Commento all'es. 1

L'esercizio propone un problema verifica/selezione su una matrice, a cui si aggiunge un problema di ritorno di dati "by reference", sia durante l'acquisizione della matrice che durante il riconoscimento dei rettangoli.

Per il problema di ritorno di dati per riferimento, si consiglia di far riferimento a libro e lucidi: di fatto si tratta di utilizzare all'interno della funzione, a partire da un parametro puntatore (es. `nrp`), espressioni con asterisco (es. `*nrp`). Si noti poi come un parametro puntatore può essere passato direttamente a `scanf` (senza `&`). Si noti come, nella funzione `riconosciRegione`, si usino internamente variabili locali `b` e `h` per base e altezza, assegnate solo come ultime istruzioni alle variabili riferite (puntate) da `pb` e `ph`.

Per quanto riguarda il problema di riconoscimento di rettangoli, si sfrutta l'assunzione che i rettangoli siano corretti, cioè che non ci siano forme diverse nella matrice. Ci si limita quindi a misurare i due lati adiacenti al vertice in alto a sinistra di un rettangolo.

Le caselle di un rettangolo già riconosciuto vengono poi "annullate" ai fini di riconoscimenti di altri rettangoli successivi.

Commento all'es. 2

L'esercizio viene risolto in due versioni, che differiscono nel modo utilizzato per gestire le date: in un caso come `struct` e nel secondo come intero (che fa in modo di associare a ogni giorno un numero intero diverso, tale da rispettare i criteri di confronto). Si noti che le date (e più in generale il tempo) sono informazioni composte (aggregati) di giorno, mese e anno; nel caso specifico a queste si aggiungono anche ora, minuti e secondi. **Si tratta in effetti di un problema di codifica**. Per operazioni (aritmetiche o di confronto) su date è possibile

- realizzare funzioni che in modo esplicito gestiscano le singola componenti,
- oppure convertire una data in giorni (un tempo in secondi), come valore intero e lavorare su questo

Indipendentemente dal modo di lavorare, una data può essere rappresentata come `struct`, come stringa o come un unico numero. La scelta dipende spesso dal tipo di operazioni che è necessario effettuare:

- se è sufficiente ricordare e visualizzare una data, una stringa può essere la soluzione più semplice. Una `struct` è più versatile se è necessario visualizzare campi in modo separato o in formati diversi a seconda dei casi.
- se occorre effettuare operazioni, si sconsiglia la stringa, mentre può essere equivalente la rappresentazione (compatta) come un unico intero, oppure come `struct`. Si tenga presente che:
    - se si effettuano solo confronti tra date, non è necessario garantire la contiguità tra gli interi che rappresentano le date (non è necessario, ad esempio, che il 1 luglio sia rappresentato dal numero corrispondente al 30 giugno + 1), è sufficiente garantire la relazione di ordine. Ad esempio, `dataInt = aa*10000+mm*100+gg`

- se si vuole effettuare aritmetica (ad esempio poter aggiungere o togliere un intero arbitrario a una data), allora è necessario garantire la contiguità delle codifiche intere (tenendo conto dei giorni in ogni mese e degli anni bisestili. Occorre inoltre spesso una funzione di decodifica

Si propone una soluzione che rappresenta le date come `struct` e una che le rappresenta internamente come intero.

Per la rappresentazione interna dei dati si usa, come tabella un vettore di `struct` (tipo `entry`, avente campi stringa, ad eccezione della data). Il vettore viene acquisito nella funzione `leggiTabella`, che ne ritorna la dimensione effettivamente usata. La gestione dei comandi è un problema di menu (fatto con utilizzo di un tipo `enum`, si veda ad esempio "Dal problema al programma" 4.4.1).
La selezione e stampa dei dati richiesti è un problema di filtro dati, che comprende in questo caso la richiesta (se necessaria) delle due date necessarie per determinare l'intervallo di selezione. La funzione `selezionaDati` comprende quindi:

- l'acquisizione delle due date
- l'iterazione sugli elementi della tabella, per ognuno dei quali si verifica eventualmente la compatibilità con le date e si stampa il campo richiesto. Si noti l'utilizzo di un costrutto `switch` per differenziare le scelte possibili sul campo da visualizzare.

Commento all'es. 3

Il problema comprende:

- un sotto-problema di acquisizione di matrice, con ritorno (by reference) delle due dimensioni
- l'acquisizione del cammino da verificare
- un problema di verifica del cammino, per il quale occorre accertare:
  - che sia effettivamente un cammino, cioè privo di "salti"
  - che percorra caselle valide (con pesi non nulli) della matrice
  - che sia "semplice", cioè non passi due o più volte sulla stessa casella.

Il problema di acquisizione non richiede particolari commenti e si rimanda direttamente alla soluzione proposta).
Cammino: si evita di acquisirlo in modo esplicito (ad esempio in un vettore di `struct`, per rappresentare i singoli punti), a patto di integrare acquisizione e verifica del cammino in un'unica iterazione. La funzione `verificaCammino` effettua quindi l'acquisizione di ogni punto del cammino in due variabili locali (`x, y`) verificandone direttamente la conformità ai criteri proposti (a cui si aggiunge un controllo sulla compatibilità con gli intervalli validi per le coordinate di riga e colonna):

- la non presenza di salti viene verificata ricordando ogni volta le coordinate del punto precedente (-1,-1 all'inizio), e verificando che la distanza del nuovo punto da questo, sia su riga che colonna, sia ≤ 1
- la verifica che la casella della matrice abbia peso non nullo è immediata

- l'assenza di cicli (cammino semplice) potrebbe essere verificata controllando che ogni nuovo punto non sia ripetuto nel cammino, ma per fare ciò occorrerebbe memorizzare il cammino, ad esempio in un vettore: il costo finale dell'algoritmo sarebbe poi quadratico sulla lunghezza del cammino (O(|cammino|)). Per ottenere una complessità (in tempo) lineare, è preferibile utilizzare la matrice, modificando il valore di ogni casella nel momento in cui vi si passa, ad esempio assegnandovi -1. Questo permette di verificare in modo diretto (O(1)) la presenza di un ciclo. Questa tecnica presuppone tuttavia che la matrice possa essere modificata, perdendo il contenuto originale (i pesi), il che è possibile nel problema proposto. Se ciò non fosse possibile, sarebbe necessario utilizzare una seconda matrice, contenente di fatto un valore logico (un flag) per ognuna delle caselle della matrice,