TECNICHE DI PROGRAMMAZIONE

14/05/19

Cammini minimi nei grafi.

L’ultima lezione sui cammini minimi è terminata con il Lemma che ci diceva che se abbiamo trovato un cammino minimo tra “A” e “B”, allora possiamo conosciamo già tutti i cammini tra “A” e “C” (con “C” punto intermedio tra “A” e “B”).   
Noi avevamo fatto l’esempio di andare da Milano a Roma. Se il cammino minimo tra Milano e Roma passa per Genova, allora sicuramente il tratto Milano-Genova è un cammino minimo.

Da questo teorema possiamo estrarre anche un Corollario che ci dice che, preso un cammino minimo tra “s” (vertice sorgente) e “v” (vertice destinazione), se dividiamo questo cammino in “s-u” e “u-v” (ove “u-v” è l’ultimo arco), allora il peso del cammino minimo “s-v” è uguale al peso “s-u” (cammino minimo per il teorema precedente) più il peso dell’ultimo arco “u-v”.

Nel caso io non sapessi qual è il cammino minimo e prendo un qualunque “ u’ ” predecessore di “v”, dobbiamo rispettare la seguente uguaglianza:

Immagine che contiene oggetto

Descrizione generata automaticamente

Da questo corollario possiamo allora estrarre una relazione importante:

“Dati un arco ‘u-v’ con peso ‘w’, se noi andiamo a vedere il peso noto fino ad ora di ‘v’ e lo andiamo a confrontare con il peso noto fino ad ora di ‘u’ più il peso dell’arco che conduce ‘u’ a ‘v’. Se il primo è maggiore del secondo, allora il predecessore di ‘v’ sarà ‘u’ e il cammino minimo per arrivare a ‘v’ sarà uguale al cammino minimo per arrivare a ‘u’ più l’arco ‘u-v’ “.

Immagine che contiene orologio, oggetto

Descrizione generata automaticamente

Nell’esempio dell’immagine sopra, io sapevo che per arrivare a ‘u’ avevo un peso di 5, mentre per arrivare a ‘v’ avevo un peso di 9. Da questo deduco che tra ‘u’ e ‘v’ era presente un qualsiasi percorso che aveva peso 4. Successivamente troviamo un nuovo arco ‘u-v’ di peso 2 che, sommato al peso per arrivare al vertice ‘u’ mi dà un peso complessivo di 7. Questo nuovo arco andrà a sostituire il percorso precedente.

Nei successivi 15 minuti viene spiegato l’algoritmo di Dijkstra che dovremmo già conoscere con Ricerca Operativa (consiglio di vederlo da RO rispetto alle sue slide in quanto viene spiegato molto meglio).

L’unica cosa che non viene affrontato in Ricerca Operativa mentre qui sì, è la complessità dell’algoritmo di Dijkstra. La sua complessità è:

Immagine che contiene oggetto, orologio

Descrizione generata automaticamente

Ma nel caso di un grafo in cui il numero di archi diventa dominante si può arrivare fino a:

Immagine che contiene oggetto

Descrizione generata automaticamente

Dove si ricorda che “E” è il numero di archi mentre “V” è il numero di vertici del grafo.

Proviamo ad applicare questi algoritmi nell’esempio della metropolitana di Parigi. Logicamente questi algoritmi ci risultano utili nel momento in cui abbiamo grafi pesati perché con i grafi non pesati, se vogliamo trovare i cammini minimi, ci basta una visita in ampiezza.

Nel nostro esercizio, se andiamo ad analizzare la tabella Connessione, notiamo che essa non ci dice quanto impiega, però mi dice quale Linea passa per di lì e per ogni Linea viene descritta la sua velocità e la distanza. Posso quindi calcolare il tempo di percorrenza tra 2 fermate, conoscendo la linea che le collega, la velocità di quella linea e le coordinate geografiche delle 2 fermate.

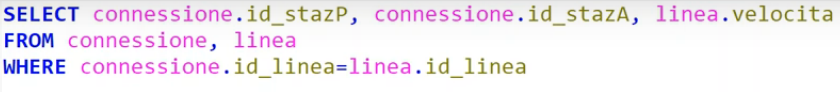
Andiamo a modificare il programma in modo tale da poter inserire questa nuova funzionalità.

Come prima cosa, il nostro grafo non sarà più solo “Semplice e Orientato”, ma sarà anche “Pesato”. Andiamo nel Model e cambiamo la definizione del nostro grafo:

Immagine che contiene screenshot

Descrizione generata automaticamente  
Per quanto riguarda la sua creazione, non modifichiamo niente ma aggiungiamo solo il peso agli archi. Con conoscere il loro peso dobbiamo impostare una query su Heidi che ci permetta di trovare quello minimo. L’informazione principale è nella tabella Connessioni, infatti ci vengono indicati tutti i collegamenti tra le fermate. Inoltre, ci viene anche indicata la linea con la quale avviene la connessione. Grazie a quest’informazione, possiamo fare un Join con la tabella Linea in modo da poter ottenere da quest’ultima la velocità di ogni Linea.

Il nostro obiettivo è quello di ottenere una tabellina con le 2 fermate e la velocità della linea. La query da scrivere è la seguente:



Mediante questa query è possibile che la stessa coppia di fermata compaia più di una volta. Dobbiamo ancora modificarla in modo da poter estrarre la velocità minima. Il modo migliore è quello di inserire una “Group By” nella raggruppiamo il risultato finale per stazione di partenza e stazione di arrivo e di estrarre la velocità massima dalla Select. Quindi:





Mediante questa query, ci rendiamo conto che ci restituisce già il peso degli archi del grafo. Possiamo quindi costruire un metodo del DAO che ci restituisce esattamente questa lista e poi nel Model usiamo i valori di questa lista per andare ad inserire i pesi degli archi.

Oltre a restituirci il peso, questa query ci dà proprio l’elenco di tutti gli archi. Possiamo quindi eliminare il pezzo di codice che avevamo prima che ci permetteva di creare gli archi e usare direttamente questa query sia per la creazione che per la definizione del peso di ogni arco.

Aggiungiamo un metodo nel DAO che ci restituisce una lista di oggetti che hanno “stazione di partenza, stazione di arrivo e velocità”. Dobbiamo quindi andare a creare una classe nuova nel package “Model” che chiamiamo “ConnessioneVelocita” e che contiene le informazioni sulla connessione. Inseriamo quindi le tre variabili con i loro Getter e Setter, il costruttore e l’hashCode e l’equals (su stazP e stazA).

Creata questa classe che mi definisce l’oggetto, ritorniamo nel DAO e creiamo il metodo. Quest’ultimo è molto simile agli altri metodi già presenti, infatti cambia praticamente solo il contenuto della stringa sql. Una volta creato il metodo (vedi immagine sotto), lo andiamo a richiamare dal Model per trovare il peso degli archi. Inoltre, nel Model dobbiamo ciclare per tutti gli archi con un “for” perché potremmo:

1. Aggiungere il peso ad un arco già esistente (“null” in caso di arco non esistente).
2. Aggiungere il peso dando la coppia di vertici.

Immagine che contiene screenshot

Descrizione generata automaticamente

Metodo getConnessioneVelocita del DAO.

Noi abbiamo deciso di definire il peso di ogni arco mediante la coppia di vertici (metodo 2). Il problema è che noi non ce l’abbiamo in quanto “ConnessioneVelocita” mi dà solamente l’ID. Per risolvere questo problema ricordiamoci che abbiamo sempre la Mappa! Essa ci permette in un secondo di passare dall’ ID all’oggetto.

“Cp” (vedi immagine sotto) mi arriva dal Database ed è la singola riga della query che ha un numero (stazione di partenza). Da lì ricavo l’oggetto Fermata in quanto tutte le fermate le ho già prese prima con il metodo “getAllFermate” e ho già costruito l’Identity Map.

Quando definiamo il peso di un arco, per noi è il tempo di percorrenza, mentre nell’oggetto “cp” ho la velocità. Dobbiamo fare quindi spazio diviso la velocità per trovare il tempo.

Creo quindi una variabile “distanza” che mi calcola lo spazio tra due Fermate mediante il metodo, visto qualche lezione fa, “LatLngTool”.

Il nostro metodo finale nel Model sarà quindi il seguente:

Immagine che contiene screenshot

Descrizione generata automaticamente

Se avessimo usato “Graphs.addEdgeWithVertices” potevo cancellare tutta la parte di codice che mi aggiungeva gli archi.

Immagine che contiene screenshot

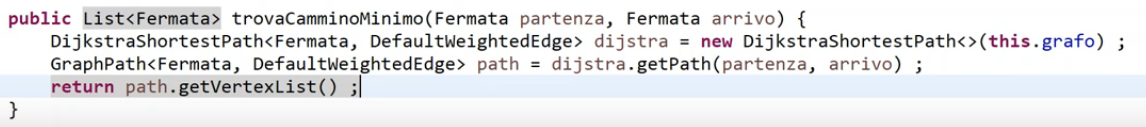
Descrizione generata automaticamente

Come ultima cosa, possiamo andare a gestire la pressione del bottone “Cerca”; prima ci dava il percorso con il minor numero di archi, mentre adesso ci deve restituire il percorso che impiega meno tempo.

Andiamo a calcolare il percorso minimo mediante l’algoritmo di Dijkstra. La classe di JgraphT che mi permette di implementare l’algoritmo di Dijkstra è “DijkstraShortestPath”.

Il professore consiglia di andare sul sito di JgraphT per vedere tutti i diversi metodi che si possono usare a partire da questa classe (alcuni molto utili).

Aggiungiamo il metodo “trovaCamminoMinimo” nel Model che mi restituisce una lista di Fermate a partire da una Fermata di partenza e una di arrivo. L’oggetto che mi viene restituito dal’algoritmo di Dijkstra è un “GraphPath”. Con il metodo “getVertexList” possiamo farci restituire tutti i vertici attraversati. Il metodo finale è quindi il seguente:



Come ultima cosa, andiamo nel Controller per gestire il bottone. Modifichiamo il codice sostituendo al metodo “fermateRaggiungibili” del Model il metodo “trovaCamminoMinimo”.





Nel Model sono ancora presenti degli errori in quanto bisogna sostituire tutti i “DefaultEdge” con “DefaultWeightedEdge”.

FINE