Laboratorium 14
Analiza danych BDOT10k w bazie Neo4j

Cele

- Wykonanie przykładów testowych
- Uruchomienie algorytmów wyszukujących najkrótszą ścieżkę pomiędzy wybranymi węzłami w grafie z bloku A.
- Testy wybranych algorytmów grafowych biblioteki GDS na grafie z bloku A.

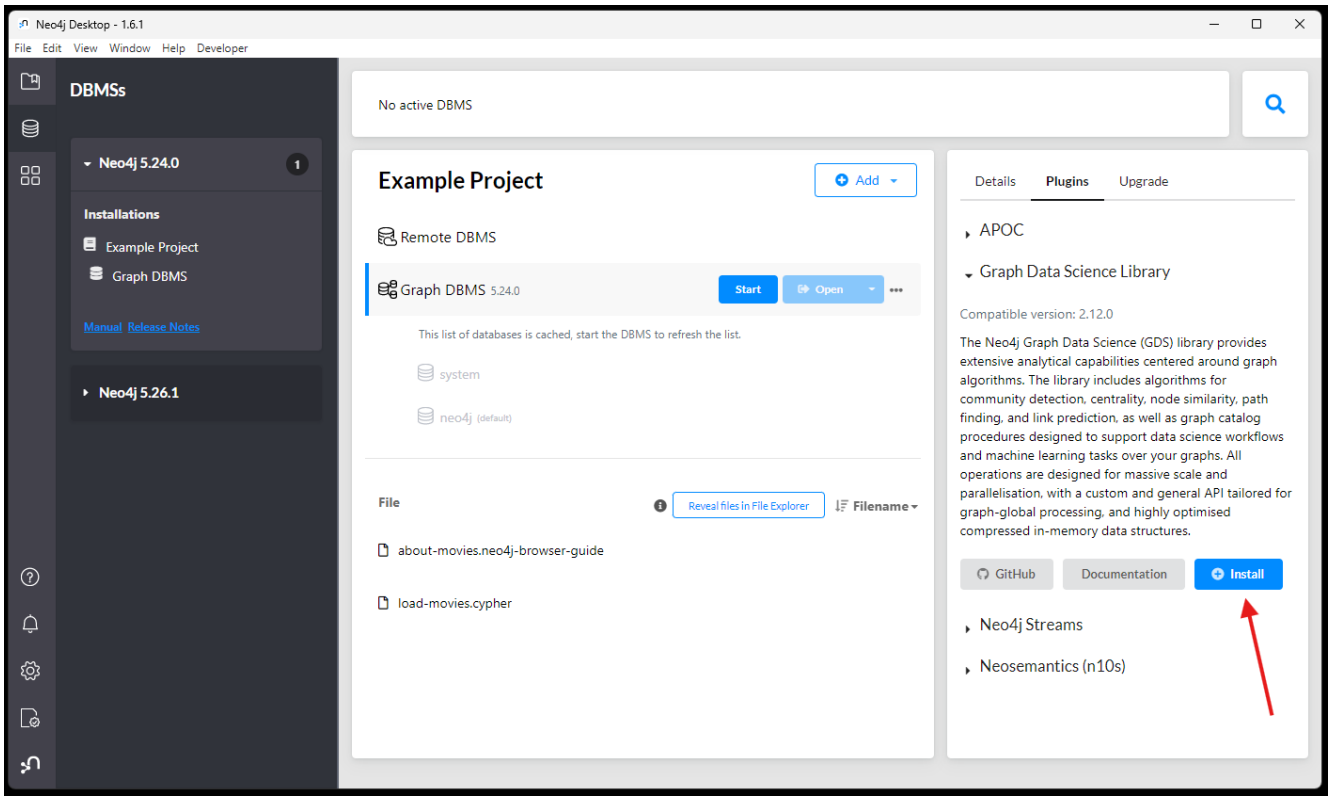
Dane

- Dane BDOT10k wykorzystywane w zadaniu w bloku A.

Neo4j Graph Data Science

Biblioteka Neo4j Graph Data Science (GDS) dostarcza wydajne wersje popularnych algorytmów grafowych, udostępnionych jako procedury języka Cypher. Ponadto GDS zawiera pipeline'y uczenia maszynowego do trenowania predykcyjnych modeli nadzorowanych w celu rozwiązywania problemów związanych z grafami, takich jak przewidywanie brakujących relacji.

Instalacja GDS w bazie Neo4j



https://neo4j.com/docs/graph-data-science/current/introduction/

Uwaga: należy zwrócić uwagę na zgodność wersji GDS z wersjami Neo4j. Użycie starej wersji Neo4j może skutkować niemożnością uruchomiania opisanych niżej przykładów.
https://neo4j.com/docs/graph-data-science/current/installation/supported-neo4j-versions/

Lista dostępnych algorytmów

https://neo4j.com/docs/graph-data-science/current/algorithms/

Proces uruchamiania algorytmów

1. Załadowanie danych do bazy.
2. Utworzenie w pamięci operacyjnej (projekcja) grafu opartego na wybranych węzłach i krawędziach, na którym będzie uruchamiany wybrany algorytm.
3. Uruchomienie algorytmu.
4. Analiza wyników.

Przykład 1. Algorytm Dijkstry

Opis

https://neo4j.com/docs/graph-data-science/current/algorithms/dijkstra-source-target/

Załadowanie danych

```
CREATE (a:Location {name: 'A'}),
       (b:Location {name: 'B'}),
       (c:Location {name: 'C'}),
       (d:Location {name: 'D'}),
       (e:Location {name: 'E'}),
       (f:Location {name: 'F'}),
       (a)-[:ROAD {cost: 50}]->(b),
       (a)-[:ROAD {cost: 50}]->(c),
       (a)-[:ROAD {cost: 100}]->(d),
       (b)-[:ROAD {cost: 40}]->(d),
       (c)-[:ROAD {cost: 40}]->(d),
       (c)-[:ROAD {cost: 80}]->(e),
       (d)-[:ROAD {cost: 30}]->(e),
       (d)-[:ROAD {cost: 80}]->(f),
       (e)-[:ROAD {cost: 40}]->(f);
```

Projekcja grafu

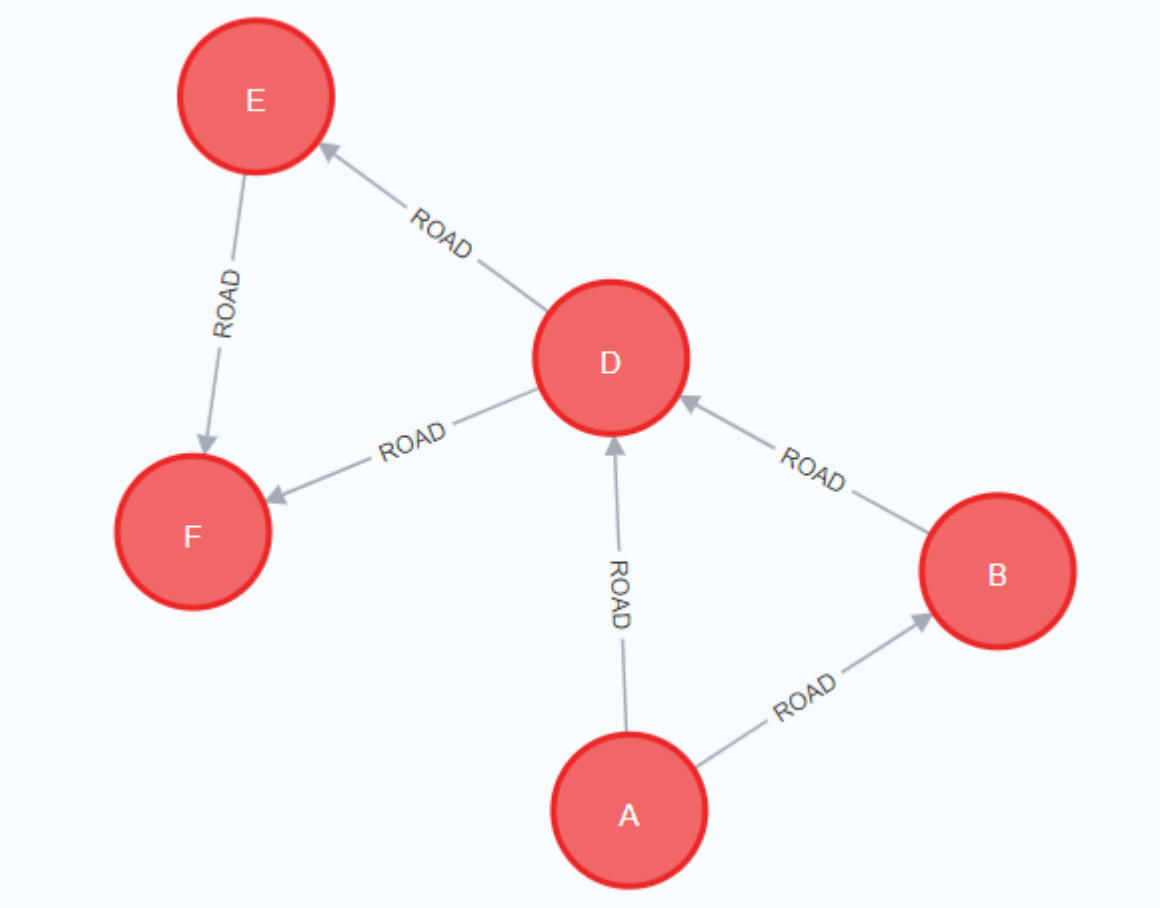
```
MATCH (source:Location)-[r:ROAD]->(target:Location)
RETURN gds.graph.project(
  'NavGraph1',
  source,
  target,
  { relationshipProperties: r { .cost } }
)
```

Uruchomienie algorytmu

```
MATCH (source:Location {name: 'A'}), (target:Location {name: 'F'})
CALL gds.shortestPath.dijkstra.stream('NavGraph1', {
    sourceNode: source,
    targetNodes: target,
    relationshipWeightProperty: 'cost'
})
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
```

Analiza wyników

index	Source Node Name	Target Node Name	totalCost	nodeNames	costs	path
0	"A"	"F"	160.0	["A", "B", "D", "E", "F"]	[0.0, 50.0, 90.0, 120.0, 160.0	[(:Location {name: "A"}), (:Location {name: "B"}), (:Location {name: "D"}), (:Location {name: "E"}), (:Location {name: "F"})]



Przykład 2. Algorytm A*

Opis

https://neo4j.com/docs/graph-data-science/current/algorithms/astar/

Załadowanie danych

```
CREATE (a:Station {name: 'Kings Cross', latitude: 51.5308, longitude: -
0.1238}),
       (b:Station {name: 'Euston', latitude: 51.5282, longitude: -
0.1337}),
       (c:Station {name: 'Camden Town', latitude: 51.5392, longitude: -
0.1426}),
       (d:Station {name: 'Mornington Crescent', latitude: 51.5342,
longitude: -0.1387}),
       (e:Station {name: 'Kentish Town', latitude: 51.5507, longitude: -
0.1402}),
       (a)-[:CONNECTION {distance: 0.7}]->(b),
       (b)-[:CONNECTION {distance: 1.3}]->(c),
       (b)-[:CONNECTION {distance: 0.7}]->(d),
       (d)-[:CONNECTION {distance: 0.6}]->(c),
       (c)-[:CONNECTION {distance: 1.3}]->(e)
```

Projekcja grafu

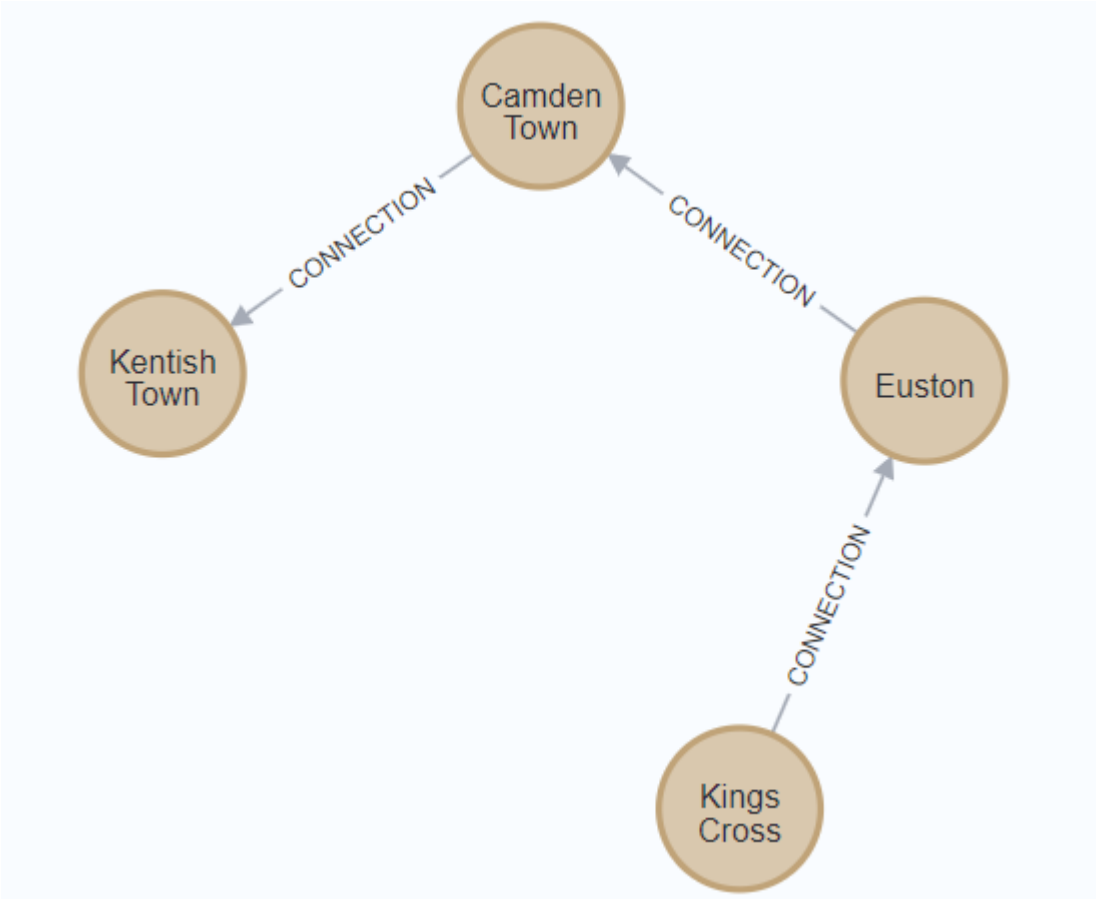
```
MATCH (source:Station)-[r:CONNECTION]->(target:Station)
RETURN gds.graph.project(
  'NavGraph2',
  source,
  target,
  {
    sourceNodeProperties: source { .latitude, .longitude },
    targetNodeProperties: target { .latitude, .longitude },
    relationshipProperties: r { .distance }
  }
)
```

Uruchomienie algorytmu

```
MATCH (source:Station {name: 'Kings Cross'}), (target:Station {name:
'Kentish Town'})
CALL gds.shortestPath.astar.stream('NavGraph2', {
    sourceNode: source,
    targetNode: target,
    latitudeProperty: 'latitude',
    longitudeProperty: 'longitude',
    relationshipWeightProperty: 'distance'
})
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
```

Analiza wyników

index	Source Node Name	Target Node Name	totalCost	nodeNames	costs	path
0	"Kings Cross"	"Kentish Town"	3.3	["Kings Cross", "Euston", "Camden Town", "Kentish Town"]	[0.0, 0.7, 2.0, 3.3]	[(:Station {name: "Kings Cross",latitude: 51.5308,longitude: -0.1238}), (:Station {name: "Euston",latitude: 51.5282,longitude: -0.1337}), (:Station {name: "Camden Town",latitude: 51.5392,longitude: -0.1426}), (:Station {name: "Kentish Town",latitude: 51.5507,longitude: -0.1402})]



Przykład 3: Degree Centrality

Opis

https://neo4j.com/docs/graph-data-science/current/algorithms/degree-centrality/

Załadowanie danych

```
CREATE
  (alice:User {name: 'Alice'}),
  (bridget:User {name: 'Bridget'}),
  (charles:User {name: 'Charles'}),
  (doug:User {name: 'Doug'}),
  (mark:User {name: 'Mark'}),
  (michael:User {name: 'Michael'}),

  (alice)-[:FOLLOWS {score: 1}]->(doug),
  (alice)-[:FOLLOWS {score: -2}]->(bridget),
  (alice)-[:FOLLOWS {score: 5}]->(charles),
  (mark)-[:FOLLOWS {score: 1.5}]->(doug),
  (mark)-[:FOLLOWS {score: 4.5}]->(michael),
  (bridget)-[:FOLLOWS {score: 1.5}]->(doug),
  (charles)-[:FOLLOWS {score: 2}]->(doug),
  (michael)-[:FOLLOWS {score: 1.5}]->(doug)
```

Projekcja grafu

```
MATCH (source:User)-[r:FOLLOWS]->(target:User)
RETURN gds.graph.project(
  'CentralityGraph',
  target,
  source,
  { relationshipProperties: r { .score } }
)
```

Uruchomienie algorytmu

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

Analiza wyników

name	followers
"Doug"	5.0
"Michael"	1.0
"Charles"	1.0
"Bridget"	1.0
"Mark"	0.0
"Alice"	0.0