Esame di programmazione 10 settembre 2014

Il problema proposto è lo stesso dello scritto del 31/3/2014, ma la parte iterativa di quell'esame va ora fatta ricorsivamente e quella ricorsiva va ora fatta iterativamente. Ricordiamo il problema:

Si tratta di un problema di pattern matching in cui il testo T è una lista concatenata ed il pattern P è un array di dimP interi. T è costituita di nodi con 2 campi informativi. Il tipo usato è nodo2 (vedi main) e solo il campo inf1 è usato per il match, mentre info2 contiene il numero di sequenza di ciascun nodo in T. I match da considerare iniziano sempre in P[0], poi P[1] eccetera, ma non devono essere necessariamente né completi né contigui . Facciamo subito un esempio.

Esempio. Sia T=(1,0)->(0,1)->(2,2)->(2,3)->(2,4)->(1,5)->(0,6) e sia P=[2,2,1,1]. Esistono al più match di lunghezza 3. Ce ne sono 2, uno inizia nel nodo di indice 2 di T, mentre il secondo inizia nel nodo di indice 3 di T. Rappresentiamo questi 2 match con le seguenti 2 liste:
X=(2,2)->(1,1) e Y=(3,3) che hanno il seguente significato:
X) il fatto che X consista di 2 nodi, indica che il match che X rappresenta non è contiguo, ma consiste di 2 parti, che i 2 nodi di X descrivono come segue. La prima componente del primo nodo (2,2) di X ci dice che troviamo un match di P[0] dopo aver saltato 2 nodi di T (cioè troviamo il match sul terzo nodo di T, cioè sul nodo di indice 2), mentre la seconda componente del nodo di X (ancora 2) ci dice che il massimo match a partire da questo nodo ha lunghezza 2, cioè P[1] fa match sul quarto nodo, ma P[2] non fa match sul quinto nodo. La prima componente del secondo nodo (1,1) di X ci dice che basta saltare 1 nodo (saltare il quinto nodo) per trovare il match di P[2] sul sesto nodo. Questo match ha massima lunghezza 1 perché P[3] non fa match con il settimo e ultimo nodo di T e quindi X non ha altri nodi. Quindi X rappresenta un match non contiguo e anche incompleto visto che P[3] non ha match. Visto che X corrisponde ad un match di P in cui ogni elemento di P è matchato su T al più presto possibile, è ovvio che il match rappresentato da X è di lunghezza complessiva massima (3) e sarà anche il match che inizia dal nodo di T di indice minimo. Nessun altro match di P in T può iniziare prima di X o matchare più elementi di P.
Y) La situazione è più semplice per Y, infatti esso ha un solo nodo che indica che il match rappresentato da Y ha una sola parte di lunghezza 3 (seconda componente del nodo) e che per trovarne l'inizio dobbiamo saltare i primi 3 nodi di T. Inoltre Y ci dice che il settimo nodo di T non matcha P[3] e visto che T non ha ulteriori nodi, anche Y non ha altri nodi.
Se P fosse [1,0,1,0,2] allora il match di P in T (di lunghezza massima e che inizia prima) sarebbe descritto dalla lista (0,2)->(3,2), che mostra come P[0..1] vengano trovati sui primi 2 nodi di T, poi è necessario saltare 3 nodi per trovare un altro pezzo di match che è lungo 2 (dopo T finisce).

Osservazioni:
(i) Si osservi che i nodi della lista X sono identici come tipo a quelli di T. Entrambi sono costituiti da 2 campi informativi interi (info1 e info2) e dal campo next di tipo nodo2*, vedi il file col main. Nella lista T il campo info1 è quello usato per il pattern match, mentre info2 è semplicemente l'indice del nodo in T ed è inserito dalla funzione crea che è data. Nei nodi di X info1 e info2 hanno il significato spiegato nell'esempio precedente.
(ii) Nel seguito si dovrà considerare il match di P in T di lunghezza massima e che inizia il più presto possibile nella lista T.
(iii) Il match non deve essere necessariamente né continuo, né completo.

Parte ricorsiva: si chiede di costruire una funzione iterativa che, dati T e P, costruisca la lista X che rappresenta il match (vedi il punto (ii) precedente) di P in T nel modo descritto nell'esempio. Nel seguito chiameremo questa lista X **"la lista del match di P in T".**
Il prototipo della funzione e le sue pre- e post-condizioni sono come segue:

PRE=(T lista corretta, dimP>=0)
nodo* M1(nodo* T, int*P, int dimP)
POST=(M1 restituisce col return la lista del match di P in T)

Consiglio importante: conviene fare in modo che M1 usi due funzioni ricorsive ausiliarie col seguente compito: data una lista L
a) una funzione ricorsiva deve calcolare la massima lunghezza di un match contiguo di P a partire dall'inizio di L;
b) l'altra funzione ricorsiva deve calcolare quanti nodi di L è necessario saltare per trovare l'inizio di un nuovo match.

Parte iterativa: in questo esercizio useremo una lista T in tutto simile a quella della parte ricorsiva ed una lista X anch'essa simile a quella della parte ricorsiva, ma in cui i campi hanno come unico vincolo il fatto di essere interi non negativi. Quindi in questo esercizio X non rappresenta necessariamente un match come nell'esercizio iterativo. Per costruire questa lista X viene data un'apposita funzione crea1 che legge entrambi i campi dei nodi da "input". I dati su "input" e il loro ordine sono rivelati dalle letture effettuate dal main.

Data una lista T ed una lista X fatta da nodi che hanno campi info1 e info2 maggiori o uguali a 0, chiamiamo (T-X) e (X di T) due sotto-liste di T come spiegato nel seguente esempio.

Esempio: sia T=(1,0)->(0,1)->(2,2)->(0,3)->(2,4)->(1,5)->(0,6) e X=(1,2)->(0,0)->(0,2)->(2,0), allora (T-X) è definita considerando i seguenti effetti dei nodi di X:
i) il nodo (1,2) di X indica di tenere il primo nodo di T (info1=1) e poi di staccare i successivi 2 nodi (info2=2), cioè i nodi di indice 1 e 2;
ii) il nodo (0,0) di X non fa nulla visto che info1=info2=0;
iii) il nodo (0,2) richiede tenere i successivi 0 nodi di T e poi di staccare i successivi 2 (info2=2), quindi verranno staccati i nodi di indice 3 e 4;
iv) il nodo (2,0) di X indica di tenere i prossimi 2 nodi di T (info1=2) e di staccare i successivi 0 nodi. A questo punto T è finita e anche se X non fosse finita, non potremmo più fare nulla.
Quindi (T-X)=(1,0)->(1,5)->(0,6) mentre (X di T) consiste degli altri nodi di T, cioè, (0,1)->(2,2)->(0,3)->(2,4).

Si chiede di sviluppare una funzione ricorsiva TB che soddisfi le seguenti specifiche:
PRE_TB=(T e X sono liste corrette, I campi info1 e info2 dei nodi di X (se ci sono nodi) sono tutti maggiori o uguali a 0, T=vT)
nodo2* TB(nodo*&T, nodo*X)
POST_TB=(la funzione restituisce (vT-X) col return e alla fine T= (X di vT)).

Consiglio importante: Conviene usare una funzione iterativa ausiliaria che soddisfi le seguenti pre- e post-condizioni ed abbia il prototipo specificato:
PRE=(L è una lista corretta e k >=0, vL=L)
nodo2* stacca(nodo2*&L, int k)
POST=(restituisce col return la parte iniziale di vL di k nodi (o l'intero vL se ha meno di k nodi) e alla fine L ha come valore quello che resta di vL dopo aver tolto la parte restituita col return)
Come al solito è necessario fare attenzione ai casi limite. Per esempio quando k=0 e quando la lista L finisce "troppo presto".

Correttezza: specificare un invariante per il ciclo principale di TB e dimostrare che è realmente un invariante. Dimostrare induttivamente la correttezza di M1 rispetto alle pre- e post-condizioni date.

Per chi deve fare l'integrazione: basta trascurare la parte di correttezza.