地图导航中的最短距离问题
伪代码:
输入：有向图
输出：最短距离

```
INF ← 无穷大       // 定义一个表示无穷大的常量
graph       // 使用邻接矩阵来表示有向图
算法: floyd_warshall(graph)     // 计算所有顶点对之间的最短路径 floyd_warshall
1.      n ← 图中顶点数
2.      dist ← 0 矩阵
3.      for i ← 0 to n do
4.          for j ← 0 to n
5.              dist[i][j] ← graph[i][j]
6.      // 计算最短路径长度
7.      for k ← 0 to n do
8.          for i ← 0 to n do
9.              for j ← 0 to n do
10.                 if dist[i][k] != INF && dist[k][j] != INF:
11.                     dist[i][j] ← min(dist[i][j], dist[i][k] + dist[k][j])
12.     return dist
```

物流配送中的背包问题思想
输入：物品列表 items(weights, values)，包括物品价值和重量，装载限重
输出：能够装载的最大价值
算法：knapsack(weights, values, capacity)

```
1. capacity ← 背包容量
2. n ← len(items)
3. dp ← 0 矩阵
4. for i ← 1 to n do
5.     for j← 1 to capacity do
6.         if j >= items[i-1][0]
7.             dp[i][j] ← max(dp[i-1][j], dp[i-1][j-items[i-1][0]] + items[i-1][1])
8.         else
9.             dp[i][j] ← dp[i-1][j]
10. return dp[n][capacity]
```

股票买卖问题中的动态规划思想
输入：股票价格列表
输出：股票买卖的最大利润
算法：max_profit(prices)

```
1. prices ← 股票价格
2. n ← length(prices)
3. if n < 2
```

```
4.      return 0
5. dp ← array(n, 2) #数组初始化，包括 dp[0][0], dp[0][1] = 0, -prices[0]
6. for i ← 1 to n do
7.      dp[i][0] ← max(dp[i-1][0], dp[i-1][1] + prices[i])
8.      dp[i][1] ← max(dp[i-1][1], -prices[i])
9. return dp[n-1][0]
```