3.1 DFS

```python
import matplotlib.pyplot as plt
import sys
from matplotlib.animation import FuncAnimation
import numpy as np

# 回溯路径
def find_the_path(lst, now):
    total_path = [now]
    while now in lst:
        now = lst[now]
        total_path.append(now)
    return total_path[::-1]

def iddfs(maze):
    rows = len(maze)
    cols = len(maze[0]) if rows > 0 else 0
    start = (0, 0)
    end = (rows - 1, cols - 1)
    max_depth = 0

    iterations = []
    final_path = None

    def dfs(now, depth, visited, lst, current_visited_order):
        if depth > max_depth:
            return False

        current_visited_order.append(now)

        if now == end:
            final_path[:] = find_the_path(lst, now)
            return True

        visited.add(now)

        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]:
            x = now[0] + dx
            y = now[1] + dy
            nxt = (x, y)
            if 0 <= x < rows and 0 <= y < cols and maze[x][y] == 0 and nxt not in visited:
                lst[nxt] = now
                if dfs(nxt, depth + 1, visited, lst, current_visited_order):
                    return True

        visited.remove(now)
        return False

    while True:
        visited = set()
        lst = {}
        current_visited_order = []
        final_path = []

        if dfs(start, 0, visited, lst, current_visited_order):
            iterations.append({
                "visited_order": current_visited_order,
                "found": True,
                "path": final_path
            })
            break

        iterations.append({
            "visited_order": current_visited_order,
            "found": False,
            "path": None
        })

        max_depth += 1

    return iterations

def visualize_maze_with_path(maze, iterations):
    # 创建图形和轴
    fig, ax = plt.subplots(figsize=(len(maze[0]), len(maze)))  # 设置图形大小
    ax.imshow(maze, cmap='Greys', interpolation='nearest')  # 使用灰度色图，并关闭插值

    # 设置坐标轴刻度
    ax.set_xticks(range(len(maze[0])))  # 设置x轴刻度
    ax.set_yticks(range(len(maze)))  # 设置y轴刻度
    ax.set_xticks([x - 0.5 for x in range(1, len(maze[0]))], minor=True)  # 设置x轴的次刻度
    ax.set_yticks([y - 0.5 for y in range(1, len(maze))], minor=True)  # 设置y轴的次刻度
    ax.grid(which="minor", color="black", linestyle='-', linewidth=2)  # 为次刻度添加网格线

    # 初始化散点图和路径线图
    scatter = ax.scatter([], [], s=10, color='blue', alpha=0.5)  # 用蓝色显示访问过的节点
    line, = ax.plot([], [], marker='o', markersize=8, color='red', linewidth=3)  # 用红色显示路径

    # 计算总帧数（访问过程+路径绘制）
    total_frames = sum(len(iter["visited_order"]) for iter in iterations)  # 计算所有访问顺序的总
    total_frames += len(iterations[-1]["path"])  # 添加最后路径的帧数

    # 更新函数，用于动画更新每一帧
    def update(frame):
        # 确定当前属于哪个阶段（访问阶段或路径阶段）
        cum_frames = 0
        current_stage = 0
        path_stage = False

        # 遍历所有的迭代，找到当前帧在哪个迭代阶段
        for i, iter in enumerate(iterations):
            if frame < cum_frames + len(iter["visited_order"]):
                current_stage = i
                break
            cum_frames += len(iter["visited_order"])

        # 判断是否进入路径阶段
        if frame >= total_frames - len(iterations[-1]["path"]):
            path_stage = True
            path_frame = frame - (total_frames - len(iterations[-1]["path"]))

        # 访问阶段
        if not path_stage:
            current_iter = iterations[current_stage]
            frames_in_stage = frame - cum_frames
            visited_x, visited_y = zip(*current_iter["visited_order"][:frames_in_stage+1])
            scatter.set_offsets(np.column_stack([visited_y, visited_x]))  # 更新散点图的位置

        # 路径阶段
        else:
            final_iter = iterations[-1]  # 获取最后的路径信息
            all_visited = set()  # 用于存储所有访问过的节点
            for iter in iterations:
                all_visited.update(iter["visited_order"])  # 合并所有访问过的节点
            visited_x, visited_y = zip(*all_visited)  # 提取所有访问节点的x、y坐标
```

```
            scatter.set_offsets(np.column_stack([visited_y, visited_x]))  # 更新散点图显示所有访

            # 更新路径，逐渐显示路径节点
            if path_frame < len(final_iter["path"]):
                path_x, path_y = zip(*final_iter["path"][:path_frame+1])  # 获取路径的前path_fra
                line.set_data(path_y, path_x)  # 更新路径线

        return scatter, line  # 返回更新后的散点图和路径线

    # 创建动画，定时调用update函数
    ani = FuncAnimation(fig, update, frames=total_frames, interval=100, blit=True, repeat=False
    plt.show()  # 展示动画

# 读取输入
input = sys.stdin.read().split()
idx = 0
n = int(input[idx])
idx += 1
m = int(input[idx])
idx += 1

maze = []
for _ in range(n):
    row = list(map(int, input[idx:idx+m]))
    maze.append(row)
    idx += m

iterations = iddfs(maze)

print(f"路径长度：{len(iterations[-1]['path']) - 1}")

total_distance = 0
final_path = iterations[-1]['path']
for j in range(len(final_path) - 1):
    if final_path[j][0] != final_path[j+1][0] and final_path[j][1] != final_path[j+1][1]:  # 斜
        total_distance += np.sqrt(2)
    else:
        total_distance += 1

print(f"实际距离（路径总代价）：{total_distance}")
```

```
# 可视化迷宫及路径
visualize_maze_with_path(maze, iterations)
```

3.2 BFS

```python
from collections import deque
import matplotlib.pyplot as plt
import sys
from matplotlib.animation import FuncAnimation
import numpy as np

# 回溯路径
def find_the_path(lst, now):
    total_path = [now]
    while now in lst:
        now = lst[now]
        total_path.append(now)
    return total_path[::-1]

def bfs(maze):
    rows = len(maze)
    cols = len(maze[0]) if rows > 0 else 0
    start = (0, 0)
    end = (rows - 1, cols - 1)

    queue = deque()
    queue.append(start)
    visited = set()
    visited.add(start)
    visited_order = []
    lst = {}

    while queue:
        now = queue.popleft()
        visited_order.append(now)
        if now == end:
            return find_the_path(lst, now), visited_order

        # 斜对角走法：除了上下左右，还可以走四个斜对角方向
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]:
            x = now[0] + dx
            y = now[1] + dy
            nxt = (x, y)

            # 确保节点在迷宫范围内并且是可行走的
            if 0 <= x < rows and 0 <= y < cols and maze[x][y] == 0:
                if nxt not in visited:
                    lst[nxt] = now
                    visited.add(nxt)
                    queue.append(nxt)

# 可视化迷宫和路径
def visualize_maze_with_path(maze, path, visited_order):
    fig, ax = plt.subplots(figsize=(len(maze[0]), len(maze))) # 设置图形大小
    ax.imshow(maze, cmap='Greys', interpolation='nearest')  # 使用灰度色图，并关闭插值

    # 设置坐标轴刻度和边框
    ax.set_xticks(range(len(maze[0])))
    ax.set_yticks(range(len(maze)))
    ax.set_xticks([x - 0.5 for x in range(1, len(maze[0]))], minor=True)
    ax.set_yticks([y - 0.5 for y in range(1, len(maze))], minor=True)
    ax.grid(which="minor", color="black", linestyle='-', linewidth=2)

    # 初始化空的散点图和线图
    scatter = ax.scatter([], [], s=10, color='blue', alpha=0.5)
    line, = ax.plot([], [], marker='o', markersize=8, color='red', linewidth=3)

    # 动画更新函数
    def update(frame):
        # 显示已访问的节点
        if frame < len(visited_order):
            visited_x, visited_y = zip(*visited_order[:frame+1])
            scatter.set_offsets(np.column_stack([visited_y, visited_x]))

        # 显示路径
        if frame >= len(visited_order):
            path_frame = frame - len(visited_order)
            if path_frame < len(path):
                path_x, path_y = zip(*path[:path_frame+1])
                line.set_data(path_y, path_x)

        return scatter, line

    # 计算总帧数（访问过程+路径绘制）
    total_frames = len(visited_order) + len(path)

    # 创建动画
    ani = FuncAnimation(fig, update, frames=total_frames, interval=1000, blit=True, repeat=False
    plt.show()

# 读取输入
input = sys.stdin.read().split()
idx = 0
n = int(input[idx])
idx += 1
m = int(input[idx])
idx += 1

maze = []
for _ in range(n):
    row = list(map(int, input[idx:idx+m]))
    maze.append(row)
    idx += m

path, visited_order = bfs(maze)

print(f"路径长度: {len(path) - 1}")

total_distance = sum(np.sqrt(2) if (path[i][0] != path[i+1][0] and path[i][1] != path[i+1][1]) e

print(f"实际距离（路径总代价）: {total_distance}")

visualize_maze_with_path(maze, path, visited_order)
```

3.3 Dijkstra

```python
import heapq
import matplotlib.pyplot as plt
import sys
from matplotlib.animation import FuncAnimation
import numpy as np

# 回溯路径
def find_the_path(lst, now):
    total_path = [now]
    while now in lst:
        now = lst[now]
        total_path.append(now)
    return total_path[::-1]

def dijkstra(maze):
    rows = len(maze)
    cols = len(maze[0])
    start = (0, 0)
    end = (rows - 1, cols - 1)

    heap = []
    heapq.heappush(heap, (0, start))

    g_score = {start: 0}
    lst = {}
    visited = set()
    visited_order = []

    while heap:
        now_g, now = heapq.heappop(heap)
        if now in visited:
            continue
        visited.add(now)
        visited_order.append(now)

        # 如果当前节点是目标节点，返回路径
        if now == end:
            return find_the_path(lst, now), visited_order

        # 斜对角走法：除了上下左右，还可以走四个斜对角方向
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]:
            x = now[0] + dx
            y = now[1] + dy
            nxt = (x, y)

            # 确保节点在迷宫范围内并且是可行走的
            if 0 <= x < rows and 0 <= y < cols and maze[x][y] == 0:
                # 如果是斜着走，代价设置为sqrt(2)，否则为1
                new_g = now_g + (1 if dx == 0 or dy == 0 else np.sqrt(2))  # 斜着走的代价为sqrt(
                if nxt not in g_score or new_g < g_score.get(nxt, float('inf')):
                    lst[nxt] = now
                    g_score[nxt] = new_g
                    heapq.heappush(heap, (new_g, nxt))


# 可视化迷宫和路径
def visualize_maze_with_path(maze, path, visited_order):
    fig, ax = plt.subplots(figsize=(len(maze[0]), len(maze))) # 设置图形大小
    ax.imshow(maze, cmap='Greys', interpolation='nearest')  # 使用灰度色图，并关闭插值

    # 设置坐标轴刻度和边框
    ax.set_xticks(range(len(maze[0])))
    ax.set_yticks(range(len(maze)))
    ax.set_xticks([x - 0.5 for x in range(1, len(maze[0]))], minor=True)
    ax.set_yticks([y - 0.5 for y in range(1, len(maze))], minor=True)
    ax.grid(which="minor", color="black", linestyle='-', linewidth=2)

    # 初始化空的散点图和线图
    scatter = ax.scatter([], [], s=10, color='blue', alpha=0.5)
    line, = ax.plot([], [], marker='o', markersize=8, color='red', linewidth=3)

    # 动画更新函数
    def update(frame):
        # 显示已访问的节点
        if frame < len(visited_order):
            visited_x, visited_y = zip(*visited_order[:frame+1])
            scatter.set_offsets(np.column_stack([visited_y, visited_x]))

        # 显示路径
        if frame >= len(visited_order):
            path_frame = frame - len(visited_order)
            if path_frame < len(path):
                path_x, path_y = zip(*path[:path_frame+1])
                line.set_data(path_y, path_x)

        return scatter, line

    # 计算总帧数（访问过程+路径绘制）
    total_frames = len(visited_order) + len(path)

    # 创建动画
    ani = FuncAnimation(fig, update, frames=total_frames, interval=1000, blit=True, repeat=False
    plt.show()


# 读取输入
input = sys.stdin.read().split()
idx = 0
n = int(input[idx])
idx += 1
m = int(input[idx])
idx += 1

maze = []
for _ in range(n):
    row = list(map(int, input[idx:idx+m]))
    maze.append(row)
    idx += m

path, visited_order = dijkstra(maze)

print(f"路径长度: {len(path) - 1}")

total_distance = sum(np.sqrt(2) if (path[i][0] != path[i+1][0] and path[i][1] != path[i+1][1])

print(f"实际距离（路径总代价）: {total_distance}")

visualize_maze_with_path(maze, path, visited_order)
```

3.4 A star

```python
import heapq
import matplotlib.pyplot as plt
import sys
from matplotlib.animation import FuncAnimation
import numpy as np

# 启发式函数
def distance(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

# 回溯路径
def find_the_path(lst, now):
    total_path = [now]
    while now in lst:
        now = lst[now]
        total_path.append(now)
    return total_path[::-1]

def A_star(maze):
    rows = len(maze)
    cols = len(maze[0])
    start = (0, 0)
    end = (rows - 1, cols - 1)

    heap = []
    heapq.heappush(heap, (0, 0, start))  # 使用堆来存储候选节点，(f_score, g_score, 当前节点)

    g_score = {start: 0}  # g_score记录到当前节点的距离
    f_score = {start: distance(start, end)}  # f_score记录估计总的距离
    lst = {}  # 用来回溯路径
    visited = set()  # 记录访问过的节点
    visited_order = []  # 记录访问顺序

    while heap:
        now_f, now_g, now = heapq.heappop(heap)  # 取出f_score最小的节点
        visited.add(now)
        visited_order.append(now)

        if now == end:  # 如果当前节点是目标节点，回溯路径
            return find_the_path(lst, now), visited_order
        # 斜对角走法：除了上下左右，还可以走四个斜对角方向
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]:
            x = now[0] + dx
            y = now[1] + dy
            nxt = (x, y)

            # 确保节点在迷宫范围内并且是可行走的
            if 0 <= x < rows and 0 <= y < cols and maze[x][y] == 0:
                # 如果是斜着走，代价设置为sqrt(2)，否则为1
                new_g = now_g + (1 if dx == 0 or dy == 0 else np.sqrt(2))  # 斜着走的代价为sqrt(
                if nxt not in g_score or new_g < g_score.get(nxt, float('inf')):
                    lst[nxt] = now
                    g_score[nxt] = new_g
                    f_score[nxt] = new_g + distance(nxt, end)
                    heapq.heappush(heap, (f_score[nxt], new_g, nxt))

# 可视化迷宫和路径
def visualize_maze_with_path(maze, path, visited_order):
    fig, ax = plt.subplots(figsize=(len(maze[0]), len(maze)))  # 设置图形大小
    ax.imshow(maze, cmap='Greys', interpolation='nearest')  # 使用灰度色图，并关闭插值

    # 设置坐标轴刻度和边框
    ax.set_xticks(range(len(maze[0])))
    ax.set_yticks(range(len(maze)))
    ax.set_xticks([x - 0.5 for x in range(1, len(maze[0]))], minor=True)
    ax.set_yticks([y - 0.5 for y in range(1, len(maze))], minor=True)
    ax.grid(which="minor", color="black", linestyle='-', linewidth=2)

    # 初始化空的散点图和路径线图
    scatter = ax.scatter([], [], s=10, color='blue', alpha=0.5)
    line, = ax.plot([], [], marker='o', markersize=8, color='red', linewidth=3)

    # 动画更新函数
    def update(frame):
        # 显示已访问的节点，按顺序依次展示
        if frame < len(visited_order):
            visited_x, visited_y = zip(*visited_order[:frame+1])
            scatter.set_offsets(np.column_stack([visited_y, visited_x]))

        # 显示路径，路径会在已访问节点之后绘制
        if frame >= len(visited_order):
            path_frame = frame - len(visited_order)
            if path_frame < len(path):
                path_x, path_y = zip(*path[:path_frame+1])
                line.set_data(path_y, path_x)

        return scatter, line

    # 计算总帧数（访问过程+路径绘制）
    total_frames = len(visited_order) + len(path)

    # 创建动画
    ani = FuncAnimation(fig, update, frames=total_frames, interval=1000, blit=True, repeat=Fals
    plt.show()

# 读取输入
input = sys.stdin.read().split()
idx = 0
n = int(input[idx])
idx += 1
m = int(input[idx])
idx += 1

maze = []
for _ in range(n):
    row = list(map(int, input[idx:idx+m]))
    maze.append(row)
    idx += m

path, visited_order = A_star(maze)
print(f"路径长度：{len(path) - 1}")

total_distance = sum(np.sqrt(2) if (path[i][0] != path[i+1][0] and path[i][1] != path[i+1][1])

print(f"实际距离（路径总代价）：{total_distance}")
visualize_maze_with_path(maze, path, visited_order)
```