Algoritmi e Strutture di Dati – Compito da 9 CFU

Testo e soluzioni dell'esame a distanza del 1° settembre 2020

Esercizio 1 (3 punti)

Discuti la complessità computazionale delle seguenti procedure 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 lista (vuota) di interi */
FUNZ_RIC(T.root,L)
return L

FUNZ_RIC(v,L)
if(v==NULL) return
AGGIUNGI_IN_CODA(L,v.info)
RIMUOVI_IN_TESTA(L)
FUNZ_RIC(v.left,L)
FUNZ_RIC(v.right,L)
```

Assumi che AGGIUNGI_IN_CODA faccia un numero di operazioni proporzionali alla lunghezza della lista corrente mentre RIMUOVI_IN_TESTA faccia un numero di operazioni costante.

Soluzione esercizio 1

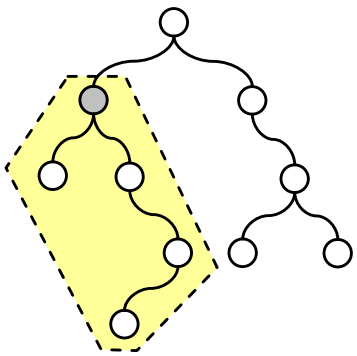
La funzione FUNZIONE(T), oltre a richiamare la funzione FUNZ_RIC(v,L), fa solamente operazioni con complessità asintotica Theta(1), quindi le due funzioni hanno la stessa complessità asintotica.

FUNZ_RIC(v,L) esplora tutto l'albero, determinando un costo complessivo Theta(n) se si escludono le operazioni con la lista. Ad ogni nodo dell'albero la funzione aggiunge un nodo in coda alla lista e ne rimuove uno in testa. La rimozione in testa ha costo Theta(1) e dunque complessivamente Theta(n) per tutte le chiamate. L'aggiunta in coda, pur avendo costo quadratico nella lunghezza della lista, ha costo Theta(1), cioè complessivamente Theta(n), poiché la lunghezza della lista è limitata dal fatto che ad ogni aggiunta avviene anche una rimozione. In conclusione il costo complessivo è Thata(n).

Esercizio 2 (27 punti)

Scrivi in linguaggio C il codice della funzione `int verifica(nodo* a)` che accetti in input un puntatore alla radice di un albero binario di interi e ritorni 1 se esiste un nodo dell'albero che è radice di un sottoalbero che ha esattamente la metà dei nodi dell'intero albero, altrimenti ritorna 0. Se i nodi dell'albero sono dispari la funzione `verifica()` ritorna ovviamente 0.

Per esempio con l'albero in figura la funzione `verifica()` ritorna 1 perché il nodo grigio è radice di un sottoalbero (evidenziato con il tratteggio) che ha 5 nodi, che sono esattamente la metà dei nodi totali dell'albero (che ha in tutto 10 nodi).



Utilizza la seguente struttura:

```
/* struttura nodo per l'albero binario */

typedef struct nodo_struct {
     struct nodo_struct* left;
     struct nodo_struct* right;
     int info;
} nodo;
```

Soluzione esercizio 2

```
/* calcola il numero dei nodi di un albero o sottoalbero */

int num_nodi(nodo* a){
     if(a == NULL) return 0;
     return  1 + num_nodi(a->left) + num_nodi(a->right);
}

/* verifica se esiste un sottoalbero che abbia il numero
   di nodi passato come parametro */

int esiste_sottoalbero(nodo* a, int nodi){
```

```
    if (a == NULL) return 0;
    if (num_nodi(a) == nodi) return 1;
    return esiste_sottoalbero(a->left, nodi) ||
           esiste_sottoalbero(a->right, nodi);
}

/* funzione richiesta */

int  verifica(nodo* a){
     int nodi = num_nodi(a);       // nodi dell'albero
     if (nodi % 2 == 1) return 0;  // se sono dispari
     return esiste_sottoalbero(a,nodi/2);

     /* nota: scrivendo "nodi/2" abbiamo fatto una
        divisione tra interi. Se "nodi" fosse dispari
        questo corrisponderebbe ad un arrotondamento in
        basso (e ad un test non corretto).
        Per questo ritorniamo 0 se nodi è dispari
        nella riga sopra. (Un errore su questa parte non
        sarebbe comunque penalizzato gravemente nella
        valutazione).
}
```