Algoritmi e Strutture di Dati – A.A. 2018-2019
Esame scritto del 20/06/19 – D.M. 270-9CFU
Libri e appunti chiusi – Tempo = 2:00h

9

☐ Note (vincoli, indisponibilità, preferenze, ecc.) ……

……

……

N.B.: gli esami orali si volgeranno dal 21 giugno al 26 luglio.

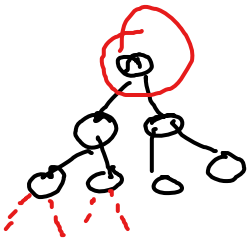
Cognome: _____ **Nome:** _____ **Matricola:** _____

DOMANDA SULLA COMPLESSITA' ASINTOTICA (3 punti su 30)

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)
if(v.left != NULL and v.right == NULL)
        AGGIUNGI-IN-TESTA(L,v.info)
if(v.left == NULL and v.right != NULL)
        AGGIUNGI-IN-TESTA(L,v.info)
FUNZ-RIC(v.left,L)
FUNZ-RIC(v.right,L)
```



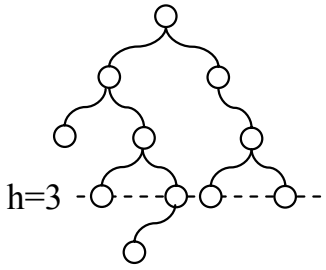
Assumi che AGGIUNGI-IN-TESTA faccia un numero di operazioni costante, mentre AGGIUNGI-IN-CODA faccia un numero di operazioni proporzionali alla lunghezza della lista corrente.

ALGORITMO IN LINGUAGGIO C (27 punti su 30)

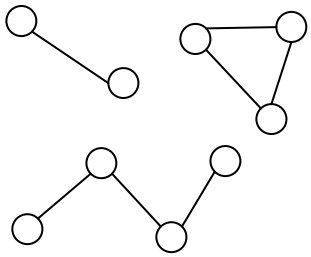
Scrivi in linguaggio C il codice della funzione

```
int verifica(int h, nodo_albero* a, grafo* g)
```

che accetti in input un intero **h**, un puntatore **a** alla radice di un albero binario di interi e un puntatore **g** ad grafo non orientato rappresentato tramite oggetti e riferimenti. La funzione restituisce 1 se il numero di nodi dell'albero **a** che sono a profondità **h** sono tanti quanti sono i nodi della componente più grande del grafo **g**, altrimenti la funzione restituisce 0. Se uno (o entrambi) tra grafo e albero è vuoto (cioè uguali a NULL) la funzione ritorna 0.



Albero a


Grafo g

Per esempio l'albero **a** in figura ha 4 nodi a profondità h=3 e la componente connessa più grande del grafo **g** ha 4 nodi, dunque `verifica(3,a,g)` ritorna 1 (true) mentre `verifica(1,a,g)` ritorna 0 (false) perché **a** ha solo 2 nodi a profondità h=1.

Usa le seguenti strutture (che si suppone siano contenute nel file "`strutture.h`"):

```
typedef struct nodo_struct {
        elem_nodi* pos;  /* posizione nodo nella
                            lista del grafo */
        elem_archi* archi;   // lista archi incidenti
        int color;
} nodo;

typedef struct arco_struct {
        elem_archi* pos;  // pos. arco lista grafo
        nodo* from;
        nodo* to;
        elem_archi* frompos;  // pos. arco nodo from
        elem_archi* topos;      // pos. arco nodo to
} arco;

typedef struct elem_lista_nodi {
        struct elem_lista_nodi* prev;
        struct elem_lista_nodi* next;
        nodo* info;
} elem_nodi;   // elemento di una lista di nodi
```

```
typedef struct elem_lista_archi {
        struct elem_lista_archi* prev;
        struct elem_lista_archi* next;
        arco* info;
} elem_archi;  // elemento di una lista di archi

typedef struct {
        int numero_nodi;
        int numero_archi;
        elem_archi* archi;  // lista degli archi
        elem_nodi* nodi;    // lista dei nodi
} grafo;

/* struttura per l'albero binario */

typedef struct nodo_albero_struct {
        struct nodo_albero_struct* left;
        struct nodo_albero_struct* right;
        int info;
} nodo_albero;
```

È possibile utilizzare qualsiasi libreria nota e implementare qualsiasi funzione di supporto a quella richiesta.

SOLUZIONE DOMANDA SULLA COMPLESSITA' ASINTOTICA

La funzione **FUNZIONE** non fa altro che richiamare **FUNZ-RIC** e dunque ha la sua stessa complessità.

La funzione **FUNZ-RIC** compie una visita dell'albero in tempo $\Theta(n)$ a cui si aggiungono le operazioni **AGGIUNGI-IN-TESTA** su ogni foglia (le foglie possono essere $\Theta(n)$, ma ogni operazione costa $\Theta(1)$, quindi queste operazioni costano complessivamente $\Theta(n)$) e **AGGIUNGI-IN-CODA** sulla sola radice (siccome la lista è vuota sulla radice, **AGGIUNGI-IN-CODA** equivale in questo caso ad **AGGIUNGI-IN-TESTA** ed ha una complessità $\Theta(1)$). Complessivamente, dunque, si ha una complessità nel caso peggiore $\Theta(n)$.

SOLUZIONE ALGORITMO LINGUAGGIO C

```
int verifica(int h, nodo_albero* a, grafo* g){
     if(a == NULL || g == NULL) return 0;
     return nodi_prof(a,h) == componente_max(g);
}

int nodi_prof(nodo_albero* a, int h){
     if( a == NULL) return 0;
     if( h == 0 ) return 1;
     return nodi_prof(a->left,h-1) + nodi_prof(a->right,h-1);
}

int componente_max(grafo* g){
     max_nodi = 0;       // nodi della componente massima
     elem_nodi* ln = g->nodi;
     while( ln != NULL ){
          ln->info->color = 0; // coloro tutto con zero
          ln = ln->next;
     }
     ln = g->nodi;
     while( ln != NULL ){
          if( ln->info->color == 0 ){
               int cur_nodi = dfs_conta(ln->info);
               if( cur_nodi > max_nodi) max_nodi = cur_nodi;
          }
          ln = ln->next;
     }
     return max_nodi;
}

int dfs_conta(nodo* n){
     int cont = 1;
     n->color = 1;
     elem_archi* el = n->archi;
     while( el != NULL ){
          nodo* altro_nodo = el->info->from;
          if( altro_nodo == n )
               altro_nodo = el->info->to;
          if( altro_nodo->color == 0 )
               cont = cont + dfs_conta(altro_nodo);
          el = el->next;
     }
     return cont;
}
```