Algoritmi e Strutture di Dati – A.A. 2011-2012
Seconda Prova dell'appello del 14 e 15 febbraio 2012

SOLUZIONI DELLA PSEUDOCODIFICA

Negli esercizi seguenti supponi che un grafo diretto sia rappresentato con un array A in cui ogni elemento A[*u*] è un riferimento al primo elemento della lista di adiacenza doppiamente concatenata del nodo *u*. Un albero binario T ha invece il solo riferimento alla radice T.root (ogni nodo dell'albero ha i campi parent, left, right e key).

Esercizio 1

Scrivi lo pseudocodice della procedura SOLO-RADICE(T) che prende in input un albero T e restituisce **true** se l'albero ha solo la radice, **false** altrimenti.

```
SOLO-RADICE(T)
     return (T.root.left == NIL) and (T.root.right == NIL)
```

Esercizio 2

Scrivi lo pseudocodice della procedura CONTA-NODI(T) che prende in input l'albero T e restituisce il numero dei nodi dell'albero T.

```
CONTA-NODI(T)  /* Input: un albero. Output: il numero dei suoi nodi */
     return CONTA-RIC(T.root)
```

```
CONTA-RIC(n)  /*  Input:  un  nodo.  Output:  il  numero  dei  nodi  del
                  sottoalbero radicato sul nodo */
     if (n == NIL) return 0
     return 1 + CONTA-RIC(n.left) + CONTA-RIC(n.right)
```

Esercizio 3

Scrivi lo pseudocodice della procedura GRAFO-DA-ALBERO(T) che prende in input un albero T (che contiene le chiavi 0, 1, 2,…, n) e restituisce un grafo A. Il grafo A contiene un nodo con indice *k* per ogni nodo dell'albero con chiave *k* e contiene un arco diretto dal nodo *u* al nodo *v* se nell'albero T *u* è parent di *v*. [Se ti serve, quando scrivi questa procedura puoi utilizzare la procedura CONTA-NODI(T) dell'Esercizio 2 o altre procedure di supporto (che però devi descrivere in dettaglio)].

```
GRAFO-DA-ALBERO(T)  /*  Input:  un  albero  T.  Output:  un  grafo  A  con  gli
                        archi diretti dai genitori ai figli */
     /* crea un array A con CONTA-NODI(T) posizioni */
     COSTRUISCI-RIC(T.root, A)
     return A
```

```
COSTRUISCI-RIC(n,A) /* aggiunge ad A tutti gli archi che vanno dai nodi
                       nel sottoalbero radicato su n ai loro genitori (n
                       può essere NIL) */
     if (n == NIL) return
     if( n.parent != NIL ) AGGIUNGI-ARCO(A, n.parent.info, n.info)
     COSTRUISCI-RIC(n.left, A)
     COSTRUISCI-RIC(n.right, A)
```

PROCEDURA ALTERNATIVA

```
COSTRUISCI-RIC-2(n,A) /* aggiunge ad A tutti gli archi che vanno dai nodi
                         nei sottoalberi destro e sinistro del nodo
                         corrente ai loro genitori (n non è mai NIL) */
     if( n.left!= NIL )
          AGGIUNGI-ARCO(A, n.info, n.left.info)
          COSTRUISCI-RIC-2(n.left, A)
     if( n.right!= NIL )
          AGGIUNGI-ARCO(A, n.info, n.right.info)
          COSTRUISCI-RIC-2(n.right, A)
```

```
AGGIUNGI-ARCO(A, u, v) /* Input: grafo A, indici u e v. Aggiunge in A
                          l'arco tra u e v */
     /* temp è un nuovo elemento con campi prev, next e info */
     temp.info = v
     temp.prev = NIL
     temp.next = A[u]
     if ( A[u] != NIL ) A[u].prev = temp
     A[u] = temp
```

Esercizio 4 (solo studenti D.M. 270/04)

Scrivi lo pseudocodice della procedura VERIFICA-ALBERO(A, *u*) che prende in input il grafo A e verifica che A rappresenti un albero con radice *u* e con gli archi diretti dai nodi genitori verso i nodi figli. [Suggerimento: non ci devono essere archi diretti verso nodi marcati come "raggiunti" durante una visita del grafo].

```
VERIFICA-ALBERO(A, u)
     /* color è un nuovo array con A.length posizioni */
     return = CHECK-DFS(A,u, color)
```

versione più sofisticata (inizializzazioni e controllo finale della connessione del grafo)

```
VERIFICA-ALBERO-2(A, u) /* versione più sofisticata del precedente */
     /* color è un nuovo array con A.length posizioni */
     for i = 0 to color.length-1     /* inizializzazione di color */
          color[i] = 0
     no-cicli = CHECK-DFS(A,u, color)
     tutto-visitato = true
     for i = 0 to color.length-1     /* chi ha omesso questo controllo
                                        non è stato penalizzato */
          if color[i] == 0
               tutto-visitato = false
     return no-cicli AND tutto-visitato
```

```
CHECK-ALBERO-DFS(A,u, color)    /* controlla che tutto ciò che è
raggiungibile da u sia un albero */
     color[u] = 1    /* marco u come raggiunto */
     albero = true
     x = A[u]
     while x != NIL
          k = x.info
          if (color[k] != 0) return FALSE /* sono tornato su un nodo già
                                             visitato. Non è un albero. */
          albero = albero AND CHECK-ALBERO-DFS(A, k, color)
          x = x.next
     color[u] = 2
     return albero
```

Esercizio 5

Discuti la complessità computazionale (nel solo caso peggiore) delle procedure che hai proposto per gli esercizi precedenti, utilizzando *n* per denotare il numero totale dei nodi dell'albero o del grafo.

SOLO-RADICE(T) = O(1) [in realtà Theta(1)]

CONTA-NODI(T) = O(n) [in realtà Theta(n). E' una visita dell'albero]

[CONTA-RIC(n) = O(p) dove p è il sottoalbero radicato sul nodo n]

GRAFO-DA-ALBERO(T) = O(n) [in realtà Theta(n). E' sostanzialmente una visita dell'albero]

[COSTRUISCI-RIC(n,A) = O(p) dove p è il sottoalbero radicato sul nodo n]

[COSTRUISCI-RIC-2(n,A) = O(p) dove p è il sottoalbero radicato sul nodo n]

[AGGIUNGI-ARCO(A,u,v) = Theta(1)]

VERIFICA-ALBERO(A,u) = stima grossolana: $O(n^2)$ occorre percorrere tutti gli archi

Stima + raffinata: O(n) in quanto se non è un albero la procedura termina. Dunque nel caso peggiore ho visitato un albero O(n) più un nodo connesso a tutti gli altri O(n)