Esame di Programmazione del 14/7/2015

Si deve realizzare una funzione **ricorsiva** F0 che, data una lista concatenata i cui nodi contengono o 0 o 1, inserisca un 1 al posto dell'ultimo 0 presente nella lista e metta a 0 tutti i campi info dei nodi che seguono quel nodo. Si osservi che questi campi info avranno necessariamente valore 1. L'ordine che assumiamo tra i nodi della lista è quello triviale dall'inizio al fondo della lista.

Esempio: sia L=0->1->0->0->1->1 allora, l'ultimo 0 è quello nel nodo di indice 3 (attenzione che gli indici partono da 0), quindi la lista va trasformata nella seguente: 0->1->0->1->0->0. Applicando di nuovo la funzione sulla lista appena ottenuta, avremmo: 0->1->0->1->0->1 e applicando la funzione per la terza volta: 0->1->0->1->1->0.

Basta osservare un pò l'esempio per rendersi conto che l'operazione che viene fatta è quella di somma di 1 ad un numero non negativo in rappresentazione binaria. La lista di partenza dell'esempio, L=0->1->0->0->1->1 rappresenta in binario il valore 19 (in base 10), e la seconda lista, 0->1->0->1->0->0, rappresenta 20, la terza lista rappresenta 21 e così via. Ovviamente tutto procede bene sino a che non otteniamo la lista 1->1->1->1->1->1 (che rappresenta 63) . In questa lista non ci sono 0 e quindi la nostra operazione è destinata a fallire. Si tratta di un overflow.

La funzione **ricorsiva** F0 che realizza l'operazione appena descritta deve soddisfare le seguenti richieste:

PRE=(list(L) corretta, L=vL)

void F0(nodo*L, bool & b)

POST=(b => lista(L) è ottenuta da lista(vL) facendo la trasformazione richiesta) &&(!b => lista(vL) è tale che ogni nodo ha campo info=1 e lista(L)=lista(vL))

Osserviamo ora il risultato di applicare F0 partendo da una lista con, per esempio, 3 nodi tutti con campo info=0: da 0->0->0 si ottiene 0->0->1, e ripetendo l'applicazione si ottiene 0->1->0, e dopo 0->1->1 e ancora 1->0->0. Dovrebbe essere chiaro che dopo 7 ripetizioni otterremo 1->1->1.

Ora ciascuna di queste liste rappresenta un cammino in un albero binario e quindi le 8 liste (contando anche quella iniziale con soli 0) che possiamo produrre in questo modo sono tutti i cammini (radice-foglia) esistenti in un albero binario completo (vedi figura alla fine del testo) di altezza 3.

Vogliamo usare questa osservazione per costruire o modificare un albero binario. Si richiede di scrivere una funzione **iterativa** F1 che riceva in input un qualsiasi albero R (R può essere anche vuoto) ed un intero alt (alt>0) e che, generando tutti i cammini di lunghezza alt con la funzione F0, aggiunga a R tutti i nodi di tutti questi cammini che non siano già presenti in R. Per ogni nodo da aggiungere il campo info va letto dal file di "input".

Si osservi che il tipo nodoT è quello dei nodi degli alberi binari (vedi programma dato), mentre nodo è quello dei nodi delle liste concatenate.

Le specifiche della funzione iterativa F1 sono le seguenti:

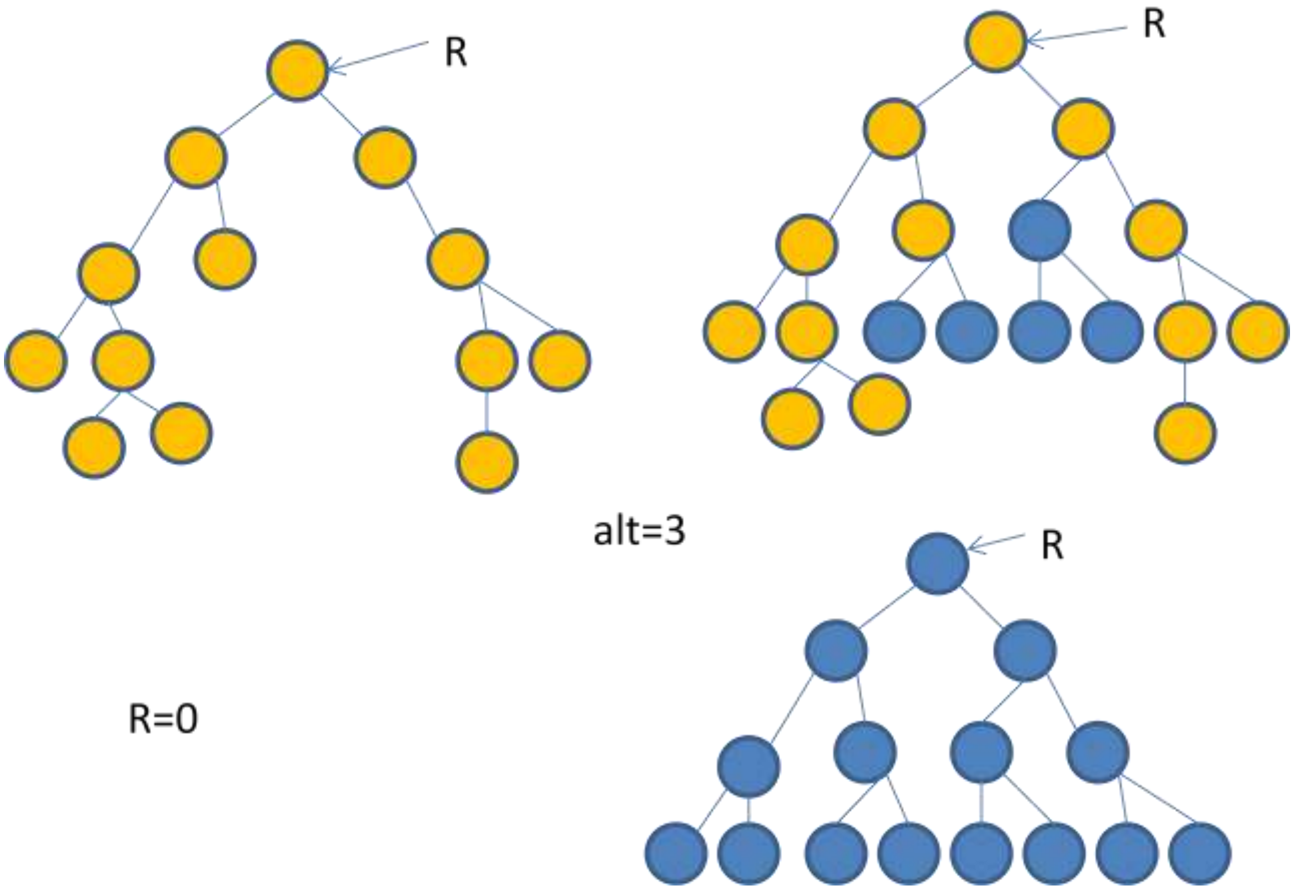
PRE=(albero(R)corretto, 0<prof, INP ifstream definito, R=vR, INP contiene (almeno) $2^{(alt+1)}$ -1 valori)

nodoT * F1(nodoT* R, int alt, ifstream & INP)

POST=(albero(R) è ottenuto da albero(vR) aggiungendo tutti i nodi non già presenti in albero(vR) di tutti i cammini di lunghezza alt, i campi info dei nodi sono letti da INP. I cammini vanno considerati partendo dal cammino con soli 0, poi applicando F0 ripetutamente)

Correttezza: i) scrivere la dimostrazione della correttezza di F0 rispetto alle pre e post-condizioni date.

ii) Scrivere l'invariante del ciclo principale di F1.



Nella figura rappresentiamo l'effetto di F1 su un albero R non vuoto e costituito inizialmente dai nodi arancioni e su un albero vuoto (R=0). I nodi blu sono quelli aggiunti da F1 nei due casi.

Attenzione: per il caso in cui inizialmente R=0, è necessario costruire immediatamente la radice del nodo prima di esaminare i cammini. L'albero d'altezza 0 ha comunque la radice.