RDF modellezés, HF2 dokumentáció

Szokoly-Angyal Armand

Cél és megoldás összefoglalása

Ezen házifeladat célja egy természetesnyelv-feldolgozó alkalmazás készítése, mely tetszőleges szövegből RDF hármasokat állít elő automatikusan, majd ezeket RDF4j adatbázisba tölti fel. A generált adatbázishoz lekérdezések fogalmazhatóak meg egy szöveges, illetve egy webes interfészen keresztül (ez utóbbin intuitívan, SparQL ismerete nélkül).

Az alkalmazás angol nyelvű, és a kódbázisban a kommentek is azok, de képes ékezetes karaktereket is kezelni, és az RDF hármasok feldolgozása magyar nyelvre van optimalizálva, azon belül is inkább jogi szövegre, de tetszőleges szövegből képes értelmezhető hármasokat kinyerni.

Megoldás áttekintése

Először a szöveg feldolgozásait fogom részletezni, ide beleértve a szöveg előfeldolgozását NLP pipeline-nal, regex-szel, majd a .rdf fájl létrehozását megvalósító python modulomat foglalom össze.

Ezután a megírt SparQL lekérdezéseket dokumentálom, valamint a példaszövegből kinyert futtatási eredményeket.

Ezután bemutatom az integrált megoldásom mind CLI, mind webes felületét.

A létrehozott adatbázis az alkalmazás gyökérkönyvtárában a hf2_database.rdf fájl.

Futtatási környezet követelményei

Az alkalmazás futtatásához szükséges Python virtuális környezet követelményei a requirements.txt fájlban találhatóak.

A feldolgozás lépései

A triple_generator.py fájl szemantikai hármasokat (alany, állítmány, tárgy) von ki egy adott szövegfájlból természetes nyelvfeldolgozó (NLP) módszerekkel. A folyamat a következő lépésekből áll:

Szöveg előfeldolgozása:

A preprocess_text függvény eltávolítja a szövegből az olyan mintákat, mint pl. az alcímkék (pl. a), (1)), csillagok és idézőjelek.

Továbbá normalizálja a szóközöket, több egymást követő szóközt egyetlen szóközzé alakítva.

Hármasok kinyerése:

A generate_triples függvény betölti a szöveget és a huspacy NLP modellt alkalmazza annak feldolgozására.

Az extract_triples függvény a tokenek függőségi viszonyai alapján az alanyokat, állítmányokat és tárgyakat azonosítja:

Alanyok: az nsubj függőség alapján kerülnek felismerésre, a megfelelő módosítókkal együtt.

Állítmányok: az alany head-jéből kerülnek kinyerésre.

Tárgyak: az állítmány függőségeiből, elsősorban a közvetlen tárgyak és attribútumok alapján.

Szűrés és utófeldolgozás:

A stop-szavak és nem-alfanumerikus tokenek figyelmen kívül maradnak a kinyerés során.

A kinyert hármasok közül csak az érvényesek (alany, állítmány, tárgy mind jelen van) maradnak, és az ismétlődő hármasok eltávolításra kerülnek.

Végső kimenet:

Az eredmény egy egyedi hármasok halmaza, amely további elemzésre vagy felhasználásra kész.

RDF4j adatbázishoz az XML fájl előállítása

Az rdf_xml.py fájl Python hármasokat (alany, állítmány, tárgy) RDF/XML formátumba konvertálja és elmenti egy fájlba. Az rdflib könyvtárat használja az RDF gráf objektum létrehozására és kezelésére. A folyamat a következő lépésekből áll:

Hármasok feldolgozása: Minden egyes hármas esetén a script biztosítja, hogy a speciális karakterek megfelelően legyenek kezelve az alany URI-jának kódolása során. Az alanyt egy előre meghatározott névtér segítségével URI-ra alakítja, és az állítmányt ugyanazzal a névtérrel állítja be.

Tárgy kezelése: A tárgy típusától függően kerül feldolgozásra. Ha numerikus, akkor literálként kerül tárolásra; ha URL, akkor URI-ként; ha más típusú, akkor literálként, magyar nyelvű kódolással.

RDF gráf létrehozása: A script minden hármas adatot hozzáad az RDF gráfhoz.

Szerializálás: A gráfot RDF/XML formátumban szerializálja, és a fájlt **UTF-8** kódolással menti el (fontos).

Naplózás: A sikeres fájlmentést követően naplóüzenet tájékoztat a fájl mentésének helyéről.

A triples_to_rdf függvény visszaadja az RDF gráf objektumot, és elmenti az RDF adatokat egy megadott kimeneti fájlba (alapértelmezetten output.rdf, de ez az rdfApp.config fájlon keresztül megváltoztatható).

RDF4j adatbázisban példa SparQL lekérdezésekre

A megoldáshoz Eclipse RDF4J-t (eclipse-rdf4j-5.0.3) használtam fel, a szervert Jetty 9.4 (jetty-distribution-9.4.56.v20240826) segítségével futtattam.

A RDF4j Workbench segítségével:

A repository létrehozása után a fentiek szerint exportált XML fájlt feltöltöttem a WorkBench felületére, majd példa lekérdezéseket futtattam:

Az összes elem lekérdezése

```
SELECT ?s ?p ?o

WHERE {

  ?s ?p ?o .

}
```

Eredmény:

S	P	O
ex:f%C3%A9nysoromp%C3%B3	ex:biztosít	"átjáró"@hu
ex:jelz%C5%91%C5%91r	ex:biztosít	"áthaladás"@hu
ex:t%C3%A1bla	ex:jelez	"pálya áthaladás"@hu
ex:t%C3%A1bla	ex:jelez	"átjáró"@hu
ex:t%C3%A1bla	ex:jelez	"várakozóhely"@hu
ex:t%C3%A1bla	ex:jelez	"fajta"@hu
ex:t%C3%A1bla	ex:jelez	"fajta átjáró"@hu
ex:t%C3%A1bla	ex:van	"időszak várakozás"@hu
ex:t%C3%A1bla	ex:jelezhet	"várakozóhely"@hu
ex:t%C3%A1bla	ex:tüntet	"fajta"@hu
ex:parkom%C3%A9ter%20m%C5%B1k%C3%B6dtet%C3%A9s	ex:kötelező	"várakozás"@hu
ex:jelz%C5%91t%C3%A1bla	ex:jelez	"mód veszély várakozóhely átjáró"@hu
ex:%C3%B3ra	ex:van	"időszak várakozás"@hu

Predikátum szerinti szűrés

```
SELECT ?s ?o

WHERE {

 ?s <http://hf2.org/jelez> ?o .

}
```

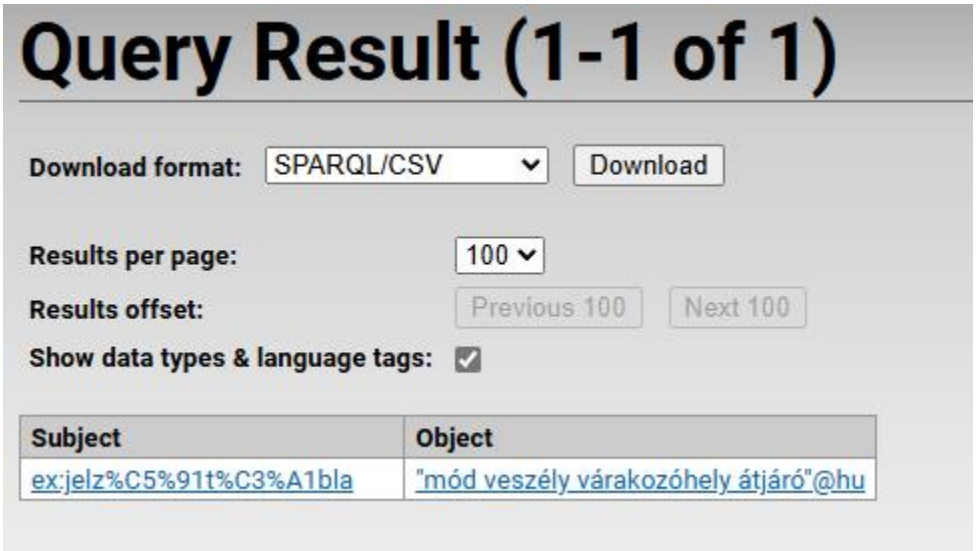
Eredmény:

S	O
ex:t%C3%A1bla	"pálya áthaladás"@hu
ex:t%C3%A1bla	"átjáró"@hu
ex:t%C3%A1bla	"várakozóhely"@hu
ex:t%C3%A1bla	"fajta"@hu
ex:t%C3%A1bla	"fajta átjáró"@hu
ex:jelz%C5%91t%C3%A1bla	"mód veszély várakozóhely átjáró"@hu

Veszélyt jelző táblák lekérdezése

```
SELECT ?subject ?object

WHERE {

 ?subject ?predicate ?object .

 FILTER(CONTAINS(STR(?object), "veszély"))

}
```

Eredmény:



Integrált CLI megoldás

A fent implementált kódot egy CLI parancssori eszközbe integráltam, melyet az rdfApp.config fájlon keresztül lehet konfigurálni a mellékelt alkalmazásban. A fájl szintaktikája az alábbiakként fest:

[Files]

input_file = input_text.txt

output_file = hf2.rdf

Ahol az input fájl a parsolandó szöveget tartalmazza raw txt formátumban, az output fájl pedig az rdf xml fájl helyét specifikálja (ha a felhasználó azt külön manuálisan is fel szeretné használni).

A példa szöveg az applikáció gyökérkönyvtárában az input_text.txt fájlban található. Ebbe tetszőlegesen más szöveget is elhelyezhetünk.

A cli.py-t elindítva az alkalmazás már el is készíti az rdf hármasokat, azokat a konzolra is kinyomtatja. Az output fájlt is automatikusan elmenti. Ezután a „>>>” jel után tudunk SparQL parancsokat kiadni, melyekre az outputot formázás nélkül kiírja az alkalmazás. Példa futás közben:

```
Generated Triples: [('tábla', 'tüntet', 'fajta'), ('parkométer működtetés', 'kötelező', 'várakozás'), ('fénysorompó', 'biztosít', 'átjáró'), ('tábla', 'jelez', '
álya áthaladás'), ('jelzőőr', 'biztosít', 'áthaladás'), ('tábla', 'jelez', 'átjáró fajta'), ('tábla', 'van', 'várakozás időszak'), ('tábla', 'jelez', 'fajta'), (
jelzőtábla', 'jelez', 'mód átjáró veszély várakozóhely'), ('tábla', 'jelezhet', 'várakozóhely'), ('tábla', 'jelez', 'várakozóhely'), ('óra', 'van', 'várakozás id
szak'), ('tábla', 'jelez', 'átjáró')]
RDF/XML file saved to: hf2.rdf

Enter SPARQL queries below (type 'exit' to quit):
>>> SELECT ?s ?o
>>> WHERE {
>>>   ?s <http://hf2.org/jelez> ?o .
>>> }
(rdflib.term.URIRef('http://hf2.org/t%C3%A1bla'), rdflib.term.Literal('pálya áthaladás', lang='hu'))
(rdflib.term.URIRef('http://hf2.org/t%C3%A1bla'), rdflib.term.Literal('átjáró fajta', lang='hu'))
(rdflib.term.URIRef('http://hf2.org/t%C3%A1bla'), rdflib.term.Literal('fajta', lang='hu'))
(rdflib.term.URIRef('http://hf2.org/jelz%C5%91t%C3%A1bla'), rdflib.term.Literal('mód átjáró veszély várakozóhely', lang='hu'))
(rdflib.term.URIRef('http://hf2.org/t%C3%A1bla'), rdflib.term.Literal('várakozóhely', lang='hu'))
(rdflib.term.URIRef('http://hf2.org/t%C3%A1bla'), rdflib.term.Literal('átjáró', lang='hu'))
>>>
```

GUI

A GUI megoldásához a már létező függvényeket csatoltam egy új, view réteghez az alkalmazásban, amely az app.py-ban található. A webes interfészhez a python „Flash” könyvtárát használtam fel, ehhez a html template a ./templates/query.html fájlban található.

Az alkalmazás (app.py) elindítása után a konzol kiírja, hogy milyen porton fut a webalkalmazás. (Alapértelmezetten 5000.)

```
INFO:root:Generated Triples: [('tábla', 'jelez', 'áthaladás pálya'), ('tábla', 'jelezhet', 'várakozóhely'), ('tábla', 'van', 'időszak várakozás'),
, ('tábla', 'jelez', 'várakozóhely'), ('óra', 'van', 'időszak várakozás'), ('fénysorompó', 'biztosít', 'átjáró'), ('parkométer működtetés', 'kötel
ó veszély'), ('tábla', 'jelez', 'átjáró fajta'), ('tábla', 'jelez', 'átjáró'), ('tábla', 'jelez', 'fajta')]
INFO:root:RDF/XML file saved to: hf2.rdf
 * Serving Flask app 'app'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
```

A megfelelő URL begépelése után a böngészőben a következő felületet láthatjuk:

RDF Model Query Interface

Subject:

Predicate:

Object:

Query

Results:

s	p	o
tábla	jelez	fajta átjáró
tábla	tüntet	fajta
tábla	van	időszak várakozás
jelzőtábla	jelez	veszély mód átjáró várakozóhely
óra	van	időszak várakozás
fénysorompó	biztosít	átjáró
jelzőőr	biztosít	áthaladás
tábla	jelez	áthaladás pálya
parkométer működtetés	kötelező	várakozás
tábla	jelezhet	várakozóhely
tábla	jelez	fajta
tábla	jelez	átjáró
tábla	jelez	várakozóhely

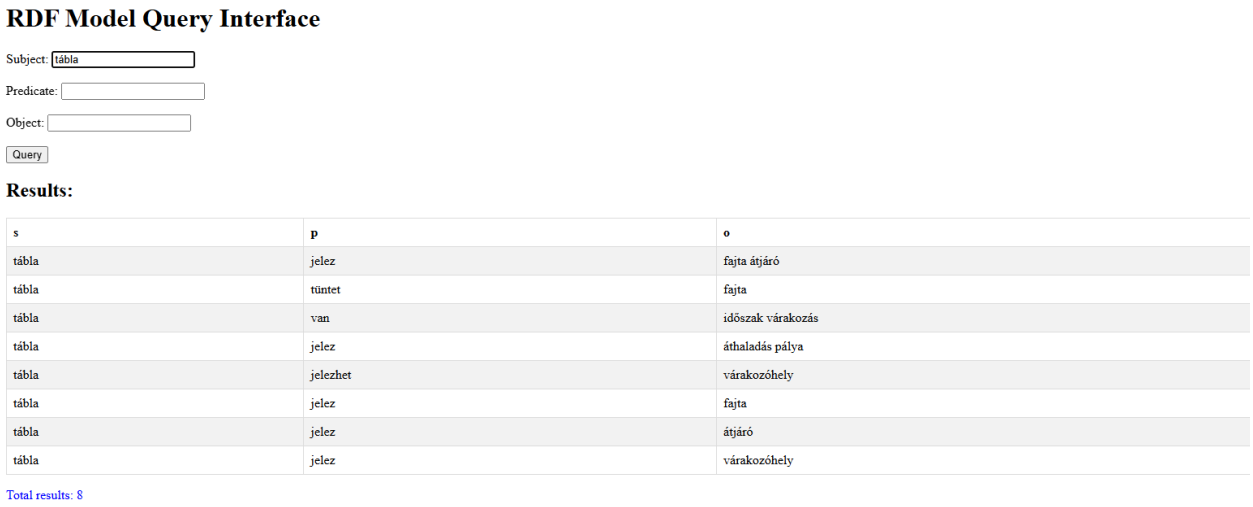
Total results: 13

Az alkalmazás alapértelmezetten (tehát ha nem írunk be semmit) minden hármast kiír. Látható, hogy az ékezetes karaktereket is jól kezeli, ez a „urllib.parse” python modul quote() és unquote() függvényének köszönhető az app.py modulban.

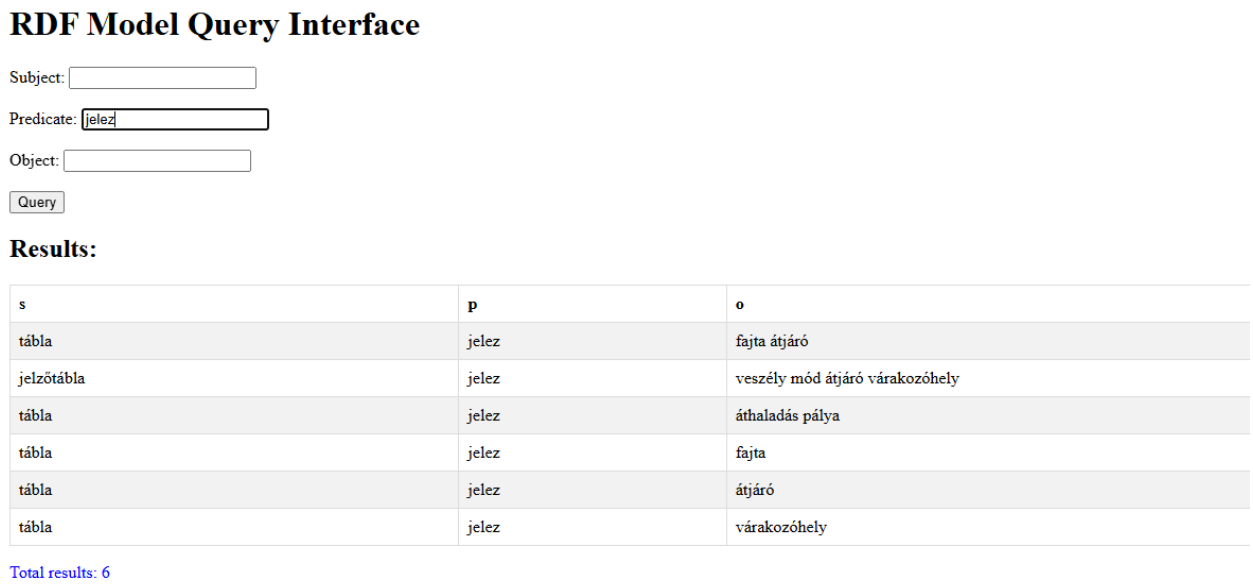
Használatra példa

A webes felület használata egyszerű, a három mező bármelyikébe írhatunk szöveget, és ha az adott szerepben talál azonos kifejezést az rdf-hármasok közt, kiírja a találatokat.

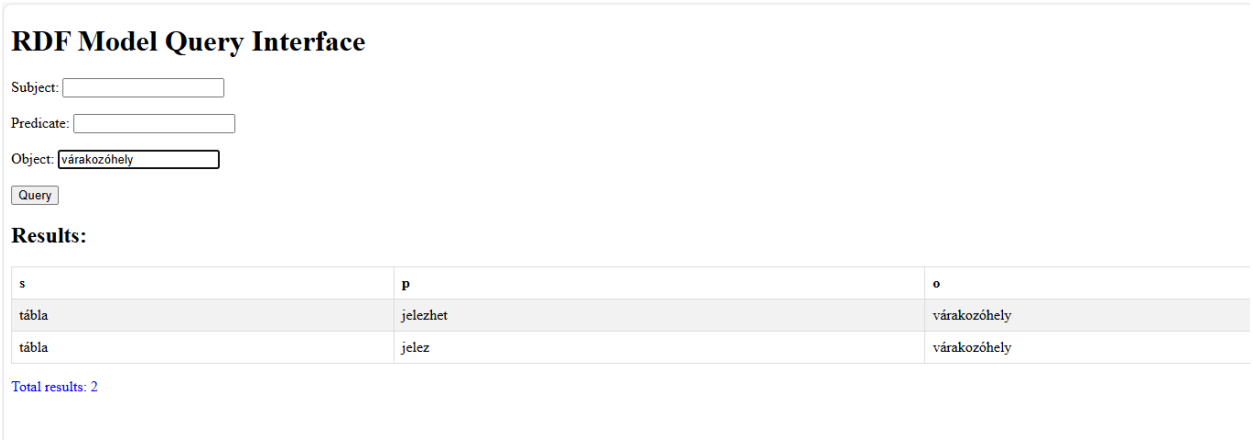
Írd ki a „tábla” alanyhoz tartozó RDF hármasokat:



Vagy írd ki, hogy a táblák (a példaszövegből kiindulva) mit jeleznek:



Minden információ a várakozóhelyekkel kapcsolatban:



Konklúzió

A házifeladat keretében összeállítottam egy olyan alkalmazást, mely integrált megoldást nyújt magyar nyelvű szövegekből RDF hármasok előállítására, illetve azok azonnali használatára automatikusan adatbázisként. Az alkalmazás lehetővé teszi a generált hármasokból történő lekérdezést SparQL segítségével, valamint a webes felületen keresztül anélkül is. Az alkalmazás grafikus felülete kényelmes lehetőséget biztosít információk kinyerésére, és a magyar nyelvben gyakran előforduló nem ASCII karakterkódolást is megfelelően kezeli mind input, mind output tekintetében.