Algoritmi e Strutture di Dati – A.A. 2017-2018
Prova scritta del 4 luglio 2018 – D.M. 270-9CFU
Libri e appunti chiusi – Tempo = 2:00h

9

☐ 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:** _____

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

```
CONTIENE_DOPPIONE(T)       /* T è un albero binario di interi */

L.head = NULL    /* L è una nuova lista (vuota) di interi */
return FUNZ_RIC(T.root,L)

FUNZ_RIC(v,L)
 if(v==NULL) return FALSE
 if(CERCA_IN_LISTA(L,v.info) == TRUE)
     return TRUE         /* l'albero contiene un doppione */
 else
     AGGIUNGI_IN_TESTA(L,v.info)
     return 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 verifica(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`) e fratello destro (`right`). La funzione restituisce 1 se il numero delle componenti connesse del grafo è uguale al numero di sottoalberi "omogenei" dell'albero **a**, altrimenti la funzione restituisce 0. Un sottoalbero è "omogeneo" se ha il valore 1 nel campo `info` di tutti i suoi nodi (compresa la radice del sottoalbero). 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;
        struct nodo_albero_struct* right;
        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,depth) 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 testa. La sua complessità è dunque $\Theta(n^2)$, perché nel caso peggiore esegue $\Theta(n)$ inserimenti e ricerche.

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. */

void DFS(nodo* n) {

      n->color = 1;             // 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);    // ... lo visito e lo marco
            }
            ea = ea->next;
      }
}



/* verifica se il sottoalbero radicato al nodo a passato come parametro
contiene tutti valori info==1. L'albero è un albero di grado arbitrario.
Ritorna 0 (cioè false) quando lanciata su un albero NULL. */

int sottoalbero_omogeneo(nodo_albero* a) {
      if ( a == NULL ) return 0;
      if ( a.info != 1 ) return 0;   // non va: la radice non contiene 1
      int output = 1;                // assumiamo che sia omogeneo
      nodo_albero* x = a->left;
      while ( x != NULL ) {
            output = output && sottoalbero_omogeneo(x);
            x = x->right;
      }
      return output;
}
```

```
/* ritorna il numero di sottoalberi omogenei dell'albero a di grado
arbitrario passato come parametro */

int sottoalberi_omogenei(nodo_albero* a) {
      int output = 0;
      if (a == NULL) return output;      // cioè return 0
      output = sottoalbero_omogeneo(a); // conta questo sottoalbero
      nodo_albero* x = a->left;          // scorro la lista dei figli
      while ( x != NULL ) {
            output = output + sottoalberi_omogenei(x);
            x = x->right;
      }
      return output;
}

/* funzione richiesta dal compito */

int verifica(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);                // visito e marco con 1
            }
            en = en->next;
      }

      return comp == sottoalberi_omogenei(a);
}
```