**实验四 八数码问题**

**一、实验目的**

理解八数码问题的具体内容，以及用A\*算法求解的过程。

**二、实验分析**

八数码问题也称为九宫问题。在3×3的棋盘，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。要求解决的问题是：给出一个初始状态和一个目标状态，找出一种从初始转变成目标状态的移动棋子步数最少的移动步骤。

所谓问题的一个状态就是棋子在棋盘上的一种摆法。棋子移动后，状态就会发生改变。解八数码问题实际上就是找出从初始状态到达目标状态所经过的一系列中间过渡状态。

**三、实验内容**

**源代码：**

#include <iostream>

#include <vector>

#include <string>

#define col 3

#define row 3

#define maxnum 10000

#define maxdistance 10000

using namespace std;

class node{

public:

int number[row][col];*//二维数组表示数码*

string str;*//数码移动方向("up","down","left","right")*

int distances;//g(n)

int depth;//h(n)

int findex;*//父节点的位置*

node(){

}

int dis();*//计算并返回distances*

void voluation(int index);*//赋值为v[index]*

bool isend();*//是否为目标状态*

bool isequal(node q);*//当前节点和节点q是否相等*

};

vector <node> v;

node father,intent;

void node::voluation(int index){

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

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

number[i][j]=v[index].number[i][j];

}

}

}

int node::dis(){

int s=0;

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

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

if(number[i][j]!=intent.number[i][j]){

s=s+1;

}

}

}

distances=s;

return distances;

}

bool node::isend(){

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

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

if(number[i][j]!=intent.number[i][j]){

return false;

}

}

}

return true;

}

bool node::isequal(node q){

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

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

if(number[i][j]!=q.number[i][j]){

return false;

}

}

}

return true;

}

bool isexpansive(node &n){*//判断n是否可以扩展，即判断v内是否有节点和n相等*

for(int i=0;i<v.size();i++){

if(v[i].isequal(n)){

return false;

}

}

return true;

}

bool isempty(){*//判断v是否还有可“访问”点*

for(int i=0;i<v.size();i++){

if(v[i].distances!=maxnum){

return false;

}

}

return true;

}

int find\_min(){*//在v中寻找评估值最小的节点作为下一步的扩展节点*

int min\_x=maxnum;

int index\_min;

for(int i=0;i<v.size();i++){

if(v[i].distances+v[i].depth<min\_x){

index\_min=i;

min\_x=v[i].distances+v[i].depth;

}

}

return index\_min;

}

void swap\_t(int &a,int &b){

int t;

t=a;

a=b;

b=t;

}

void breath(int index){*//扩展v[index]*

int index\_x,index\_y;

int flag=0;

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

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

if(v[index].number[i][j]==0){*//寻找空格的位置[index\_x][index\_y]*

index\_x=i;

index\_y=j;

flag=1;

break;

}

if(flag==1){

break;

}

}

}

node upnode,downnode,leftnode,rightnode;*//使用分治方法分别将空格往四个方向移动得出新的节点*

upnode.voluation(index);

downnode.voluation(index);

leftnode.voluation(index);

rightnode.voluation(index);

int up\_dis=maxdistance;

int down\_dis=maxdistance;

int left\_dis=maxdistance;

int right\_dis=maxdistance;

if(index\_x>0){*//空格向上，即空格上的数码往下移动一格，下同*

swap\_t(upnode.number[index\_x][index\_y],upnode.number[index\_x-1][index\_y]);

if(isexpansive(upnode)){

up\_dis=upnode.dis();

upnode.findex=index;

upnode.depth=v[index].depth+1;

upnode.str="down";

v.push\_back(upnode);

}

}

if(index\_x<2){

swap\_t(downnode.number[index\_x][index\_y],downnode.number[index\_x+1][index\_y]);

if(isexpansive(downnode)){

down\_dis=downnode.dis();

downnode.findex=index;

downnode.depth=v[index].depth+1;

downnode.str="up";

v.push\_back(downnode);

}

}

if(index\_y>0){

swap\_t(leftnode.number[index\_x][index\_y],leftnode.number[index\_x][index\_y-1]);

if(isexpansive(leftnode)){

left\_dis=leftnode.dis();

leftnode.findex=index;

leftnode.depth=v[index].depth+1;

leftnode.str="right";

v.push\_back(leftnode);

}

}

if(index\_y<2){

swap\_t(rightnode.number[index\_x][index\_y],rightnode.number[index\_x][index\_y+1]);

if(isexpansive(rightnode)){

right\_dis=rightnode.dis();

rightnode.findex=index;

rightnode.depth=v[index].depth+1;

rightnode.str="left";

v.push\_back(rightnode);

}

}

v[index].distances=maxnum;*//v[idnex]已扩展完毕，设为不可“访问”点*

}

ostream& operator<<(ostream& os, node& no)

{

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

for (int j = 0; j < col; j++)

os << no.number[i][j] << ' ';

os << endl;

}

return os;}

void print(int index, vector<node>& rstep\_v)*//利用rstep\_v输出从index到达目标节点的过程*

{

rstep\_v.push\_back(v[index]);

index = v[index].findex;

while (index != 0)

{

rstep\_v.push\_back(v[index]);*//把过程中所有的节点保存在rstep\_v中*

index = v[index].findex;

}

for (int i = rstep\_v.size() - 1; i >= 0; i--){

cout << "Step " << rstep\_v.size() - i<< endl;

node p=rstep\_v[i];

cout<<p.str<<endl;*//输出数码移动的方向*

cout<< p << endl;

}

}

void process(){

while(1){

int t=0;*//break标志*

if(isempty()){*//发生异常：还未到目标状态时v中节点都已扩展完毕*

cout << "error" << endl;

exit(-1);

}

else{

int best=find\_min();

node temp=v[best];*//找出最优节点*

if(temp.isend()){*//最优结点为目标状态时，输出过程*

t=1;*//已到达目标状态，t为1*

vector<node> rstep\_v;

print(best,rstep\_v);

}

else{*//未到达目标状态，扩展最优结点*

breath(best);

}

}

if(t)*//已到达目标状态，退出循环*

break;

}

}

int main()

{

cout << "Input source:" << endl;

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

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

cin>>father.number[i][j];

}

}

father.findex=0;

father.depth=0;

cout << "Input end:" << endl;

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

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

cin>>intent.number[i][j];

}

}

v.push\_back(father);

process();

return 0;

}

运行结果：



