

合肥学院
计算机科学与技术系

课程设计报告

2016 ~ 2017 学年第 2 学期

课	程	数据结构与算法
课 程 设 计 名 称		八数码问题求解
学 生 姓 名		曹志
学 号		
专 业 班 级		软件工程 2 班
指 导 教 师		王骏

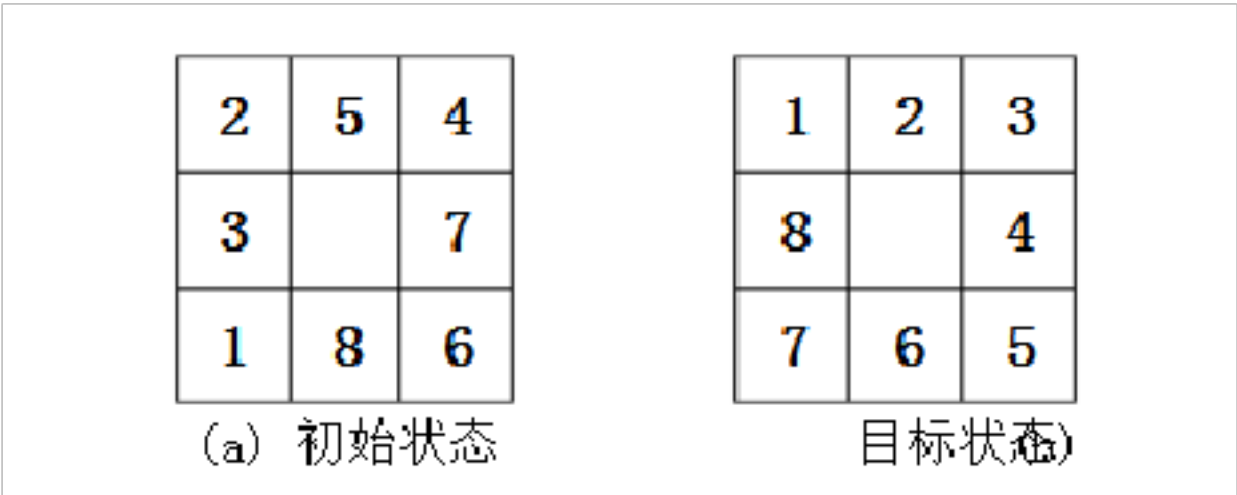
2017 年 2 月

题目

八数码问题求解

【问题描述】

八数码问题：在 3×3 的方格棋盘上，摆放着 1 到 8 这八个数码，有 1 个方格是空的，其初始状态如图 1 所示，要求对空格执行空格左移、空格右移、空格上移和空格下移这四个操作使得棋盘从初始状态到目标状态。



图一

请使用一种盲目搜索算法（深度或广度优先搜索）和一种启发式搜索方法 编程求解八数码问题，并对两种算法的搜索步骤，搜索时间进行分析对比。

1、问题分析和任务定义

1.1 问题分析

由题目知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步骤。所谓问题的一个状态就是棋子在棋盘上的一种摆法。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。要求分别采用广度优先搜索和启发式搜索两种方法对八数码问题进行求解。且棋盘通过空格的上、下、右四种操作来改变状态。所以需要解决的问题为每个状态的表示和每个状态变化的操作，设计出适合相应搜索的数据结构，完成相应算法思想。

1.2 任务定义

- (1) 完成棋盘状态表示。
- (2) 完成空格上下左右移动操作变化的表示。
- (3) 分别完成适应于广度优先搜索，启发式搜索的数据结构设计。
- (4) 得出两种搜索的搜索路径，及时间进行对比分析。

2、数据结构的选择和概要设计

2.1 广度优先搜索

(1) 广度优先搜索思想介绍

从初始节点 **h** 开始逐层的对其进行扩展，并考察其是否为目的节点，若不是将其放入待考察队列中，在第 **n** 层节点没有完全进行考察并扩展完成前，不对第 **n+1** 层进行扩展。队列中节点总是按照先后顺序进行排列，先进的排在前面，后进的排在后面。

由此可知我们可以通过队列的思想完成搜索，其具体搜索过程如下。

- 1) 初始化头结点 **front**, **front=real**;
 - 2) 若 **front==null**, 问题无解, 程序结束。
 - 3) 对 **front** 指向的节点进行考察，若为目的节点程序结束。
 - 4) 判断 **front** 指向的节点的空格是否可以上下左右移动若可以，扩展节点置于队尾。
- 重复操作(2)。

(2) 数据结构的选择和概要设计

1) 棋盘状态的表示

因为每一个棋盘状态是静止且确定的所以采取一个一维数组便可将其状态表示出来，数组按照下标顺序从左至右从上至下依次对应。空格可由零来表示。

2	5	4
3		7
1	8	6

图二

此状态可表示为 `int num[9]={2,5,4,3,0,7,1,8,6}`。

2)搜索结构设计

根据算法要求，需要对棋盘不断的进行动态扩展，且盲目搜索产生的数据巨大，所以不能采用静态链表或数组存储，因此选择采用动态链表来存储搜索过程中的每一个状态。

因为搜索是根据一种状态对其进行扩展，直至达到目的状态，由于要返回一个最优路径因此需要确定扩展关系，我选择用一个 `*parent` 来记录此关系。此时存储结构可由此图表示

图三

当扩展至目的状态时可通过 `parent` 指针找到路径。由此我们可以设置节点类型为

```
typedef struct Node
{
    struct Node *parent;
    int num[9];
    Node *next;
}QNode;
```

2.2 启发式搜索

(1) 启发式搜索的思想介绍

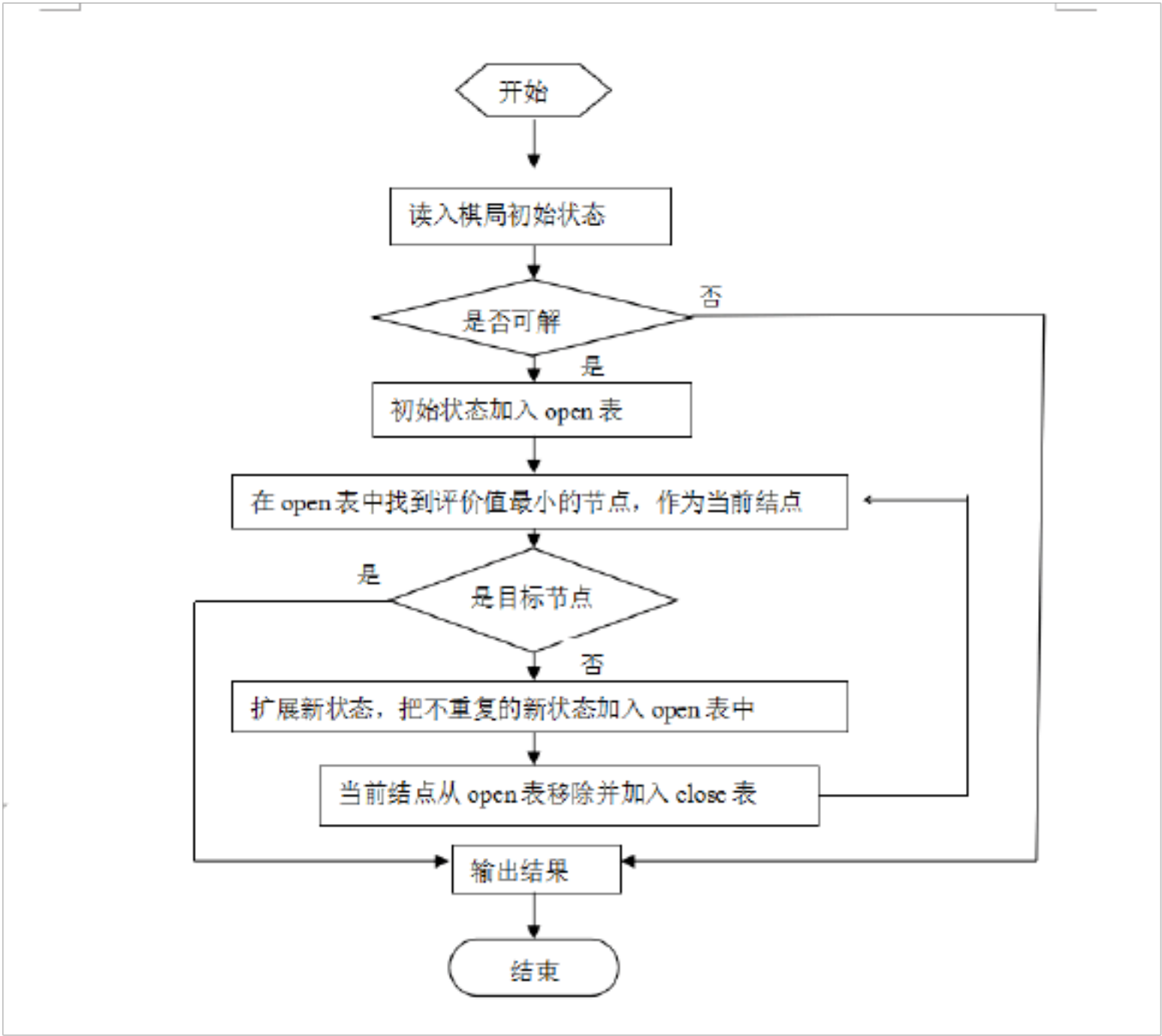
搜索算法可分为两大类：无信息的搜索算法和有信息的搜索算法。无信息的搜索又称盲目搜索，盲目搜索不考虑节点好坏，而对于八数码问题的解决过程是有迹可循的，我们通过是否接近目标状态来判断节点的好坏，因此可以通过启发式搜索中的 `A*` 算法来解决这个问题。

简单来说 A*就是将估值函数分成两个部分，一个部分是路径价值，另一个部分是一般性启发价值，合在一起算估整个结点的价值。在本实验中使用 A*算法求解。A*搜索是一种有效的搜索算法，它把到达节点的耗散 $g(n)$ 和从该节点到目标节点的消耗 $h(n)$ 结合起来对节点进行评价： $f(n)=g(n)+h(n)$ 。

由此设计如下

- (1) 把起始节点放到 OPEN 表中，并计算节点 S 的 $f(S)$
- (2) 如果 OPEN 是空表，则失败退出，无解；
- (3) 从 OPEN 表中选择一个 f 值最小的节点 i 。如果有几个节点值相同，当其中有一个为目标节点时，则选择此目标节点；否则就选择其中任一节点作为节点 i ；
- (4) 把节点 i 从 OPEN 表中移出，并把它放入 CLOSED 的已扩展节点表中；如果 i 是个目标节点，则成功退出，求得一个解；
- (5) 扩展节点 i ，生成其全部后继节点。对于 i 的每一个后继节点 j ：计算 $f(j)$ ；如果 j 既不在 OPEN 表中，又不在 CLOSED 表中，则用估价函数 f 把它添入 OPEN 表中。从 j 加一指向其父节点 i 的指针，以便一旦找到目标节点时记住一个解答路径；如果 j 已在 OPEN 表或 CLOSED 表中，则比较刚刚对 j 计算过的 f 和前面计算过的该节点在表中的 f 值。如果新的 f 较小，则 (I) 以此新值取代旧值。(II) 从 j 指向 i ，而不是指向他的父节点。(III) 如果节点 j 在 CLOSED 表中，则把它移回 OPEN 表中。转向 (2)。

流程图如下



图四

(2) 数据结构的选择和概要设计

由于搜索特点，采用的基础存储结构和广度优先搜索相同，不同之处为启发式搜索使用两条链表，所以设计一个表结构，用来声明两条链表。

节点结构

```
typedef struct Node
{
    double g,f;
    struct Node *parent;
    int num[9];
```

```
}Node;
```

表结构

```
typedef struct Stack
{
    Node * npoint;//指向状态节点，相当于表节点的数据域。
    struct Stack * next;
}Stack;
```

3、详细设计和编码

3.1 广度优先搜索

(1) 相关函数

1) 空格移动函数

```
int move_up(int num[]);
int move_down(int num[]);
int move_left(int num[]);
int move_right(int num[]);
```

以上移为例，找到 0 的位置将其与下标比其小三的值交换，且下标为 0,1,2 的数字不可移动。

2) 搜索过程函数

int bfs(int init[],int target[]); 接收初始状态和目的状态总领搜索过程。

int cansolve(int num1[], int num2[]); 判断两个数列的逆序数奇偶性，从而判断八数码问题是否可解

void get_num(QNode *node, int temp[]); 将节点中的数组获取出来。

void set_num(QNode *node, int temp[]); 设置节点中数组的数据。

void print(QNode *node); 输出节点中信息

(2) 算法伪码

输入初始状态数组 unit，目标状态数组 target

If(问题可解)

While(队头指针非空时&&未找到目的节点)

 If(队头节点为目的节点)

 结束循环;

 End if

 对队头节点进行扩展，将扩展节点加入队尾;

 队头指针后移;

End while

End if

3.2 启发式搜索设计

(1) 所用函数

int Equal(Node * suc,Node * goal); 判断节点是否相等，相等，不相等。

Node * Belong(Node * suc,Lstack * list)判断节点是否属于 OPEN 表或 CLOSED 表，

是则返回节点地址，否则返回空地址。

void Putinto(Node * suc,Lstack * list)把节点放入 OPEN 或 CLOSED 表中。

double Fvalue(Node suc, Node goal, float speed)计算 f 值。

double Distance(Node suc, Node goal, int i)计算方格的错位距离。

int BelongProgram(Lnode * suc ,Lstack * Open ,Lstack * Closed ,Node goal ,float speed)判断子节点是否属于 OPEN 或 CLOSED 表并作出相应的处理。

int Canspread(Node suc, int n)判断空格可否向该方向移动，表示空格向上向下向左向右移。

void Spreadchild(Node * child,int n)扩展 child 节点的子节点 n 表示方向表示空格向上向下向左向右移。

int Spread(Lnode * suc, Lstack * Open, Lstack * Closed, Node goal, float speed)扩展后继节点总函数。

Node * Process(Lnode * org, Lnode * goal, Lstack * Open, Lstack * Closed, float speed)

总执行函数。

void qf(int init[9],int target[9])启发式搜索总函数。

(2) 搜索过程伪码

输入：初始状态，目标状态

输出：从初始状态到目标状态的一系列过程

算法描述：

Begin:

 读入初始状态和目标状态，并计算初始状态评价函数值 f；

 根据初始状态和目标状态，判断问题是否可解；

 If(问题可解)

 把初始状态加如 open 表中；

 While (未找到解&&状态表非空)

 ①在 open 表中找到评价值最小的节点，作为当前结点,；

 ②判断当前结点状态和目标状态是否一致，若一致，跳出循环；否则跳转到③；

 ③对当前结点，分别按照上、下、左、右方向移动空格位置来扩展新的状态结点，并在并计算新扩展结点的评价值 f 并记录其父节点若；

 ④对于新扩展的状态结点，判断其是否重复，若不重复，把其加入到 open 表中；

 ⑤把当前结点从 open 表中移至 Close 表；

 End while

End if

 输出结果；

End

4、上机调试过程

4.1 遇到的问题和分析

(1) 本次课程设计过程是两种搜索方式分开开发，因此在综合在一起的时候两种方式的代码合在一起略显杂糅，程序可读性较差。

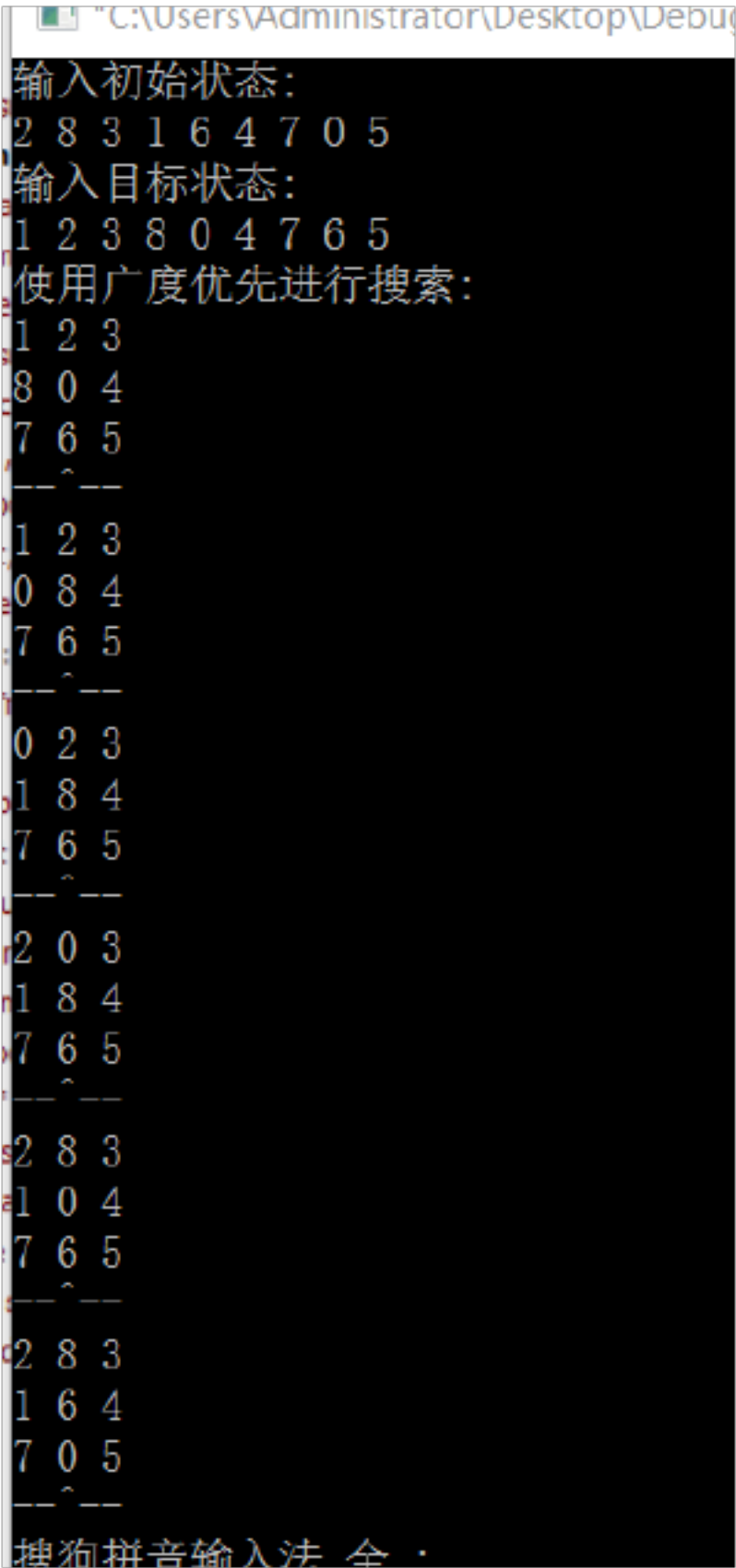
(2) 在做启发式搜索时，发现效率并未达到预期，查阅资料得知 h 值取得不好，没有完全切合搜索规律，h 值应为每个方格到其正确位置的路径之和，而我仅统计的是错位之和同时发现若 h 值取得越大其搜索速度越快单难以保证得出的是最优解。因此我设计了一个系数 1000 来进行加速。

(3) 在设计广度优先搜索时指针操作出现失误，导致进度一度停滞，后放弃改为数组操作才完成程序

4.2 时空效率分析

- (1) 广度优先搜索是一种十分浪费空间的搜索方法，有时会陷入死循环。同时在时间效率上广度优先搜索效率不稳定，若搜索深度过大则时间效率异常缓慢。但可确定找到最优解。
- (2) 启发式搜索效率较稳定，当搜索深度较小时时间效率不及广度优先但在深度搜索表现优异，同时启发式搜索有选择的创建节点，大大节省空间效率。

5、测试结果及其分析



图五

```

S 1 0 0
  ^
  ---
  共花费时间4.0000ms, 得出最优解共需6个节点,
  使用启发式进行搜索:
  1 2 3
  8 0 4
  7 6 5
  ^
  ---
  1 2 3
  0 8 4
  7 6 5
  -
  ---
  0 2 3
  1 8 4
  7 6 5
  ^
  ---
  2 0 3
  1 8 4
  7 6 5
  ^
  ---
  2 8 3
  1 0 4
  7 6 5
  ^
  ---
  2 8 3
  1 6 4
  7 0 5
  ^
  ---
  搜索所用时间:109.0000ms, 得出路径所需节点6
  搜狗拼音输入法 全 :
```

图六

图七

```
"C:\Users\Administrator\Desktop\Debug\课程设计改进版.exe"
输入初始状态:
1 2 3 4 5 6 7 8 0
输入目标状态:
1 2 3 4 0 5 6 7 8
使用广度优先进行搜索:
1 2 3
4 0 5
6 7 8
--^--
1 2 3
4 5 0
6 7 8
--^--
1 2 3
4 5 8
6 7 0
--^--
1 2 3
4 5 8
6 0 7
--^--
1 2 3
4 5 8
0 6 7
--^--
1 2 3
```

图八

图九

```
1 2 3
0 5 6
4 7 8
--^--
1 2 3
4 5 6
0 7 8
--^-
1 2 3
4 5 6
7 0 8
--^--
1 2 3
4 5 6
7 8 0
--^-
搜索所用时间:196.000000ms, 得出路径所需节点15
Press Enter key to exit!Press any key to continue
```

图十

图十一

```
7 8 0
---^---
共花费时间48.0000ms, 得出最优解共需9个节点,
使用启发式进行搜索:
0 1 3
8 2 5
4 7 6
---^---
1 0 3
8 2 5
4 7 6
---^---
1 2 3
8 0 5
4 7 6
---^---
1 2 3
0 8 5
4 7 6
---^---
1 2 3
4 8 5
0 7 6
---^---
1 2 3
4 8 5
7 0 6
---^---
1 2 3
```

图十二

图十

(1) 实验结果

实验数据	广度优先		启发式搜索	
	时间	最优节点数	时间	最优节点数
初始状态 283164705 目标状态 123804765	4ms	6	109ms	6
初始状态 254307186 目标状态 123804765	无解 0	无解	无解	无解
初始状态 123456780 目标状态 123405678	4399ms	15	196ms	15
初始状态 123456780 目标状态 013825476	48ms	9	128ms	9
初始状态 123456780 目标状态 123045678	12537ms	16	300ms	28
初始状态 183265470 目标状态 836107245	112ms	11	347ms	25

表一

(2) 结果分析。

当最优解步骤较小时，广度优先时间效能更好。造成这种现象的原因为当搜索深度小时，启发式搜索由于有较多的查重和链表操作导致时间性能下降，而当搜索深度较深时启发式搜索对节点优劣的考察使其在时间上展现出较大优势。但启发式搜索可能无法获取最优解。

6、用户使用说明

在输入八数码状态时将八数码上数字从上至下从左至右，依次输入其中空格用 0 代替，且每个数字用空格隔开。

7、参考文献

[1] 钱莹. 基于广度优先的八数码问题解决方案. 电脑学习, 2008:45-46
[2] 张鸿. 人工智能中求解八数码问题算法与实现. 软件导刊, 2009:62-64

8、附录

```
#include "stdio.h"
#include <stdlib.h>
#include <time.h>
#include <math.h>
```

```

#include<process.h>

typedef struct Node
{
    struct Node *parent;
    double f,g;
    int num[9];
    // int step;
    Node *next;
}QNode,*Lnode;

typedef struct Stack
{//OPEN CLOSED 表结构体
    Node * npoint;
    struct Stack * next;
}Stack,* Lstack;

int cansolve(int num1[], int num2[]);
void input(int num[]);
int move_up(int num[]);
int move_down(int num[]);
int move_left(int num[]);
int move_right(int num[]);
void get_num(QNode *node, int temp[]);
void set_num(QNode *node, int temp[]);
void print(QNode *node);
int isequal(QNode *node, int temp[]);
int bfs(int init[9], int target[9]);
Node * Minf(Lstack * Open);
Node * Belong(Node * suc,Lstack * list);
void Putinto(Node * suc,Lstack * list);
int Equal(Node * suc,Node * goal);
int Canspread(Node suc, int n);
void Spreadchild(Node * child,int n);
void Putinto(Node * suc,Lstack * list);
double Distance(Node suc, Node goal, int i);
void Spreadchild(Node * child,int n);
int BelongProgram(Lnode * suc ,Lstack * Open ,Lstack * Closed ,Node goal ,float
speed);
Node * Process(Lnode * org, Lnode * goal, Lstack * Open, Lstack * Closed, float
speed);
void qf(int init[9],int target[9]);
int main()

```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/827035045115006066>