Scritto di Programmazione del 27/8/2015

Abbiamo un array int A[400] che vogliamo vedere come se avesse 3 dimensioni, cioè, come se fosse int X[lim1][lim2][lim3]. Chiariamo subito che non si deve in alcun modo dichiarare un array int X[lim1][lim2][lim3], ma che si deve solo "vedere" A come se fosse X.

Vedendo X come una torta, le sue V-fette sono fette tagliate lungo le colonne degli strati di X. Quindi la V-fetta 0 è costituita dalle colonne 0 degli strati di X, la V-fetta 1 è costituita dalle colonne 1 degli strati di X e così via. In questo esercizio considereremo un particolare ordine degli elementi delle V-fette, **l'ordine per tasselli**, che è spiegato nel seguente esempio:

Esempio: sia lim1=3, lim2=4 e lim3=5. Supponiamo che i 3 strati di X siano riempiti completamente nel modo seguente:

strato 0	strato 1	strato 2
0 0 1 0 1	0 0 0 2 1	1 1 2 2 0
1 0 0 1 1	0 0 0 2 2	0 0 0 0 0
0 0 2 3 1	3 3 1 0 0	2 2 2 0 3
3 1 1 2 0	2 2 2 2 2	1 2 2 3 4

allora la V-fetta 0, ordinata per tasselli, consiste dei 4 tasselli caratterizzati dalle posizioni (0,0), (1,0),(2,0) e (3,0) e quindi gli elementi della V-fetta 0 in questo ordine sono: 0 0 1 1 0 0 0 3 2 3 2 1. Il tassello (0,0) è costituito da X[0][0][0]=0, X[1][0][0]=0, e X[2][0][0]=1, il tassello (1,0) consiste di X[0][1][0]=1, X[1][1][0]=0, e X[2][1][0]=0, il tassello (2,0) corrisponde a X[0][2][0]=0, X[1][2][0]=3, e X[2][2][0]=2 e il tassello (3,0) corrisponde a X[0][3][0]=3, X[0][3][1]=2, e X[0][3][2]=1.
Prendiamo un'altra V-fetta, per esempio quella di indice 3: anch'essa è costituita da 4 tasselli, che sono caratterizzati dalle posizioni, (0,3), (1,3), (2,3), e (3,3) e quindi i suoi elementi, in ordine per tasselli, sono: 0 2 2 1 2 0 3 0 0 2 2 3.

Esercizio iterativo: come di consueto, è dato un programma con alcune dichiarazioni ed un main. Il main compie alcune operazioni di input e invoca la funzione iterativa legge che è da fare. La funzione legge ha il compito di leggere dei valori presenti su "input" e di inserirli in A, visto come X, riempiendo prima la V-fetta 0 in ordine per tasselli, poi, la V-fetta 1 in ordine per tasselli e così via finché o incontra la sentinella -2 o legge lim1*lim2*lim3 valori riempiendo quindi tutta X . **Ricordiamo ancora una volta che in realtà X non esiste e che legge deve riempire A che viene "vista" come X[lim1][lim2][lim3].**

Ecco la segnatura della funzione legge e la sua pre e post-condizione:

PRE=(lim1, lim2 e lim3 sono valori positivi, A contiene almeno lim1*lim2*lim3 elementi, lo stream IN contiene o -2 o almeno lim1*lim2*lim3 valori interi)
int legge(int*A, int lim1, int lim2, int lim3, ifstream & IN)
POST=(La funzione restituisce il numero di valori letti da IN (escludendo l'eventuale sentinella), i valori letti sono inseriti in A (visto come X[lim1][lim2][lim3]) per V-fette, secondo l'ordine per tasselli. La lettura si interrompe o quando viene letta la sentinella -2 oppure quando sono stati letti lim1*lim2*lim3 valori).

Facendo riferimento all'esempio precedente, se i primi valori letti da IN, fossero 0 0 1 1 0 0 0 3 2 3 2 1, allora la funzione legge li dovrebbe inserire in A in modo da occupare la V-fetta 0 di X, vista in ordine per tasselli, proprio come nell'esempio.

Attenzione: *non è detto che X sia riempita di valori alla fine di legge. In generale, alcune V-fette di X (le prime) possono essere piene e dopo ci può essere un'altra V-fetta solo parzialmente riempita (secondo l'ordine per tasselli).*

La funzione legge **deve fare uso di una funzione calc_coord specificata some segue**:
PRE=(lim1, lim2 e lim3 sono positivi, 0<=VF<lim3, 0<=n<lim1*lim2)
coord calc_coord(int lim1, int lim2, int lim3, int VF, int n)
POST=(la funzione restituisce un valore a di tipo coord tale che X[a.s][a.r][a.c] è l'elemento di indice n della V-fetta VF di X, vista in ordine per tasselli))[1].
La definizione del tipo struct coord è nel programma dato.

Esercizio ricorsivo: si tratta di definire una funzione ricorsiva F1 che esegua un particolare match di P nelle V-fette di X, considerando solo gli elementi definiti delle V-fette. Il pattern matching che vogliamo realizzare funziona secondo questo principio:
si cerca P[0] nella V-fetta 0 scandendo gli elementi della V-fetta 0 secondo l'ordine per tasselli. Supponiamo che nella V-fetta 0 compaia P[0] e che la sua prima occorrenza sia l'elemento della V-fetta di indice V0. Allora dobbiamo considerare il match di P[1] sulla V-fetta 1 a partire dal suo elemento di indice V0+1. Se P[1] viene trovato all'indice V1 della V-fetta 1, allora si cercherà P[2] sulla V-fetta 2 a partire da V1+1. E così via. Nel caso che la ricerca di P[i] nella V-fetta i a partire dall'indice Y fallisca, allora si deve cercare P[i] sulla prossima V-fetta (i+1) a partire dallo stesso indice Y.

Esempio: supponiamo che lim1=3, lim2=4 e lim3=5 e di aver letto in X per V-fette in ordine per tasselli i seguenti valori: 3 0 0 1 0 0 1 3 3 3 3 5 3 3 0 1 1 0 -2. Si tratta di 18 valori (escludendo il -2) che quindi riempiranno completamente la V-fetta 0 e per metà la V-fetta 1, precisamente sono riempiti i primi 2 tasselli della V-fetta 1. Supponiamo che il pattern sia 1 0 0. Allora P[0] viene trovato nell'elemento di indice 3 della V-fetta 0 e P[1] viene trovato nell'elemento di indice 5 della V-fetta 1. Dopo di che il match deve terminare perché sono finite le V-fette che contengono valori letti. In questo caso F1 deve effettuare la stampa su OUT dei 2 indici trovati 3 5.
Se invece il pattern fosse 5 0 0, allora P[0] verrebbe trovato sull'ultimo elemento della V-fetta 0 e quindi non sarebbe più possibile andare avanti col match. Quindi in questo caso F1 dovrebbe stampare 11 e basta.
Infine, se il pattern fosse 1 5 0, allora P[0] verrebbe trovato nella posizione 3 della V-fetta 0, mentre la ricerca di 5 sulla V-fetta 1 (a partire dall'indice 4) fallirebbe. In questo caso F1 dovrebbe stampare: 3 -1 a indicare il fallimento del match di P[1] sulla V-fetta 1. Dopo il match termina perché non ci sono più V-fette. Altrimenti sarebbe continuato cercando P[1] nella V-fetta 2 a partire dall'indice 4.
F1 è specificata come segue:
PRE=(lim1,lim2 e lim3 sono positivi, A contiene almeno lim1*lim2*lim3 elementi, P ha dimP elementi, 0<=VF<lim3, 0<=restoelem, 0<=inizio<minimo(lim1*lim2,restoelem))
void F1(int*A,int lim1, int lim2, int lim3, int*P, int dimP, int restoelem, int inizio, int VF, ofstream & OUT)
POST=(esegue il match come delineato prima e stampa su OUT l'indice del match nei casi di successo e -1 in quelli di insuccesso).

Attenzione: -il parametro inizio è l'indice a partire dal quale va iniziata la ricerca di P[0] nella V-fetta VF (seguendo l'ordine per tasselli);
-restoelem indica il numero di elementi definiti delle V-fette che restano da esaminare (dalla VF in poi);
- osservate che la PRE richiede che 0<=inizio<minimo(lim1*lim2,restoelem) che implica che inizio sia l'indice di un elemento definito della V-fetta VF.
- conviene che F1 invochi un'altra funzione (ricorsiva) che si occupi di cercare il prossimo elemento di P sulla prossima V-fetta a partire da un certo indice; questa funzione ausiliaria dovrà molto probabilmente usare la funzione calc_coord della parte iterativa in modo da scorrere la V-fetta seguendo l'ordine per tasselli come richiesto.
Correttezza:
a) scrivere l'invariante del ciclo principale della funzione legge.
b) Delineare la prova induttiva della correttezza di F1 rispetto a PRE e POST date.

[1] Avendo a.s, a.r e a.c è facile trovare il corrispondente elemento di A.