合肥工業大學

“数据结构与算法”

课程设计报告

设计题目 八数码问题

姓　　名 _____

学　　号 _____

专　　业 _____

班　　级 _____

完成日期 2017/5

（一） 需求和规格说明

八数码问题又称重排九宫问题，在一个 3×3 的棋盘上，随机放置 1 到 8 的数字棋子，剩下一个空位，如图所示。数字可以移动到空位（编程时，空位可用 0 代替，且可以理解为是空位的上、下、左、右移动），经过若干次移动后，棋局到达指定目标状态。

2	8	3
1	6	4
7		5

一种初始状态 S

说明：重排九宫问题，对任意给定初始状态，可达下图所示两个目标之一，不可互换。

目标一：如下图 G

1	2	3
8		4
7	6	5

目标一 G

目标二：如下图 G1 或 G2

1	2	3
4	5	6
7	8	

目标二 G1

	1	2
3	4	5
6	7	8

目标二 G2

提示：

可用数组表示棋局状态。用函数表示空格（0）的移动，使用函数具有前提条件，满足条件才可以使用。

编程自动解决问题，不得用手工判断求解。

实现提示：

设计数据结构表示问题的状态。设计几个函数用来改变问题的状态。设计数据结构标记状态是否已经到达过。

用广度优先或深度优先搜索自动求解。还可以使用启发式求解，以提高搜索效率。需要用到递归求解。

要求：

① 编程自动求解问题；

② 给出解题过程中途经的中间状态；

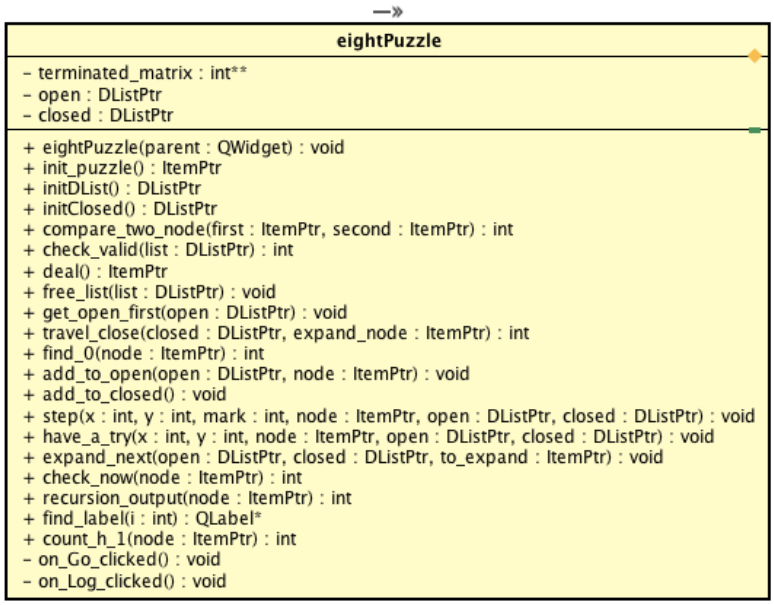
③ 给出解序列（从初始状态到目标状态的函数调用序列）。

（二） 设计

根据上述要求采用以下实现方式

1. 使用 A*算法实现八数码问题的通解
2. 使用 C++作为开发语言，选用 Clang 作为编译器，并使用 QML 作为 GUI 显示框架
3. 使用 Git 作为版本管理工具，并在 Github 托管本项目代码
4. 选取 GNU Make 3.81，IDE 生成 Makefile

设计一个基于 QWidget 的 eightPuzzle 类作为主要算法功能实现，设计一个基于 QWidget 的 Show 类实现主要 GUI 显示功能，设计一个 item 类作为存储信息和状态的数据结构.

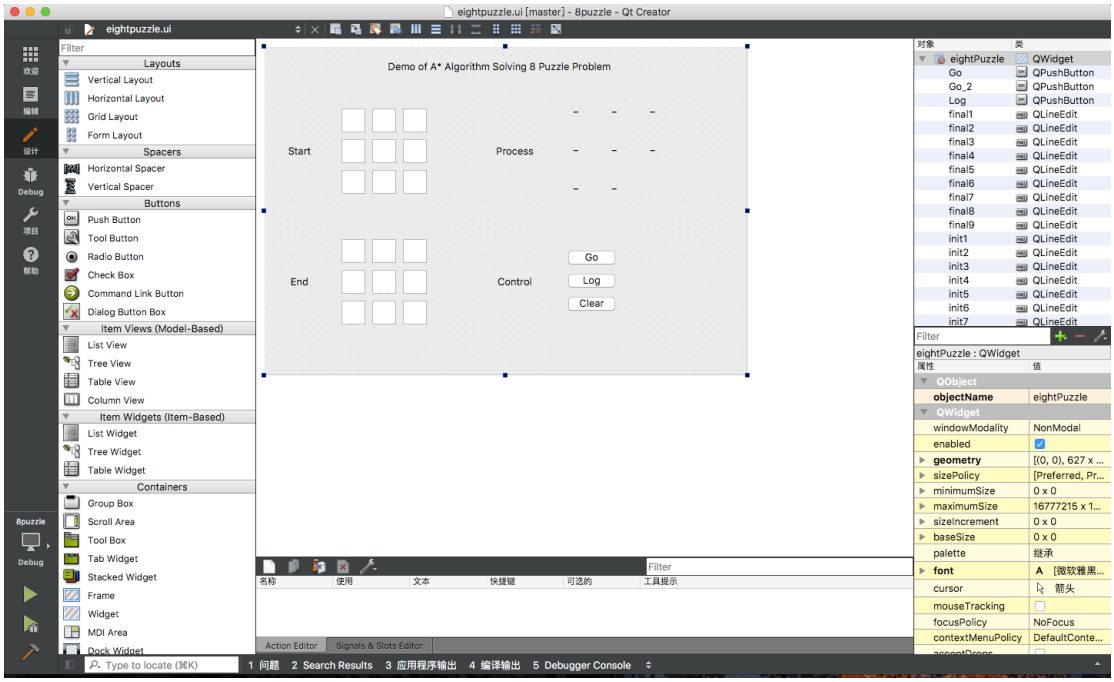



```
typedef struct item {
    int matrix[Matrix_N][Matrix_N];
    int h_x;
    int g_x;
    int f_x;
    struct item *next;
    struct item *hide_pre;
}Item;
typedef Item* ItemPtr;

typedef struct {
    ItemPtr head;
```

```
    ItemPtr tail;
}DList;
typedef DList* DListPtr;
```

GUI 设计界面



编译

Make: make in /Users/xieandong/project/build-8puzzle-Desktop_Qt_5_8_0_clang_64bit-Debug
qmake: qmake 8puzzle.pro -spec macx-clang CONFIG+=debug CONFIG+=x86_64 CONFIG+=qml_debug

（三） 原理

(1) 启发式搜索

广度优先搜索和双向广度优先搜索都属于盲目搜索，这在状态空间不大的情况下是很合适的算法，可是当状态空间十分庞大时，它们的效率实在太低，往往都是在搜索了大量无关的状态结点后才碰到解答，甚至更本不能碰到解答。

搜索是一种试探性的查寻过程，为了减少搜索的盲目性引，增加试探的准确性，就要采用启发式搜索了。所谓启发式搜索就是在搜索中要对每一个搜索的位置进行评估，从中选择最好、可能容易到达目标的位置，再从这个位置向前进行搜索，这样就可以在搜索中省略大量无关的结点，提高了效率。

(2) A*算法

在 A*算法中，一个结点位置的好坏用估价函数来对它进行评估。A*算法的估价函数可表示为：

$$f'(n) = g'(n) + h'(n)$$

这里，f'(n)是估价函数，g'(n)是起点到终点的最短路径值（也称为最小耗费或最小代价），h'(n)是 n 到目标的最短路经的启发值。由于这个 f'(n)其实是无法预先知道的，所以实际上使用的是下面的估价函数：

$$f(n) = g(n) + h(n)$$

其中 g(n)是从初始结点到节点 n 的实际代价，h(n)是从结点 n 到目标结点的最佳路径的估计代价。在这里主要是 h(n)体现了搜索的启发信息，因为 g(n)是已知的。用 f(n)作为 f'(n)的近似，也就是用 g(n)代替 g'(n)，h(n)代替 h'(n)。这样必须满足两个条件：

（1）g(n)>=g'(n)（大多数情况下都是满足的，可以不用考虑），且 f 必须保持单调递增。

（2）h 必须小于等于实际的从当前节点到达目标节点的最小耗费 h(n)<=h'(n)。第二点特别的重要。可以证明应用这样的估价函数是可以找到最短路径的。

（四） 用户手册

本项目程序已完成 GUI 开发,输入符合要求的初始状态和目标状态即可自动完成.在数字框内输入数字,空出的位置不输入. 点击 GO 即可开始搜索, 随后演示结果的过程,点击 log 可以查询具体的流程.

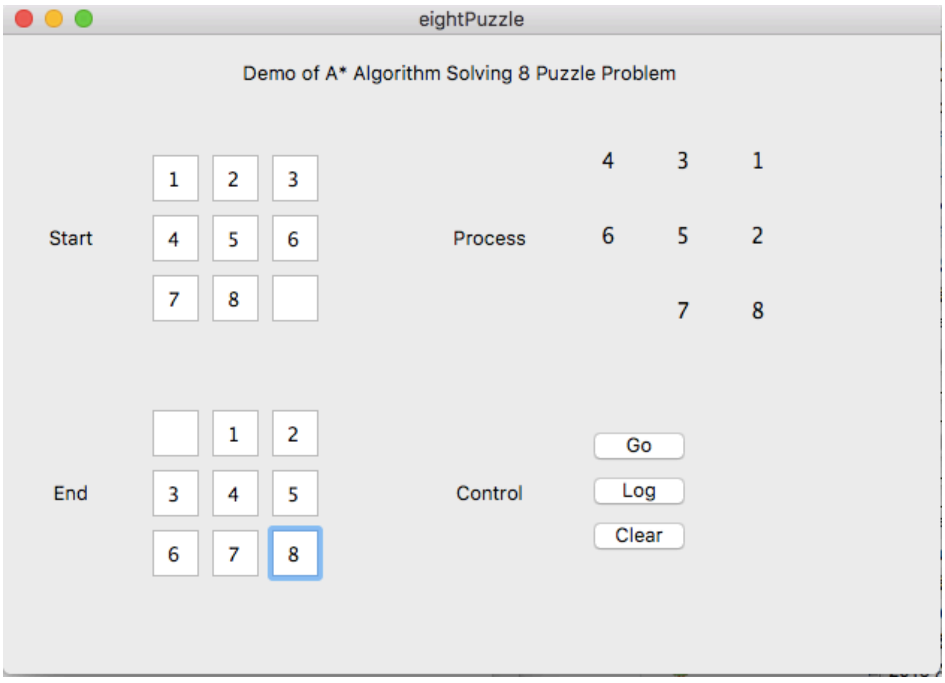
本项目代码托管于 Github

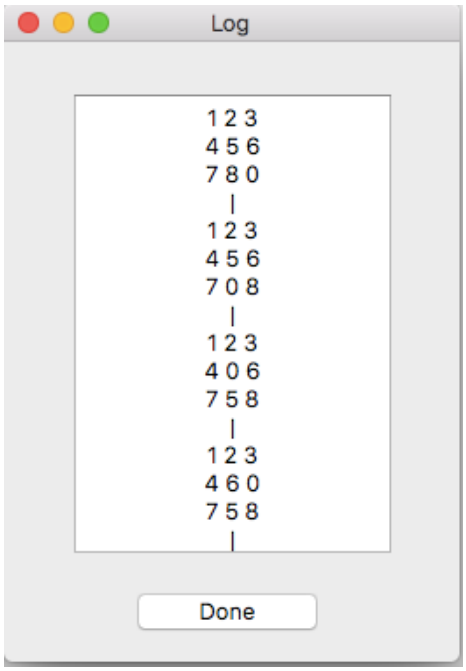
Link: https://github.com/xernyuii/eightFigurePuzzles

（五） 运行实例：

初始状态:1234567890

目标状态:0123456789





（六）进一步改进

（1）搜索时可以采用效率更高的随机算法生成解序列

（2）可以进一步优化算法，使整个搜索过程更加高效

（七）附录——源程序

file: 8puzzle.pro

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widget
TARGET = 8puzzle
TEMPLATE = app
SOURCES += main.cpp\
        eightpuzzle.cpp \
    mode.cpp \
    deal.cpp \
    show.cpp
HEADERS  += eightpuzzle.h \
    head.h \
    show.h
FORMS    += eightpuzzle.ui \
     show.ui
```

file: eightpuzzle.h

```
#ifndef EIGHTPUZZLE_H
#define EIGHTPUZZLE_H

#include <QWidget>
#include <QDebug>
#include <QMessageBox>
#include <QLabel>
#include <QElapsedTimer>
```

```
#include <iostream>
#include "head.h"

namespace Ui {
class eightPuzzle;
}

class eightPuzzle : public QWidget{
    Q_OBJECT

public:
    explicit eightPuzzle(QWidget *parent = 0);
    ~eightPuzzle();

    //初始化 puzzle problem 状态
    ItemPtr init_puzzle();
    //init open list
    DListPtr initDList();
    //init closed list
    DListPtr initClosed();
    //compare two node
    //same 0 diff 1
    int compare_two_node(ItemPtr first, ItemPtr second);
    //check input and output valid
    //valid 1 invalid 0
    int check_valid(DListPtr list);
    //程序入口函数，维护 open close 表
    ItemPtr deal();
    //free function
    void free_list(DListPtr list);
    //取 open 表的第一个节点
    ItemPtr get_open_first(DListPtr open);
    //遍历 close 表查找该状态是否已经走过
    //走过 1 没走过 0
    int travel_close(DListPtr closed, ItemPtr expand_node);
    //找到 0 在那个格子
    int find_0(ItemPtr node);
    //将节点加到 open 表里
    void add_to_open(DListPtr open, ItemPtr node);
    //将 to_expand 节点加入 closed 表，结束该节点的拓展流程
    void add_to_closed(DListPtr closed, ItemPtr node);
    //根据 mark = 1 2 3 4 进行上 下 左 右填充
    void step(int x, int y, int mark, ItemPtr node, DListPtr open, DListPtr closed);
    //根据 0 的位置进行移动并进行试探
    //进行判重 如果已有则判断步数哪个更优 不存在则加入 open 表
    void have_a_try(int x, int y, ItemPtr node, DListPtr open, DListPtr closed);
    //拓展该节点的下一个状态
    void expand_next(DListPtr open, DListPtr closed, ItemPtr to_expand);
    //check now 看看取得 open 表的第一个节点是否为目标
    //这个可以放在 expand 的时候判断，不过为了不中断整个拓展流程所以放在这里，初步考虑两者应该是相同的一位
h_x 是已走步数
    //1 为目标节点 0 则不是
    int check_now(ItemPtr node);
    //recursion output
    int recursion_output(ItemPtr node);
    //find QLabel
    QLabel *find_label(int i);
```

```
    //mode
    int count_h_1(ItemPtr node);
    int count_h_2(ItemPtr node);

private slots:
    void on_Go_clicked();

    void on_Log_clicked();


private:
    int terminated_matrix[Matrix_N][Matrix_N];
    DListPtr open = NULL, closed = NULL;
    ItemPtr first = NULL;
    Ui::eightPuzzle *ui;
};
#endif // EIGHTPUZZLE_H
```

file: head.h

```
#ifndef EIGHTPUZZLE_H
#define EIGHTPUZZLE_H

#include <QWidget>
#include <QDebug>
#include <QMessageBox>
#include <QLabel>
#include <QElapsedTimer>
#include <iostream>
#include "head.h"

namespace Ui {
class eightPuzzle;
}

class eightPuzzle : public QWidget{
    Q_OBJECT

public:
    explicit eightPuzzle(QWidget *parent = 0);
    ~eightPuzzle();

    //初始化 puzzle problem 状态
    ItemPtr init_puzzle();
    //init open list
    DListPtr initDList();
    //init closed list
    DListPtr initClosed();
    //compare two node
    //same 0 diff 1
    int compare_two_node(ItemPtr first, ItemPtr second);
    //check input and output valid
    //valid 1 invalid 0
    int check_valid(DListPtr list);
    //程序入口函数，维护 open close 表
    ItemPtr deal();
    //free function
```

```
    void free_list(DListPtr list);
    //取 open 表的第一个节点
    ItemPtr get_open_first(DListPtr open);
    //遍历 close 表查找该状态是否已经走过
    //走过 1 没走过 0
    int travel_close(DListPtr closed, ItemPtr expand_node);
    //找到 0 在那个格子
    int find_0(ItemPtr node);
    //将节点加到 open 表里
    void add_to_open(DListPtr open, ItemPtr node);
    //将 to_expand 节点加入 closed 表，结束该节点的拓展流程
    void add_to_closed(DListPtr closed, ItemPtr node);
    //根据 mark = 1 2 3 4 进行上 下 左 右填充
    void step(int x, int y, int mark, ItemPtr node, DListPtr open, DListPtr closed);
    //根据 0 的位置进行移动并进行试探
    //进行判重 如果已有则判断步数哪个更优 不存在则加入 open 表
    void have_a_try(int x, int y, ItemPtr node, DListPtr open, DListPtr closed);
    //拓展该节点的下一个状态
    void expand_next(DListPtr open, DListPtr closed, ItemPtr to_expand);
    //check now 看看取得 open 表的第一个节点是否为目标
    //这个可以放在 expand 的时候判断，不过为了不中断整个拓展流程所以放在这里，初步考虑两者应该是相同的一位
h_x 是已走步数
    //1 为目标节点 0 则不是
    int check_now(ItemPtr node);
    //recursion output
    int recursion_output(ItemPtr node);
    //find QLabel
    QLabel *find_label(int i);

    //mode
    int count_h_1(ItemPtr node);
    int count_h_2(ItemPtr node);

private slots:
    void on_Go_clicked();

    void on_Log_clicked();


private:
    int terminated_matrix[Matrix_N][Matrix_N];
    DListPtr open = NULL, closed = NULL;
    ItemPtr first = NULL;
    Ui::eightPuzzle *ui;
};
#endif // EIGHTPUZZLE_H
```

file: show.h

```
#ifndef SHOW_H
#define SHOW_H

#include <QWidget>
#include "head.h"

namespace Ui {
class Show;
```

```
}

class Show : public QWidget{
    Q_OBJECT

public:
    int recursion_output(ItemPtr node);
    int find_0(ItemPtr node);
    explicit Show(ItemPtr first, QWidget *parent = 0);
    ~Show();

private slots:
    void on_done_clicked();

private:
    Ui::Show *ui;
};

#endif // SHOW_H
```

file: deal.cpp

```
#include "head.h"
#include "eightpuzzle.h"
#include "ui_eightpuzzle.h"
extern int terminated_matrix[Matrix_N][Matrix_N];

//初始化 puzzle problem 状态
ItemPtr eightPuzzle::init_puzzle(){
    int temp_i = 0, temp_j = 0, mark = 1, signal[9]={0};
    ItemPtr head = (ItemPtr)malloc(sizeof(Item));
    if (head != NULL) {
        mark = 0;
        for (temp_i = 0; temp_i < 9; ++temp_i) {
            signal[temp_i] = 0;
        }
        //scanf data
        head->matrix[0][0] = ui->init1->text().toInt();
        head->matrix[0][1] = ui->init2->text().toInt();
        head->matrix[0][2] = ui->init3->text().toInt();
        head->matrix[1][0] = ui->init4->text().toInt();
        head->matrix[1][1] = ui->init5->text().toInt();
        head->matrix[1][2] = ui->init6->text().toInt();
        head->matrix[2][0] = ui->init7->text().toInt();
        head->matrix[2][1] = ui->init8->text().toInt();
        head->matrix[2][2] = ui->init9->text().toInt();
        //check input data available?
        for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
            for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
                signal[head->matrix[temp_i][temp_j]] = 1;
            }
        }
        for (temp_i = 0; temp_i < 9; ++temp_i) {
            if (!signal[temp_i]) {
                mark = 1;
            }
        }
        if (mark) {
```

```
            //
            QMessageBox::information(NULL, "Waring", "初始状态有误~", QMessageBox::Yes,
QMessageBox::Yes);
            return NULL;
        }

        head->g_x = 0;
        head->h_x = count_h_1(head);
        head->f_x = head->h_x + head->g_x;
        head->next = NULL;
        head->hide_pre = NULL;
    }
    else {
        //printf("cannt malloc ItemPtr\n");
        QMessageBox::warning(NULL, "warning", "Cannt malloc ItemPtr", QMessageBox::Yes,
QMessageBox::Yes);
        return NULL;
    }
    return head;
}

//check input and output valid
//valid 1 invalid 0
int eightPuzzle::check_valid(DListPtr list){
    int sum1 = 0, sum2 = 0, signal[9]={0}, mark = 0;
    int temp_i = 0, temp_j = 0;
    int arr1[9] = {0}, arr2[9] = {0};
    for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
        for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
            arr1[temp_i*3 + temp_j] = terminated_matrix[temp_i][temp_j];
            arr2[temp_i*3 + temp_j] = list->head->matrix[temp_i][temp_j];
        }
    }

    for (temp_i = 0; temp_i < 9; ++temp_i) {
        for (temp_j = 0; temp_j < temp_i; ++temp_j) {
            if (arr1[temp_i] && (arr1[temp_i] < arr1[temp_j])) {
                ++sum1;
            }
            if (arr2[temp_i] && (arr2[temp_i] < arr2[temp_j])) {
                ++sum2;
            }
        }
    }
    //check terminated data available?
    for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
        for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
            ++signal[terminated_matrix[temp_i][temp_j]];
        }
    }
    for (temp_i = 0; temp_i < 9; ++temp_i) {
        if (!signal[temp_i]||signal[temp_i]>1) {
            ++mark;
        }
    }
    if (mark > 1) {
        QMessageBox::information(NULL, "Waring", "目标状态有误~", QMessageBox::Yes,
QMessageBox::Yes);
```

```
        return -1;
    }
    if (((sum1%2) == (sum2%2))) {
        return 1;
    }
    return 0;
}

//init open dlist
DListPtr eightPuzzle::initDList(){
    DListPtr plist = (DListPtr)malloc(sizeof(DList));
    ItemPtr head = init_puzzle();
    if (plist != NULL) {
        if (head != NULL) {
            plist->head = head;
            plist->tail = head;
        }
        else {
            free(plist);
            return NULL;
        }
    }
    else {
        printf("cannt not malloc DList\n");
        exit(0);
    }
    return plist;
}

//init closed dlist
DListPtr eightPuzzle::initClosed(){
    DListPtr closed = (DListPtr)malloc(sizeof(DList));
    if (closed != NULL) {
        closed->head = NULL;
        closed->tail = NULL;
    }
    else {
        printf("cannt not malloc DList\n");
        exit(0);
    }
    return closed;
}

//free resource
void eightPuzzle::free_list(DListPtr list){
    ItemPtr node = NULL;
    if (list != NULL) {
        while (list->head != NULL) {
            node = list->head;
            list->head = list->head->next;
            free(node);
            node = NULL;
        }
        free(list);
        list = NULL;
    }
}
```

```
//取 open 表的第一个节点
ItemPtr eightPuzzle::get_open_first(DListPtr open){
    ItemPtr first = NULL;
    first = open->head;
    if (open->head == open->tail) {
        open->head = NULL;
        open->tail = NULL;
    }
    else {
        open->head = open->head->next;
    }
    return first;
}

//compare two node
//same 0 diff 1
int eightPuzzle::compare_two_node(ItemPtr first, ItemPtr second){
    int temp_i = 0, temp_j = 0;
    for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
        for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
            if (first->matrix[temp_i][temp_j] != second->matrix[temp_i][temp_j]) {
                return 1;
            }
        }
    }
    return 0;
}

//遍历 close 表查找该状态是否已经走过
//走过 1 没走过 0
int eightPuzzle::travel_close(DListPtr closed, ItemPtr expand_node){
    ItemPtr node = NULL;
    if (closed->head == NULL) {
        return 0;
    }
    node = closed->head;
    while (node != NULL) {
        if (!compare_two_node(node, expand_node)) {
            if (node->g_x > expand_node->g_x) {
                node->g_x = expand_node->g_x;
                node->f_x = node->h_x + node->g_x;
                node->hide_pre = expand_node->hide_pre;
            }
            break;
        }
        node = node->next;
    }
    if (node == NULL) {
        return 0;
    }
    else {
        return 1;
    }
}

//找到 0 在那个格子
int eightPuzzle::find_0(ItemPtr node){
    int temp_i = 0, temp_j = 0, result = 0;
```

```
    for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
        for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
            if (node->matrix[temp_i][temp_j] == 0) {
                result = (temp_i+1)*10 + temp_j;
            }
        }
    }
    return result;
}

//将节点加到 open 表里
void eightPuzzle::add_to_open(DListPtr open, ItemPtr node){
    ItemPtr ptr = NULL, prePtr = NULL;
    if (open->head == NULL) {
        open->head = node;
        open->tail = node;
    }
    else {
        ptr = open->head;
        while (ptr != NULL && ptr->f_x <= node->f_x) {
            prePtr = ptr;
            ptr = ptr->next;
        }
        if (ptr == NULL) {
            open->tail->next = node;
            open->tail = node;
        }
        else {              //插入到 prePtr 和 ptr 中间
            if (prePtr == NULL) {
                node->next = open->head;
                open->head = node;
            }
            else {
                prePtr->next = node;
                node->next = ptr;
            }
        }
    }
    ptr = NULL;
    prePtr = NULL;
}

//将 to_expand 节点加入 closed 表，结束该节点的拓展流程
void eightPuzzle::add_to_closed(DListPtr closed, ItemPtr node){
    if (closed->head == NULL) {
        closed->head = node;
        closed->tail = node;
    }
    else {
        node->next = NULL;
        closed->tail->next = node;
        closed->tail = node;
    }
}

//根据 mark = 1 2 3 4 进行上 下 左 右填充
void eightPuzzle::step(int x, int y, int mark, ItemPtr node, DListPtr open, DListPtr
closed){
```

```
    ItemPtr new_node = NULL;
    int temp_i = 0, temp_j = 0;
    new_node = (ItemPtr)malloc(sizeof(Item));
    if (new_node) {
        for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
            for (temp_j = 0; temp_j <Matrix_N; ++temp_j) {
                new_node->matrix[temp_i][temp_j] = node->matrix[temp_i][temp_j];
            }
        }
        switch (mark) {
            case 1:
                new_node->matrix[x][y] = new_node->matrix[x+1][y];
                new_node->matrix[x+1][y] = 0;
                break;
            case 2:
                new_node->matrix[x][y] = new_node->matrix[x-1][y];
                new_node->matrix[x-1][y] = 0;
                break;
            case 3:
                new_node->matrix[x][y] = new_node->matrix[x][y+1];
                new_node->matrix[x][y+1] = 0;
                break;
            case 4:
                new_node->matrix[x][y] = new_node->matrix[x][y-1];
                new_node->matrix[x][y-1] = 0;
                break;
            default:
                break;
        }
        new_node->g_x = node->g_x + 1;
        new_node->hide_pre = node;
        new_node->next = NULL;
        new_node->h_x = count_h_1(new_node);
        new_node->f_x = new_node->h_x + new_node->g_x;
        if (!travel_close(closed, new_node)) {      //没走过
            add_to_open(open, new_node);
            new_node = NULL;
        }
        else {             //走过
            free(new_node);
            new_node = NULL;
        }
    }
    else {
        printf("cannt malloc new_node\n");
        exit(0);
    }
}
//根据 0 的位置进行移动并进行试探
//进行判重 如果已有则判断步数哪个更优 不存在则加入 open 表
void eightPuzzle::have_a_try(int x, int y, ItemPtr node, DListPtr open, DListPtr closed){
    int count = 0;
    if (x == 1 && y == 1) {           //[1][1]    上下左右
        count = 0;
        while (++count < 5) {
            switch (count) {
                case 1:
                {
```

```
                step(x, y, 1, node, open, closed);
                break;
            }
            case 2:
            {
                step(x, y, 2, node, open, closed);
                break;
            }
            case 3:
            {
                step(x, y, 3, node, open, closed);
                break;
            }
            case 4:
            {
                step(x, y, 4, node, open, closed);
                break;
            }

            default:
                break;
        }
    }
}
else if (x == 1 && y == 0) {       //[1][0]    上下左
    count = 0;
    while (++count < 4) {
        switch (count) {
            case 1:
            {
                step(x, y, 1, node, open, closed);
                break;
            }
            case 2:
            {
                step(x, y, 2, node, open, closed);
                break;
            }
            case 3:
            {
                step(x, y, 3, node, open, closed);
                break;
            }
            default:
                break;
        }
    }
}
else if (x == 1 && y == 2) {       //[1][2]    上下右
    count = 0;
    while (++count < 4) {
        switch (count) {
            case 1:
            {
                step(x, y, 1, node, open, closed);
                break;
            }
            case 2:
```

```
            {
                step(x, y, 2, node, open, closed);
                break;
            }
            case 3:
            {
                step(x, y, 4, node, open, closed);
                break;
            }
            default:
                break;
        }
    }    }
else if (x == 0 && y == 1) {       //[0][1]    上左右
    count = 0;
    while (++count < 4) {
        switch (count) {
            case 1:
            {
                step(x, y, 1, node, open, closed);
                break;
            }
            case 2:
            {
                step(x, y, 3, node, open, closed);
                break;
            }
            case 3:
            {
                step(x, y, 4, node, open, closed);
                break;
            }
            default:
                break;
        }
    }
}
else if (x == 2 && y == 1) {       //[2][1]    下左右
    count = 0;
    while (++count < 4) {
        switch (count) {
            case 1:
            {
                step(x, y, 2, node, open, closed);
                break;
            }
            case 2:
            {
                step(x, y, 3, node, open, closed);
                break;
            }
            case 3:
            {
                step(x, y, 4, node, open, closed);
                break;
            }
            default:
                break;
```

```
        }
    }    }
else if (x == 0 && y == 0) {       //[0][0]    上左
    count = 0;
    while (++count < 3) {
        switch (count) {
            case 1:
            {
                step(x, y, 1, node, open, closed);
                break;
            }
            case 2:
            {
                step(x, y, 3, node, open, closed);
                break;
            }
            default:
                break;
        }
    }    }
else if (x == 0 && y == 2) {       //[0][2]    上右
    count = 0;
    while (++count < 3) {
        switch (count) {
            case 1:
            {
                step(x, y, 1, node, open, closed);
                break;
            }
            case 2:
            {
                step(x, y, 4, node, open, closed);
                break;
            }
            default:
                break;
        }
    }    }
else if (x == 2 && y == 0) {       //[2][0]    下左
    count = 0;
    while (++count < 3) {
        switch (count) {
            case 1:
            {
                step(x, y, 2, node, open, closed);
                break;
            }
            case 2:
            {
                step(x, y, 3, node, open, closed);
                break;
            }
            default:
                break;
        }
    }
}
else {                          //[2][2]    下右
```

```
        count = 0;
        while (++count < 3) {
            switch (count) {
                case 1:
                {
                    step(x, y, 2, node, open, closed);
                    break;
                }
                case 2:
                {
                    step(x, y, 4, node, open, closed);
                    break;
                }
                default:
                    break;
            }
        }
    }
}


//拓展该节点的下一个状态
void eightPuzzle::expand_next(DListPtr open, DListPtr closed, ItemPtr to_expand)
{
    int position = 0, position_x = 0, position_y = 0;
    position = find_0(to_expand);
    position_x = position / 10 - 1;
    position_y = position % 10;
    have_a_try(position_x, position_y, to_expand, open, closed);

}

//check now 看看取得 open 表的第一个节点是否为目标
//这个可以放在 expand 的时候判断，不过为了不中断整个拓展流程所以放在这里，初步考虑两者应该是相同的一位 h_x 是已走步数
//1 为目标节点 0 则不是
int eightPuzzle::check_now(ItemPtr node)
{
    int temp_i = 0, temp_j = 0;
    for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
        for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
            if (node->matrix[temp_i][temp_j] != terminated_matrix[temp_i][temp_j]) {
                return 0;
            }
        }
    }
    return 1;
}

//find QLabel
QLabel* eightPuzzle::find_label(int i)
{
    int a = i/10 - 1;
    int b = i%10;
    switch(a*3 + b + 1) {
    case 1:
        return ui->show1;
        break;
```

```
    case 2:
        return ui->show2;
        break;
    case 3:
        return ui->show3;
        break;
    case 4:
        return ui->show4;
        break;
    case 5:
        return ui->show5;
        break;
    case 6:
        return ui->show6;
        break;
    case 7:
        return ui->show7;
        break;
    case 8:
        return ui->show8;
        break;
    case 9:
        return ui->show9;
        break;
    }
}

//recursion output
int eightPuzzle::recursion_output(ItemPtr node)
{
    if (node != NULL) {
        int i = recursion_output(node->hide_pre);

        if (i == 0) {
            ui->show1->setText(QString::number(node->matrix[0][0]));
            ui->show2->setText(QString::number(node->matrix[0][1]));
            ui->show3->setText(QString::number(node->matrix[0][2]));
            ui->show4->setText(QString::number(node->matrix[1][0]));
            ui->show5->setText(QString::number(node->matrix[1][1]));
            ui->show6->setText(QString::number(node->matrix[1][2]));
            ui->show7->setText(QString::number(node->matrix[2][0]));
            ui->show8->setText(QString::number(node->matrix[2][1]));
            ui->show9->setText(QString::number(node->matrix[2][2]));
            int result = find_0(node);
            find_label(result)->setText("");
            return result;
        }
        else {
            int temp = find_0(node);
            int row_now = i/10 - 1;
            int column_now = i%10;
            int a = node->matrix[row_now][column_now];
            QLabel* label_i = find_label(i);
            QLabel* label_temp = find_label(temp);

            QElapsedTimer t;
            t.start();
            while(t.elapsed() < 1000)
```

```
            QCoreApplication::processEvents();

            label_i->setText(QString::number(a));
            label_temp->setText("");

            return temp;
        }
    }
    else {
        return 0;
    }
}

//deal with the problem
ItemPtr eightPuzzle::deal()
{
    int end = 1;
    ItemPtr first = NULL;
    while (end && open->head!=NULL) {
        first = get_open_first(open);
        if (check_now(first)) {
            end = 0;
            break;
        }
        add_to_closed(closed, first);
        expand_next(open, closed, first);
        first = NULL;
    }
    if (end) {
        QMessageBox::critical(this, "无解", "无解", QMessageBox::Yes, QMessageBox::Yes);
        return NULL;
    }
    return first;
}
```

file: eightpuzzle.cpp

```
#include "head.h"
#include "eightpuzzle.h"
#include "ui_eightpuzzle.h"
#include "Show.h"

eightPuzzle::eightPuzzle(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::eightPuzzle)
{
    ui->setupUi(this);
    QRegExp regx("[0-9]$");
    QValidator *validator = new QRegExpValidator(regx);
    ui->init1->setValidator( validator );
    ui->init2->setValidator( validator );
    ui->init3->setValidator( validator );
    ui->init4->setValidator( validator );
    ui->init5->setValidator( validator );
    ui->init6->setValidator( validator );
    ui->init7->setValidator( validator );
    ui->init8->setValidator( validator );
    ui->init9->setValidator( validator );
```

```
    ui->final1->setValidator( validator );
    ui->final2->setValidator( validator );
    ui->final3->setValidator( validator );
    ui->final4->setValidator( validator );
    ui->final5->setValidator( validator );
    ui->final6->setValidator( validator );
    ui->final7->setValidator( validator );
    ui->final8->setValidator( validator );
    ui->final9->setValidator( validator );
}

eightPuzzle::~eightPuzzle()
{
    free(first);
    free_list(open);
    free_list(closed);
    delete ui;
}

void eightPuzzle::on_Go_clicked()
{
    if (first) {
        free(first);
        first = NULL;
    }
    if (open) {
        free_list(open);
        open = NULL;
    }
    if (closed) {
        free_list(closed);
        closed = NULL;
    }

    open = initDList();
    closed = initClosed();
    terminated_matrix[0][0] = ui->final1->text().toInt();
    terminated_matrix[0][1] = ui->final2->text().toInt();
    terminated_matrix[0][2] = ui->final3->text().toInt();
    terminated_matrix[1][0] = ui->final4->text().toInt();
    terminated_matrix[1][1] = ui->final5->text().toInt();
    terminated_matrix[1][2] = ui->final6->text().toInt();
    terminated_matrix[2][0] = ui->final7->text().toInt();
    terminated_matrix[2][1] = ui->final8->text().toInt();
    terminated_matrix[2][2] = ui->final9->text().toInt();
    //qDebug() << terminated_matrix[0][0] << endl;
    if(open && closed) {//所有条件都ok
        //数据ok
        int vali = check_valid(open);
        if(vali == 1) {
            first = deal();
            int non = recursion_output(first);
        }
        else if(vali == 0){
            QMessageBox::critical(this, "无解", "此问题状态下无解", QMessageBox::Yes,
QMessageBox::Yes);
        }
    }
```

```
}

void eightPuzzle::on_Log_clicked()
{
    Show* my_show = new Show(first);
    my_show->show();
}
```

file: main.cpp

```
#include "eightpuzzle.h"
#include "head.h"
#include <QApplication>

using namespace std;

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    eightPuzzle w;
    w.show();

    return a.exec();
}
```

file: mode.cpp

```
#include "head.h"
#include "eightpuzzle.h"

//计算 h_x 方法 1：不在对应位置点的个数
int eightPuzzle::count_h_1(ItemPtr node)
{
    int mark = 0;
    int temp_i = 0, temp_j = 0;
    for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
        for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
            if (node->matrix[temp_i][temp_j] != terminated_matrix[temp_i][temp_j]) {
                ++mark;
            }
        }
    }
    return mark;
}

//计算 h_x 方法 2：逆序数
int eightPuzzle::count_h_2(ItemPtr node)
{
    int mark = 0, temp_i = 0, temp_j = 0;
    int arr[9] = {0};
    for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
        for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
            arr[temp_i*3 + temp_j] = node->matrix[temp_i][temp_j];
        }
    }
    for (temp_i = 0; temp_i < 9; ++temp_i) {
```

```
        for (temp_j = 0; temp_j < temp_i; ++temp_j) {
            if (arr[temp_j] > arr[temp_i]) {
                ++mark;
            }
        }
    }
    return mark;
}
```

file: show.cpp

```
#include "show.h"
#include "ui_show.h"

Show::Show(ItemPtr first, QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Show)
{
    ui->setupUi(this);
    ui->result->document()->setDefaultTextOption(QTextOption(Qt::AlignHCenter));
    //ui->result->setAlignment(Qt::AlignCenter);
    int k = recursion_output(first);
    ui->result->append("end");
}

Show::~Show()
{
    delete ui;
}
//recursion output
int Show::recursion_output(ItemPtr node)
{
    if (node != NULL) {
        int i = recursion_output(node->hide_pre);

        if (i == 0) {
            QString result_str;
            result_str = QString::number(node->matrix[0][0]) + " " +
QString::number(node->matrix[0][1]) + " " + QString::number(node->matrix[0][2]) + "\n";
            result_str += QString::number(node->matrix[1][0]) + " " +
QString::number(node->matrix[1][1]) + " " + QString::number(node->matrix[1][2]) + "\n";
            result_str += QString::number(node->matrix[2][0]) + " " +
QString::number(node->matrix[2][1]) + " " + QString::number(node->matrix[2][2]) + "\n";
            result_str += "|";
            ui->result->setText(result_str);
            return find_0(node);
        }
        else {
            QString result_str;
            result_str = QString::number(node->matrix[0][0]) + " " +
QString::number(node->matrix[0][1]) + " " + QString::number(node->matrix[0][2]) + "\n";
            result_str += QString::number(node->matrix[1][0]) + " " +
QString::number(node->matrix[1][1]) + " " + QString::number(node->matrix[1][2]) + "\n";
            result_str += QString::number(node->matrix[2][0]) + " " +
QString::number(node->matrix[2][1]) + " " + QString::number(node->matrix[2][2]) + "\n";
            result_str += "|";
            ui->result->append(result_str);
            return find_0(node);
```

```
        }
    }
    else {
        return 0;
    }
}

//找到 0 在那个格子
int Show::find_0(ItemPtr node)
{
    int temp_i = 0, temp_j = 0, result = 0;
    for (temp_i = 0; temp_i < Matrix_N; ++temp_i) {
        for (temp_j = 0; temp_j < Matrix_N; ++temp_j) {
            if (node->matrix[temp_i][temp_j] == 0) {
                result = (temp_i+1)*10 + temp_j;
            }
        }
    }
    return result;
}
void Show::on_done_clicked()
{
    this->close();
}
```