一共有八块拼图放到一个3\*3的方格中，判断是否可以通过移动拼图到空白格这种方式将相应拼图移动到相应的空格上，拼图和方格用数字表示。

本次实现ida\*算法

算法描述：ida\*每次搜索一定深度的A\*算法

本次实现我的ida\*算法并没有每次加一，因为比较慢，我在采用了网上的一种策略，详情见代码。

#include <iostream>

#include <time.h>

using namespace std;

struct ED\_Node{

int arr[9];

int blankPos;

}ed\_node;

ED\_Node tag\_node = {1,2,3,4,5,6,7,8,0,8};

bool isMove[][4]= {{0,1,0,1},{0,1,1,1}, {0,1,1,0},

{1,1,0,1},{1,1,1,1}, {1,1,1,0},

{1,0,0,1}, {1,0,1,1},{1,0,1,0}};

int move\_distance[] = {-3,3,-1,1};

int ED\_weight[]={0,1,2,6,24,120,720,5040,40320}; //康托展开式中各项权重

int str[400000]; //记录路径（存储direction的下标）

char direction[] ="udlr";

int direc[]={-1,1,2,-2};//防止向回走

int currStep;//记录已走步数，其对应的方向存放在str中

int Max\_deepth; //最大深度，不断递增

int Min\_Distance;

int get\_h(const ED\_Node& node)

{

int current\_x,current\_y;

int tag\_x,tag\_y;

int sum\_distance = 0;

for ( int i=0; i<9; ++i ){

if ( node.arr[i] != 0 ){

tag\_x = (node.arr[i]-1)/3;

tag\_y = (node.arr[i]-1)%3;

current\_x = i/3;

current\_y = i%3;

sum\_distance += abs(tag\_x-current\_x)+abs(tag\_y-current\_y);

}

}

return sum\_distance;

}

bool isSloved (const ED\_Node& src,const ED\_Node& tag)

{

int countSrcNX = 0;

int countTagNX = 0;

for ( int i=0; i<9; ++ i ){

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

if ( src .arr[i]>0 ){

countSrcNX = src .arr[j]>src.arr[i] ?

( countSrcNX+1) :countSrcNX ;

}

if ( tag .arr[i]>0 ){

countTagNX = tag.arr[j]>tag.arr[i] ?

( countTagNX+1) :countTagNX ;

}

}

}

return (countSrcNX &1) == (countTagNX&1);

}

void sweap(int& a,int &b)

{

if ( a!=b ){ //自我检测

a = a+b;

b = a-b;

a = a-b;

}

}

bool DFS()

{

int temp\_distance = get\_h(ed\_node);

if ( temp\_distance == 0 ){

return true;

}

int estimateDistance = temp\_distance+currStep;

//cout<<"estimateDistance::"<<estimateDistance<<endl;

Min\_Distance = Max\_deepth>estimateDistance ? Min\_Distance :

(Min\_Distance<estimateDistance ? estimateDistance:Min\_Distance);

if ( estimateDistance>Max\_deepth ){

return false;

}

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

if ( !isMove[ed\_node.blankPos][i] || !(direc[str[currStep]] +direc[i]) ){

continue;

}

str[++currStep] = i;

sweap(ed\_node.arr[ed\_node.blankPos],ed\_node.arr[ed\_node.blankPos+move\_distance[i]]);

ed\_node.blankPos = ed\_node.blankPos+move\_distance[i];

if ( DFS() ){

return true;

}

--currStep;

ed\_node.blankPos = ed\_node.blankPos-move\_distance[i];

sweap(ed\_node.arr[ed\_node.blankPos],ed\_node.arr[ed\_node.blankPos+move\_distance[i]]);

}

return false;

}

void IDAstar()

{

if ( !isSloved(ed\_node,tag\_node) ){

cout<<"unsolvable"<<endl;

return ;

}

currStep = 0;

Max\_deepth = get\_h(ed\_node);

Min\_Distance = Max\_deepth;

while ( !DFS() ){

//Max\_deepth++;

Max\_deepth = Min\_Distance>Max\_deepth ? Min\_Distance : (Max\_deepth+1);

}

for (int i=1; i<=currStep; ++i) {

cout<<direction[str[i]];

}

cout<<endl;

}

int main()

{

char str [100];

while ( gets (str)!=NULL ){

int k =0;

for ( int i=0; i<9; ++ i,k ++ ){

if ( str [k] == 'x' ){

ed\_node.arr [i] = 0;

ed\_node.blankPos = i;

} else if ( str[k]>= '1' && str [k]<= '9'){

ed\_node.arr [i] = str[k]-'0';

} else {

-- i;

}

}

IDAstar();

}

}

代码实现结果：

