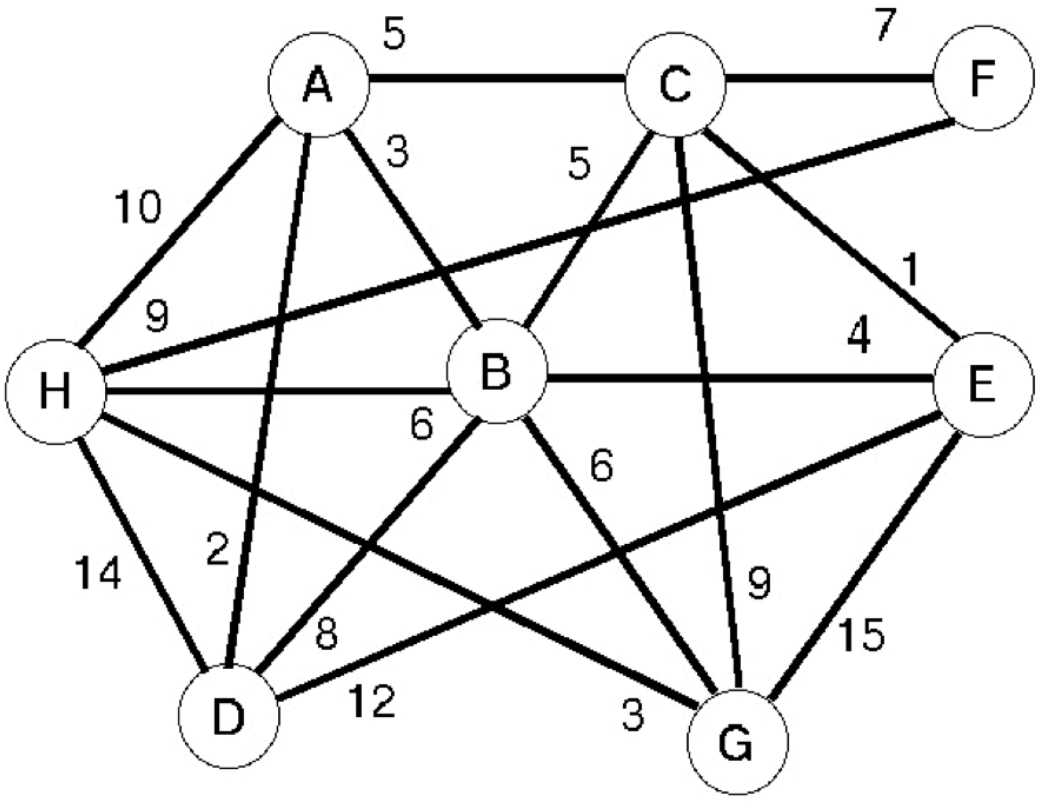
Big Data Aplicado

UD2 - Procesado y Presentación de Datos Almacenados

Práctica 1 Neo4j

> Recuerda que para hacer la prácticas puedes optar por cualquiera de la las 3 opciones.
>
> - Instalarla en tu máquina local
> - Usar SandBox Neo4j.com: https://sandbox.neo4j.com/ con Graph Data Science
> - Crear un contenedor docker.

Ejercicio 1. Dado el grafo de la figura:



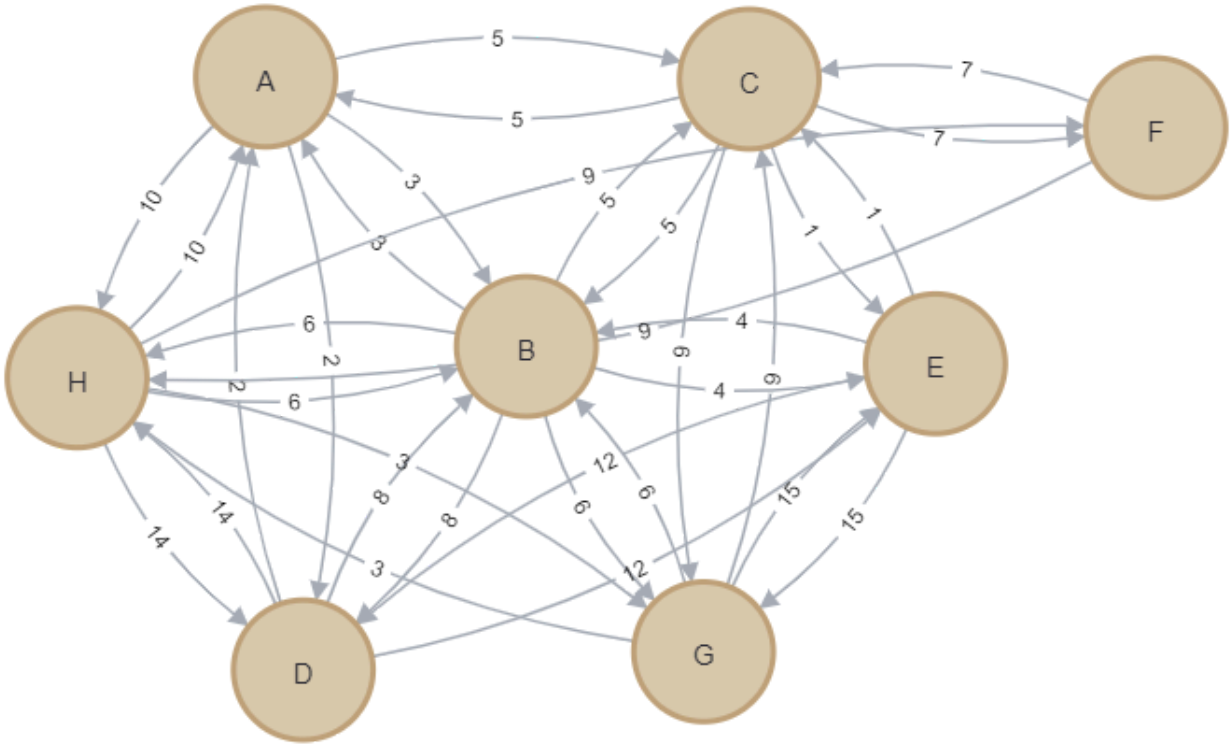
1. Crear el grafo en Neo4j

Para crear el grafo usaremos el siguiente código:

```
// Definimos los nodos y aristas en una lista de mapas
WITH [
  {name: 'A', conexiones: [{destino: 'B', distancia: 3}, {destino: 'C', dis
  {name: 'B', conexiones: [{destino: 'C', distancia: 5}, {destino: 'D', dis
  {name: 'C', conexiones: [{destino: 'E', distancia: 1}, {destino: 'F', dis
  {name: 'D', conexiones: [{destino: 'E', distancia: 12}, {destino: 'H', di
  {name: 'E', conexiones: [{destino: 'G', distancia: 15}]},
  {name: 'F', conexiones: [{destino: 'H', distancia: 9}]},
  {name: 'G', conexiones: [{destino: 'H', distancia: 3}]}
] AS ciudades


// Creamos nodos y aristas de esta forma
UNWIND ciudades AS ciudad
MERGE (c:Location {name: ciudad.name})
FOREACH (conexión IN ciudad.conexiones |
  MERGE (dest:Location {name: conexión.destino})
  MERGE (c)-[:CAMINO {distancia: conexión.distancia}]->(dest)
  MERGE (dest)-[:CAMINO {distancia: conexión.distancia}]->(c)
)
```

El resultado queda representado de esta forma:



2. Recorrer el grafo en anchura y en profundidad, comenzando en el nodo H

Para realizar las búsquedas en anchura y en profundidad debemos primero hacer una proyección del grafo para cada búsqueda y luego la búsqueda en si.

- Búsqueda en Anchura

```
// Primero hacemos la proyección
MATCH (source:Location)-[r:CAMINO]->(target:Location)
RETURN gds.graph.project(
  'myGraph_BFS',
  Location,
  CAMINO
)
```

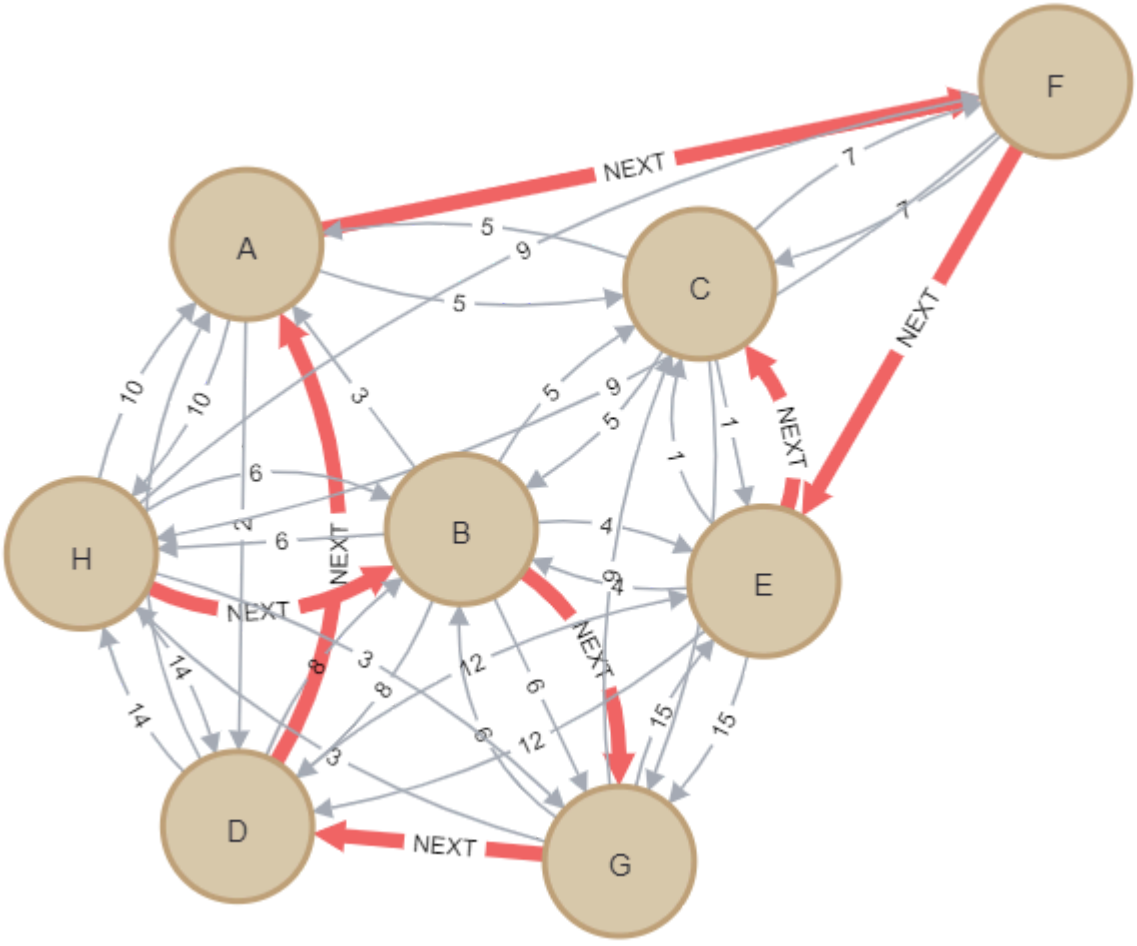
Y obtenemos la proyección:

```
  {
"relationshipCount": 34,
"graphName": "myGraph_BFS",
"query": "MATCH (source:Location)-[r:CAMINO]->(target:Location)
  RETURN gds.graph.project(
      'myGraph_BFS',
      source,
      target
  )",
    "projectMillis": 3264,
    "configuration": {
      "readConcurrency": 4,
      "undirectedRelationshipTypes": [],
```

```
        "jobId": "ceb96dfb-acd8-413b-bcab-c3496ee15ff2",
        "logProgress": true,
        "query": "MATCH (source:Location)-[r:CAMINO]->
    (target:Location)
      RETURN gds.graph.project(
          'myGraph_BFS',
          source,
          target
      )",
          "inverseIndexedRelationshipTypes": []
        },
        "nodeCount": 8
      }
```

Ahora hacemos la búsqueda:

```
MATCH (source:Location {ciudad:'H'})
CALL gds.bfs.stream('myGraph_BFS', {
  sourceNode: source
})
YIELD path
RETURN path
```



El resultado obtenido es: H -> B -> G -> D -> A -> F -> E -> C

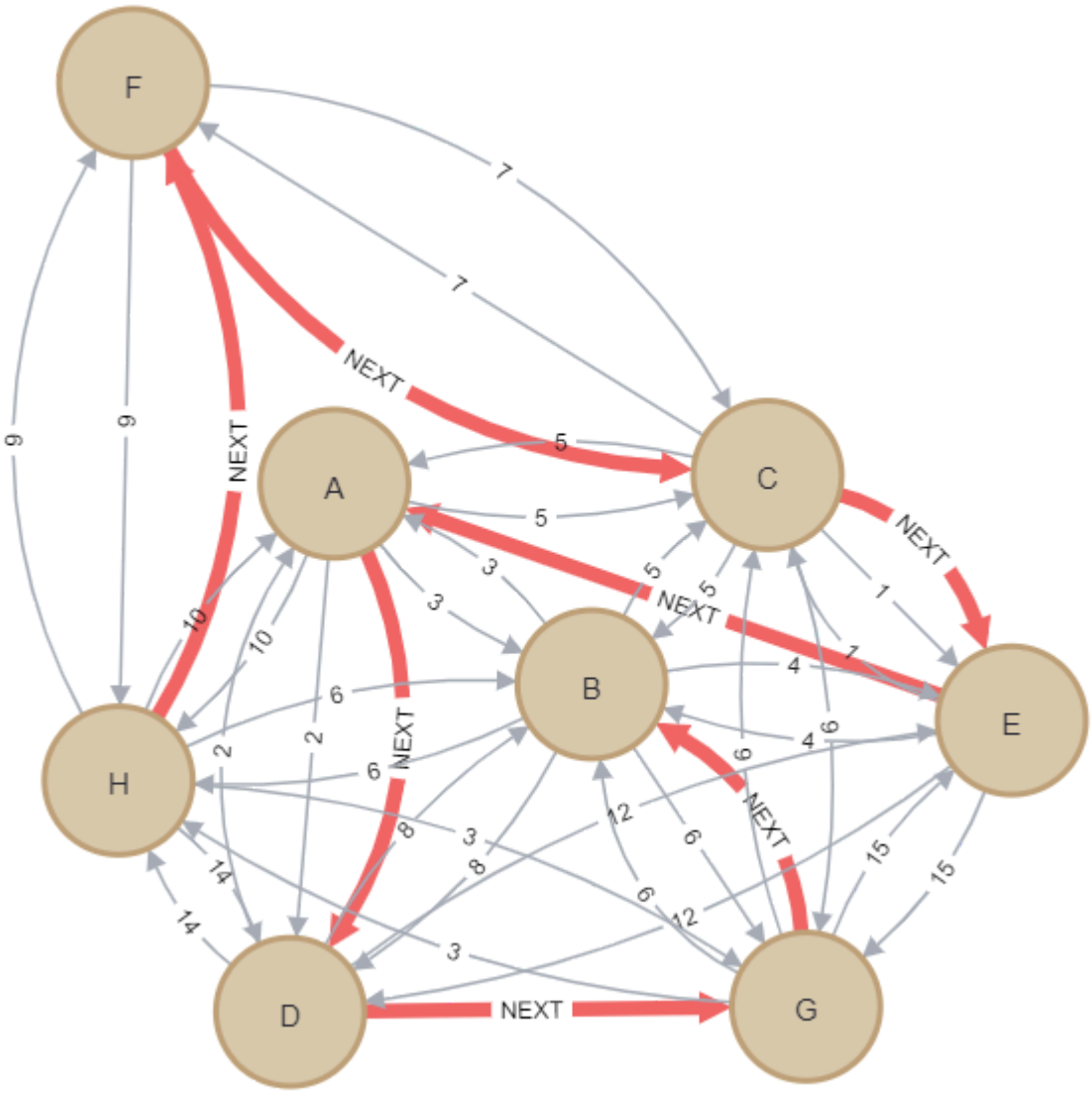
- Búsqueda en Profundidad

```
// Creamos el grafo.
MATCH (source:Location)-[r:CAMINO]->(target:Location)
RETURN gds.graph.project(
  'myGraph_DFS',
  source,
  target
)
```

Y obtenemos lo siguiente:


```
  {
"relationshipCount": 34,
"graphName": "myGraph_DFS",
"query": "// Creamos el grafo.
  MATCH (source:Location)-[r:CAMINO]->(target:Location)
  RETURN gds.graph.project(
      'myGraph_DFS',
      source,
      target
  )",
    "projectMillis": 84,
    "configuration": {
      "readConcurrency": 4,
      "undirectedRelationshipTypes": [],
      "jobId": "0026a874-44cb-47ae-9b9c-5d6247a4f3e0",
      "logProgress": true,
      "query": "// Creamos el grafo.
  MATCH (source:Location)-[r:CAMINO]->(target:Location)
  RETURN gds.graph.project(
      'myGraph_DFS',
      source,
      target
  )",
      "inverseIndexedRelationshipTypes": []
    },
    "nodeCount": 8
  }
```


```
MATCH (source:Location {name:'H'})
CALL gds.dfs.stream('myGraph_DFS', {
  sourceNode: source
})
YIELD path
RETURN path
```



El camino resultante es el siguiente: H -> F -> C -> E -> A -> D -> G -> B

3. Obtener el camino mínimo, utilizando el algoritmo [All Pairs Shortest Path](), entre todos los pares de nodos del grafo desde H.

Hacemos la proyección:

```
MATCH (src:Location)-[r:CAMINO]->(trg:Location)
RETURN gds.graph.project(
  'cypherGraph',
  src,
  trg,
  {
    relationshipType: type(r),
    relationshipProperties: r { .distancia }
  },
  { undirectedRelationshipTypes: ['CAMINO'] }
)
```

Lo que devuelve lo siguiente:

```
{
  "relationshipCount": 68,
  "graphName": "cypherGraph",
  "query": "MATCH (src:Location)-[r:CAMINO]->(trg:Location)
RETURN gds.graph.project(
  'cypherGraph',
  src,
  trg,
  {
    relationshipType: type(r),
    relationshipProperties: r { .distancia }
  },
  { undirectedRelationshipTypes: ['CAMINO'] }
)",
  "projectMillis": 236,
  "configuration": {
    "readConcurrency": 4,
    "undirectedRelationshipTypes": [
      "CAMINO"
    ],
    "jobId": "a8d12713-1821-41f4-8dc7-98f864ea08a7",
    "logProgress": true,
    "query": "MATCH (src:Location)-[r:CAMINO]->(trg:Location)
RETURN gds.graph.project(
  'cypherGraph',
  src,
  trg,
  {
    relationshipType: type(r),
    relationshipProperties: r { .distancia }
  },
  { undirectedRelationshipTypes: ['CAMINO'] }
)",
    "inverseIndexedRelationshipTypes": []
  },
  "nodeCount": 8
}
```


Y luego hacemos la búsqueda del camino mínimo:

```
CALL gds.allShortestPaths.stream('cypherGraph', {
  relationshipWeightProperty: 'distancia'
})
YIELD sourceNodeId, targetNodeId, distance
WITH sourceNodeId, targetNodeId, distance
WHERE gds.util.isFinite(distance) = true
WITH gds.util.asNode(sourceNodeId) AS source, gds.util.asNode(targetNodeId)
WHERE source.name = 'H' AND source <> target

RETURN source.name AS source, target.name AS target, distance
```

```
ORDER BY distance DESC, target.name ASC
LIMIT 10
```

Este código devuelve la siguiente tabla:

Nodo de partida	Nodo objetivo	Distancia
H	C	11
H	D	11
H	E	10
H	A	10
H	F	9
H	B	6
H	G	3

4. Obtener el camino mínimo, utilizando el algoritmo de Dijkstra, entre D y F

Hacemos la proyección:

```
// Creamos la Proyección del grafo
CALL gds.graph.project(
  'myGraph_Dijkstra',
  'Location',               // Nombre del nodo en el grafo
  'CAMINO',                 // Tipo de relación
  {
    relationshipProperties: 'distancia'  // Propiedad de la relación que re
  }
)
```

Nos devuelve la siguiente tabla:

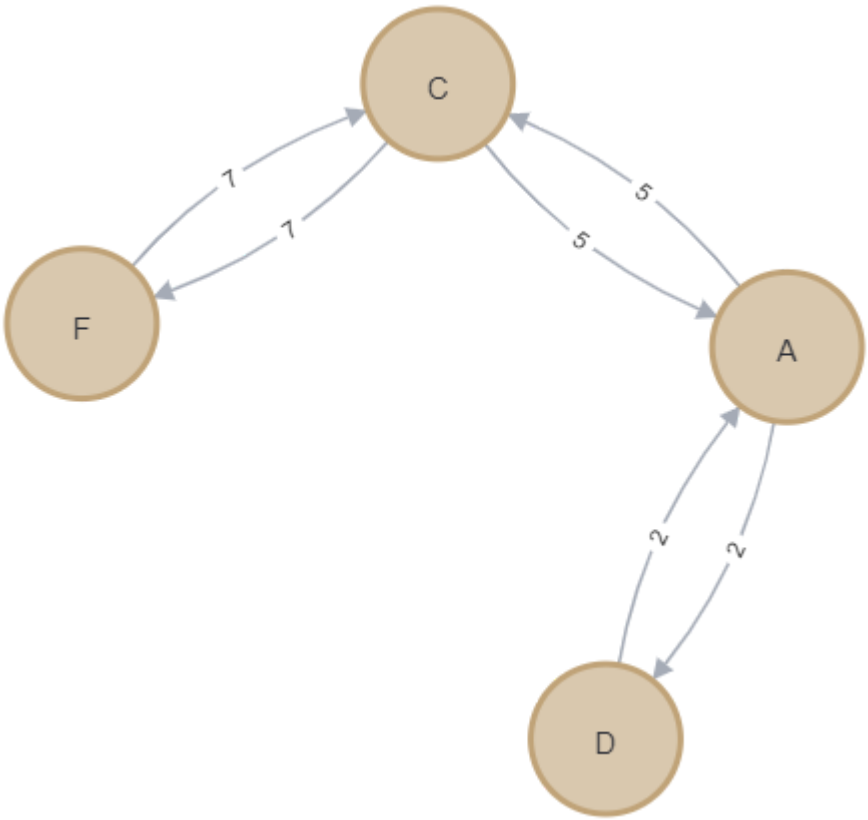
```
╒═══════════════════════════════════════════════════════════╤═══════════════════════════════════
│nodeProjection                                             │relationshipProjection
│graphName         │nodeCount│relationshipCount│projectMillis│
╞═══════════════════════════════════════════════════════════╪═══════════════════════════════════
│{Location: {label: "Location", properties: {}}}│{CAMINO: {aggregation:
"DEFAULT", orientation: "NATURAL", indexInverse│"myGraph_Dijkstra"│8
│34                │45                 │
│                                                           │: false, properties:
{distancia: {aggregation: "DEFAULT", property: "d│                   │
│                  │                   │
```

```
|                                                  |istancia",
defaultValue: null}}, type: "CAMINO"}}                      |
|        |                |            |
└──────────────────────────────────────────────┘└──────────────────────────┘
```

Y buscamos con la proyección:

```
// Encontramos la ruta más corta entre los nodos D y F
MATCH (source:Location {name: 'D'}), (target:Location {name: 'F'})
CALL gds.shortestPath.dijkstra.stream('myGraph_Dijkstra', {
    sourceNode: source,
    targetNode: target,
    relationshipWeightProperty: 'distancia'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN
    index,
    gds.util.asNode(sourceNode).name AS sourceNodeName,
    gds.util.asNode(targetNode).name AS targetNodeName,
    totalCost,
    [nodeId IN nodeIds | gds.util.asNode(nodeId).name] AS nodeNames,
    costs,
    nodes(path) AS path
ORDER BY index
```

Y obtenemos el siguiente grafo:



Indicandonos que el camino más corto entre D y F es: D -> A -> C -> F

5. Obtener el camino mínimo, utilizando el algoritmo de Yen, entre D y F

Creamos la proyección:

```
CALL gds.graph.project(
  'caminoGrafo',
  'Location',
  {
    CAMINO: {
      properties: 'distancia'
    }
  }
)
```

Aqhí obtenemos el resultado del código anterior:

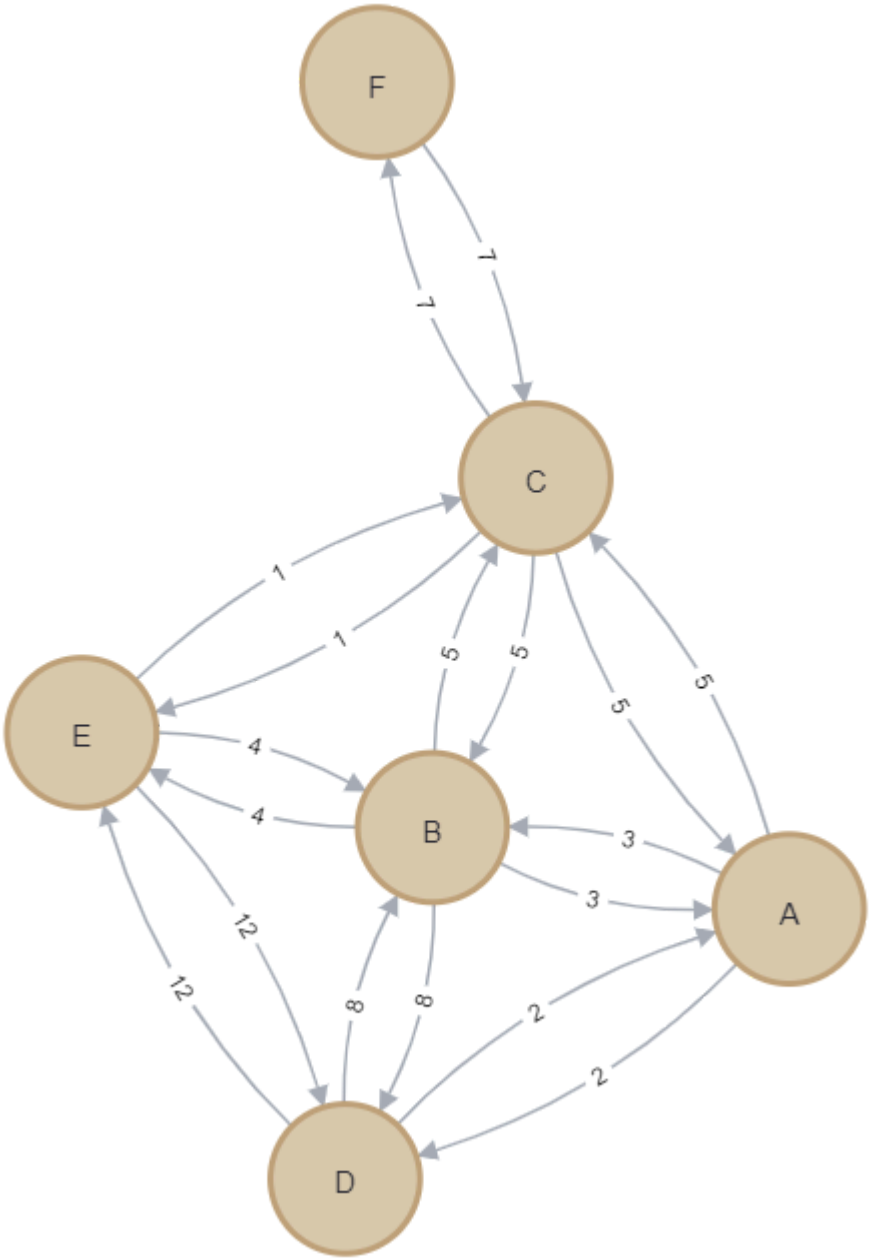
```
╒═══════════════════════════════════════════════════════════════════════╤═══════════════════════════════════════════════════════════════════════╕
│nodeProjection                                                         │relationshipProjection                                                 │
│graphName    │nodeCount│relationshipCount│projectMillis│               │                                                                       │
╞═══════════════════════════════════════════════════════════════════════╪═══════════════════════════════════════════════════════════════════════╡
│{Location: {label: "Location", properties: {}}}│{CAMINO: {aggregation:
"DEFAULT", orientation: "NATURAL", indexInverse│"caminoGrafo"│8
│34               │28          │
│                                                                       │: false, properties:
{distancia: {aggregation: "DEFAULT", property: "d│             │
│                 │            │
│                                                                       │istancia",
defaultValue: null}}, type: "CAMINO"}}                    │
│         │                 │             │
└─────────────────────────────────────────────────────────────────────────┴─────────────────────────────────────────────────────────────────────────┘
```

Y luego trabajamos el algoritmo de Yen sobre ella:

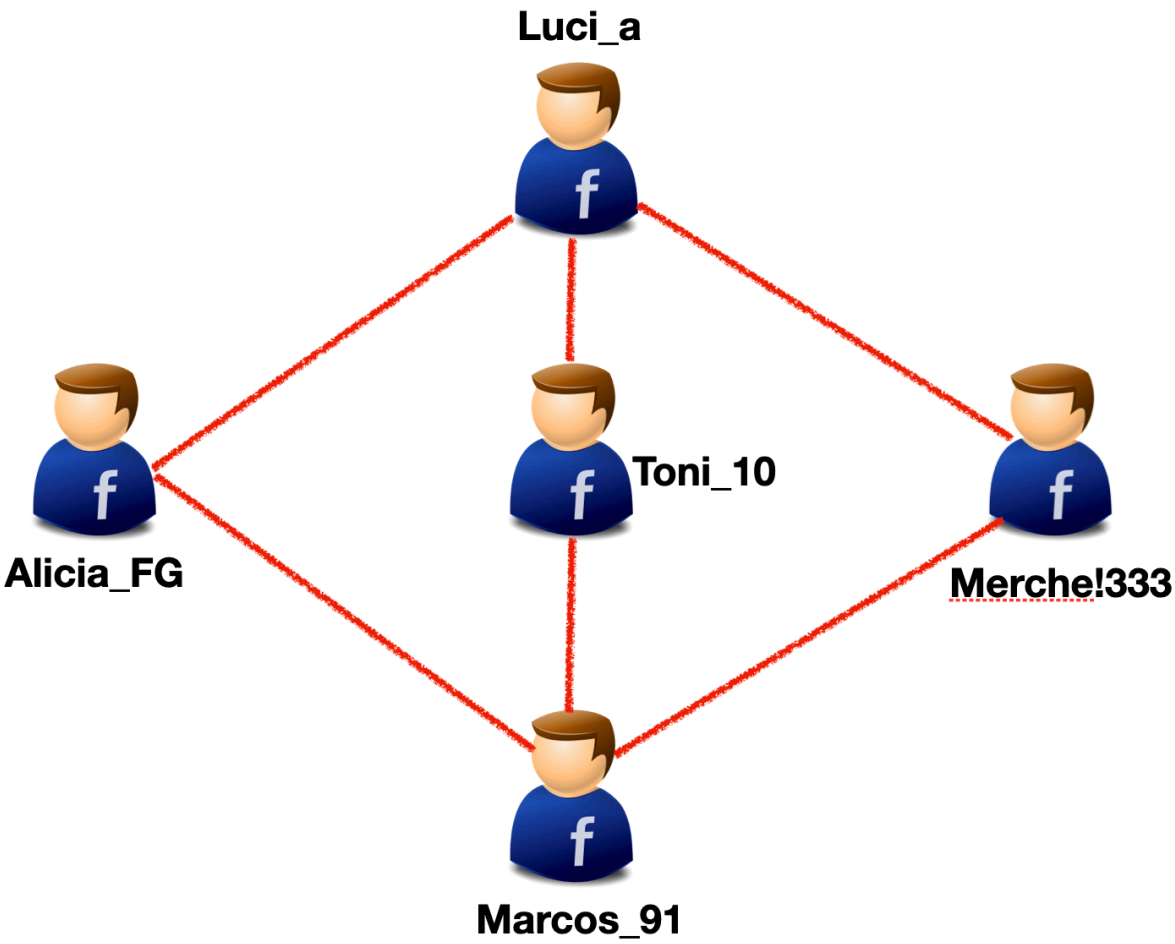
```
MATCH (source:Location {name: 'D'}), (target:Location {name: 'F'})
CALL gds.shortestPath.yens.stream('caminoGrafo', {
    sourceNode: source,
    targetNode: target,
    k: 3,
    relationshipWeightProperty: 'distancia'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN
    index,
    gds.util.asNode(sourceNode).name AS sourceNodeName,
    gds.util.asNode(targetNode).name AS targetNodeName,
    totalCost,
```

```
    [nodeId IN nodeIds | gds.util.asNode(nodeId).name] AS nodeNames,
    costs,
    nodes(path) as path
ORDER BY index
```

Lo que devuelve el siguiente grafo:



Ejercicio 2. Dado el grafo de la figura, que representa una red social de Facebook:

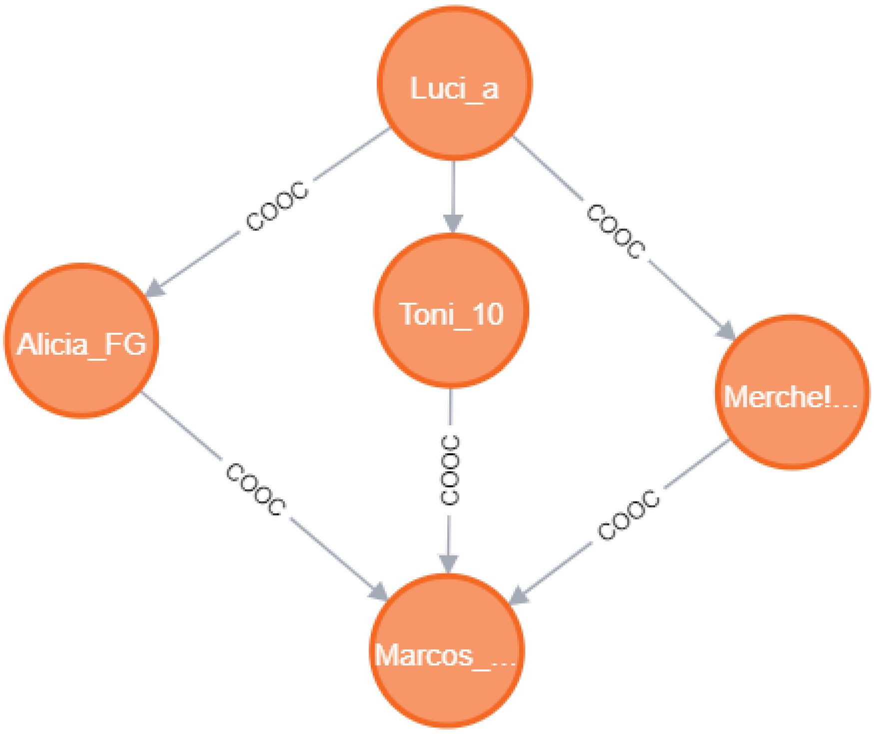


Para crear el grafo de red social, haremos el siguiente código:

```
CREATE
(LU:Hashtag {name: 'Luci_a'}),
(TN:Hashtag {name: 'Toni_10'}),
(MC:Hashtag {name: 'Marcos_91'}),
(AL:Hashtag {name: 'Alicia_FG'}),
(ME:Hashtag {name: 'Merche!333'}),

(LU)-[:COOC {ntweet: 52}]->(TN),
(TN)-[:COOC {ntweet: 52}]->(MC),
(LU)-[:COOC {ntweet: 183}]->(AL),
(AL)-[:COOC {ntweet: 183}]->(MC),
(LU)-[:COOC {ntweet: 73}]->(ME),
(ME)-[:COOC {ntweet: 73}]->(MC)
```

Lo que da el siguiente resultado:



1. Obtener las medidas de centralidad de grado, cercanía e intermediación para cada uno de los nodos.

Al igual que antes, primero hacemos la proyección y luego hacemos la consulta.

- Centralidad del Grado

```
CALL gds.graph.project(
  'myGraph',
  ['Hashtag'],
  {
    COOC: {
      properties: 'ntweet'
    }
  }
);
```

El resultado dado se refleja en la siguiente tabla:

```
╒═════════════════════════════════════════════════════════════════════╤═══════════════════════════════════════════════════════
│nodeProjection
│relationshipProjection
│graphName│nodeCount│relationshipCount│projectMillis│
╞═════════════════════════════════════════════════════════════════════╪═══════════════════════════════════════════════════════
│{Hashtag: {label: "Hashtag", properties: {}}}│{COOC: {aggregation:
"DEFAULT", orientation: "NATURAL", indexInverse: │"myGraph"│5
│6                 │2862          │
│                                             │false, properties:
{ntweet: {aggregation: "DEFAULT", property: "ntweet│          │
│                  │              │
```

```
|                                                  |", defaultValue:
null}}, type: "COOC"}}                                  |         |
|                 |             |
└──────────────────────────────────────────────────┴─────────────────────
```

```
CALL gds.degree.stream('myGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score AS degree
ORDER BY degree DESC;
```

La tabla devuelta es la siguiente:

Usuario	Conexiones
Luci_a	3
Toni_10	1
Alicia_FG	1
Merche!333	1
Marcos_91	0

- Centralidad de cercanía

```
// Crear la proyección del grafo con los nodos y relaciones de Hashtag
CALL gds.graph.project(
  'myGraph_cercania',
  ['Hashtag'],          // Tipo de nodo
  ['COOC']              // Tipo de relación
);
```

```
╒══════════════════════════════════════════════════╤═══════════════════
|nodeProjection
|relationshipProjection
|graphName         |nodeCount|relationshipCount|projectMillis|
└──────────────────────────────────────────────────┴───────────────────
╒══════════════════════════════════════════════════╤═══════════════════
|{Hashtag: {label: "Hashtag", properties: {}}}|{COOC: {aggregation:
"DEFAULT", orientation: "NATURAL", indexInverse:
|"myGraph_cercania"|5        |6                |16           |
|                                                  |false, properties:
{}, type: "COOC"}}                                     |
```

```
|         |                  |             |
└─────────────────────────────────────────┴─────────────────────────┘
```

```
// Calcular centralidad de cercanía
CALL gds.closeness.stream('myGraph_cercania')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score AS closeness
ORDER BY closeness DESC;
```

Usuario	Cercanía
Toni_10	1
Alicia_FG	1
Merche!333	1
Marcos_91	0.8
Luci_a	0

- Centralidad de Intermediación

```
// Proyectar el grafo en GDS usando nodos Hashtag y relaciones COOC
CALL gds.graph.project(
  'myGraph_intermediacion',
  ['Hashtag'],
  'COOC'
);
```

```
╒═══════════════════════════════════════════════════════════════════════╤═══════════════════════════════════════════════════════════════════════╕
│nodeProjection                                                         │
│relationshipProjection                                                 │
│graphName                │nodeCount│relationshipCount│projectMillis│
╞═══════════════════════════════════════════════════════════════════════╪═══════════════════════════════════════════════════════════════════════╡
│{Hashtag: {label: "Hashtag", properties: {}}}│{COOC: {aggregation:
"DEFAULT", orientation: "NATURAL", indexInverse:
│"myGraph_intermediacion"│5        │6                │29           │
│                                          │false, properties:
{}, type: "COOC"}}                                  │
│         │                  │             │
└─────────────────────────────────────────┴─────────────────────────┘
```

```
CALL gds.betweenness.stream('myGraph_intermediacion')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score AS betweenness
ORDER BY betweenness DESC;
```

Usuario	Influencia
Toni_10	0.33
Alicia_FG	0.33
Merche!333	0.33
Luci_a	0
Marcos_91	0

1. Calcular, mediante todos los métodos de predicción de enlace vistos, los valores de posibilidad de que se produzca un nuevo enlace entre Alicia_FG y Merche!33 y entre Toni_10 y Alicia_FG.

- Vecinos comunes

```
//Alice_FG y Merche!333
MATCH (x:Hashtag {name: 'Alicia_FG'})
MATCH (y:Hashtag {name: 'Merche!333'})
RETURN gds.alpha.linkprediction.commonNeighbors(x, y) AS score
```

El resultado de **score** sería de 2.

```
//Toni_10 y Alicia_FG
MATCH (x:Hashtag {name: 'Toni_10'})
MATCH (y:Hashtag {name: 'Alicia_FG'})
RETURN gds.alpha.linkprediction.commonNeighbors(x, y) AS score
```

Volvemos a obtener una **score** de 2.

- Adhesión preferencial

```
MATCH (x:Hashtag {name: 'Alicia_FG'})
MATCH (y:Hashtag {name: 'Merche!333'})
RETURN gds.alpha.linkprediction.preferentialAttachment(x, y) AS score
```

El resultado de **score** sería de 4.

```
MATCH (x:Hashtag {name: 'Toni_10'})
MATCH (y:Hashtag {name: 'Alicia_FG'})
RETURN gds.alpha.linkprediction.preferentialAttachment(x, y) AS score
```

Volvemos a obtener una **score** de 4.

- Asignación de recursos

```
MATCH (x:Hashtag {name: 'Alicia_FG'})
MATCH (y:Hashtag {name: 'Merche!333'})
RETURN gds.alpha.linkprediction.resourceAllocation(x, y) AS score
```

El resultado de **score** sería de 0.66.

```
MATCH (x:Hashtag {name: 'Toni_10'})
MATCH (y:Hashtag {name: 'Alicia_FG'})
RETURN gds.alpha.linkprediction.resourceAllocation(x, y) AS score
```

El **score** es la misma, 0.66.

Los resultados arrojados indican que Alicia_FG puede crear con mayor probabilidad un nuevo enlace con Toni_10 que con Merche!333.