Q4

I. Create an unsorted set called "friends" and add the values "liz", "tom", "harry":

```

ZADD friends 1 liz 2 tom 3 harry

```

II. Show all the values in the set "friends":

```

ZRANGE friends 0 -1

```

III. Show if the value "liz" exists in the set "friends":

```

ZSCORE friends liz

```

IV. Show if the value "john" exists in the set "friends":

```

ZSCORE friends john

```

V. Show how many elements are in the set "friends":

```

ZCARD friends

```

VI. Create another set called "old friends" and add the values "liz", "john", "scarlette":

```

ZADD old\_friends 1 liz 2 john 3 scarlette

```

VII. See which friends are both in the "old friends" set AND in the "friends" set:

```

ZINTERSTORE common\_friends 2 friends old\_friends

ZRANGE common\_friends 0 -1

```

VIII. Move "john" from the "old friends" set to the "friends" set:

```

ZREM old\_friends john

ZADD friends 2 john

```

IX. Show the content of "old friends":

```

ZRANGE old\_friends 0 -1

```

X. Show the values in both "friends" and "old friends" together:

```

ZUNIONSTORE all\_friends 2 friends old\_friends

ZRANGE all\_friends 0 -1

```

These commands should perform the operations you described in Redis.

Q3

To compute the Dijkstra Single-Source algorithm in Neo4j, you can use the following Cypher query:

```cypher

MATCH (start:Node {name: 'A'}), (end:Node)

WITH start, end

OPTIONAL MATCH (start)-[r:CONNECTED\_TO]->(end)

WITH start, end, coalesce(r.cost, 1000000) AS cost

WITH start, end, collect(end) AS nodes, collect(cost) AS costs

WITH start, nodes, costs, [node in nodes | toString(node.id)] AS nodeIds

WITH start, nodes, costs, nodeIds, apoc.coll.zip(nodeIds, costs) AS zipped

WITH start, nodes, costs, nodeIds, zipped, apoc.coll.toMap(zipped) AS costMap

WITH start, nodes, costs, nodeIds, costMap, algo.dijkstra(start, nodes, 'cost', {costProperty: costMap}) AS pathInfo

UNWIND pathInfo AS path

WITH path.startNode AS sourceNode, path.endNode AS targetNode, path.totalCost AS totalCost, nodes, path.nodeIds AS nodeIds, path.costs AS costs, path.path AS path

RETURN id(sourceNode) AS index, sourceNode.name AS sourceNode, targetNode.name AS targetNode, totalCost, nodeIds, costs, [node in path | node.name] AS path

```

This query assumes you have nodes labeled as `Node` and connected with relationships labeled as `CONNECTED\_TO`, where each relationship has a property `cost`. Replace `'A'` with your actual source node name if it's different.

Make sure you have the APOC library installed in your Neo4j instance to use the `apoc.coll` functions.

This query will compute the shortest paths from node 'A' to all reachable nodes and return the index, source node, target node, total cost, node IDs, costs, and the path itself.

Q2

Sure, here's the Cypher query to perform the tasks A, B, and C:

A. Create eight "PRODUCT" nodes and relate the products in terms of "SUPPLIES" relationship:

```cypher

CREATE (P1:PRODUCT {name: 'P1'}),

(P2:PRODUCT {name: 'P2'}),

(P3:PRODUCT {name: 'P3'}),

(P4:PRODUCT {name: 'P4'}),

(P5:PRODUCT {name: 'P5'}),

(P6:PRODUCT {name: 'P6'}),

(P7:PRODUCT {name: 'P7'}),

(P8:PRODUCT {name: 'P8'})

CREATE (P1)-[:SUPPLIES]->(P2),

(P2)-[:SUPPLIES]->(P3),

(P3)-[:SUPPLIES]->(P6),

(P3)-[:SUPPLIES]->(P5),

(P3)-[:SUPPLIES]->(P4),

(P2)-[:SUPPLIES]->(P8),

(P8)-[:SUPPLIES]->(P7),

(P1)-[:SUPPLIES]->(P4)

```

B. Add a property for relationship "SUPPLIER\_NAME" where Product name is P2:

```cypher

MATCH (p1:PRODUCT)-[s:SUPPLIES]->(p2:PRODUCT {name: 'P2'})

SET s.SUPPLIER\_NAME = p1.name

```

C. Delete a "SUPPLIES" relationship between product P3 and P5:

```cypher

MATCH (p3:PRODUCT {name: 'P3'})-[s:SUPPLIES]->(p5:PRODUCT {name: 'P5'})

DELETE s

```

This script will create eight PRODUCT nodes and define the relationships between them as specified in task A. Then, it adds a property "SUPPLIER\_NAME" to the relationship between P1 and P2 as specified in task B. Finally, it deletes the SUPPLIES relationship between P3 and P5 as specified in task C.

Q1