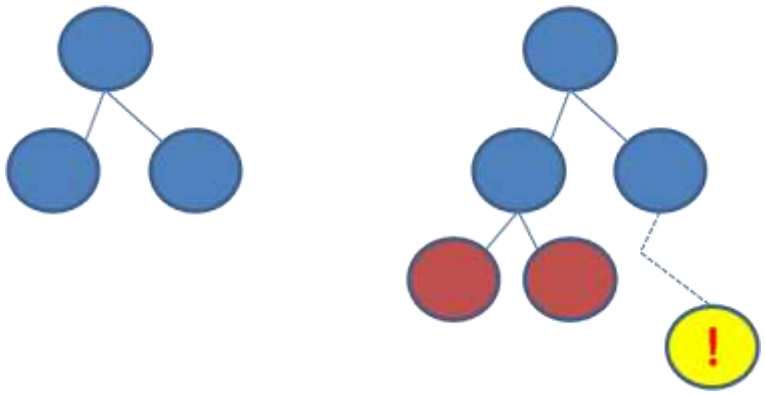
Esame di programmazione 19 marzo 2014

Il problema da risolvere è quello di aggiungere nodi ad un albero binario. Viene richiesta una soluzione iterativa ed una ricorsiva dello stesso problema.

I nodi da aggiungere sono rappresentati da una sequenza di cammini (composti da 0 e 1) e per ogni cammino si tratta di percorrerlo nell'albero per aggiungere un nuovo nodo alla fine del cammino percorso. Ovviamente, per un dato albero, ci sono cammini che non permettono di aggiungere un nodo all'albero, per esempio, potrebbe essere che il cammino sia "troppo corto" e quindi che il posto in cui aggiungere il nodo è già occupato oppure che il cammino sia "troppo lungo" e che il nodo da aggiungere non trovi nell'albero un nodo padre a cui venire appeso. Vediamo con un esempio.

Esempio. Si consideri l' albero binario a sinistra e la seguente sequenza di cammini: 00-101-1101-1, ogni cammino è terminato dalla sentinella -1. A destra si può osservare l'albero ottenuto aggiungendo al primo i nodi corrispondenti ai 3 cammini dati.



In realtà solo 2 cammini si possono usare per appendere nuovi nodi (quelli segnati in rosso), mentre il terzo è "troppo lungo" e quindi nessun nodo verrà aggiunto all'albero in corrispondenza di questo cammino. l nodo che NON può essere aggiunto è evidenziato in giallo nell'albero a destra.

Invece il cammino 0-1 sarebbe "troppo corto" rispetto all'albero di sinistra in quanto termina in un nodo che è già presente nell'albero. Lo stesso dicasi del cammino vuoto che corrisponde alla radice dell'albero ed è rappresentato semplicemente da -1.

Come di consueto, viene fornito un main che apre i file "input" e "output" e compie le letture dei dati che servono all'esercizio. Il file "input" contiene i seguenti dati:

1) contiene i cammini che sono sequenze di 0/1, ogni cammino è separato dal successivo con un -1 e la sequenza termina con la sentinella -2. Il main legge i cammini in un array int C[200].

2) immediatamente dopo i cammini, "input" contiene 4 sequenze di valori, ognuna contenente tanti interi quanti sono i cammini stessi. L'idea è che questi valori dovranno essere inseriti nel campo info dei nodi da aggiungere, corrispondentemente ai cammini. Servono 4 sequenze in quanto l'esercizio richiede di costruire 4 alberi (vedere il main). Qualora un cammino si riveli inutile e nessun nodo viene aggiunto in corrispondenza ad esso, si dovrà aver cura di "consumare" il corrispondente valore e di scriverlo su "output" accompagnato da una frase appropriata.

Il main invoca due funzioni build_iter e build_ric (2 volte per ciascuna funzione) che sono identiche a parte il fatto che la prima invoca add_iter e la seconda invoca add_ric. Le due invocazioni di build_iter e di build_ric ricevono come alberi su cui lavorare, l'albero X costruito dalla funzione costruz (data) e l'albero nullo. Dopo ciascuna invocazione, il main stampa in forma lineare l'albero prodotto (funzione stampa_l data). Le due funzioni add_iter e add_ric cercano entrambe di aggiungere un nuovo nodo ad un albero dato, una in modo iterativo e l'altra in modo ricorsivo. Le funzioni add_iter e add_ric **sono da fare**. Le due funzioni build_iter e build_ric invocano anche la funzione inizio_cam che è anch'essa da fare e **può essere a scelta sia iterativa che ricorsiva**. Le funzioni da fare sono descritte nel seguito.

Per inizio_cam:

PRE_inizio=(C contiene n cammini (sequenze possibilmente vuote di 0 e 1), ogni cammino termina con -1 e dopo l'ultimo cammino (che termina, come tutti i cammini, con -1) c'è la sentinella -2, n >= 0, i >= 0)

int inizio_cam(int*C, int i);

POST_inizio=(se i in [0..n-1] allora inizio_cam restituisce l'indice di C in cui inizia il cammino i, altrimenti, inizio_cam restituisce -2)

Avvertenza: se ci sono n cammini su C, la posizione del loro inizio in C verrà trovata invocando inizio_cam con i=0, i=1,...,i=n-1. Per ogni i>=n, la funzione deve restituire -2 a segnalare che il cammino richiesto non c'è. E' necessario evitare che, nel caso i=n, la funzione restituisca la posizione del -2 come inizio di un inesistente cammino in più.

Per add_iter ed add_ric:

PRE_add=(C[0...] contiene una sequenza w di 0 e 1 (possibilmente vuota) seguita da -1, r è un albero corretto, INP contiene almeno un intero che chiamiamo x, r=vr)

void add_....(int* C, nodo*& r, ifstream & INP, ofstream & OUT);

POST_add=(se w è troppo corto o troppo lungo rispetto a vr, allora v resta uguale a vr e su "output" verrà scritto "cammino troppo corto" oppure "cammino troppo lungo" seguito da x, altrimenti r è un albero corretto che si ottiene da vr aggiungendogli un nuovo nodo tale che il cammino dalla radice di r al nuovo nodo sia w e tale che il suo campo info sia x)

Avvertenza: consiglio decisamente di realizzare prima add_ric e poi add_iter. Infatti la seconda è più noiosa da fare ed aver già fatto la versione ricorsiva è un importante aiuto. Per questo motivo il main invoca prima build_ric e poi build_iter.

Correttezza: E' richiesta la prova induttiva della correttezza della funzione add_ric. Si chiede anche di definire un invariante per il ciclo principale di inizio_cam e, se possibile, di mostrare che inizio_cam è corretta rispetto alla PRE_inizio e POST-inizio date.

Per chi deve fare l'integrazione: al posto della parte di correttezza, scrivere una funzione che listi su "output" i campi info dei nodi di un albero binario elencati secondo l'ordine breath-first (cioè a livelli, prima il livello 0 (la radice), poi il livello 1 (i figli della radice), poi il livello 2 (nipoti della radice) e così via). Ogni livello va stampato da sinistra a destra.