实验三

A\*算法解决八数码问题：

在一个3\*3的棋盘中，分别用1,2,3,...,8表示八个数码方格，用0表示空缺的方格，现给出一个初始状态和目标状态，寻找出在评估函数f(n)=g(n)+h(n)的限制下，以最少的步数到达目标状态（一次将一个数码方格移动到空缺的方格中）；其中，g(n)表示n节点与目标状态的“距离”，h(n)表示n节点与初始状态的距离。

启发式：f(n)=g(n)+h(n)

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];

string str;

int distances;

int depth;

int findex;

node(){

}

int dis();

void voluation(int index);

bool isend();

bool isequal(node q);

};

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 s;

}

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){

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

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

return false;

}

}

return true;

}

bool isempty(){

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

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

return false;

}

}

return true;

}

int find\_min(){

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){

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){

index\_x=i;

index\_y=j;

flag=1;

break;

}

if(flag==1){

break;

}

}

}

node upnode,downnode,leftnode,rightnode;

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){

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;

}

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)

{

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

index = v[index].findex;

while (index != 0)

{

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

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;

cout<< p << endl;

}

}

void process(){

while(1){

int t=0;

if(isempty()){

cout << "error" << endl;

exit(-1);

}

else{

int best=find\_min();

node temp=v[best];

if(temp.isend()){

t=1;

vector<node> rstep\_v;

print(best,rstep\_v);

}

else{

breath(best);

}

}

if(t)

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( 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：

