Coursework Project (Part 2)

Member 1: Kai Sun Anson Lam (Anson)

A. Tabular Data to Knowledge Graph (Task RDF)

Subtask RDF.0: Dataset Extension
The provided CSV dataset is extended with 4 rows, which are 2 new restaurants selling 2 new pizzas each. While the first new restaurant 'Anson Pizzeria' sells pizzas 'Anson Meat Pizza' and 'Anson Vegan Pizza', the second restaurant 'Sepideh Pizzeria' sells 'Sepideh Meat Pizza' and 'Sepideh Vegan Pizza'. These pizzas all have their own specific ingredients.

Subtask RDF.1: URI Generation
The function *createURIForEntity* is defined to create fresh URI. This is called in the 2 type triple creation functions of **Subtask RDF.2: RDF Generation**, which are *mappingToCreateTypeTriple* and *mappingToCreateTypeTripleIndividually*. In other words, URI is generated when type triple is created.

Subtask RDF.2: RDF Generation
There are 6 functions defined to create RDF triples. For each of type, literal, and object triples, there are two function variants. While one function variant creates triples for all entries of entire relevant columns at a single time, another function variant creates one triple for specific entries row by row. An overview table of names of these function variant pairs is as follow.

Type of Triple	Function with 'Entire Columns' Approach	Function with 'Row by Row' Approach
Type Triple	*mappingToCreateTypeTriple*	*mappingToCreateTypeTripleIndividually*
Literal Triple	*mappingToCreateLiteralTriple*	*mappingToCreateLiteralTripleIndividually*
Object Triple	*mappingToCreateObjectTriple*	*mappingToCreateObjectTripleIndividually*

For the 3 types of triples, while the functions with 'entire column' approach are default, those with 'row by row approach' are useful when there are various types of entities on columns, for instance, multiple restaurant categories in each cell of 'categories' column, and multiple ingredients in each cell of 'item description' column.

Subtask RDF.3: URIs from State-of-the-art Knowledge Graph

Subtask RDF.3.1: Use Google KG
URIs from Google KG are extracted but not saved.

Subtask RDF.3.2: Use Wikidata
URIs from Wikidata KG are extracted but not saved.

Subtask RDF.4: Reasoning with Ontology and Generated RDF Data
Reasoning is performed with the provided ontology and generated RDF data using the defined function *performReasoning*. In light of efficiency, OWL2 RL Semantics is used to perform reasoning after loading the 2 graphs. Eventually, through expanding the ontology with generated RDF data can the extended graph be produced.

B. SPARQL and Reasoning (Task SPARQL)

Subtask SPARQL.1

We used a SPARQL query with two triple patterns to retrieve restaurant names and their corresponding cities. The query filters results to include only those restaurants located in Los Angeles by matching the city keyword "los_angeles." The retrieved data is then written to a CSV file.

Content of query_1.txt is as follows.

```
SELECT ?rest_name ?city WHERE
{
   ?rest_name rdf:type cw:PizzaPlace .
   ?rest_name cw:locatedInCity ?city .
   FILTER (STR(?city) = "http://www.semanticweb.org/city/in3067-inm713/2024/restaurants#los_angeles")
   }
ORDER BY ?city
```

Subtask SPARQL.2

We are enhancing the query by adding another triple pattern and using the CONCAT function to combine menu_item and item_value into a new variable called combined_info. The query is further filtered to include only menu items that have the ingredient "pepperoni." Finally, we use the ORDER BY clause to sort the results based on pizzas.

Content of query_2.txt is as follows.

```
SELECT ?menu_item ?item_value ?ingredient (CONCAT(STR(?menu_item), " - ", STR(?item_value)) AS ?combined_info) WHERE
{
   ?menu_item rdf:type cw:MenuItem .
   ?menu_item cw:hasValue ?item_value .
   ?menu_item cw:hasIngredient ?ingredient .
   FILTER (STR(?ingredient) = "http://www.semanticweb.org/city/in3067-inm713/2024/restaurants#pepperoni")
   }
ORDER BY ?pizza
```

Subtask SPARQL.3

In this task, we utilize the UNION operator to combine Italian restaurants with American restaurants, and then apply the MINUS operator to exclude pizza places from the combined results.

Content of query_3.txt is as follows.

```
SELECT ?rest_name ?rest_cat WHERE {
   {  ?rest_name rdf:type cw:ItalianRestaurant  }
   UNION
   {  ?rest_name rdf:type cw:AmericanRestaurant  }
   MINUS
   {  ?rest_name rdf:type cw:PizzaPlace  }
   }
```

Subtask SPARQL.4
In this task, we group restaurants by city names and use the COUNT function as an aggregation to determine the number of restaurants in each city. We then filter the results to include only those cities that have more than one restaurant.

Content of query_4.txt is as follows.

```
SELECT ?city (COUNT(?rest_name) AS ?num_pizza_places) WHERE
{
   ?rest_name rdf:type cw:Restaurant .
   ?rest_name cw:locatedInCity ?city .
   }
GROUP BY ?city
HAVING (COUNT(?rest_name) > 1)
```

Subtask SPARQL.5
For this task, we modified the query to focus on the state instead of the city, allowing us to query a different column. Additionally, we applied an ORDER BY clause to sort the results based on both the state and the number of menu items.

Content of query_5.txt is as follows.

```
SELECT ?state (COUNT(?rest_name) AS ?num_pizza_places) WHERE
{
   ?rest_name rdf:type cw:Restaurant .
   ?rest_name cw:locatedInState ?state .
   }
GROUP BY ?state
HAVING (COUNT(?rest_name) > 1)
ORDER BY ?state ?num_menu_items
```

C. Ontology Alignment (Task OA)

Subtask OA.1
For the first task of ontology alignment, we created an array of equivalent_classes and defined pairs of classes that are similar across the two ontologies. For instance, the PizzaTopping class in the pizza ontology is equivalent to the Ingredient class in the cw ontology. Similarly, we defined an array for equivalent_properties, such as mapping hasTopping in the pizza ontology to hasIngredient in the cw ontology. Using OWL.equivalentClass and OWL.equivalentProperty as predicates, we added these equivalences to a newly created graph and serialized it.

Subtask OA.2
For the second task, to compute the precision and recall of our mappings, we used sets to extract the triples from both our generated equivalences graph and the reference mappings graph. We then calculated True Positives (TP), False Positives (FP), and False Negatives (FN), and computed precision and recall using methods provided in the

Subtask OA.3
For the third task, we performed reasoning by merging all the graphs into one and applying the DeductiveClosure method to infer additional triples.

Subtask OA.4

For the final task, we wrote a SPARQL query using the vocabulary of pizza.owl to retrieve city names. The results were then saved in a CSV file for further analysis.

```
PREFIX pizza: <http://www.co-ode.org/ontologies/pizza/pizza.owl#>
PREFIX cw: <http://example.org/cw_onto#>

SELECT ?restaurant ?pizza ?price ?currency
WHERE {
  ?restaurant a pizza:Pizzeria ;
          cw:serves ?pizza .

  ?pizza a pizza:Pizza ;
       pizza:hasTopping ?topping ;
       cw:hasValue ?price ;
       cw:currency ?currency .
}
```

D. Ontology Embeddings (Task Vector)

Subtask Vector.1

A merged OWL graph is created by parsing separate graphs of the dataset and generated data of **Tabular Data to Knowledge Graph (Task RDF)**. 2 models with different configurations are created and saved in binary (Gensim) and textual (Txt) formats respectively. This means that in total 4 files are generated in the folder 'Task Vector Result'. The main difference between the 2 configurations is the Walker parameter, given one is random walk and another one is Weisfeiler-Lehman (wl) subtree kernel. Other configuration settings remain constant.

Subtask Vector.2

6 pairs of entities are selected, with 3 pairs expected to be closely similar and 3 other pairs expected to be dissimilar. While the similar pairs are pairs of pizza place restaurants, the dissimilar pairs are pairs of pizza place restaurant and unpopular pizza ingredient.

Subtask Vector.2.1

For configuration 1 (random walk), similarities of the similar pairs are 0.75646794, 0.91293836, and 0.794361 respectively, while similarities of the dissimilar pairs are 0.4776516, 0.6426604, 0.5272115.

Subtask Vector.2.2

For configuration 2 (Weisfeiler-Lehman (wl) subtree kernel), similarities of the similar pairs are 0.7573579, 0.90975535, 0.764236 respectively, while similarities of the dissimilar pairs are 0.52856016, 0.7192131, 0.5640587.

For the similar pairs, configuration 1 (random walk) results in higher similarities than configuration 2 (Weisfeiler-Lehman (wl) subtree kernel). For the dissimilar pairs, configuration 1 (random walk) results in lower similarities than configuration 2 (Weisfeiler-Lehman (wl) subtree kernel). This implies that configuration 1 (random walk) produces more extreme similarities compared to configuration 2 (Weisfeiler-Lehman (wl) subtree kernel).

E. Reference

Submitted code is with reference to approaches in labs.