8数码

*潘林朝*

## 八数码介绍

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，找到一种最少步骤的移动方法，实现从初始布局到目标布局的转变。

## 实验目的

1．熟悉状态空间表示法；  
2．掌握深度优先、广度优先和等代价无信息搜索算法；  
3．掌握启发式函数设计，实现面向实际问题的A\*搜索算法；  
本文档仅介绍深度优先（**DFS**）和广度优先（**BFS**）。

## 实验内容

1. 利用**至少一种无信息**搜索算法实现八数码难题求解（可选多种）；
2. 设计至少一种启发式信息函数，利用A\*搜索实现八数码难题求解（选做题）；

## 状态空间表示

搜索问题是求解如何从开始状态转换成目标状态。  
实验的代码需要使用python表示八数码问题的状态，比如使用一些基本类型list、str表示八数码的9个数字，同时使用其它的类型表示状态包含的一些**附加信息**。因此，可以考虑将该实验的状态空间编码成python的class。

## 求解八数码问题

最简单的方法是穷举法，即穷举问题的所有状态，然后找到一条从开始状态到目标状态的合法的路径。

同时存在多条合法路径，怎么确定最优的路径？

穷举法是对搜索问题的朴素求解，一般情况下求解效率较低。我们可以考虑从两个角度加速问题求解：

1. 算法角度。有目的地穷举，如A\*算法；
2. 程序执行角度。尽可能使用原生的python数据结构，或者使用一些加速库。比如，使用numpy编写代码的同时，使用numba进行加速。

## DFS

可以使用**栈**数据结构/递归实现。

### 算法执行的具体流程

1. 创建空栈，并把初始状态压入栈中。
2. 如果栈不为空，则取出栈顶存储的状态S；如果栈为空，执行第4步。
3. 扩展S状态，如果扩展的状态是目标状态，执行第4步；否则，将并未访问的状态压入栈中，继续第2步。
4. 完成DFS，算法退出。

算法的伪代码如下所示：

# Stack is the data structure implemented by yourself.  
class DFSSolver:  
 def \_\_init\_\_(self, start\_node, end\_node, max\_depth):  
 # Initialize the parameters.  
 self.start\_node = start\_node  
 self.end\_node = end\_node  
 self.max\_depth = max\_depth  
 self.cur\_depth = 0  
 self.path = []  
  
 def node\_expand(self, cur\_node):  
 # expand the current state.  
 pass  
  
 def dfs(self):  
 """  
 Return a path.  
 """  
 open\_table = Stack()  
 open\_table.push(self.start\_node)  
 visited: dict = {} # dict  
 while not open\_table.empty():  
 cur\_node = open\_table.top()  
 open\_table.pop()  
 visited[cur\_node] = True  
 cur\_st\_list = self.node\_expand(cur\_node)  
 if cur\_st\_list is None:  
 continue  
 for st in cur\_st\_list:  
 if st not in visited:  
 if st == self.end\_node:  
 # successfully find a valid path  
 return path  
 # not in visited, st can be reconstructed for updating the parent relation  
 open\_table.push(st)  
 # save the path  
 return path  
Copy

## BFS

可以使用**队列**数据结构实现。

### 算法执行的具体流程

1. 创建空队列，并令初始状态入队。
2. 如果队列不为空，则取出队首存储的状态S；如果队列为空，执行第4步。
3. 扩展S状态，如果扩展的状态是目标状态，执行第4步；否则，令未访问的状态入队，继续第2步。
4. 完成BFS，算法退出。

算法的伪代码如下所示：

# Queue is the data structure implemented by yourself.  
class BFSSolver:  
 def \_\_init\_\_(self, start\_node, end\_node, max\_depth):  
 # Initialize the parameters.  
 self.start\_node = start\_node  
 self.end\_node = end\_node  
 self.max\_depth = max\_depth  
 self.cur\_depth = 0  
 self.path = []  
  
 def node\_expand(self, cur\_node):  
 # expand the current state.  
 pass  
  
 def bfs(self):  
 """  
 Return a path.  
 """  
 open\_table = Queue()  
 open\_table.push(self.start\_node)  
 visited: dict = {} # dict  
 while not open\_table.empty():  
 cur\_node = open\_table.front()  
 open\_table.pop()  
 visited[cur\_node] = True  
 cur\_st\_list = self.node\_expand(cur\_node)  
 if cur\_st\_list is None:  
 continue  
 for st in cur\_st\_list:  
 if st not in visited:  
 if st == self.end\_node:  
 # successfully find a valid path  
 return path  
 # not in visited, st can be reconstructed for updating the parent relation  
 open\_table.push(st)  
 # save the path  
 return path  
Copy

## Tips

1. 如何确定遍历终止条件？
2. 如何扩展状态空间的节点？
3. 如何防止遍历进入死循环？
4. 如何记录遍历的路径？
5. 可以统一一下遍历框架，思考它们不一样的地方。

## 可考虑扩展的地方（可选）

1. 实现界面可视化，如qt、turtle等。
2. 随机生成多组数据，比较不同算法的性能，并使用matplotlib、seaborn可视化实验结果。

## 实验报告

需要注意以下几点：

1. 不仅仅是把代码复制到报告文档，而是需要对编写的代码做出文字解释，描述实验思路；
2. 在实验报告展示代码的时候建议以图片形式，图片建议添加水印；
3. 实验报告引用的资料需要附上出处；
4. 需要修改实验报告封面的日期、名字等信息；
5. 一般都需要提交源码文件，即.py后缀的文件，不需要提交Python实验环境；
6. 一般需要在实验报告展示编写代码的运行结果，并对运行结果进行分析，如性能分析、正确率分析等。

## 参考资料

* <https://vijos.org/p/1360>
* <https://blog.csdn.net/u012283461/article/details/79078653>
* <https://en.wikipedia.org/wiki/Depth-first_search>
* <https://en.wikipedia.org/wiki/Breadth-first_search>
* [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)