Astar算法

### A\*算法解决八数码问题

代码：

#include <iostream>

#include <ctime>

#include <vector>

using namespace std;

const int ROW = 3;

const int COL = 3;

const int MAXDISTANCE = 10000;

const int MAXNUM = 10000;

typedef struct \_Node{

int digit[ROW][COL];

int dist; // distance between one state and the destination

int dep; // the depth of node

// So the comment function = dist + dep.

int index; // point to the location of parent

} Node;

Node src, dest;

vector<Node> node\_v; // store the nodes

bool isEmptyOfOPEN() {

for (int i = 0; i < node\_v.size(); i++) {

if (node\_v[i].dist != MAXNUM)

return false;

}

return true;

}

bool isEqual(int index, int digit[][COL]) {

for (int i = 0; i < ROW; i++)

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

if (node\_v[index].digit[i][j] != digit[i][j])

return false;

}

return true;

}

ostream& operator<<(ostream& os, Node& node) {

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

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

os << node.digit[i][j] << ' ';

os << endl;

}

return os;

}

void PrintSteps(int index, vector<Node>& rstep\_v) {

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

index = node\_v[index].index;

while (index != 0) {

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

index = node\_v[index].index;

}

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

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

<< endl << rstep\_v[i] << endl;

}

void Swap(int& a, int& b) {

int t;

t = a;

a = b;

b = t;

}

void Assign(Node& node, int index) {

for (int i = 0; i < ROW; i++)

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

node.digit[i][j] = node\_v[index].digit[i][j];

}

int GetMinNode() {

int dist = MAXNUM;

int loc; // the location of minimize node

for (int i = 0; i < node\_v.size(); i++) {

if (node\_v[i].dist == MAXNUM)

continue;

else if ((node\_v[i].dist + node\_v[i].dep) < dist) {

loc = i;

dist = node\_v[i].dist + node\_v[i].dep;

}

}

return loc;

}

bool isExpandable(Node& node) {

for (int i = 0; i < node\_v.size(); i++) {

if (isEqual(i, node.digit))

return false;

}

return true;

}

int Distance(Node& node, int digit[][COL]) {

int distance = 0;

bool flag = false;

for(int i = 0; i < ROW; i++)

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

for (int k = 0; k < ROW; k++) {

for (int l = 0; l < COL; l++) {

if (node.digit[i][j] == digit[k][l]) {

distance += abs(i - k) + abs(j - l);

flag = true;

break;

}

else

flag = false;

}

if (flag)

break;

}

return distance;

}

int MinDistance(int a, int b) {

return (a < b ? a : b);

}

void ProcessNode(int index) {

int x, y;

bool flag;

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

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

if (node\_v[index].digit[i][j] == 0) {

x =i; y = j;

flag = true;

break;

}

else flag = false;

}

if(flag)

break;

}

Node node\_up;

Assign(node\_up, index);

int dist\_up = MAXDISTANCE;

if (x > 0) {

Swap(node\_up.digit[x][y], node\_up.digit[x - 1][y]);

if (isExpandable(node\_up)) {

dist\_up = Distance(node\_up, dest.digit);

node\_up.index = index;

node\_up.dist = dist\_up;

node\_up.dep = node\_v[index].dep + 1;

node\_v.push\_back(node\_up);

}

}

Node node\_down;

Assign(node\_down, index);

int dist\_down = MAXDISTANCE;

if (x < 2) {

Swap(node\_down.digit[x][y], node\_down.digit[x + 1][y]);

if (isExpandable(node\_down)) {

dist\_down = Distance(node\_down, dest.digit);

node\_down.index = index;

node\_down.dist = dist\_down;

node\_down.dep = node\_v[index].dep + 1;

node\_v.push\_back(node\_down);

}

}

Node node\_left;

Assign(node\_left, index);

int dist\_left = MAXDISTANCE;

if (y > 0) {

Swap(node\_left.digit[x][y], node\_left.digit[x][y - 1]);

if (isExpandable(node\_left)) {

dist\_left = Distance(node\_left, dest.digit);

node\_left.index = index;

node\_left.dist = dist\_left;

node\_left.dep = node\_v[index].dep + 1;

node\_v.push\_back(node\_left);

}

}

Node node\_right;

Assign(node\_right, index);

int dist\_right = MAXDISTANCE;

if (y < 2) {

Swap(node\_right.digit[x][y], node\_right.digit[x][y + 1]);

if (isExpandable(node\_right)) {

dist\_right = Distance(node\_right, dest.digit);

node\_right.index = index;

node\_right.dist = dist\_right;

node\_right.dep = node\_v[index].dep + 1;

node\_v.push\_back(node\_right);

}

}

node\_v[index].dist = MAXNUM;

}

int main() {

int number;

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

for (int i = 0; i < ROW; i++)

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

cin >> number;

src.digit[i][j] = number;

}

src.index = 0;

src.dep = 1;

cout << "Input destination:" << endl;

for (int m = 0; m < ROW; m++)

for (int n = 0; n < COL; n++) {

cin >> number;

dest.digit[m][n] = number;

}

node\_v.push\_back(src);

cout << "Search..." << endl;

clock\_t start = clock();

while (1) {

if (isEmptyOfOPEN()) {

cout << "Cann't solve this statement!" << endl;

return -1;

}

else {

int loc; // the location of the minimize node

loc = GetMinNode();

if(isEqual(loc, dest.digit)) {

vector<Node> rstep\_v;

cout << "Source:" << endl;

cout << src << endl;

PrintSteps(loc, rstep\_v);

cout << "Successful!" << endl;

cout << "Using " << (clock() - start) / CLOCKS\_PER\_SEC

<< " seconds." << endl;

break;

}

else

ProcessNode(loc);

}

}

return 0;

}







