deti universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Semantic Web

Knowledge Representation

Using Semantic Data

TripleStore

Implementation and Use

Triplestore

- A Triplestore is used to save, manage and search for triples of data.
- There are currently several triplestores on the market but requires immediate mastery of the various concepts and standards that make up the semantic web.
- So, at the beginning, we will use a simple python implementation of a triplestore, giving the possibility:
  - To start immediately the use of knowledge graphs;
  - And to have the opportunity to observe the triplestore construction “from the inside”.

Triplestore - Implementation

- This implementation, in python, is based on indexes, performing cross-indexing of the three terms of the triple, allowing direct access to the triples through any of the terms.

```
class SimpleGraph:
    def __init__(self):
        self._spo = {}  # subject - predicate - object
        self._pos = {}  # predicate - object - subject
        self._osp = {}  # object - subject - predicate
```

- Each index is fact a dictionary of dictionaries, containing sets.

```
self._spo = {subject:{predicate:set([object])}}
```

Triplestore – Adding

- add() method
  - Add all permutations of the triple's terms to all indexes.

```
def add(self, sub, pred, obj):
    self._addToIndex(self._spo, sub, pred, obj)
    ...
```

- _addToIndex() method
  - Add all the triple's terms to the index if they are not already present.

```
def _addToIndex(self, index, a, b, c):
    if a not in index:
        index[a] = {b:set([c])}
    ...
```

Triplestore – Removing

- remove() method
  - Remove the triple from all indexes.

```
def remove(self, sub, pred, obj):
    triples = list(self.triples(sub, pred, obj))
    for (delSub, delPred, delObj) in triples:
        self._removeFromIndex(self._spo, delSub, delPred, 
delObj)
        ...
```

- _removeFromIndex() method
  - Iterate through the index and clean it when removing the triple.

```
def_removeFromIndex(self, index, a, b, c):
    try:
        bs = index[a]
        cset = bs[b]
        cset.remove(c)
        ...
```

Triplestore – Loading and Saving

- load() method
  - Loads triples from a CSV file and add them to the triplestore.

```
def load(self, filename):
    f = open(filename, "r", newline="", encoding="utf-8")
    reader = csv.reader(f)
    for sub, pred, obj in reader:
        self.add(sub, pred, obj)
    f.close()
```

- save() method
  - Saves all triples to a CSV file.

```
def save(self, filename):
    f = open(filename, "w", newline="", encoding="utf-8")
    writer = csv.writer(f)
    for sub, pred, obj in self.triples(None, None, None):
        writer.writerow([sub, pred, obj])
    f.close()
```

Triplestore – Graphs Merging

- mergeFromFile() method
  - Loads a graph from a file and merges it with an existing specific graph.

```
def mergeFromFile(graph):
    tmp = SimpleGraph()
    tmp.load(input("Nome do ficheiro: "))
    for sub, pred, obj in tmp.triples(None, None, None):
        graph.add(sub, pred, obj)
```

Triplestore – Filtering

- triples() method
  - Filters existing triples based on a given pattern triple.
  - Arguments with None value, mean anything – “*”

```
def triples(self, sub, pred, obj):
    try:
        if sub != None:
            if pred != None:
                # sub pred obj
                if obj != None:
                    if obj in self._spo[sub][pred]:
                        yield (sub, pred, obj)
                # sub pred None
                else:
                    for retObj in self._spo[sub][pred]:
                        yield (sub, pred, retObj)
            ...
```

Filtering – Usage

- Having a knowledge graph about movies, it’s possible to ask for all triples related with the direction of movies.

```
graph.triples(None, "directed_by", None)
```

- This gives all triples with predicate “directed_by”.

Triplestore – Querying

- query() method
  - Executes a query to the graph, using a set of query triples as its criteria.

```
def query(self, clauses):
    bindings = None
    for clause in clauses:
        bpos = {}
        qc = []
        ...
```

  - This method is fully commented to explain what each line does.

Querying – Usage

- query() method uses variables instead of value None to ask specific information.
- The responses are not triples, but values of that variables.

```
graph.query([“ridley_scott”, “directed_by”, ?movie])
```

- This gives all movies directed by entity with ID equal to ”ridley_scott”.

Querying

- Having a knowledge graph about organizations and their financial transactions, it’s possible to ask:
    - Which banks from Lisbon made donations to Mr. Josh and the respective amounts.

```
graph.query([

             (‘?organization’, ‘headquarters', ‘Lisbon'),
             (‘?organization’, ‘sector’, ‘Banking Investment’),
             (‘?Organization’, ‘offer’, ‘?donation’),
             (‘?donation’, ‘recipient’, ‘Josh’),
             ('?donation ‘, ‘amount’, '?euros’)

            ])
```

Another Query

- Having a knowledge graph about famous people, from social networks, it’s possible to ask:
  - People who started a relationship in the year they ended a relationship with Britney Spears.

```
graph.query([
             (‘?rel1’, ‘with’,  ‘Britney Spears’)
             (‘?rel1’, ‘with’,  ‘?person’),
             (‘?rel1’, ‘end’,   ‘?year’),
             (‘?rel2’, ‘with’,  ‘?person’),
             (‘?rel2’, ‘start’, ‘?year’)
            ])
```