Algoritmi e Strutture di Dati – A.A. 2018-2019
Esame scritto del 02/09/19 – 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 3 al 13 settembre e dal 24 al 30 settembre

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.left != NULL and 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)
```

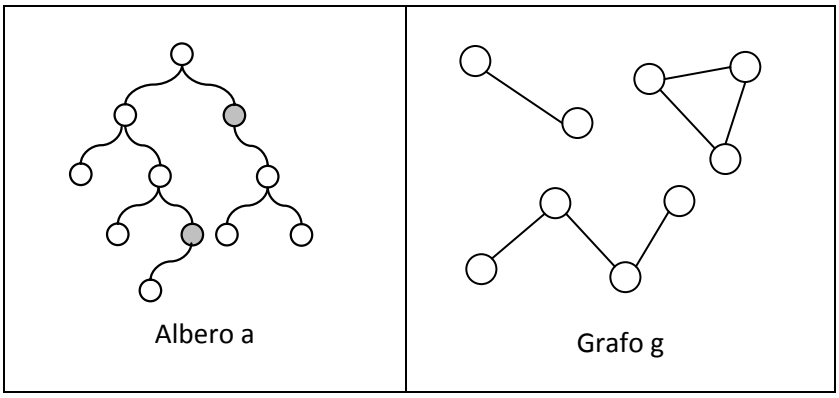
Assumi che AGGIUNGI-IN-TESTA faccia un numero di operazioni costante, mentre AGGIUNGI-IN-CODA faccia un numero di operazioni proporzionali alla lunghezza della lista corrente.

ALGORITMO IN LINGUAGGIO C (27 punti su 30)

Scrivi in linguaggio C il codice della funzione

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

che accetti in input 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 hanno un solo figlio sono tanti quanti sono i nodi della componente più piccola 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.



Per esempio l'albero **a** in figura ha 2 nodi con un solo figlio (quelli colorati di grigio) e la componente connessa più piccola del grafo **g** ha 2 nodi, dunque **verifica**(a,g) ritorna 1 (true).

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 (3 punti su 30)

FUNZIONE(T) non fa altro che richiamare **FUNZ-RIC**(n,L) sulla radice dell'albero, quindi ha la sua stessa complessità.

FUNZ-RIC(n,L) percorre tutto l'albero e quindi ha una complessità almeno Theta(n). Solamente per quei nodi che hanno solo il figlio destro esegue AGGIUNGI-IN-CODA, altrimenti esegue AGGIUNGI-IN-TESTA. La complessità di AGGIUNGI-IN-TESTA è Theta(1), dunque anche se fosse eseguita un numero lineare di volte porterebbe un contributo totale di Theta(n), che dominato dal Theta(n) della visita.

I nodi che hanno solo il figlio destro possono essere in numero lineare (per esempio se l'albero è un cammino di figli destri). Ne consegue che nel caso peggiore vengono eseguiti un numero lineare di inserimenti in coda. Questo determina un costo totale Theta(n^2).

SOLUZIONE ALGORITMO IN LINGUAGGIO C (27 punti su 30)

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


int nodi_solo_figlio(nodo_albero* a){
     int out = 0;
     if ( a == NULL ) return out;
     if ( (a->left == NULL) && (a->right != NULL) ) out++;
     if ( (a->left != NULL) && (a->right == NULL) ) out++;
     return out+nodi_solo_figlio(a->left)+nodi_solo_figlio(a->right);
}

int nodi_componente_piccola(grafo* g){
     int min_nodi; // nodi della componente più piccola
     elem_nodi* ln = g->nodi;
     while( ln != NULL ){
          ln->info->color = 0; // coloro tutto con zero
          ln = ln->next;
     }
     ln = g->nodi;
     if( ln == NULL ) return 0;  // il grafo non ha nodi
     min_nodi = dfs_conta(ln->info);
     ln = ln->next;
     while( ln != NULL ){
          if( ln->info->color == 0 ){
               int cur_nodi = dfs_conta(ln->info);
               if( cur_nodi < min_nodi) min_nodi = cur_nodi;
          }
          ln = ln->next;
     }
     return min_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;
}
```