

RELAZIONE PROGETTO DI INGEGNERIA DELLA CONOSCENZA

Anno Accademico 2021 - 2022

AUTOBIBLO PROJECT : Gestione di una biblioteca



Studenti:

Mario Franco, matricola 717768, e-mail: m.franco34@studenti.uniba.it

Antonio Curione, matricola 716131, e-mail: a.curione7@studenti.uniba.it

Rosanna Acquafredda, matricola 724972, e-mail: r.acquafredda8@studenti.uniba.it

Indice

1. [Introduzione](#)
2. [Ontologia](#)
3. [CSP](#)
4. [Ricerca sul grafo](#)
5. [Recommender system con KNN](#)
6. [Valutazione dei risultati](#)
7. [Note finali](#)

INTRODUZIONE

La seguente documentazione rappresenta la descrizione tecnica del progetto “**AUTOBIBLO project**”, realizzato dagli studenti Mario Franco, Antonio Curione e Rosanna Acquafredda, per il corso di Ingegneria della Conoscenza (A.A. 2021-2022).

Panoramica del progetto

Il progetto realizzato rappresenta un’automazione della gestione di una biblioteca scolastica, con l’aggiunta di un sistema di raccomandazione dei libri, con lo scopo di incentivare la lettura , offrendo un servizio semplice ed efficace.

Obiettivi

- Progettare un sistema sotto forma di grafo, che specifichi l’organizzazione della biblioteca, così da poterlo utilizzare nelle fasi successive
- Creare un’ontologia adeguata, che possa rappresentare al meglio il sistema gestionale della biblioteca.
 - o Di seguito, popolare tale ontologia, inserendo le istanze entità-individui, così da poter effettuare delle simulazioni
 - o dimostrare le funzionalità attraverso delle query, che ci permettono di visualizzare l’ontologia
- Modellare, utilizzando CSP (Constraint Satisfaction Problem), un sistema per gestire le prenotazioni dei libri presenti nella biblioteca e le relative restituzioni
- Definire ed implementare un algoritmo di ricerca sul grafo pesato, realizzato al primo punto, così da poter esplicitare il percorso da seguire per trovare, più agilmente, un determinato libro nell’ambiente della biblioteca
- Appropriarsi di un dataset di libri adeguato e strutturare un recommender system, utilizzando il KNN, per poter consigliare agli studenti dei libri, basandoci sulle similarità tra il libro di interesse e quelli presenti nel dataset
- Fare un resoconto del sistema realizzato, effettuando vari test e confrontando i risultati ottenuti con quelle che erano le aspettative

Specifiche di realizzazione

- Per realizzare il progetto abbiamo utilizzato come IDE PyCharm, con il linguaggio Python (versione 3.10)
- L’ontologia è stata realizzata su Protégé (versione 5.5.0)
- Altri dettagli sulle librerie utilizzate sono presenti nel file requirement.txt

Come prima cosa, abbiamo individuato una planimetria di una biblioteca, che meglio si adattasse alla nostra idea di “libreria scolastica”. Per costruire il nostro spazio di ricerca abbiamo fatto riferimento alla biblioteca della città di Roncade, prendendone la piantina dal sito comunale.

A partire da tale piantina, abbiamo individuato i nodi principali e i percorsi che li collegano.

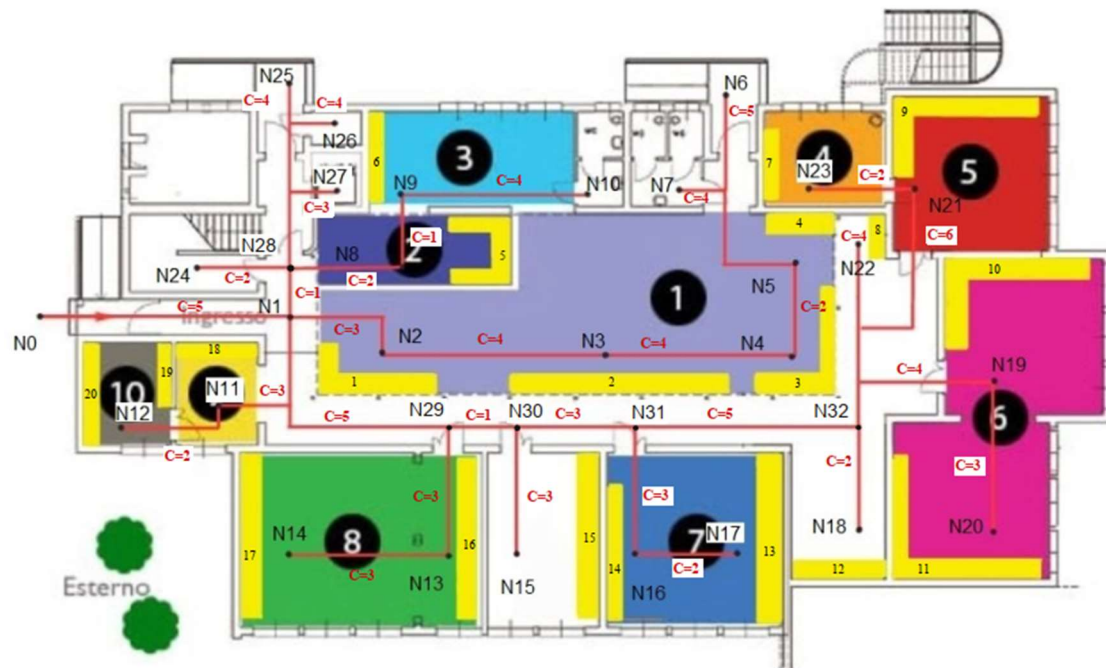


Figura 1 - Planimetria della biblioteca

I vari nodi (numerati da N0 a N32) sono collegati tra di loro da degli archi pesati, dove il peso è rappresentato dalla funzione costo, che in questo caso stima la distanza in cm, basandoci sulla piantina, tra i nodi :

$$\langle ni, nj \rangle \rightarrow \text{costo}(\langle ni, nj \rangle),$$

dove :

- **ni, nj** sono due generici nodi
- con **$\langle ni, nj \rangle$** indichiamo l'arco che li collega, con il peso equivalente alla distanza.

Nella planimetria, l'esito della funzione costo è indicato in forma compatta con C=numero; per ragioni di spazio non abbiamo inserito i costi di alcuni percorsi intermedi:

$$\text{costo}(\langle N25, N26 \rangle) = 2$$

$$\text{costo}(\langle N25, N27 \rangle) = 3$$

$$\text{costo}(\langle N26, N27 \rangle) = 3$$

$$\text{costo}(\langle N29, N11 \rangle) = 5$$

$$\text{costo}(\langle N22, N19 \rangle) = 6$$

$$\text{costo}(\langle N21, N22 \rangle) = 6$$

$$\text{costo}(\langle N21, N19 \rangle) = 8$$

$$\text{costo}(\langle N6, N7 \rangle) = 3$$

All'interno della biblioteca, i libri sono partizionati per genere e organizzati in scaffali, rappresentati in giallo ed enumerati. Oltre al numero, ad ognuno di essi abbiamo associato un nodo, il più vicino.

BOOK GENRE	LIBRARY SHELF	NODE	BOOK GENRE	LIBRARY SHELF	NODE
Novel	1	N2	Romance	11	N20
Narrative	2	N3	Finance	12	N18
Poem	3	N4	Humor	13	N17
Drama	4	N5	Horror	14	N16
Mythology	5	N8	Fable	15	N15
Fiction	6	N9	Folklore	16	N13
Scientific	7	N23	Short Story	17	N14
Math	8	N22	Poetry	18	N11
Crime	9	N21	Fairy Tale	19	N12
Computer Science	10	N19	Historical	20	N12

A partire da tale spazio, ossia dalla planimetria in figura 1, abbiamo creato un grafo equivalente, dove il nodo radice corrisponde a N0 (che nella figura rappresenta l'ingresso all'edificio), tutti gli archi, che collegano gli altri nodi, hanno associato il peso corrispondente e rappresentano tutti i vari percorsi individuati.

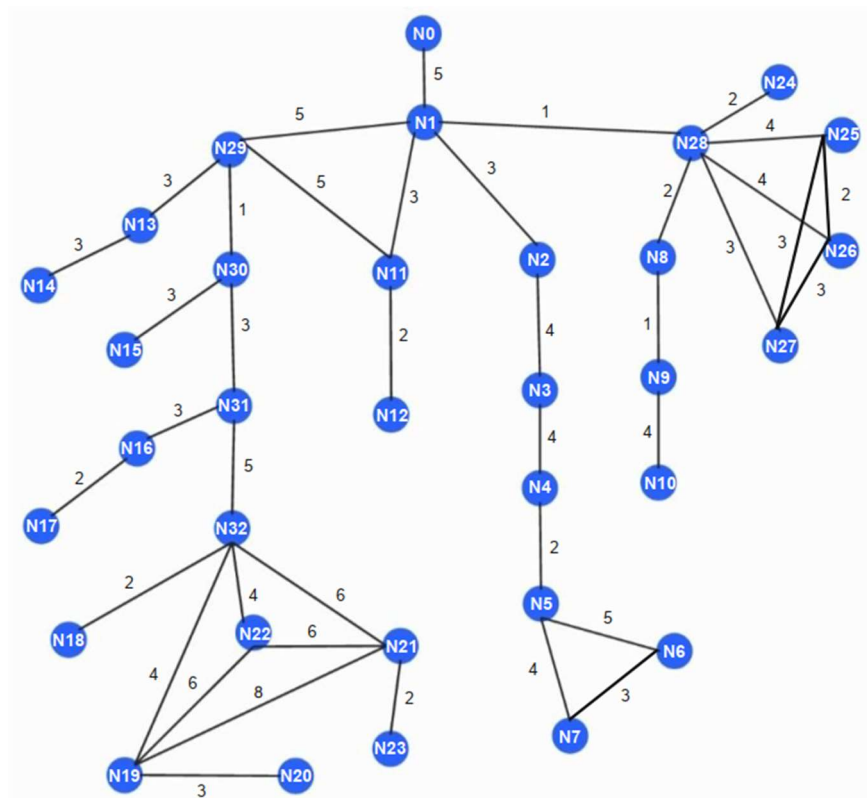


Figura 2– Grafo pesato corrispondente piantina in figura 1

Per adattarci meglio ad un'ipotetica situazione reale, abbiamo stabilito degli orari di apertura e chiusura della biblioteca. Gli orari rappresentano un vincolo, che lo studente (utente medio che potrebbe utilizzare il nostro sistema), deve tener presente per potervi accedere e prendere in prestito e/o restituire un qualsiasi libro.

	Lunedì	Martedì	Mercoledì	Giovedì	Venerdì	Sabato	Domenica
ore 9	aperto	aperto	aperto	aperto	aperto	aperto	chiuso
ore 10	aperto	aperto	aperto	aperto	aperto	aperto	chiuso
ore 11	aperto	aperto	aperto	aperto	aperto	aperto	chiuso
ore 12	aperto	aperto	aperto	aperto	aperto	aperto	chiuso
ore 13	chiuso	chiuso	chiuso	chiuso	chiuso	chiuso	chiuso
ore 14	chiuso	chiuso	chiuso	chiuso	chiuso	chiuso	chiuso
ore 15	chiuso	chiuso	chiuso	chiuso	chiuso	chiuso	chiuso
ore 16	aperto	aperto	aperto	aperto	aperto	chiuso	chiuso
ore 17	aperto	aperto	aperto	aperto	aperto	chiuso	chiuso
ore 18	aperto	aperto	aperto	aperto	aperto	chiuso	chiuso
ore 19	aperto	aperto	aperto	aperto	aperto	chiuso	chiuso

Figura 3 – Tabella riassuntiva degli orari di apertura della biblioteca

In più, per effettuare delle simulazioni del sistema, in particolare per la prenotazione di un libro e la ricerca, si assume che la biblioteca contenga un catalogo di 28 libri.

Dopo queste fasi preliminari, utili per iniziare a modellare la realtà presa in considerazione, siamo passati a progettare un'ontologia adatta.

ONTOLOGIA

Una ontologia specifica il significato dei simboli utilizzati dal sistema. Essa si modella decomponendo la realtà, da rappresentare, individuandone individui e relazioni che intercorrono tra di essi. Le ontologie vengono impiegate, nei sistemi intelligenti, per ovviare alle problematiche che si incontrano durante l'acquisizione di conoscenza (per rappresentare una KB), in quanto, documentando le associazioni simboli -> concetto, riducono le ambiguità di interpretazione (dato che lo stesso simbolo, in base al dominio del problema, può assumere significati diversi) e favoriscono l'interoperabilità semantica e sintattica della conoscenza, ossia la capacità di diverse KB di collaborare tra loro, senza conflitti, rispettando il significato dei simboli.

Le ontologie distribuite sulla rete vengono descritte attraverso il linguaggio OWL (che sta per Web Ontology Language); esso permette di descrivere il mondo in termini di :

- Individui: entità del mondo
- Classi: insieme di individui, accomunati da delle caratteristiche comuni
- Proprietà: relazioni che associano agli elementi del proprio dominio (individui), un elemento del codominio (valori che può assumere). In particolare, si dividono in:
 - o Datatype property : se il codominio contiene solo tipi primitivi (interi, stringhe,...)
 - o Object property: quando il codominio contiene altre classi

Descrizione dell'ontologia

Per strutturare e definire la nostra ontologia abbiamo utilizzato il software l'applicativo **Protégé**, il quale ne permette la realizzazione in maniera intuitiva e completa, con l'aggiunta di plug-in molto utili al fine progettuale.

Innanzitutto, abbiamo creato il file dell'ontologia (.owl) per poi localizzarla, assieme a tutti i suoi elementi, in modo univoco nel Semantic Web (tecnologia che garantisce l'interoperabilità delle ontologie distribuite sulla rete), utilizzando l'Ontology IRI (Internationalized Resource Identifier).

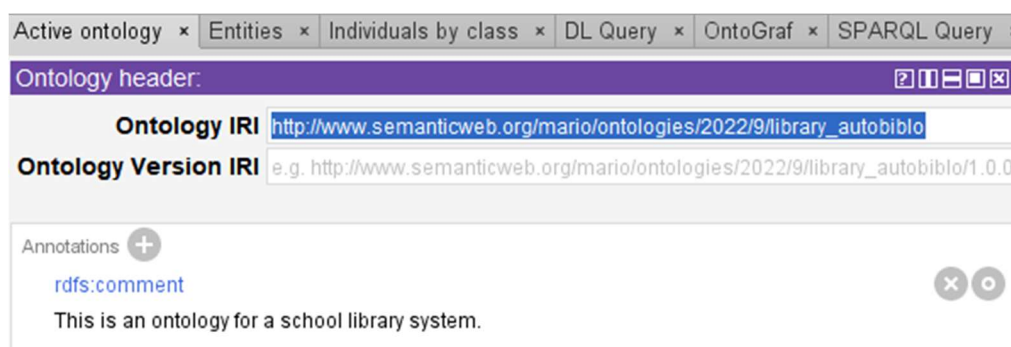


Figura 4 - IRI Ontology

Come primo step per la realizzazione dell'ontologia su Protégé, abbiamo individuato le classi, che vanno a schematizzare il nostro sistema.

In base all'importanza dei concetti che esprimono, in relazione al nostro problema, abbiamo scelto di decomporre la nostra realtà nelle classi:

“Book”, “Author”, “Genre”, “Publisher”, “Room”, “Shelf” e “Student”.

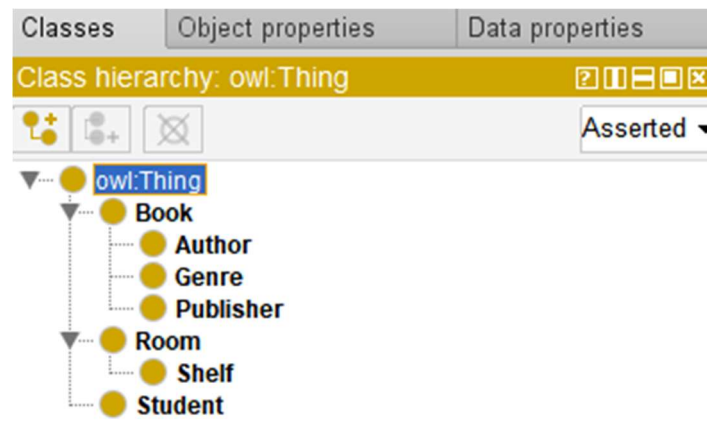


Figura 5 - Classi

Dopo di che, pensando a ciò che il nostro sistema può manipolare e visualizzare, tramite delle query, abbiamo formulato le proprietà, suddivise in datatype e object properties:

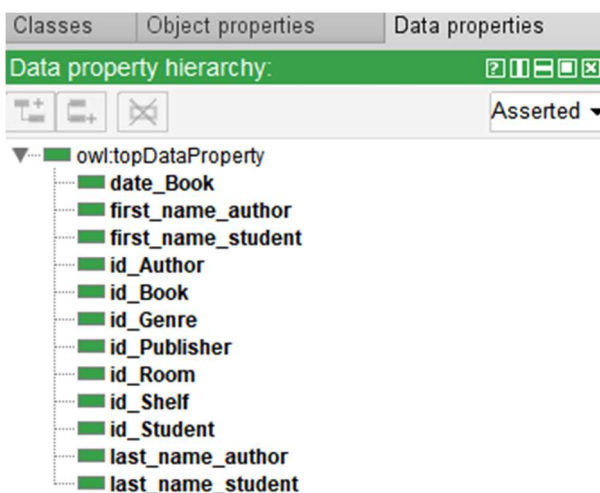


Figura 6 - Datatype properties

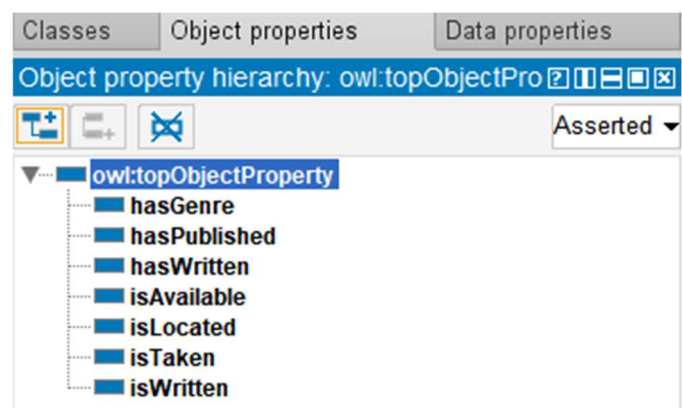


Figura 7 - Object properties

Dopo aver modellato le proprietà e, di conseguenza, le relazioni che intercorrono tra le classi/entità, prima di poter simulare il funzionamento di un sistema basato su questa ontologia, l'abbiamo popolata con le entità (individui) adatte ad ogni classe.

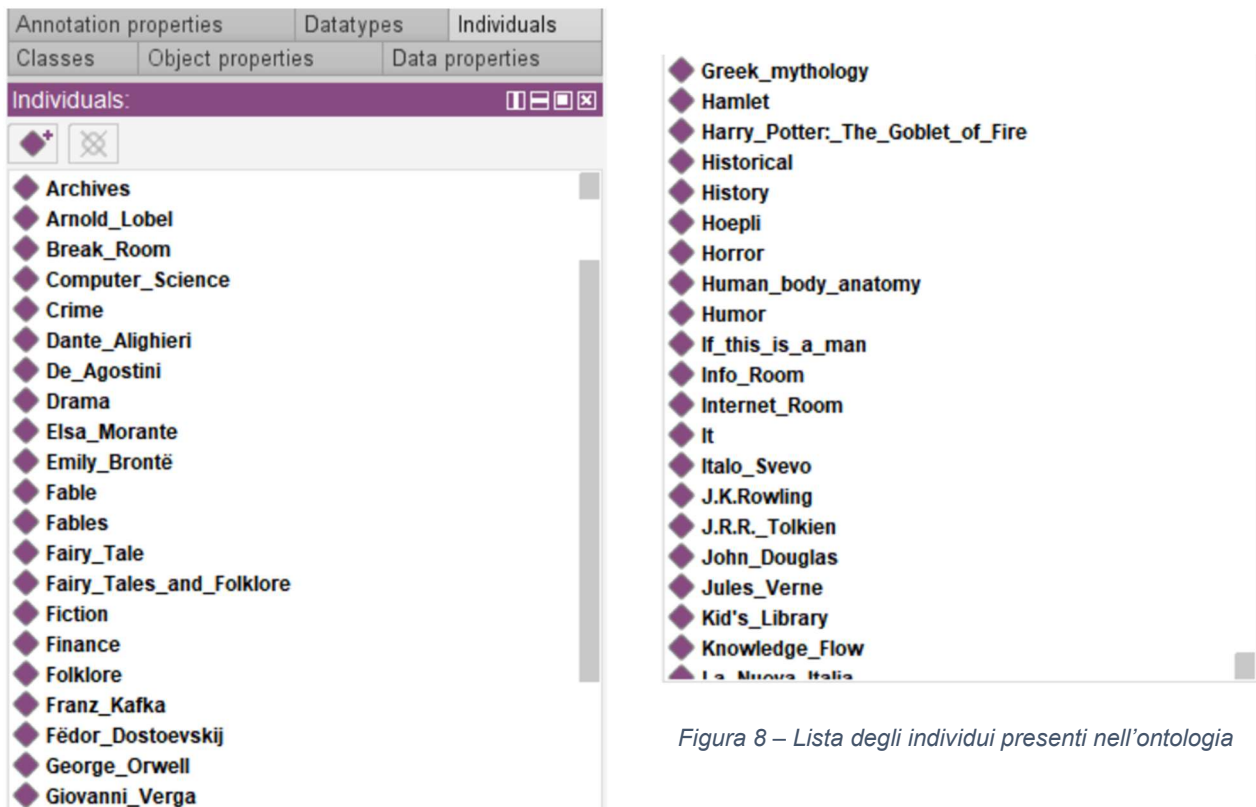


Figura 8 – Lista degli individui presenti nell'ontologia

Ogni individuo possiede delle proprietà, che eredita dalla classe di appartenenza; esempio: l'individuo *Franz Kafka*, appartenente alla classe *Author*, ha come datatype properties il proprio *nome* e *cognome*, *ID* e come object properties la relazione *hasWritten*, che lo connette ad un libro che ha scritto.

Nella figura, invece, è rappresentato un ulteriore esempio, per l'individuo *If this is a man*, che appartiene alla classe *Book*, con le proprietà descritte nell'ultima sezione.

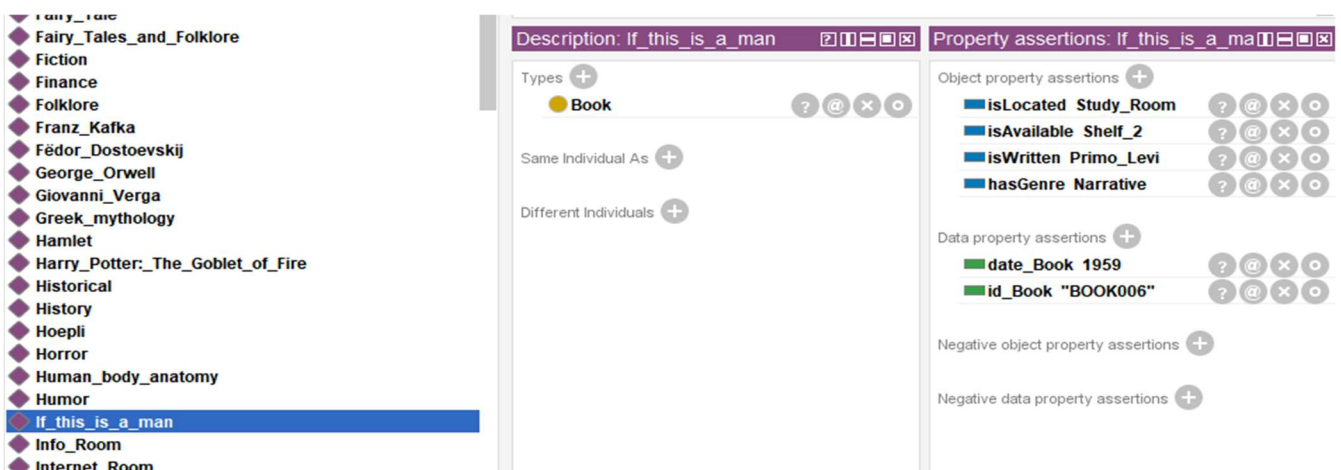


Figura 9 - Esempio "if this is a man"

L'ontologia realizzata rappresenta e decompone, correttamente, tutti i concetti utili per modellare la realtà del nostro problema.

Interrogazioni

Il nostro sistema si basa sull'ontologia appena descritta; per poter interrogarla e visualizzarne il comportamento (così da poter verificare anche il funzionamento del sistema) abbiamo utilizzato il **Reasoner** di default del software (HermiT 1.4.3.456) sfruttando i due plug-in già integrati: “**DL query**” e “**SPARQL query**” .

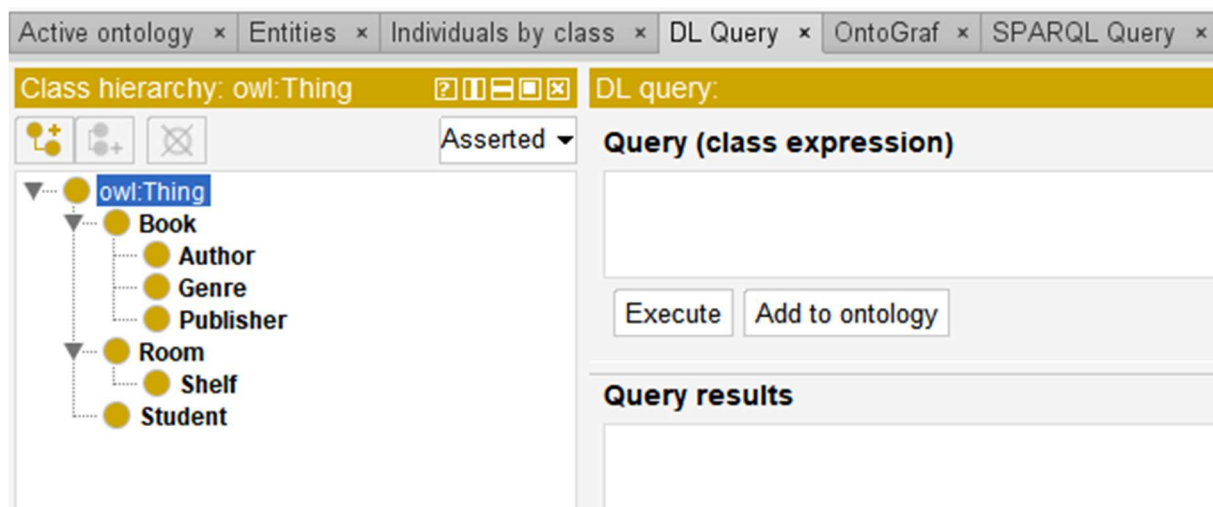


Figura 10 – Primo tool : DL query

Inizialmente, abbiamo effettuato le query tramite DL query:

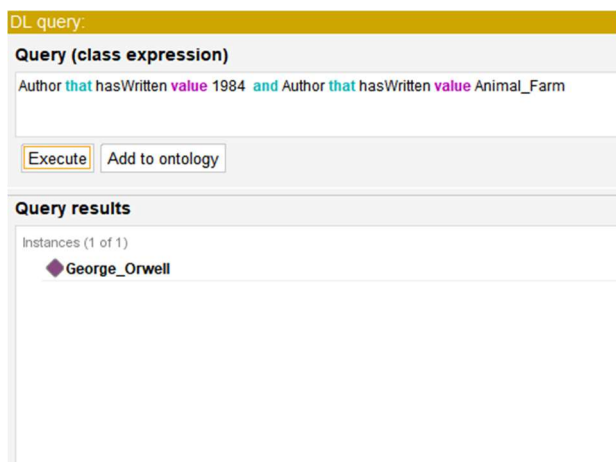


Figura 11 - query 1

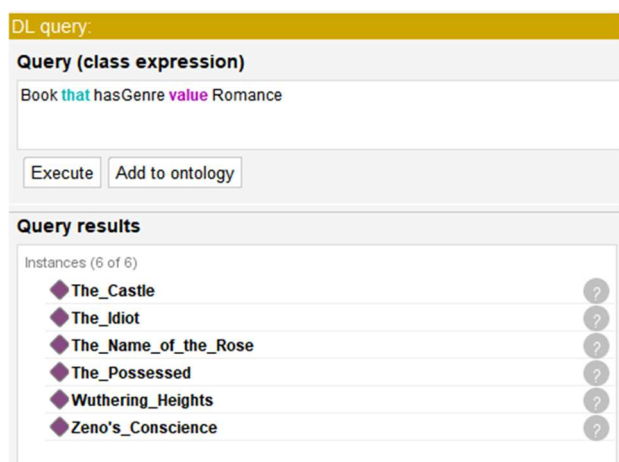


Figura 12 - query 2

DL query:

Query (class expression)

Book that isAvailable value Shelf_2

Execute Add to ontology

Query results

Instances (6 of 6)

- One_No_One_and_One_Hundred_Thousand
- 1984
- Harry_Potter:_The_Goblet_of_Fire
- History
- If_this_is_a_man
- The_Lord_of_the_Rings

Figura 13 - query 3

DL query:

Query (class expression)

Book that isLocated value Study_Room

Execute Add to ontology

Query results

Instances (10 of 10)

- One_No_One_and_One_Hundred_Thousand
- 1984
- Animal_Farm
- Hamlet
- Harry_Potter:_The_Goblet_of_Fire
- History
- If_this_is_a_man
- The_Divine_Comedy
- The_Lord_of_the_Rings
- The_Metamorphosis

Figura 14 - query 4

Poi siamo passati a SPARQL

Active ontology x Entities x Individuals by class x DL Query x OntoGraf x SPARQL Query x

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?subject ?object
WHERE { ?subject rdfs:subClassOf ?object }
```

Figura 15 – Secondo tool: SPARQL

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX table: <http://www.semanticweb.org/mario/ontologies/2022/9/library_autobiblio#>
SELECT *
WHERE {
  ?Book table:id_Book ?ID.
  ?Book table:hasGenre ?Genre.
  ?Book table:isWritten ?Author
} ORDER BY ASC(?ID)
```

Book	ID	Genre	Author
One_No_One_and_One_Hundred_Thou	"BOOK001"	Narrative	Luigi_Pirandello
1984	"BOOK002"	Narrative	George_Orwell
Animal_Farm	"BOOK003"	Novel	George_Orwell
Hamlet	"BOOK004"	Drama	William_Shakespeare
History	"BOOK005"	Narrative	Elsa_Morante
If_this_is_a_man	"BOOK006"	Narrative	Primo_Levi
The_Betrothed	"BOOK007"	Fiction	Alessandro_Manzoni
The_Castle	"BOOK008"	Romance	Franc_Kafka
The_Divine_Comedy	"BOOK009"	Poem	Dante_Alighieri
The_Idiot	"BOOK010"	Romance	Fedor_Dostoevskij
The_Lord_of_the_Rings	"BOOK011"	Narrative	J.R.R._Tolkien
The_Metamorphosis	"BOOK012"	Novel	Franc_Kafka
The_Name_of_the_Rose	"BOOK013"	Romance	Umberto_Eco
Harry_Potter:_The_Goblet_of_Fire	"BOOK014"	Narrative	J.K.Rowling
The_Possessed	"BOOK015"	Romance	Fedor_Dostoevskij
The_Trial	"BOOK016"	Novel	Franc_Kafka
Zeno's_Conscience	"BOOK017"	Romance	Italo_Svevo
Fables	"BOOK018"	Fable	Arnold_Lobel
Fairy_Tales_and_Folklore	"BOOK019"	Folklore	William_Gray
Book authors	"BOOK020"	Book authors	William_Gray

Execute

Figura 16 - SPARQL query n.1

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX table: <http://www.semanticweb.org/mario/ontologies/2022/9/library_autobiblio#>
SELECT *
WHERE {
  ?Author table:id_Author ?ID.
  ?Author table:hasWritten ?Book.
} ORDER BY ASC(?ID)
```

Author	ID	Book
Alessandro_Manzoni	"AUT002"	The_Betrothed
Primo_Levi	"AUT015"	If_this_is_a_man
Arnold_Lobel	"AUT018"	Fables
Emily_Brontë	"AUT019"	Wuthering_Heights
John_Douglas	"AUT020"	Mindhunter
Knowledge_Flow	"AUT021"	Human_body_anatomy
Marc_Loy	"AUT022"	Learning_Java
Michelle_Sandoval	"AUT023"	This_book_is_funny
Roger_Mason	"AUT024"	The_Ultimate_Finance_book
Stephen_King	"AUT025"	It
Uranus_Sagona	"AUT026"	Greek_mythology
Walter_Rudin	"AUT027"	Principles_of_Mathematical_analysis
William_Gray	"AUT028"	Fairy_Tales_and_Folklore

Figura 17 - SPARQL query n.2

SPARQL query

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX table: <http://www.semanticweb.org/mario/ontologies/2022/9/library_autobiblio#>

SELECT *
WHERE {
  ?Book table:id_Book ?ID.
  ?Book table:isAvailable ?Shelf.
  ?Book table:isLocated ?Location.
} ORDER BY ASC(?ID)

```

Book	ID	Shelf	Location
The_Iliad	"BOOK010"	Shelf_11	Library
The_Lord_of_the_Rings	"BOOK011"	Shelf_2	Study_Room
The_Metamorphosis	"BOOK012"	Shelf_1	Study_Room
The_Name_of_the_Rose	"BOOK013"	Shelf_11	Library
Harry_Potter_The_Goblet_of_Fire	"BOOK014"	Shelf_2	Study_Room
The_Possessed	"BOOK015"	Shelf_11	Library
The_Trial	"BOOK016"	Shelf_1	Library
Zeno's_Conscience	"BOOK017"	Shelf_11	Library
Fables	"BOOK018"	Shelf_15	Library
Fairy_Tales_and_Folklore	"BOOK019"	Shelf_16	Narrative_Room
Greek_mythology	"BOOK020"	Shelf_5	Info_Room
Human_body_anatomy	"BOOK021"	Shelf_7	Laboratory
It	"BOOK022"	Shelf_14	Phonoteque
Learning_Java	"BOOK023"	Shelf_10	Library
Mindhunter	"BOOK024"	Shelf_9	Kids_Library
Principles_of_Mathematical_analysis	"BOOK025"	Shelf_8	Narrative_Room
The_Ultimate_Finance_book	"BOOK026"	Shelf_12	Library
This_book_is_funny	"BOOK027"	Shelf_13	Phonoteque
Withering_Heights	"BOOK028"	Shelf_11	Library

Figura 18 - SPARQL query n.3

SPARQL query

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX table: <http://www.semanticweb.org/mario/ontologies/2022/9/library_autobiblio#>

SELECT *
WHERE {
  ?Student table:id_Student ?ID.
  ?Book table:isTaken ?Student.
} ORDER BY ASC(?ID)

```

Student	ID	Book
Antonio_Curione	"STU0001"	The_Betrothed
Mario_Franco	"STU0002"	One_No_One_and_One_Hundred_Thous
Rosanna_Acquafredda	"STU0004"	Hamlet

Figura 19 - SPARQL query n.4

Una volta verificato il giusto funzionamento di tutte le componenti dell'ontologia, tramite le interrogazioni varie, abbiamo creato il grafo corrispondente, utilizzando il plugin **"Ontograf"**, che permette di **visualizzare graficamente** tutte le componenti e le relazioni. Ogni nodo del grafo indica una classe, mentre gli archi indicano le proprietà (archi orientati, grazie ai quali si individuano facilmente i rispettivi domini e codomini).

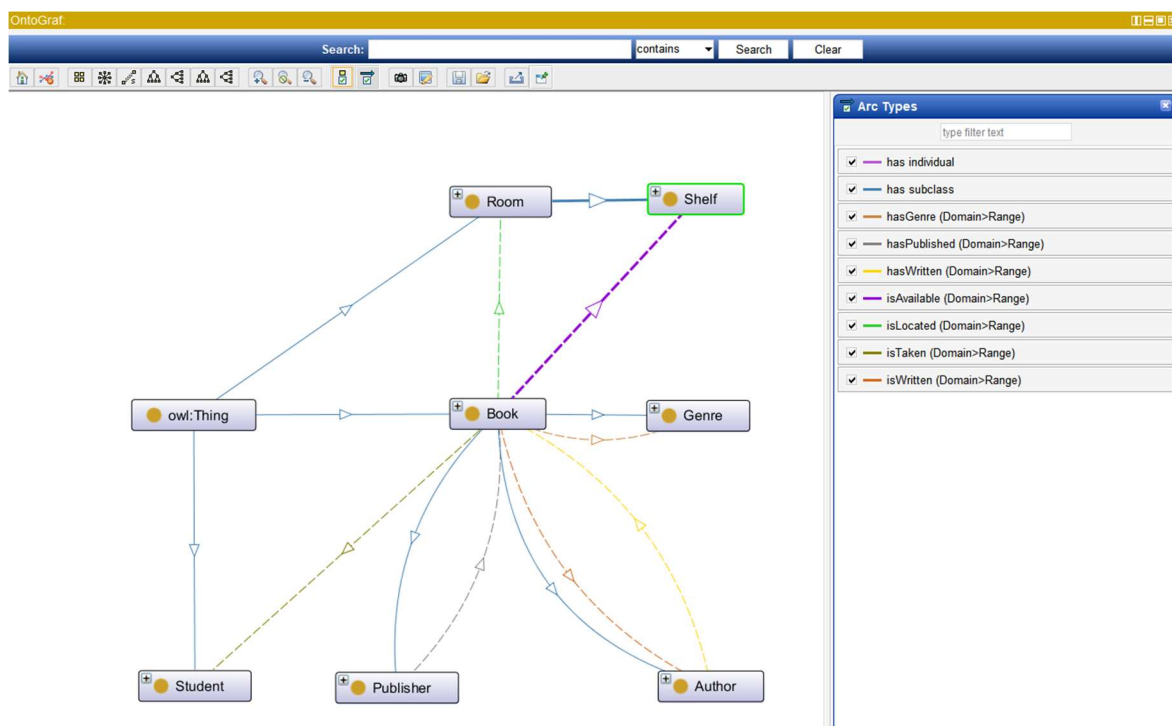


Figura 20 – Rappresentazione dell'ontologia mediante un grafo

Di seguito, l'ontologia è stata interamente codificata in Python e inserita nel progetto, nel modulo **"Ontology.py"**, attraverso il quale è possibile visualizzarla.

CONSTRAINT SATISFACTION PROBLEM

Come abbiamo già detto, il nostro progetto sviluppa un'automazione della gestione di una biblioteca scolastica. Una delle realtà che abbiamo voluto rappresentare riguarda il prestito dei libri, perché è l'azione che più frequentemente viene effettuata in un contesto come il nostro.

Prima di prendere in prestito un libro, in una comune biblioteca, bisogna considerare alcuni aspetti:

1. Innanzitutto, dato che questa operazione viene effettuata in sede, devo rispettare gli orari di apertura della biblioteca, poiché non posso prendere un libro se tale servizio non è accessibile;
 - Nel nostro caso, l'orario è riassunto nella tabella della Figura 3
2. Di seguito, ne va verificata la disponibilità negli archivi, al momento della richiesta, in quanto se un libro è stato già preso da un'altra persona, oppure, la biblioteca non ne dispone nemmeno una copia, l'operazione non è fattibile.

Un discorso analogo vale per la restituzione dei libri (precedentemente presi in prestito).

Riportando questa situazione nell'ambito dell'intelligenza artificiale, ci è sembrato che il modo più consono per risolverla fosse trattarla come un **CSP**.

Per CSP si intende un problema di soddisfacimento dei vincoli, ed è composto da una tripla

$$\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$$

Dove :

- \mathbf{X} = insieme delle variabili $\{X_1, X_2, \dots, X_n\}$;
- \mathbf{D} = insieme dei domini $\{d_1, d_2, \dots, d_n\}$, dove il generico dominio d_i contiene i valori ammessi dalla variabile X_i ;
- \mathbf{C} = insieme dei vincoli (rigidi) che specificano le combinazioni di assegnazioni ammesse.

La soluzione di un problema così strutturato è un modello, ossia un'assegnazione totale (fatta su tutte le variabili, rispetto ai propri domini) consistente (che rispetti tutti i vincoli specificati in C).

Per adattarci ad un problema simile, abbiamo modellato la nostra realtà, in modo da ricavare un insieme di variabili, individuare i relativi domini e tradurre in vincoli le riflessioni fatte in precedenza.

Individuare le variabili

L'oggetto principale del nostro sistema è il libro (da prendere in prestito o da restituire), perciò è appropriato prendere *Libro* come variabile.

L'osservazione 1 verte sull'importanza di rispettare l'orario di apertura, per tanto, rifacendoci alla figura 3, che lo riassume in giorni e fasce orarie, le altre due variabili sono rappresentate da *Giorno* e *Ora*.

Ricapitolando, il nostro insieme delle variabili è composto da {Libro, Giorno, Ora}.

Individuare i domini

Dopo aver individuato le variabili, siamo passati a definire i domini (insieme di possibili valori assumibili) per ognuna, facendo delle semplici riflessioni:

- Dato che la biblioteca è aperta 6 giorni su 7 (dal lunedì al sabato), il dominio della variabile *Giorno* è rappresentato dall'insieme : {1,2,3,4,5,6} (dove ogni numero corrisponde ad un giorno della settimana)
- La biblioteca, generalmente, è aperta dalle 9 di mattina alle 19 di sera; perciò, tale intervallo rappresenta il dominio entro il quale la variabile *Ora* può variare
- Infine, il dominio della variabile *Libro*, così come ci suggerisce l'osservazione 2, è costituito dalla lista di libri disponibili per il prestito (o per il reso, a seconda del caso)

Nota: dato che sia l'insieme delle variabili, sia i singoli domini, sono di cardinalità finita, in questo caso siamo in presenza di un CSP finito.

Individuare i vincoli

Per codificare i vincoli sulle assegnazioni *Variabile – Valore nel dominio* , ci siamo basati principalmente sull'orario di apertura della biblioteca e sull'osservazione 2, che impone la disponibilità del libro per effettuare un'azione.

In generale, abbiamo imposto che:

- La variabile *Libro* corrisponda necessariamente ad un libro della lista
- La variabile *Giorno* possa variare da 1 a 6
- La variabile *Ora* rispetti i turni della biblioteca, dove per la mattina $9 \leq \text{Ora} < 12$, mentre per il pomeriggio $16 \leq \text{Ora} < 19$

In più, abbiamo incluso un vincolo “speciale” che interessa solo il sabato, poiché in questo giorno la nostra biblioteca risulta aperta solo di mattina; perciò, le uniche assegnazioni valide saranno quelle che rispetteranno:

$$\text{Libro} \in \text{Lista di libri} \ \& \ \text{Giorno} = 6 \ \& \ 9 \leq \text{Ora} < 12$$

Dopo aver modellato la nostra realtà, così da adattarla ad un problema di soddisfacimento dei vincoli, siamo passati alla codifica del csp in Python, nell'omonimo modulo "**Csp.py**".

Nello specifico, abbiamo incluso nel modulo la libreria "**Constraint**", la quale contiene metodi adatti a trattare problemi simili.

```
problem = Problem()
problem.addVariable("Day", [1, 2, 3, 4, 5, 6])
problem.addVariable("Time", [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
problem.addVariable("idBook", list)
problem.addConstraint(lambda Day, Time, idBook:
    (Day == 1 and 9 <= Time < 12) or
    (Day == 1 and 16 <= Time < 19) or
    (Day == 2 and 9 <= Time < 12) or
    (Day == 2 and 16 <= Time < 19) or
    (Day == 3 and 9 <= Time < 12) or
    (Day == 3 and 16 <= Time < 19) or
    (Day == 4 and 9 <= Time < 12) or
    (Day == 4 and 16 <= Time < 19) or
    (Day == 5 and 9 <= Time < 12) or
    (Day == 5 and 16 <= Time < 19) or
    (Day == 6 and 9 <= Time < 12)
)
objectProblem = problem.getSolutions()
return objectProblem
```

Figura 21 - Porzione del codice utilizzata per codificare il csp in python

La prima operazione fatta, per codificare il csp individuato per il nostro caso, è stata istanziare la classe Problem, la quale ci ha permesso di definire il nostro problema e ottenere delle soluzioni.

Dopodiché, abbiamo aggiunto all'oggetto problem :

- le variabili e i domini, attraverso il metodo addVariable(), specificando le singole variabili individuate e i rispettivi domini;
 - o nota: dato che stiamo automatizzando il prestito e la restituzione di un articolo, il dominio della variabile *Libro* è rappresentato da una lista dinamica e non da un insieme statico, come per gli altri casi. La lista varia in base all'operazione da svolgere e si aggiorna al termine dell'azione.
- la codifica dei vincoli, sfruttando il metodo addConstraint(), con il quale abbiamo specificato le assegnazioni (variabile-elemento del dominio) ammesse dal nostro problema, trasformando i turni della biblioteca in restrizioni per Giorno e Ora e imponendo alla variabile Libro di assumere un valore che rispetti il proprio dominio

Dopo aver correttamente creato il nostro csp in Python, utilizzando il metodo getSolution() abbiamo ricavato tutti i modelli che possono rappresentare una soluzione per il nostro problema. Nello specifico, getSolution() restituisce una lista di dizionari che mappano le assegnazioni possibili tra variabili e valori, e lo fa sfruttando un *solver*.

Il solver di default, stabilito dalla libreria Constraint, è il Backtracking (seleziona un'opzione e, prima di passare alla successiva, completa l'intero percorso), ma in alternativa è possibile utilizzare il Recursive Backtracking o il Minimum Conflicts.

Data la natura del nostro problema, non troppo complicato, con uno spazio di ricerca ridotto e senza cicli infiniti, ci è sembrato consono utilizzare il solver di default.

Per poter stabilire se l'utente possa prendere in prestito, o rendere, un libro, effettuiamo un confronto tra i dati inseriti (giorno, ora e nome del libro) e la lista prodotta da `getSolution`, con lo scopo di trovare un modello per cui le assegnazioni rispecchiano i dati di input:

- Se tale modello esiste, vuol dire che l'utente ha rispettato tutti i vincoli, perciò è possibile effettuare l'operazione.
- Qualora i dati inseriti dall'utente non rispettassero i vincoli, dal confronto non si evidenzierà alcuna corrispondenza con i modelli consistenti e verrà comunicato allo studente l'impossibilità di procedere.

RICERCA SUL GRAFO

Per arricchire il nostro progetto, abbiamo pensato di collegare la parte del csp alla ricerca sul grafo: nel momento in cui viene preso in prestito un libro, il sistema indicherà allo studente il percorso da fare per recarsi allo scaffale dove si trova il libro in questione, così da poterlo prendere autonomamente.

Come già detto nell'introduzione, abbiamo ricavato un grafo pesato a partire dalla planimetria della piantina, evidenziando così tutti i percorsi possibili. Tale grafo rappresenta il nostro spazio di ricerca; viste le sue dimensioni ridotte, abbiamo deciso di utilizzare il metodo di ricerca il Best-First-Search, un algoritmo di ricerca informata che, per risolvere il problema, sfrutta la conoscenza a disposizione (nel nostro caso, le informazioni rispetto al peso degli archi) per restituire il percorso più corto e che rappresenti la soluzione ottimale (ossia, cammino con il minimo numero di archi e il minimo costo).

Tale metodo, utilizzato nel campo dell'Intelligenza Artificiale, si occupa di espandere i nodi del grafo, così da aumentare la distanza dalla radice(starting node) fino al raggiungimento del goal node (ossia il nodo in corrispondenza dell'obiettivo da raggiungere)

La completezza e l'ottimalità di questo algoritmo sono altri due motivi che ci hanno portato a sceglierlo, poiché se esiste una soluzione, riesce sempre a ritrovarla e a restituire quella con il costo minimo

Implementazione dell'algoritmo di ricerca:

```
visited = []
queue = [(start, 0)]

while queue:
    queue.sort(key=path_cost)
    path = queue.pop(0)
    node = path[-1][0]
    visited.append(node)
    if node == end:
        return path

    else:
        adjacent_nodes = add_cost(graph, node, graph.neighbors(node))
        for (node2, cost) in adjacent_nodes:
            new_path = path.copy()
            new_path.append((node2, cost))
            queue.append(new_path)
```

Figura 22 - Algoritmo

Questo codice prende in input il nostro grafo e produce il percorso a costo minimo, ossia la sequenza di nodi $\langle N_0, \dots, N_k \rangle$, dove N_0 indica l'ingresso alla biblioteca (dove si trova la reception) ed N_k lo scaffale di interesse, dov'è contenuto il libro.

Per la realizzazione e la stampa del grafo abbiamo incluso la libreria “**Igraph**”.

Il grafo, passato in input all'algoritmo di ricerca, viene creato “virtualmente” dalla funzione `setup_graph()`, rispettando i nodi, gli archi e i relativi pesi.

```
def setup_graph():
    a = [('0', '1', 5), ('1', '2', 3), ('2', '3', 4), ('3', '4', 4), ('4', '5', 2), ('5', '6', 5), ('5', '7', 4),
        ('6', '7', 3), ('8', '9', 1), ('9', '10', 4), ('1', '11', 3), ('11', '12', 5), ('11', '29', 5), ('1', '29', 5),
        ('29', '13', 3), ('13', '14', 3), ('29', '30', 1), ('30', '31', 3), ('30', '15', 3), ('31', '16', 3), ('16', '17', 2),
        ('31', '32', 5), ('32', '18', 2), ('32', '19', 4), ('19', '20', 3), ('32', '22', 4), ('32', '21', 6), ('21', '23', 2),
        ('22', '19', 6), ('22', '21', 6), ('19', '21', 8), ('1', '28', 1), ('28', '24', 2), ('28', '8', 2),
        ('28', '27', 3), ('28', '26', 4), ('28', '25', 4), ('27', '25', 3), ('25', '26', 2), ('26', '27', 3)]
```

Figura 23 – Porzione della funzione `setup_graph()`, nella quale vengono inseriti manualmente tutti gli archi e i pesi relativi

```
edge = []
weights = []
for i in range(40):
    for j in range(2):
        k = 2
        edge.append(a[i][j])

    weights.append(a[i][k])

edges = [(i, j) for i, j in zip(edge[:2], edge[1:2])]
list1 = []
for i in range(len(edges)):
    list1.append((int(edges[i][0]), int(edges[i][1])))

graph = Graph()
for i in range(0, 33):
    graph.add_vertex(i)

graph.add_edges(list1)
graph.es['weight'] = weights
graph.es['label'] = weights
edges = graph.get_edgelist()
return graph, edges, weights
```

Figura 24 - Porzione della funzione `setup_graph`, dove si utilizzano i metodi della libreria `igraph`

Le relazioni tra scaffali, contenenti i libri, e i nodi che gli identificano (tale associazione scaffale – nodo è riassunta nella tabella a pagina 4) sono codificate dalle funzioni `get_nodes` (che restituisce a partire dallo scaffale, il nodo corrispondente) e `get_shelf` (che restituisce lo scaffale dov'è stipato il libro).

```
def get_nodes(shelf):
    if shelf == 1:
        return 2
    if shelf == 2:
        return 3
    if shelf == 3:
        return 4
    if shelf == 4:
        return 5
    if shelf == 5:
        return 8
    if shelf == 6:
        return 9
    if shelf == 7:
```

Figura 25 - Porzione della funzione `get_nodes`

```
def get_shelf(book_id):
    if book_id == 2 or book_id == 14 or book_id ==
        n_shelf = 1
    if book_id == 1 or book_id == 3 or book_id ==
        n_shelf = 2
    if book_id == 10:
        n_shelf = 3
    if book_id == 4:
        n_shelf = 4
    if book_id == 27:
        n_shelf = 5
    if book_id == 8:
        n_shelf = 6
    if book_id == 22:
        n_shelf = 7
    if book_id == 21:
        n_shelf = 8
    if book_id == 28:
        n_shelf = 9
    if book_id == 20:
        n_shelf = 10
    if book_id == 9 or book_id == 11 or book_id ==
        n_shelf = 11
```

Figura 26 - Porzione della funzione `get_shelf`

La stampa, effettuata dopo aver trovato il cammino minimo, viene effettuata dalla funzione `print_solution`, che crea un'immagine del grafo completo nel file "grafico.png" (situato all'interno della cartella `src`), dov'è possibile visualizzare tutti i nodi etichettati, i vari archi pesati ed il percorso restituito (i cui nodi sono evidenziati in viola, rispetto a tutti gli altri che sono gialli).

```
edges = graph.get_edgelist()
g = Graph(edges)
vertex_set = set(solution)
g.vs["color"] = "yellow"
g.vs["size"] = 22
g.es["label"] = weights
try:
    sol_edges = g.vs.select(vertex_set)
except TypeError:
    vertex_set = (x[0] for x in vertex_set)
    sol_edges = g.vs.select(vertex_set)

sol_edges["color"] = color
g.layout(layout='auto')
plot(g, "grafico.png", vertex_label=vertex_values)
```

Figura 27 - Funzione `print_solution`

Esempio di una ricerca:

```
Write the name of the book chosen
it
You can take this book
Operation Done ... Thank you for borrowing a book

Do you want to see the path for reaching the book?
write 1 to search it inside the library or 2 to only borrow it
1
Thank you for borrowing the book: It
Your book is on the shelf number: 14
That's identified by the node: 16
Path:
[0, 1, 29, 30, 31, 16]
Check 'grafico.png' for visualizing the path

Do you need something more? press 1 to continue or 2 to leave:
```

Figura 28 - Esempio dell'utilizzo della ricerca nel grafo

Come si può vedere, dopo aver correttamente effettuato l'operazione di “borrow a book”, e aver richiesto di visualizzare il cammino, l'algoritmo

- dapprima fornisce delle informazioni sul libro preso, ricapitolandone il nome e specificando lo scaffale in cui è stipato (con associato il nodo che lo identifica);
- di seguito, prosegue stampando il percorso (Path) sotto forma di sequenza di nodi, che partono dalla reception, fino ad arrivare allo scaffale di interesse, ossia:

< 0, 1, 29, 30, 31, 16 >

E come si può vedere, il nodo 16 (identifica lo scaffale) rappresenta il nodo “goal”

- In contemporanea alla stampa della sequenza di nodi, il programma plotta, sul grafo presente nel file ‘grafico.png’, il percorso corrispondente (in viola)

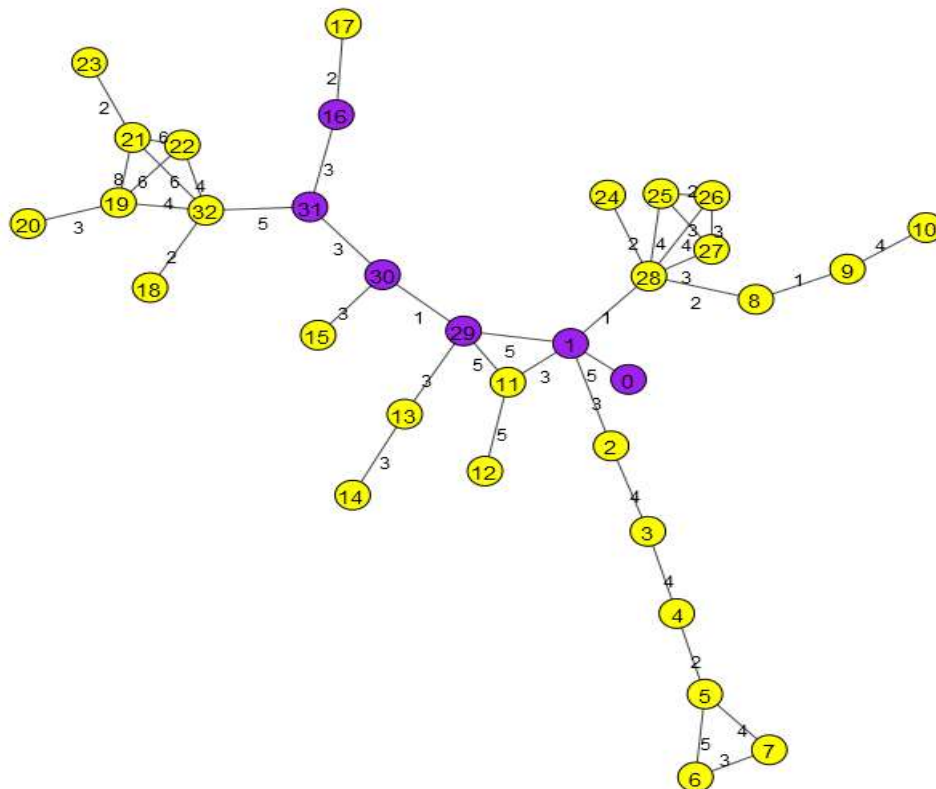


Figura 29 - output plottato sul grafo

Per avere un'idea reale del percorso restituito dall'algoritmo, lo abbiamo riportato sulla planimetria della biblioteca, cosicché lo studente possa avere delle indicazioni più precise sulla strada da intraprendere.

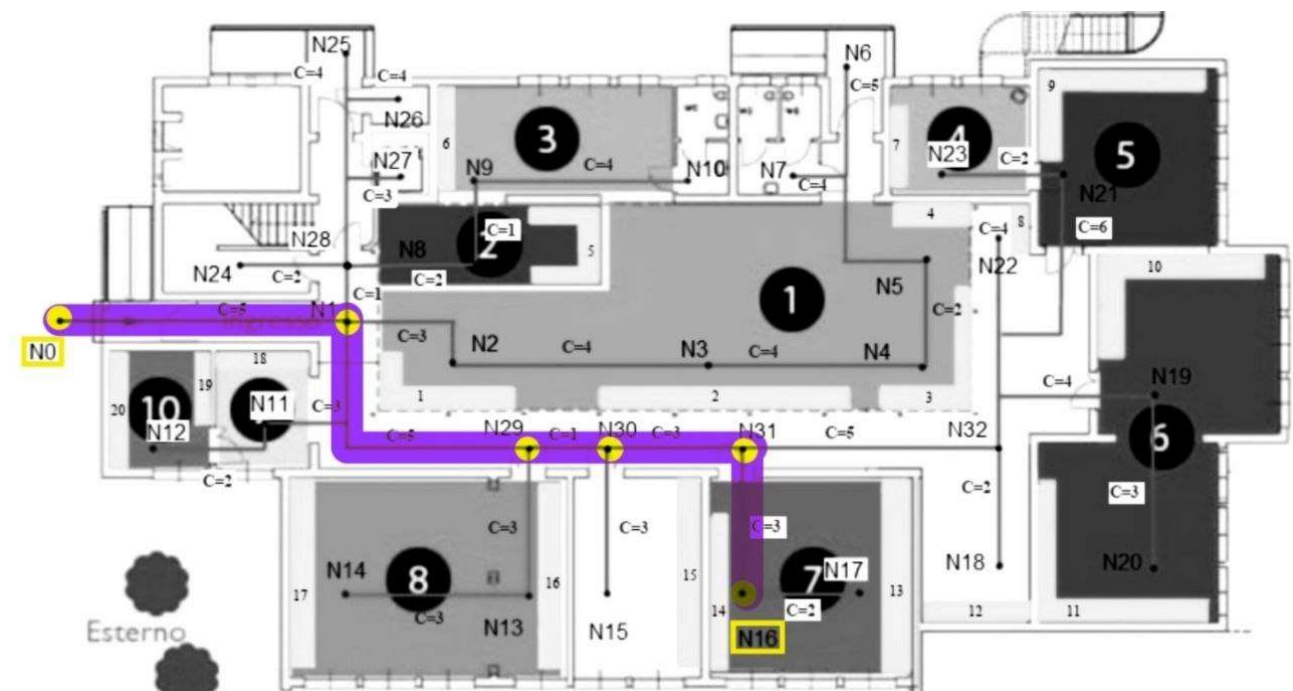


Figura 30 – percorso riportato sulla planimetria

RECOMMENDER SYSTEM – KNN

Con l'obiettivo di incentivare gli studenti alla lettura, nel nostro programma abbiamo inserito una forma di apprendimento supervisionato, ossia un recommender system che, partendo da un dataset di libri, potesse consigliare allo studente altre letture simili a quella di interesse.

I sistemi di raccomandazione sono largamente utilizzati, poiché permettono di creare raccomandazioni personalizzate, rispetto alle preferenze dell'utente. Ne esistono di diverse tipologie e quello che abbiamo implementato noi è il **KNN (k-nearest neighbors)**, che rappresenta uno degli algoritmi di apprendimento supervisionato più popolari per i problemi di classificazione.

Abbiamo scelto il KNN, poiché ci è sembrato il più adatto per il nostro obiettivo, ossia scandire una lista dei libri più simili a quello specificato dall'utente, basando le predizioni su un dataset.

Esso si compone di una fase di apprendimento che, a differenza di altri algoritmi dove si costruisce un modello per fare le predizioni, si limita a memorizzare gli esempi del training set (nel nostro caso, il dataset con i libri e le relative valutazioni) e di una fase di classificazione, che consiste nel restituire i K elementi più simili al nuovo esempio da classificare, restituendone la lista.

Tutto ciò viene fatto servendosi di una misura di similarità, ossia di una metrica che stimi quanto siano simili il libro, fornito in input dall'utente, e ogni altro libro presente nel dataset. Grazie alla misura di similarità, infatti, siamo in grado di individuare i K libri più simili, da consigliare all'utente.

La misura di similarità utilizzata per il nostro problema è nota come *similarità del coseno*, la quale misura la somiglianza tra due vettori. Nel nostro caso i 2 vettori su cui, volta per volta, si calcola la metrica, sono :

- Il vettore delle valutazioni fatte sul libro dato in input
- Il vettore delle valutazioni dell'i-esimo libro, tra quelli presenti nel dataset, che stiamo attualmente prendendo in considerazione

Fasi preliminari dell'implementazione

Il primo passo per l'implementazione del sistema di raccomandazione è stato cercare un dataset adatto al nostro scopo.

Nella cartella "data" del nostro progetto sono presenti 3 file .csv:

- BX-Books.csv, con tutte le informazioni sui libri presenti nel dataset (ISBN, titolo, autore, anno di pubblicazione,)
- BX-Users.csv, con le informazioni relative agli utenti che hanno fornito le valutazioni (id, posizione e età)
- BX-Book-Ratings.csv, che collega e riassume i libri alle valutazioni fornite dagli utenti

Che abbiamo deciso di unire, con un merge, per fornire un dataset più completo al nostro algoritmo, con tutte le informazioni dei libri e le valutazioni.

Prima di utilizzare il dataset, abbiamo filtrato gli elementi, ponendo due condizioni:

- Ogni libro deve aver ricevuto almeno 10 voti,
- Ciascun utente deve aver espresso almeno 10 voti.

Per gestire al meglio un dataset di tali dimensioni abbiamo utilizzato la libreria “**Pandas**”.

Di seguito abbiamo impostato il parametro K a 10, perciò il sistema restituirà una lista con i 10 libri più simili a quello fornito in input.

Funzionamento

```
book_name = input('Insert favourite or last read book name:\n')
top_n = 10
recommender = KnnRecommender()
recommendation = recommender.make_recommendations(book_name, top_n)
```

Innanzitutto, lo studente inserisce il nome del libro di suo interesse.

Di seguito, viene istanziata la classe KnnRecommender (da noi creata, contenente tutti i metodi utili) e richiamato il metodo make_recommendations(), che restituisce l'output, ossia la lista di libri raccomandati.

Una delle prime cose che fa la funzione make_recommendations è restituire, utilizzando la funzione fuzz.ratio della libreria “**Fuzzywuzzy**”, una lista di titoli e indici dal dataframe, il cui titolo ha una certa corrispondenza, espressa in percentuale, con quello del libro dell'utente (secondo la distanza di Levenshtein, che misura la differenza tra due stringhe).

Questa lista viene processata dalla funzione NearestNeighbors() della libreria “**Sklearn.neighbors**” con la quale viene stilata la lista di raccomandazioni, utilizzando la similarità del coseno.

Dato che abbiamo trovato un'incoerenza tra alcuni libri presenti nei file .csv delle valutazioni utenti e dei libri, abbiamo filtrato la lista delle raccomandazioni ottenute in precedenza, secondo i dati appartenenti esclusivamente al dataframe dei libri.

VALUTAZIONE DEI RISULTATI

In questa sezione abbiamo riassunto un po' quelle che sono le nostre considerazioni rispetto al sistema appena creato, basandoci sui risultati ottenuti da una serie di test.

L'obiettivo comune a tutti i test effettuati è vedere come il sistema risponde ai diversi input, che rispecchiano non solo le normali situazioni di normale utilizzo, ma anche i casi limite, come ad esempio l'uso di input errati.

Test relativi al sistema di prenotazione/reso di un libro (Csp):

#Test	Obiettivo test	Risultato ottenuto
1	Effettuare la prenotazione di un libro disponibile, come It, di lunedì, alle 10.10 (questo test rappresenta un caso in cui tutti i vincoli sono rispettati)	L'operazione è andata a buon fine. Il sistema lavora correttamente con gli input corretti
<pre>Day: 1 Insert time in the following format: [HH].[MM] (example 10.20) Time: 10.10 We're open Do you want to borrow or return a book? write 1 to borrow or 2 to return 1</pre>		<pre>It Fairy tales and folklore This book is funny Greek mythology Write the name of the book chosen It You can take this book Operation Done ... Thank you for borrowing a book</pre>
2	Verificare che il sistema di prenotazione, inserendo input corretti, non sia case sensitive	Scrivere il nome del libro tutto in minuscole (mindhunter) o tutto in maiuscole (THE TRIAL) non impatta sul corretto funzionamento
<pre>Day: 1 Insert time in the following format: [HH].[MM] (example 10.20) Time: 10.20 We're open Do you want to borrow or return a book? write 1 to borrow or 2 to return 1</pre>		<pre>Write the name of the book chosen mindhunter You can take this book Operation Done ... Thank you for borrowing a book</pre> <pre>Write the name of the book chosen THE TRIAL You can take this book Operation Done ... Thank you for borrowing a book</pre>
3	Provare a prenotare un libro disponibile, inserendo input scorretti per il Giorno o l'ora.	Il sistema comunica correttamente all'utente il verificarsi di un errore e termina il programma
<pre>Day: 10 Wrong day format Process finished with exit code 0</pre>		<pre>Day: monday Invalid input type Process finished with exit code 0</pre>

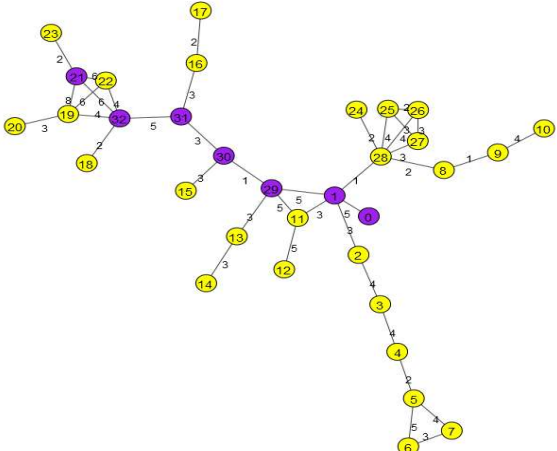
	<pre>Day: 2 Insert time in the following format: [HH].[MM] (example 10.20) Time: 15.00 We're sorry, but the library is currently closed. Process finished with exit code 0</pre>	<pre>Time: 10,20 Invalid input type Process finished with exit code 0</pre>
4	Provare a prenotare un libro non disponibile al momento(es: the idiot) o che proprio non è presente nella biblioteca (es: ciao)	In entrambi i casi il sistema non permette di prendere il libro, comunicandolo all'utente. Nel caso 2 viene visualizzato un messaggio in più poiché la stringa inserita rappresenta un input scorretto.
<pre>Write the name of the book chosen the idiot We are sorry, but for some reason you can't take this book (constraints problems occurred)</pre>		
<pre>Write the name of the book chosen ciao Please, insert the name of an available book! We are sorry, but for some reason you can't take this book (constraints problems occurred)</pre>		

Nota: i test sopra effettuati sono stati replicati per il caso “return”, ottenendo gli stessi esiti

Test relativi alla ricerca del libro nella biblioteca (Ricerca nel grafo):

il modulo della ricerca nel grafo, utilizzato per ricavare il percorso dall'ingresso della biblioteca, allo scaffale dov'è contenuto il libro, viene richiamato solo dopo che lo studente ha preso in prestito un libro e non richiede l'inserimento di altri dati; perciò, l'interazione con l'utente è minima in questa fase.

Per tali ragioni, in questo caso, non è possibile verificare la correttezza del sistema in situazioni particolari. Ci limitiamo quindi a fornire un semplice esempio:

#Test	Obiettivo test	Risultato ottenuto
1	Dopo aver preso in prestito un libro, visualizzare il percorso che porti lo studente allo scaffale in cui è contenuto	Il sistema elenca la sequenza di nodi che interessano il percorso e lo plotta correttamente sul grafo
<pre>Write the name of the book chosen mindhunter You can take this book Operation Done ... Thank you for borrowing a book Do you want to see the path for reaching the book? write 1 to search it inside the library or 2 to only borrow it 1 Thank you for borrowing the book: Mindhunter Your book is on the shelf number: 9 That's identified by the node: 21 Path: [0, 1, 29, 30, 31, 32, 21] Check 'grafico.png' for visualizing the path</pre>		

Test relativi alla raccomandazione dei libri (Recommender system con KNN):

#Test	Obiettivo test	Risultato ottenuto
1	Stilare una lista di 10 libri simili a quello dato in input	<p>Il sistema restituisce la lista dei libri con titolo simile a quello scelto, di seguito esprime le 10 raccomandazioni per il titolo dato in input</p> <pre> Insert favourite or last read book name: First Lady Your book is: First Lady Found results: ['To Please a Lady', 'First Lady', 'The Lady', 'It Was on Fire When I Lay Down on It', 'Clay', 'Lead Me On', 'Daddy', 'Candy', 'The Iliad', 'LADY BOSS', 'Weep No More, My Lady', 'Weep No More My Lady', 'Lost Lady', 'Dark Lady', 'Lizard', 'The Legacy', 'Poland', 'RepLay', 'Legacy', 'Island', 'Up Island'] Recommendations for : First Lady 1: Death at Sandringham House (Her Majesty Investigates), with distance of 0.3753049373626709 2: Countess Misbehaves, with distance of 0.3753049373626709 3: Bound by Desire, with distance of 0.4478423595428467 4: The Day the Rabbi Resigned, with distance of 0.4824294447898865 5: Spock's World (Star Trek: The Original Series), with distance of 0.5121951103210449 6: Falling Stars (Topaz Historical Romances), with distance of 0.5298690795898438 7: The Man in the Iron Mask (Signet Classics (Paperback)), with distance of 0.5298690795898438 8: Letters of Abelard and Heloise, The, with distance of 0.5330970287322998 9: ISHMAEL, with distance of 0.5330970287322998 10: The Snow Queen, with distance of 0.556048572063446 </pre>
2	Da quest'ultimo test, come possiamo notare, nella lista delle raccomandazioni ci sono delle posizioni contrassegnate da "not found". Questo non indica un errore del recommender system, ma solo il verificarsi di un'incoerenza nel dataset (di cui abbiamo parlato nella sezione dedicata al KNN)	<pre> It Your book is: It Found results: ['Into The Fire', 'Twelve Across', 'Above And Beyond', 'There And You', 'Out of Nowhere', 'Others', 'Twelve', 'Make Them Cry', 'Never', 'Then She Found Me', 'By Any Other Name', 'Now You See Her', 'To All', 'The Three of Us', 'It', 'He, She and It', 'I Do, I Do, I Do', 'FIFTEEN', 'Now or Never', 'One', 'At Last', 'If I Had You', 'The Here and Now', 'Together', 'It Had to Be You', 'Only You', 'Only The Last Don', 'The Eight', 'Once', 'Now You See Me', 'Was', 'To Recommendations for : It 1: Secrets of the Heart, with distance of 0.10926008224487305 2: Black Silk (Avon Romance), with distance of 0.2382117509841919 3: Not found 4: Wind of the Wolf, with distance of 0.36627572774887085 5: Gentleman Caller, with distance of 0.36627572774887085 6: Not found 7: Thunder Creek, with distance of 0.36627572774887085 8: Not found 9: Tidewater, with distance of 0.36627572774887085 10: Not found </pre>

Test dell'Ontologia:

Il nostro programma, più che manipolare l'ontologia, permette di visualizzarla, stampando la lista degli individui che popolano le singole classi e delle query di esempio (permettono di verificare che l'ontologia sia stata correttamente codificata).

Il modo migliore per testare l'ontologia è effettuare delle query in SPARQL, come abbiamo già fatto nel capitolo dedicato, utilizzando il file **library.owl** (dov'è presente per intero l'ontologia descritta, in formato OWL). È possibile prendere spunto dal file **ontology_queries.txt**, nella cartella docs, dove sono presenti delle query già fatte.

Considerazioni finali sui test

I test effettuati hanno evidenziato un corretto funzionamento del sistema, anche nei casi limite, in cui sono stati inseriti input errati, o in situazioni atipiche.

Considerando che:

- i risultati ottenuti sono sempre coerenti con le aspettative,
- le situazioni più insolite, che sono state gestite opportunamente con la raccolta delle eccezioni, non portano ad arresti anomali dell'applicazione,
- e soprattutto, il sistema intrattiene una buona comunicazione con l'utente, in quanto, dopo ogni azione corretta o meno che sia, stampa sempre dei messaggi riguardanti l'esito dell'operazione compiuta,

ci sentiamo soddisfatti del lavoro svolto. Il sistema realizzato, rispecchia la nostra idea iniziale e si adatta bene realtà che abbiamo preso in considerazione e modellato.

NOTE FINALI

Per la realizzazione di questo progetto abbiamo lavorato in pair-programming, sfruttando il repository GitHub appositamente creato.

Durante la durata del progetto, come canali di comunicazione, abbiamo utilizzato un gruppo Telegram ed un canale su Teams, che abbiamo sfruttato soprattutto per fare delle riunioni, per discutere del lavoro fatto o su come suddividerci i compiti restanti.