人工智能导论作业一

学院:	软件学院
姓名:	陈子昂
学号:	2012217
邮箱:	1812432358@qq.com
时间:	2022.11.19

1. 实验题目

八数码、十五数码问题的搜索

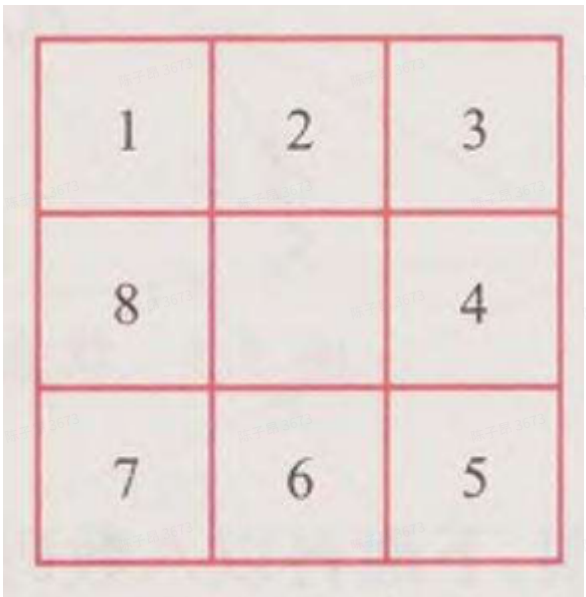
2. 实验目标

通过使用宽度优先搜索、深度优先搜索、启发式搜索完成八数码问题和十五数码问题，并进行比较

3. 原理方法

3.1 八数码问题

八数码问题（重排九宫问题）是在一个 3x3 的方格盘上放有1~8的数码，余下一格为空。空格四周上下左右的数码可移到空格。需要找到一个数码移动序列使初始的无序数码转变为一些特殊的排列。在本次实验中，我们的目标排列为中间为空，四周1-8顺时针排列的形状，如图所示。



3.2 状态空间

状态空间是利用状态变量和操作符号，表示系统或问题的有关知识的符号体系。对八数码问题，一 个 3x3的阵列便是一个合适的状态描述方式。八数码的任何一种摆法就是一个状态，所有的摆法即为状态集它们构成了一个状态空间，其数目为9!，对于操作算子，如着眼在空格上，共有上下左右四种，移动时要确保空格不会移出方格盘之外，因此要根据边界对每次操作判断算子个数。

3.3 何时有解

判断有解需要借助逆序数的概念：

首先要对3x3的棋盘进行编码，变成一个1x8的一维数组（空白不计入）。

对于棋子数列中任何一个棋子c[i]（1≤i≤8），如果有j>i且c[j]<c[i]，那么 c[j]是c[i]的一个逆序，或者c[i]和c[j]构成一个逆序对。定义棋子c[i]的逆序数为c[i]的逆序个数；棋子数列的逆序数为该数列所有棋子的逆序数总和。

通过数学引理可知，对于一组数列，每次交换相邻的数字都会导致整个序列的逆序数奇偶性发生改变。而对于八数码问题，空白处和左右的数字交换不会导致其他数字顺序的改变，对上下的数字进行移动时，两个数字相隔两个数码，相当于进行了两次相邻数码交换，奇偶性不变。因为移动不会改变序列的奇偶性，对于八数码问题，只有初始状态和目标状态的逆序数奇偶性相同，才是有解的八数码问题。

而对于十五数码问题，由于上下移动进行三次数码交换，奇偶性互换。但空白的移动上下互补，所以需要判断空白的初态和终态的位置。如果初态和终态的行数差为偶数，则会多出偶数次的上下数码移动，初态和终态奇偶性相同；行数差为奇数则需要初态和终态的奇偶性相异。

3.4 估价函数

启发式搜索能够利用与该问题有关的信息来简化搜索过程，称此类信息为启发信息。启发信息共有三种，即陈述性、过程性、控制性。其中控制性启发信息即控制搜索的策略，通常使用估价函数去衡量。估价函数由两部分组成，一部分为初始结点到目标结点的实际代价，而另一部分是到目的结点的最佳路径。前一部分可以通过生成树的树高来表示，后一部分倘若我们拥有充分的控制知识作为依据，搜索的每一步选择都是正确的，但这并不现实，因此我们需要进行经验性估计，去逼近最优情况。对于八数码问题，其估价函数可以选择为：

a. 与目的棋盘位置不符的数码数目

b. 各数码移到目的位置所需移动的距离的总和

c. 对每一对逆转数码乘以一个倍数

d. 位置不符数码数目的总和与3倍数码逆转数目

本次实验中我实现了1、2、4这三种估价函数，第三种类似于第四种故没有实现。而对于深度优先和宽度优先算法是特殊的启发式搜索，即完全不使用控制性知识进行搜索依据，故对于宽度优先搜索，搜索时放弃后一部分的搜索启发信息，对于深度优先搜索，将后一部分的比重调大（选用100倍），此时可以忽略前一部分的影响，提高搜索效率减少搜索的完备性。

3.5 A算法

A算法的伪代码如下所示，本实验基于以下代码实现：

```
procedure heuristic-_search
    open: = [ start] ;closed: = [ ] ;f( s) : =g( s) +h( s) ;
    while open # [] do
    begin
    从open表中删除第一个状态，称之为n;
    if n = 目的状态 then return(success);
        生成n的所有子状态 ;
        if n 没有任何子状态then continue;
        for n 的每个子状态 do
        case 子状态 is not already on open 表 or closed 表 ;
        begin
            计算该子状态的估价函数值;
            将该子状态加到open表中;
        end;
        case 子状态 is already on open 表 :
        if 该子状态是沿着一条比在open表已有的更短路径而到达
        then记录更短路径走向及其估价函数值;
        case 子状态 is already on closed 表 :
        if 该子状态是沿着一条比在closed表已有的更短路径而到达
        then
        begin
            将该子状态从closed表移到open表中;
            记录更短路径走向及其估价函数值;
        end ;
        case end;
        将n放入closed表中;
        根据估价函数值，从小到大重新排列open表 ;
    end; * open表中结点已耗尽
    return (failure )
```

3.6 代码实现细节

a. 八数码同十五数码的转换：设全局变量width，判断输入类型来改变width，同时需要判断有解情况的不同。

b. 一维同二维的转换：二维适合计算使用操作算子运算后的结果，也适合过程展示；一维适合计算逆序数，适合可视化展现时的使用。

c. 判断是否为目标状态：使用python的all（）函数进行全部的判断。

d. 生产子状态的方式：每一次遍历初始化四个操作算子，通过当前遍历的空格位置来判断当前遍历可以产生多少个操作可能，对每一个可能的子状态进行判断。

e. 判断更短路径：根据原有的score属性和当前使用__inspiration_score函数新计算的score值进行比较。

f. 记录路径走向：使用链表的方式，修改当前节点的parent来改变更短路径走向。

g. 记录时间和内存：在3.10的python版本中，time的记录不再使用start和end，而是使用perf_counter（）进行记录；内存需要通过os库获取当前进程并使用memory_info函数来获取进程的内存信息。

4. 详细代码

```
import os
import time
import numpy as np
import copy
import math

import psutil

def get_reverse(pre):
    inverse = 0
    global width
    for i in range(1, width ** 2):
        for j in range(i):
            try:
                if pre[i] != 0 and pre[j] != 0 and pre[i] > pre[j]:
                    inverse = inverse + 1
            except:
                pass
    return inverse

def is_reverse(pre, final):
    global width
    if width % 2 == 1:
        preCode = get_reverse(pre)
        finalCode = get_reverse(final)
        return preCode % 2 == finalCode % 2
    else:
        preCode = get_reverse(pre)
        pre_zero_index = pre.index(0) + 1
        pre_row = math.ceil(pre_zero_index / width)
        finalCode = get_reverse(final)
        final_zero_index = final.index(0)
        final_row = math.ceil(final_zero_index / width)
        if abs(pre_row-final_row) % 2 == 0:
            return preCode % 2 == finalCode % 2
```

```
        else:
            return preCode % 2 != finalCode % 2

width = 3

class Node:
    def __init__(self, data, level, parent, score):
        self.data = data
        self.level = level
        self.parent = parent
        self.score = score

class EightCode:
    global width

    def __init__(self, initial, goals, typeScout):
        # 一维变二维
        self.initial = Node(np.reshape(initial, (width, width)), 0, "None", 0)
        self.goals = Node(np.reshape(goals, (width, width)), 0, "None", 0)

        # 初始化初始节点估价数值
        self.scout = typeScout
        self.__inspiration_score(self.initial, self.goals)

        # 初始化open和close
        self.open_ = [self.initial]
        self.close_ = []
        self.lists = []

    # 估价函数
    def __inspiration_score(self, node, goals):
        # d(n)
        a = node.level
        # w(n)
        b = 0
        if (self.scout == 1):
            # 所需移动距离
            for i in range(width ** 2):
                position = np.where(node.data == i)
                x = position[0]
                y = position[1]
                position1 = np.where(goals.data == i)
                x1 = position1[0]
                y1 = position1[1]
                b = b + abs(x-x1) + abs(y-y1)
            node.score = a + b
            return node
```

```
83          elif (self.scout == 2):
84              # 位置不同
85              b = np.count_nonzero(node.data-goals.data)
86              node.score = a+b
87              return node
88          elif (self.scout == 3):
89              for i in range(width ** 2):
90                  position = np.where(node.data == i)
91                  x = position[0]
92                  y = position[1]
93                  position1 = np.where(goals.data == i)
94                  x1 = position1[0]
95                  y1 = position1[1]
96                  b = b + abs(x-x1) + abs(y-y1)
97              flatter = node.data.reshape(1, width ** 2)
98              node.score = a + b + 3 * get_reverse(flatter)
99              return node
100         elif (self.scout == 4):
101             # 宽度优先
102             node.score = a
103             return node
104         elif (self.scout == 5):
105             # 深度优先
106             b = np.count_nonzero(node.data-goals.data)
107             node.score = b
108             return node
109
110     # 展示数据函数
111     def __show_data(self, a):
112         self.lists.append(np.reshape(a.data, (1, width ** 2))[0])
113         for i in a.data:
114             print(i)
115
116     def view_array(self, a):
117         newArray = np.reshape(a.data, (1, width ** 2))[0]
118         return newArray
119
120     # 移动函数
121     def __move(self, n, position, row, col):
122         if position == "left":
123             n[row, col], n[row, col-1] = n[row, col-1], n[row, col]
124         elif position == "right":
125             n[row, col], n[row, col+1] = n[row, col+1], n[row, col]
126         elif position == "up":
127             n[row, col], n[row-1, col] = n[row-1, col], n[row, col]
128         elif position == "down":
129             n[row, col], n[row+1, col] = n[row+1, col], n[row, col]
```

```
130         return n
131
132     def __exist_both(self, move_result, close, open_):
133         for i in range(len(close)):
134             if (move_result == close[i].data).all():
135                 return True
136         for i in range(len(open_)):
137             if (move_result == open_[i].data).all():
138                 return True
139         return False
140
141     def __exists_open(self, move_result, open_):
142         if len(open_) == 0:
143             return False
144         for i in range(len(open_)):
145             if (move_result == open_[i].data).all():
146                 return open_[i]
147         return False
148
149     def __exists_close(self, move_result, close):
150         if len(close) == 0:
151             return False
152         for i in range(len(close)):
153             if (move_result == close[i].data).all():
154                 return close[i]
155         return False
156
157     def __sort_by_score(self, arr):
158         n = len(arr)
159         for i in range(n):
160             for j in range(0, n-i-1):
161                 if arr[j].score > arr[j+1].score:
162                     arr[j], arr[j+1] = arr[j+1], arr[j]
163         return arr
164
165     def find_path(self):
166         # 遍历次数
167         flag = 0
168         while self.open_:
169             flag = flag + 1
170             direction = ['up', 'down', 'right', 'left']
171             # 从open表中删除第一个状态并放入close表中
172             first_state = self.open_.pop(0)
173             self.close_.append(first_state)
174             # 如果n为目标状态则返回求解路径
175
176             if (first_state.data == self.goals.data).all():
```

```
177                 # 匹配成功
178                 resultList = []
179                 resultList.append(first_state)
180                 print("搜索完成，路径如下：")
181                 while first_state.parent != "None":
182                     resultList.append(first_state.parent)
183                     first_state = first_state.parent
184                 for j in range(len(resultList)):
185                     print(str(j)+"——>")
186                     result = resultList.pop(-1)
187                     self.__show_data(result)
188                 print(
189                     "-----------------------------------结束搜索------------------
190                 return True
191 
192             position = np.where(first_state.data == 0)
193 
194             i = position[0]
195             j = position[1]
196             length_down = first_state.data.shape[0]
197             length_right = first_state.data.shape[1]
198 
199             # 操作算子更新
200             if i == 0:
201                 direction.remove("up")
202             if j == 0:
203                 direction.remove("left")
204             if i == length_down-1:
205                 direction.remove("down")
206             if j == length_right-1:
207                 direction.remove("right")
208 
209             # 找到子状态
210             for p in range(len(direction)):
211                 first_copy = copy.deepcopy(first_state)
212                 move_result = self.__move(first_copy.data, direction[p], i, j)
213                 # 判断是否存在于open\close表
214                 if (self.__exist_both(move_result, self.close_, self.open_)):
215                     # 比较self.score和原有的score
216                     if (self.__exists_open(move_result, self.open_) != False):
217                         old = self.__exists_open(
218                             move_result, self.open_)  # 原有的
219                         score_t = self.__inspiration_score(
220                             Node(move_result, first_state.level+1, first_state,
221                         if (score_t.score < old.score):
222                             old.score = score_t.score
223                             old.parent = score_t.parent
```

```
224                     if (self.__exists_close(move_result, self.close_) != False):
225                         old = self.__exists_close(move_result, self.close_)
226                         score_t = self.__inspiration_score(
227                             Node(move_result, first_state.level+1, first_state,
228                         if (score_t.score < old.score):
229                             self.close_.remove(old)
230                             self.open_.append(old)
231                             old.score = score_t.score
232                             old.parent = score_t.parent
233                 else:
234                     # 评估节点
235                     score_t = self.__inspiration_score(
236                         Node(move_result, first_state.level+1, first_state, 0),
237 
238                     self.open_.append(score_t)
239             # open表排序
240             self.open_ = self.__sort_by_score(self.open_)
241 
242             print("第"+str(flag)+"次查找的中间项为: \n", first_state.data)
243             print("层数", first_state.level)
244             print("估价函数值", first_state.score)
245 
246 # 初始化状态
247 a = [1, 2, 8, 0, 6, 3, 7, 5, 4]
248 b = [1, 2, 3, 8, 0, 4, 7, 6, 5]
249 c = [5, 1, 2, 4, 9, 6, 3, 8, 13, 15, 10, 11, 14, 0, 7, 12]
250 d = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
251 typeMode = eval(input("请选择八数码或十五数码"))
252 if (typeMode == 8):
253     width = int(math.sqrt(len(a)))
254     typeScout = eval(input("请选择估价函数"))
255     ans = EightCode(a, b, typeScout=typeScout)
256     judge = is_reverse(a, b)
257 else:
258     width = int(math.sqrt(len(c)))
259     typeScout = eval(input("请选择估价函数"))
260     ans = EightCode(c, d, typeScout=typeScout)
261     judge = is_reverse(c, d)
262 if (not judge):
263     if width == 3:
264         print("不存在八数码解")
265     else:
266         print("不存在十五数码解")
267 else:
268     start = time.perf_counter()
269     ans.find_path()
270     end = time.perf_counter()
```

```
271     print(u'当前进程的内存使用：%.4f MB' %
272           (psutil.Process(os.getpid()).memory_info().rss / 1024 / 1024))
273     print('运行时间 : %.4f 秒' % (end-start))
274
```

4.1 测试用例

八数码问题的测试例子为：

1	2	8
	6	3
7	5	4

目标形式为：

2	3	4
1		5
8	7	6

十五数码的测试例子为：

5	1	2	4
9	6	3	8
13	15	10	11
14		7	12

目标形式为：

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

4.2 程序运行结果分析

测试终端输出：

八数码问题

a. 八数码问题 估价函数为各数码移到目的位置所需移动的距离的总和

```
第30次查找的中间项为:
 [[1 2 3]
 [0 8 4]
 [7 6 5]]
层数 10
估价函数值 [12]
搜索完成，路径如下:
0-->
[1 2 8]
[0 6 3]
[7 5 4]
1-->
[0 2 8]
[1 6 3]
[7 5 4]
2-->
[2 0 8]
[1 6 3]
[7 5 4]
3-->
[2 8 0]
[1 6 3]
[7 5 4]
4-->
[2 8 3]
[1 6 0]
[7 5 4]
```

```
 [1 6 3]
 [7 5 4]
 4-->
 [2 8 3]
 [1 6 0]
 [7 5 4]
 5-->
 [2 8 3]
 [1 6 4]
 [7 5 0]
 6-->
 [2 8 3]
 [1 6 4]
 [7 0 5]
 7-->
 [2 8 3]
 [1 0 4]
 [7 6 5]
 8-->
 [2 0 3]
 [1 8 4]
 [7 6 5]
 9-->
 [0 2 3]
 [1 8 4]
 [7 6 5]
 10-->
 [1 2 3]
 [0 8 4]
 [7 6 5]
 11-->
 [1 2 3]
 [8 0 4]
 [7 6 5]
 ----------------------------------结束搜索----------------------------------
 当前进程的内存使用：28.9219 MB
 运行时间 ：0.1310 秒
```

b. 八数码问题 估价函数为与目的棋盘位置不符的数码数目

```
 第83次查找的中间项为：
  [[1 2 3]
  [0 8 4]
  [7 6 5]]
 层数 10
 估价函数值 12
 搜索完成，路径如下：
 0-->
 [1 2 8]
 [0 6 3]
 [7 5 4]
 1-->
 [0 2 8]
 [1 6 3]
 [7 5 4]
 2-->
 [2 0 8]
 [1 6 3]
 [7 5 4]
 3-->
 [2 8 0]
 [1 6 3]
 [7 5 4]
```

```
4-->
[2 8 3]
[1 6 0]
[7 5 4]
5-->
[2 8 3]
[1 6 4]
[7 5 0]
6-->
[2 8 3]
[1 6 4]
[7 0 5]
7-->
[2 8 3]
[1 0 4]
[7 6 5]
8-->
[2 0 3]
[1 8 4]
[7 6 5]
9-->
[0 2 3]
[1 8 4]
[7 6 5]
10-->
[1 2 3]
[0 8 4]
[7 6 5]
11-->
[1 2 3]
[8 0 4]
[7 6 5]
----------------------------------结束搜索----------------------------------
当前进程的内存使用：28.8867 MB
运行时间 ：0.4182 秒
```

c. 八数码问题 估价函数为位置不符数码数目的总和与3倍数码逆转数目

```
第30次查找的中间项为:
 [[1 2 3]
 [0 8 4]
 [7 6 5]]
层数 10
估价函数值 [12]
搜索完成，路径如下:
0-->
[1 2 8]
[0 6 3]
[7 5 4]
1-->
[0 2 8]
[1 6 3]
[7 5 4]
2-->
[2 0 8]
[1 6 3]
[7 5 4]
3-->
[2 8 0]
[1 6 3]
[7 5 4]
```

```
4-->
[2 8 3]
[1 6 0]
[7 5 4]
5-->
[2 8 3]
[1 6 4]
[7 5 0]
6-->
[2 8 3]
[1 6 4]
[7 0 5]
7-->
[2 8 3]
[1 0 4]
[7 6 5]
8-->
[2 0 3]
[1 8 4]
[7 6 5]
9-->
[0 2 3]
[1 8 4]
[7 6 5]
10-->
[1 2 3]
[0 8 4]
[7 6 5]
11-->
[1 2 3]
[8 0 4]
[7 6 5]
----------------------------------结束搜索----------------------------------
当前进程的内存使用：28.8359 MB
运行时间 : 0.2015 秒
```

d. 八数码问题 宽度优先

```
第862次查找的中间项为：
 [[1 2 3]
 [7 8 4]
 [0 6 5]]
层数 11
估价函数值 11
搜索完成，路径如下：
0-->
[1 2 8]
[0 6 3]
[7 5 4]
1-->
[0 2 8]
[1 6 3]
[7 5 4]
2-->
[2 0 8]
[1 6 3]
[7 5 4]
3-->
[2 8 0]
[1 6 3]
[7 5 4]
```

```
4-->
[2 8 3]
[1 6 0]
[7 5 4]
5-->
[2 8 3]
[1 6 4]
[7 5 0]
6-->
[2 8 3]
[1 6 4]
[7 0 5]
7-->
[2 8 3]
[1 0 4]
[7 6 5]
8-->
[2 0 3]
[1 8 4]
[7 6 5]
9-->
[0 2 3]
[1 8 4]
[7 6 5]
10-->
[1 2 3]
[0 8 4]
[7 6 5]
11-->
[1 2 3]
[8 0 4]
[7 6 5]
----------------------------------结束搜索----------------------------------
当前进程的内存使用：29.3438 MB
运行时间 ：14.3518 秒
```

e. 八数码问题 深度优先

```
第459次查找的中间项为：
 [[1 2 3]
 [0 8 4]
 [7 6 5]]
层数 36
估价函数值 2
搜索完成，路径如下：
0-->
[1 2 8]
[0 6 3]
[7 5 4]
1-->
[1 2 8]
[7 6 3]
[0 5 4]
2-->
[1 2 8]
[7 6 3]
[5 0 4]
3-->
[1 2 8]
[7 0 3]
[5 6 4]
4-->
```

```
[1 3 8]
[7 6 5]
30-->
[2 0 4]
[1 3 8]
[7 6 5]
31-->
[2 3 4]
[1 0 8]
[7 6 5]
32-->
[2 3 4]
[1 8 0]
[7 6 5]
33-->
[2 3 0]
[1 8 4]
[7 6 5]
34-->
[2 0 3]
[1 8 4]
[7 6 5]
35-->
[0 2 3]
[1 8 4]
[7 6 5]
36-->
[1 2 3]
[0 8 4]
[7 6 5]
37-->
[1 2 3]
[8 0 4]
[7 6 5]
----------------------------------结束搜索----------------------------------
当前进程的内存使用：29.6055 MB
运行时间 ：6.1463 秒
```

通过表格对运行情况进行比较：

	估价函数一	估价函数二	估价函数三	宽度优先	深度优先
时间	0.1310s	0.4182s	0.2015s	14.3518s	6.1463s
占用空间	28.9219MB	28.8867MB	28.8359MB	29.3438MB	29.6055MB
路径长度	11	11	11	11	37
搜索次数	30	83	30	862	459

可以看到，宽度优先用最长的运行时间和最多的搜索次数，找到了最佳的路径长度11，这与启发式函数得到的结果相同，印证了启发式函数的正确性；深度优先的搜索时间和搜索次数优于宽度优先，但路径长度过长。而启发式搜索的性能均普遍优于宽度优先和深度优先这种没有控制性启发信息的算法，而在估价函数内，可以看到对于测试用例来说，估价函数一，即考虑所需移动距离总和的性能更加优秀，而估价函数三综合了估价函数一和二，也有较好的性能和普适性；估价函数二的性能要逊色于估价函数一和估价函数三。

十五数码问题

a. 十五数码问题 估价函数为各数码移到目的位置所需移动的距离的总和

```
第67次查找的中间项为:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11  0]
 [13 14 15 12]]
层数 13
估价函数值 [15]
搜索完成，路径如下:
0-->
[5 1 2 4]
[9 6 3 8]
[13 15 10 11]
[14  0  7 12]
1-->
[5 1 2 4]
[9 6 3 8]
[13  0 10 11]
[14 15  7 12]
2-->
[5 1 2 4]
[9 6 3 8]
[13 10  0 11]
[14 15  7 12]
3-->
[5 1 2 4]
[9 6 3 8]
[13 10  7 11]
[14 15  0 12]
4-->
[5 1 2 4]
[9 6 3 8]
[13 10  7 11]
[14  0 15 12]
```

```
9-->
[1 0 2 4]
[5 6 3 8]
[ 9 10  7 11]
[13 14 15 12]
10-->
[1 2 0 4]
[5 6 3 8]
[ 9 10  7 11]
[13 14 15 12]
11-->
[1 2 3 4]
[5 6 0 8]
[ 9 10  7 11]
[13 14 15 12]
12-->
[1 2 3 4]
[5 6 7 8]
[ 9 10  0 11]
[13 14 15 12]
13-->
[1 2 3 4]
[5 6 7 8]
[ 9 10 11  0]
[13 14 15 12]
14-->
[1 2 3 4]
[5 6 7 8]
[ 9 10 11 12]
[13 14 15  0]
----------------------------------结束搜索----------------------------------
当前进程的内存使用：29.0508 MB
运行时间 : 0.4971 秒
```

b. 十五数码问题 估价函数为与目的棋盘位置不符的数码数目

```
第65次查找的中间项为:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11  0]
 [13 14 15 12]]
层数 13
估价函数值 15
搜索完成，路径如下:
0-->
[5 1 2 4]
[9 6 3 8]
[13 15 10 11]
[14  0  7 12]
1-->
[5 1 2 4]
[9 6 3 8]
[13  0 10 11]
[14 15  7 12]
2-->
[5 1 2 4]
[9 6 3 8]
[13 10  0 11]
[14 15  7 12]
```

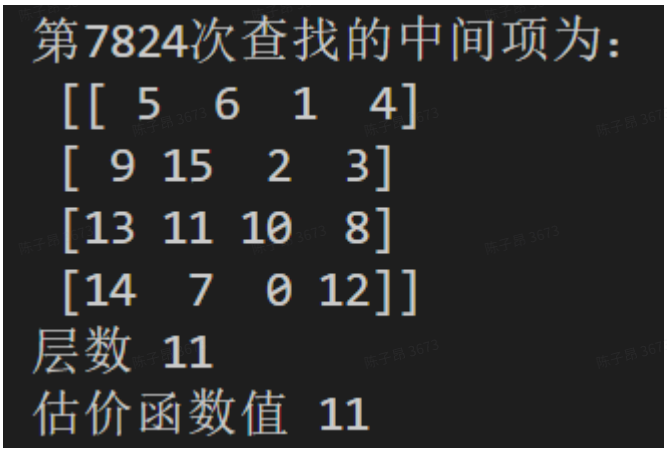
```
9-->
[1 0 2 4]
[5 6 3 8]
[ 9 10  7 11]
[13 14 15 12]
10-->
[1 2 0 4]
[5 6 3 8]
[ 9 10  7 11]
[13 14 15 12]
11-->
[1 2 3 4]
[5 6 0 8]
[ 9 10  7 11]
[13 14 15 12]
12-->
[1 2 3 4]
[5 6 7 8]
[ 9 10  0 11]
[13 14 15 12]
13-->
[1 2 3 4]
[5 6 7 8]
[ 9 10 11  0]
[13 14 15 12]
14-->
[1 2 3 4]
[5 6 7 8]
[ 9 10 11 12]
[13 14 15  0]
---------------------------------结束搜索---------------------------------
当前进程的内存使用：28.9688 MB
运行时间 : 0.3831 秒
```

c. 十五数码问题 估价函数为位置不符数码数目的总和与3倍数码逆转数目

```
第67次查找的中间项为：
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11  0]
 [13 14 15 12]]
层数 13
估价函数值 [15]
搜索完成，路径如下：
0-->
[5 1 2 4]
[9 6 3 8]
[13 15 10 11]
[14  0  7 12]
1-->
[5 1 2 4]
[9 6 3 8]
[13  0 10 11]
[14 15  7 12]
2-->
[5 1 2 4]
[9 6 3 8]
[13 10  0 11]
[14 15  7 12]
3-->
```

```
[0 1 2 4]
[5 6 3 8]
[ 9 10  7 11]
[13 14 15 12]
9-->
[1 0 2 4]
[5 6 3 8]
[ 9 10  7 11]
[13 14 15 12]
10-->
[1 2 0 4]
[5 6 3 8]
[ 9 10  7 11]
[13 14 15 12]
11-->
[1 2 3 4]
[5 6 0 8]
[ 9 10  7 11]
[13 14 15 12]
12-->
[1 2 3 4]
[5 6 7 8]
[ 9 10  0 11]
[13 14 15 12]
13-->
[1 2 3 4]
[5 6 7 8]
[ 9 10 11  0]
[13 14 15 12]
14-->
[1 2 3 4]
[5 6 7 8]
[ 9 10 11 12]
[13 14 15  0]
----------------------------------结束搜索----------------------------------
当前进程的内存使用：28.9492 MB
运行时间 ：0.4261 秒
```

d. 十五数码问题 宽度优先



由于运行时间过长，跑到第十一层基本上就跑不动了，故停止

e. 十五数码问题 深度优先

```
第348次查找的中间项为:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14  0 15]]
层数 37
估价函数值 2
搜索完成，路径如下:
0-->
[5 1 2 4]
[9 6 3 8]
[13 15 10 11]
[14  0  7 12]
1-->
[5 1 2 4]
[9 6 3 8]
[13 15 10 11]
[ 0 14  7 12]
2-->
[5 1 2 4]
[9 6 3 8]
[ 0 15 10 11]
[13 14  7 12]
```

```
32-->
[1 2 3 4]
[ 5  6  8 12]
[ 9 10  0  7]
[13 14 11 15]
[5 6 0 8]
[ 9 10  7 12]
[13 14 11 15]
36-->
[1 2 3 4]
[5 6 7 8]
[ 9 10  0 12]
[13 14 11 15]
37-->
[1 2 3 4]
[5 6 7 8]
[ 9 10 11 12]
[13 14  0 15]
38-->
[1 2 3 4]
[5 6 7 8]
[ 9 10 11 12]
[13 14 15  0]
----------------------------------结束搜索----------------------------------
当前进程的内存使用：29.4414 MB
运行时间 ：6.9653 秒
(venv) PS D:\Python实验课程>
```

通过表格对运行情况进行比较：

	估价函数一	估价函数二	估价函数三	宽度优先	深度优先
时间	0.4971s	0.3831s	0.4261s	几小时	6.9653s
占用空间	29.0508MB	28.9688MB	28.9492MB	未知	29.4414MB
路径长度	14	14	14	未知	38
搜索次数	67	65	67	成千上万	348

可以看到，宽度优先在寻找到10层11层就基本卡住了，运行时间过长，搜索次数过多。深度优先依旧有路径长度过长的问题，但是在十五数码的问题中效率明显高于宽度优先，因此在搜索层数超过大概11层时还是应当选择深度优先算法。启发式搜索的性能依旧优秀，不过在这个测试用例中，估计函数二更为占优，可以看到在估计函数的选择中仍要根据具体问题进行判断，估计函数三性能发挥更加平均，适合更多的情况。

4.3 可视化实现

可视化的实现主要基于八数码问题，通过使用pyqt进行实现。具体运行过程在确保第三方库安装完毕后运行ShowTest.py即可查看。运行方式即点击选择函数，完成后其他按钮禁用只有路径显示按钮可以点击，点击即可查看动画演示还原过程。

5. 总结心得

本次实验完成了基于A算法和启发式函数的八数码问题和十五数码问题，同时进行了启发式算法同宽度优先算法和深度优先算法的性能比较，并基于pyqt完成了八数码问题的可视化动态演示。