Algoritmi e Strutture di Dati – A.A. 2017-2018
Prova scritta del 19 febbraio 2018 – D.M. 509-5CFU
Libri e appunti chiusi – Tempo = 1:45h

5

☐ Note (es: correzione veloce, eventuali indisponibilità, ecc.) ……
……
……

Sono anche disponibile a sostenere l'orale: ☐ dal 1° al 2 marzo 2018 ☐ dal 12 al 16 marzo 2018

Cognome: _____ Nome: _____Matricola:_____

PSEUDOCODIFICA

Nell'esercizio seguente un albero binario `T` con n nodi è un oggetto che ha il solo campo `T.root` che è un riferimento al nodo radice dell'albero, dove ogni nodo ha i campi `parent` (genitore), `left` (figlio sinistro), `right` (figlio destro) e `info` (un intero tra 0 e n-1 che identifica il nodo).

Un grafo non orientato è rappresentato con un array A in cui ogni elemento A[*u*] è un riferimento al primo elemento della lista di adiacenza doppiamente concatenata del nodo *u* (con i campi `prev`, `info` e `next`). Essendo il grafo non orientato esiste un arco (*u*,*v*) per ogni arco (*v*,*u*).

Esercizio 1

Scrivi lo pseudocodice della procedura **TRASFORMA**(`T`) che accetti in input un albero binario `T` e produca in output un grafo non orientato A che ha la stessa struttura di `T` (cioè ha lo stesso numero di nodi e un arco tra due nodi se nell'albero c'è una relazione padre-figlio tra questi).

Esercizio 2

Discuti la complessità computazionale della seguente procedura nel caso peggiore fornendo O-grande, Omega e Theta in funzione del numero n di elementi dell'albero.

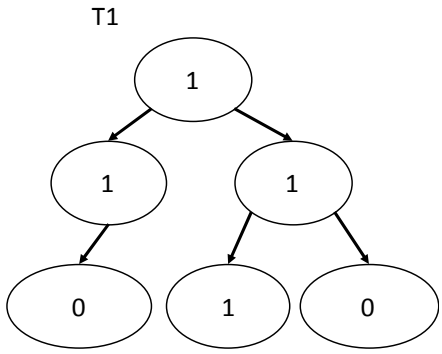
```
FUNZIONE(T)      /* T è un albero binario di interi */
L.head = NULL    /* L è una nuova lista (vuota) di interi */
FUNZ-RIC(T.root,L)
return L

FUNZ-RIC(v,L)
if(v==NULL) return
if(v.parent == NULL)
      AGGIUNGI-IN-CODA(L,v.info)
else
      AGGIUNGI-IN-TESTA(L,v.info)
FUNZ-RIC(v.left,L)
FUNZ-RIC(v.right,L)
```

Assumi che AGGIUNGI-IN-TESTA faccia un numero di operazioni costante, mentre AGGIUNGI-IN-CODA faccia un numero di operazioni proporzionali alla lunghezza della lista corrente.

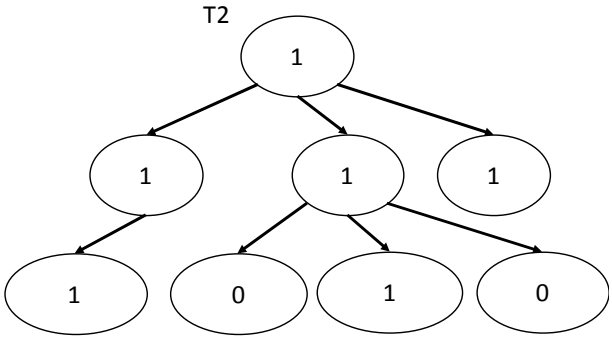
LINGUAGGIO C

Si consideri la libreria **bool.h** che implementa quanto segue:

```
typedef struct elem {
            int valore;
            struct elem* sx;
            struct elem* dx;
   } nodo_albero;
typedef nodo_albero* BTree;
```




```
typedef struct elem2 {
            int valore;
            struct elem2* primofiglio;
            struct elem2* fratello;
   } nodo_alberoN;
typedef nodo_alberoN* NTree;
```



Tale libreria implementa valori booleani (1 o 0) di tipo **int**, alberi binari i cui nodi contengono booleani nel tipo **BTree**, e alberi di grado arbitrario i cui nodi contengono booleani nel tipo **BNTree**.

Utilizzando la libreria **bool.h**, si richiede di implementare in linguaggio C i seguenti metodi:

1) `int` **tutto_vero**`(BTree T)`, che restituisca `1` se tutto il sottoalbero radicato in `T` (compreso `T`) contiene esclusivamente valori `1` e nessun valore `0`. Altrimenti restituisce `0`. Se l'albero `T` è vuoto restituisce il valore `1`. Ad esempio **tutto_vero**`(T1)` restituirà il valore `0` perché l'albero `T1` contiene sia nodi `0` che nodi `1`.
2) `int` **sottoalberi_veri**`(Btree T)` che dato un albero binario `T` conta il numero di nodi di T che sono radice di sottoalberi contenenti esclusivamente il valore 1 (dalla radice del sottoalbero alle foglie). Se l'albero `T` è vuoto la funzione restituirà il valore 0. Ad esempio **sottoalberi_veri**`(T1)` restituirà il valore `1` (nell'albero `T1` c'è un solo sottalbero vero, quello radicato nella foglia centrale).
3) `int` **esiste_uniforme**`(NTree T)`, che dato un albero di grado arbitrario `T` verifica se esiste in T un nodo che abbia tutti i figli con lo stesso suo valore (che può essere indifferentemente `0` o `1`). Se il nodo N è uguale a NULL oppure non ha figli allora **esiste_uniforme**`(N)` ritorna `0`. Ad esempio **esiste_uniforme**`(T2)` restituirà il valore `1` (sia la radice che il suo primo figlio sinistro hanno tutti i figli contenenti il loro valore).

È possibile utilizzare qualsiasi libreria nota e implementare qualsiasi metodo di supporto a quelli richiesti.

SOLUZIONI

Esercizio 1

```
/* conta il numero dei nodi del sottoalbero radicato al nodo dato */

CONTA-NODI(n)
        if (n==NULL) return 0
        return 1+CONTA-NODI(n.left)+CONTA-NODI(n.right)

AGGIUNGI-ARCO(A,i,j)
        /* temp è un nuovo elemento della lista */
        temp.info = j
        temp.prev = NULL
        temp.next = A[i]
        if (A[i] != NULL)   // c'èra già un nodo in lista
                A[i].prev = temp
        A[i] = temp

VISITA-ALBERO(n,A)
        if (n == NULL) return
        if(n.parent != NULL)
                AGGIUNGI-ARCO(A,n.info,n.parent.info)
                AGGIUNGI-ARCO(A,n.parent.info,n. info)
        VISITA-ALBERO(n.left,A)
        VISITA-ALBERO(n.right,A)

TRASFORMA(T)
        nodi = CONTA-NODI(T.root)
        /* A è un nuovo array di riferimenti ad elementi della lista con "nodi" posizioni */
        VISITA-ALBERO(n, A)
        return A
```

Esercizi 2 e 3: vedi soluzione del compito DM 270/04 da 6 cfu del 19 febbraio 2018