Scritto di Programmazione 28/3/2013

Informazioni importanti: trovate nella vostra home 2 file esercizio1.cpp ed esercizio2.cpp. Il primo concerne la domanda iterativa ed il secondo quella ricorsiva. A ciascun file dovete aggiungere le funzioni richieste dalla domanda corrispondente. I 2 file contengono il main che esegue l'apertura dei file e le letture e scritture da e sui file. Inoltre ciascun main invoca le funzioni che sono da sviluppare per l'esame. Il comando di consegna è, come sempre, "consegna esercitazione". Le parti che riguardano la correttezza vanno aggiunte in fondo al file come commento.

Programmazione iterativa: L'esercizio è molto simile a quello del compito del 21/3. Solo la definzione di fette cambia. Ora esse sono costituite da colonne e non da righe di X. Ecco l'esercizio.

Abbiamo un array int X[lim1][5][10], lim1>0, che contiene dim>0 elementi definiti e due interi k1 e k2>=0. Chiamiamo fette di X le sequenze di colonne di strati sovrapposti. Quindi la fetta 0 sarà la sequenza delle colonne 0 degli strati di X, la fetta 1 la sequenza delle colonne 1 e così via. Solo i primi dim elementi di X sono definiti (guardate il main di esercizio1.cpp se avete dubbi su quali siano questi dim elementi di X) e quindi si deve avere cura di percorrrere le fette in modo da toccare solo elementi definiti di X. Si tratta di scrivere una funzione F che restituisca l'indice minimo di una fetta di X che soddisfi la seguente condizione (*): la fetta contiene k2 volte il valore k1.

Esempio: *Supponiamo per semplicità che X[3][3][4] e che dim=17, questo significa che c'è un solo strato completamente definito e che il secondo ha la prima riga completamente piena, la seconda con 1 solo valore definito e la terza completamente indefinita. Infine il terzo strato è completamente indefinito. Caratterizzando con X gli elementi definiti e con E quelli indefiniti, i 3 strati di X sono così:*

strato

0	*1*	*2*
XXXX	*XXXX*	*EEEE*
XXXX	*XEEE*	*EEEE*
XXXX	*EEEE*	*EEEE*

Quindi le fette saranno così: la fetta 0 sarà costituita da 2 colonne: le prime colonne dello strato 0 e 1 di X, dove la prima sarà completa di 3 valori, mentre la seconda avrà solo 2 valori. La fetta 1 di X sarà costituita dalle colonne 1 dei primi 2 strati di X dove la prima sarà piena mentre la seconda contiene 1 solo elemento. Le fette 2 e 3 sono composte come la fetta 1.

Una volta che si è capito quali sono gli elementi definiti delle fette, scrivere un codice che calcoli se una fetta soddisfa (*) non dovrebbe essere difficile. La pre- e postcondizione di F ed il suo prototipo sono come segue:

PRE_F=(X ha dimensioni [][5][10], lim1>0, dim>0, i primi dim elementi di X sono definiti, k1 e k2 sono definiti e k2 >=0)

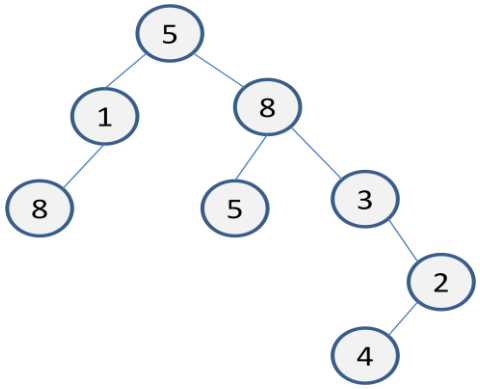
int F(int(*X)[5][10], int lim1, int dim, int k1, int k2)

POST_F=(F restituisce il minimo indice tra le fette che soddisfano (*) , se nessuna fetta soddisfa (*) allora F restituisce -1)

Ogni funzione ausiliaria che venga introdotta deve essere accompagnata da pre- e postcondizioni.

E' richiesto di specificare l'invariante del ciclo principale della funzione F.

Programmazione ricorsiva: Si tratta di fare un pattern match su un albero binario. Il pattern è il solito array P che contiene dimP interi. Il testo è rappresentato dai cammini di un albero binario dato. Consideriamo il seguente albero binario e supponiamo che P=[8,2] e dimP=2. Troviamo un solo match di P sul cammino più a destra nell'albero. Il cammino che caratterizza questo match è: [1,1,1,-1]. Si osservi che il match non è necessariamente contiguo, ma deve essere completo, cioè l'intero pattern P deve venire trovato lungo un cammino di R.



Il -1 finale serve da sentinella per segnalare che il cammino col match finisce. Se P=[5,5], allora il cammino sarebbe [1,0,-1]. Quindi un 1 significa "il cammino va a destra" e uno 0 significa "il cammino va a sinistra". I cammini partono sempre dalla radice e si arrestano con -1 non appena il match è completo. Nel caso ci siano più match di P, vogliamo quello che viene prima rispetto all'attraversamento prefisso dell'albero. Per esempio se P=[5,8], allora in R esistono 2 match che corrispondono ai cammini: [0,0,-1] e [1,-1]. Si chiede di produrre il primo dei 2. Per farlo basta esaminare i cammini dell'albero da sinistra a destra seguendo l'ordine prefisso. Quando invece non c'è alcun match, come succede per esempio con P=[5,9,3], per convenzione il corrispondente match è [2].

Si tratta di scrivere una funzione **ricorsiva** bool M(nodo*R, int prof, int*P, int dimP, int*C) che soddisfi le seguenti pre e postcondizione:

PRE_M=(R è un albero corretto, prof>=0 , P ha dimP>=0 elementi def., C ha abbastanza elementi[1])

POST_M(se M restituisce false, allora C[0]=2 e R non contiene match di P)&&(se M restituisce true, allora C=[i1,..,im,-1] è il primo cammino (rispetto all'attraversamento in prefisso) che caratterizza un match di P in R)

Si richiede di dimostrare la correttezza di M rispetto a queste asserzioni.

[1] C ha un n. di elementi almeno uguale al valore del parametro prof più l'altezza H dell'albero R. Quindi finché restiamo in questi limiti non dobbiamo preoccuparci di controllare che C sia abbastanza lungo.