**课程编号：B080104010**

**人工智能导论**

**作业报告**



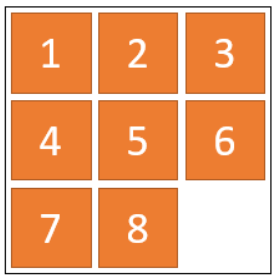
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **姓名** | **周佳男** | **学号** | | **20154994** |
| **班级** | **软件1507** | **指导教师** | | **郝培峰** |
| **实验名称** | **八数码问题** | | | |
| **开设学期** | **2017-2018第二学期** | | | |
| **开设时间** | **第1周——第6周** | | | |
| **报告日期** | **2018年3月31日** | | | |
| **评定成绩** |  | | **评定人** |  |
| **评定日期** |  |

**东北大学软件学院**

**一.问题描述**

八数码问题也称为九宫问题。在3×3的棋盘上摆有八个棋子，每个棋子上标有1至8的某一数字，不同棋子上标的数字不相同。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。给出一个初始状态和一个目标状态，要求通过一系列的数码移动，将初始状态转化为目标状态，并求出从初始状态转变为目标状态的最优移动路径。

后台算法程序(C++)采用了启发式算法(A\*算法)来求解，曼哈顿距离作为h(n)，使用康托展开作为哈希值并使用哈希值来判重，open表使用优先权队列，降低了运行速度因排序而带来的影响。前端动画使用QT来实现矩阵的动态移动效果。



**二.设计**

**1.功能设计**

输入初始状态和目标状态后，程序首先会判断输入的正确性，然后判断是否有解，若有解，则根据A\*算法求出最优移动路径，打印出棋子每步移动后的棋盘的状态和最优解路径，最终将初始状态矩阵信息和最佳解的移动路径写入文件。

接下来使用QT来实现前端动画，QT程序从文件中读取信息，根据文件信息展示出矩阵的动态移动效果。

主要的功能如下：

①判断用户输入的正确性；

②判断从初始状态到目标状态是否有解；

③利用A\*算法求解从初始状态到目标状态的最优解路径(移动方向)；

④打印出从初始状态到目标状态矩阵的变换情况；

⑤计算算法的运行时间；

⑥ QT实现矩阵动态变化效果。

**2.变量设计**

①后台算法程序：

int hash1[9]={1,1,2,6,24,120,720,5040,40320}; //0!-8!，用于康托展开

int dir[4][2] = { { -1,0 },{ 1,0 },{ 0,-1 },{ 0,1 } }; //direction:u d l r 按某一方向移动后的操作

int visited[400000]; //1--O表中 2--C表中 0--不在O和C表中，康拓展开得到的值作为hash值，小于等于400000，判断当前状态的结点是否已被扩展，及在哪个表中

int times=0; //移动次数

string direction="udlr"; //注：在数组中的下标：(2,2)->(1,2)

char path[100]; //记录最优解的移动路径

struct node{ //每一个状态结点

int map[maxn]; //当前状态

int f,g,h; //f=g+h

int x,y; //空格的位置

int prehash; //用于记录前一结点

char nodepath; //用于记录路径

int hash; //hash用于判重

bool operator < (const node &n) const{ //使用的是越小的整数优先权约低的优先权队列

return f!=n.f?f>n.f:h>n.h; //f相等的情况下比较h

}

};

node start; //初始结点(状态)

node end1; //目标结点(状态)

priority\_queue<node> pq; //为了打印路径函数printpath

clock\_t begin,finish; //记录程序开始和结束时间，用于比较算法的效率

②QT程序：

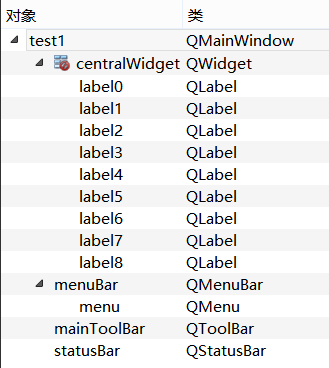
QLabel \*label[9]; //QLabel数组，用于管理ui中的所有QLabel，每一个label为一个数字

QString matrix = in.readLine(); //文件中的初始状态

QString path = in.readLine(); //从初始状态到目标状态的最优解路径

QPropertyAnimation\* animation[length]; //动画数组，用于保存多个不同的动作

QSequentialAnimationGroup \*group; //动画的串行组，用于多个动画的顺序播放



**3.算法设计**

**①有无解及输入正确性的判断：**

判断有无解：求出输入的初始状态和目标状态的逆序数，比较它们的奇偶性是否相同，若相同则有解，否则无解；

正确性：从输入流中取出数据赋值给数组，判断输入的值是否在0-8之间且各不相同，用户的输入只有前9个数有效，只有将输入流清空，保证下次从输入流读取数据不会出错。

**②康托展开：**

Node.hash=a[n]\*(n-1)!+a[n-1]\*(n-2)!+...+a[i]\*(i-1)!+...+a[1]\*0! ，其中a[i]为当前未出现的元素中是排在第几个（从0开始）。在数组中，hash1[i]\*count(count为在第i个元素前比a[i]大的元素个数)为康托展开一项的值，将9个结果相加即为当前状态的康拓展开结果，不同的状态康拓展开结果各不相同，因此将其作为hash值来判断是否该状态出现过。

**③曼哈顿距离：**

d（i，j）=|xi-xj|+|yi-yj|即为曼哈顿距离，求出当前状态所有数字距离目标状态的位置的曼哈顿距离之和，将其作为h(n)的值，构成评价函数f(n)=g(n)+h(n)的一部分。将其用在八数码问题上，相当于将所有数字归位需要的最少移动次数总和。

作为启发函数，A\*需要满足两个条件：1.h(n)>h'(n)，h'(n)为从当前节点到目标点的实际的最优代价值。2.每次扩展的节点的f值大于等于父节点的f值小。将曼哈顿距离作为h(n)，第一个条件自然满足；对于第二个，至多向目标前进1，而深度作为g(n)每次是增加1的，这样g(n)+h(n)至少和原来相等，那么，第二个条件也满足了。因此，我们将曼哈顿距离作为h(n)。

**④最优解路径的记录及打印：**

在结构体node中有一个属性prehash，用于记录该路径上的前一个结点的hash值，以及属性nodepath用于记录前一个状态到当前状态的移动方向，当A\*算法找到最优解路径后，通过目的状态node end的prehash能找到该最优解路径上的前一个状态，因此写一个递归函数直到找到初始状态node start，然后打印各个状态的矩阵，并把路径加入到字符数组path中，最终输出即可。

**⑤队列中弹出某个特定hash值的元素：**

从当前队列中pop结点node并比较node.hash是否等于目标hash值targethash，若不等于则将其push到另一个队列temp中，若等于则找到了目标结点，退出循环，然后将temp中所有结点pop并push到原队列中，函数最终返回该目标结点。

**⑥启发式算法A\*算法的实现：**

在算法A的评价函数中，使用的启发式函数h(n)处在h\*(n)下界范围，即h(n)≤h\*(n),称为A\*算法。我们使用曼哈顿距离作为估价值，即f(n)=g(n)+ |xn-xi|+|yn-yi|，满足1.h(n)>h'(n)，h'(n)为从当前节点到目标点的实际的最优代价值。2.每次扩展的节点的f值大于等于父节点的f值小。

**主要搜索过程：**

创建两个表，OPEN表保存所有已生成而未扩展的节点，CLOSED表中记录已扩展过的节点。本程序中OPEN表使用优先权队列，CLOSED表使用队列：  
While(OPEN!=NULL){  
 从OPEN表中取估价值f最小的节点n(如果f值相同则比较h);  
 if(n节点==目标节点) return;

for(int i=0;i<4;i++){

扩展n结点，Xi为其子结点，初始化Xi;   
if(Xi in OPEN&&f(n,Xi)<f(Xi))   
　　更新OPEN表中的估价值; 更新指针(路径);  
if(Xi in CLOSED&& f(n,Xi)<f(Xi))

更新CLOSED表中的估价值; 把Xi节点放入OPEN; 更新指针(路径);  
if(Xi not in both)  
 将Xi插入到OPEN表中;

}

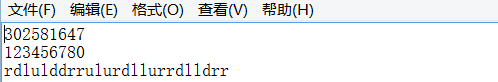
将n结点插入到CLOSED表中;

根据启发式函数将OPEN表中的结点排序; (由于使用了优先权队列，无需排序)

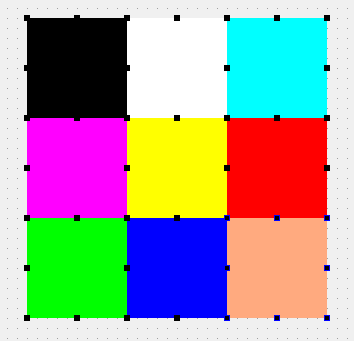
}

⑦QT动画的实现：

后台算法程序将结果写入文件中，文件格式如下，第一行为初始状态，第二行为最优解路径：



QT程序将从文件中读取信息，并将它们分别展示在9个QLabel中，QLabel UI如下所示：



我们将这9个label放入QLabel数组中QLabel \*label[9]，分别初始化其中的文本内容等信息；

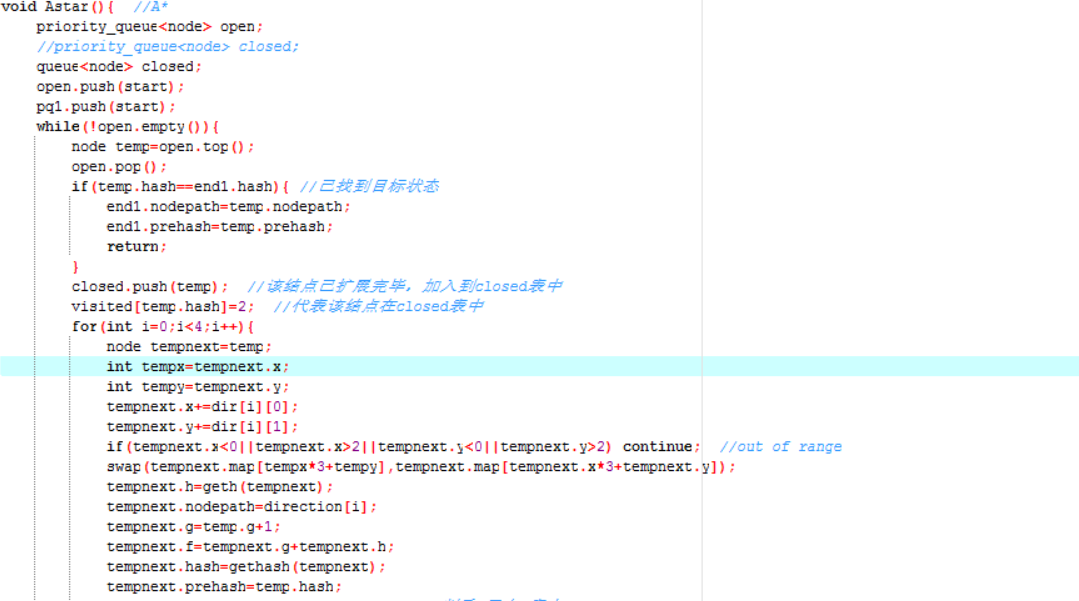
之后我们制作动画QPropertyAnimation\* animation[length]，从初始状态到目标状态的每一步移动都是一个动画animation[i]，动画根据不同的方向来选择每一步要移动的label，根据给定的坐标来生成动画效果；

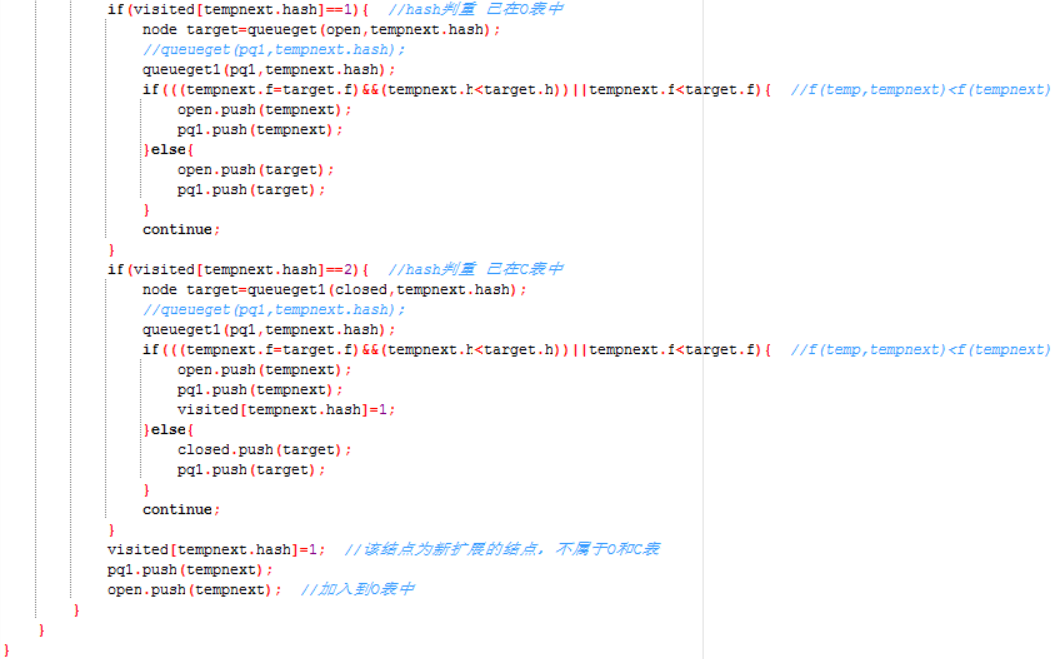
其中我们使用了一个数组int pre[9]来记录每一步动画开始前，3\*3的网格中label的相对位置分布，因为动画开始后，实际上label的xy坐标并没有发生变化，但是在动画中label之间的相对位置已经变化，比如经过第一步label[1]与label[2]交换位置，那么接下来label[5]上面并不是label[2]，而是label[1]，因此我们需要一个数组来记录label们之间的相对位置；

最后我们使用QSequentialAnimationGroup将所有动画按顺序播放，得到了从初始状态到目标状态的动态移动效果。

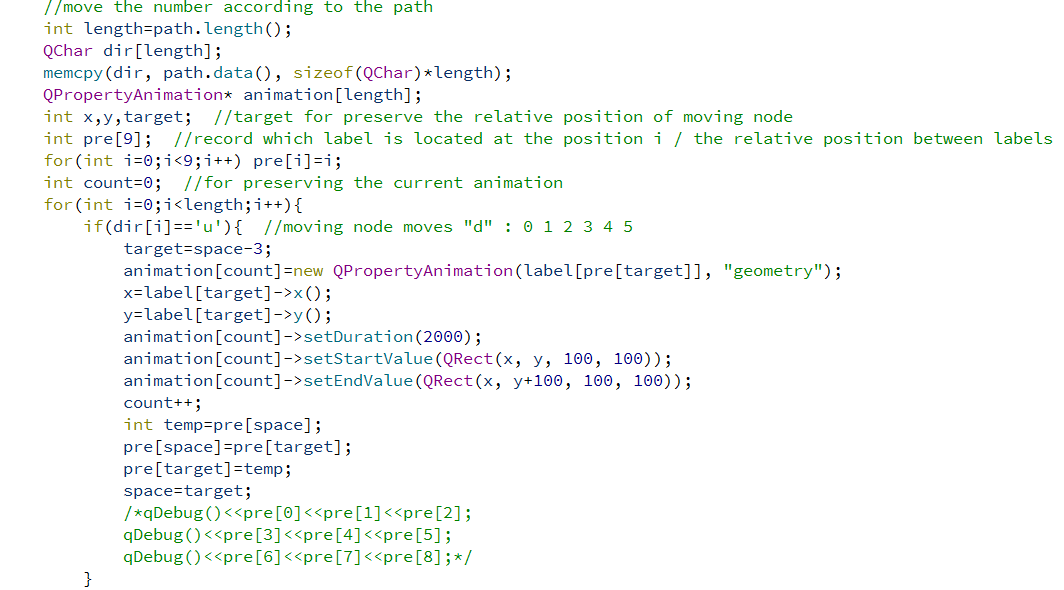
**三.关键代码**

**①A\*算法：(后台算法程序)**

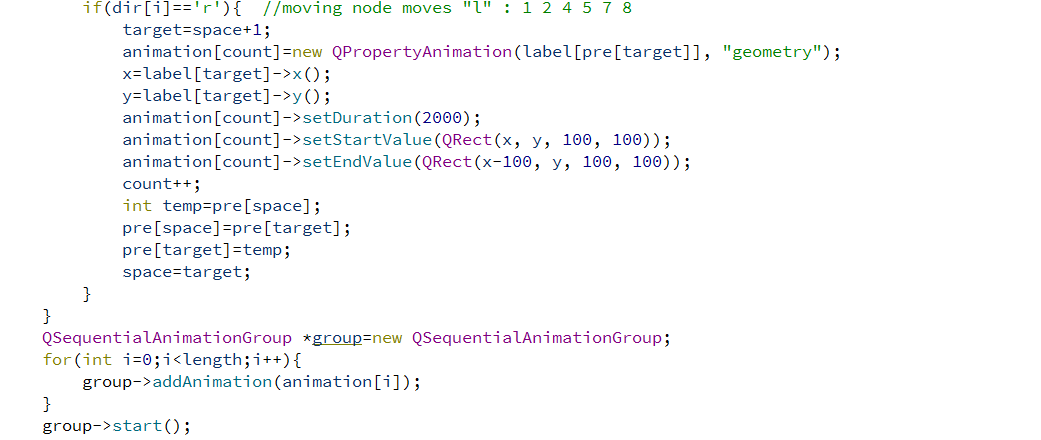




**②动态效果的生成：(QT程序)**







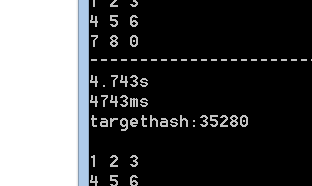
**四.实验过程描述**

输入初始状态3 0 2 5 8 1 6 4 7

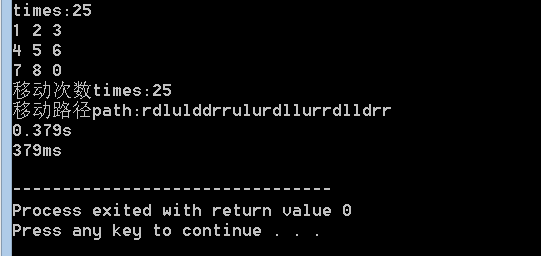
输入目标状态1 2 3 4 5 6 7 8 0

①优先权队列与队列的使用：

在开始编写的代码中，OPEN表和CLOSED表均使用了优先权队列，同时函数从队列中弹出某个特定hash值的元素也是用了优先队列temp用于存储被弹出的结点，所有结点(OPEN表和CLOSED表中所有的结点)也保存在了一个优先权队列pq1中(为了方便遍历来打印最优解路径)，但是导致程序运行时间过长：

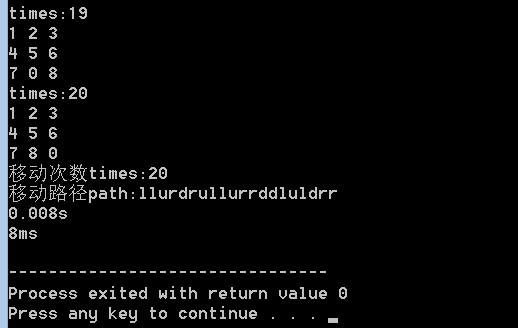


后来经过思考，全部使用优先权队列并没有什么用处，某些地方使用队列也可以实现，因此把除了OPEN表外其他所有的优先权队列改为了队列，A\*算法运行时间如下：



②不同输入：

不同的输入对应着不同的移动次数，一般地，次数越少，A\*算法运行时间越少，如：输入初始状态：5 1 2 4 7 6 3 8 0；输入目标状态：1 2 3 4 5 6 7 8 0



③可改进的地方：

本程序算法可以使用改进的A\*算法来进行优化，A\*算法对m1类结点重新放回O表的操作,将引起多次扩展同一个结点的可能,所以即使扩展的结点数少但重复某些结点也将导致搜索效率下降，因此对启发式函数施加一定限制即单调限制，就可使算法选某一结点扩展时就已经找到到该结点的最佳路径。

通过查阅资料，八数码问题的算法基本分为8中情况：(效率由低到高)

a. 暴力广搜+STL

b. 广搜+哈希

c. 广搜+哈希+打表

d. 双向广搜+哈希

e. A\*+哈希+简单估价函数

f. A\*+哈希+曼哈顿距离

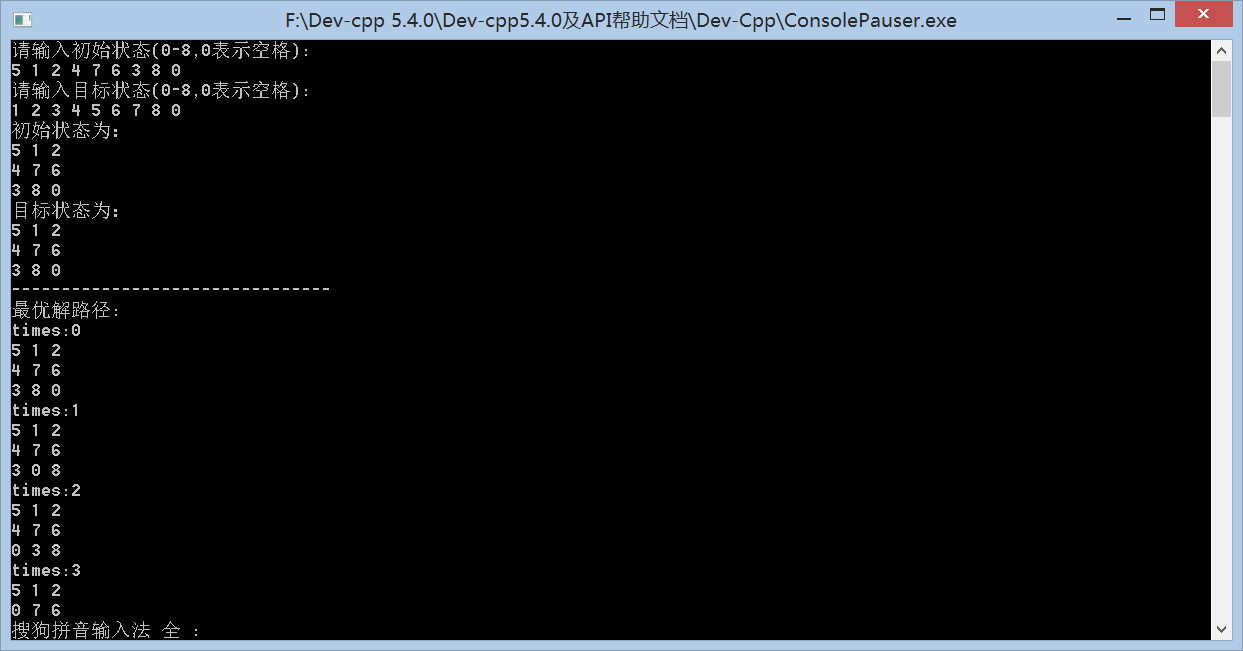
g. A\*+哈希+曼哈顿距离+小顶堆

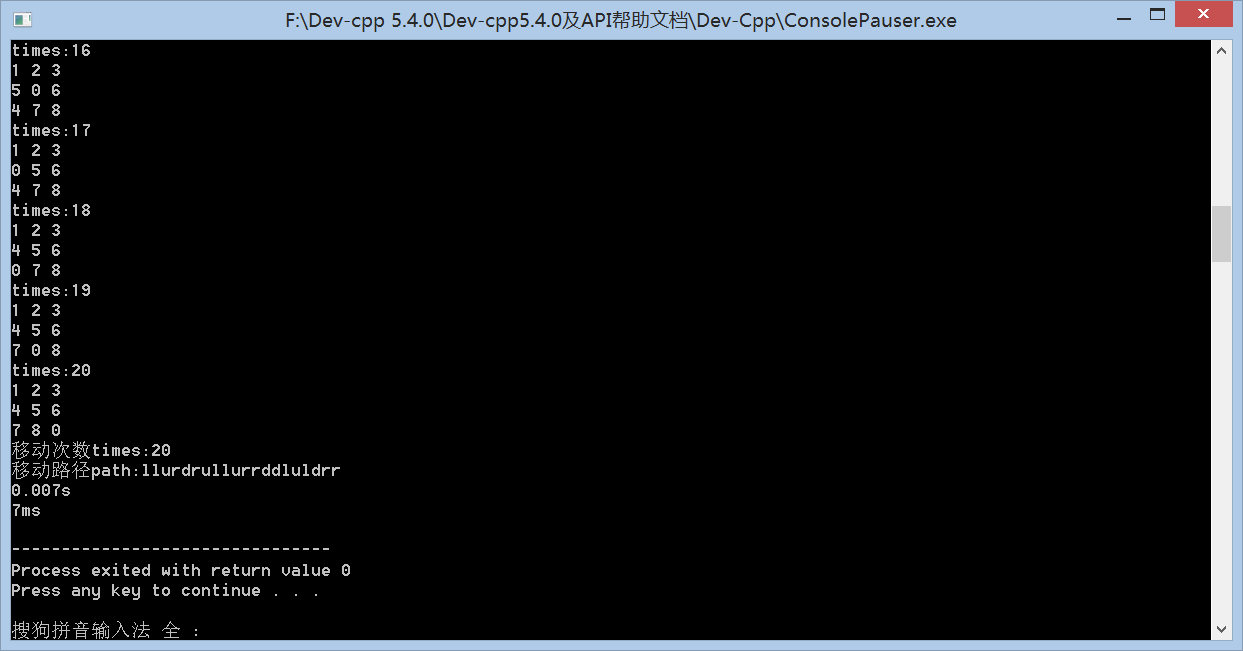
h. IDA\*+曼哈顿距离

**五.主要功能界面截图**

**①后台算法程序：**输入初始状态和结束状态，程序将给出最优解路径及A\*算法运行时间：

(中间的4-15步移动省略)





**②QT程序实现动态移动：**根据后台算法给出的最优解路径，QT将其从初始状态到目标状

态每步的移动以动态形式展现：

