



华南师范大学本科生实验报告

姓名： 苏春铭 学号： 20162180146

院系： 计算机学院 专业： 计算机科学与技术

年级： 2016 班级： 数据库 2 班

小组实验任务分工： 独立完成

实验时间： 2019 年 6 月 11 日

实验名称： 启发式搜索算法

指导老师： 陈振洲

实验课程： 人工智能导论（综设型实验）

启发式搜索算法实验报告

第一部分 实验内容

1. 实验目标

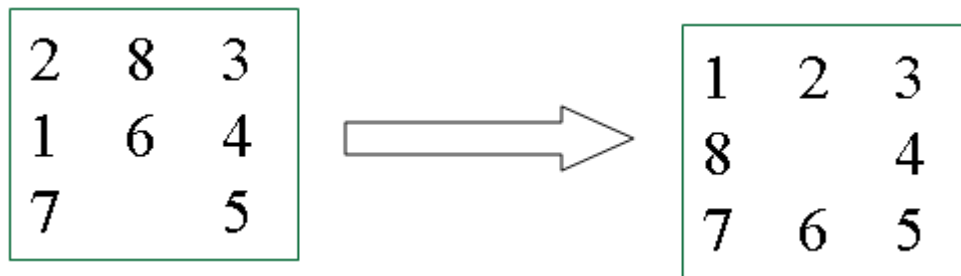
(1) 综合应用“深度优先搜索”、“宽度优先搜索”、“启发式搜索”这三种人工智能搜索技术的基本知识以及程序设计的相关知识。

(2) 通过设计一个八数码程序，学习、了解、比较相关搜索算法的特点，进一步加深对人工智能课程相关知识的理解。

2. 实验任务

九宫排字问题（又称八数码问题）是人工智能当中有名的难题之一。问题是在 3×3 方格盘上，放有八个数码，剩下第九个为空，每一空格其上下左右的数码可移至空格。问题给定初始位置和目标位置，要求通过一系列的数码移动，将初始位置转化为目标位置。

如：



3. 实验设备及环境

实验设备：MI Air 13.3

系统版本：Windows10 X64

硬件：Intel® Core™ i5-7200U @ 2.5GHz 2.71GHz

RAM 8.00G

4. 实验主要步骤

- (1) 根据实验目标，明确实验的具体任务；
- (2) 设计求解问题的流程图，并编写程序实现算法；
- (3) 实验后的心得体会。

启发式搜索算法实验报告

第二部分 问题及算法

1. 问题描述

(1) 九宫排字问题（又称八数码问题）是人工智能当中有名的难题之一。问题是在 3×3 方格盘上，放有八个数码，剩下第九个为空，每一空格其上下左右的数码可移至空格。问题给定初始位置和目标位置，要求通过一系列的数码移动，将初始位置转化为目标位置。

(2) 针对八数码问题，在 Windows 环境下用 Python 语言实现几种搜索算法(最好是图形界面)：

深度优先搜索

宽度优先搜索

启发式搜索算法 ($h_1(n) = W(n)$ “不在位”的将牌数)

启发式搜索算法 ($h_2(n) = P(n)$ 将牌“不在位”的距离和)

启发式搜索算法 ($h_3(n) = h(n) = P(n) + 3S(n)$)

(3) 随机产生或手动输入初始状态，对于同一个初始状态，分别用上面的 5 种方法进行求解，并对比结果

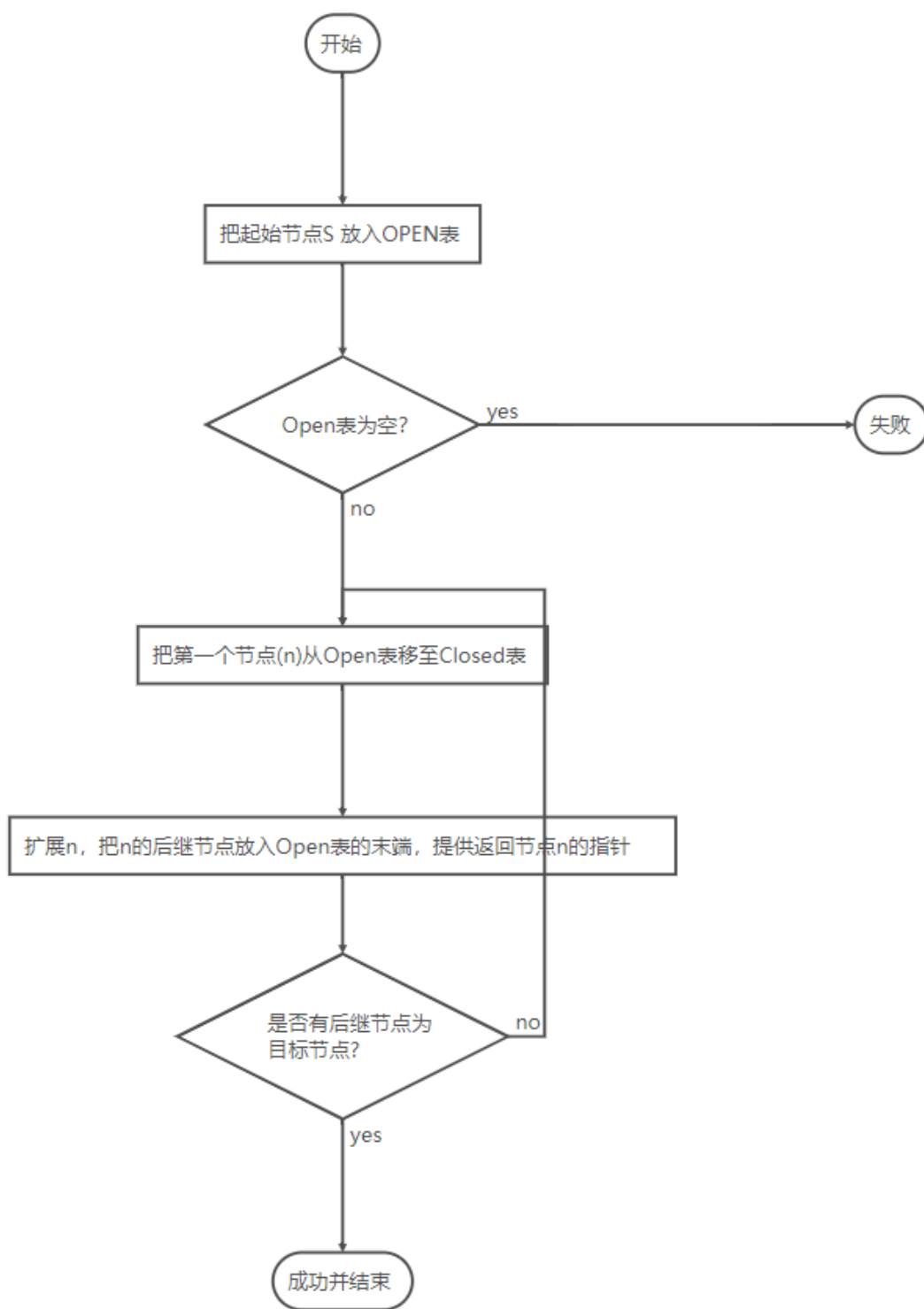
2. 算法的一般思路

BFS 伪代码如下：

1. $G := G_0(G_0 = s)$, OPEN: $=(s)$, CLOSED: $=()$;
2. LOOP: IF OPEN $=()$ THEN EXIT (FAIL);
3. $n := \text{FIRST}(\text{OPEN})$;
4. IF GOAL(n) THEN EXIT (SUCCESS);
5. REMOVE(n , OPEN), ADD(n , CLOSED);
6. EXPAND(n) $\rightarrow \{m_i\}$, $G := \text{ADD}(m_i, G)$;
7. IF 目标在 $\{m_i\}$ 中 THEN EXIT(SUCCESS);
8. ADD(OPEN, m_j), 并标记 m_j 到 n 的指针;
9. GO LOOP;

宽度优先搜索算法实现思路如流程图所示：

启发式搜索算法实验报告



DFS 伪代码如下:

1. $G: = G_0(G_0 = s)$, OPEN: $=(s)$, CLOSED: $=()$;
2. LOOP: IF OPEN= $()$ THEN EXIT (FAIL);
3. n: $= \text{FIRST}(\text{OPEN})$;

启发式搜索算法实验报告

4. IF GOAL(n) THEN EXIT (SUCCESS);
5. REMOVE(n , OPEN), ADD(n , CLOSED);
6. IF DEPTH(n) $\geq D_m$ GO LOOP;
7. EXPAND(n) $\rightarrow \{m_i\}$, G: =ADD(m_i , G);
8. IF 目标在 $\{m_i\}$ 中 THEN EXIT(SUCCESS);
9. ADD(m_j , OPEN), 并标记 m_j 到 n 的指针;
10. GO LOOP;

A 算法伪代码如下:

1. OPEN: =(s), $f(s)$: = $g(s)+h(s)$;
2. LOOP: IF OPEN=() THEN EXIT(FAIL);
3. n : =FIRST(OPEN);
4. IF GOAL(n) THEN EXIT(SUCCESS);
5. REMOVE(n , OPEN), ADD(n , CLOSED);
6. EXPAND(n) $\rightarrow \{m_i\}$,
 计算 $f(n, m_i)$: = $g(n, m_i)+h(m_i)$;
 ADD(m_j , OPEN), 标记 m_j 到 n 的指针;
 IF $f(n, m_k) < f(m_k)$ THEN $f(m_k)$: = $f(n, m_k)$,
 标记 m_k 到 n 的指针;
 IF $f(n, m_1) < f(m_1)$ THEN $f(m_1)$: = $f(n, m_1)$, 标记 m_1 到 n 的指
 针, ADD(m_1 , OPEN);
7. OPEN 中的节点按 f 值从小到大排序;
8. GO LOOP;

3. 求解问题的算法描述

BFS: 将新生成的结点存入 OPEN 表末端, 而每次从 OPEN 表头部取一个结点进行扩展, 当所扩展得到的结点为目标结点时, 搜索结束。

DFS: 和 BFS 几乎一样, 不同的是将新生成的结点插入到 OPEN 表的头部。

启发式搜索: 关键是拓展出来的结点有三种可能:

1. 之前没有生成过的新结点

启发式搜索算法实验报告

2.当前在 OPEN 中的结点

3.已经被 Expand 的结点

而启发式函数 $F(n) = g(n) + h(n)$ 在本实验中 $h(n)$ 是存储在结点对象中的，因此在以上的 2、3 情况的处理中，实际 fn 需比较的只是结点的深度变化。

4. 算法实现的关键技巧

在实现 DFS 和 BFS 中，通过 list 自带的 `extend()`、numpy 的 `all()`等函数完成了结点的插入和判断是否到达了目标状态。

在启发式搜索中，因为在结点类中增加了 `fn` 和 `wn/pn`、`inOPEN` 等成员变量，使得在判断扩展所得的结点属于哪种情况减少了许多麻烦。如在判断 $f(n,m)$ 和 $f(m)$ 的大小时，因为 `wn` 和结点本身状态有关的，即使同一个结点被两个或两个以上的结点生成，也只需要比较不同父结点生成的相同的子结点相对于根节点的深度即可，而不用重新计算 `fn` 等。

第三部分 实验结果与分析

1. 实验数据及结果

初始状态 1:

`['2' ' ' ' ' '3']`

`['1' '8' ' ' '4']`

`['7' '6' ' ' '5']`

实验结果:

方法	宽度优先	深度优先	H1 (n)	H2 (n)	H3 (n)
扩展结点数	30	9	5	4	4
生成节点数	55	16	9	6	6

初始状态 2:

`['2' ' ' ' ' '5']`

`['1' '3' ' ' '4']`

启发式搜索算法实验报告

['7' '6' '8']

实验结果：

方法	宽度优先	深度优先	H1 (n)	H2 (n)	H3 (n)
扩展结点数	6661	未得到结果	555	114	212
生成节点数	11598	未得到结果	1029	204	342

2. 实验分析及结论

由实验结果一可知，输入相同，且都能到达目标状态时，宽度优先的性能最差，深度次之，而三种启发水搜索的性能都要比非启发式搜索要高效，而且随着启发条件的改进，性能能得到提升。

由实验结果而可知，输入相同，但是深度优先可能需要很长的时间才能得到答案、甚至无法得到答案，而宽度优先虽然性能较低，但能保证到达目标结点。在实验结果二中，启发式搜索的性能明显优于非启发式搜索，而 H2 (n)、H (3) 所耗费的时间则要比 H1 (n) 要提升了许多。

因此，虽然宽度优先在大多情况下的性能较低，但是很大概率能到达目标结点；而深度优先虽然有时性能优于宽度优先，但是很可能出现过度扩展，导致长时间或无法到达目标结点；相比较之下，启发式搜索的性能整体优于非启发式搜索，且提升是非常明显的，随着启发式条件的改进，性能能够得到进一步的提升。

第四部分 心得与展望

1. 自我评价及心得体会

目前程序的程序虽然完成了提供了基本的输入、搜索、输出结果等基本功能，但是没有针对部分没有结果，或者短时间内没有答案的情况设定一个上限，如深度优先遍历没有设定遍历的深度的上限，这导致程序运行时可能会出现长时间无响应；除此之外本次实验完成的程序仍然使用控制台作为交互界面，而不是提供图形交互。

通过本次实验，提高了 Python 实际运用能力，加深了对启发式搜索的理解，同时也体会到提供适当的启发式条件能够大幅度提高搜索的性能。

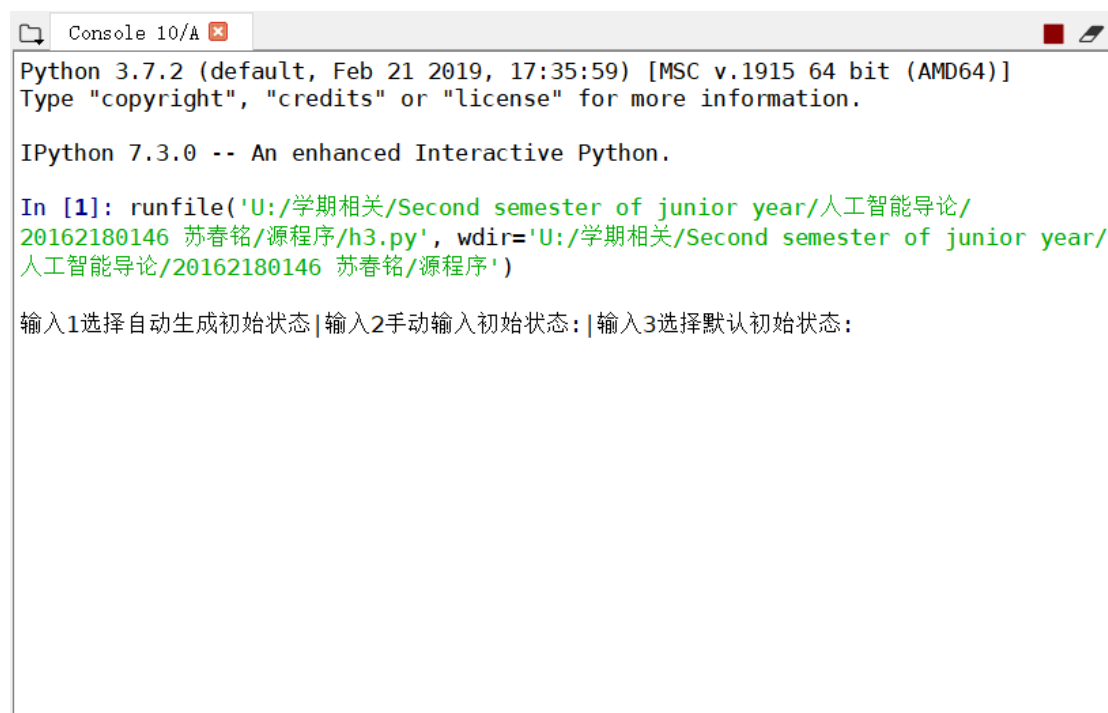
启发式搜索算法实验报告

2. 展望

针对当前完成的程序中的不足和问题，下一步的工作是为程序提供图形交互界面，以及优化搜索限制，提高搜索性能和体验。

第五部分 附录

1. 程序主要界面



```
Python 3.7.2 (default, Feb 21 2019, 17:35:59) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.3.0 -- An enhanced Interactive Python.

In [1]: runfile('U:/学期相关/Second semester of junior year/人工智能导论/
20162180146 苏春铭/源程序/h3.py', wdir='U:/学期相关/Second semester of junior year/
人工智能导论/20162180146 苏春铭/源程序')

输入1选择自动生成初始状态|输入2手动输入初始状态:|输入3选择默认初始状态:
```

2. 源程序

h2 源代码如下:

```
# -*- coding: utf-8 -*-

import numpy as np

from random import randrange

#将牌在位时的正确坐标
correctPos = {1:[0, 0], 2:[0, 1], 3:[0,2],
               4:[1, 2], 5:[2, 2], 6:[2,1],
               7:[2, 0], 8:[1, 0]}
```


启发式搜索算法实验报告

```
class State:

    def __init__(self, state, directionFlag=None, parent=None):

        self.state = state

        self.direction = ['up', 'down', 'right', 'left']

        if directionFlag:

            self.direction.remove(directionFlag)

        self.parent = parent

        self.symbol = ' '

        self.fn = 0    #耗散值

#        self.dn = 0    #当前结点的深度

        self.pn = 0    #当前结点不在位的将牌数

        self.isInOpen = False

        self.isNew = True

    def GetEmptyPos(self):

        postion = np.where(self.state == self.symbol)

        return postion

    def GetPn(self): #更新当前结点不在位将牌

        distance = 0

        temp = 0

        tempState = self.state.copy()

        for x in range(0,3):

            for y in range(0,3):

                if(' ' == tempState[x,y]):

                    continue

                temp = int(tempState[x,y])

                distance += abs(correctPos[temp][1] - y)

                distance += abs(correctPos[temp][0] - x)
```

启发式搜索算法实验报告

```
self.pn = distance
```

```
def GetDepth(self):    #计算当前结点的深度，根节点深度为 0
```

```
    depth = 0
```

```
    path = []    #临时存储指针，用于恢复结点的 parent
```

```
    path.append(self)
```

```
    while self.parent and self.parent != originState:
```

```
        path.append(self.parent)
```

```
        self = self.parent
```

```
        depth += 1
```

```
    #恢复 parent 的指向
```

```
    length = len(path)
```

```
    i = 0
```

```
    while(i < length - 1):
```

```
        path[i].parent = path[i + 1]
```

```
        i += 1
```

```
    return depth
```

```
def GetDirection(self):
```

```
    return self.direction
```

```
def GetPath(self):    #返回含有目标的搜索路径
```

```
    path = []
```

```
    path.append(self)
```

```
    while self.parent and self.parent != originState:    #记录含有目标的
```

路径

```
        path.append(self.parent)
```

```
        self = self.parent
```

启发式搜索算法实验报告

```
path.reverse()
```

```
return path
```

```
def SeekTarget(self):
```

```
    OPEN = []
```

```
    CLOSED = []
```

```
    OPEN.append(self)
```

```
    steps = 1
```

```
    while len(OPEN) > 0:
```

```
        n = OPEN.pop(0)
```

```
        n.isInOpen = False
```

```
        CLOSED.append(n)
```

```
        childStates = n.Expand() # (6) 开始扩展当前结点
```

```
        """
```

拓展出来的结点有三种可能：

1. 之前没有生成过的新结点

2. 当前在 OPEN 中的结点

3. 已经被 Expand 的结点

```
        """
```

```
        for s in childStates:
```

```
            if (s.state == s.answer).all():
```

```
                #如果 s 为目标结点，则返回路径及 step
```

```
                s.parent = n
```

```
                return s.GetPath(), steps+1
```

```
            if (True == s.isNew): #当前结点为首次出现
```

```
                s.GetPn()
```

```
                s.parent = n
```

```
                s.fn = s.GetDeepth() + s.pn
```

启发式搜索算法实验报告

```
s.isInOpen = True
s.isNew = False
OPEN.append(s)

elif (True == s.isInOpen): #当 s 当前已在 OPEN 中
    indexoS = OPEN.index(s)
    s = OPEN.pop(indexoS) #将 s 从 OPEN 中取出
    newDepth = n.GetDepth() + 1
    if(s.GetDepth() > newDepth):
        #f(n,mk)和 f(mk)之间的比较其实就是深度的比较
        s.fn = s.wn + newDepth
        s.parent = n
#    print("第二种情况")

else: #第三种情况: 当前结点已经被扩展过
    newDepth = n.GetDepth() + 1
    if(s.GetDepth() > newDepth):
        s.parent = n
        s.fn = s.wn + newDepth
        OPEN.append(s)
#    print("第三种情况")
steps += 1
OPEN.sort(key = GetFn)
# (7) OPEN 中的结点按 f 值由小到大排序
else:
    return None

def Expand(self): #扩展当前结点
    if not self.direction:
        return []
```

启发式搜索算法实验报告

```
childStates = []

boarder = len(self.state) - 1

row, col = self.GetEmptyPos()

if 'up' in self.direction and row > 0:    #判断是否可以向上移动
    s = self.state.copy()
    temp = s.copy()
    s[row, col] = s[row-1, col]
    s[row-1, col] = temp[row, col]
    news = State(s, directionFlag='down')
    childStates.append(news)

if 'down' in self.direction and row < boarder: #是否可以向下移动
    s = self.state.copy()
    temp = s.copy()
    s[row, col] = s[row+1, col]
    s[row+1, col] = temp[row, col]
    news = State(s, directionFlag='up')
    childStates.append(news)

if 'left' in self.direction and col > 0: #是否可以向左移动
    s = self.state.copy()
    temp = s.copy()
    s[row, col] = s[row, col-1]
    s[row, col-1] = temp[row, col]
    news = State(s, directionFlag='right')
    childStates.append(news)

if self.direction.count('right') and col < boarder:    #是否可以向
右移动
```

启发式搜索算法实验报告

```
s = self.state.copy()

temp = s.copy()

s[row, col] = s[row, col+1]

s[row, col+1] = temp[row, col]

news = State(s, directionFlag='left')

childStates.append(news)

return childStates


def GetFn(s):    #返回当前结点的耗散值

    return s.fn


def GenerateMatrixAuto():    #自动生成矩阵

#   NewState = np.array([[2, ' ', 5], [8, 1 , 4], [7, 6, 3]]).copy()
#   NewState = np.array([[2, 1 , 5], [8, ' ', 4], [7, 6, 3]]).copy()
    NewState = np.array([[2, ' ', 3], [1, 8 , 4], [7, 6, 5]]).copy()

#   return NewState

    emptyPosX = 0;
    emptyPosY = 1;
    X = Y = 0;

    while(1):

        X = randrange(0,3)

        Y = randrange(0,3)

        if(NewState[X][Y] != NewState[emptyPosX][emptyPosY]):

            temp = NewState[X,Y]

            NewState[X,Y] = NewState[emptyPosX,emptyPosY]

            NewState[emptyPosX,emptyPosY] = temp

            break

    print(NewState)

    return NewState
```

启发式搜索算法实验报告

```
def GenerateMatrixHand(): #手动输入矩阵

    NewState = np.array([[2, ' ', 3], [1, 8, 4], [7, 6, 5]]).copy()

    str = input("请依次输入矩阵元素，0 代表空格：")

    inputMatrix = []

    inputMatrix = str.split(",")

    emptyPos = inputMatrix.index('0')

    count = 0

    while(count < 9):

        if(count != emptyPos):

            NewState[count // 3, count % 3] = int(inputMatrix[count])

        else:

            NewState[count // 3, count % 3] = ' '

        count = count + 1

    print(NewState)

    return NewState


if __name__ == '__main__':

    while(1):

        choic = eval(input("输入 1 选择自动生成初始状态|输入 2 手动输入初始状态|

输入 3 选择默认初始状态："))

        if choic != 1 and choic != 2 and choic != 3:

            print("输入有误！","\n")

        else:

            break

    print("生成的初始状态为：")

    if(choic == 1):

        originState = State(GenerateMatrixAuto())

    elif(choic == 2):

        originState = State(GenerateMatrixHand())

    else:
```

启发式搜索算法实验报告

```
newState = np.array([[2, ' ', 5], [8, 1, 4], [7, 6, 3]]).copy()
originState = State(newState)
print(newState)

State.answer = np.array([[1, 2, 3], [8, ' ', 4], [7, 6, 5]])
s = State(state=originState.state)
path, steps = s.SeekTarget()
if path:    #打印含目标结点的路径
    print("从起始状态到目标结点的路径为：")
    for node in path:
        print(node.state)
        print("    |")
        print("    v")
    print("生成结点数为: %d" %steps)
else:
    print("目标结点不可到达")
```

参考文献

[1] 《A*算法详解》 Colin` <https://blog.csdn.net/hitwhylz/article/details/23089415>