A logo of a book and tree

Description automatically generated

Αναπαράσταση Γνώσης στον Παγκόσμιο Ιστό

Project Χειμερινού Εξαμήνου

*Δεμερτζόγλου Ευστράτιος | ΤΗ20580*

*Μητσόπουλος Νικόλαος |ΤΗ20591*

*Τσιάρας Άγγελος |ΤΗ20465*

Table of Contents

[Εισαγωγή 3](#_Toc188722190)

[Επεξήση Κώδικα: 3](#_Toc188722191)

[Libraries Used: 3](#_Toc188722192)

[Function **find\_path(start\_entity, end\_entity, max\_depth=4)**: 3](#_Toc188722193)

[Function **build\_query(start\_entity, end\_entity, depth)**: 4](#_Toc188722194)

[Παραδείγματα Χρήσης 5](#_Toc188722195)

[start\_entity: Einstein , stop\_entity: Italy 5](#_Toc188722196)

[start\_entity:Olives , end\_entity:Crete 7](#_Toc188722197)

[Code that was implemented for the project 8](#_Toc188722198)

# Εισαγωγή

Αυτή η αναφορά περιγράφει τη διαδικασία για την εύρεση μονοπατιού μεταξύ δύο οντοτήτων σε έναν γράφο γνώσης, όπως το DBpedia, μέσω SPARQL ερωτημάτων και Python. Αναλύεται ο κώδικας, οι τροποποιήσεις που πραγματοποιήθηκαν, καθώς και παραδείγματα εκτέλεσης και αποτελέσματα.

# Επεξήση Κώδικα:

## Libraries Used:

from SPARQLWrapper import SPARQLWrapper, JSON

Χρήση της βιβλιοθήκης **SPARQLWrapper** για τη διασύνδεση endpoints SPARQL σε format **JSON**

## Function **find\_path(start\_entity, end\_entity, max\_depth=4)**:

Δέχεται arguments την αρχική οντότητα και την τελική οντότητα αλλά και το μέγιστο βάθος που μπορεί να φτάσει η συνάρτηση.

**sparql = SPARQLWrapper("http://dbpedia.org/sparql"):**

Δημιουργούμε ένα SPARQLWrapper αντικείμενο που συνδέεται στο endpoint του DBpedia

**start\_entity\_uri & end\_entity\_uri :**

Μετατρέπει τις οντότητες σε URI μορφή του DBpedia

**for loop:**

εξετάζει διαδοχικά βάθη από 1 έως max\_depth

θέτει τα ερωτήματα και καθορίζει τη μορφή επιστροφής

καταγράφει τυχόν σφάλματα

## Function **build\_query(start\_entity, end\_entity, depth)**:

Δημιουργεί το SPARQL ερώτημα για το καθορισμένο βάθος.

Αν το βάθος είναι 1, δημιουργούμε απλό ερώτημα για άμεση σχέση.

Για βάθος μεγαλύτερο του 1, δημιουργούμε επαναληπτικά τις τριπλέτες.

Πρώτη τριπλέτα ξεκινά με την οντότητα εκκίνησης.

Τελευταία τριπλέτα συνδέει την τελευταία μεταβλητή με την οντότητα τερματισμού.

Ενδιάμεσες τριπλέτες συνδέουν μεταβλητές.

Επιστρέφουμε το τελικό SPARQL ερώτημα.

# Παραδείγματα Χρήσης

### start\_entity: Einstein , stop\_entity: Italy

**Output:**

Trying depth: 1

Constructed Query:

SELECT \* WHERE {

<http://dbpedia.org/resource/Einstein> ?relation <http://dbpedia.org/resource/Italy> .

}

Trying depth: 2

Constructed Query:

SELECT \* WHERE {

<http://dbpedia.org/resource/Einstein> ?relation0 ?x0 . ?x0 ?relation1 <http://dbpedia.org/resource/Italy> .

}

Trying depth: 3

Constructed Query:

SELECT \* WHERE {

<http://dbpedia.org/resource/Einstein> ?relation0 ?x0 . ?x0 ?relation1 ?x1 . ?x1 ?relation2 <http://dbpedia.org/resource/Italy> .

}

Path found!

{'relation0': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}, 'x0': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Albert\_Einstein'}, 'relation1': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}, 'x1': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Pavia'}, 'relation2': {'type': 'uri', 'value': 'http://www.w3.org/2000/01/rdf-schema#seeAlso'}}

{'relation0': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageRedirects'}, 'x0': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Albert\_Einstein'}, 'relation1': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}, 'x1': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Pavia'}, 'relation2': {'type': 'uri', 'value': 'http://www.w3.org/2000/01/rdf-schema#seeAlso'}}

{'relation0': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}, 'x0': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Albert\_Einstein'}, 'relation1': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}, 'x1': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Milan'}, 'relation2': {'type': 'uri', 'value': 'http://www.w3.org/2000/01/rdf-schema#seeAlso'}}

{'relation0': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageRedirects'}, 'x0': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Albert\_Einstein'}, 'relation1': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}, 'x1': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Milan'}, 'relation2': {'type': 'uri', 'value': 'http://www.w3.org/2000/01/rdf-schema#seeAlso'}}

### start\_entity:Olives , end\_entity:Crete

**Output:**

Trying depth: 1

Constructed Query:

SELECT \* WHERE {

<http://dbpedia.org/resource/Olives> ?relation <http://dbpedia.org/resource/Crete> .

}

Trying depth: 2

Constructed Query:

SELECT \* WHERE {

<http://dbpedia.org/resource/Olives> ?relation0 ?x0 . ?x0 ?relation1 <http://dbpedia.org/resource/Crete> .

}

Path found!

{'relation0': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}, 'x0': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Olive'}, 'relation1': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}}

{'relation0': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageRedirects'}, 'x0': {'type': 'uri', 'value': 'http://dbpedia.org/resource/Olive'}, 'relation1': {'type': 'uri', 'value': 'http://dbpedia.org/ontology/wikiPageWikiLink'}}

# Code that was implemented for the project

from SPARQLWrapper import SPARQLWrapper, JSON

def find\_path(start\_entity, end\_entity, max\_depth=4):

    sparql = SPARQLWrapper("http://dbpedia.org/sparql")

    # Convert entities into DBpedia URIs

    start\_entity\_uri = f"http://dbpedia.org/resource/{start\_entity}"

    end\_entity\_uri = f"http://dbpedia.org/resource/{end\_entity}"

    # Iteratively expand the query

    for depth in range(1, max\_depth + 1):

        print(f"Trying depth: {depth}")

        query = build\_query(start\_entity\_uri, end\_entity\_uri, depth)

        print(f"Constructed Query:\n{query}")

        sparql.setQuery(query)

        sparql.setReturnFormat(JSON)

        try:

            results = sparql.query().convert()

            if results["results"]["bindings"]:

                print("Path found!")

                return results["results"]["bindings"]

        except Exception as e:

            print(f"An error occurred: {e}")

            break

    print("No path found.")

    return None

def build\_query(start\_entity, end\_entity, depth):

    """

    Build a SPARQL query with the specified depth.

    """

    if depth == 1:

        # Direct relationship query

        return f"""

        SELECT \* WHERE {{

            <{start\_entity}> ?relation <{end\_entity}> .

        }}

        """

    else:

        # Iterative query for depth > 1

        path = []

        for i in range(depth):

            if i == 0:

                # First triple starts with the start\_entity

                path.append(f"<{start\_entity}> ?relation{i} ?x{i}")

            elif i == depth - 1:

                # Final triple connects to the end\_entity

                path.append(f"?x{i - 1} ?relation{i} <{end\_entity}>")

            else:

                # Intermediate triples link variables

                path.append(f"?x{i - 1} ?relation{i} ?x{i}")

        return f"""

        SELECT \* WHERE {{

            {' . '.join(path)} .

        }}

        """

# Example Execution

if \_\_name\_\_ == "\_\_main\_\_":

    start = "Olives"

    end = "Crete"

    result = find\_path(start, end, max\_depth=4)

    if result:

        for binding in result:

            print(binding)

    else:

        print("No path found.")