USE-CASE 2

Sam who works in the financial & Operation services department need to sort routing problems faced in carpooling. To demonstrate the problem, we use a simplified (not-so-real) map of few German cities with in [North Rhine-Westphalia](https://en.wikipedia.org/wiki/North_Rhine-Westphalia) state. The cities linked with roads.

CREATE ( COLOGNE:PLACE { name: "COLOGNE"})

CREATE (DÜSSELDORF:PLACE { name: "DÜSSELDORF"})

CREATE (DORTMUND:PLACE { name: "DORTMUND"})

CREATE (ESSEN:PLACE { name: "ESSEN"})

CREATE (DUISBURG:PLACE { name: "DUISBURG"})

CREATE (BOCHUM:PLACE { name: "BOCHUM"})

CREATE (WUPPERTAL:PLACE { name: "WUPPERTAL"})

CREATE (BIELEFELD:PLACE { name: "BIELEFELD"})

CREATE (BONN:PLACE { name: "BONN"})

CREATE (MÜNSTER:PLACE { name: "MÜNSTER"})

CREATE (GELSENKIRCHEN:PLACE { name: "GELSENKIRCHEN"})

CREATE ( COLOGNE)-[:LINK {name:'A-1',DistanceKM:320}]-> (DÜSSELDORF)

CREATE (DÜSSELDORF)-[:LINK {name:'A-2',DistanceKM:80}]->(DORTMUND)

CREATE (DORTMUND)-[:LINK {name:'A-3',DistanceKM:240}]-> (ESSEN)

CREATE (BOCHUM)-[:LINK {name:'A-4',DistanceKM:170}]-> (ESSEN)

CREATE (DUISBURG)-[:LINK {name:'A-5',DistanceKM:90}]-> (ESSEN)

CREATE (WUPPERTAL)-[:LINK {name:'A-6',DistanceKM:420}]-> (ESSEN)

CREATE (DÜSSELDORF)-[:LINK {name:'A-7',DistanceKM:350}]-> (BOCHUM)

CREATE (BOCHUM)<-[:LINK {name:'A-8',DistanceKM:50}]- (WUPPERTAL)

CREATE (BIELEFELD)-[:LINK {name:'A-9',DistanceKM:590}]-> (WUPPERTAL)

CREATE (BIELEFELD)-[:LINK {name:'A-10',DistanceKM:870}]-> (ESSEN)

CREATE (BIELEFELD)<-[:LINK {name:'A-11',DistanceKM:220}]- (BONN)

CREATE (GELSENKIRCHEN)-[:LINK {name:'A-12',DistanceKM:100}]->(BONN)

CREATE ( COLOGNE)-[:LINK {name:'A-13',DistanceKM:120}]->(GELSENKIRCHEN)

CREATE (GELSENKIRCHEN)-[:LINK {name:'A-14',DistanceKM:280}]->(MÜNSTER)

CREATE (MÜNSTER)-[:LINK {name:'A-15',DistanceKM:280}]->(DUISBURG)

CREATE (WUPPERTAL)<-[:LINK {name:'A-16',DistanceKM:390}]- ( COLOGNE)

Graphical user interface, text, application

Description automatically generated

Once the queries are run successfully, we get the complete map as a graph.

Bubble chart

Description automatically generated with low confidence

First we will find the shortest path between two cities i.e COLOGNE &ESSEN .Cypher as a built-in shortestpath() function.

MATCH (from:PLACE { name:"COLOGNE" }),

  (to:PLACE { name: "ESSEN"}),

  path = shortestpath((from)-[:LINK\*]-(to))

RETURN path

Diagram, bubble chart

Description automatically generated

It returns the path “COLOGNE-> WUPPERTAL->ESSEN” with a total distance 810 km.

Secondly we can use REDUCE to calculate the total distance in a route.

MATCH (from:PLACE { name:"COLOGNE" }),

  (to:PLACE { name: "ESSEN"}),path = (from) -[road:LINK\*]- (to)

WITH path, REDUCE (sum = 0, r IN road | sum +  r.DistanceKM) AS dist

ORDER BY dist

LIMIT 1

RETURN path

A picture containing text, accessory, vector graphics

Description automatically generated

We can further format the result a bit with cities passed through, the roads traversed (remember, they are named like A-1 etc.) and the total distance. The head() function takes the first item in a list and tail() takes all but the first.

MATCH path = (:PLACE { name:"COLOGNE" }) -[road:LINK\*]- (:PLACE { name: "ESSEN"})

WITH path, REDUCE (sum = 0, r IN road | sum + r.DistanceKM) AS dist

ORDER BY dist

LIMIT 1

RETURN REDUCE (cities = head(nodes(path)).name, n IN tail(nodes(path)) | cities + '->' + n.name) AS Route,

  REDUCE (rd = head(relationships(path)).name, r IN tail(relationships(path)) | rd + ' > ' + r.name) AS Roads,

  dist + ' km' AS Distance

Graphical user interface, text, application

Description automatically generated

To improvise our solution for real time we are adding road blocks to this scenario

roads A-9 and A6 (between WUPPERTAL - ESSEN and WUPPERTAL -BOCHUM)

MATCH () -[r:LINK]-> ()

WHERE r.name IN ['A-8','A-9','A-6','A-1']

SET r.isBlocked = true

Graphical user interface, text, application

Description automatically generated

Now we consider the road blocks, we filter out and include only the routes where none of the roads are blocked.

MATCH path = (:PLACE  {name: "COLOGNE"}) -[road:LINK\*]- (:PLACE {name: "ESSEN"})

WHERE ALL (r IN road WHERE NOT exists(r.isBlocked)) //all roads are non-blocked

WITH path, REDUCE (sum = 0, r IN road | sum + r.DistanceKM) AS dist

ORDER BY dist

LIMIT 1

RETURN REDUCE (cities = head(nodes(path)).name, n IN tail(nodes(path)) | cities + '->' + n.name) AS Route,

  REDUCE (rd = head(relationships(path)).name, r IN tail(relationships(path)) | rd + ' > ' + r.name) AS Roads,

  dist + ' km' AS Distance

Graphical user interface, text, application, email

Description automatically generated

It produces the following results:

Route: COLOGNE->GELSENKIRCHEN->MÜNSTER->DUISBURG>ESSEN

Roads: A-13 > A-14 > A-15 > A-5

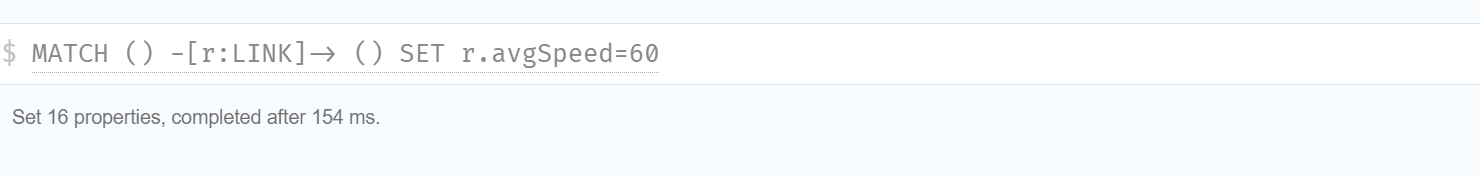
Distance: 770 km.

Now lets introduce the avg speed as well into the scenario as many a times, we want the “fastest route” rather than the “shortest route”.

We update roads (i.e. [:LINK] relationships) with property avgSpeed to the relationship

MATCH () -[r:LINK]-> ()

SET r.avgSpeed=60



we can calculate the fastest route using similar logic, we will print out all the possible routes with order of travel time.

MATCH path = (:PLACE {name:"COLOGNE"}) -[road:LINK\*]-> (:PLACE {name:"ESSEN"})

WHERE ALL (r IN road WHERE NOT EXISTS (r.isBlocked))

WITH path, REDUCE (sum = 0, r IN road | sum + (r.DistanceKM/r.avgSpeed)) AS time

ORDER BY time

RETURN  REDUCE(cities = head(nodes(path)).name, n IN tail(nodes(path)) | cities + '->' + n.name) AS Route,

  REDUCE (rd = head(relationships(path)).name, r IN tail(relationships(path)) | rd + ' > ' + r.name) AS Roads,

  toString((round(time \* 100))/100) + ' hr' AS TotalTime

Table

Description automatically generated