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

E04 Futoshiki Puzzle (Forward Checking)

17341189 姚森舰

2019 年 9 月 22 日

目录

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/.

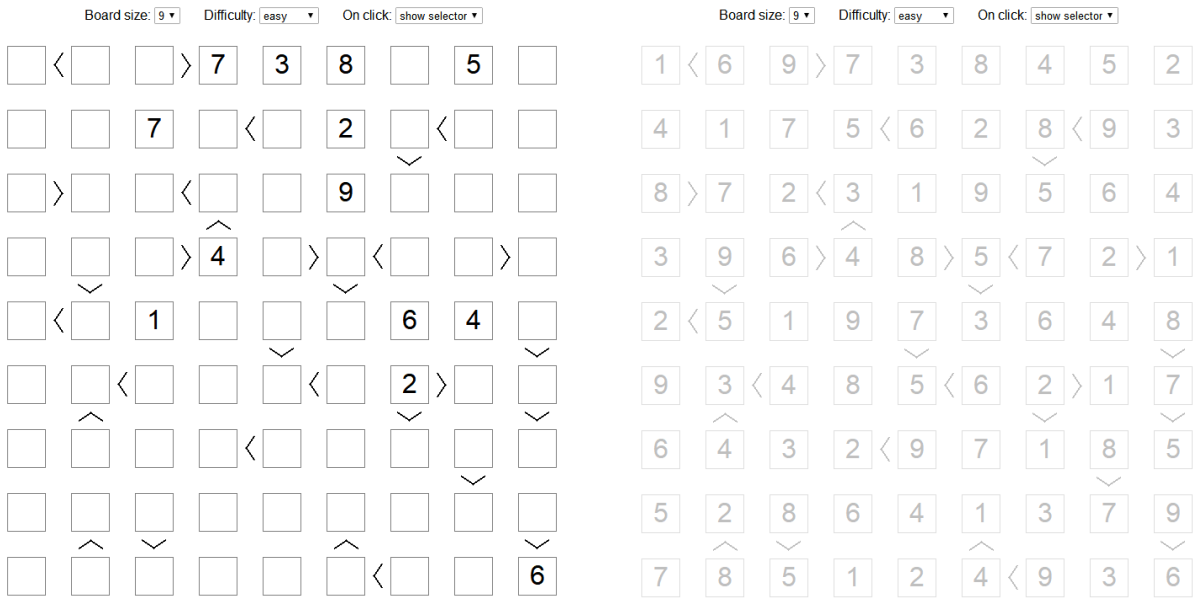


图 1: An Futoshiki Puzzle

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

```
1
#include <iostream>
```

```cpp
#include <fstream>
#include <vector>
#include <map>


using namespace std;
#define N 9
int board[N][N] = {0};
bool assigned[N][N] = {false};          // 记录某一个格子是否已被赋值

int num_of_constrains = 0;

// 限制条件，两个坐标，前者大于后者
struct constraint{
    // x1y1 > x2y2
    int x1,y1;
    int x2,y2;
};

// 坐标
struct coordinate{
    int x;
    int y;
    // 重载 < 是为了 map 排序
    bool operator < (const coordinate &a) const
    {
        // 必须保证每个不同
        return x*10 + y < a.x*10 + a.y;
    }
};

map<coordinate, vector<int> > domain;               // 每个格子的值域
vector<constraint> constraints;                     // 所有的限制条件


// 列检查，去除该列的格子的值域中 与 cur_value 值相同的值
bool col_test(int r, int c, int cur_value){
    for(int j = 0; j < N; j++){
        if(!assigned[j][c]){                        // 首先保证该点还没有赋值
            vector<int>::iterator it;
            coordinate test{j,c};                   // 生成坐标，查map来查值域
            for (it = domain[test].begin(); it != domain[test].end(); ){
                if(*it == cur_value){
                    domain[test].erase(it);         //后面元素前移
                    if(domain[test].size() == 0) return 0;
                }
                else{
```

```
                    it++;
51                }
            }
53        }
    }
55    return 1;
}
57
// 行检查，去除该行的格子的值域中 与 cur_value 值相同的值
59bool row_test(int r, int c, int cur_value){
    for(int j = 0; j < N; j++){
61        if(!assigned[r][j]){
            vector<int>::iterator it;
63            coordinate test{r,j};                    //生成坐标
            for (it = domain[test].begin(); it != domain[test].end(); ){
65                if(*it == cur_value){
                    domain[test].erase(it);          //后面元素前移
67                    if(domain[test].size() == 0) return 0;
                }
69                else{
                    it++;
71                }
            }
73
        }
75    }
    return 1;
77}

79// 限制条件的检查， 输入的坐标是应该较大的那个格子的坐标
bool constraint_test1(int x, int y, int value){
81    coordinate test{x,y};
    vector<int>::iterator it;
83    for (it = domain[test].begin(); it != domain[test].end(); ){
        if(value <= *it){                        // cur_vlaue 要大于后面的，所以删除大的
85            domain[test].erase(it);              //后面元素前移
            if(domain[test].size() == 0) return 0;
87        }
        else{
89            it++;
        }
91    }
    return 1;
93}

95// 限制条件的检查， 输入的坐标是应该较小的那个格子的坐标
bool constraint_test2(int x, int y, int value){
```

```
97      coordinate test{x,y};
        vector<int>::iterator it;
99      for (it = domain[test].begin(); it != domain[test].end(); ){
            if(value >= *it){                    // cur_vlaue 要大于后面的, 所以删除大的
101             domain[test].erase(it);              //后面元素前移
                if(domain[test].size() == 0) return 0;
103         }
            else{
105             it++;
            }
107     }
        return 1;
109 }

111 // 读文件生成最初的board, 并对部分值域做初步限制
    int read_board(string filename){
113     ifstream data;
        int r = 0, c = 0, num =0;
115     int n = 0;
        data.open(filename);
117     while(!data.eof()){
            data >> r >> c >> num;
119         r--;           // 转换为数组下标
            c--;
121         assigned[r][c] = true;
            board[r][c] = num;
123         n++;
            // 行检查
125         bool test_row = row_test(r,c,num);

127         // 列检测
            bool test_col = col_test(r,c,num);
129     }

131     data.close();
        return n;
133 }

135 // 读取大于小于限制条件, 并保存到一个vector中
    void read_constraints(string filename, vector<constraint> & v){
137     ifstream data;
        int x1,y1,x2,y2;
139     data.open(filename);
        while(!data.eof()){
141         data >> x1 >> y1 >> x2 >> y2;
            x1--;           // 转换为数组下标
143         y1--;
```

```
            x2--;
            y2--;
            constraint c{x1,y1,x2,y2};
            v.push_back(c);
            // 如果x1, y1已经取值而x2, y2还未取值
            if( (assigned[x1][y1]) && (!assigned[x2][y2]) ){
               bool t = constraint_test1(x2,y2,board[x1][y1]);
            }
             // 如果x2, y2已经取值而x1, y1还未取值
            if( (assigned[x2][y2]) && (!assigned[x1][y1]) ){
                bool t = constraint_test2(x1,y1,board[x2][y2]);
            }

        }
        num_of_constrains = v.size();
        data.close();
}

// 输出board
void print_board(){
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            cout << board[i][j] << " ";
        }
        cout << endl;
    }
}

// 判断是否找出答案
bool is_solved(){
    int n = 0;
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            if(assigned[i][j]) n++;
        }
    }
    if(n == N*N) return true;
    else return false;
}

// 输出每个格子的值域, debug用
void print_domain(map<coordinate, vector<int> > domain_n){
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            coordinate c{i,j};
            int len = domain_n[c].size();
            cout<<"["<<i<<","<<j<<"]"<<":";
```

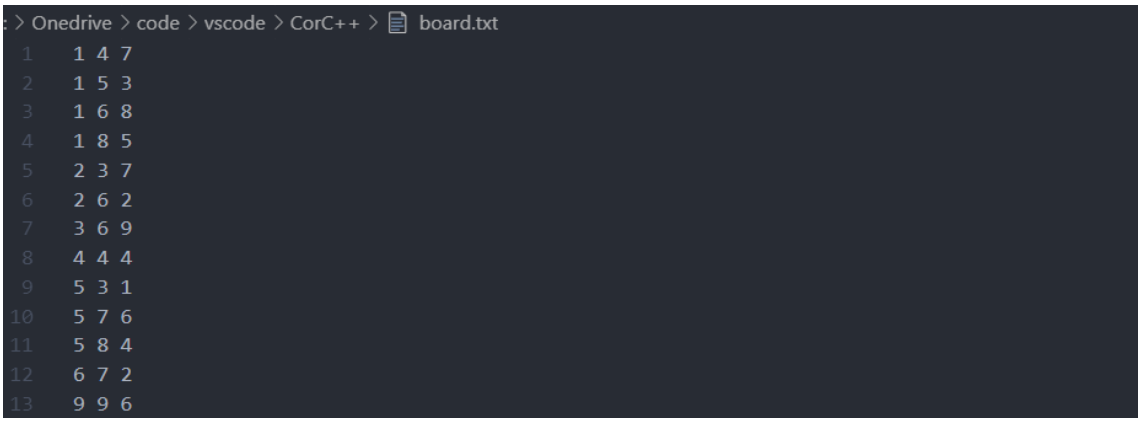
```
191             for(int k = 0; k < len; k++){
                    cout<<domain_n[c][k]<<" ";
193             }
                cout<<endl;
195         }
    }
197 }

199 // 输出所有格子是否被赋值， debug用
    void print_assigned(){
201     for(int i = 0; i < N; i++){
            cout<<"*";
203         for(int j = 0; j < N; j++){
               cout<<assigned[i][j]<<" ";
205         }
            cout<<endl;
207     }
    }
209

211 // 用MRV来选择下一个赋值的格子，返回其坐标
    coordinate MRV(){
213     int min_i = 0, min_j =0, min = 9999;
        for(int i = 0; i < N; i++){
215         for(int j = 0; j < N; j++){
                if(!assigned[i][j]){                    //  首先是要未确定的点
217                 coordinate c{i,j};
                    int num = domain[c].size();         // 可取值的数的个数最少
219                 if(num < min ){
                        min = num;
221                     min_i = i;
                        min_j = j;
223                 }
                }
225         }
        }
227     coordinate min_domain{min_i,min_j};
        return min_domain;
229 }

231 // 向前检查，去掉不和要求的取值
    bool FCCheck(coordinate now, int cur_value){
233     bool satisfied = true;
        int r = now.x;
235     int c = now.y;
         // 行检查
237     bool test_row = row_test(r,c,cur_value);
```

```
    if(!test_row) return false;
    // 同列检测
    bool test_col = col_test(r,c,cur_value);
    if(!test_col) return false;
    // 限制条件检查
    for(int j = 0; j < num_of_constrains; j++){
        int x1 = constraints[j].x1;
        int y1 = constraints[j].y1;
        int x2 = constraints[j].x2;
        int y2 = constraints[j].y2;
        bool t = true;
        if(x1 == now.x && y1 == now.y && (!assigned[x2][y2])){
            t = constraint_test1(x2,y2,cur_value);
        }
        if(!t) return false;
        if(x2 == now.x && y2 == now.y && (!assigned[x1][y1])){
            t = constraint_test2(x1,y1,cur_value);
        }
        if(!t) return false;

    }
    return true;        //有值满足
}



void FC(){
    if(is_solved()){
        print_board();
        return;
    }
    coordinate now = MRV();                    // 得到下一个赋值的格子坐标
    map<coordinate, vector<int> > domain_copy = domain;       // 暂存目前所有格子的值域
    assigned[now.x][now.y] = true;
    int temp = board[now.y][now.y];
    vector<int> curDom = domain[now];
    int len = curDom.size();
    for(int i = 0; i < len; i++){
        domain = domain_copy;                  // 每次对本格子赋值时都要恢复值域
        board[now.x][now.y] = curDom[i];
        bool exist_satisfy_all = FCCheck(now, curDom[i]);
        if(!exist_satisfy_all){
            board[now.y][now.y] = temp;
            continue;
        }
        else{
            FC();
```

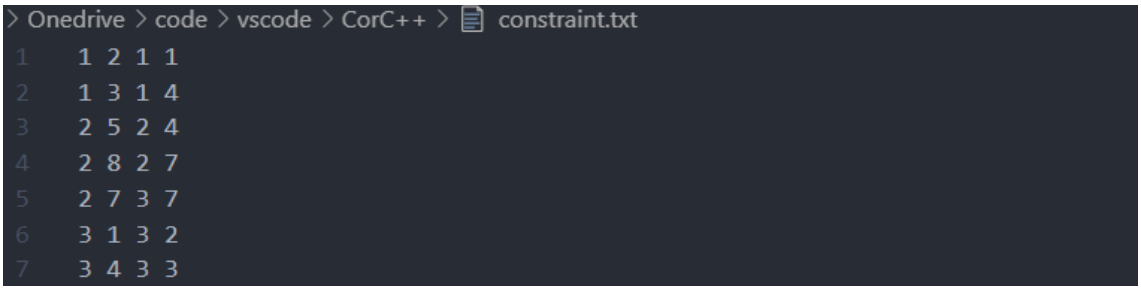
```
285             }
        }
        assigned[now.x][now.y] = false;          // 恢复未赋值状态
        return;

}


int main(){
    int num_assigned = 0;
    vector<int> domain_of_each_tile;
    // 每个格子初始值域 1-9
    for(int i = 1; i <= N; i++) {
        domain_of_each_tile.push_back(i);
    }
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            coordinate c{i,j};
            domain[c] = domain_of_each_tile;
        }
    }

    num_assigned = read_board("board.txt");
    read_constraints("constraint.txt", constraints);
    FC();
}
```

4 Results

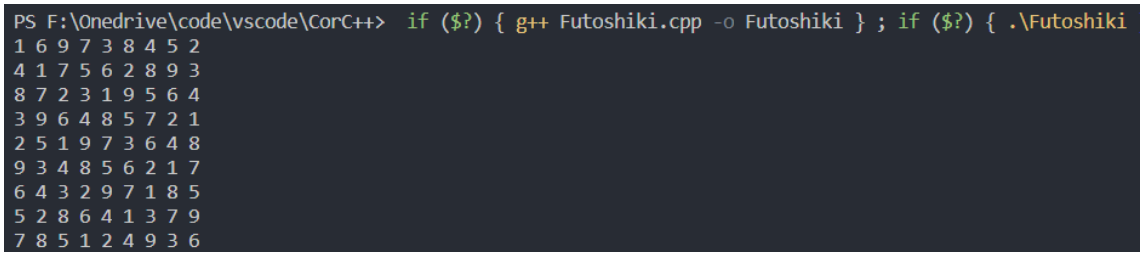
需要注意的是，我将问题条件写入了文件，即需将 board 中预先设定的点，按每一行 row，col，value 的顺序写到“board.txt”中，比如 1 4 7 就表示第 1 行第 4 列的值为 7，见下图:



将所有的限制按每行row1，col1，row2，col2 的顺序写到“constraint.txt”文件中，比如，1 2 1 1 就表示第1 行第2 列的值要大于第1 行第1 列的值，见下图:



下面是输出结果:



可以看到结果正确。

这次的实验原理不算太难，但是实现却花了我比较多的时间。听同学说用 Python 来做，跑程序需要很多分钟，于是便使用了 C++，最后跑一次大概 20 秒，可能是用了 STL 的东西，所以也不是特别快。但是 C++ 写起来确实难度更大一些。

实现时要注意的问题主要是及时保存格子的值域并且在正确的位置恢复，还要特别注意一下初始就赋了值而且附近还有大小约束的这种格子。

此外，在使用 map 时，可以用自己写的结构体作为键值，只不过要重载一下”<”, 因为 map 是用红黑树实现的，要保证每个键之间能比较大小或者顺序。