# 使用Python解决最长路径问题

在Python中，我们可以使用多种方法来解决有向无环图(DAG)的最长路径问题。以下是几种实现方式：

## 方法1：使用拓扑排序（推荐用于DAG）

from collections import defaultdict, deque  
  
def longest\_path(graph, weights, start, end):  
 # 拓扑排序  
 in\_degree = defaultdict(int)  
 for u in graph:  
 for v in graph[u]:  
 in\_degree[v] += 1  
   
 queue = deque([u for u in graph if in\_degree[u] == 0])  
 topo\_order = []  
   
 while queue:  
 u = queue.popleft()  
 topo\_order.append(u)  
 for v in graph[u]:  
 in\_degree[v] -= 1  
 if in\_degree[v] == 0:  
 queue.append(v)  
   
 # 初始化距离  
 dist = {node: -float('inf') for node in graph}  
 dist[start] = 0  
 prev = {node: None for node in graph}  
   
 # 按照拓扑顺序计算最长路径  
 for u in topo\_order:  
 for v in graph[u]:  
 if dist[v] < dist[u] + weights[(u, v)]:  
 dist[v] = dist[u] + weights[(u, v)]  
 prev[v] = u  
   
 # 重建路径  
 path = []  
 node = end  
 while node is not None:  
 path.append(node)  
 node = prev[node]  
 path.reverse()  
   
 return dist[end], path  
  
# 构建图  
graph = {  
 'A': ['B', 'C'],  
 'B': ['E'],  
 'C': ['D', 'F'],  
 'D': ['E'],  
 'E': ['G'],  
 'F': ['G', 'H'],  
 'G': ['I'],  
 'H': ['I'],  
 'I': []  
}  
  
weights = {  
 ('A', 'B'): 9,  
 ('A', 'C'): 9,  
 ('B', 'E'): 5,  
 ('C', 'D'): 3,  
 ('C', 'F'): 3,  
 ('D', 'E'): 6,  
 ('E', 'G'): 4,  
 ('F', 'G'): 7,  
 ('F', 'H'): 7,  
 ('G', 'I'): 5,  
 ('H', 'I'): 6  
}  
  
distance, path = longest\_path(graph, weights, 'A', 'I')  
print(f"最长路径长度: {distance}")  
print(f"路径: {' -> '.join(path)}")

## 方法2：使用递归+记忆化（DFS）

from functools import lru\_cache  
  
def longest\_path\_dfs(graph, weights, start, end):  
 @lru\_cache(maxsize=None)  
 def dfs(node):  
 if node == end:  
 return (0, [end])  
   
 max\_dist = -float('inf')  
 best\_path = []  
   
 for neighbor in graph[node]:  
 dist, path = dfs(neighbor)  
 current\_dist = dist + weights[(node, neighbor)]  
 if current\_dist > max\_dist:  
 max\_dist = current\_dist  
 best\_path = [node] + path  
   
 return (max\_dist, best\_path) if max\_dist != -float('inf') else (-float('inf'), [])  
   
 distance, path = dfs(start)  
 return distance, path  
  
distance, path = longest\_path\_dfs(graph, weights, 'A', 'I')  
print(f"最长路径长度: {distance}")  
print(f"路径: {' -> '.join(path)}")

## 方法3：使用networkx库（最简单）

import networkx as nx  
  
def longest\_path\_networkx(edges, start, end):  
 G = nx.DiGraph()  
 G.add\_weighted\_edges\_from(edges)  
   
 try:  
 path = nx.dag\_longest\_path(G, weight='weight')  
 length = nx.dag\_longest\_path\_length(G, weight='weight')  
 return length, path  
 except nx.NetworkXUnfeasible:  
 return None, None  
  
edges = [  
 ('A', 'B', 9),  
 ('A', 'C', 9),  
 ('B', 'E', 5),  
 ('C', 'D', 3),  
 ('C', 'F', 3),  
 ('D', 'E', 6),  
 ('E', 'G', 4),  
 ('F', 'G', 7),  
 ('F', 'H', 7),  
 ('G', 'I', 5),  
 ('H', 'I', 6)  
]  
  
distance, path = longest\_path\_networkx(edges, 'A', 'I')  
print(f"最长路径长度: {distance}")  
print(f"路径: {' -> '.join(path)}")

## 输出结果

以上三种方法都会输出：

最长路径长度: 27  
路径: A -> C -> D -> E -> G -> I

## 注意事项

1. 这些方法只适用于有向无环图(DAG)，如果图中存在环，需要先检测并处理
2. 拓扑排序方法的时间复杂度是O(V+E)，适合大多数情况
3. networkx库提供了最简洁的实现，但需要安装额外的包
4. 如果图中有负权边，最长路径问题会变得更复杂

您可以根据自己的需求选择最适合的方法。对于您提供的具体问题，所有方法都能正确计算出最长路径为A→C→D→E→G→I，总长度为27。