1. Why do we use a link for the shortest path computation only if it exists in our database in both directions? What would happen if we used a directed link AB when the link BA does not exist?
    - We only use bidirectional links because they can be used for bidirectional communication, acknowledgements, and to ensure paths are valid in both directions, even if they only look valid in one direction
    - If we used unidirectional links, we wouldn't be able to acknowledge if something was transmitted correctly, routing loops or inconsistencies might form as routers might believe they can reach other routers even if it might not be the case, and paths might look valid in one direction but fail in a real network.
2. Does your routing algorithm produce symmetric routes (that follow the same path from X to Y in reverse when going from Y to X)? Why or why not?
    - My algorithm always produces symmetric routes when the path between two nodes is the shortest path. When there are multiple equal-cost paths, Dijkstra's algorithm may choose a different path when going the other way.
3. What would happen if a node advertised itself as having neighbors, but never forwarded packets? How might you modify your implementation to deal with this case?
    - If a node advertised itself as having neighbors but never forwarded packets, it would become a routing black hole. All of the nodes would include the faulty node in their topology, but once the packets reach that node, they are lost forever.
    - If this happened in my case, I might implement a timeout-based detection, where absences of expected replies within reasonable timeframes might trigger suspicion about intermediate nodes, as well as E2E acknowledgement to track whether packets have successfully traversed their expected paths.
4. What happens if link-state packets are lost or corrupted?
    - If packets are lost or corrupted, different nodes might develop inconsistent views of the topology, leading to suboptimal routing or communication failure. However, flooding creates redundant transmission paths so LSAs reach destinations even if some copies are lost, sequence numbers allow nodes to detect missed updates and request retransmission, and periodic LSA regeneration automatically corrects any accumulated inconsistencies over time.
5. What would happen if a node alternated between advertising and withdrawing a neighbor, every few milliseconds? How might you modify your implementation to deal with this case?
    - If the node created a link flap, the entire network would be destabilised, triggering constant LSA floods and Dijkstra recalculations on every node. Routing tables would constantly fluctuate, and packets would follow different paths, consuming a large amount of CPU
    - The solution is implementing rate-limiting mechanisms that suppress LSA generation for rapidly changing neighbor relationships, using exponential backoff timers to prevent a single unstable node from disrupting the network while still allowing legitimate topology changes to propagate normally.