Algoritmi e Strutture di Dati – A.A. 2017-2018
Prova scritta del 19 febbraio 2018 – D.M. 270-9CFU
Libri e appunti chiusi – Tempo = 2:00h

9

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

……

……

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

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 AGGIUNGI_IN_CODA e CERCA_IN_CODA facciano entrambe un numero di operazioni proporzionali alla lunghezza della lista passata in input.

```
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(CERCA_IN_CODA(L,v.info) == FALSE)
      AGGIUNGI_IN_CODA(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 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 binario di interi. La funzione restituisce 1 se il numero di nodi della componente connessa più grande (cioè con più nodi) del grafo è uguale alla profondità del nodo con indice (campo info) più alto dell'albero, altrimenti restituisce 0.

Assumi che l'albero contenga tutti valori interi distinti e non negativi.

Usa le seguenti strutture:

```
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 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 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 $\Theta(n^2)$

ALGORITMO IN LINGUAGGIO C

```
#include <stdlib.h>

/* qui ci vanno le strutture fornite nel testo */

/* visita DFS del grafo rappresentato mediante oggetti e
   riferimenti e marcatura dei nodi raggiunti con il valore passato. */

void DFS(nodo* n, int mark) {

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

/* ritorna il numero di nodi della componente specificata */

int nodi_componente(grafo_oggetti* g, int comp){
      int cont = 0;                // questo sarà l'output
      elem_nodi* en = g->nodi;     // scorro la lista dei nodi
      while (en != NULL) {
            if(en->info->color == comp) {
                  cont++;
            }
            en = en->next;
      }
      return cont;
}

/* ritorna il massimo tra due interi */

int massimo(int a, int b) {
      if (a > b) return a;
      return b;
}
```

```
/* ritorna il massimo valore contenuto in un nodo nell'albero */

int massimo_albero(nodo_albero* a) {
      if ( a == NULL ) return -1;
      int left = massimo_albero(a->left);
      int right = massimo_albero(a->right);
      return massimo(a->info,massimo(left, right));
}

/* ritorna la profondità del nodo che contiene il valore intero id */

int profondita_nodo(nodo_albero* a, int id) {
      if (a == NULL) return -1;
      if (a->info == id) return 0;
      int left = profondita_nodo(a->left, id);
      int right = profondita_nodo(a->right, id);
      int max = massimo(left, right);
      if (max == -1) return -1;  // non lo abbiamo trovato
      else return max+1;
}

/* funzione richiesta dal compito */

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

      elem_nodi* en = g->nodi;
      while (en != NULL) {
            en->info->color = 0; // marco tutti i nodi con 0
            en = en->next;
      }
      int comp = 0;

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

      int nodi_comp_max = 0;
      int j;
      for (j=1; j <= comp; j++) {
            nodi_comp_max = massimo(nodi_comp_max, nodi_componente(g, j));
      }

      int prof_nodo = profondita_nodo(a, massimo_albero(a));
      return prof_nodo == nodi_comp_max;
}
```