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

E4 Futoshiki Puzzle (Forward Checking)

18340013 Conghao Chen

September 21, 2020

Contents

1 Futoshiki

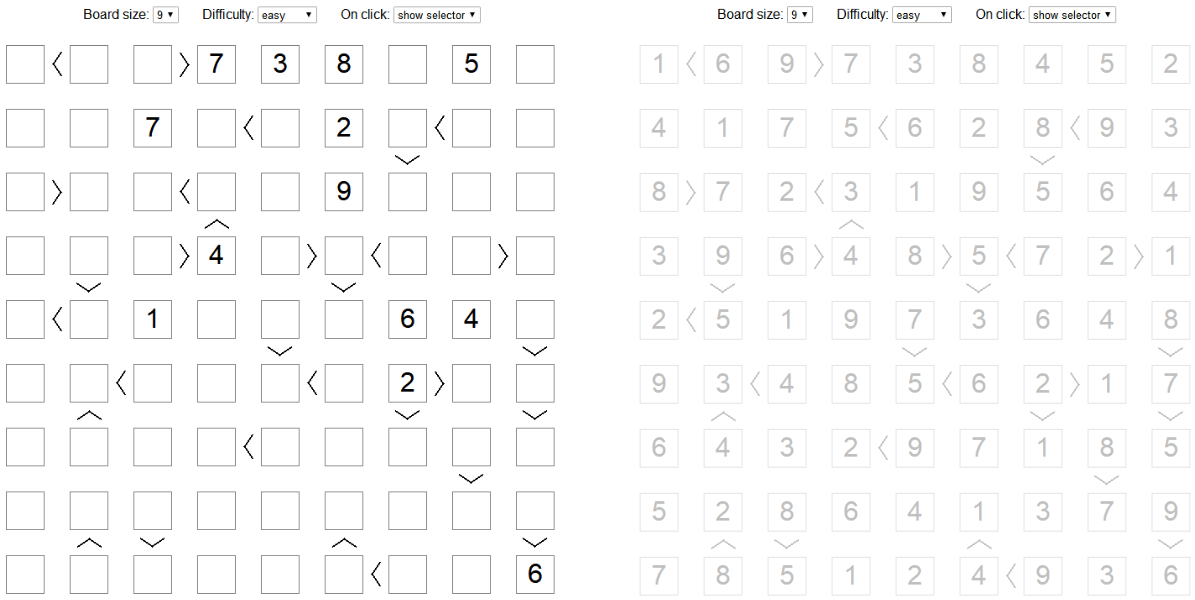
Futoshiki is a board-based puzzle game, also known under the name Unequal. It is playable on a square board having a given fixed size (4 × 4 for example).

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/.



Board size: 9 ▾ Difficulty: easy ▾ On click: show selector ▾

Board size: 9 ▾ Difficulty: easy ▾ On click: show selector ▾

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_2020@foxmail.com.

3 Codes

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

using namespace std;

struct my_tuple{
```

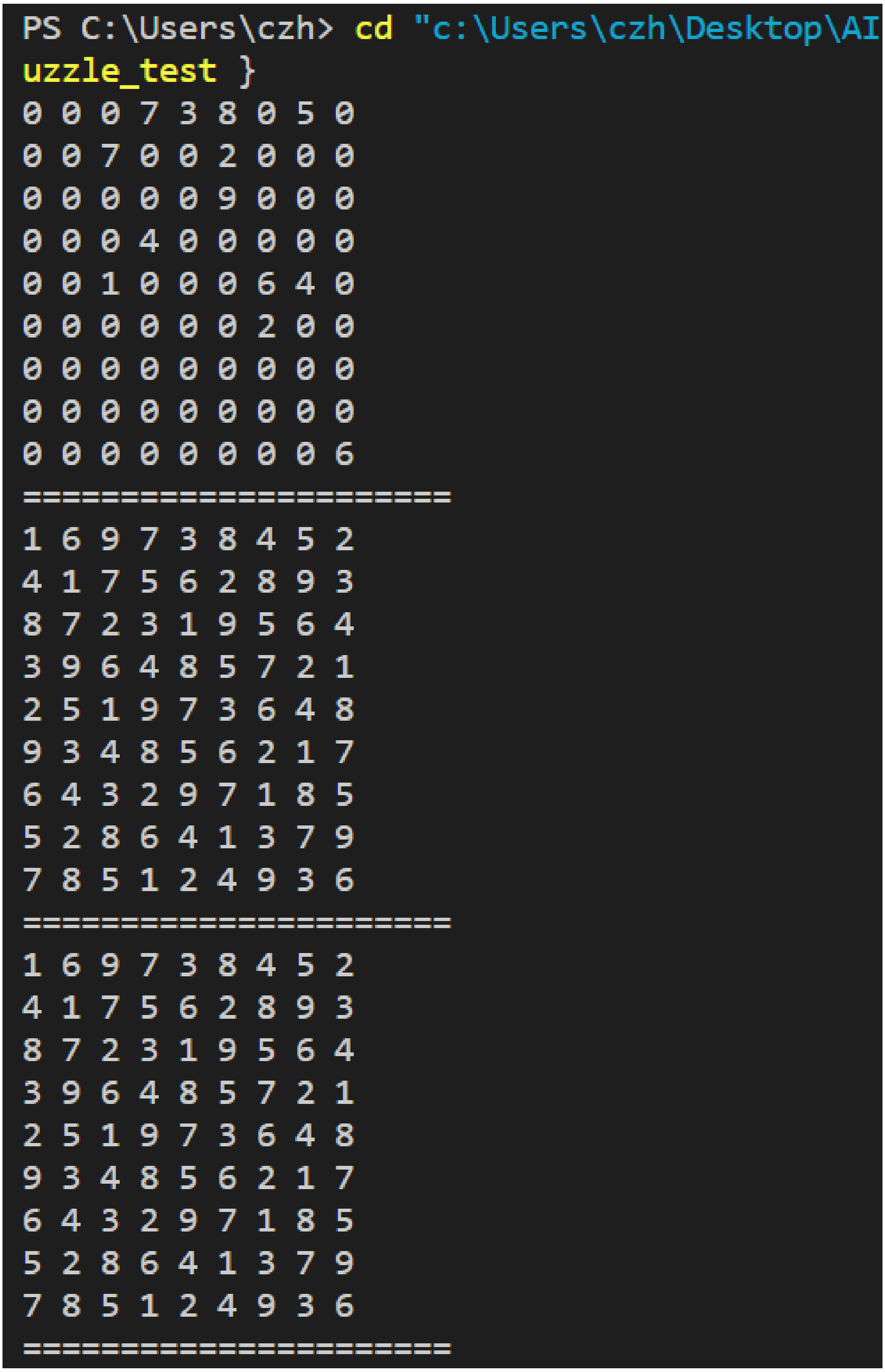
```
int x;
int y;
int value;
};

class FutoshikiPuzzle{
public:
vector<vector<int>> maps;
vector<pair<pair<int, int>, pair<int, int>>> less_constraints;
int nRow, nColumn;
//表示第x行中某个数字是否存在
int Count_RowNumbers[9][10];
//表示第y列某个数字是否存在
int Count_ColumnNumbers[9][10];
int total = 0;
//表示(x,y)点value值是否因FC被剪枝
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);
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);

//初始化域
memset(is_pruned, 0, sizeof(is_pruned));
for (int x = 0; x < 9; x++){
for (int y = 0; y < 9; y++){
    int i = maps[x][y];
    if (i){
        Count_RowNumbers[x][i]++;
        Count_ColumnNumbers[y][i]++;
        for (int row_or_col = 0; row_or_col < 9; row_or_col++)
        {
            if (row_or_col != x)
            {
                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;
}

void find_next(int &next_x, int &next_y)
{
for (next_x = 0; next_x < 9; next_x++)
{
    for (next_y = 0; next_y < 9; next_y++)
    {
        if (!maps[next_x][next_y])
        {
            goto next;
        }
    }
}
next:
int temp_x, temp_y;
for (temp_x = next_x, temp_y = next_y + 1; temp_y < 9; temp_y++)
{
    if (!maps[temp_x][temp_y] && pruned_num[next_x][next_y] <
        pruned_num[temp_x][temp_y])
    {
        next_y = temp_y;
    }
}
for (temp_x = next_x + 1; temp_x < 9; temp_x++)
{
    for (temp_y = 0; temp_y < 9; temp_y++)
    {
        if (!maps[temp_x][temp_y] && pruned_num[next_x][next_y] <
            pruned_num[temp_x][temp_y])
        {
            next_x = temp_x;
            next_y = temp_y;
        }
    }
}
}

bool search(int x, int y)
{
if (maps[x][y] == 0)
{
total++;
for (int i = 1; i < 10; i++)
{
    maps[x][y] = i;
    Count_RowNumbers[x][i]++;
    Count_ColumnNumbers[y][i]++;
    if (check(x, y))
    {
        if (x == 8 && y == 8)
        {
            return true;
        }
        int next_x, next_y;
        if (y != 8)
        {
            next_x = x;
            next_y = y + 1;
        }
        else
        {
            next_x = x + 1;
            next_y = 0;
        }

        if (search(next_x, next_y))
        {
            return true;
        }
    }
    maps[x][y] = 0;
    Count_RowNumbers[x][i]--;
    Count_ColumnNumbers[y][i]--;
}
}
else
{
if (x == 8 && y == 8)
{
    return true;
}
int next_x, next_y;
if (y != 8)
{
    next_x = x;
    next_y = y + 1;
}
else
{
    next_x = x + 1;
    next_y = 0;
}

if (search(next_x, next_y))
{
    return true;
}
}
return false;
}

bool FC_search(int x, int y)
{
total++;
my_tuple back;
for (int i = 1; i < 10; i++)
{
if (!is_pruned[x][y][i])
```

```
{
        maps[x][y] = i;
        Count_RowNumbers[x][i]++;
        Count_ColumnNumbers[y][i]++;
        if (check(x, y))
        {
        int restore_num = 0;
        for (int row_or_col = 0; row_or_col < 9; row_or_col++)
        {
                if (!maps[row_or_col][y] && !is_pruned[row_or_col][y][i])
                {
                        is_pruned[row_or_col][y][i] = 1;
                        pruned_num[row_or_col][y]++;
                        restore.push_back({row_or_col, y, i});
                        restore_num++;
                }
                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});
                                                restore_num++;
                                        }
                                }
                        }
                }
        }
        int next_x, next_y;
        find_next(next_x, next_y);
        if (next_x == 9)
        {
                return true;
        }
        if (FC_search(next_x, next_y))
        {
                return true;
        }
        while (restore_num--)
        {
                back = restore.back();
                is_pruned[back.x][back.y][back.value] = 0;
                pruned_num[back.x][back.y]--;
                restore.pop_back();
        }
}
}
maps[x][y] = 0;
Count_RowNumbers[x][i]--;
Count_ColumnNumbers[y][i]--;
}
return false;
}
};

int main()
{
FutoshikiPuzzle *futoshikiPuzzle = new FutoshikiPuzzle();
futoshikiPuzzle->initial();
futoshikiPuzzle->show();
futoshikiPuzzle->search(0, 0);
futoshikiPuzzle->show();
delete futoshikiPuzzle;
futoshikiPuzzle = new FutoshikiPuzzle();
futoshikiPuzzle->initial();
int next_x, next_y;
futoshikiPuzzle->find_next(next_x, next_y);
futoshikiPuzzle->FC_search(next_x, next_y);
futoshikiPuzzle->show();
}
```

4 Results

第一个输出是所给题目，第二个输出是使用普通搜索得到的结果，第三个输出是使用 FC 剪枝算法得到的结果，可以看到结果是正确的：



```
PS C:\Users\czh> cd "c:\Users\czh\Desktop\AI
uzzle_test }
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
======================
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
======================
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
======================
```