TECNICHE DI PROGRAMMAZIONE

04/06/19

Tema d’esame “UfoSightings” (parte 2).

Come prima cosa, dobbiamo modificare una parte del codice fatta la volta precedente in quanto c’è un errore. Nel Controller, abbiamo inserito il codice per la creazione del grafo nel bottone sbagliato. Dobbiamo quindi spostare il codice inserito nel metodo “handleAnalizza” nel metodo “handleAvvistamenti”.

Proseguiamo con lo svolgimento del punto “C” del problema:

Immagine che contiene oggetto

Descrizione generata automaticamente

Per ottenere tutti gli stati raggiungibili da un determinato nodo, abbiamo diverse alternative tra cui una visita in profondità o una visita in ampiezza. Noi abbiamo scelto la visita in profondità; andiamo a definire l’iteratore (itererà su delle stringhe e gli archi sono dei “DefaultEdge”). Esso sarà sicuramente sul grafo a partire dallo stato selezionato. Successivamente andiamo a scorrere l’iteratore con un “while” e grazie ai mteodi “hasNext” e “next” andiamo ad aggiungere i risultati nella lista. Il risultato sarà quindi:

Immagine che contiene screenshot

Descrizione generata automaticamente

Abbiamo fatto un “next” fuori dal while per eliminare il primo stato tra i risultati (lo stato da cui partiamo). Come ultima cosa, ritorniamo dal metodo la lista di stati “raggiungibili”.

Ora abbiamo i 3 metodi finali:

Immagine che contiene screenshot

Descrizione generata automaticamente

Possiamo andarli a inserire nel metodo del bottone “handleAnalizza”. Come prima cosa andiamo a recuperare lo stato dal Box. Se lo stato è “null” restituiamo l’errore e ritorniamo. In caso contrario chiediamo al Model di dirci tutte le cose. Creiamo quindi una lista di predecessori , una di successori e una di raggiungibili. Per ognuna di esse, chiamiamo il metodo del Model corrispondente (getPredecessori….).

Fatto questo, andiamo a stampare i risultati.

Immagine che contiene screenshot

Descrizione generata automaticamente

Possiamo passare ora al secondo punto del punto “C”. Questo punto richiede la ricorsione la quale è più veloce da scrivere rispetto alla simulazione, ma è più difficile da impostare e da capire.

Per prima cosa dobbiamo pensare a quale sia a soluzione completa che vogliamo ottenere. Noi vogliamo una lista di stati (stringhe). Quindi come prima cosa definiamo la struttura dati che dovrà contenere il nostro percorso ottimo (la definiamo nella classe Model). Questa lista conterrà di sicuro lo stato iniziale e un insieme di altri stati. Abbiamo però un vincolo sugli altri stati infatti non vogliamo che siano ripetuti.

Quindi nel Model avremo un metodo public che mi restituisce una lista di stringhe e che mi permette di ottenere il percorso massimo. Questo metodo restituirà la lista ottima.

Dobbiamo poi pensare alla nostra soluzione parziale. Sarà sempre una lista di stringhe che però non definisco globalmente ma dentro al metodo. Successivamente la passerò alla procedura ricorsiva che la utilizzerà per costruire i vari percorsi.

Quando sarò arrivato ad un determinato nodo e non avrò più successori, avrò raggiunto la condizione di terminazione.

Come faccio a generare una nuova soluzione a partire da una soluzione parziale? Dato l’ultimo nodo inserito nella soluzione parziale, considero tutti i successori di quel nodo che non ho ancora considerato.

Alla fine, ritornerò una sola soluzione 🡪 quella per cui la “size” è massima. Il livello di ricorsione è uguale alla lunghezza della soluzione parziale.

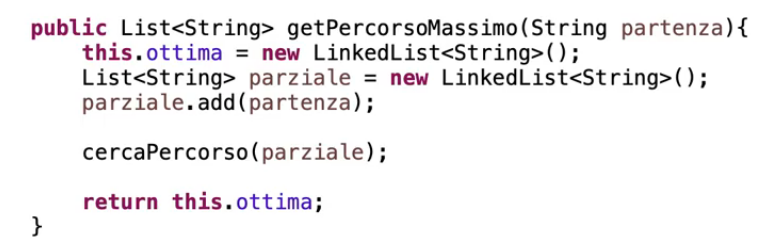
La condizione inziale è quella per cui la lista parziale contiene il mio stato di partenza.

Tenuto a mente di tutti questi punti, posso iniziare a creare il codice.

**Bisogna sempre scrivere nella ricorsione tutti questi punti in modo da capire bene cosa dobbiamo gestire.**

Ritorniamo al metodo esterno che verrà richiamato dalla procedura ricorsiva. Alla lista parziale dovrò aggiungere il mio stato di partenza passato come parametro dall’utente. Fatto questo, l’idea della ricorsione è quello di creare un metodo a cui passo parziale e lui faccio tutto in modo ricorsivo.

Il metodo che andrà a chiamare la procedura ricorsiva sarà quindi il seguente:



Posso creare ora la funzione ricorsiva “cercaPercorso” e qui gestiamo la ricorsione vera e propria.

Per prima cosa controlliamo se la soluzione corrente (parziale) è migliore di quella ottima corrente. Se la dimensione della lista parziale è maggiore di quella della lista ottima, posso sostituire alla soluzione ottima quella parziale (che diventa quindi quella ottima). Attenzione che non mi basta mettere l’uguale ma devo clonare la lista.

Fatto questo vado a selezionare i candidati del livello successivo, cioè tutti i successori del nodo corrente. Per far questo ci aiutiamo col metodo “getSuccessori” e come stato inseriamo l’ultimo elemento della lista parziale. Per ora il metodo si presenta così:

Immagine che contiene screenshot

Descrizione generata automaticamente

Trovati i candidati, li provo ad uno ad uno aggiungendoli alla lista parziale, lancio la ricorsione e poi lo tolgo (per poter provare gli altri). Andiamo a gestire questa parte mediante un “for” in cui scorriamo tutti i candidati e un “if” che controlla che il candidato non sia già presente nella listsa parziale.

Nel nostro caso, il “for” e l’if insieme, modellano implicitamente la condizione di terminazione perché se non ci sono più candidati, non entriamo più nel “for” e non continuiamo la ricorsione.

Se troviamo un candidato che non ho ancora considerato, lo aggiungo alla lista parziale, lancio la ricorsione con la nuova soluzione parziale e poi lo tolgo (voglio provare tutti i candidati).

Essendo una lista, per rimuovere l’ultimo elemento, devo dargli l’indice di questo elemento e lo posso fare con il metodo “size”. Il metodo ricorsivo è terminato qui.

Immagine che contiene screenshot

Descrizione generata automaticamenteColleghiamo al Controller mediante il bottone “handleSequenza”. Per recuperare il primo stato scelto dall’utente possiamo copiare il codice del metodo “handleAnalizza”. Ci facciamo dare poi dal Model il percorso massimo utilizzando il metodo “getPercorsoMassimo” creato prima (dobbiamo passargli lo stato).

Come ultima cosa stampiamo il risultato finale.

Immagine che contiene screenshot

Descrizione generata automaticamente

FINE.