### Algorithm: Cycle Detection in Directed Graph (Kahn’s Algorithm)

Princess Elara needed to detect cycles in the supply routes of her navy to prevent logistical failures. She used Kahn’s Algorithm to detect cycles in the directed graph of supply routes.

#### Initialize Data Structures:

* Princess Elara used a magical net (queue) to keep track of nodes with no incoming edges.
* She also used a compass (array) to keep track of in-degrees.

#### Detect Cycles:

* She removed nodes with no incoming edges and checked if any nodes were left unvisited.

#### Implementation:

| **from** **collections** **import** deque  **def** detect\_cycle(supply\_routes: Dict[int, List[int]], V: int) -> bool:  in\_degree = [0] \* V  **for** v **in** supply\_routes:  **for** neighbor **in** supply\_routes[v]:  in\_degree[neighbor] += 1  queue = deque([v **for** v **in** range(V) **if** in\_degree[v] == 0])  visited = 0  **while** queue:  v = queue.popleft()  visited += 1  **for** neighbor **in** supply\_routes[v]:  in\_degree[neighbor] -= 1  **if** in\_degree[neighbor] == 0:  queue.append(neighbor)  **return** visited != V  *# Example usage:*  supply\_routes = {  0: [1, 2],  1: [2],  2: [0, 3],  3: [3]  }  V = 4  print(detect\_cycle(supply\_routes, V)) *# Output: True (cycle detected)* |
| --- |

#### Explanation:

Initialize:

* in\_degree: Array to track in-degrees of nodes.
* queue: A magical net to keep track of nodes with no incoming edges.

Detect Cycles:

### Princess Elara removed nodes with no incoming edges and checked if any nodes were left unvisited.