Algoritmi e Strutture di Dati – A.A. 2017-2018
Prova scritta del 12 settembre 2018 – D.M. 270-9CFU
Libri e appunti chiusi – Tempo = 2:00h

9

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

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

DOMANDA SULLA COMPLESSITA' ASINTOTICA (3 punti su 30)

Discuti la complessità computazionale nel caso peggiore (in termini di O-grande, Omega e Theta) delle seguenti procedure in funzione del numero n di elementi dell'albero. Assumi che

- CERCA_IN_LISTA faccia un numero di operazioni proporzionali alla lunghezza della lista passata in input
- AGGIUNGI_IN_TESTA faccia un numero di operazioni costante
- AGGIUNGI_IN_CODA faccia un numero di operazioni proporzionali alla lunghezza della lista passata in input

```
CONTIENE_DOPPIONE(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(CERCA_IN_LISTA(L,v.info) == TRUE)
     AGGIUNGI_IN_CODA(L,v.info)   /* i doppioni li aggiungo in coda */
 else
     AGGIUNGI_IN_TESTA(L,v.info)
 FUNZ_RIC(v.left,L)
 FUNZ_RIC(v.right,L)
```

ALGORITMO IN LINGUAGGIO C (27 punti su 30)

Scrivi in linguaggio C il codice della funzione

```
int foglie_comp(grafo_oggetti* g, nodo_albero* a)
```

che accetti in input un puntatore ad grafo non orientato **g** rappresentato tramite oggetti e un puntatore **a** alla radice di un albero di grado arbitrario di interi rapresentato tramite la struttura figlio-sinistro (`left_child`) e fratello destro (`right_sibling`). La funzione restituisce 1 se almeno una componete connessa del grafo **g** ha tanti nodi quante le foglie dell'albero **a**, altrimenti la funzione restituisce 0. Se grafo e albero sono entrambi vuoti (cioè uguali a NULL) la funzione ritorna true. Se uno è vuoto e uno no, allora ritorna false.

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_oggetti;
```

```
typedef struct nodo_albero_struct {
        struct nodo_albero_struct* left_child;
        struct nodo_albero_struct* right_sibling;
        int info;
} nodo_albero;
```

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

SOLUZIONI

DOMANDA SULLA COMPLESSITA' ASINTOTICA

Risposta sufficiente nel compito d'esame:

- La funzione CONTIENE_DOPPIONE(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 nodo (cioè un numero lineare di volte) esegue una ricerca e, nel caso peggiore, un inserimento in coda. La sua complessità è dunque $\Theta(n^2)$, sia a causa della valutazione della condizione dell'istruzione if, sia a causa dell'aggiunta in coda (entrambe le operazioni danno globalmente un contributo quadratico che viene sommato).

ALGORITMO IN LINGUAGGIO C

```
#include <stdlib.h>      /* ora posso usare NULL */
#include "strutture.h"   /* includo le strutture fornite nel testo */

/* visita DFS del grafo rappresentato mediante oggetti e
   riferimenti e marcatura dei nodi visitati con il numero della
   componente attuale. */

void DFS(nodo* n, int comp) {

      n->color = comp;             // coloro il nodo come visitato
      elem_archi ea = n->archi;
      while( ea != NULL) {
            nodo* altro_nodo = ea->info->from;
            if( altro_nodo == n) {
                  altro_nodo = ea->info->to;
            }
            if( altro_nodo->color == 0) {  // se non e' gia' visitato...
                  DFS(altro_nodo, comp);    // ... lo visito e lo marco
            }
            ea = ea->next;
      }
}

/* conto le foglie di un albero di grado arbitrario rappresentato tramite
una strutura "figlo-sinistro fratello-destro". */

int conta_foglie(nodo_albero* a) {
      if ( a == NULL ) return 0;
      if ( a->left_child == NULL) {   // trovata foglia
            return 1 + conta_foglie(a->right_sibling;
       }
      return conta_foglie(a->left_child) +
              conta_foglie(a->right_sibling);
}
```

```
/* conta i nodi di una componente connessa */

int conta_nodi(grafo_oggetti* g, int comp) {

      int numero_nodi = 0;
      elem_nodi* en = g->nodi;
      while (en != NULL) {
            if(en->info->color == comp)
                  numero_nodi++;
            en = en->next;
      }
      return numero_nodi;
}

/* funzione richiesta dal compito */

int foglie_comp(grafo_oggetti* g, nodo_albero* a) {

      if(g == NULL && a == NULL) return 1;
      if(g == NULL || a == NULL) return 0;
      elem_nodi* en = g->nodi;
      while (en != NULL) {
            en->info->color = 0; // marco tutti i nodi con 0
            en = en->next;
      }
      int comp = 0;   // numero delle componenti connesse

      en = g->nodi;
      while (en != NULL) {
            if(en->info->color == 0) {  // trovato nodo non visitato
                  comp++;                // incremento numero componenti
                  DFS(n, comp);          // visito e marco con comp
            }
            en = en->next;
      }

      int foglie_albero = conta_foglie(a);
      int c;  // indice di una componente connessa
      for(c=1; c <= comp; c++)
            if( conta_nodi(g,c) == foglie_albero )  // trovata
                  return 1;
      return 0; // nessuna componente connessa soddisfa i requisiti
}
```