UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO
DIPARTIMENTO DI INFORMATICA

Documentazione Progetto
ICON
A.A. 2023/2024

Gruppo di lavoro

- Ruggiero Palmitessa, 738751, r.palmitessa3@studenti.uniba.it
- Flavio Palma, 717161, f.palma14@studenti.uniba.it

SOMMARIO

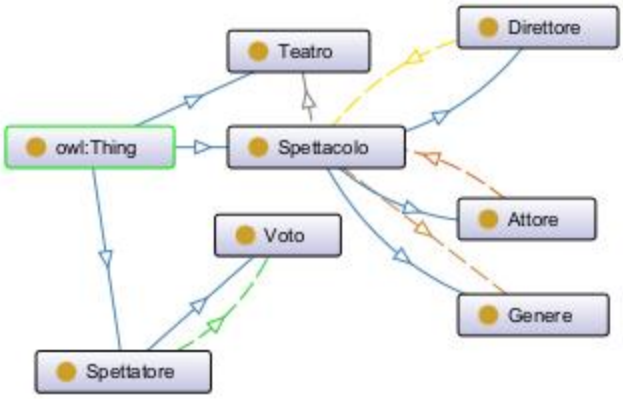
INTRODUZIONE

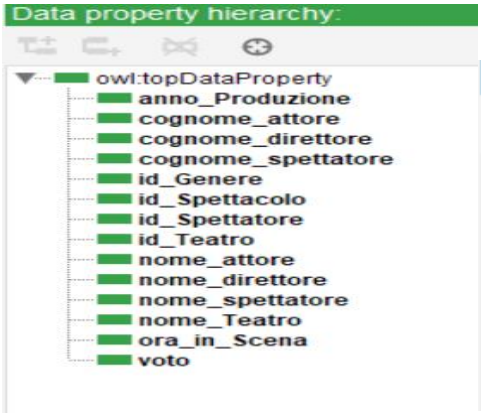
Il progetto svolto si occupa della gestione di un teatro sociale aperto al pubblico gratuitamente, dove il feedback degli utenti facenti parte sarà elaborato da un recommender system basato su KNN(Nearest-Neighbor). L'idea nasce nel voler creare un sistema per permettere a chiunque si affacci, anche per la prima volta ad un locale, di potersi muovere attraverso i molteplici spazi in maniera ottimizzata per evitare di perdersi e quindi perdere tempo, che nel nostro caso specifico equivale a perdere pezzi di spettacoli.

ONTOLOGIA

Per creare l'ontologia si è usato il programma Protègè (https://protege.stanford.edu/).

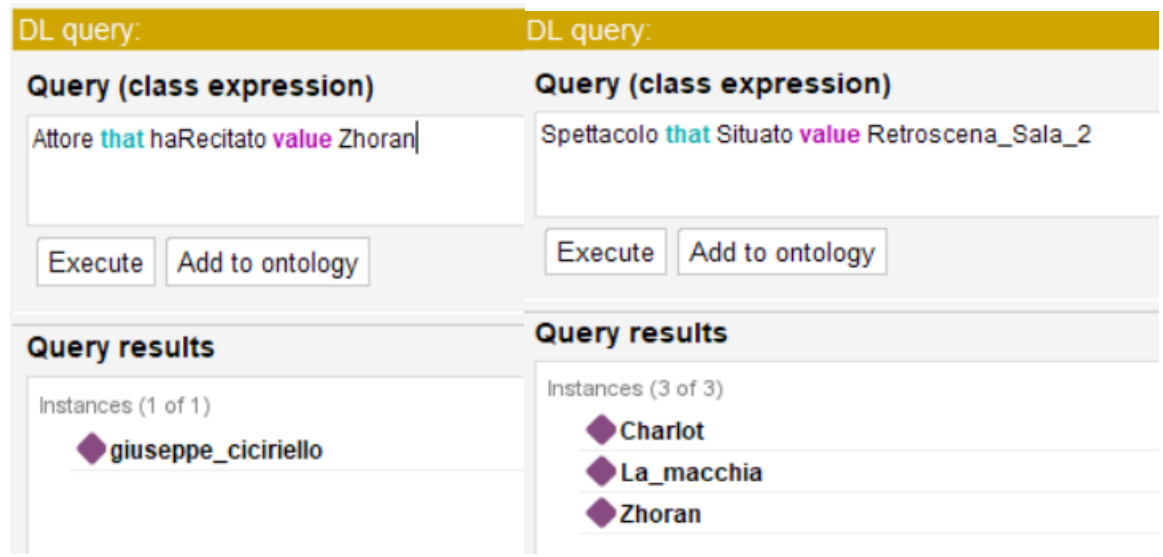
Il primo passo è stato immaginare le relazioni che tutti i dati avrebbero avuto tra loro e quindi creare un'ontologia. Su Protegè abbiamo impostato le diverse classi "Spettacolo" "Teatro" "Spettatore", le loro sottoclassi e le proprietà che le avvalorano.



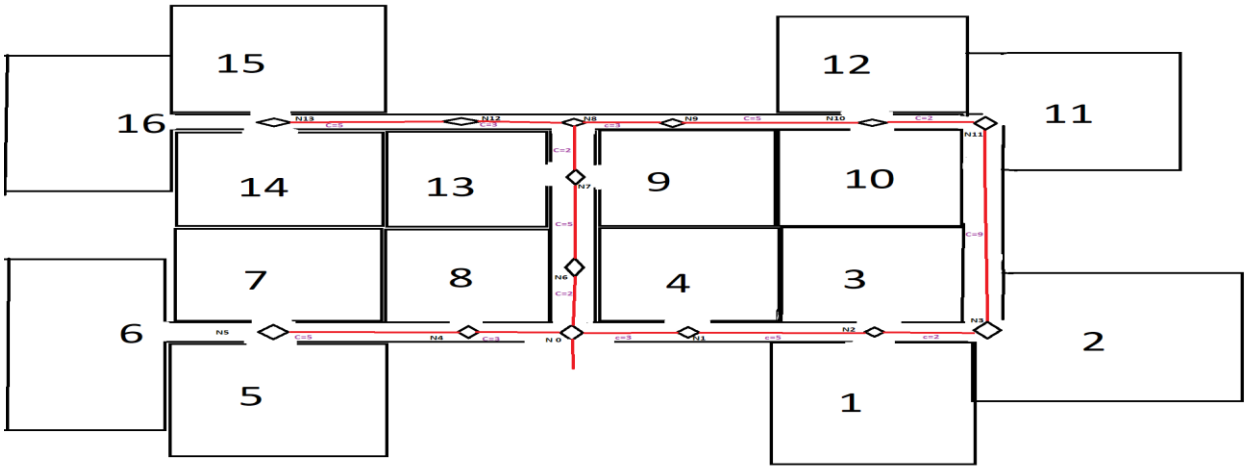


Proprietà dei dati ^ Proprietà degli oggetti ^

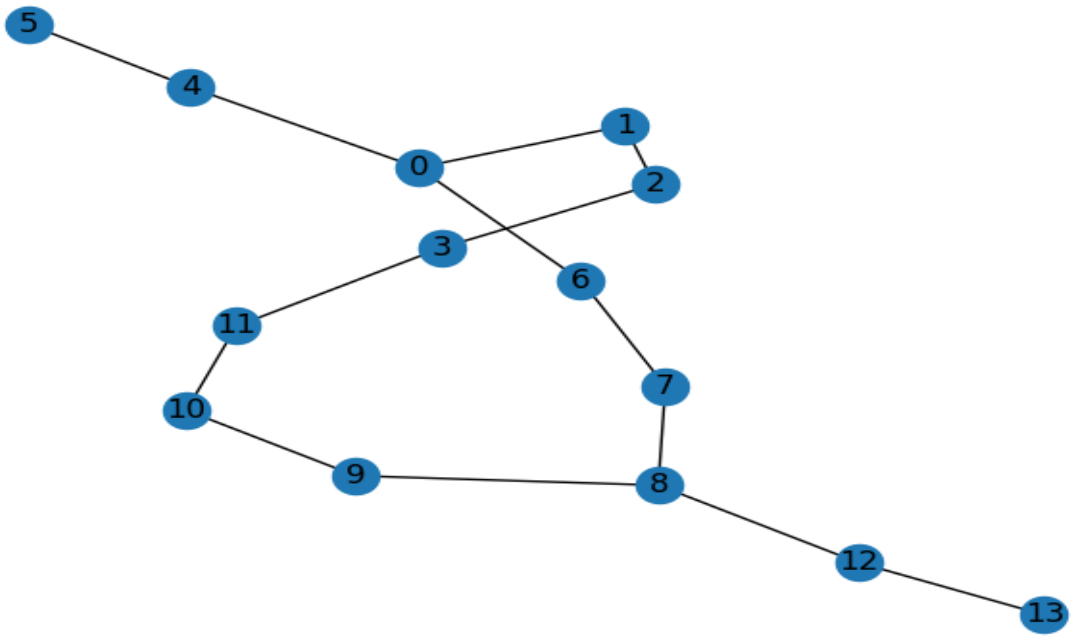
Dopo che l'ontologia è stata creata abbiamo proceduto ad inserire delle istanze sulle quali lanciare delle query; protegè permette, out of the box, di interrogare l'ontologia tramite la schermata DL QUERY. Il reasoner utilizzato è quello di default di Protègè (HermiT).



Successivamente abbiamo creato una mappa che rispecchia la conformazione del teatro sociale così da poter individuare i diversi percorsi che si possono intraprendere per spostarsi da uno spettacolo ad un altro. Il percorso viene così immaginato come un grafo formato da nodi e da archi pesati in base alla distanza tra i nodi.



Planimetria del teatro ^



Grafo dei percorsi possibili ^

Per convertire la planimetria in un grafo è stato utilizzato il linguaggio python implementando le librerie networkx e matplotlib.pyplot le quali permetto prima di creane la struttura e successivamente di stamparlo a schermo.

```python
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()

# CreazioneNodi
G.add_node(0)
G.add_nodes_from([1, 2, 3])
G.add_nodes_from([4, 5])
G.add_nodes_from([6, 7, 8])
G.add_nodes_from([9, 10, 11])
G.add_nodes_from([12, 13])

# Creazione Archi
G.add_edges_from([(0, 1), (1, 2), (2, 3)])
G.add_edges_from([(0, 4), (4, 5)])
G.add_edges_from([(0, 6), (6, 7), (7, 8)])
G.add_edges_from([(8, 9), (9, 10), (10, 11)])
G.add_edges_from([(8, 12), (12, 13)])
G.add_edges_from([(3, 11)])

# Visualizzazione Grafo
nx.draw(G, with_labels=True)
plt.show()

print("Numero di nodi: ", G.number_of_nodes())
print("Numero di archi: ", G.number_of_edges())
```

Creazione del grafo^

Possiamo anche consultare, utilizzando il programma ontologia.py, la nostra ontologia grazie alla libreria OwlReady2.

ontologia.py

C: > Users > Flavio Palma > Desktop > ICON > Progetto > Prog23_24 > src > ontologia.py > ...

```python
from owlready2 import *

print("ONTOLOGIA\n")
onto = get_ontology("theatre_show.owl").load()

# Stampa il contenuto principale dell'ontologia
print("Lista delle classi nell'ontologia:\n")
print(list(onto.classes()), "\n")

# Stampa le proprietà oggetto dell'ontologia
print("Proprietà oggetto nell'ontologia:\n")
print(list(onto.object_properties()), "\n")

# Stampa le proprietà dati dell'ontologia
print("Proprietà dati nell'ontologia:\n")
print(list(onto.data_properties()), "\n")

# Stampa gli individui per classi specifiche
print("Lista degli Attori nell'ontologia:\n")
attori = onto.search(is_a=onto.Attore)
print(attori, "\n")

print("Lista dei Direttori nell'ontologia:\n")
direttori = onto.search(is_a=onto.Direttore)
print(direttori, "\n")

print("Lista dei Generi nell'ontologia:\n")
generi = onto.search(is_a=onto.Genere)
print(generi, "\n")

print("Lista degli Spettacoli nell'ontologia:\n")
spettacoli = onto.search(is_a=onto.Spettacolo)
print(spettacoli, "\n")

print("Lista degli Spettatori nell'ontologia:\n")
spettatori = onto.search(is_a=onto.Spettatore)
print(spettatori, "\n")

print("Lista dei Teatri nell'ontologia:\n")
teatri = onto.search(is_a=onto.Teatro)
print(teatri, "\n")

print("Lista dei Voti nell'ontologia:\n")
voti = onto.search(is_a=onto.Voto)
print(voti, "\n")


#esempi di query
print("Lista degli spettacoli di genere musical:\n")
spettacoli = onto.search(is_a = onto.Spettacolo, haGenere = onto.search(is_a = onto.musical))
print(spettacoli, "\n")
```

Codice Ontologia.py ^

```
ONTOLOGIA

Lista delle classi nell'ontologia:

[theatre_show.Spettacolo, theatre_show.Teatro, theatre_show.Direttore, theatre_show.Genere, theatre_show.Attore, theatre_show.Spettatore, theatre_show.Voto]

Proprietà oggetto nell'ontologia:

[theatre_show.Situato, theatre_show.haDiretto, theatre_show.haGenere, theatre_show.haRecitato, theatre_show.haVotato]

Proprietà dati nell'ontologia:

[theatre_show.anno_Produzione, theatre_show.cognome_attore, theatre_show.cognome_direttore, theatre_show.cognome_spettatore, theatre_show.id_Genere, theatre_show.id_Spettacolo, th
eatre_show.id_Spettatore, theatre_show.id_Teatro, theatre_show.nome_Teatro, theatre_show.nome_attore, theatre_show.nome_direttore, theatre_show.nome_spettatore, theatre_show.ora_i
n_Scena, theatre_show.voto]

Lista degli Attori nell'ontologia:

[theatre_show.Attore, theatre_show.Charlie_chaplin, theatre_show.Piero_Santoro, theatre_show.giuseppe_ciciriello, theatre_show.mario_rossi, theatre_show.serena_callia]

Lista dei Direttori nell'ontologia:

[theatre_show.Direttore, theatre_show.Fabrizio_Suma, theatre_show.mirko_lodedo, theatre_show.william_shakespear]
```

Output Ontologia.py ^

DATASET UTILIZZATI

Il dataset Opere è stato creato sulla base di opere teatrali o spettacoli esistenti mentre il dataset dei ratings è stato generato in maniera automatica.

Tutto il lavoro è stato scritto in linguaggio Python, utilizzando il code editor VsCode e pyCharm. Per l'intero progetto è stato fatto uso delle librerie presenti all' interno del documento di testo **requirements.txt**

idutente,idopera,valutazione
1,10,1
1,11,1
1,7,2
1,110,4
1,100,4
1,60,1
1,1,5
1,28,4
1,20,3
1,37,2
1,51,4
1,73,2

Dataset Rating ^

idopera,titolo,genere
1,Sogno di una notte di mezza estate,Commedia
2,Il mercante di Venezia,Dramma
3,Romeo e Giulietta,Tragedia
4,La dodicesima notte,Commedia
5,Otello,Tragedia
6,La bisbetica domata,Commedia
7,Macbeth,Tragedia
8,Amleto,Tragedia
9,Le allegre comari di Windsor,Commedia
10,La tempesta,Commedia
11,Il misantropo,Commedia
12,Cyrano de Bergerac,Dramma|Romantico
13,Edipo re,Tragedia
14,Antigone,Tragedia
15,Medea,Tragedia
16,Le Troiane,Tragedia
17,Prometeo incatenato,Tragedia
18,Elettra,Tragedia
19,La notte dei re,Commedia
20,Il Tartufo,Commedia
21,Il malato immaginario,Commedia
22,Don Giovanni,Opera
23,Le nozze di Figaro,Opera

Dataset Opere ^

Classificatore KNN

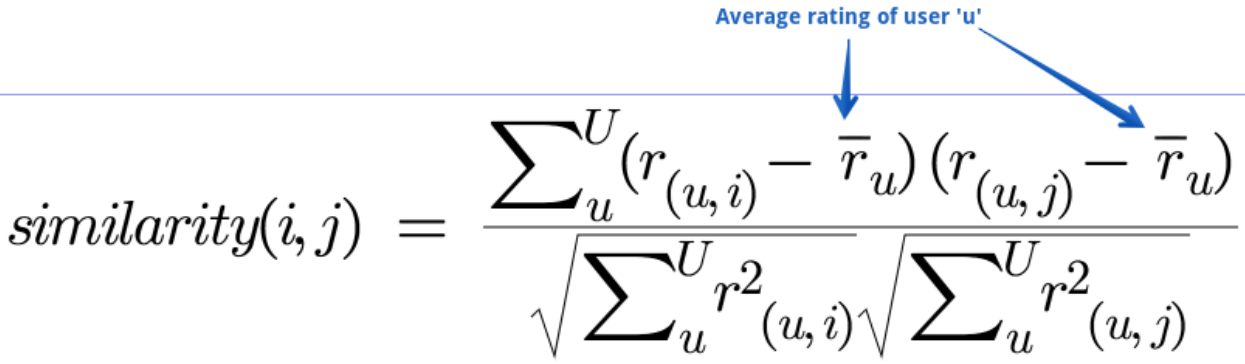
Il sistema di raccomandazione basato su KNN (K-Nearest Neighbors) utilizza un approccio di apprendimento supervisionato per fornire raccomandazioni personalizzate agli utenti. In questo contesto, l'apprendimento si riferisce alla capacità di migliorare le prestazioni del sistema utilizzando l'esperienza passata. Nello specifico, l'apprendimento supervisionato coinvolge la generazione di output basata su un set di dati di addestramento etichettato, che serve come base per dedurre nuova conoscenza.

Il KNN, o "lazy learning", è un algoritmo che opera nel contesto della classificazione supervisionata. La sua logica consiste nell'individuare i K esempi più simili o vicini a un dato in input che deve essere classificato.

L'algoritmo prende in input il valore di K e l'opera preferita dell'utente. Successivamente, utilizza un metodo di matching approssimato di stringhe fornito dalla libreria "fuzzywuzzy". Questo metodo consente di cercare una corrispondenza approssimata tra il titolo dello spettacolo inserito dall'utente e i titoli presenti nel dataset. Una volta trovata una corrispondenza, l'algoritmo estrae l'ID corrispondente all' opera.

Collaborative Filtering

L'id dello spettacolo ci permette di estrarre tutti gli utenti che lo hanno valutato, omettendo quelli che hanno votato meno di 4 spettacoli in passato e le opere con meno di 10 valutazioni.
Detto ciò, il programma inizierà ad elaborare una matrice user-items, dove le colonne n saranno è il numero degli spettacoli, le righe saranno equivalenti al numero di neighbors u trovati dal knn e gli elementi in posizione M (u, n) avranno, nel caso siano stati votati, dei giudizi da 1 a 5.



Adjusted Cosine Similarity ^

User-item rating matrix

	m1	m2	m3
u1	2	?	3
u2	5	2	?
u3	3	3	1
u4	?	2	2

Matrice User item ^

Vincoli (CSP)

I vincoli nei Problemi di Soddisfacimento dei Vincoli (CSP, Constraint Satisfaction Problems) rappresentano le condizioni o le regole che devono essere rispettate dalle variabili nel trovare una soluzione al problema. In un CSP, hai variabili con domini di possibili valori e vincoli che limitano le combinazioni accettabili di valori per queste variabili. L'obiettivo è trovare un'assegnazione coerente di valori alle variabili che soddisfi tutti i vincoli dati. Questo processo coinvolge l'esplorazione sistematica delle possibili

assegnazioni fino a trovare una soluzione o determinare che non ne esiste alcuna. Una soluzione a un problema CSP è un'assegnazione completa e consistente, possiamo affermare quindi che una soluzione è un modello. La libreria "python-constraints" che è stata utilizzata" che consente di creare dei csp e di generare tutti i modelli. Generate le soluzioni l'utente dovrà inserire un giorno, un orario e uno spettacolo tra quelli disponibili tra quelli aggiunti al dominio. Successivamente verrà ricercato un modello che accetti le variabili inserite dall'utente. La libreria ha a disposizione 3 solver: backtracking, recursive backtracking e minimum conflicts. Quindi utilizzando il backtracking classico troviamo la nostra soluzione o in caso contrario un messaggio "Spettacolo non disponibile". Nel nostro caso i vincoli non ci permettono di mostrare la strada per raggiungere uno spettacolo se lo spettacolo stesso non è in scena nell'orario selezionato dall'utente.

```
if __name__ == "__main__":

    problem = Problem()
    problem.addVariable( variable: "Day",  domain: [1, 2, 3, 4, 5, 6, 7])
    problem.addVariable( variable: "Time",  domain: [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])
    problem.addVariable( variable: "Idopera",
                         domain: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
                         27, 28,
                         29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
                         43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
                         53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
                         68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,
                         84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
                         100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112])
    # vincoli orari settimana
    problem.addConstraint(lambda Day, Time, Idopera:
                          (Day == 1 and 8 <= Time <= 9 and Idopera == 1)
                          or (Day == 1 and 9 <= Time <= 10 and Idopera == 8)
                          or (Day == 1 and 10 <= Time <= 11 and Idopera == 15)
                          or (Day == 1 and 11 <= Time <= 12 and Idopera == 22)
```

Nei CSP applicati a domini come la pianificazione, un mondo potrebbe rappresentare una situazione specifica, ad esempio, un particolare stato della programmazione degli spettacoli potrebbe essere utilizzato per indicare diversi stati del modello, ognuno corrispondente a una possibile combinazione di variabili.

Ricerca nel Grafo

Il "Lowest Cost Search", è un algoritmo di ricerca informata come l'A* (A-star). Questo algoritmo è progettato per trovare il percorso con il costo più basso da un punto di partenza a una destinazione, utilizzando sia la distanza effettiva percorsa che una stima della distanza rimanente. L'A* è ottimale e completo quando l'euristica è ammissibile.

A* è completo in quanto se esiste una soluzione al problema, la troverà. Questa proprietà si basa sulla strategia di esplorazione degli stati che tiene conto dei costi correnti e delle stime di costo futuri.

È anche ottimale dato che quando l'euristica utilizzata è ammissibile e consistente, A* è garantito di trovare la soluzione con il costo minimo.

Ciò è possibile grazie al fatto che l'algoritmo utilizza una combinazione di costo effettivo e stima euristica per selezionare i nodi da esplorare, favorendo quelli con costi inferiori totali stimati.

La ricerca grafo ci permette di andare ad individuare all'interno del nostro grafo pesato i percorsi tra gli spettacoli o per uno spettacolo singolo.

A ogni sala corrisponde uno e un solo nodo il più vicino ed a ogni nodo però possono corrispondere più sale. Quindi abbiamo una tabella di traduzione per identificare a che nodo appartiene una determinata opera.

Nodo	Sala
0	\
1	4
2	1,3
3	2
4	8
5	5,6,7

Nodo	Sala
6	\
7	9,13
8	\
9	\
10	10,12
11	11

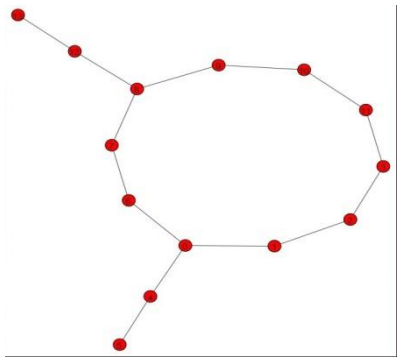
Nodo	Sala
12	\
13	14,15,16

Abbiamo stabilito un ordinamento tra le stanze e gli spettacoli assegnandone 7 a testa.

Gli spettacoli sono 112 ed attraverso l'idopera ritrovata noi possiamo calcolare la sala di appartenenza con questa funzione **(idopera /7) +1** e dalla tabella risaliamo al nodo.

Quindi richiamata la funzione di ricerca, ricevendo come parametro la lista degli spettacoli, per ogni spettacolo, ritrovi e stampi a video i percorsi per arrivare dall' entrata del teatro fino alla sala dove è presente la prima opera interessata e, nel caso di altre opere della lista, si itera la ricerca prendendo come nodo di partenza del nuovo percorso, il nodo del precedente spettacolo.

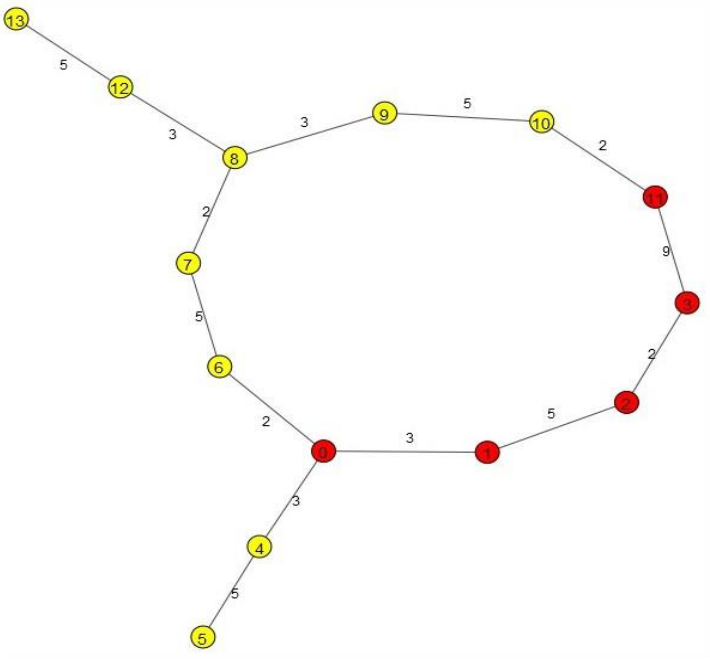
Simulazione di funzionamento



< Grafo raffigurante i percorsi

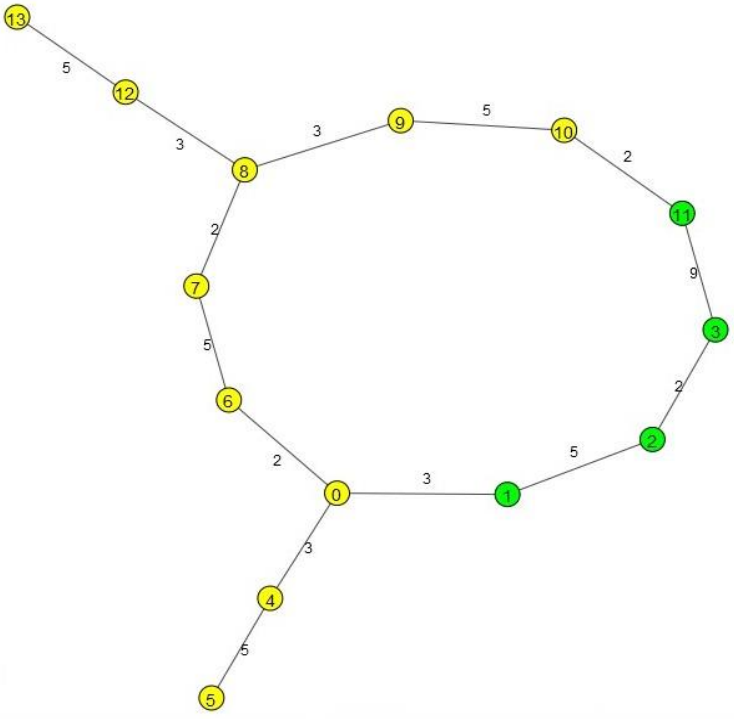
```
Inserisci il nome dell'opera preferita:
la bisbetica
La tua opera è: la bisbetica
Risultati trovati:
['La bisbetica domata']
Raccomandazioni per: la bisbetica
1: Il circo magico, con una distanza di 0.1673162579536438
2: La Bohème, con una distanza di 0.184443593020752
```

Nella prima iterazione data la lista dobbiamo ritrovare l'opera "Il circo magico" che ha come idopera= (72). Applicando la funzione enunciata calcoliamo quindi che la stanza corrispondente è la stanza 11 e successivamente il nodo da raggiungere sarà l'11.



<Prima iterazione

La seconda opera da raggiungere sarà “La Boheme” che come idopera= (27) quindi la stanza corrispondente sarà la stanza 4 ed il nodo di destinazione sarà di conseguenza il nodo 1



<Seconda iterazione

Conclusioni

La repository con tutti i file del progetto:
https://github.com/Rino120/Icon.git

La lista delle dipendenze del progetto è presente nel file *requirement.txt* caricato sulla repository.

Ciascuno dei due membri del gruppo ha ricevuto specifici compiti da svolgere durante il progetto. Per tutta la durata del progetto, il lavoro è stato coordinato su Teams.