# IDS解决八数码问题

## 实验目的

用IDS算法完成八数码问题

## 二、算法分析

IDS深度迭代搜索：在有界的深度优先搜索的基础上迭代的设置边界，即先考虑一层的DFS，若找不到目标则找两层的DFS，一直迭代下去直到找到目标节点。所以这种方法也是一定能找到界的，只是时间消耗会很多。

## 三、实验题目

 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。棋盘中留有一个空格，空格用0来表示。空格周围的棋子可以移到空格中。要求解的问题是：给出一种初始布局（初始状态）和目标布局，用IDS实现从初始状态到目标状态。

## 四、实验代码

**#include<iostream>**

**#include<string>**

**#include<vector>**

**#include<time.h>**

**#include<algorithm>**

**#include<utility>**

**#include<stack>**

**using namespace std;**

**//状态结构体**

**struct State{**

**State(){**

**}**

**State(const State &st){**

**for(int i=0;i<3;i++)**

**for(int j=0;j<3;j++)**

**matrix[i][j] = st.matrix[i][j];**

**zero\_x=st.zero\_x; zero\_y=st.zero\_y;**

**g=st.g;**

**// f=st.f; g=st.g; h=st.h;**

**parent = st.parent;**

**}**

**~State(){**

**parent = NULL;**

**}**

**char matrix[3][3]; //二维数组**

**int zero\_x,zero\_y; //0（即空格）的位置**

**int g; //起始状态到当前状态的代价**

**// int f,g,h; //代价函数f=g+h,g：起始状态到当前状态的代价，h: 估计当前状态到最终状态的代价**

**//int parent\_idx; //父节点, 这个int指的是在对象数组的index**

**struct State\* parent;**

**// friend bool operator < (const State &a,const State &b){**

**// if(a.f==b.f) return a.g<=b.g;**

**// return a.f>=b.f;**

**// }**

**};**

**State start; //开始状态（开始节点）**

**State goal; //目标状态**

**//矩阵转字符串形式，以便存储到set中方便查重**

**string matrix2string(char a[3][3] ){**

**string s="";**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**s+=a[i][j];**

**}**

**}**

**return s;**

**}**

**//打印二维数组**

**void print\_matrix(char a[3][3]){**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**cout<<a[i][j]<<" ";**

**}**

**cout<<endl;**

**}**

**}**

**//找空白格的位置,即'0'点**

**void find\_blankTile(){**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**if(start.matrix[i][j]=='0'){**

**start.zero\_x = i;**

**start.zero\_y = j;**

**return;**

**}**

**}**

**}**

**}**

**int move\_x[4] = {-1,0,1,0}; //上右下左**

**int move\_y[4] = {0,1,0,-1};**

**stack<pair<State,int> > que;**

**int node\_num;**

**stack<State> stack\_path; //记录路径**

**int IDS(int max\_depth){**

**//初始化start节点**

**find\_blankTile(); start.g=0; start.parent=NULL;**

**int depth=0; //深度**

**que.push(make\_pair(start,depth)); //将<节点，深度>对 入栈**

**int curDepth,nextDepth; //当前节点深度，扩展节点深度**

**node\_num=0; //记录总扩展节点数**

**while(!que.empty()){**

**node\_num++;**

**State \*curState = new State(que.top().first);**

**curDepth = que.top().second;**

**que.pop();**

**if(matrix2string(curState->matrix)==matrix2string(goal.matrix)){**

**int step = curState->g;**

**// 据父节点一步步往回寻路，存入stack\_path中**

**while(!stack\_path.empty()) stack\_path.pop(); //清空栈**

**State\* p = curState;**

**while(p!=NULL){**

**stack\_path.push(\*p);**

**p = p->parent;**

**}**

**return step;//返回所需步数**

**}**

**nextDepth = curDepth + 1;**

**if(nextDepth > max\_depth) continue;**

**string curSeq = matrix2string(curState->matrix); //当前状态的字符串序列**

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

**State\* nextState = new State(\*curState);//下一个状态.先拷贝父节点，之后再改**

**nextState->zero\_x = curState->zero\_x+move\_x[i];**

**nextState->zero\_y = curState->zero\_y+move\_y[i];**

**if(nextState->zero\_x<0||nextState->zero\_x>2**

**||nextState->zero\_y<0||nextState->zero\_y>2) continue; //移动超出范围，跳过**

**nextState->matrix[nextState->zero\_x][nextState->zero\_y] = '0';**

**nextState->matrix[curState->zero\_x][curState->zero\_y]**

**= curState->matrix[nextState->zero\_x][nextState->zero\_y];**

**//防止移回上一个状态**

**if(curState->parent!=NULL){ //排除初始节点，因为起始节点没有父节点**

**if(matrix2string(nextState->matrix)==matrix2string(curState->parent->matrix))**

**continue; //下个状态的字符串不能跟上个状态一样**

**}**

**nextState->g += 1;**

**nextState->parent = curState;**

**que.push(make\_pair(\*nextState,nextDepth));**

**}**

**}**

**return -1; //找不到解，返回-1**

**}**

**//打印路径**

**void print\_path(){**

**State\* st;**

**cout<<"path: "<<endl;**

**while(!stack\_path.empty()){**

**State\*st = &stack\_path.top();**

**print\_matrix(st->matrix);**

**cout<<endl;**

**stack\_path.pop();**

**}**

**}**

**//判断是否有解，初始状态的逆序数应与目标状态的同奇偶， 目标状态为偶排列**

**bool check\_if\_solvable()**

**{**

**string start\_seq = matrix2string(start.matrix);**

**int cnt=0;**

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

**if(start\_seq[i]=='0') continue;**

**for(int j=i-1;j>=0;j--){**

**if(start\_seq[j]>start\_seq[i]) cnt++;**

**}**

**}**

**if(cnt%2==0) return true;**

**return false;**

**}**

**// 随即生成函数**

**int myrandom (int i) {**

**return rand()%i;**

**}**

**//生成0~8的随机整数序列，作为开始状态**

**void randperm()**

**{**

**vector<char> temp;**

**for (int i = 0; i < 9; ++i) temp.push\_back(i+'0'); //生成0~8**

**random\_shuffle(temp.begin(), temp.end(),myrandom); //随机乱序**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**start.matrix[i][j] = temp[i\*3+j];**

**}**

**}**

**}**

**int main(){**

**//目标状态**

**char goal\_matrix[3][3]={{'1','2','3'},{'4','5','6'},{'7','8','0'}};**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**goal.matrix[i][j] = goal\_matrix[i][j];**

**}**

**}**

**char a[3][3]={{'1','2','3'},{'4','6','0'},{'7','5','8'}}; //开始状态**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**start.matrix[i][j] = a[i][j];**

**}**

**}**

**clock\_t startTime,endTime;**

**startTime = clock();**

**cout << "steps: " << IDS(15) << endl;**

**endTime = clock();**

**cout<<"node extended: "<<node\_num<<endl;**

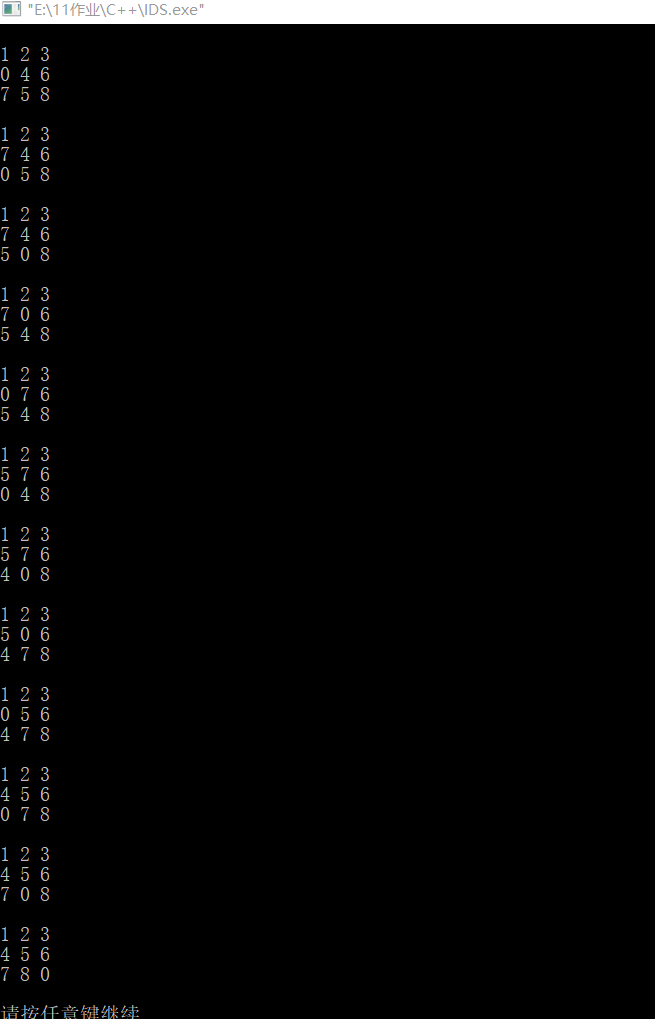
**cout<<endl<<"time: "<<(double)(endTime-startTime)/CLOCKS\_PER\_SEC<<"s."<<endl;**

**print\_path();**

**return 0;**

**}**

## 五、实验结果：



## 六、实验总结

迭代深度索搜其实是深搜（DFS）的一个优化版，普通深搜是一直往下扩展，这样可能会陷入无限循环，IDS是有界的深度搜索，当搜索到一定深度的时候若还没找到解就不再往纵向搜索了，转而向横向搜索。这样，可以先设深度为1，然后迭代的递增深度，若在当前深度搜不到解就继续迭代下一个深度，这样如果问题是有解的，那么总能找到解，且是最优解。因为这种思想类似于广搜，一层一层的迭代，最优解总在较浅层找到。