华东师范大学数据科学与工程学院实验报告

课程名称：AI 基础	**年级**：2023 级	**上机实践日期**：2025 年 04 月 09 日
指导教师：杨彬	**姓名**：吕佳鸿	
上机实践名称：实验1	**学号**：10235501436	

一、实验任务

Warm-up

一个农民要带着一只狼，一只羊，一颗白菜过河。人不在的时候，狼会吃羊，羊会吃草。猎人每次只能带一样东西过河。

Q1：基于你的状态空间图，分别用BFS和DFS算法画出搜索路径

• Q2：挑选BFS或DFS算法，写出搜索到最终状态的流程

• BFS展示每一步的队列

• DFS展示每一步的递归栈

图最短路问题

给定一个 个点 条边的有向图，图中可能存在重边和自环。
所有边的长度(权重)都是 1，点的编号为 1～n。
请你求出 1号点到n 号点的最短距离，如果从1 号点无法走到n 号点，则输出-1 。

八数码问题

在一个3 × 3的网格中，1 ~ 8这8个数字和一个 x 恰好不重不漏地分布在这3 × 3的网格中。

例如:

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

在游戏过程中，可以把 x 与其上、下、左、右四个方向之一的数字交换（如果存在）。

我们的目标是通过交换，使得网格变为如下排列（称为正确排列）：

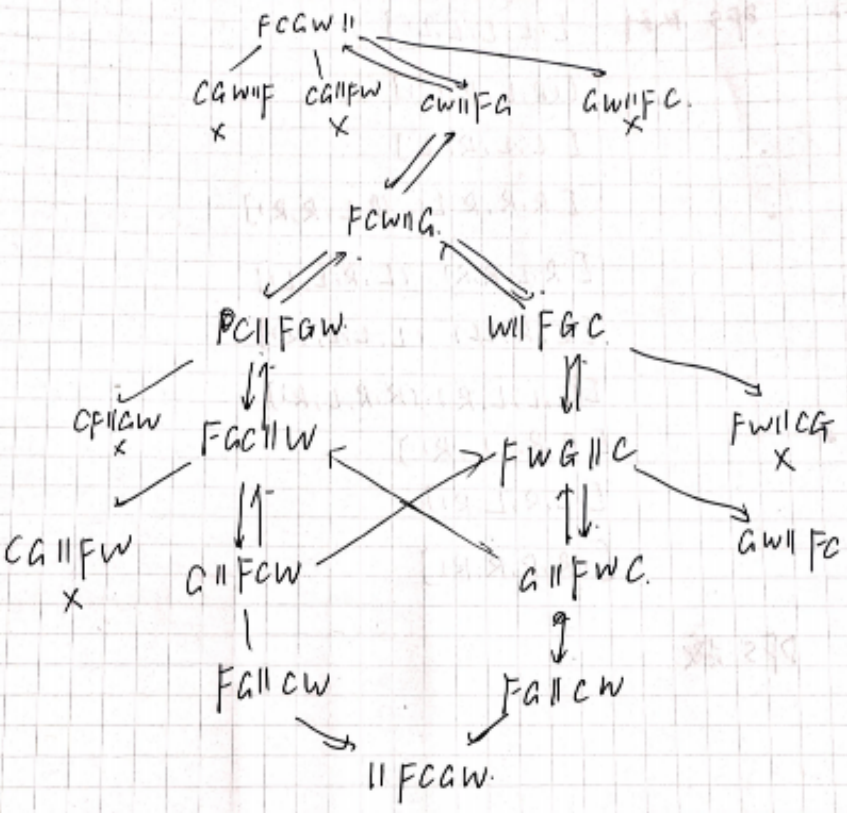
```
1 2 3
4 5 6
7 8 x
```

二、使用环境

vscode, python3.9

三、实验过程

Warm-up

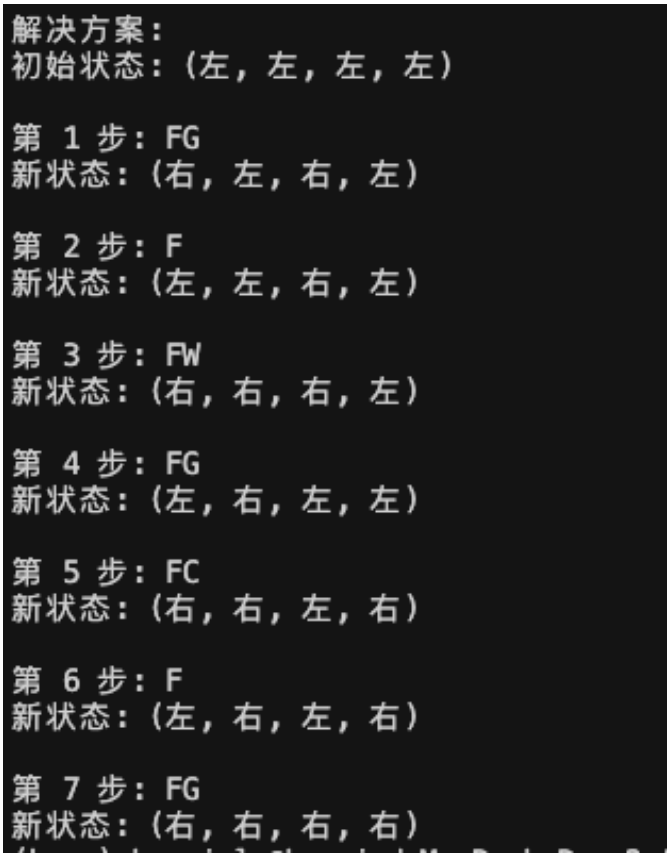


R_1 :

S_0 (L, L, L, L)	S_6 (L, L, R, R)
S_1 (R, L, R, L)	S_7 (L, L, L, R)
S_2 (L, L, R, L)	S_8 (R, R, L, R)
S_3 (R, R, R, L)	S_9 (R, L, R, R)
S_4 (L, R, R, L)	S_{10} (L, R, L, R)
S_5 (R, L, R, R)	S_{11} (R, R, R, R)

BFS: $S_0 \to S_1 \to S_2 \to S_3 \to S_5 \to S_4 \to S_7 \to S_8 \to S_{10} \to S_9$

DFS $S_0 \to S_1 \to S_2 \to S_5 \to S_7 \to S_8 \to S_{10} \to S_{11}$



图的最短路径问题(具体代码在仓库)

1_1.py: BFS实现

问题分析：

给定一个无向图，求从节点1到节点n的最短路径长度。使用邻接表存储图结构。所有边的权重为1

算法复杂度分析：

- 时间复杂度：O(V + E)，其中V是节点数，E是边数
- 空间复杂度：O(V)，用于存储距离数组和队列

算法伪代码：

```
BFS(n, edges):
    构建邻接表graph
    初始化distance数组，所有值为-1
    distance[1] = 0
    queue = [1]

    while queue不为空：
        node = queue出队
        for neighbor in graph[node]:
            if distance[neighbor] == -1:
                distance[neighbor] = distance[node] + 1
```

```
                queue入队neighbor

    return distance[n]
```

1_2.py: Dijkstra实现（无优先队列）

问题分析：

给定一个带权有向图，求从节点1到节点n的最短路径长度。使用邻接矩阵存储图结构。边权可能为任意正数

算法复杂度分析：

- 时间复杂度：$O(V^2)$，其中V是节点数
- 空间复杂度：$O(V^2)$，用于存储邻接矩阵

算法伪代码：

```
Dijkstra(n, edges):
    构建邻接矩阵grid
    初始化dist数组，所有值为无穷大
    dist[1] = 0
    visited = [False] * (n+1)

    for i from 1 to n:
        找到未访问的最小dist节点u
        visited[u] = True
        for v from 1 to n:
            if not visited[v] and dist[u] + grid[u][v] < dist[v]:
                dist[v] = dist[u] + grid[u][v]

    return dist[n]
```

1_3.py: Dijkstra实现（优先队列）

问题分析：

与1_2.py相同的问题，但使用优先队列优化。使用邻接表存储图结构

算法复杂度分析：

- 时间复杂度：$O(E \log V)$，其中V是节点数，E是边数
- 空间复杂度：$O(V + E)$，用于存储邻接表和优先队列

算法伪代码：

```
Dijkstra_Heap(n, edges):
    构建邻接表graph
    初始化dist数组，所有值为无穷大
    dist[1] = 0
```

```
    pq = [(0, 1)]

    while pq不为空:
        curr_dist, curr_node = pq出队
        if curr_dist > dist[curr_node]:
            continue
        for neighbor, weight in graph[curr_node]:
            new_dist = curr_dist + weight
            if new_dist < dist[neighbor]:
                dist[neighbor] = new_dist
                pq入队(new_dist, neighbor)

    return dist[n]
```

八数码问题（具体代码在仓库）

2_1.py: DFS实现

问题分析：

- 求解八数码问题的最少移动步数。使用深度优先搜索。需要处理状态重复访问问题

算法伪代码：

```
DFS(state):
    if state == goal:
        return steps
    visited = set()
    queue = [(state, 0)]

    while queue:
        state, steps = queue.popleft()
        if state in visited:
            continue
        visited.add(state)
        for new_state in generate_new_states(state):
            if new_state == goal:
                return steps + 1
            queue.append((new_state, steps + 1))
    return -1
```

2_2.py: BFS实现

问题分析：

- 与2_1.py相同的问题，但使用广度优先搜索。保证找到最短路径

算法伪代码：

```
BFS(start, goal):
    if start == goal:
        return 0
    queue = [(start, 0)]
    visited = set()

    while queue:
        state, steps = queue.popleft()
        for new_state in next_state(state):
            if new_state == goal:
                return steps + 1
            if new_state not in visited:
                visited.add(new_state)
                queue.append((new_state, steps + 1))
    return -1
```

2_3.py: Dijkstra实现

问题分析：

- 与2_2.py相同的问题，但使用Dijkstra算法

算法伪代码：

```
Dijkstra(start, goal):
    heap = [(0, start)]
    dist = {start: 0}

    while heap:
        f, state = heap.pop()
        if state == goal:
            return f
        for new_state in next_state(state):
            new_f = f + 1
            if new_state not in dist or dist[new_state] > new_f:
                dist[new_state] = new_f
                heap.push((new_f, new_state))
    return -1
```

2_4.py: A*实现

问题分析：

- 使用A*算法求解八数码问题
- 使用欧几里得距离作为启发式函数
- 增加了可解性判断
- 输出移动序列

算法伪代码：

```
A_Star(start, goal):
    if not is_solvable(start):
        return "unsolvable"
    heap = [(distance(start), 0, start, "")]
    visited = {start: 0}

    while heap:
        f, g, state, path = heap.pop()
        if state == goal:
            return path
        for new_state, move in next_state(state):
            new_g = g + 1
            new_f = new_g + distance(new_state)
            if new_state not in visited or visited[new_state] > new_g:
                visited[new_state] = new_g
                heap.push((new_f, new_g, new_state, path + move))
    return "unsolvable"
```

四、总结

这个lab深化了对BFS，DFS，Dijkstra的了解 并学习了A*算法