_col][y][i])
```

```
                        {
                            is_pruned[row_or_col][y][i] = 1;
                            pruned_num[row_or_col][y]++;
                        }
                    }
                    if (row_or_col != y)
                    {
                        if (!is_pruned[x][row_or_col][i])
                        {
                            is_pruned[x][row_or_col][i] = 1;
                            pruned_num[x][row_or_col]++;
                        }
                    }
                }
            }
        }
    }
    for (auto &less_constraint : less_constraints)
    {
        int x1 = less_constraint.first.first;
        int y1 = less_constraint.first.second;
        int x2 = less_constraint.second.first;
        int y2 = less_constraint.second.second;
        int value1 = maps[x1][y1];
        int value2 = maps[x2][y2];
        if (value1 && !value2)
        {
            for (int value = 1; value <= value1; value++)
            {
                if (!is_pruned[x2][y2][value])
                {
                    is_pruned[x2][y2][value] = 1;
                    pruned_num[x2][y2]++;
                }
            }
        }
        else if (!value1 && value2)
        {
            for (int value = value2; value <= 9; value++)
            {
                if (!is_pruned[x1][y1][value])
```

```
                    {
                        is_pruned[x1][y1][value] = 1;
                        pruned_num[x1][y1]++;
                    }
                }
            }
        }
        return;
    }

    void addConstraints(int x, int y, int x1, int y1)
    {
        less_constraints.push_back({{x, y},
                                    {x1, y1}});
    }

    //检查当前位置是否可行
    bool check(int x, int y)
    {
        for (int i = 1; i < 10; i++)
        {
            if (Count_RowNumbers[x][i] > 1 || Count_ColumnNumbers[y][i] > 1)
            {
                return false;
            }
        }

        for (auto &less_constraint : less_constraints)
        {
            if (less_constraint.first.first == x && less_constraint.first.second == y)
            {
                if (maps[x][y] == 9)
                {
                    return false;
                }
                if (maps[less_constraint.second.first][less_constraint.second.second]
    > 0 &&
                    maps[less_constraint.second.first][less_constraint.second.second]
    <= maps[x][y])
                {
                    return false;
```

```
                }
            }
        }

        for (auto &less_constraint : less_constraints)
        {
            if (less_constraint.second.first == x && less_constraint.second.second ==
    y)
            {
                if (maps[x][y] == 1)
                {

                    return false;
                }
                if (maps[less_constraint.first.first][less_constraint.first.second] >
    0 &&
                    maps[less_constraint.first.first][less_constraint.first.second] >=
     maps[x][y])
                {

                    return false;
                }
            }
        }
        return true;
    }

    //显示图片
    void show()
    {
        for (int i = 0; i < nRow; i++)
        {
            for (int j = 0; j < nColumn; j++)
            {
                cout << maps[i][j] << " ";
            }
            cout << endl;
        }
        cout << "==========================" << endl;
    }

```

```
217     void find_next(int &next_x, int &next_y)
218     {
219         for (next_x = 0; next_x < 9; next_x++)
220         {
221             for (next_y = 0; next_y < 9; next_y++)
222             {
223                 if (!maps[next_x][next_y])
224                 {
225                     goto next;
226                 }
227             }
228         }
229     next:
230         int temp_x, temp_y;
231         for (temp_x = next_x, temp_y = next_y + 1; temp_y < 9; temp_y++)
232         {
233             if (!maps[temp_x][temp_y] && pruned_num[next_x][next_y] < pruned_num[
    temp_x][temp_y])
234             {
235                 next_y = temp_y;
236             }
237         }
238         for (temp_x = next_x + 1; temp_x < 9; temp_x++)
239         {
240             for (temp_y = 0; temp_y < 9; temp_y++)
241             {
242                 if (!maps[temp_x][temp_y] && pruned_num[next_x][next_y] < pruned_num[
    temp_x][temp_y])
243                 {
244                     next_x = temp_x;
245                     next_y = temp_y;
246                 }
247             }
248         }
249     }
250
251     bool search(int x, int y)
252     {
253         if (maps[x][y] == 0)
254         {
255           total++;
```

```
256             for (int i = 1; i < 10; i++)
257             {
258                 maps[x][y] = i;
259                 Count_RowNumbers[x][i]++;
260                 Count_ColumnNumbers[y][i]++;
261                 if (check(x, y))
262                 {
263                     if (x == 8 && y == 8)
264                     {
265                         return true;
266                     }
267                     int next_x, next_y;
268                     if (y != 8)
269                     {
270                         next_x = x;
271                         next_y = y + 1;
272                     }
273                     else
274                     {
275                         next_x = x + 1;
276                         next_y = 0;
277                     }
278 
279                     if (search(next_x, next_y))
280                     {
281                         return true;
282                     }
283                 }
284                 maps[x][y] = 0;
285                 Count_RowNumbers[x][i]--;
286                 Count_ColumnNumbers[y][i]--;
287             }
288         }
289         else
290         {
291             if (x == 8 && y == 8)
292             {
293                 return true;
294             }
295             int next_x, next_y;
296             if (y != 8)
```

```
297             {
298                 next_x = x;
299                 next_y = y + 1;
300             }
301             else
302             {
303                 next_x = x + 1;
304                 next_y = 0;
305             }
306 
307             if (search(next_x, next_y))
308             {
309                 return true;
310             }
311         }
312         return false;
313     }
314 
315     bool FC_search(int x, int y)
316 {
317     total++;
318     my_tuple back;
319     for (int i = 1; i < 10; i++)
320     {
321       if (!is_pruned[x][y][i])
322       {
323         maps[x][y] = i;
324         Count_RowNumbers[x][i]++;
325         Count_ColumnNumbers[y][i]++;
326         if (check(x, y))
327         {
328           int restore_num = 0;
329           for (int row_or_col = 0; row_or_col < 9; row_or_col++)
330           {
331             if (!maps[row_or_col][y] && !is_pruned[row_or_col][y][i])
332             {
333               is_pruned[row_or_col][y][i] = 1;
334               pruned_num[row_or_col][y]++;
335               restore.push_back({row_or_col, y, i});
336               restore_num++;
337             }
```

```
            if (!maps[x][row_or_col] && !is_pruned[x][row_or_col][i])
            {
              is_pruned[x][row_or_col][i] = 1;
              pruned_num[x][row_or_col]++;
              restore.push_back({x, row_or_col, i});
              restore_num++;
            }
          }
          for (auto &less_constraint : less_constraints)
          {
            int x1 = less_constraint.first.first;
            int y1 = less_constraint.first.second;
            int x2 = less_constraint.second.first;
            int y2 = less_constraint.second.second;
            if (x1 == x && y1 == y)
            {
              if (!maps[x2][y2])
              {
                for (int value = 1; value <= i; value++)
                {
                  if (!is_pruned[x2][y2][value])
                  {
                    is_pruned[x2][y2][value] = 1;
                    pruned_num[x2][y2]++;
                    restore.push_back({x2, y2, value});
                    restore_num++;
                  }
                }
              }
            }
            if (x2 == x && y2 == y)
            {
              if (!maps[x1][y1])
              {
                for (int value = i; value <= 9; value++)
                {
                  if (!is_pruned[x1][y1][value])
                  {
                    is_pruned[x1][y1][value] = 1;
                    pruned_num[x1][y1]++;
                    restore.push_back({x1, y1, value});
```

```
379                     restore_num++;
380                   }
381                 }
382               }
383             }
384           }
385           int next_x, next_y;
386           find_next(next_x, next_y);
387           if (next_x == 9)
388           {
389             return true;
390           }
391           if (FC_search(next_x, next_y))
392           {
393             return true;
394           }
395           while (restore_num--)
396           {
397             back = restore.back();
398             is_pruned[back.x][back.y][back.value] = 0;
399             pruned_num[back.x][back.y]--;
400             restore.pop_back();
401           }
402         }
403       }
404       maps[x][y] = 0;
405       Count_RowNumbers[x][i]--;
406       Count_ColumnNumbers[y][i]--;
407     }
408     return false;
409   }
410 };
411
412 int main()
413 {
414     FutoshikiPuzzle *futoshikiPuzzle = new FutoshikiPuzzle();
415     futoshikiPuzzle->initial();
416     futoshikiPuzzle->show();
417     clock_t start = clock();
418     futoshikiPuzzle->search(0, 0);
419     clock_t end = clock();
```

```
420     double endtime = (double)(end - start) / CLOCKS_PER_SEC;
421     cout << "无：FC" << endl;
422     futoshikiPuzzle->show();
423     cout << "时间：" << endtime << " s" << endl;
424     cout << "数：search" << futoshikiPuzzle->total << endl;
425     delete futoshikiPuzzle;
426     futoshikiPuzzle = new FutoshikiPuzzle();
427     futoshikiPuzzle->initial();
428   int next_x, next_y;
429   futoshikiPuzzle->find_next(next_x, next_y);
430     start = clock();
431     futoshikiPuzzle->FC_search(next_x, next_y);
432     end = clock();
433     endtime = (double)(end - start) / CLOCKS_PER_SEC;
434     cout << "有：FC" << endl;
435     futoshikiPuzzle->show();
436     cout << "时间：" << endtime << " s" << endl;
437     cout << "数：search" << futoshikiPuzzle->total << endl;
438 }
```

4 Results

```
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 6
======================
无FC:
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
======================
时间: 3.993 s
search数: 3219268
有FC:
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
======================
时间: 0.6544 s
search数: 309083
```

Figure 1: result