Algoritmi e Strutture di Dati – A.A. 2017-2018
Prova scritta del 4 luglio 2018 – D.M. 270-6CFU
Libri e appunti chiusi – Tempo = 2:00h

6

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

Sono disponibile a sostenere l'orale: ☐ domani 5 luglio 2018 ☐ dal 23 luglio in poi

Cognome: _____ **Nome:** _____ **Matricola:** _____

PSEUDOCODIFICA

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 **VERIFICA**(A) che accetti in input un grafo non orientato A e verfichi che la componente connessa con più nodi abbia esattamente il doppio dei nodi della componente connessa con meno nodi.

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.

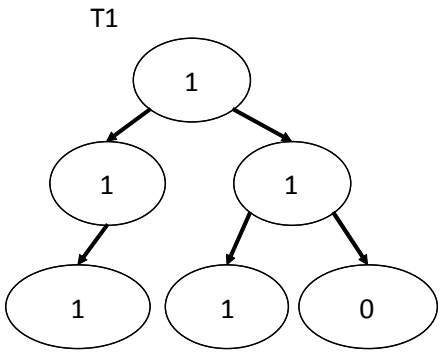
```
FUNZIONE(T)      /* T è un albero binario di interi */
L.head = NULL    /* L è una nuova lista (vuota) di interi */
FUNZ-RIC(T.root,L)
return L

FUNZ-RIC(v,L)
if(v==NULL) return
if(v.left == NULL && v.right == 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.

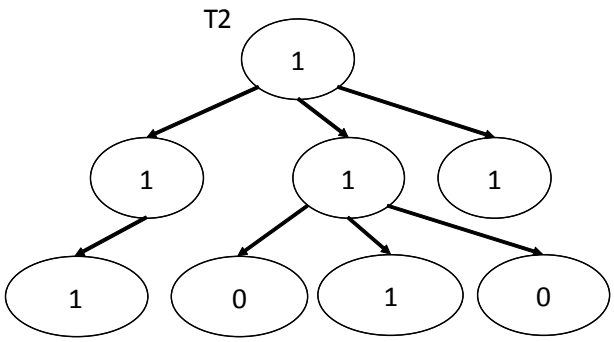
LINGUAGGIO C

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 **`NTree`**.

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

1) `int` **`misto`**`(BTree T)`, che restituisca `1` se il sottoalbero radicato in `T` (compreso `T`) contiene sia valori `1` che valori `0`. Altrimenti restituisce `0`. Se l'albero `T` è vuoto la funzione restituisce il valore `0`. Ad esempio **`misto`**`(T1)` restituirà il valore `1` perché l'albero `T1` contiene sia nodi `0` che nodi `1`. Invece **`misto`**`(T1->sx)` restituirà il valore `0` perché l'albero `T1->sx` contiene tutti nodi `1`.
2) `int` **`sottoalberi_misti`**`(Btree T)` che dato un albero binario `T` conta il numero di nodi di T che sono radice di sottoalberi contenenti sia il valore `1` che il valore `0`. Se l'albero `T` è vuoto la funzione restituirà il valore 0. Ad esempio **`sottoalberi_misti`**`(T1)` restituirà il valore `2` (nell'albero `T1` solo la radice e il suo sottoalbero destro sono misti).
3) `int` **`esiste_difforme`**`(NTree T)`, che dato un albero di grado arbitrario `T` verifica se esiste in T un nodo che abbia tra i suoi figli sia nodi con il valore `0` che nodi con il valore `1`. Se il nodo N è uguale a NULL oppure non ha figli allora **`esiste_uniforme`**`(N)` ritorna `0`. Ad esempio **`esiste_difforme`**`(T2)` restituirà il valore `1` (il figlio centrale della radice ha tra i suoi figli sia nodi con il valore `0` che nodi con il valore `1`).

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

SOLUZIONI

PSEUDOCODIFICA – ESERCIZIO 1

```
DFS(A, u, color, cont)
      color[u] = cont
      x = A[u]
      while( x != NULL)
            if( color(x.info)==0 )
                  DFS(A,x.info,color,cont)
            x = x.next

SIZE_COMPONENTE(color, cont)
      output = 0
      for i=0 to color.length-1
            if (color[i] == cont)
                  output = output + 1
      return output

VERIFICA(A)
      for i=0 to color.length-1
            color[i] = 0  // color array con A.length interi

      cont = 0   // conta (e marca) le componenti connesse
      comp_max = 0          // nodi componente più grande
      comp_min = A.length   // nodi componente più piccola
      for i=0 to color.length-1
            if (color[i] == 0)
                  cont = cont + 1
                  DFS(A,u,color,cont)  // visito e marco con cont
                  numero_nodi = SIZE_COMPONENTE(color,cont)
                  if (numero_nodi > comp_max)
                        comp_max = numero_nodi
                  if (numero_nodi < comp_min)
                        comp_min = numero_nodi
      return comp_max = 2 * comp_min
```

PSEUDOCODIFICA – ESERCIZIO 2

Risposta sufficiente nel compito d'esame:

- La funzione FUNZIONE(T) non fa che richiamare FUNZ-RIC(v,L,depth) e dunque ha la stessa complessità asintotica nel caso peggiore.
- La funzione FUNZ-RIC(v,L) esegue una visita dell'albero e per ogni foglia (che ha entrambi i figli == NULL) esegue un AGGIUNGI-IN-CODA. Per tutti gli altri nodi esegue AGGIUNGI-IN-TESTA. La complessità è $\Theta(n^2)$ perché il numero di foglie può essere O(n).

```
#include <stdlib.h>
#include "bool.h"

int contiene_valore(Btree T, int valore) {
     if(T==NULL) return 0;  // l'albero vuoto non contiene "valore"
     if(T.info == valore) return 1;  // trovato il valore cercato
     return     contiene_valore(T->sx,valore) ||
                contiene_valore(T->dx,valore);
}

int misto(BTree T){
     if(T==NULL) return 0;
     return contiene_valore(T,0) && contiene_valore(T,1)
}

int sottoalberi_misti(BTree T){
     if(T == NULL) return 0;
     int cont = misto(T); // verrà 1 se il nodo T soddisfa
     return cont+sottalberi_misti(T->sx)+sottalberi_misti(T->dx);
}

int esiste_difforme(NTree T){
     if (T == NULL) return 0;
     if (T->primofiglio == NULL) return 0;
     int figli_zero = 0;  // booleano: 1 se T ha figli con valore==0
     int figli_uno = 0;   // booleano: 1 se T ha figli con valore==1
     NTree x = T->primofiglio;
     while( x != NULL ) {
          if( x->valore == 0)
               figli_zero=1;  // ci sono figli contenenti zero
          if( x->valore == 1)
               figli_uno=1;   // ci sono figli contenenti uno
          x = x->fratello;
     }
     if (figli_zero && figli_uno)
          return 1;   // esiste un nodo difforme ed è questo
     x = T->primofiglio;
     while( x != NULL ) {
          if (esiste_difforme(x))
               return 1;
          x = x->fratello;
     }
     return 0;
}
```