Relazione Progetto Movida

Di Pasquale Alessio, Colamonaco Stefano

Febbraio 2021, Algoritmi e Strutture Dati

1 IMovidaConfig

Le due strutture che ci sono state assegnate sono: **AVL** e **HashConcatenamento**, mentre i due algoritimi di ordinamento sono: **QuickSort** e **SelectionSort**.
IMovidaConfig si occupa semplicemente di settare queste due scelte, variando quindi la struttura del database e salvando l'algoritmo di ordinamento scelto.

2 IMovidaDB

IMovidaDB si occupa di tutta la gestione del database. La prima funzione fondamentale svolta dalla classe "Database" è la **LoadFromFile** che, una volta controllata che struttura si è scelta, legge man mano il file e popola la struttura dati selezionata.
Nel caso della tabella hash accediamo all'hashCode della chiave e inseriamo all'interno della lista, mentre per l'AVL, una volta inserito il nodo è necessario effettuare tutte le operazioni di ribilanciamento dell'albero.
Una scelta implementativa che abbiamo effettuato all'interno di questa funzione è quella di popolare già durante lo scan del file in input il grafo delle collaborazioni, che verrà poi usato da IMovidaCollaborations.
All'interno di "Database", oltre le funzioni di supporto per la load, abbiamo i metodi che ci permettono di accedere a **MovieData** e **PersonData**, ovvero le coppie Chiave, Valore dei dati salvati.
Infatti il dato elementare di queste strutture dati è una semplice coppia chiave valore implementata all'interno della classe astratta **Map**.
Infatti sia **AVL** che **HashConcatenamento** estendono la classe Map, per avere quindi una omogeneità dei comandi anche quando variamo le strutture dati.
Un'altra funzione richiesta da IMovidaDB è la scrittura su file, implementata dal metodo **saveToFile**. Questa funzione non fa altro che accedere alle strutture dati e scrivere man mano su un file di output, formattando il tutto. Ci sono poi altre funzioni "elementari", come i **count** sia per i Film che per le Persone, oppure le **search** per nome del film o nome della persona.
Queste ricerche vengono implementate all'interno delle due strutture dati, effettuano una visita per l'AVL o scorrendo la HashTable.

Un'altro compito di IMovidaDb è quello di avere la possibilità di Eliminare un film, quindi per la HashTable occorre calcolare l'hashcode e scorrere la lista e poi eliminare, mentre per l'AVL sono neccessarie le solite operazioni di ribllianciamento una volta trovato il nodo ed eliminato.

3 IMovidaSearch

IMovidaSearch si occupa di effettuare tutte quelle "query" sul database più complesse che una ricerca per chiave come in IMovidaDB.
Per le richieste che non richiedono ordinamento, si utilizza la funzione **getData()** della classe astratta Map (e quindi implementata nelle due strutture) che restituiscono un' ArrayList di tutti gli elementi presenti nella struttura, permettendoci quindi di effettuare una ricerca più specifica utilizzando le stringhe.
Per le richieste come **searchMostVotedMovies, searchMostRecentMovies** o **searchMostActiveActors** è necessario utilizzare algoritmi di ordinamento. Quindi, una volta visto quale algoritmo utilizzare, si accede alla istanza dell'algoritmo di ordinamento selezionato (Questo per non dover creare ogni volta un oggetto diverso) e passiamo alla funzione, oltre ai dati, un **Comparator** (definiti nella classe Comparator) che ci permette di effettuare l'ordinamento su valori diversi utilizzando la stessa funzione e i vari metodi **compareTo** da noi definiti.

4 IMovidaCollaborators

Questa parte ci ha richiesto di implementare un grafo di collaborazioni fra attori. Come detto in precedenza, la popolazione di questo grafo avviene alla lettura del file, creando una nuova **Collaboration** per ogni due attori che hanno lavorato allo stesso film. Si può pensare alla Collaboration come se fosse l'arco che collega due attori.
La prima funzione da implementare è stata **getDirectCollaboratorsOf**, che ritorna solo i collaboratori diretti, quindi dato un attore, dobbiamo semplicemente tornare (sotto forma di ArrayList) le collaborazioni adiacenti all'attore.
La seconda funzione invece, ci chiede di tornare tutto il team, quindi sia collaboratori diretti che non. Effettuiamo ciò eseguendo una semplice visita in ampiezza.
L'ultima funzione, la più complessa, è **maximizeCollaborationsInTheTeamOf** che, dato un attore, deve tornare una lista di collaborazioni che massimizzi lo score, ovvero il punteggio delle collaborazioni. Abbiamo risolto questo problema con l'algoritmo di Prim modificandolo in modo da ottenere un Maximum Spanning Tree, anziche un Minimum Spanning Tree.