Corso di Programmazione 1

Ottava Esercitazione di Laboratorio

Osservazione preliminare: In tutti gli esercizi seguenti, tranne magari l'esercizio 4 e l'esercizio 5, suddividere il programma in due classi: `Programma` e `Metodi`. La classe `Programma` conterrà solamente il metodo main(), mentre la classe `Metodi` conterrà tutti gli altri metodi di supporto.

Esercizio 1 (Array parzialmente riempito)

Scrivere un programma Java che consenta all'utente di effettuare operazioni su un array di interi parzialmente riempito. L'array può contenere da zero a un massimo di 100 elementi interi; il numero di interi contenuto in un certo momento è indicato dalla variabile intera `numeroElementi`. Le operazioni sono implementate come metodi della classe `Metodi`, mentre il metodo main() della classe `Programma` viene utilizzato per testare tali operazioni.
Le operazioni da implementare nella classe `Metodi` sono le seguenti:

- creaArray(): alloca un array di 100 elementi interi, e ne restituisce un riferimento al chiamante;
- aggiungiElemento(array,numeroElementi,elemento): aggiunge all'array specificato (contenente attualmente `numeroElementi` valori, presenti dall'indice 0 all'indice `numeroElementi` – 1) l'elemento specificato, se non c'è già, e in tal caso restituisce il valore booleano `true` al chiamante. Se l'elemento è già presente nell'array, lascia l'array inalterato e restituisce `false`. Si noti che se `numeroElementi` vale 100 non è possibile inserire l'elemento specificato, anche se non è presente nell'array: in tal caso lasciare l'array inalterato e restituire `false` al chiamante;
- rimuoviElemento(array,numeroElementi,elemento): rimuove dall'array, contenente `numeroElementi` valori, l'elemento specificato, se c'è, e in tal caso restituisce il valore booleano `true` al chiamante. Se l'elemento non è presente nell'array, lascia l'array inalterato e restituisce `false`. Si noti che rimuovendo un elemento si può creare un "buco" nella sequenza dei valori; gestire la cosa, facendo in modo che all'uscita del metodo i valori validi presenti nell'array siano quelli con indice compreso tra 0 e `numeroElementi` – 2.

Esercizio 2 (Istruzioni a un robot)

Scrivere un programma Java che consenta all'utente di specificare a un robot i passi da fare per raggiungere (eventualmente) un obiettivo. La posizione dell'obiettivo e i passi da effettuare vengono specificati da riga di comando, come nel seguente esempio:

```
java Programma 5 4 sssseeeenno
```

che indica che l'obiettivo da raggiungere si trova alla riga 5, colonna 4, e i passi che deve fare il robot sono: andare verso sud 3 volte, andare verso est 4 volte, andare verso nord 2 volte e infine andare verso ovest 1 volta.
Nel metodo main(), il programma crea anzitutto un mondo virtuale costituito da una matrice di 20 righe e 20 colonne. Posiziona poi l'obiettivo alle coordinate specificate dai primi due argomenti forniti sulla riga di comando. Quindi, posiziona 10 ostacoli in posizioni casuali: se al momento di posizionare un ostacolo, la posizione dovesse essere già occupata – da un altro ostacolo o dall'obiettivo – riprovare con un'altra posizione; alla fine, gli ostacoli posizionati devono essere 10. Infine, segna le posizioni occupate dal robot durante il percorso indicato da riga di comando, partendo dalla riga 0, colonna 0. Il percorso di interrompe se il robot va a finire nella posizione occupata dall'obiettivo, oppure in una posizione occupata da un ostacolo, oppure tenta di andare oltre a uno dei bordi del mondo. Stampa quindi a video il mondo virtuale con indicati obiettivo,

ostacoli e percorso del robot, e stampa un messaggio che dice se il robot ha raggiunto l'obiettivo, oppure è finito contro un ostacolo o se ha cercato di attraversare uno dei bordi del mondo.
I metodi da implementare nella classe `Metodi` sono i seguenti:

- creaMondo(): alloca una matrice di 20 × 20 interi, e restituisce al chiamante un riferimento a tale matrice. Il mondo appena creato ha tutte le posizioni vuote, cioè la matrice contiene 0 in ogni elemento;
- aggiungiObiettivo(mondo,riga,colonna): se la posizione specificata da `riga` e `colonna` è libera (contiene 0), aggiunge l'obiettivo (rappresentato dal numero 1) e restituisce `true` al chiamante. Se la posizione specificata è invece occupata, oppure è fuori dalla matrice, lascia il mondo inalterato e restituisce `false` al chiamante;
- aggiungiOstacolo(mondo): aggiunge un ostacolo (rappresentato dal numero 2) in una posizione casuale del mondo. Se la posizione tirata a caso è occupata, riprova fino a che non trova una posizione libera;
- aggiungiPosizioniRobot(mondo,percorso): segna nella matrice corrispondente al mondo virtuale, usando il numero 3, le posizioni occupate dal robot durante il suo cammino. Il robot parte sempre dalla posizione (0,0); se va a sud si incrementa il numero di riga, se va a est si incrementa il numero di colonna. L'argomento `percorso` è una stringa formata dai caratteri `s` (sud), `n` (nord), `e` (est) e `o` (ovest). Il percorso si interrompe se il robot va a occupare una posizione occupata da un ostacolo o dall'obiettivo, oppure se va contro uno dei bordi del mondo (cioè se cerca di uscire dalla matrice). Il metodo restituisce al chiamante: 1 se il robot ha raggiunto l'obiettivo, 2 se è finito contro un ostacolo, 3 se ha cercato di uscire dalla matrice, 0 se nessuna di queste condizioni si è verificata;
- stampaMondo(mondo): stampa a video una rappresentazione grafica del mondo virtuale, come segue:

```
**********************
*O   OO              #    *
*O   #O                   *
*OOOOO                    *
*                #        *
*                         *
*     X                   *
* #          #            *
*                         *
*                         *
*                #        *
*                         *
*                         *
*                         *
*                         *
*     #           #       *
*                         *
*                         *
*                      #  *
*           #             *
*                         *
**********************
```

dove i bordi del mondo sono rappresentati da asterischi, le posizioni vuote da spazi, gli ostacoli da `#`, l'obiettivo da `X` e il percorso del robot da `O`. Si noti che la posizione dell'obiettivo e il percorso corrispondono agli argomenti passati da riga di comando, nell'esempio dato sopra.

Esercizio 3 (Simulazione di una matrice bidimensionale)

Scrivere un programma Java che simuli le operazioni di accesso e modifica di un elemento in una matrice bidimensionale, usando un array unidimensionale. In altre parole, il programma lavora su un array unidimensionale `array[]` di interi, e i metodi scriviElemento() e leggiElemento() specificati sotto danno l'illusione di lavorare con una matrice bidimensionale `matrice[][]` di interi.

I metodi da implementare nella classe Metodi sono i seguenti, mentre il metodo main() della classe `Programma` viene utilizzato per testare tali operazioni:

- creaMatrice(n,m): crea un array unidimensionale di $n \times m$ elementi interi, che verrà usato per simulare una matrice di n righe ed m colonne, e ne restituisce un riferimento al chiamante;
- scriviElemento(array,n,m,i,j,elemento): dato il riferimento ad un array unidimensionale di $n \times m$ elementi, considerarlo come una matrice di n righe ed m colonne i cui elementi sono memorizzati una riga dopo l'altra, e simulare l'operazione di assegnamento `array[i][j] = elemento`. Il metodo non restituisce nulla al chiamante. Verificare che la lunghezza dell'array sia effettivamente uguale a $n \times m$, e che gli indici i e j non vadano fuori dalla matrice; in caso contrario, stampare un messaggio d'errore e uscire dal programma;
- leggiElemento(array,n,m,i,j): i parametri sono da interpretare come nel metodo precedente. Viene restituito al chiamante il valore dell'elemento `array[i][j]`. Anche in questo caso, verificare che la lunghezza dell'array sia effettivamente uguale a $n \times m$, e che gli indici i e j non vadano fuori dalla matrice; in caso contrario, stampare un messaggio d'errore e uscire dal programma;
- stampaMatrice(array,n,m): stampa gli elementi della matrice, una riga dopo l'altra, andando a capo al termine di ogni riga. Per accedere agli elementi della matrice, questo metodo fa uso del metodo leggiElemento(array,n,m,i,j).

Esercizio 4 (Somma ricorsiva)

Scrivere un programma Java che calcoli la somma $a+b$ tra due numeri interi non negativi (cioè maggiori o uguali a 0) a e b in maniera ricorsiva, sfruttando la seguente definizione induttiva della somma:

$$a + 0 = a$$

$$a + b = (a + 1) + (b - 1) \qquad \text{per } b > 0$$

Definire pertanto un metodo ricorsivo somma(a,b), che restituisce al chiamante il valore della somma $a+b$. Basandosi sulla definizione induttiva data sopra, impostare la ricorsione sul secondo parametro.

Esercizio 5 (MCD ricorsivo)

Scrivere un programma Java che calcoli il massimo comun divisore (MCD) tra due numeri interi positivi a e b in maniera ricorsiva, sfruttando la definizione induttiva già vista nella quarta esercitazione:

$$\text{MCD}(x,x) = x$$

$$\text{MCD}(x,y) = \text{MCD}(y,x)$$

$$\text{MCD}(x,y) = \text{MCD}(x - y,y) \qquad \text{se } x > y$$

Definire pertanto un metodo ricorsivo mcd(a,b), che restituisce al chiamante il valore del massimo comun divisore tra a e b. Basandosi sulla definizione induttiva data sopra, impostare la ricorsione sul primo parametro, scambiando fra loro gli argomenti passati quando è necessario farlo.

Esercizio 6 (Gioco del Tris)

Scrivere un programma Java che consenta all'utente di giocare a Tris contro il computer.
Il computer gioca secondo le seguenti regole, prese dal Capitolo 1 del libro "*L'Algoritmo Definitivo. La macchina che impara da sola e il futuro del nostro mondo*", di Pedro Domingos, edizioni Bollati Boringhieri (2016):

1. se l'avversario ha occupato due caselle di fila, occupare la casella rimanente;
2. altrimenti, se c'è una mossa che crea due coppie di caselle adiacenti in un colpo solo, scegliere quella;
3. altrimenti, se la casella centrale è libera, occuparla;
4. altrimenti, se l'avversario ha occupato una casella d'angolo, occupare la casella opposta;
5. altrimenti, se c'è un angolo vuoto, occuparlo;
6. altrimenti, occupare una casella vuota qualsiasi.

Il programma chiederà anzitutto se l'utente vuole fare o no la prima mossa, e poi giocherà secondo le regole date sopra. Assegnare il simbolo O all'utente, mentre il computer userà il simbolo X.

La classe `Metodi` conterrà almeno i seguenti metodi:

- stampaConfigurazioneDiGioco(): stampa la configurazione attuale del gioco (di cui si passa un riferimento, come argomento), secondo lo schema che segue:

```
O|O|O
-+-+-
 |X|
-+-+-
X|X|
```

- creaNuovaConfigurazione(): crea una nuova tabella, vuota, pronta per giocare. Restituisce al chiamante un riferimento a tale tabella;
- mossaComputer(): prende come argomento un riferimento alla configurazione attuale del gioco, ed esegue la prossima mossa del computer, secondo le regole riportate sopra;
- mossaUtente(): prende come argomento un riferimento alla configurazione attuale del gioco, e chiede all'utente di fare la propria mossa. Se l'utente specifica una casella già occupata, stampa un opportuno messaggio d'errore e chiede nuovamente di specificare la mossa;
- verificaVittoria(): prende come argomento un riferimento alla configurazione attuale del gioco, e restituisce al chiamante: 1 se l'utente ha vinto, – 1 se il computer ha vinto, 0 se non ha vinto nessuno.

Prima di invocare i metodi mossaComputer() e mossaUtente(), assicurarsi che sia effettivamente possibile fare una mossa.

Osservazione finale: nel libro di Domingos si dice che l'algoritmo è *ottimale*, nel senso che non perde mai. Giocare qualche partita per vedere se è vero!