TECNICHE DI PROGRAMMAZIONE

13/05/19  
Esercizio sui Grafi.

L’esercizio che faremo è un tema d’esame che ha creato molti problemi, non tanto per la query ma per la costruzione del grafo. Il testo dell’esercizio è il seguente:

Immagine che contiene screenshot

Descrizione generata automaticamente

Quindi abbiamo un insieme di aeroporti ed esiste un arco tra due aeroporti solo se c’è almeno un volo tra questi due aeroporti e la distanza media tra tutti i loro voli è almeno “x” miglia. Questo esercizio è un po’ più complicato in quanto il grafo non è orientato e quindi bisogna considerare tutti i voli da “A” a “B” ma anche quelli da “B” ad “A”.

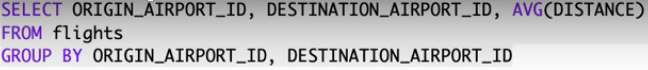
Carichiamo ed eseguiamo il database su Heidi e notiamo che la tabella dei vertici (aeroporti) ha 322 elementi; in questo caso si potrebbe usare come opzione la numero 1 (vedi lezioni precedenti), in cui eseguiamo un doppio “for” tra i vertici e chiediamo al DB se esista un arco. Quest’ opzione impiegherebbe all’incirca un quarto d’ora in quanto gli elementi non sono tantissimi.

Come prima cosa pensiamo alla query che potremmo usare in Heidi. Prendiamo la struttura della tabella “flights” e notiamo che quest’ultima ha un’ID del volo ma anche l’ID dei due aeroporti, quello di partenza e quello di arrivo.

Cosa vogliamo ottenere dalla query?

Mi aspetto di avere delle triple in cui i primi due campi rappresentino i due ID di un aeroporto, mentre l’ultimo rappresenti la distanza media. Così facendo, posso prendere i 2 ID, andare nel grafo e aggiungere un arco fra i 2 aeroporti con peso uguale al terzo campo.

Quindi nella nostra query avremo un “select” che mi estrarrà i due aeroporti e la distanza media dei voli tra di essi, nel “from” avrò la tabella “flights” e infine avrò una “group by” in cui raggruppo per i 2 aeroporti. Proprio grazie a quest’ultima “group by” riesco a calcolare la distanza media perché avrò per ogni coppia di aeroporti, tutti i voli presenti tra di essi.

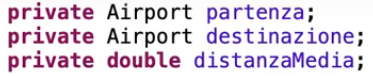


Il testo però mi richiede un controllo in più, cioè che la distanza sia maggiore di “x” miglia (“x” inserito dall’utente dall’interfaccia). Possiamo gestire questo problema aggiungendo la clausola “having” dopo il “group by”. Dobbiamo dare un nome alla media delle distanze sennò non possiamo usarla nel “having”.

La query finale sarà quindi la seguente:



Prendiamo quindi la nostra query e la copiamo nel nostro progetto di Eclipse. Da essa noi ricaviamo delle triple, le quali rappresentano la rotta presente tra un aeroporto “A” e un aeroporto “B”. Quindi dobbiamo creare una classe dentro il package “Model” che mi rappresenti queste informazioni (aeroporto partenza-aeroporto arrivo-distanza media). Decidiamo di chiamare questa classe “Rotta”. In questa classe aggiungiamo le 3 variabili che andranno a rappresentare le nostre triple ottenute dalla query. Per ogni variabile aggiungiamo i Getter() e Setter(), il Costruttore e infine l’HashCode e l’Equals. Per quest’ultimi dobbiamo analizzare quale sia la chiave primaria. Noi vogliamo due rotte uguali quando l’aeroporto di partenza e quello di destinazione sono uguali. Nella creazione dell’HashCode e dell’Equals togliamo come elemento la distanzaMedia. Le 3 variabili di questa nuova classe sono:



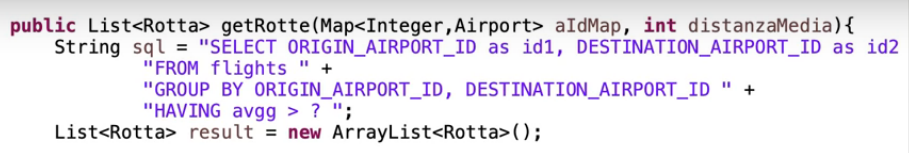
Dopo aver fatto questo, andiamo sul DAO e creiamo un metodo “getRotte” che ci permetta appunto di ottenere tutte le rotte. Questo metodo dovrà contenere quindi la nostra query (ricordarsi quando si passa da Heidi e Eclipse di modificare la query togliendo /r e /n alla fine e aggiungendo uno spazio prima di andare a capo) una lista di “Rotta” dove carichiamo tutti i risultati e infine il try-catch con il PreparedStatement, il while ecc…. (possiamo copiarlo da sopra e modificarlo). Dobbiamo gestire il punto di domanda della query e il while nel quale andiamo a creare una rotta e ad inserirla nella nostra lista “result”. Il metodo dovrà quindi essere così:

Immagine che contiene screenshot

Descrizione generata automaticamente

Dalla query mi ritornano 3 valori ma, per nostra sfortuna, sono degli “ID” degli aeroporti, mentre in “Rotta” ci siamo salvati degli oggetti “Airport”. Siamo quindi nel caso in cui una “Identity Map” è fondamentale. Ricordiamo che quest’ultima la usiamo quando noi abbiamo solo una parte dell’oggetto completo (nel nostro caso abbiamo solo l’ID) e vogliamo andare a recuperare tutto l’oggetto.

Nel nostro metodo quindi ora dovremo passare anche una mappa che chiamiamo “aIdMap” e nel “while” andiamo a recuperare gli oggetti “aeroporto partenza” e “aeroporto arrivo”. Come ultima cosa inseriamo un “if” per controllare se gli elementi sono presenti nella mappa e andiamo ad aggiungere questi elementi in “rotta”.



Primo cambiamento nel metodo.

Immagine che contiene screenshot

Descrizione generata automaticamente

Secondo cambiamento nel metodo.

Ora possiamo passare al Model. In esso avremo sicuramente la creazione del grafo che deve essere semplice, non orientato e pesato e in cui i vertici sono gli aeroporti e gli archi sono “DefaultWeightedEdge”. Inoltre, nel DAO, abbiamo usato l’Identity Map per gli aeroporti e quindi dovrò averla nel Model per poi poterla passare al DAO. Infine, nel costruttore, andiamo a creare questi due oggetti che sono ancora vuoti.

Immagine che contiene screenshot

Descrizione generata automaticamente

Andiamo successivamente ad implementare il metodo “creaGrafo()”. Se andiamo nel DAO notiamo che abbiamo già un metodo “loadAllAirlines” che mi restituisce tutti gli aeroporti ma non è strutturato per utilizzare un Identity Map. Devo quindi passare a questo metodo del DAO la mappa e prima di creare l’oggetto nel while, verifico di non avercelo già nella mappa. Nel caso in cui non fosse ancora presente nella mappa, creo l’oggetto e lo inserisco nella Mappa. Nel caso in cui invece fosse già presente, aggiungo direttamente l’aeroporto che ottengo dalla mappa alla lista.



Primo cambiamento nel metodo (aggiunta la mappa come parametro da passare).

Immagine che contiene screenshot

Descrizione generata automaticamente

Secondo cambiamento (controllo e aggiunta dell’oggetto alla mappa).

Torniamo ora nel metodo “creaGrafo” della classe Model. Come prima cosa vado a richiamare il metodo del DAO per ottenere tutti gli aeroporti in modo da popolare la mappa. Successivamente vado ad aggiungere i vertici così da poter recuperare un insieme di rotte tra di essi. Per ogni rotta, aggiungo l’arco che mi sono calcolato. Quindi mi basterebbe ciclare con un “for” su tutte le rotte e per ogni rotta aggiungere l’arco corrispondente con il peso e tutto. Questo però non possiamo farlo in questo esercizio in quanto noi dobbiamo modellare il grafo sia da “A” a “B” che da “B” ad “A”. Noi dobbiamo quindi controllare se è già presente un arco tra i due vertici (indipendentemente da qual è il nodo di sorgente e quello di destinazione) e nel caso in cui esista, aggiorno il peso. Quindi, se c’è già un arco tra “A” e “B” con distanza media uguale a 5, se io trovo un’altra rotta da “B” ad “A” con peso 10, devo fare la media con l’arco di prima e aggiornare il valore. Questa era la parte un po’ più difficile quindi bisogna guardare bene il codice per capirla.

Immagine che contiene screenshot

Descrizione generata automaticamente

Proviamo ora a testare il programma con la classe TestModel (già creata).

Immagine che contiene screenshot

Descrizione generata automaticamente

Come ultima cosa, abbiamo aggiunto un messaggio a video che mi permette di controllare quando è stato cambiato il peso.

Immagine che contiene screenshot

Descrizione generata automaticamente  
Proseguiamo ora con il punto “c” e il punto “d” dell’esercizio. Il nostro obiettivo ora è quello di verifica se è possibile raggiungere un aeroporto di arrivo da un aeroporto di partenza (scelti dall’utente). Per fare questo, devo prendere l’aeroporto di partenza e visitare tutti gli aeroporti collegati ad esso e vedere se tra essi c’è l’aeroporto di arrivo richiesto.

Andiamo quindi nel Model e creiamo un metodo Booleano chiamato “testConnessione” che riceve due ID, e mi dice “vero” o “falso” se i due aeroporti con quei due ID sono connessi oppure no. Noi abbiamo la mappa quindi possiamo velocemente recuperare gli aeroporti. Decidiamo poi di fare una Visita in ampiezza (non cambia niente se facciamo la Visita in profondità). Dobbiamo quindi creare un iteratore definito sul grafo e al quale dobbiamo definire il nodo di partenza.

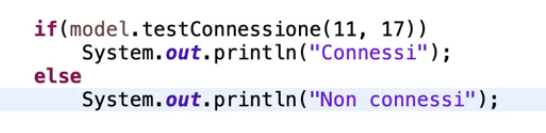
Ci serve una struttura dati per tenere traccia degli aeroporti che ho visitato durante il ciclo. Definisco quindi un “Set” di aeroporti e faccio un ciclo while e aggiungo ai miei visitati il “next”. Infine, per indicare se i 2 aeroporti sono connessi o no, vado a vedere nel “Set” se c’è la destinazione. Se così fosse, ritorno “true”, altrimenti ritorno “false”. Di solito per il punto “d” non servono tante linee di codice (vedi prossima immagine). Se vediamo che stiamo scrivendo tanto, vuol dire che c’è qualcosa che non va.

Immagine che contiene screenshot

Descrizione generata automaticamente

In molti esami viene chiesto di stampare l’elenco dei vicini di un determinato nodo (c’è una classe apposta che calcola quello e che vedremo la prossima lezione). In qualsiasi caso non servono più di 5/6 linee di codice.

Proviamo questo ultimo punto dell’esercizio. Andiamo nella classe testModel e proviamo il seguente esempio:



Come ultima cosa bisognare poter stampare un possibile percorso presente tra i due aeroporti scelti dall’utente. Per fare questo, torniamo nel Model e creiamo un nuovo metodo. Il nostro obiettivo è quello di agganciare dei “Listener” all’iteratore di una visita.

Creiamo quindi un metodo che ritorni una Lista che modelli il mio percorso. A questo metodo passiamo i 2 aeroporti “a1” e “a2”. Per salvarmi il percorso qua non uso un Set ma una Lista. Anche in questo metodo, come nel precedente, dobbiamo fare una visita in ampiezza quindi possiamo copiare il codice. L’unica differenza rispetto al metodo di prima è che ora al nostro iteratore, affianchiamo un “TraversalListener”.

Nella classe Model definisco una mappa “visita” e la vado a creare nel costruttore. Grazie a questa mappa noi potremmo modellare le relazioni Padre-Figlio tra i vertici. Tutte le volte che traversiamo un Edge, posso recuperare l’aeroporto di sorgente passandogli appunto l’arco attraversato. Lo stesso si può fare con l’aeroporto di arrivo.

Fatto ciò bisogna aggiungere il codice visto già l’altra volta con il professore. Se la mappa “visita” non contiene la destinazione ma contiene la sorgente, posso modellare la visita tra destinazione e sorgente. Essendo un grafo non orientato, devo gestire anche il caso opposto che è l’inverso del caso sopra citato.

Il codice per gestire tutto ciò che abbiamo detto è il seguente:



Mappa “visita”.

Immagine che contiene screenshot

Descrizione generata automaticamente

Metodo “trovaPercorso”.

Immagine che contiene screenshot

Descrizione generata automaticamente  
Metodo “edgeTraversed” di “TraversalListener”.

Non abbiamo ancora finito perché dobbiamo ancora visitare il grafo e, ogni volta che attraversiamo un arco, verrà chiamato il metodo “edgeTraversed”. Al fondo quindi aggiungiamo il while. Ci siamo inoltre dimenticati di indicare il vertice sorgente che andiamo ad aggiungere (ultima riga nella prossima immagine)

Immagine che contiene screenshot

Descrizione generata automaticamente

Nel while andiamo a definire una variabile “step” che ci consentirà di muoverci dal Figlio verso il Padre per potere risalire appunto al vertice sorgente. Quindi il codice sarà il seguente:

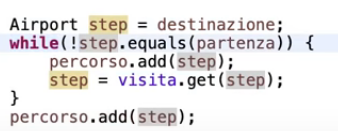
Immagine che contiene screenshot

Descrizione generata automaticamente

Come ultima cosa andiamo a provare il metodo nel TestModel per vedere se funziona il tutto (esempio con i vertici 11 e 297 che sappiamo sono connessi).



Runnando il programma notiamo che non ci viene indicato nel percorso (per rendere più chiara la visualizzazione abbiamo cambiato il metodo “toSting” nella classe “Airport” il modo tale da restituirci solo il codice dell’aereoporto) l’ultimo aeroporto cioè quello di partenza. Andiamo quindi ad aggiungerlo dopo il while.



FINE