



数据结构与算法 (Python版)

骑士周游问题算法分析与改进

陈斌 北京大学 gischen@pku.edu.cn

- 个层次为n的树
-
- ```
graph TD; A(()) --- B1(()); A --- B2(()); A --- B3(()); A --- B4(()); A --- B5(()); A --- B6(()); A --- B7(()); B1 --- C1(()); B1 --- C2(()); B1 --- C3(()); B1 --- C4(()); B1 --- C5(()); B1 --- C6(()); B2 --- C7(()); B2 --- C8(()); B2 --- C9(()); B2 --- C10(()); B2 --- C11(()); B2 --- C12(()); B3 --- C13(()); B3 --- C14(()); B3 --- C15(()); B3 --- C16(()); B4 --- C17(()); B4 --- C18(()); B4 --- C19(()); B4 --- C20(()); B5 --- C21(()); B5 --- C22(()); B6 --- C23(()); B7 --- C24(());
```

# 骑士周游算法改进

- ❖ 幸运的是，即便是指数时间复杂度算法也可以在实际性能上加以大幅度改进

对nbrList的灵巧构造，以特定方式排列顶点访问次序

可以使得 $8 \times 8$ 棋盘的周游路径搜索时间降低到秒级！

- ❖ 这个改进算法被特别以发明者名字命名：**Warnsdorff算法**

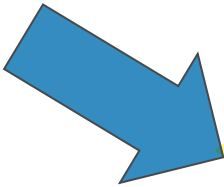
# 骑士周游算法改进

❖ 初始算法中nbrList，直接以原始顺序来确定深度优先搜索的分支次序

❖ 新的算法，仅修改了遍历下一格的次序

将u的合法移动目标棋盘格排序为：具有最少合法移动目标的格子优先搜索

```
def orderByAvail(n):
 resList = []
 for v in n.getConnections():
 if v.getColor() == 'white':
 c = 0
 for w in v.getConnections():
 if w.getColor() == 'white':
 c = c + 1
 resList.append((c,v))
 resList.sort(key=lambda x: x[0])
 return [y[1] for y in resList]
```



# 骑士周游算法改进

❖ 采用先验的知识来改进算法性能的做法，  
称作为“启发式规则heuristic”

启发式规则经常用于人工智能领域；

可以有效地减小搜索范围、更快达到目标等等；

如棋类程序算法，会预先存入棋谱、布阵口诀、  
高手习惯等“启发式规则”，能够在最短时间内  
从海量的棋局落子点搜索树中定位最佳落子。

例如：黑白棋中的“金角银边”口诀，指导程序  
优先占边角位置等等

