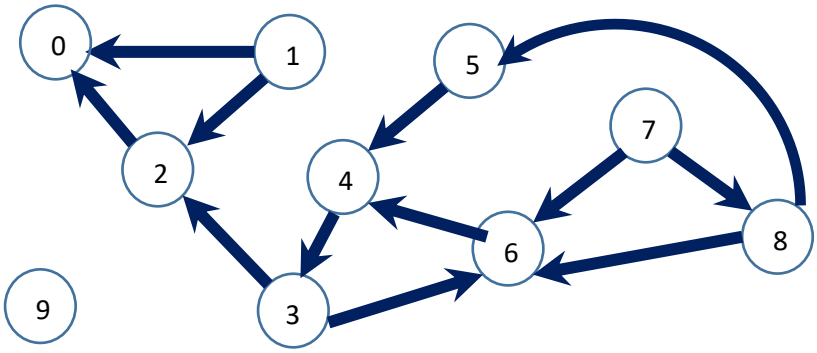
Activity 2

Graphs in Neo4J: Simple queries

Exercise 1: We will express queries over graph databases using Cypher, the high-level query language for Neo4J. To be able to easily check the correctness of the results, we will start with a small graph representing a subset of the web structure, as shown below:



The **minigraphweb database** contains this information and will be used in this activity. Check your answers.

Use Case 1: Vertex Degree

For each vertex compute its in-degree and out-degree. The result set is a list of vertices and both values. Those vertices that do not have outgoing edges and/or incoming edges, must not appear in the answer. (Note that the result set is not a graph)

Use Case 2: Vertex Degree variation

For each vertex calculate its in-degree and out-degree. The result set is a list **of all vertices in the graph**, together with the two values above, for each vertex. Those vertices that do not have outgoing edges and/or incoming edges, **must appear** in the answer with value "0".

Use Case 3: Calculating a maximum value

Find the maximum vertex in-degree. (the result set is not a graph)

Use Case 4: Find influential nodes

Find the subgraph which contains nodes whose in-degree is maximal in the graph (you should obtain **only one node**). Do the same for the out-degree (you should obtain **four nodes**).

Use Case 5: Distance between nodes

For each pair of vertices, calculate the distance, i.e. the shortest simple path between them (without repeated edges in the path). Do not show the distance between two disconnected nodes (infinite distance). Exclude paths when source and target are the same node.

Use Case 6: Distance between nodes using Cypher function

Solve the query in Use Case 5, but using the **shortestPath** built-in Cypher function**.** It has one parameter that represents a pattern path and returns the shortest path that matches this pattern. If there exists more than one shortest path, it returns any of them. For example, shortestPath((n1) –[*1..2] -> (n2)) returns a shortest path that connects n1 and n2, with distance 1 or 2.

Use Case 7: Diameter

Compute the diameter of the graph, i.e. the longest distance between two nodes in the graph (excluding disconnected pairs of nodes).

Use Case 8: webgraph3.

Repeat use cases 1 to 7 using the webgraph3 database, which represents the same information as the corresponding relational database in Activity 1.

Use Case 9: Paths

Compute all the 1, 2, 3, and n-hops in the graph, and compare against the results obtained using PostgreSQL in Activity 1. Note: start with “limit X”, increasing “X” to prevent that the algorithm runs indefinitely.