A*算法实验报告

- 10215501403 沈桐乐

算法设计思路

A*算法的主要关注点在于其每个节点n的综合优先级$F(n) = g(n) + h(n)$,

$g(n)$表示当前节点由起始节点转移状态的开销，是可知的

$h(n)$是一个启发式函数，表示对当前节点和目标节点的开销期望

根据这个优先级的数值大小可以构建优先队列，对每一个最优的值进行节点扩展（把该节点的下一状态加入到优先队列），就是A*算法运作的基本逻辑，因此，我们需要关注如何获得$g(n)$，设计$h(n)$是合理的

题1

8-puzzle是一个经典问题，我们可以把它的所有状态空间构成一个图

对于这个图描述为：

定义一种算子k，该算子在一个$N^{3\times3}$的矩阵上操作，该操作是互换0的位置`M[i][j]`与`M[i+1][j]`,`M[i-1][j]`,`M[i][j+1]`,`M[i][j-1]`，限制条件为i+1,i-1, j+1,j-1在{1, 2, 3}当中

对于所有k成立的所有矩阵可以建立群G，G上的所有元素构成一张图，每个节点通过算子k转移到邻居节点

- 例子：当前节点$n \in G$为

```
1 2 3
4 5 6
7 0 8
```

则该节点的邻居节点是：

$k(n, \uparrow)$

```
1 2 3
4 0 6
7 5 8
```

$k(n, \leftarrow)$

```
1 2 3
4 5 6
0 7 8
```

$k(n, \rightarrow)$

```
1 2 3
4 5 6
7 8 0
```

定义图之后我们可以得到$F(n)$的第一个部分：

$g(n)$:从初始节点开始，每使用一次算子k，计数+1，初始节点为0

同样的，由于每次只能换一个位置的数字，而且只有上下或者左右两个方向，显然我们可以定义启发函数：

$h(n)$:当前节点的每一个位置，到目标节点对应位置的**曼哈顿距离**

- 例子：当前节点为

```
1 2 4
3 6 7
5 8 0
```

目标节点为

```
1 2 3
4 5 6
7 8 0
```

则$h(n)=0+0+3+3+1+3+2+1+1=14$

对于这个启发函数的设计有非常多的评价，比如教材中提到的effective branching factor，以及研究这个函数的性质等等，这里就不再展开了

实现代码详见`dmac/algorithm/puzzle.py`

题2

题2是一个天然的有向图路径k条最短路径问题，因此不需要额外的建模。

然而这道题需要的结果不只有最优的最短路径，而且要计算获得前k个最短路径，对于A*算法的实现需要做出一定的调整。

调整1:原先搜索算法遇到目标节点之后即刻终止：

现在取消终止，当遇到目标节点的时候，把成功路径记录下来，继续搜索优先队列当中的其他路径，直到优先队列为空，或者成功路径数量达到要求数量

对于本道题目来说，由于状态空间较小，这个改进策略足以完成k最短路径的任务。

对于这个问题，实现当中$F(n)$分为以下两个部分：

- $g(n)$:当前已经走过的路径长度综合
- $h(n)$，由dijkstra算法搜索到的节点到目标节点的距离

对于$h(n)$还需要做补充说明：

1. 为什么用dijkstra算法搜索的结果作为启发函数?

对于这张有向图来说，我们没有任何的先验知识获得一个节点x到目标节点n的距离，虽然提前搜索到开销较大，但是有向图较小，这样做能够保证$h(n)$一定是最优的

2. 如何做dijkstra算法搜索一个节点的$h(n)$值?

对于原始图G，我们构建一个反向图$\tilde{G}$，该图中每一条有向边$\tilde{e} = reverse(e)$，最后，我们以原先的目标节点为起始点，进行单源最短路径搜索（由于题设中不存在负环路，所以dijkstra算法是可行的）

实现代码详见`dmac/algorithm/pyramid.py`

运行结果

对于每个测试用例，第一个代码块表示测试输入，第二个代码块表示测试输出

8-puzzle

- Case 1

```
135720684
```

```
1
```

- Case 2

```
105732684
```

```
1
```

- Case 3

```
015732684
```

```
2
```

- Case 4

```
135782604
```

```
1
```

- Case 5

```
715032684
```

```
3
```

Pyramid

- Case 1

```
5 6 4
1 2 1
1 3 1
2 4 2
2 5 2
3 4 2
3 5 2
```

```
3
3
-1
-1
```

- Case 2

```
6 9 4
1 2 1
1 3 3
2 4 2
2 5 3
3 6 1
4 6 3
5 6 3
1 6 8
2 6 4
```

```
4
5
6
7
```

- Case 3

```
7 12 6
1 2 1
1 3 3
2 4 2
2 5 3
3 6 1
4 7 3
5 7 1
6 7 2
1 7 10
2 6 4
3 4 2
4 5 1
```

```
5
5
6
6
7
7
```

- Case 4

```
5 8 7
1 2 1
1 3 3
2 4 1
2 5 3
3 4 2
3 5 2
1 4 3
1 5 4
```

```
4
4
5
-1
-1
-1
-1
```

- Case 5

```
6 10 8
1 2 1
1 3 2
2 4 2
2 5 3
3 6 3
4 6 3
5 6 1
1 6 8
2 6 5
3 4 1
```

```
5
5
6
6
6
8
-1
-1
```