TECNICHE DI PROGRAMMAZIONE

21/05/19

Simulazione ad eventi: esercizio “GestioneMigrazioni”.

Tema d’esame relativo alle Simulazioni viste le scorse lezioni.

Gli esami si differenziano in due tipi principali:

1. Ricorsioni.
2. Simulazioni.

Il primo punto dell’esercizio è però comune a tutti e due gli esami. Su GitHub è presente già il progetto base dove è stato già svolto in primo punto in quanto non ci interessa.

Immagine che contiene interni

Descrizione generata automaticamente

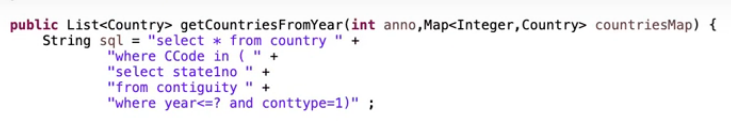
Il grafo ha come vertici i “Country” e come arco il fatto che i 2 “countries” in considerazione siano confinanti o no via terra (a partire dall’anno specificato dall’utente).

Vediamo la soluzione del primo punto aprendo la classe “Model” del nostro progetto. Abbiamo un grafo semplice, non orientato e non pesato, i cui vertici sono tutti gli stati (che ottengo dal DAO) a partire da un certo anno. Per l’aggiunta degli archi, ho un metodo del DAO che mi restituisce una serie di coppie di adiacenza. Abbiamo definito una classe “Adiacenza” in cui modello il fatto che due stati siano confinanti.

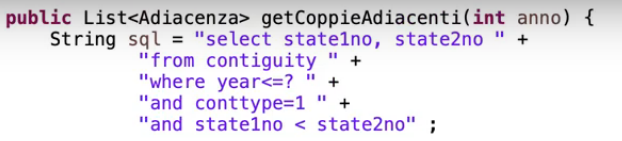
Immagine che contiene screenshot

Descrizione generata automaticamente

Andiamo ad analizzare la query nella classe “BordersDAO”. Essa ci permette di recuperare tutti i “country” per i quali c’è almeno un confine a partire dall’anno specificato. Grazie a questa query recupero i vertici.



Abbiamo un altro metodo del DAO chiamato “getCoppieAdiacenti” che va a riempire le adiacenze tra i vertici grazie alla tabella “contiguity”. Per ogni riga di questa tabella “contiguity” crea una nuova “Adiacenza” e questa lista di adiacenze viene poi usata per aggiungere l’arco.



Il punto “C”, richiedeva di stampare tutti i vertici con il numero di “country” confinanti. Il model ha il metodo “getCountryAndNumber” e anche qui mi sono definito una classe per salvare le informazioni “CountryAndNumber”. Per ogni vertice del grafo, creo un oggetto “CountryAndNumber” in cui il “country” è quello che sto analizzando e il numero associato è il grado di quel “country”. Termino il metodo con un “sort” per ordinare la lista (la classe “CountryAndNumber deve implementare Comparable e nel metodo “compareTo” ordino in modo decrescente di numero).

Immagine che contiene screenshot

Descrizione generata automaticamente

Quando ho più criteri di comparazione, non posso usare questa tecnica, ma devo andare a definire dinamicamente un comparatore quando faccio la “sort”.

Passiamo ora al punto 2 che è quello che ci interessa. Esso chiedeva di simulare la diffusione di 1000 immigrati che vengono inseriti in uno stato iniziale (scelto dall’utente).

Immagine che contiene screenshot

Descrizione generata automaticamente

La simulazione si esaurirà quando tutti saranno diventati stanziali e quindi questa è la nostra condizione di terminazione.

Immagine che contiene screenshot

Descrizione generata automaticamente

Cerchiamo di dividere bene il codice, quindi anche per le simulazioni è sempre bene avere una classe “Simulatore”. La andiamo a creare nel Package Model.

Partiamo con inserire in questa classe i valori di output che dovrà restituire. Il testo richiede di stampare il numero di passi che abbiamo impiegato per distribuire i migranti. Avremo una variabili intera “t”. Ci viene poi chiesto di stampare l’elenco degli stati con le eventuali persone divenute stanziali. Ciò si modella bene con una Mappa in cui la chiave è il “paese” mentre il valore sia un numero.

I parametri della simulazione. Sicuramente avremo il numero di migranti di cui andremo ad analizzare il flusso (1000 nel nostro caso). Abbiamo poi lo “stato di partenza” che possiamo sia definirlo come parametro della classe sia usarlo direttamente nei metodi.

Il nostro Modello (cioè lo stato del sistema ad ogni passo != classe Model) dovrà sicuramente avere un riferimento al grafo per poter sapere dove spostarci ad ogni passo (questo sarà sempre cosi all’esame; nel punto 1 si crea il grafo mentre nel punto 2 si utilizza).

Dobbiamo ancora ragionare sui tipi di evento ( i quali danno inizio alla coda prioritaria). Nel nostro caso avremo un solo tipo di evento in quanto l’unico evento da modellare è quello che succede al tempo T.

Gli eventi vanno modellati all’interno di una classe separata e quindi creo una nuova classe “Evento” all’interno sempre del package Model. La classe “Simulatore”, per adesso, si presenterà così:

Immagine che contiene screenshot

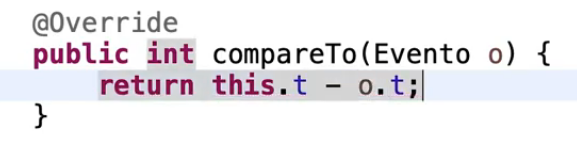
Descrizione generata automaticamente

Nel caso in cui ci fossero più tipi di eventi, nella classe “Evento” ci sarebbe un “enum” con i vari tipi di evento. In questo caso ne abbiamo solo 1 e quindi non dobbiamo differenziarli.

Il nostro Evento avrà un tempo “t” in cui si verifica, il numero di migranti che arrivano che sono arrivate e che si sposteranno “n” ed infine il paese (in cui le persone arrivano e da cui si sposteranno). Inoltre, questa classe avrà un costruttor e, dei Getter (dei Setter non ce ne facciamo niente) per tutti e 3 i campi e il Comparable. Quest’ultimo lo aggiungiamo in quanto il concetto chiave della simulazione è usare una Priority Queue in cui buttiamo tutti i nostri eventi e poi li estraiamo uno alla volta secondo un criterio.

Implementiamo quindi il Comparable alla classe Evento e aggiungo tutti i metodi non implementati (compareTo).





Il return nell’immagine ci permette di ordinare gli eventi in ordine crescente. Torno al Simulatore; abbiamo un solo tipo di evento ma dobbiamo ancora aggiungere la Pryority Queue. Definiti questi 4 punti (molto importante), restano da implementare 2 metodi, uno di inizializzazione del Simulatore e uno di Run.

Immagine che contiene screenshot

Descrizione generata automaticamente

Incominciamo definendo il metodo “init”. Esso riceverà come parametro il Country di partenza e il grafo (entrambi da settare). Successivamente avremo l’impostazione dello stato iniziale, in cui dovrò sicuramente impostare il tempo iniziale. Nel nostro caso lo modelliamo come interi (1,2,3,4…). Imposto il tempo iniziale uguale a 1. Dobbiamo creare la mappa “stanziali” e riempirla con il suo stato iniziale. Per ogni “country” presente nel grafo, lo andiamo ad aggiungere con il tempo uguale a 0. Ultima cosa da creare è la Priority Queue. Dopo aver impostato lo stato iniziale, inserisco il primo evento a cui devo passare il tempo, il numero di persone che arrivano e lo stato in cui arrivare. Nel nostro caso avremo T=1, N\_MIGRANTI=1000 e il Country=partenza. Il nostro metodo “init” quindi sarà:

Immagine che contiene screenshot

Descrizione generata automaticamente

Passiamo ora al metodo “run()”. Il pattern è sempre lo stesso: estraggo un evento per volta dalla coda e lo eseguo finché la coda non si svuota. Avremo quindi un while che andrà avanti fino a quando la coda non si svuota e nel quale eseguiremo l’evento si spostare i migranti negli stati confinanti. Ricordarsi di usare “poll” per estrarre un evento. Il numero di persone e lo stato li posso prendere dall’evento. Fatto questo, l’esercizio ci chiede di spostare il 50% delle persone in parti uguali negli stati confinanti. Dovremo quindi come prima cosa recuperare gli stati confinati allo stato da cui siamo partiti. Ora che abbiamo gli stati confinanti, dobbiamo calcolare quante persone spostare in ognuno di essi. Le persone che si spostano sono la metà di quelle che arrivano. Questa metà devo dividerla in parti uguali per il numero di stati confinanti. Siamo fortunati perché l’approssimazione per difetto la fa già la divisione. Se questo valore è maggiore di zero, riusciamo a spostare i migranti e posso aggiungere un evento per ogni stato confinante.

Calcoliamo successivamente le persone stanziali sottraendo al numero di persone totali i migranti totali. Andiamo così ad aggiornare il nostro stato indicando che ci sono solo più “stanziali” persone.

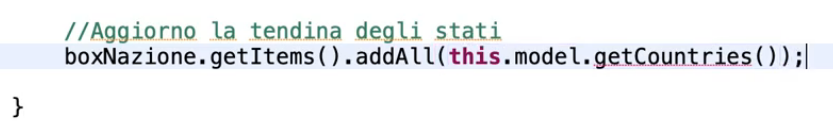
Come ultima cosa dobbiamo stampare il numero di passi T simulati e gli stati in cui vi sia almeno una persona stanziata. Per contare il numero di passi, mi basta tenerne traccia nel while attraverso un “e.getT().

Il metodo “run” finale è il seguente:

Immagine che contiene screenshot

Descrizione generata automaticamente

Passiamo ora al Controller. Facendo runnare il programma, dopo aver inserito un anno, riusciamo a calcolare i confini. Il problema è che, in base ai confini trovati, dovremmo inserirli nella tendina “nazione” per simulare. Andiamo quindi nel Controller e, dopo aver creato il grafo con il metodo “doCalcolaConfini”, aggiorno la tendina degli stati (prima di chiudere il metodo). Aggiungiamo un metodo “getCountries” del Model che andremo a gestire successivamente.



Andiamo quindi nel Model a gestire questo metodo che ci deve ritornare una Collection di “Country”. In questo metodo creiamo una lista temporanea di “Country” e la andiamo a riempire secondo i valori che ci sono nella Mappa. Abbiamo creato una lista così da poterla ordinare e avere i valori nella tendina in ordine alfabetico. Dobbiamo definire però il criterio di ordinamento, quindi andiamo nella classe “Country” e implementiamo il Comparable e andiamo a modificare il “compareTo” e il “toString” nel seguente modo:

Immagine che contiene screenshot

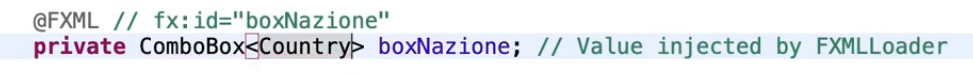
Descrizione generata automaticamente

Così facendo, ordiniamo i valori in ordine alfabetico crescente. Il metodo “getCountries” risulta quindi:

Immagine che contiene screenshot

Descrizione generata automaticamente

Come ultima cosa, dobbiamo dire al boxNazione (nel Controller) che conterrà dei “Country”.

  
Runnando il programma, notiamo che tutto funziona (la tendina è stata riempita correttamente).

Passiamo ora alla gestione del Bottone “doSimula”. La prima cosa da fare è andare a recuperare lo stato selezionato dall’utente nella tendina. Verifichiamo che l’utente ne abbia scelto uno attraverso un “if”.

Nel caso in cui l’utente abbia scelto uno Stato, possiamo simulare. Creiamo allora un metodo del Model che si chiama “simula”. È questo metodo che si occupa di creare il simulatore e di runnarlo. Quando questo metodo ritornerà qualcosa, posso chiedere al modello i risultati della simulazione.

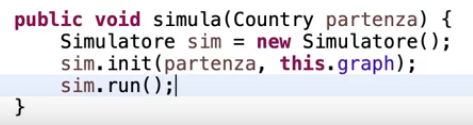
Suppongo che il modello mi restituisca sia l’ultimo passo che l’elenco degli stati con il numero di persone diventate stanziali. Dato che questo ultimo dato vogliamo ordinarlo, anche qui ci servirà una lista. Per fortuna, abbiamo una classe “CountryAndNumber” che contiene già queste informazioni e quindi ci basta avere una lista di “CountryAndNumber”. Il metodo “doSimula” conterrà quindi il seguente codice:

Immagine che contiene screenshot

Descrizione generata automaticamente

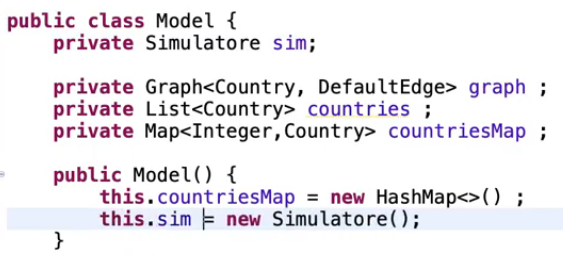
Non ci basta che definire i 3 metodi mancanti nel model.

Aggiungiamo il metodo “simula” all’interno del Model. Ora che siamo nel modello, posso creare il Simulatore e richiamare il metodo “init”. Notiamo però che questo ultimo metodo ha bisogno del vertice di partenza e del grafo, quindi dobbiamo passarli come parametri al metodo. Come ultima cosa, aggiungo il metodo “run”.

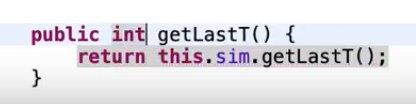


Ricordarsi di aggiungere il parametro anche nel controller.

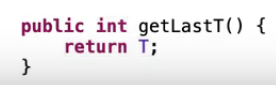
Andiamo ora ad aggiungere il metodo “getLastT”. Notiamo che il questa classe ci serve di nuovo il Simulatore. Noi però l’abbiamo stanziato nella classe “simula” e quindi non possiamo accederci. Definiamo allora il Simulatore come attributo del Model e lo creiamo nel costruttore del Model.



Logicamente nel metodo “simula” dobbiamo togliere la prima riga (definizione del Simulatore). Il metodo “getLastT” ha come obiettivo quello di ritornare il valore dell’ultimo passo.



Dobbiamo ancora creare il metodo “getLastT” del Simulatore!!



Come ultima cosa definisco il metodo “getStanziali”. In esso avrò sicuramente bisogno della Mappa “stanziali” presa dal Simulatore e di una lista sempre di “stanziali” in quanto ho bisogno di ordinare i valori. Scorriamo le chiavi e per ognuna di queste chiavi vado a prendere il numero e creo un oggetto “CountryAndNumber”.

Questo oggetto lo andiamo poi ad aggiungere nella nostra lista temporanea per poter ordinare il tutto. Per ordinarli uso il “compareTo” usato per ordinare il menu a tendina e come ultima cosa, ritorno la lista.

Il metodo finale “getStanziali” è quindi il seguente:

Immagine che contiene screenshot

Descrizione generata automaticamente

Runnando il programma, notiamo che ci vengono stampati anche gli stati per cui il numero di migranit=0. Questo a noi non interessa, quindi andiamo a gestirlo. Inoltre, andiamo ad aggiungere qualche “\n” per rendere il tutto più ordinato.

Immagine che contiene screenshot

Descrizione generata automaticamente

Facendo partire il programma notiamo che i numeri che escono fuori sono troppo piccoli. C’è un problema nella classe Simulatore. Nel codice che abbiamo scritto non aggiornavamo la mappa ma la sovrascrivevamo.

Immagine che contiene screenshot

Descrizione generata automaticamente

Bisogna modificare l’ultima riga come in figura. Non è solo “stanziali” ma sono quelli che c’erano prima più il nuovo valore di “stanziali”. Programma giusto!

FINE