E04 Futoshiki Puzzle (Forward Checking)

学号	姓名	日期
18340215	张天祎	2020.9.26

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、Task

- Please solve the above Futoshiki puzzle (Figure 1) with forward checking algorithm.
- Write the related codes and take a screenshot of the running results in the file named E04 YourNumber.pdf, and send it to ai 2020@foxmail.com.

Figure 1: An Futoshiki Puzzle

3、Codes

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

#include <iostream>
#include <vector>
```

```
using namespace std;

class FutoshikiPuzzle
{
public:
    vector<vector<int>> maps;
    vector<pair<pair<int, int>, pair<int, int>>> less_constraints;
    int nRow, nColumn;
    //表示第i行第j列的可选值,值表示被限制次数,0为可选,
    char domain[9][9][9];

    bool set(int i, int j, int val, int sign) //sign=1为选定,0为回退
    {
        if (sign)
        {
            maps[i][j] = val;
            for (int k = 0; k < 9; k++)
            {
                domain[i][j][k]++;
            }
            for (int m = 0; m < 9; m++)
            {
                domain[m][j][maps[i][j] - 1]++;
            }
            for (int m = 0; m < 9; m++)
            {
                domain[i][m][maps[i][j] - 1]++;
            }
            for (auto k : less_constraints)
            {
                pair<int, int> curr(i, j);
                if (i < k.first.first && i < k.second.first)
                    break;
                if (curr == k.first)
                {
                    for (int m = 0; m < val; m++)
                    {
                        domain[k.second.first][k.second.second][m]++;
                    }
                }
                else if (curr == k.second)
                {
                    for (int m = val - 1; m < 9; m++)
                    {
                        domain[k.first.first][k.first.second][m]++;
                    }
                }
            }
            for (int m = 0; m < 9; m++)
            {
                for (int n = 0; n < 9; n++)
                {
                    if (maps[m][n])
                        continue;
                    int flag = false;
                    for (int k = 0; k < 9; k++)
                    {
```

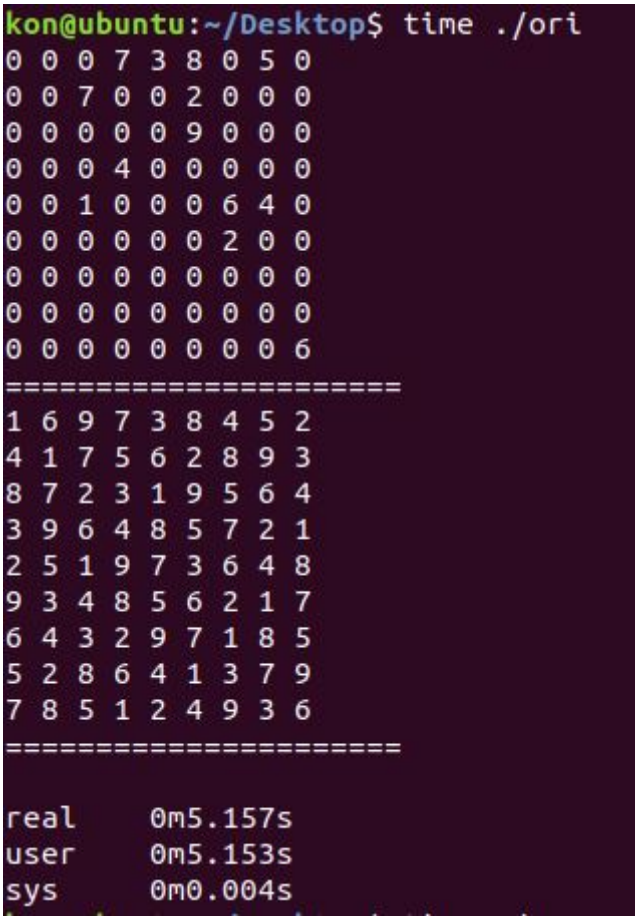
```
                    if (!domain[m][n][k])
                    {
                        flag = true;
                        break;
                    }
                }
                if (!flag)
                    return false;
            }
        }
        return true;
    }
    else
    {
        for (int k = 0; k < 9; k++)
        {
            domain[i][j][k]--;
        }
        for (int m = 0; m < 9; m++)
        {
            domain[m][j][maps[i][j] - 1]--;
        }
        for (int m = 0; m < 9; m++)
        {
            domain[i][m][maps[i][j] - 1]--;
        }
        for (auto k : less_constraints)
        {
            pair<int, int> curr(i, j);
            if (i < k.first.first && i < k.second.first)
                break;
            if (curr == k.first)
            {
                for (int m = 0; m < val; m++)
                {
                    domain[k.second.first][k.second.second][m]--;
                }
            }
            else if (curr == k.second)
            {
                for (int m = val - 1; m < 9; m++)
                {
                    domain[k.first.first][k.first.second][m]--;
                }
            }
        }
        maps[i][j] = 0;
        return true;
    }
}

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

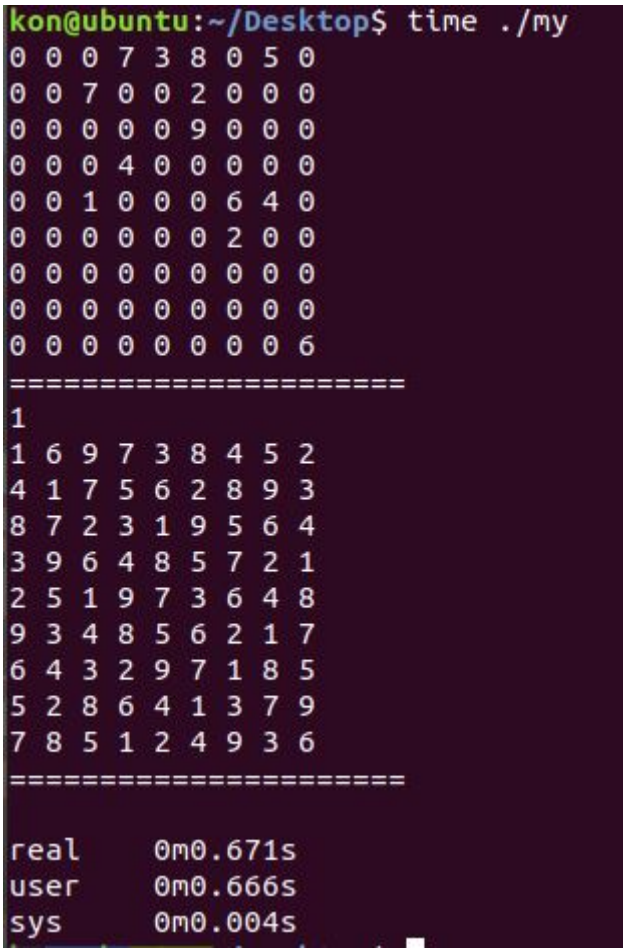
```
        addConstraints(2, 6, 1, 6);
        addConstraints(2, 1, 2, 0);
        addConstraints(2, 2, 2, 3);
        addConstraints(2, 3, 3, 3);
        addConstraints(3, 3, 3, 2);
        addConstraints(3, 5, 3, 4);
        addConstraints(3, 5, 3, 6);
        addConstraints(3, 8, 3, 7);
        addConstraints(4, 1, 3, 1);
        addConstraints(4, 5, 3, 5);
        addConstraints(4, 0, 4, 1);
        addConstraints(5, 4, 4, 4);
        addConstraints(5, 8, 4, 8);
        addConstraints(5, 1, 5, 2);
        addConstraints(5, 4, 5, 5);
        addConstraints(5, 7, 5, 6);
        addConstraints(5, 1, 6, 1);
        addConstraints(6, 6, 5, 6);
        addConstraints(6, 8, 5, 8);
        addConstraints(6, 3, 6, 4);
        addConstraints(7, 7, 6, 7);
        addConstraints(7, 1, 8, 1);
        addConstraints(8, 2, 7, 2);
        addConstraints(7, 5, 8, 5);
        addConstraints(8, 8, 7, 8);
        addConstraints(8, 5, 8, 6);
        //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();
        for (int i = 0; i < 9; i++)
        {
            for (int j = 0; j < 9; j++)
            {
                for (int k = 0; k < 9; k++)
                {
                    domain[i][j][k] = 0;
                }
            }
        }

        for (int i = 0; i < 9; i++)
        {
            for (int j = 0; j < 9; j++)
            {
                if (maps[i][j] != 0)
                {
                    for (int k = 0; k < 9; k++)
                    {
                        domain[i][j][k]++;
```

```
                    }
                    for (int m = 0; m < 9; m++)
                    {
                        domain[m][j][maps[i][j] - 1]++;
                    }
                    for (int m = 0; m < 9; m++)
                    {
                        domain[i][m][maps[i][j] - 1]++;
                    }
                    for (auto k : less_constraints)
                    {
                        pair<int, int> curr(i, j);
                        if (i < k.first.first && i < k.second.first)
                            break;
                        if (curr == k.first)
                        {
                            for (int m = 0; m < maps[i][j]; m++)
                            {
                                domain[k.second.first][k.second.second][m]++;
                            }
                        }
                        else if (curr == k.second)
                        {
                            for (int m = maps[i][j] - 1; m < 9; m++)
                            {
                                domain[k.first.first][k.first.second][m]++;
                            }
                        }
                    }
                }
            }
        }
    }

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

    //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;
    }

    bool search(int x, int y)
    {
        if (x == 8 && y == 8)
        {
```

```
            if (maps[x][y])
                return true;
            for (int i = 0; i < 9; i++)
            {
                if (!domain[x][y][i])
                {
                    maps[x][y] = i + 1;
                    return true;
                }
            }
            return false;
        }
        else
        {
            int ny = y + 1;
            int nx = x;
            if (y == 8)
            {
                ny = 0;
                nx = x + 1;
            }
            if (maps[x][y])
                return search(nx, ny);
            for (int i = 0; i < 9; i++)
            {
                if (!domain[x][y][i])
                {

                    if (set(x, y, i + 1, 1) && search(nx, ny))
                    {
                        return true;
                    }
                    set(x, y, i + 1, 0);
                }
            }
            return false;
        }
    }
};

int main()
{
    FutoshikiPuzzle *futoshikiPuzzle = new FutoshikiPuzzle();
    futoshikiPuzzle->initial();
    futoshikiPuzzle->show();
    cout << futoshikiPuzzle->search(0, 0) << endl;
    futoshikiPuzzle->show();
}
```

4、Results





以上是在Linux下进行的测试，ori是原文件的测试，运行超过5s，my是上方文件的测试，运行仅0.671s，效果拔群。