TECNICHE DI PROGRAMMAZIONE

01/04/19

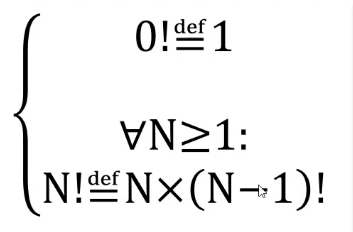
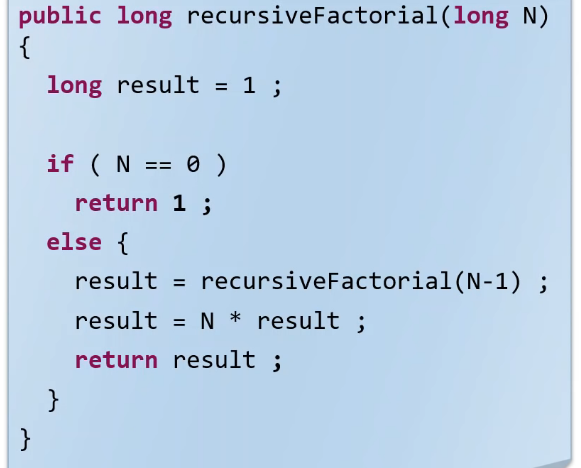
LA RICORSIONE è un metodo che ci permette di fare un salto in avanti nei tipi di algoritmi che realizzeremo. Andiamo quindi a decidere. Si basa sul dividere il problema principale in tanti piccoli problemi che a loro volta possono essere divisi in altri problemini.

La Ricorsione ci permette di elencare sistematicamente un insieme molto ampio di soluzioni.

Si parla di ricorsione ogni qual volta io ho un metodo o una funzione di una classe che richiama sé stessa, cioè all’interno del corpo del metodo c’è una chiamata del metodo stesso. Questo può avvenire direttamente o indirettamente (un metodo chiama un metodo che chiama un altro metodo che alla fine chiama il primo).

Questo meccanismo però va controllato perché sennò potrebbe venirsi a creare un metodo infinito.

Il primo esempio che si fa spesso per ribadire un concetto è quello del fattoriale di una funzione. Deriva dal fatto che in matematica il fattoriale è definito come il numero di cui si vuole fare il fattoriale moltiplicato per il fattoriale del numero precedente.

** **

Quindi ho espresso un problema in termini di sé stesso ma più facile. Saprei risolvere N se sapessi risolvere N-1 e così via. Quindi ho bisogno di diverse cose:

1. Una regola di ricorrenza che mi permette di descrivere un problema in termini dello stesso problema più piccolo.
2. Condizione di terminazione che sarebbe qualcosa che mi dica quando è che il problema è abbastanza piccolo da poter essere risolto solamente guardandolo in faccia.

Nella ricorsione, un problema lo divido e lo ricombino. Questa operazione viene ripetuta tante volte, ogni volta che viene richiamato un livello successivo della ricorsione. Ogni livello deve affrontare un problema più semplice di quello precedente.

Ma che cos’è che risolve ogni livello della ricorsione? Devo avere un parametro, una variabile, un campo, un valore, un passaggio che viene risolto a ciascun livello.

ESEMPIO 1 ESERCIZIO DELLA SCHEDINA DEL TOTOCALCIO (espandere la schedina).

Attraverso il primo livello per esempio noi vogliamo che venga risolta la prima riga. Avere risolto la prima riga non vuol dire, logicamente, aver risolto tutta la schedina, perché poi devo vedere tutto le combinazioni con le altre righe. Però intanto ho già risolto una parte della schedina.

Il secondo livello si occuperà quindi di risolvere la seconda riga della schedina. Questo livello si vedrà già la prima riga della schedina risolta, pensa alle alternative possibili sulla seconda riga e se ne frega della 3 riga perché non è un problema suo.

Quindi ogni livello pensa alla propria riga e se ne frega di tutte le altre righe.

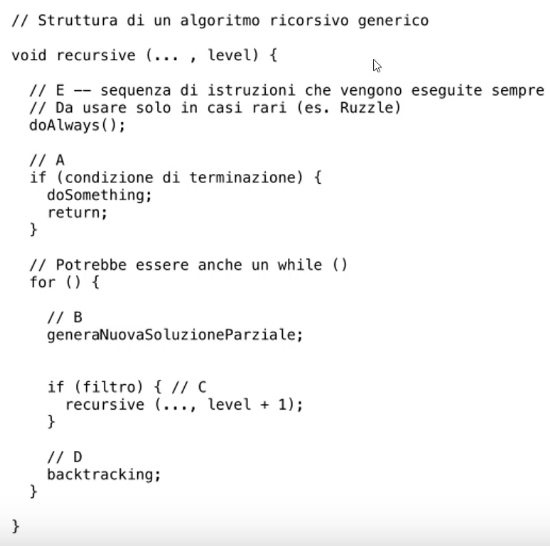
La cosa più difficile nella ricorsione sarà proprio quella di immaginarsi come risolvere un determinato problema per gradi.

Soluzione parziale si ha quando vengono risolti solo alcuni livelli, mentre se risolviamo tutti i livelli del problema avremo una soluzione totale.

Come faccio dalla soluzione parziale “i” a costruire tutte le soluzioni parziali del livello “i+1”?

Nel nostro caso della schedina, il problema è facile in quanto non c’è dipendenza tra i vari livelli, ma più avanti ci può essere e le cose si complicano.

Lo schema della ricorsione è sempre lo stesso, non ci sono molte cose da scrivere; il problema è che ogni riga ha un grande impatto in tutto il codice.



Ho una chiamata ricorsiva che ha una serie di parametri e poi un livello che sia esplicito. Questa variabile deve essere molto chiara perché sarà quella che verrà incrementata o decrementata.

Il passo ricorsivo di solito è un ciclo in cui si generano delle soluzioni parziali e su ciascuna soluzione parziale si chiama ricorsivamente il problema. Dopodiché c’è un passo di backtracking che dopo che io ho modificato la soluzione per crearne una nuova parziale, rimette le cose a posto (ho aggiunto un elemento, lo tolgo prima di provare ad aggiungerne altri; tipo la gomma del sudoku).

Poi ho il passo di terminazione che dice che, se il problema ha caratteristiche di essere un problema terminale, allora deve gestirlo facendo qualcosa.

Quindi i 3-4 passi fondamentali sono:

1. Generazione
2. Ricorsione
3. Backtracking
4. Terminazione

Poi abbiamo anche un filtro che ci permette di non provare tutte le soluzioni che creiamo perché alcune non sono valide.

Ci son o dei casi anche in cui tutte le soluzioni parziali sono utili (per esempio in ruzzle quando trovi una parola da 3 lettere, potrebbe anche essere allungata per trovarne un’altra). Quindi è una soluzione che è sia una soluzione completa che una soluzione parziale, e quindi noi dobbiamo trattarla nei due casi.

Proprio per questo motivo il blocco E che gestisce questo caso si trova prima del blocco di terminazione, perché il risultato trovato non è il risultato che fa terminare tutto il programma ma solo una soluzione completa che però potrebbe essere anche parziale.

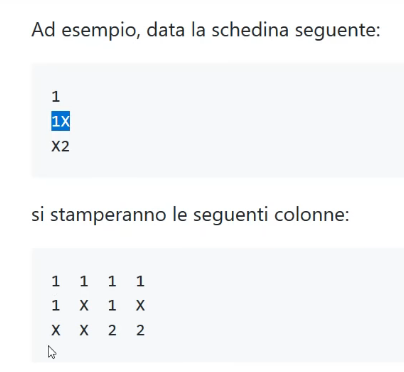
Logicamente, per ogni problema che affronteremo, lo schema andrà modificato aggiungendo o rimovendo alcune parti.

Le soluzioni potrebbero essere delle liste, degli array, sei set, delle mappe, dei grafi, dei percorsi…

Noi dobbiamo cercare di ricondurre il nostro problema a queste categorie.

Noi abbiamo già delle strutture dati per gestire i dati nel caso dell’esercizio sul totocalcio. Infatti nel progetto di GitHub abbiamo una “Enum” che si chiama risultato che contiene 3 costanti (1,2, X) e una classe schedina che sarebbe la colonna del risultato che altro non è che una lista di risultati di lunghezza N.

Quindi aprendo il progetto su Eclipse noi avremo una classe di tipo Enum che serve appunto per raggruppare una serie di costanti.



Quindi ogni colonna di quella matrice in figura è un oggetto di tipo “Schedina” il cui attributo “colonna” è una lista di 3 risultati. Quindi la nostra soluzione finale sarà una lista di “Schedine”.

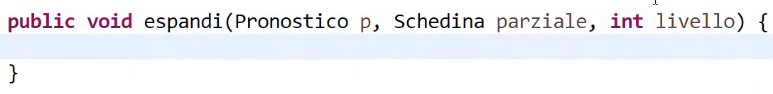
La classe “Previsione” invece è un insieme di valori, di risultati possibili.

Il “Pronostico” è ciò che l’utente sceglie, seleziona, è una colonna fatta di “Previsioni”, dove la previsione potrà essere un risultato singolo, doppio o triplo.

Noi ora creiamo una nuova classe “Espandi” che conterrà il metodo di espansione.

Specifichiamo cosa è un livello: singola partita (livello=0 🡪 schedina con 0 risultati).

Quindi noi abbiamo la nostra procedura (metodo) “espandi” che riceverà in ingresso un problema da risolvere ma parzialmente risolto. Ma il problema da risolvere qua è? La classe Pronostico, che ci dice cosa ha messo l’utente. Poi riceverà una schedina già parzialmente compilata, che è la soluzione parziale a cui sono arrivato fino ad ora. Infine, avremo il livello a cui sono arrivato. Quindi il metodo all’inizio di presenterà cosi:



Quindi per esempio questo mi sta dicendo: “questo è il problema da risolvere, tu sei al livello 5, io ti passo una schedina che ha già 4 previsioni”. Infatti, se sono al livello 5 devo produrre la quinta previsione.

La lista “parziale” contiene già ‘livello’ valor che sono messi nelle posizioni da 0…livello-1, io devo mettere il ‘livello’ valore, sulla base del pronostico corrispondente.

Quindi al livello 0 riempirò la casella 0 prendendo la previsione 0.

Al livello 1 riempirò la casella 1 espandendo la previsione 1 e così via.

Se devo mettere a posto la posizione 5, vado a vedere quali sono le previsioni che devo considerare (dove ‘p’ è il pronostico che contiene tutte le righe) e prendo la riga su cui devo lavorare io che è quella indicata dal mio livello.



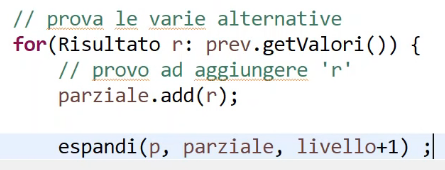
Quali sono i tentativi che posso fare a questo livello? Sono tutti i valori presenti nel Set “Risultato”.

Quindi io posso provare le varie alternative semplicemente iterando tra i risultati che ottengo dalla previsione.

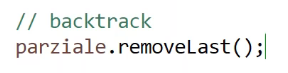
getValori() mi dà il Set che sono un Set di risultati. Io itero su questo Set estraendo un risultato per volta. Quindi se su quella riga c’è scritto “1, X”, questo ciclo verrà eseguito 2 volte, la prima volta con risultato 1 e la seconda volta con risultato X.

Ora provo ad aggiungere ‘r’ (vedi immagine successiva) alla soluzione parziale. Avevo una soluzione parziale che aveva solo 4 elementi al 4 livello, ora ne avrà 5.

Io ora ho finito ciò che dovevo fare al mio livello perché il mio scopo era appunto aggiungere un elemento in questa posizione. Quindi posso andare avanti e fare la ricorsione richiamando il metodo con lo stesso problema, la stessa schedina parziale, ma al livello+1.

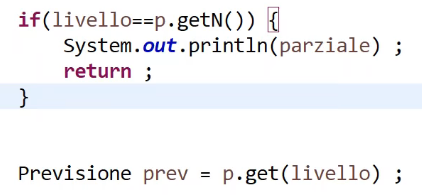


Ovviamente dopo che ho aggiunto un elemento, devo toglierlo (backtracking). Se aggiungo 1, se voglio aggiungere X non posso aggiungerlo dopo l’1, devo prima toglierlo e poi aggiungo X.



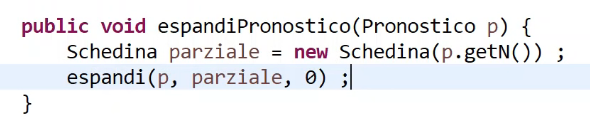
Finito qua quello che devo fare. Se il pronostico ha 13 righe, il posso fare la p.get(12) , di 13 non va bene. Quindi prima del get impongo che, se il livello fosse già arrivato alla fine del vettore p, (pronostico grande 13 e il mio livello è 13) la mia soluzione parziale è in realtà una soluzione completa e sono arrivato in fondo.

Noi non sapendo cosa fare decidiamo solo di stampare la soluzione finale.



Come ultima cosa creiamo un’altra funzione prima di questa che abbiamo creato ora e rendiamo quella di prima privata. Questa nuova funzione che chiamiamo “espandiPronostico”, riceve come parametro solamente il pronostico. Questa classe creerà una nuova schedina con dimensione massima pari alla dimensione massima del pronostico e chiamerà la funzione “espandi” che è ora privata passandogli pronostico p, la soluzione parziale che ho appena creato e 0 come livello.

Quindi la funzione ricorsiva che ha bisogno di tutti i suoi parametri per poter lavorare ai livelli, viene chiamata da una funzione mia che ha un’interfaccia grafica più semplice. Chi chiama questa funzione non deve capire nulla.



Ora come ultima cosa proviamo a vedere se il programma funziona. Nella classe TestPronostico, creiamo una classe Espandi al suo interno e chiediamo a questa classe di espandere il pronostico p.

