



数据结构与算法 (Python版)

骑士周游问题算法实现

陈斌 北京大学 gischen@pku.edu.cn

骑士周游算法实现

- ❖ 用于解决骑士周游问题的图搜索算法是深度优先搜索 (Depth First Search)
- ❖ 相比前述的广度优先搜索，其逐层建立搜索树的特点
- ❖ 深度优先搜索是沿着树的单支**尽量深入**向下搜索

如果到无法继续的程度还未找到问题解

就回溯上一层再搜索下一支

骑士周游算法实现

❖ 后面会介绍DFS的两个实现算法

一个DFS算法用于解决骑士周游问题，其特点是每个顶点仅访问一次

另一个DFS算法更为通用，允许顶点被重复访问，可作为其它图算法的基础

骑士周游算法实现

❖ 深度优先搜索解决骑士周游的关键思路

如果沿着单支深入搜索到无法继续（所有合法移动都已经被走过了）时

路径长度还没有达到预定值（ 8×8 棋盘为63）

那么就清除颜色标记，返回到上一层

换一个分支继续深入搜索

❖ 引入一个栈来记录路径

并实施返回上一层的回溯操作

骑士周游算法代码

数据结构与算法 (Python版)

当前顶点加入路径

选择白色未经过的
顶点深入

都无法完成总深度, 回
溯, 试本层下一个顶点

```
def knightTour(n, path, u, limit):  
    u.setColor('gray')  
    path.append(u)  
    if n < limit:  
        nbrList = list(u.getConnections())  
        i = 0  
        done = False  
        while i < len(nbrList) and not done:  
            if nbrList[i].getColor() == 'white':  
                done = knightTour(n+1, path, nbrList[i], limit)  
                i = i + 1  
        if not done: # prepare to backtrack  
            path.pop()  
            u.setColor('white')  
        else:  
            done = True  
    return done
```

n:层次; path:路径; u:当前顶点;
limit:搜索总深度

对所有合法移动逐一深入

层次加
1, 递
归深入

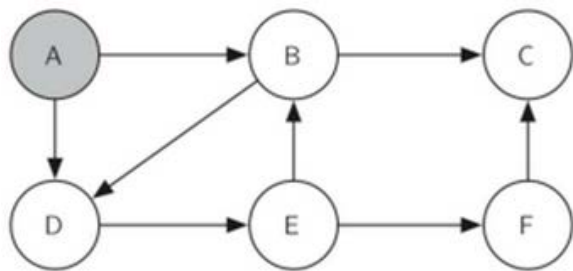


Figure 3: Start with node A

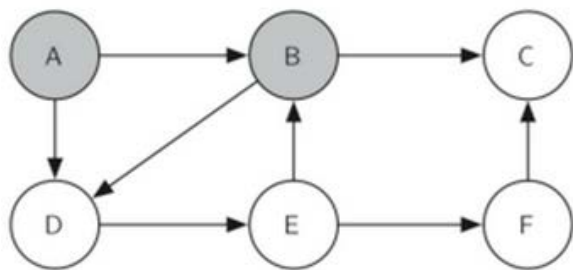


Figure 4: Explore B

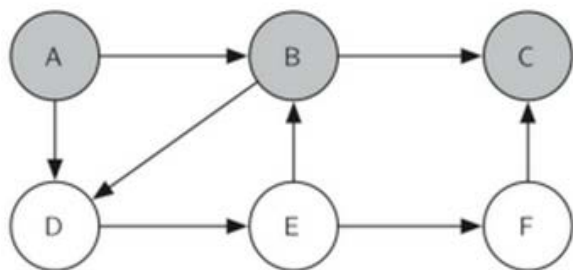


Figure 5: Node C is a dead end

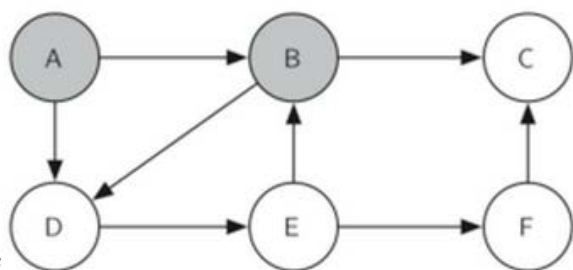


Figure 6: Backtrack to B

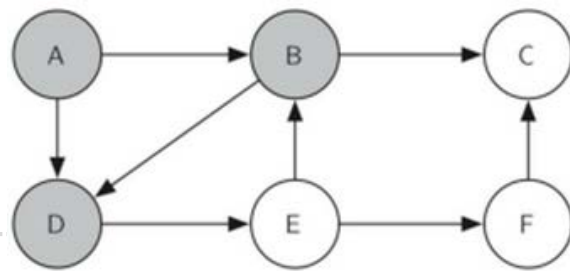


Figure 7: Explore D

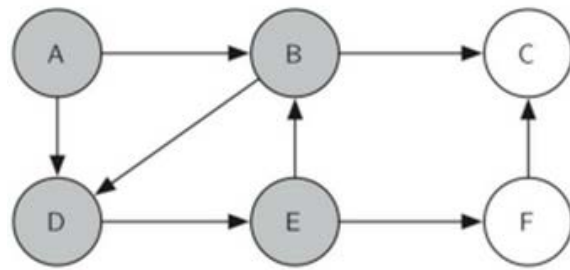


Figure 8: Explore E

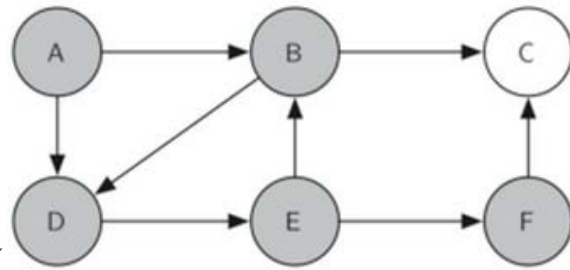


Figure 9: Explore F

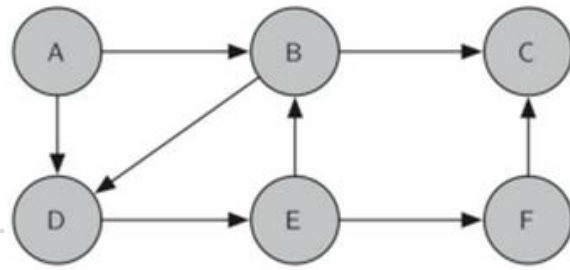


Figure 10: Finish

骑士周游问题：一个解

