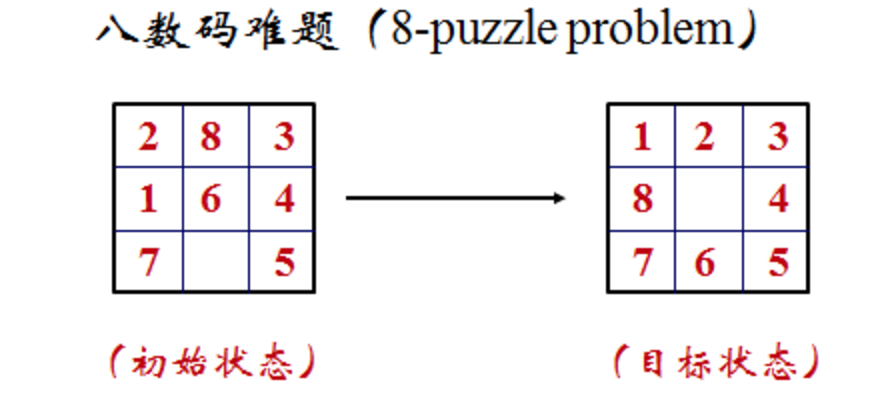
**实验一 八数码问题**

**姓名：杨志烨 学号：16S103168**

1. **问题描述**

**(1)待解决问题的解释**

八数码问题给定一个3\*3的棋盘，放置数码1-8，剩下一个空格，只能通过棋子向空格的移动来改变棋盘的布局。要求给定初始布局和目标布局，求得如何移动棋子能从初始布局到达目标布局，寻找到合法的移动序列。



**(2)问题的搜索形式描述**

初始状态：如上图所示(图片来自网络)，8个数字在九宫格棋牌上的随机状态。

后继函数：通过四种动作(上、下、左、右)，将空格周围的棋子移到空格处，达到新的合法状态。

目标测试：比较当前状态是否和目标状态棋子分布一样。

路径消耗：每一次移动耗散值为1，总耗散值即移动的总步数。

**(3)解决方案介绍**

解决八数码问题首先要判断对于给定的初始状态和目标状态是否有解，如果无解，那么输出无解。有解的话再进行搜索。由数学知识可知，可以计算初始状态和目标状态的逆序值，如果两个状态的逆序值都是奇数或者偶数，那么这个八数码可解，否则无解。

如果该问题有解，那么可以通过搜索来解决问题。比如深度优先搜索和广度优先搜索，但这两种算法都是在一个给定的状态空间中枚举，而八数码问题有9！个状态空间，还是比较费时的。所以本实验采用A\*算法作为搜索策略，即启发式搜索。

1. **算法介绍**

**(1)搜索算法的一般介绍**

本实验中采用A\*算法求解，A\*算法是启发式搜索的一种，A\*算法的核心部分，就在它的估值函数的设计上，即如何设计f(n) = g(n) + h(n)。八数码问题的估价函数也为f(n) = g(n) + h(n)，f(n)是节点n的估价函数，g(n)是在状态空间中从初始节点到n的代价，h(n)是从n到目标节点最佳路径的估计代价。在八数码问题中，g(n)就是从初始状态到当前状态移动棋子的次数，h(n)就是当前状态各个数字不在目标状态位置的个数。

**(2)算法伪代码**

算法功能：求解8数码问题

输入：8数码问题的初始状态

默认目标状态为：

1. 2 3

4 5 6

7 8 x

输出：从初始状态移动到目标状态的移动序列

算法描述：

Begin：

While(读入初始状态数据)

初始化

判断是否可解

if(可解)

得到初始状态哈希值，并与目标状态哈希值比较

if(当前状态哈希值==目标状态哈希值)

return

将当前状态push进优先级队列

while(队列非空)

取队顶元素

遍历可取移动状态

如果状态合法则push进优先级队列

if(当前状态哈希值==目标状态哈希值)

return

else

return

输出解路径

End While

End

1. **算法实现**

**(1)实验环境与问题规模**

实验环境:

系统:Macbook Pro , Mac OS 10.11.5

语言及编译器:C++,g++

IDE:CodeRunner

问题规模:N组初始状态数据

**(2)数据结构：**

1. Node结构体代表每一个状态，其属性及函数如下：

属性：

state : 3\*3 int型数组，存放数码状态

x,y : 记录空位置的坐标

h,g : f(n) = g(n)+h(n) 记录A\*算法中的g(n)和h(n)

hash : 该状态的哈希值

函数：

operator<:重载小于号，因为STL优先级队列需要里面的元素有<操作。

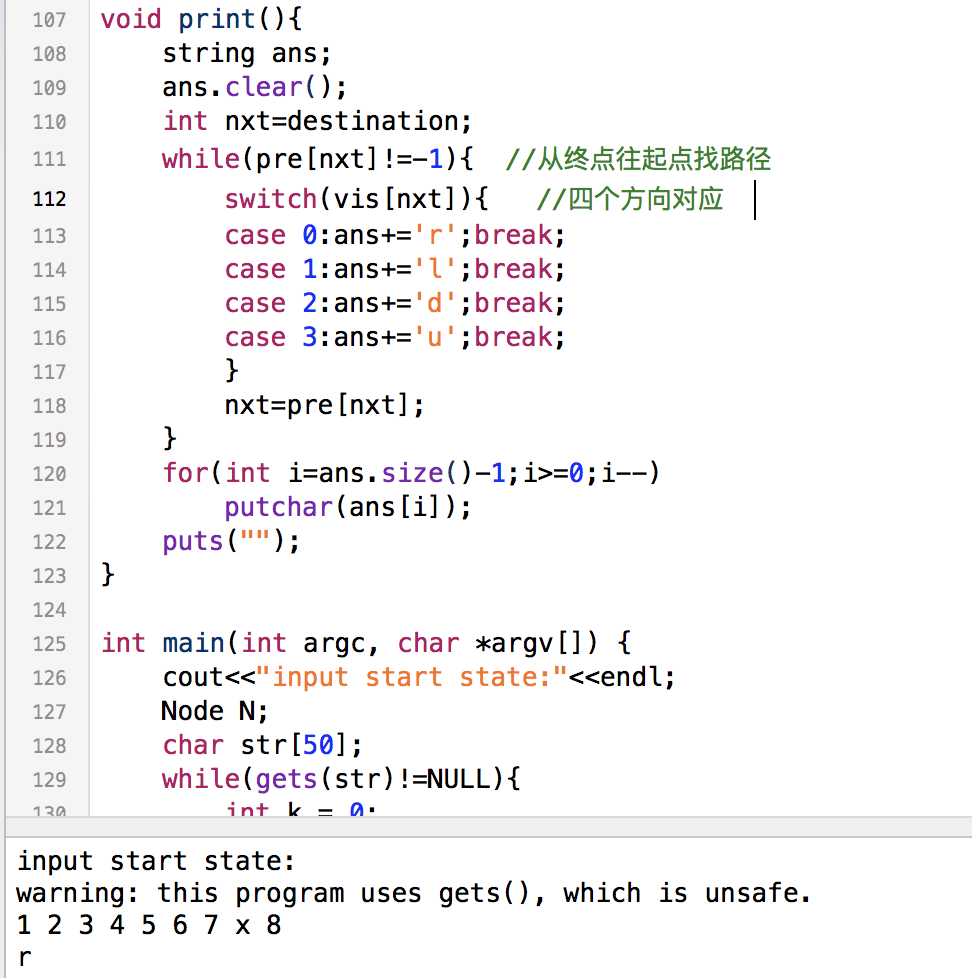
Bool isRightState: 判断该状态是否合法

**(3)实验结果:**

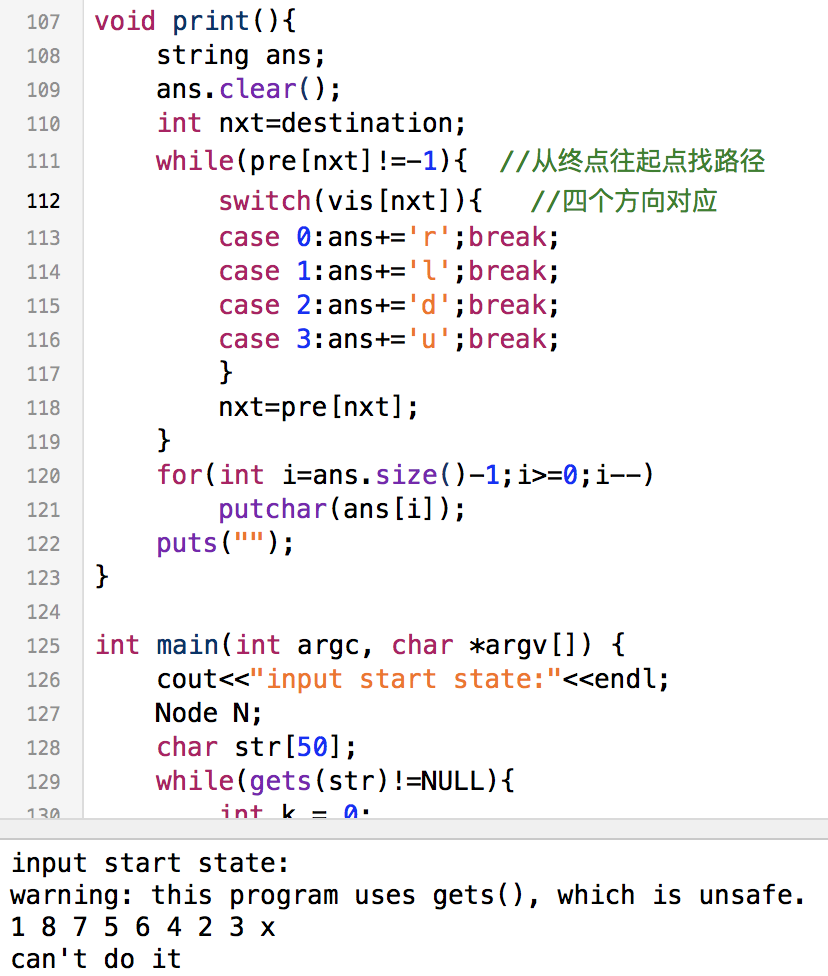
可以快速计算八数码问题

**(4)系统中间及最终结果**

例子1：显然对于这组初始状态只需r操作就可以到目标状态。



例子2：对于初始状态无解的输出。



例子3：对较复杂状态的输出(l:左,r:右,u:上,d:下)



**参考文献**

[1] Stuart J.Russell , Peter Norvig《人工智能-一种现代的方法(第三版)》

[2] Brent Chaters 《精通搜索分析》

[3]部分代码参考自http://blog.csdn.net/damotiansheng/article/details/40017107

**附录 代码及其注释**

**#include <iostream>**

**#include <algorithm>**

**#include <string>**

**#include <vector>**

**#include <queue>**

**#include <math.h>**

**using namespace std;**

**struct Node{**

**int state[3][3];//该节点的状态**

**int x,y;//记录空位置的xy坐标**

**int h,g;//f(n) = g(n) + h(n)**

**int Hash;//该节点的哈希值**

**bool operator<(const Node N)const{**

**if(h==N.h)**

**return g>N.g;**

**else**

**return h>N.h;**

**}**

**bool isRightState(){**

**if(x>=0&&x<=2&&y>=0&&y<=2)**

**return true;**

**else**

**return false;**

**}**

**}s,u,v,tt;**

**int HASH[9] = {1,1,2,6,24,120,720,5040,40320};**

**int destination = 322560;**

**/\***

**destination state:**

**1 2 3**

**4 5 6**

**7 8 0**

**\*/**

**int vis[400000];**

**int pre[400000];**

**int way[4][2] = {{0,1},{0,-1},{1,0},{-1,0}};**

**int get\_hash(Node tmp){ //得到当前状态HASH值**

**int a[9],k=0;**

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

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

**a[k++]=tmp.state[i][j];**

**int res = 0;**

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

**int k = 0;**

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

**if(a[j]>a[i])**

**k++;**

**res+=HASH[i]\*k;**

**}**

**return res;**

**}**

**bool isok(Node tmp){ //判断是否有解**

**int a[9],k=0;**

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

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

**a[k++]=tmp.state[i][j];**

**int sum = 0;**

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

**for(int j = i+1 ; j < 9 ; ++j)**

**if(a[j]&&a[i]&&a[i]>a[j])**

**sum++;**

**return !(sum&1);**

**}**

**int get\_h(Node tmp){ //得到h值**

**int ans = 0;**

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

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

**if(tmp.state[i][j])**

**ans+=abs(i-(tmp.state[i][j]-1)/3)+abs(j-(tmp.state[i][j]-1)%3);**

**return ans;**

**}**

**void astar(){**

**priority\_queue<Node> que;**

**que.push(s);**

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

**u = que.top();**

**que.pop();**

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

**v = u;**

**v.x+=way[i][0];**

**v.y+=way[i][1];**

**if(v.isRightState()){**

**swap(v.state[v.x][v.y], v.state[u.x][u.y]);**

**v.Hash=get\_hash(v);**

**if(vis[v.Hash]==-1&&isok(v)){**

**vis[v.Hash]=i;**

**v.g++;**

**pre[v.Hash]=u.Hash;**

**v.h=get\_h(v);**

**que.push(v);**

**}**

**if(v.Hash==destination)**

**return ;**

**}**

**}**

**}**

**}**

**void print(){**

**string ans;**

**ans.clear();**

**int nxt=destination;**

**while(pre[nxt]!=-1){ //从终点往起点找路径**

**switch(vis[nxt]){ //四个方向对应**

**case 0:ans+='r';break;**

**case 1:ans+='l';break;**

**case 2:ans+='d';break;**

**case 3:ans+='u';break;**

**}**

**nxt=pre[nxt];**

**}**

**for(int i=ans.size()-1;i>=0;i--)**

**putchar(ans[i]);**

**puts("");**

**}**

**int main(int argc, char \*argv[]) {**

**cout<<"input start state:"<<endl;**

**Node N;**

**char str[50];**

**while(gets(str)!=NULL){**

**int k = 0;**

**memset(vis,-1,sizeof(vis));**

**memset(pre,-1,sizeof(pre));**

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

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

**if((str[k]<='9'&&str[k]>='0')||str[k]=='x'){**

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

**s.state[i][j]=0;**

**s.x=i;**

**s.y=j;**

**}**

**else**

**s.state[i][j]=str[k]-'0';**

**}**

**else**

**j--;**

**k++;**

**}**

**}**

**if(!isok(s)){**

**cout<<"can't do it"<<endl;**

**continue;**

**}**

**s.Hash = get\_hash(s);**

**if(s.Hash == destination){**

**puts(" ");**

**continue;**

**}**

**vis[s.Hash] = -2;**

**s.g = 0;**

**s.h = get\_h(s);**

**astar();**

**print();**

**}**

**return 0;**

**}**