Compitino 1 del 22/4/2015 Turno 1

Programmazione:

L'esercizio tratta di pattern matching ed è parecchio simile all' esercizio 1 del 13/4. Il programma dichiara un array int X[400] che va "visto" come int Y[lim1][lim2][lim3] e legge n_el valori mettendoli in Y per strati. A differenza dell'esercizio 1, in questo esercizio è sempre vero che n_el è sempre maggiore o uguale a lim1*lim2*lim3. Quindi Y è sempre completamente definito. Ma anziché fare il pattern matching sulle fette di Y, lo faremo su un sotto-array di Y che è una semplice estensione della nozione di tassello già visto in precedenza e che chiariamo nel seguente esempio.

Esempio supponiamo che lim1=3, lim2=4 e lim3=5, che n_el=60 e che i 60 elementi letti in Y siano i seguenti già distribuiti sui 3 strati di Y:

i 3 rettangoli rappresentano un sotto-array di Y che è costituito dai 18 elementi contenuti nei 3 rettangoli, quindi 6 elementi per ciascun strato. Come mostrato dalla figura i 3 rettangoli si trovano esattamente nella stessa posizione di ciascuno strato. Infatti un sotto-array è definito da 4 quantità:

i) riga r e colonna c dell'elemento in alto a sinistra, nell'esempio r=0 e c=2

ii) l'altezza H e la larghezza L del rettangolo, nell'esempio H=3 e L=2. La profondità del sotto-array è sempre lim1 dato che Y è completo.

Nel caso H=L=1, il sotto-array diventa un tassello. Variando i 4 parametri, si possono definire sotto-array a piacimento. Ovviamente, con r=c=0, H=lim2 e L=lim3 il sotto-array coincide con Y.

Un sotto-array di Y in un certo senso è un array (a 3 dimensioni [lim1][H][L]), ma si deve osservare che i suoi elementi non sono tutti disposti in memoria in modo contiguo come il C++ prescrive per gli elementi degli array. Per esempio nella figura il valore 4 della seconda riga del rettangolo nello strato 0 non segue immediatamente in memoria il valore 0 che termina la prima riga del rettangolo. Tra i 2 valori ci sarà una distanza di 3 interi: uno per terminare la prima riga dello strato 0 e 2 della seconda riga per arrivare al valore 4.

Problema: sugli elementi di un sotto-array è possibile considerare vari ordini. In questo esercizio noi considereremo l'ordine determinato dalle V-fette del sotto-array, cioè prima gli elementi della V-fetta 0 del sotto-array, poi quella della V-fetta 1 e così via. Dove la V-fetta 0 del sotto-array è costituita dalle colonne 0 degli strati del sotto-array, la V-fetta 1 dalle colonne 1 degli strati e così via. Quindi le V-fette dei sotto-array sono definite esattamente nello stesso modo in cui le abbiamo definite per gli array a 3 dimensioni normali.

Esempio ancora: se applichiamo l'ordine per V-fette al sotto-array della figura, otteniamo la seguente lista, 1 4 0 2 2 2 0 0 3 0 1 1 3 1 2 1 1 1, dove le diverse colonne sono separate da uno spazio aggiuntivo per facilitare la comprensione. Le prime 3 colonne sono le colonne di indice 0 del sotto-array e le ultime 3 sono quelle di indice 1. Gli elementi di questa lista hanno un indice 0,1,2,3,...,17. Quindi per esempio l'elemento di indice 5 del sotto-array è il terzo 2 della sequenza e quello di indice 8 è il 3.

Il problema da risolvere consiste nel contare quanti match non sovrapposti esistono tra un dato pattern P e gli elementi di un dato sotto-array considerati nell'ordine per V-fette, appena descritto.

Il main del programma è dato per quanto riguarda l'i/o. Si chiede di completare il ciclo principale del main che scorre gli elementi del sotto-array in ordine per V-fette e invoca una funzione match (da fare) che deve soddisfare le seguenti specifiche:

PRE=(X è un array che contiene almeno lim1*lim2*lim3 elementi e va listo come int Y[lim1][lim2][lim3], (r,c,H,L) definisce un sotto-array di Y (r+H<=lim2 e c+L<=lim3), P è un array che contiene dimP elementi, el è l'indice di un elemento del sotto-array (r,c,H,L) a partire dal quale il sotto-array ha ancora almeno dimP elementi)

bool match(int* X, int r, int c,int H, int L, int lim1, int lim2, int lim3,int * P, int dimP, int el)

POST=(restituisce true se il sotto-array (r,c,H,L) ha un match contiguo a partire dall'elemento el con P[0..dimP-1] e false altrimenti)

Consiglio: oltre a match conviene avere una funzione che, dato l'indice di un elemento di un sotto-array, calcola la posizione dell'elemento all'interno del sotto-array, dove la posizione è determinata dalla riga, colonna e strato in cui si trova l'elemento relativamente al primo elemento del sotto-array, cioè rispetto a (r,c). A questo scopo la funzione potrebbe usare il tipo struttura tripla che viene dato prima del main, completo di costruttore. Altrimenti la funzione potrà restituire i risultati con parametri passati per riferimento.

Caveat: il programma non dovrebbe in alcun modo accedere gli elementi di Y esterni al sotto-array considerato. Inoltre si sconsiglia l'uso di un array ausiliario che serva a contenere gli elementi del sotto-array in ordine per V-fette. E' considerato un errore definire Y[lim1][lim2][lim3] come un array distinto da X oppure definire un array Z[lim1][H][L] in cui ricopiare il sotto-array di Y.

Correttezza: si richiede l' invariante e la prova di correttezza del ciclo della funzione match. Pre e Post di funzioni ausiliarie. Argomenti a sostegno della loro correttezza.