E04 Futoshiki Puzzle (Forward Checking)

学号	姓名	日期
18340052	何泽	2020.09.24

目录

I Futoshiki

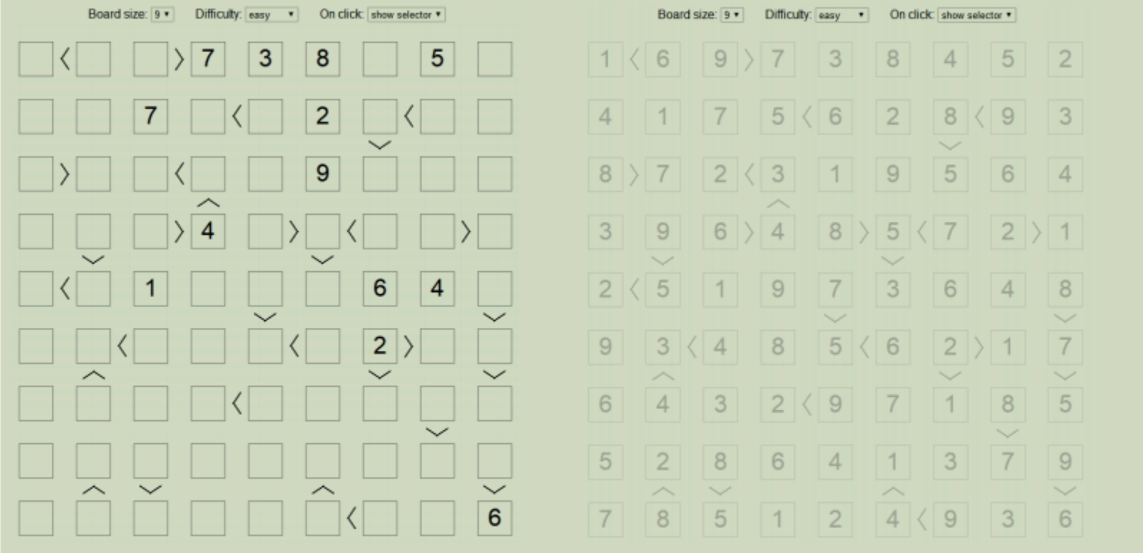
Futoshiki is a board-based puzzle game, also known under the name Unequal. It is playable on a square board having a given fixed size (4 4 for example).

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

II 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 ai2020@foxmail.com.

III Codes

> 因为我在写的时候忘记已经给了框架，所以我没有在提供的代码上修改，而是自己写的框架

- 首先，我将每个元素都定义为一个结构体

```
struct item {
  int row,col,val;
  int l, r, u, d;
  bool domain[9];
  bool assigned;
};
```

  - `row,col,val` 代表行数，列数，元素值

    - `l,r,u,d` 分别代表地图上四个方向是否有大小关系，值为1代表大于，-1代表小于
    - `domain` 代表当前元素可行域
    - `assigned` 代表当前元素是否被赋值

- 整个地图就是如下定义：

```
item board[9][9];
```

- 计算当前元素的可行解数量：

```
int domain_count(item* m) {
  int a = 0;
  for (int i = 0;i<9;i++)
    a += m->domain[i];
  return a;
}
```

- 检测当前元素所在行是否有重复元素

```
bool row_check(futoshiki* board, item* m) {
  int row[9];
  for(int i = 0;i<9;i++)
    row[i] = board->board[m->row][i].val;
  sort(row, row + 9);
  for(int i = 0;i<8;i++) {
    if (!row[i])
      continue;
    if (row[i] == row[i+1])
      return false;
  }
  return true;
}
```

- 检测当前元素所在列是否有重复元素

```
bool col_check(futoshiki* board, item* m) {
  int col[9];
  for(int i = 0;i<9;i++)
    col[i] = board->board[i][m->col].val;
  sort(col, col + 9);
  for(int i = 0;i<8;i++) {
    if (!col[i])
      continue;
    if (col[i] == col[i+1])
      return false;
  }
  return true;
}
```

- 检测当前是否满足地图的大小关系

```
bool check(futoshiki* board, item* m) {
  int v = m->val;
  if (m->u && board->board[m->row - 1][m->col].assigned) {
    if (m->u == -1 && v > board->board[m->row - 1][m->col].val)
      return false;
    else if (m->u == 1 && v < board->board[m->row - 1][m->col].val)
      return false;
  }
  if (m->d && board->board[m->row + 1][m->col].assigned) {
    if (m->d == -1 && v > board->board[m->row + 1][m->col].val)
      return false;
    else if (m->d == 1 && v < board->board[m->row + 1][m->col].val)
      return false;
  }
  if (m->l && board->board[m->row - 1][m->col].assigned) {
    if (m->l == -1 && v > board->board[m->row][m->col - 1].val)
      return false;
```

```c
    else if (m->l == 1 && v < board->board[m->row][m->col - 1].val)
      return false;
  }
  if (m->r && board->board[m->row - 1][m->col].assigned) {
    if (m->r == -1 && v > board->board[m->row][m->col + 1].val)
      return false;
    else if (m->r == 1 && v < board->board[m->row][m->col + 1].val)
      return false;
  }
  return true;
}
```

- 根据功能号c的值判断某一行、某一列或者上下左右的元素是否被赋值

```c
bool assign_check(futoshiki* board, int c, item* m) {
  int R = m->row, C = m->col, s = 0;
  if (c == 0) {
    for (int i = 0;i<9;i++)
      s += board->board[R][i].assigned;
    return (s == 8);
  }
  else if (c == 1) {
    for (int i = 0;i<9;i++)
      s += board->board[i][C].assigned;
    return (s == 8);
  }
  else if (c == 2) {
    s += (R == 0) ? 1 : board->board[R - 1][C].assigned;
    s += (R == 8) ? 1 : board->board[R + 1][C].assigned;
    s += (C == 0) ? 1 : board->board[R][C - 1].assigned;
    s += (C == 8) ? 1 : board->board[R][C + 1].assigned;
    return (s == 8);
  }
  return false;
}
```

- 向前检测，同样是根据功能号c的值检测某一行、某一列或者上下左右的元素

```
bool fc_check(futoshiki* board, int c, item* m) {
  if (c == 0)
    for (int i = 0;i<9;i++) {
      if (!m->domain[i]) {
        m->domain[i] = 1;
        if(row_check(board, m))
          m->domain[i] = 0;
      }
    }
  else if (c == 1)
    for (int i = 0;i<9;i++) {
      if (!m->domain[i]) {
        m->domain[i] = 1;
        if(col_check(board, m))
          m->domain[i] = 0;
      }
    }
  else if (c == 2)
    for (int i = 0;i<9;i++) {
      if (!m->domain[i]) {
        m->domain[i] = 1;
        if(check(board, m))
          m->domain[i] = 0;
      }
    }
  if (domain_count(m) == 9)
    return false;
  else
    return true;
}
```

- MRV函数

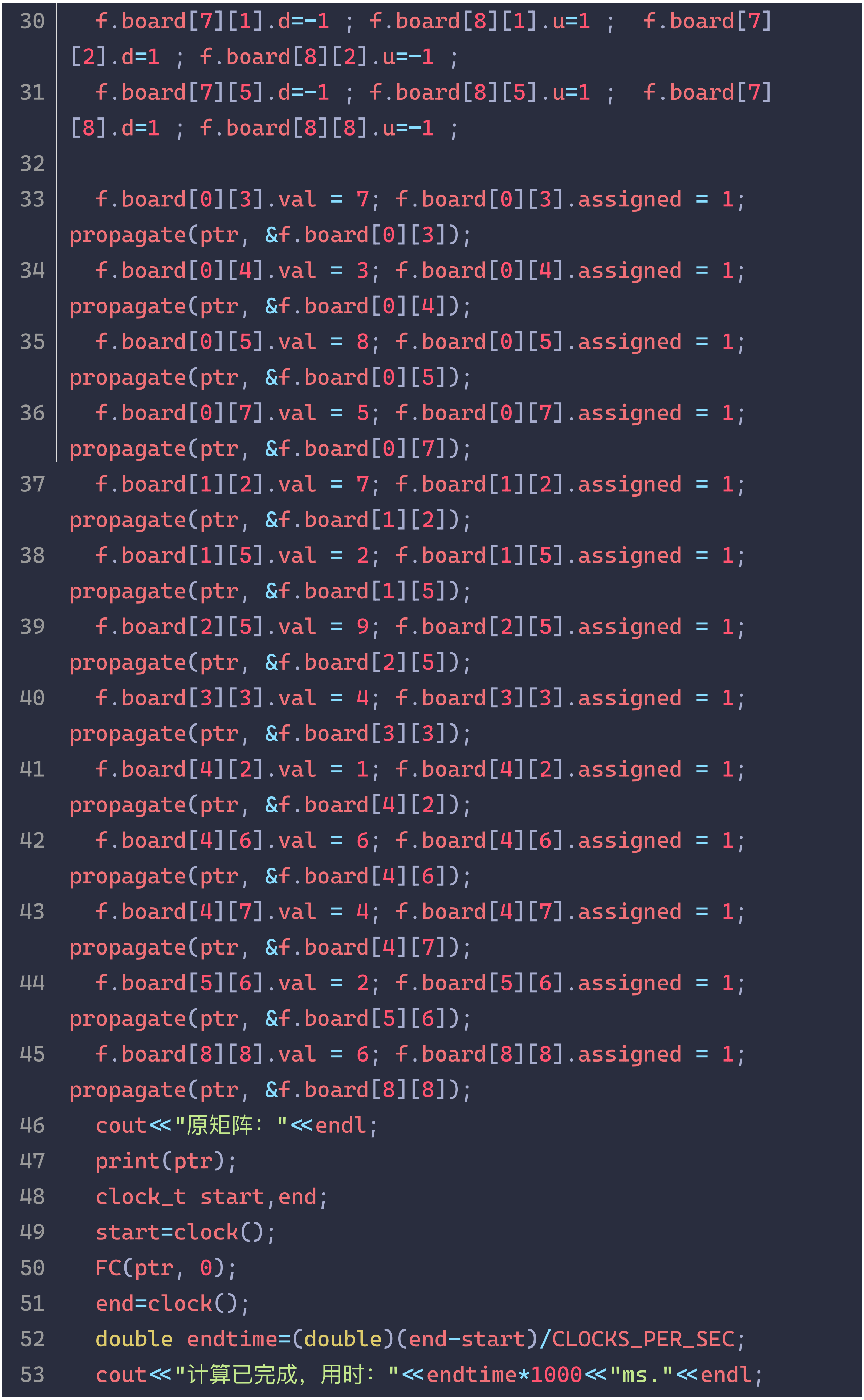
```
item* heuristicpick(futoshiki* board) {
  item* maxi = &board->board[0][0];
  for (int i = 0;i<9;i++) {
    for (int j = 0;j<9;j++) {
      if(board->board[i][j].assigned)
        continue;
      if(domain_count(maxi) < domain_count(&board->board[i][j]) || maxi->assigned) {
        maxi = &board->board[i][j];
        if (domain_count(maxi) == 8)
          return maxi;
      }
    }
  }
  return maxi;
}
```

- 遍历过程

```
bool FC(futoshiki* board, int level) {
  if (is_finished(board)) {
    cout<<"完成后矩阵："<<endl;
    print(board);
    return true;
  }
  item* v = heuristicpick(board);
  v->assigned = true;
  bool dwo = false;
  int pos = 0;
  for (int i = 0;i<9;i++)
    if (!v->domain[i]) {
      futoshiki boardcopy;
      memcpy(&boardcopy, board, sizeof(futoshiki));
      v->val = i+1;
      propagate(board, v);
      dwo = false;
      if (!dwo && assign_check(board, 0, v)) {
        for (int i = 0;i<9;i++)
          if (!board->board[v->row][i].assigned)
            dwo = !fc_check(board, 0, &board->board[v->row][i]);
      }
      if (!dwo && assign_check(board, 1, v)) {
        for (int i = 0;i<9;i++)
          if (!board->board[i][v->col].assigned)
            dwo = !fc_check(board, 1, &board->board[i][v->col]);
      }
      if (!dwo && assign_check(board, 2, v)) {
        if (v->row && board->board[v->row - 1][v->col].assigned)
          dwo = !fc_check(board, 2, &board->board[v->row - 1][v->col]);
        else if (v->row!=8 && board->board[v->row + 1][v->col].assigned)
          dwo = !fc_check(board, 2, &board->board[v->row + 1][v->col]);
        else if (v->col && board->board[v->row][v->col - 1].assigned)
          dwo = !fc_check(board, 2, &board->board[v->row][v->col - 1]);
        else if (v->col!=8 && board->board[v->row][v->col + 1].assigned)
          dwo = !fc_check(board, 2, &board->board[v->row][v->col + 1]);
      }
      if(!dwo && FC(board, level + 1))
        return true;
      memcpy(board, &boardcopy, sizeof(futoshiki));
    }
  v->assigned = false;
  return false;
}
```

- 主函数，前面是对初始地图元素及大小关系的赋值，后面是调用FC函数并计时

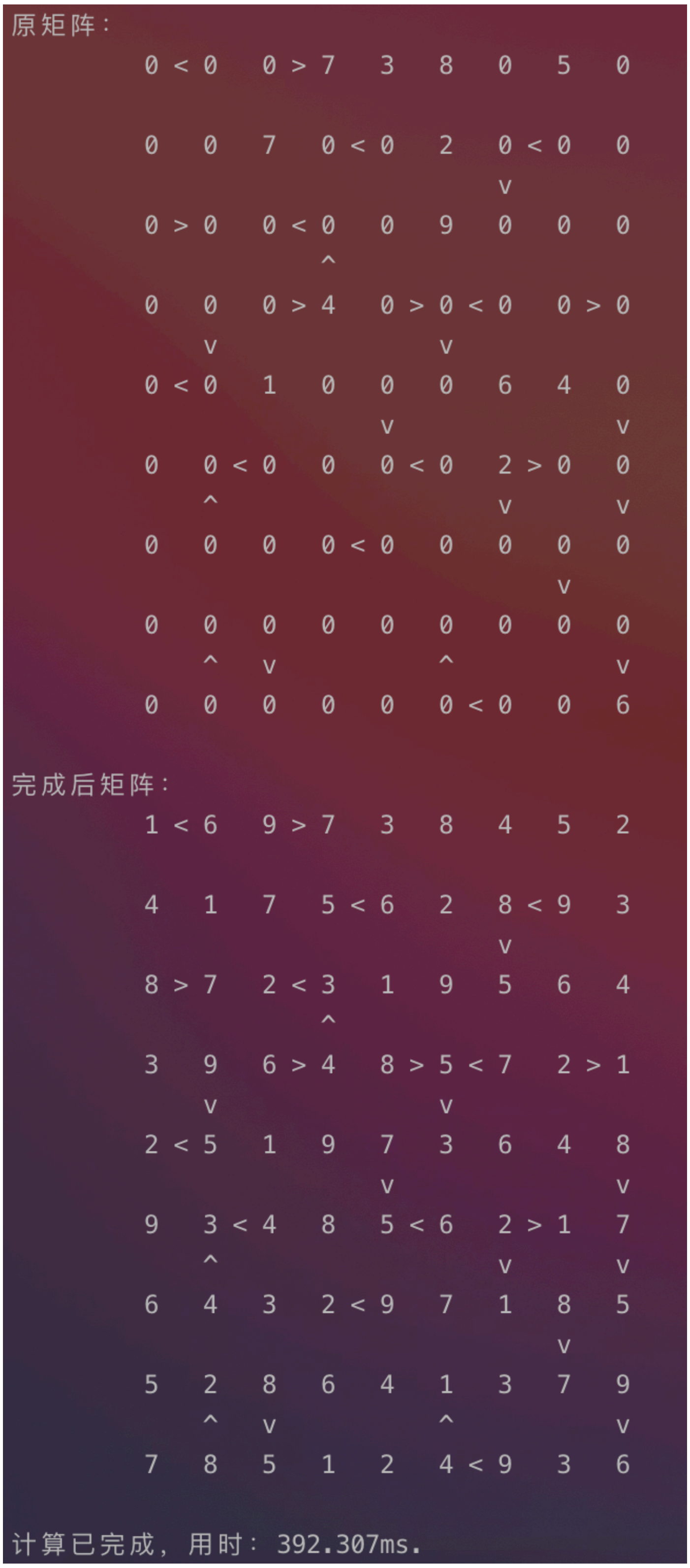
```
int main() {
  futoshiki f;
  futoshiki* ptr = &f;
  for (int i = 0;i<9;i++) {
    for (int j = 0;j<9;j++) {
      f.board[i][j].val = 0;
      f.board[i][j].row = i;
      f.board[i][j].col = j;
      f.board[i][j].u = 0;
      f.board[i][j].d = 0;
      f.board[i][j].l = 0;
      f.board[i][j].r = 0;
      f.board[i][j].assigned = 0;
      memset(f.board[i][j].domain, 0, sizeof(f.board[i][j].domain));
    }
  }
  f.board[0][0].r=-1 ; f.board[0][1].l=1 ;  f.board[0][2].r=1 ; f.board[0][3].l=-1 ;
  f.board[1][3].r=-1 ; f.board[1][4].l=1 ;  f.board[1][6].r=-1 ; f.board[1][7].l=1 ;
  f.board[1][6].d=1 ; f.board[2][6].u=-1 ;  f.board[2][0].r=1 ; f.board[2][1].l=-1 ;
  f.board[2][2].r=-1 ; f.board[2][3].l=1 ;  f.board[2][3].d=-1 ; f.board[3][3].u=1 ;
  f.board[3][2].r=1 ; f.board[3][3].l=-1 ;  f.board[3][4].r=1 ; f.board[3][5].l=-1 ;
  f.board[3][5].r=-1 ; f.board[3][6].l=1 ;  f.board[3][7].r=1 ; f.board[3][8].l=-1 ;
  f.board[4][0].r=-1 ; f.board[4][1].l=1 ;  f.board[5][1].r=-1 ; f.board[5][2].l=1 ;
  f.board[5][4].r=-1 ; f.board[5][5].l=1 ;  f.board[5][6].r=1 ; f.board[5][7].l=-1 ;
  f.board[6][3].r=-1 ; f.board[6][4].l=1 ;  f.board[8][5].r=-1 ; f.board[8][6].l=1 ;
  f.board[3][1].d=1 ; f.board[4][1].u=-1 ;  f.board[3][5].d=1 ; f.board[4][5].u=-1 ;
  f.board[4][4].d=1 ; f.board[5][4].u=-1 ;  f.board[4][8].d=1 ; f.board[5][8].u=-1 ;
  f.board[5][1].d=-1 ; f.board[6][1].u=1 ;  f.board[5][6].d=1 ; f.board[6][6].u=-1 ;
  f.board[5][8].d=1 ; f.board[6][8].u=-1 ;  f.board[6][7].d=1 ; f.board[7][7].u=-1 ;
```



```cpp
  f.board[7][1].d=-1 ; f.board[8][1].u=1 ;  f.board[7][2].d=1 ; f.board[8][2].u=-1 ;
  f.board[7][5].d=-1 ; f.board[8][5].u=1 ;  f.board[7][8].d=1 ; f.board[8][8].u=-1 ;

  f.board[0][3].val = 7; f.board[0][3].assigned = 1; propagate(ptr, &f.board[0][3]);
  f.board[0][4].val = 3; f.board[0][4].assigned = 1; propagate(ptr, &f.board[0][4]);
  f.board[0][5].val = 8; f.board[0][5].assigned = 1; propagate(ptr, &f.board[0][5]);
  f.board[0][7].val = 5; f.board[0][7].assigned = 1; propagate(ptr, &f.board[0][7]);
  f.board[1][2].val = 7; f.board[1][2].assigned = 1; propagate(ptr, &f.board[1][2]);
  f.board[1][5].val = 2; f.board[1][5].assigned = 1; propagate(ptr, &f.board[1][5]);
  f.board[2][5].val = 9; f.board[2][5].assigned = 1; propagate(ptr, &f.board[2][5]);
  f.board[3][3].val = 4; f.board[3][3].assigned = 1; propagate(ptr, &f.board[3][3]);
  f.board[4][2].val = 1; f.board[4][2].assigned = 1; propagate(ptr, &f.board[4][2]);
  f.board[4][6].val = 6; f.board[4][6].assigned = 1; propagate(ptr, &f.board[4][6]);
  f.board[4][7].val = 4; f.board[4][7].assigned = 1; propagate(ptr, &f.board[4][7]);
  f.board[5][6].val = 2; f.board[5][6].assigned = 1; propagate(ptr, &f.board[5][6]);
  f.board[8][8].val = 6; f.board[8][8].assigned = 1; propagate(ptr, &f.board[8][8]);
  cout<<"原矩阵："<<endl;
  print(ptr);
  clock_t start,end;
  start=clock();
  FC(ptr, 0);
  end=clock();
  double endtime=(double)(end-start)/CLOCKS_PER_SEC;
  cout<<"计算已完成，用时："<<endtime*1000<<"ms."<<endl;
```

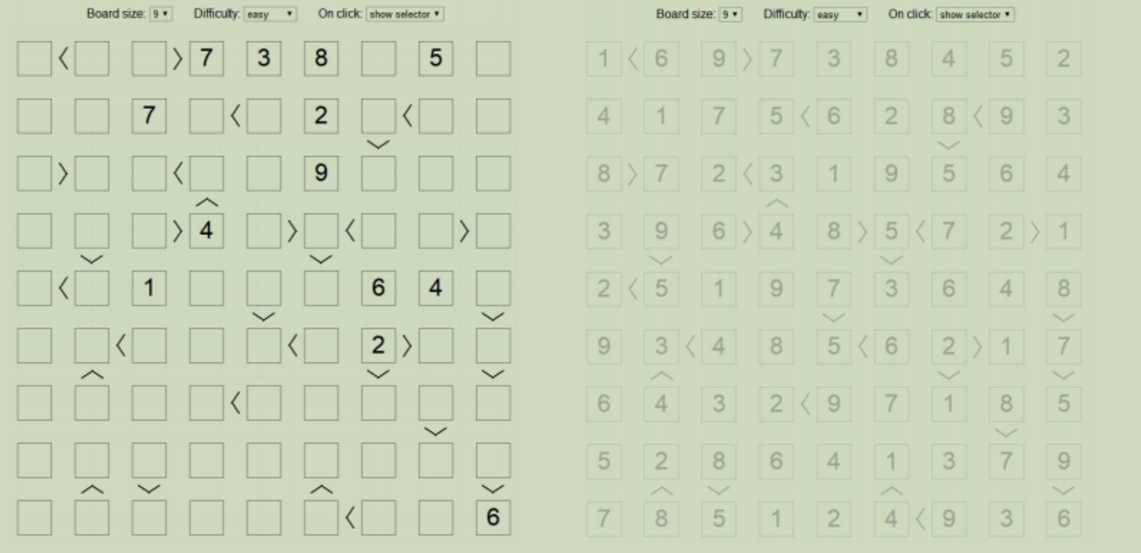
```
    return 0;
}
```

IV Results



```
原矩阵：
          0 < 0   0 > 7   3   8   0   5   0

          0   0   7   0 < 0   2   0 < 0   0
                                  v
          0 > 0   0 < 0   0   9   0   0   0
                      ^
          0   0   0 > 4   0 > 0 < 0   0 > 0
              v               v
          0 < 0   1   0   0   0   6   4   0
                          v               v
          0   0 < 0   0   0 < 0   2 > 0   0
              ^                   v       v
          0   0   0   0 < 0   0   0   0   0
                                      v
          0   0   0   0   0   0   0   0   0
              ^   v           ^           v
          0   0   0   0   0   0 < 0   0   6

完成后矩阵：
          1 < 6   9 > 7   3   8   4   5   2

          4   1   7   5 < 6   2   8 < 9   3
                                  v
          8 > 7   2 < 3   1   9   5   6   4
                      ^
          3   9   6 > 4   8 > 5 < 7   2 > 1
              v               v
          2 < 5   1   9   7   3   6   4   8
                          v               v
          9   3 < 4   8   5 < 6   2 > 1   7
              ^                   v       v
          6   4   3   2 < 9   7   1   8   5
                                      v
          5   2   8   6   4   1   3   7   9
              ^   v           ^           v
          7   8   5   1   2   4 < 9   3   6

计算已完成，用时：392.307ms.
```

对比：



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

```
 _ < _   _ > 7   3   8   _   5   _
                                   
 _   _   7   _ < _   2   _ < _   _
                         v         
 _ > _   _ < _   _   9   _   _   _
             ^                     
 _   _   _ > 4   _ > _ < _   _ > _
     v               v             
 _ < _   1   _   _   _   6   4   _
                 v               v 
 _   _ < _   _   _ < _   2 > _   _
     ^                   v       v 
 _   _   _   _ < _   _   _   _   _
                             v     
 _   _   _   _   _   _   _   _   _
     ^   v           ^           v 
 _   _   _   _   _   _ < _   _   6
```

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

```
 1 < 6   9 > 7   3   8   4   5   2
                                   
 4   1   7   5 < 6   2   8 < 9   3
                         v         
 8 > 7   2 < 3   1   9   5   6   4
             ^                     
 3   9   6 > 4   8 > 5 < 7   2 > 1
     v               v             
 2 < 5   1   9   7   3   6   4   8
                 v               v 
 9   3 < 4   8   5 < 6   2 > 1   7
     ^                   v       v 
 6   4   3   2 < 9   7   1   8   5
                             v     
 5   2   8   6   4   1   3   7   9
     ^   v           ^           v 
 7   8   5   1   2   4 < 9   3   6
```

Figure 1: An Futoshiki Puzzle

可以看到结果正确。