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

E04 Futoshiki Puzzle (Forward Checking)

17341111 Xuehai Liu

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/`.

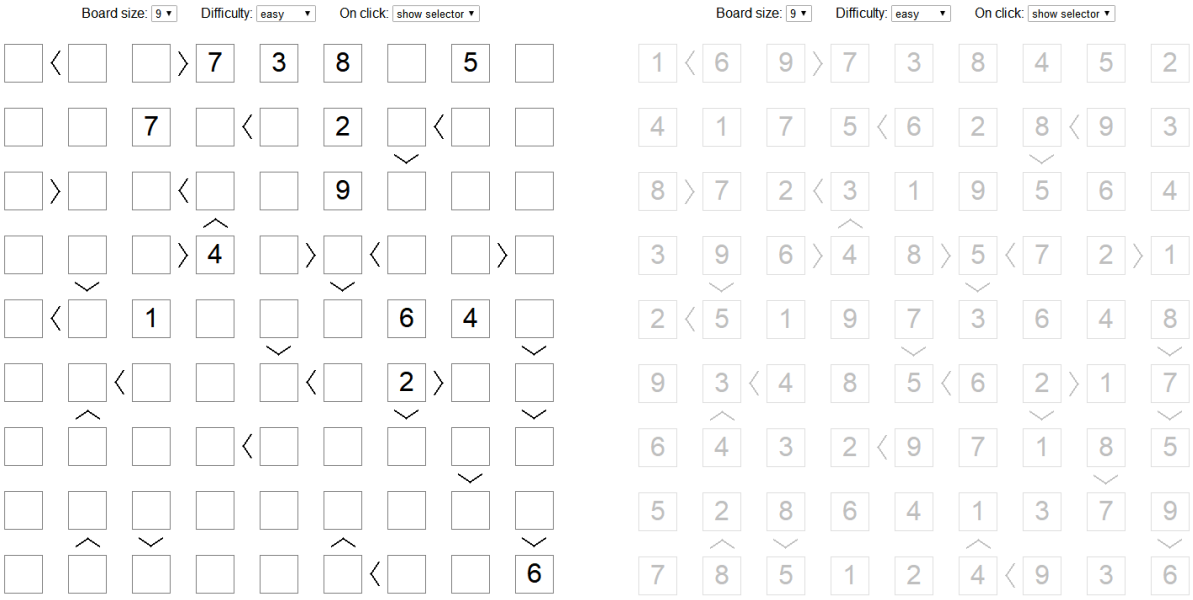

图 1: An Futoshiki Puzzle

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

```
#include <bits/stdc++.h>
#include <windows.h>
using namespace std;

int puzzle[10][10]={0};//存储每个位置实际的值
int values[10][10][10]={1};//存储每个位置可能的值，初始化时全部可能为1，不可能则为
int constraints[10][10][8]={0};//0-7 8 种约束条件
//0-7分别小于左，大于左，小于右，大于右，小于上，大于上，小于下，大于下

int row, col;
int n;

//更新节点的函数，包含对节点(x,y)赋值与对所有与其相关的约束条件更新
void updateNode(int value,int x,int y)
{
    puzzle[x][y] = value;
    for (int i=1;i<=col;i++)
    {
        values[x][i][value] = 0;
    }
    for (int i=1;i<=row;i++)
    {
        values[i][y][value] = 0;
    }
    //上两句对行列alldif进行更新，同行列不能与其相等。
    values[x][y][value] = 1;
    for (int i=0;i<8;i++)
    {
        //对八种约束条件进行检查并更新周围节点的取值范围
        if(constraints[x][y][i]==1)
        {
```

```
switch(i)
{
case 0:
    for (int i=1;i<=value;i++)
        values[x][y-1][i] = 0 ;
    break;
case 1:
    for (int i=value;i<=n;i++)
        values[x][y-1][i] = 0 ;
    break;
case 2:
    for (int i=1;i<=value;i++)
        values[x][y+1][i] = 0 ;
    break;
case 3:
    for (int i=value;i<=n;i++)
        values[x][y+1][i] = 0 ;
    break;
case 4:
    for (int i=1;i<=value;i++)
        values[x-1][y][i] = 0 ;
    break;
case 5:
    for (int i=value;i<=n;i++)
        values[x-1][y][i] = 0 ;
    break;
case 6:
    for (int i=1;i<=value;i++)
        values[x+1][y][i] = 0 ;
    break;
case 7:
    for (int i=value;i<=n;i++)
        values[x+1][y][i] = 0 ;
```

```
                break;
            }
        }
    }
}

//判定终止状态
int allAssigned()
{
    for (int i=1;i<=row;i++)
        for (int j=1;j<=col;j++)
    {
        if(!puzzle[i][j])
            return 0;
    }
    return 1;
}
//输出values数组
void printval()
{
    for (int i=1;i<=row;i++){
        for (int j=1;j<=col;j++){
            cout<<i<<","<<j<<":"<<endl;
            for(int k=1;k<=n;k++)
            {
                if(values[i][j][k])
                    cout<<k<<"␣";
            }
            cout<<endl;
        }
    }
}
```

```
//用MRV的策略选择一个index
int MRVPick()
{
    int minnum = 10;
    int minIndex = 0;
    for (int i=1;i<=row;i++)
        for (int j=1;j<=col;j++)
    {
        int count = 0;
            for (int k=1;k<=n;k++)
                if(values[i][j][k])
                    count++;
        if(count<minnum & !puzzle[i][j])
        {
            minnum = count;
            minIndex = (i-1)*n + j;
        }
    }
    return minIndex;
}

//向前检测的递归搜索函数。
void FC(int level)
{
    if(allAssigned())
    {
        for (int i=1;i<=row;i++){
            for (int j=1;j<=col;j++)
                cout<<puzzle[i][j]<<" ";
            cout<<endl;
        }
        exit(0);
    }
```

```
//使用一个栈内的小数组记录对应当时状态的取值范围。
int oldValues[10][10][10];
memcpy(oldValues,values,sizeof(values));
int MinIndex = MRVPick();
if(MinIndex == 0)
    return;
//获取MRV得到的行与列
int c = (MinIndex-1) % n + 1;
int r = (MinIndex-1) / n + 1;

//接下来对每一个该位置可能的取值分析
for (int i=1;i<=n;i++)
    if(values[r][c][i])
{
    int d = i;
    //使用这个值更新节点
    updateNode(d,r,c);
    bool DWOoccured = false;
    //接下来作FCcheck。检查是否有因为这个点的更新而失去所有取值可能的点。
    for (int i=1;i<=row;i++)
    {
        int count = 0;
        for (int k=1;k<=n;k++)
            if(values[i][c][k])
                count++;
        if(!count)
        {
            DWOoccured = true;
            break;
        }
    }
    for (int j=1;j<=col;j++)
    {
```

```
            int count = 0;
            for (int k=1;k<=n;k++)
                if(values[r][j][k])
                    count++;
            if(!count)
            {
                DWOoccured = true;
                break;
            }
        }
        //如果无事发生
        if(!DWOoccured)
        {
            //cout<<"the point " << r<<","<<c<<" is chosen "<<"as "<< d<<endl;
            FC(level + 1);
        }
        //最后恢复到遍历此节点前的状态，恢复剪枝。
        memcpy(values,oldValues,sizeof(oldValues));
    }
    puzzle[r][c] = 0;
    //最后要把这个点的值取消。因为已遍历所有可能
    return;
}

int main()
{
    cin>>n;
    row = col = n;
    for (int i=1;i<10;i++)
        for(int j=1;j<10;j++)
            for (int k=1;k<10;k++)
                values[i][j][k]=1;
```

```
    int numC = 0;
    cin>>numC;
    for (int i=0;i<2*numC;i++)
    {
        int x,y,t;
        cin>>x>>y>>t;
        constraints[x][y][t] = 1;

    }
    for (int i=1;i<=row;i++)
        for (int j=1;j<=col;j++)
    {
        int value;
        cin>>value;
        if(value)
            updateNode(value,i,j);
    }
    cout<<endl;
    FC(0);
    Sleep(100000);

}
```

4 Results

图 2: Input futoshiki

The result is perfectly matched to the answer in Figure 1.

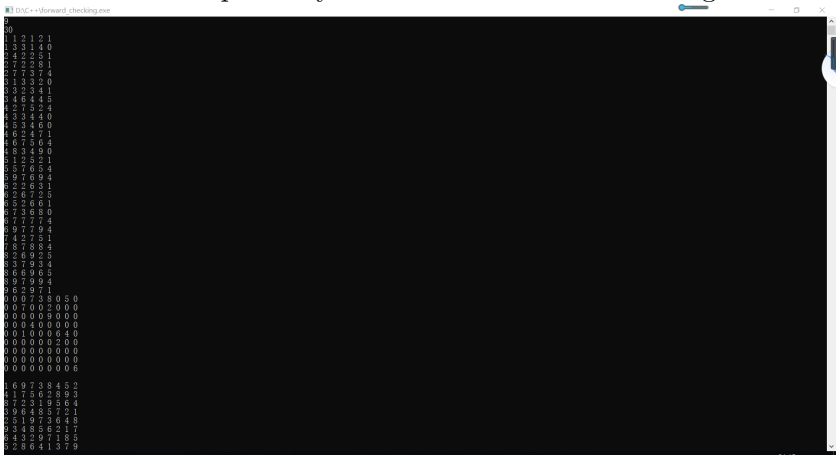

图 3: My Result