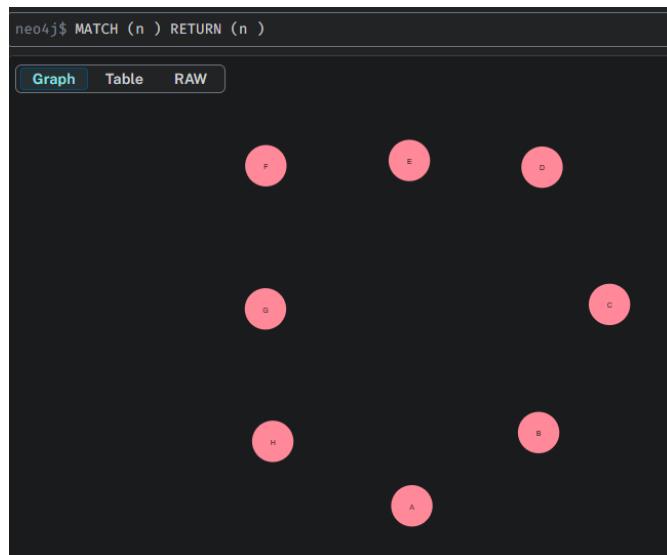


SOLUCIÓN TAREA 1-4

1. Creamos los nodos

```
neo4j$ CREATE (p8:Point {name: 'H'})  
✓ Created 1 node, set 1 property, added 1 label  
  
neo4j$ CREATE (p7:Point {name: 'G'})  
✓ Created 1 node, set 1 property, added 1 label  
  
neo4j$ CREATE (p6:Point {name: 'F'})  
✓ Created 1 node, set 1 property, added 1 label  
  
neo4j$ CREATE (p5:Point {name: 'E'})  
✓ Created 1 node, set 1 property, added 1 label  
  
neo4j$ CREATE (p4:Point {name: 'D'})  
✓ Created 1 node, set 1 property, added 1 label  
  
neo4j$ CREATE (p3:Point {name: 'C'})  
✓ Created 1 node, set 1 property, added 1 label  
  
neo4j$ CREATE (p2:Point {name: 'B'})  
✓ Created 1 node, set 1 property, added 1 label  
  
neo4j$ CREATE (p1:Point {name: 'A'})  
✓ Created 1 node, set 1 property, added 1 label
```



2. Creamos las conexiones entre los nodos

```
neo4j$ MATCH (x:Point), (y:Point) WHERE x.name = 'H' AND y.name = 'D' CREATE (x )-[:CONNECTED {distance: 14}]->(y )
```

✓ Created 1 relationship, set 1 property
➤ 03N90: Cartesian product

```
neo4j$ MATCH (x:Point), (y:Point) WHERE x.name = 'H' AND y.name = 'G' CREATE (x )-[:CONNECTED {distance: 3}]->(y )
```

✓ Created 1 relationship, set 1 property
➤ 03N90: Cartesian product

```
neo4j$ MATCH (x:Point), (y:Point) WHERE x.name = 'H' AND y.name = 'B' CREATE (x )-[:CONNECTED {distance: 6}]->(y )
```

✓ Created 1 relationship, set 1 property
➤ 03N90: Cartesian product

```
neo4j$ MATCH (x:Point), (y:Point) WHERE x.name = 'H' AND y.name = 'F' CREATE (x )-[:CONNECTED {distance: 9}]->(y )
```

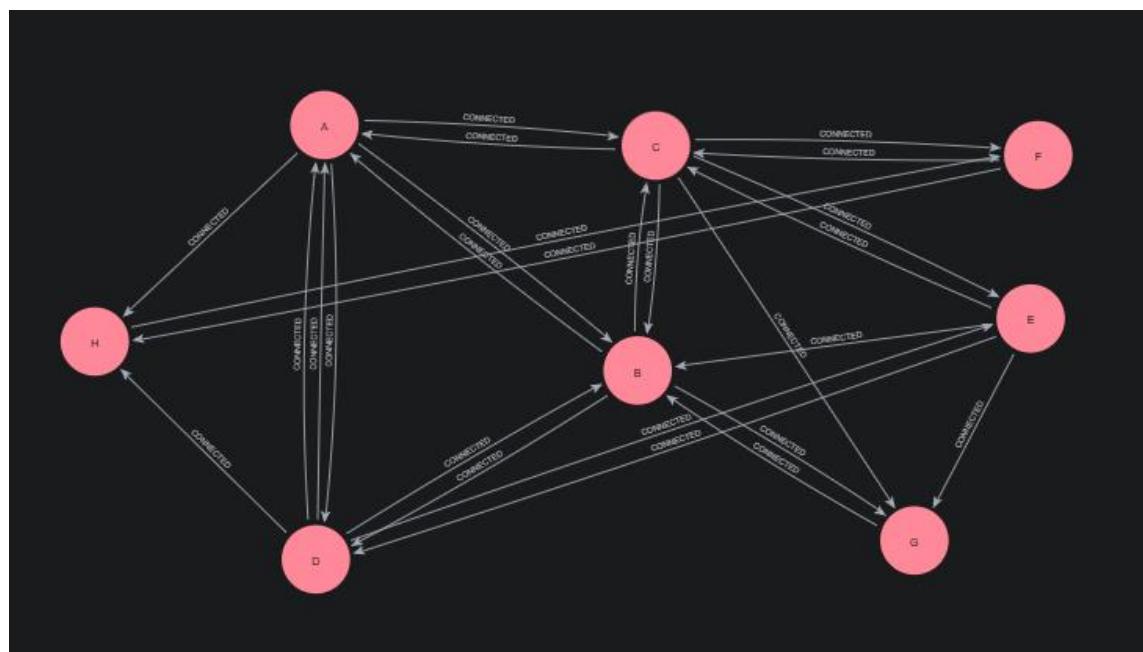
✓ Created 1 relationship, set 1 property
➤ 03N90: Cartesian product

```
neo4j$ MATCH (x:Point), (y:Point) WHERE x.name = 'H' AND y.name = 'A' CREATE (x )-[:CONNECTED {distance: 10}]->(y )
```

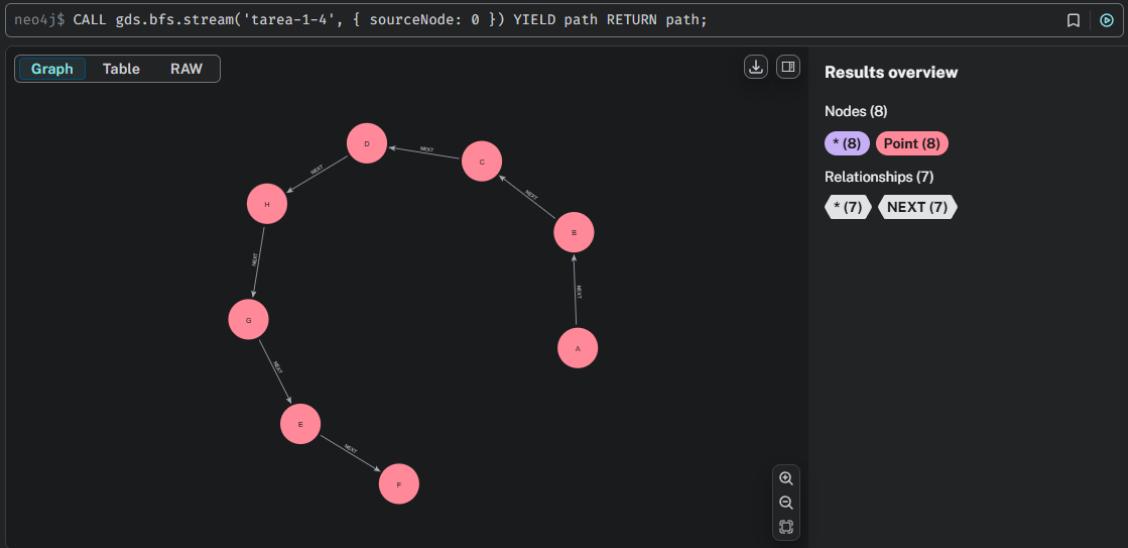
✓ Created 1 relationship, set 1 property
➤ 03N90: Cartesian product

```
neo4j$ MATCH (x:Point), (y:Point) WHERE x.name = 'G' AND y.name = 'E' CREATE (x )-[:CONNECTED {distance: 15}]->(y )
```

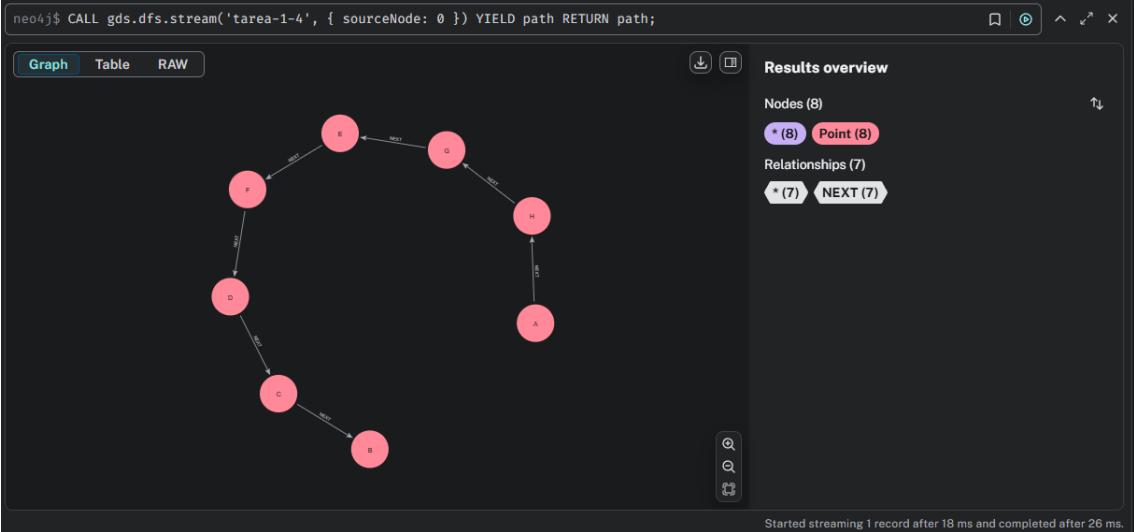
✓ Created 1 relationship, set 1 property
➤ 03N90: Cartesian product



3. Recorremos el grafo en anchura



4. Recorremos el grado en profundidad



5. Hayamos el camino mínimo entre H y G

```
1 MATCH (start:Point {name: 'H'}), (end:Point {name: 'G'})  
2 CALL gds.shortestPath.dijkstra.stream('tarea-1-4', {  
3   sourceNode: start,  
4   targetNode: end,  
5   relationshipWeightProperty: 'distance'  
6 })  
7 YIELD path  
8 RETURN path
```

Graph Table RAW

Results overview

Nodes (2)

- * (2) Point (2)

Relationships (1)

- * (1) PATH_0 (1)

The screenshot shows a Neo4j browser interface. At the top, there is a code editor with a query using the GDS library to find the shortest path between nodes H and G. Below the code are three tabs: Graph, Table, and RAW, with Graph selected. To the right of the tabs is a results overview section. This section includes a summary of nodes (2 total, including 1 Point type) and relationships (1 total, including 1 PATH_0 type). Below this summary is a graph visualization. It features two pink circular nodes labeled 'H' and 'G'. A single directed edge, also pink, connects node H at the bottom to node G at the top. The edge is labeled 'PATH_0'.