算法设计与分析实验报告

实验名称：周游骑士问题

一、问题陈述，相关背景、应用及研究现状的综述分析

在一张国际象棋棋盘上（8*8 方格），骑士（knight，马）位于任意一个位置。问如何才能让骑士不重不漏的经过棋盘上的每个格？本问题中已知骑士位置(m,n)，其中 0=<m,n<=8，要求给出骑士行走路径，路径可用 8*8 矩阵输出，其中值表示骑士到达此位置行走的步数（初始为 1）。

把棋盘当做 8*8 的结点集合，任意成日字型两格点间连一条无向线，那么整个棋盘就成了一个无向图。骑士对每个结点只能走一次，所以骑士的路径是一条哈密顿回路。现在的问题就是求图上任意一点作为起点的一条哈密顿通路。

结合上一次实验知道，一个图上的哈密顿通路是否存在是 NP 完全问题。所以考虑模拟骑士运动，并且在此基础上进行优化。模拟的时候可以使用递归的办法。

二、模型拟制、算法设计和正确性证明

因为图中的路径是高度对称的，即成日字型分布，且方向固定为 8 个方向，长度也固定，所以考虑用方向数组来表示图上路径。

考虑到骑士运动时不能走重复的路径，而且最后要输出骑士的运动轨迹，所以用一个 8*8 的状态数组保存格点的状态，0 的话就是格点没有被走过，i!=0 的话就是格点被遍历的顺序是第 i 个。

考虑递归模拟骑士的运动：

如果当前已经跳完了所有格点，就结束递归，输出路径。

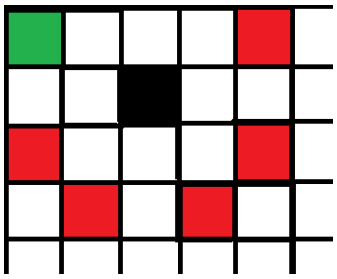
如果当前没有可达格点（没有走过的可以到达的格点）而且还没有跳完所有格点，就返回上一步。

如果当前有可达格点，就依方向数组的顺序，并且递归下一层。

这个算法可以找出所有可能的路径，相当于是模拟一个骑士在棋盘上移动，如果进入了死胡同就会回退，所以可以找出所有的路径。

只要一条路径的话就在找到第一条可行路径时退出所有循环就可以了。

在测试过程中，发现有些情况是比较特殊的：



如图，黑色格点表示当前骑士所在的格点（2，3），5 个红色格点和 1 个绿色格点是黑色格点的可达格点。

在这种情况下，更推荐黑色格点先遍历绿色格点，因为当前情况下绿色格点的可达格点就 2 个，如果骑士不走绿色格点，而走其他的红色格点的话，那绿色格点的可达格点就只有一个了（3，2），于是，绿色格点就必须是所有后续可行路径上的最后一个结点，这极大地限制的后续路径成功的可能性，所以说推荐先遍历绿色的格点，这样可以先以大概率去找可行的路径，而不是先去找小概率（以绿色格点为终点）的可行的路径（还可能多半没有）。

所以想到一个优化枚举的办法：将当前格点的各个可达格点按照其可达格点数从小到大排序遍历。优化后的递归方法为：

考虑递归模拟骑士的运动：

如果当前已经跳完了所有格点，就结束递归，输出路径。

如果当前没有可达格点而且还没有跳完所有格点，就返回上一步。

如果当前有可达格点，就依可达格点数从小到大的顺序，并且递归下一层。

三、时间和空间复杂性分析

如果时找出所有的路径的话，算法的时间复杂度是 O(8^(n*n))，因为一般一个格点的下一个格点有 8 个选择，共选 n*n 次，所以时间复杂度是 O(8^(n*n))。但实际运行中由于只需要找出一条路径，所以算法的时间复杂度会小于 O(8^(n*n))，不知道时间具体的复杂度怎么算，但是最坏情况下还是 O(8^(n*n))。

空间复杂度在于储存路径的二维数组，其占用的空间是 O(n*n)。

四、程序实现和实验测试过程

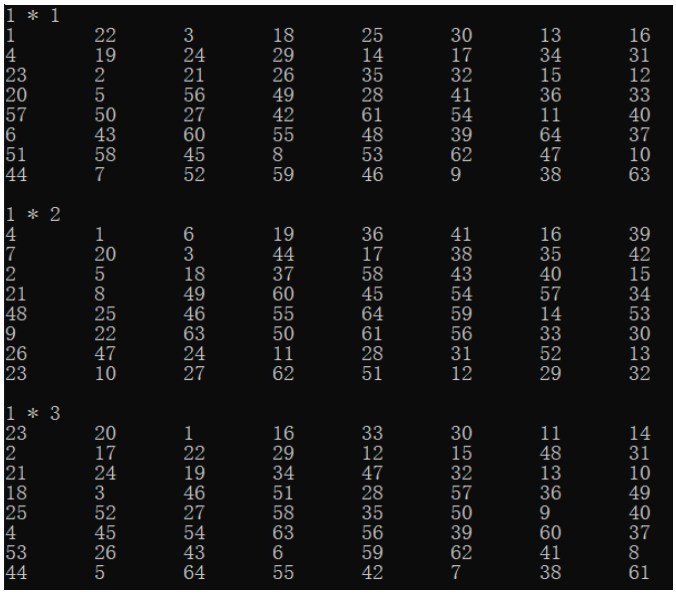
递归算法用 dfs(th,x,y,n,m)函数实现，th 表示当前走第 th 步，x,y 表示当前走到了(x,y)格点，n,m 表示格子大小。

找一个格点的可达格点数用 cnt(x,y,n,m)函数实现。

由于格点是高度对称的，有三条对称轴，所以枚举 1/8 的格点作为起点就可以了。

测试过程是先枚举出起点位置（i,j），然后 dfs 就可以了。

运行结果如下：



所有结果：

1 * 1

1 22 3 18 25 30 13 16

4 19 24 29 14 17 34 31

23 2 21 26 35 32 15 12

20 5 56 49 28 41 36 33

57 50 27 42 61 54 11 40

6 43 60 55 48 39 64 37

51 58 45 8 53 62 47 10

44 7 52 59 46 9 38 63

1 * 2

4 1 6 19 36 41 16 39

7 20 3 44 17 38 35 42

2 5 18 37 58 43 40 15

21 8 49 60 45 54 57 34

48 25 46 55 64 59 14 53

9 22 63 50 61 56 33 30

26 47 24 11 28 31 52 13

23 10 27 62 51 12 29 32

1 * 3

23 20 1 16 33 30 11 14
2 17 22 29 12 15 48 31
21 24 19 34 47 32 13 10
18 3 46 51 28 57 36 49
25 52 27 58 35 50 9 40
4 45 54 63 56 39 60 37
53 26 43 6 59 62 41 8
44 5 64 55 42 7 38 61

1 * 4

40 37 16 1 52 23 14 21
17 2 39 42 15 20 51 24
38 41 36 19 48 53 22 13
3 18 47 56 43 50 25 54
46 35 44 49 62 55 12 29
7 4 63 32 57 28 61 26
34 45 6 9 64 59 30 11
5 8 33 58 31 10 27 60

2 * 2

19 16 51 2 21 6 45 4
50 1 20 17 52 3 22 7
15 18 49 56 23 44 5 46
60 57 24 53 48 55 8 31
25 14 59 62 43 32 47 36
58 61 42 33 54 37 30 9
13 26 63 40 11 28 35 38
64 41 12 27 34 39 10 29

2 * 3

2 21 4 35 52 19 14 33
5 36 1 20 15 34 53 18
22 3 38 51 62 17 32 13
37 6 61 16 39 50 59 54
44 23 40 49 60 63 12 31
7 26 45 64 41 48 55 58
24 43 28 9 46 57 30 11
27 8 25 42 29 10 47 56

2 * 4

5 2 7 20 37 40 17 46
8 21 4 1 18 45 36 39

3 6 19 44 41 38 47 16
22 9 52 55 60 43 50 35
53 26 61 42 51 56 15 48
10 23 54 59 64 49 34 31
27 62 25 12 29 32 57 14
24 11 28 63 58 13 30 33

3 * 3
15 2 27 36 17 12 21 24
28 37 16 13 26 23 18 11
3 14 1 44 35 20 25 22
38 29 54 33 46 43 10 19
55 4 45 42 61 34 47 50
30 39 32 53 58 49 64 9
5 56 41 60 7 62 51 48
40 31 6 57 52 59 8 63

3 * 4
32 15 30 41 2 17 20 39
29 42 33 16 35 40 3 18
14 31 44 1 60 19 38 21
43 28 59 34 45 36 57 4
48 13 46 61 58 53 22 37
27 62 49 54 51 56 5 8
12 47 64 25 10 7 52 23
63 26 11 50 55 24 9 6

4 * 4
21 18 33 28 23 8 3 6
32 29 22 19 2 5 26 9
17 20 31 34 27 24 7 4
30 35 40 1 52 45 10 25
39 16 53 44 61 48 57 46
36 41 38 49 56 51 64 11
15 54 43 60 13 62 47 58
42 37 14 55 50 59 12 63

五、总结

递归算法要注意起始条件和终止条件。