Algoritmi e Strutture di Dati – A.A. 2017-2018
Prova scritta del 19 febbraio 2018 – D.M. 270-6CFU
Libri e appunti chiusi – Tempo = 2:00h

6

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

……

……

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

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 **CONNESSA-PIU-DENSA**(A) che accetti in input un grafo non orientato A e produca in output il numero degli archi della componente connessa di A che ha il numero più grande di archi.

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.

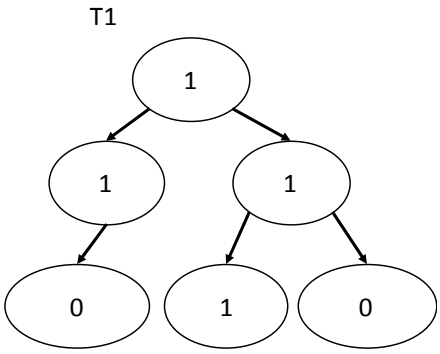
```
FUNZIONE(T)      /* T è un albero binario di interi */
L.head = NULL    /* L è una nuova lista (vuota) di interi */
FUNZ-RIC(T.root,L)
return L

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.

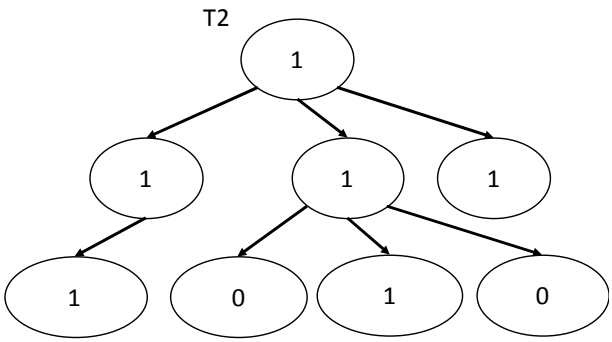
LINGUAGGIO C

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 **NTree**.

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

PSEUDOCODIFICA – ESERCIZIO 1

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


CONTA-ARCHI(A,u)
      cont = 0
      x = A[u]
      while(x != NULL)
            cont++
            x = x.next
      return cont


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

      comp = 0;   // indice della componente connessa
      for i=0 to A.length-1
            if(color[i]==0)
                  comp++
                  DFS(A,i,color,comp)

      max_archi = 0;
      for c=1 to comp
            archi = 0
            for i=0 to A.length-1
                  if(color[i]==c)
                        archi = archi + conta_archi(A,i)
            if(archi_componente/2 > max_archi)
                  max_archi = archi_componente/2
      return max_archi
```

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,depth) esegue una visita dell'albero e solo per la radice (che ha v.parent == NULL) esegue un AGGIUNGI-IN-CODA. Per tutti gli altri nodi esegue AGGIUNGI-IN-TESTA, e dunque ha complessità $\Theta(n)$

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


int tutto_vero(BTree T){
     if(T==NULL) return 1;
     return (T->valore == 1) && tutto_vero(T->sx) && tutto_vero(T->dx);
}

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

int esiste_uniforme(NTree T){
     if (T == NULL) return 0;
     if (T->primofiglio == NULL) return 0;
     int uniforme = 1;
     NTree x = T->primofiglio;
     while( x != NULL ) {
          if( x->valore != T->valore)
               uniforme = 0;
          x = x->fratello;
     }
     if( uniforme ) return 1; // il nodo corrente è uniforme

     // se il nodo corrente non è uniforme verifico nei sottalberi

     int esiste = 0;
     x = T->primofiglio;
     while( x != NULL ) {
          esiste = esiste || esiste_uniforme(x);
          x = x->fratello;
     }
     return esiste;
}
```