TECNICHE DI PROGRAMMAZIONE

28/05/19

Simulazione ad eventi “Power Outages”.

In questo esercizio ci vengono date a disposizione 3 tabelle in cui vengono modellati i Nerc (fornitori di energia), le loro relazioni di confini (nercRelations) e le interruzioni di corrente (powerOutages). Il punto 1 è stato già svolto e chiedeva di costruire un grafo in cui i vertici erano i Nerc e gli archi erano le relazione di vicinanza. Il peso andava calcolato come il numero di mesi distinti (tenendo conto degli anni) in cui c’è stata almeno un’interruzione di corrente in ciascuno dei due Nerc considerati.

Nell’ultimo punto del 1 punto, bisognava popolare un menù a tendina con la lista di nerc. Al click del bottone “Visualizza vicini” bisognava stampare, per il Nerc selezionato dal menu a tendina, la lista dei vicini ed il peso dell’arco che li collega (ordinati per peso decrescente).

Nel punto 2 (quello che vedremo ora) bisogna effettuare la seguente simulazione:

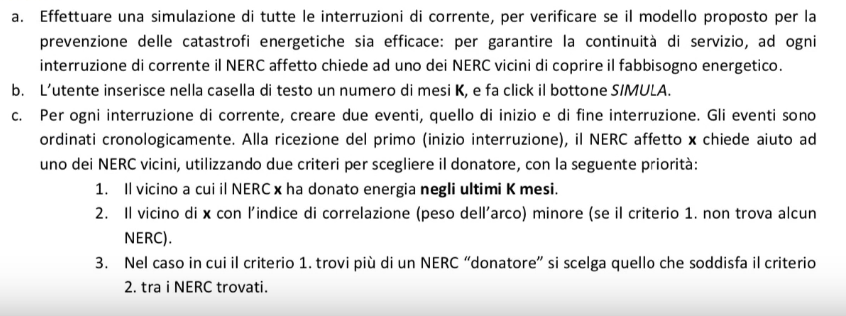


Immagine che contiene interni, screenshot, tavolo

Descrizione generata automaticamente

L’esercizio non è facilissimo e può essere risolto in molti modi differenti.

Supponiamo di avere un grafo semplice e pesato che contiene già tutto quello che ci serve. Esso ha i Nerc come vertici e gli indici di correlazione come peso degli archi.

Come prima cosa andiamo a creare nel package Model una classe per il Simulatore e una per l’Evento. Inoltre, vediamo che nella classe Nerc non vengono modellate le interruzioni di corrente. Dobbiamo quindi creare un Java bin e il metodo del DAO per recuperare tutte le interruzioni di corrente dal Database. Creiamo la classe “powerOutage” e ci chiediamo cosa dobbiamo recuperare dal DB. Ci servirà sicuramente un riferimento al Nerc e poi la data di inizio e la data di fine. Li andiamo a creare sotto forma di variabili e ci facciamo creare da eclipse tutti i Getter, i Setter, il costruttore,l’hashcode e l’equals. Essendo nell’ambito dei Database, di questi ultimi due ci basta mettere la chiave primaria (l’id).

Definiamo poi nel DAO un metodo che ci restituisce tutte le connessioni presenti. Questo metodo quindi ci restituirà una lista di oggetti Connessioni. Ci servirà la mappa perché vogliamo avere un riferimento ai Nerc e quindi gliela passiamo come parametro al metodo. Copiamo il codice dal metodo “loadAllNercs” e lo andiamo a modificare. Come prima cosa, modifichiamo la query in quanto dobbiamo andare a prendere l’id, il NercId e le due date. Nel while andiamo a creare un’ interruzione di corrente. Come ultima cosa, modifichiamo il return.

Immagine che contiene screenshot

Descrizione generata automaticamente

Ci spostiamo ora alla classe Simulatore. Come prima cosa, andiamo a definire il modello/stato del sistema. Dovremo ricevere un riferimento al nostro grafo, tutte le interruzioni di corrente che vogliamo simulare e una mappa “prestiti” in cui la chiave sia un Nerc e il valore sia un elenco di vicini a cui ho prestato energia (i miei futuri debitori). Come parametro avrò i mesi “k” che saranno quelli inseriti dall’utente e su cui noi dobbiamo andare a simulare. Come valori in Output abbiamo il numero di catastrofi e una mappa contenente tutti i bonus ottenuti dai Nerc che hanno imprestato energia. Infine, abbiamo la coda che sarà una “priorityqueue”.

Immagine che contiene screenshot

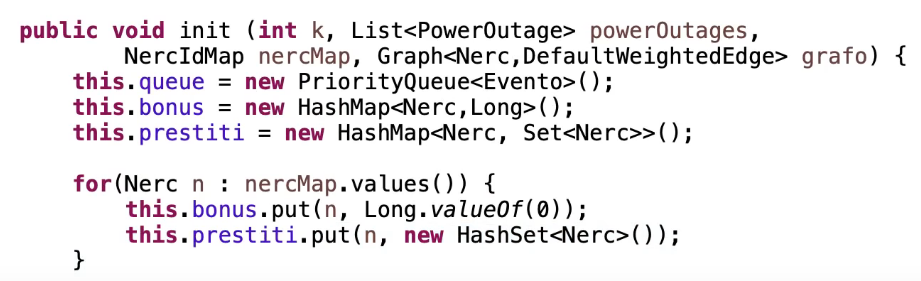
Descrizione generata automaticamente

Dobbiamo inoltre gestire il vincolo per cui un Nerc può aiutare un solo altro Nerc alla volta. Si potrebbe salvare questa informazione direttamente nel Nerc. Andiamo quindi a modificare la classe Nerc aggiungendo un booleano “staPrestando”. Esso per default è impostato a false. Andiamo ad aggiungere questa nuova variabile nel costruttore e aggiungiamo i Getter e i Setter.

Tornando al Simulatore, andiamo ad aggiungere i due metodi “init” e “run”.

Il primo riceverà dall’esterno il parametro “k”, la lista di PowerOutages”, l’identityMap dei Nerc (forse la utilizzeremo) e il riferimento al grafo. Creiamo successivamente la coda e le due mappe (quella per i bonus e quella dei prestiti) e le impostiamo con i loro valori iniziali. Per quanto riguarda la mappa dei prestiti, non ci basta fare la new della mappa, ma per ogni campo della mappa dobbiamo fare la new del Set. Aggiungiamo quindi un “for” per tutti i Nerc (posso prenderli dall’identityMap) e metto il loro bonus uguale a 0. Questo passaggio è molto importante sennò non funziona niente.

Per adesso quindi il metodo è il seguente:



Impostiamo ora il numero di catastrofi uguale a 0 e riceviamo i vari parametri (k, le interruzioni di corrente e il grafo). Fatto questo, possiamo incominciare a ragionare sugli eventi.

Inserisco nella coda tutti gli eventi di inizio interruzione. Questo perché allo scatenarsi di ogni evento, devo vedere se c’è un vicino disponibile. Nel caso in cui non ci sia, c’è una catastrofe, nel caso in cui c’è devo schedulare l’evento di fine (ho trovato qualche Nerc che mi aiuta).

Nella classe Evento, io mi immagino 3 tipi di evento:

1. Un evento che schedula l’inizio dell’interruzione.
2. Un evento che schedula la fine dell’interruzione.
3. Un evento che la cancellazione del prestito.

Abbiamo poi bisogno del Nerc a cui è relativo l’evento, del Nerc donatore, il tipo di evento, la data esatta in cui si verifica l’evento e 2 riferimenti alle due date. La prima data sarà quella usata per ordinare la coda, gli altri due saranno dei riferimenti che usiamo nel bisogno. La classe “Evento” quindi per ora si presenta così:

Immagine che contiene screenshot

Descrizione generata automaticamente

Mi faccio generare da Eclipse i Getter/Setter e il costruttore. Un’altra cosa molto importante è che questa classe deve implementare “Comparable” per poter essere inserita nella coda prioritaria. Nel metodo “compareTo” usiamo la data per ordinare gli eventi.

Torniamo quindi al Simulatore e inserisco gli eventi iniziali. Aggiungiamo un “for” che mi scorre tutti le connessioni e mi creo un Evento per ogni connessione (attenzione all’ordine dei parametri). Una volta creato l’evento, lo aggiungo alla mia coda.

Immagine che contiene screenshot

Descrizione generata automaticamente

Possiamo ora passare al metodo “run”. La struttura è sempre la stessa; vado a prendere l’evento dalla coda e continuo fino a quando la coda non mi restituisce “null”. In questo caso dobbiamo aggiungere uno switch-case per controllare il tipo di evento che stiamo analizzando. Abbiamo tre casi:

1. Inizio interruzione: il Nerc che ha subito l’interruzione lo recupero dall’evento. Cerco poi se c’è un donatore. In caso negativo, indichiamo che c’è una Catastrofe. Definisco quindi una variabile “donatore” uguale a null e in cui andrò ad inserire il donatore nel caso in cui esso sia presente. Nel caso in cui sia presente un donatore, dobbiamo dire che sta imprestando e che quindi non può essere scelto fino a quando non ha finito di imprestare. Dopo di che andiamo a creare l’evento “fine” in cui i campi rimangono gli stessi tranne che per il “tipo di evento” e per la data di fine. Come ultima cosa, lo aggiungo alla coda. Dobbiamo poi andare a tenere traccia del prestito e quindi dobbiamo andare ad inserire nella nostro mappa “prestiti” il Nerc a cui il donatore ha donato. Bisogna anche aggiungere l’evento “cancella” per cui questo prestito verrà cancellato. I parametri di questo evento saranno gli stessi tranne che per la data alla quale dobbiamo aggiungere i k mesi. Anche questo evento va aggiunto alla coda. Quale debitore scegliamo tra i tanti? Creo una variabile “min” che avrà un numero enorme, scorro il Set e, ogni volta che trovo un vicino con un grado di correlazione minore, sovrascrivo il valore e mi salvo il riferimento. Nel caso in cui non ci siano debitori, prendo i vicini. Mi faccio dare la lista di vicini e uso lo stesso codice usato prima. L’unica differenza sta nello scorrere i vicini e non i prestiti.

Immagine che contiene screenshot, testo

Descrizione generata automaticamente

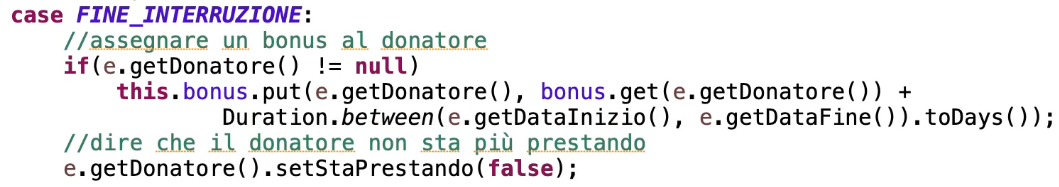
Immagine che contiene screenshot

Descrizione generata automaticamente

Immagine che contiene screenshot

Descrizione generata automaticamente

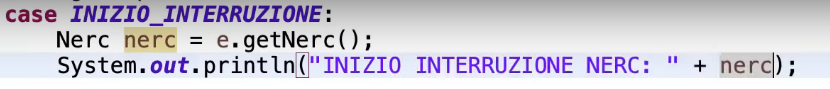
1. Fine interruzione: dobbiamo assegnare un bonus al Nerc e dire che il donatore non sta più imprestando. Per il bonus abbiamo la mappa e possiamo andare a vedere se nell’ evento c’è un donatore. In caso positivo, gli diamo il bonus corrispondente. Successivamente impostiamo il campo del donatore “staImprenstando” uguale a false.



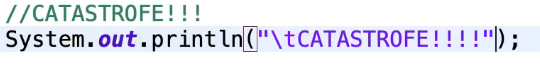
1. Cancella prestito: abbiamo la tabella “prestiti”. Possiamo quindi rimuovere dal Set dai prestiti del donatore il suo Nerc. Così facendo, rimuoviamo la sua traccia del prestito.



Il programma è terminato e bisogna solo più aggiungere qualche stampa per controllare i risulatati.



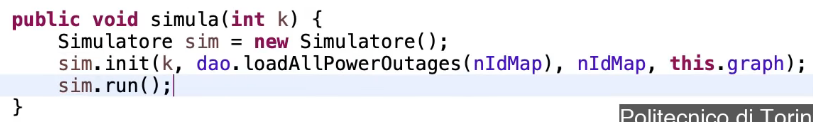








Ci siamo dimenticati di collegare il Simulatore al Model, quindi andiamo a creare un metodo “simula” nel Model in cui creiamo il Simulatore e chiamiamo il metodo “init” e “run”.



Definiamo il “TestModel” (con il main) in cui andiamo a definire il nostro modello, creiamo il grafo, e chiamiamo il metodo “simula” appena creato.

C’è qualche errorino che l’esercitatore correggerà a casa. Soluzione giusta come sempre su GitHub.

FINE.