Algoritmi e Strutture di Dati – Compito da 9 CFU – A

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

Esercizio 1 (3 punti)

Discuti la complessità computazionale della seguente procedura 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 nuova lista (vuota) di interi */
FUNZ-RIC(T.root,L)
return L

FUNZ-RIC(v,L)
if(v==NULL) return
contaFigli = 0
if(v.right != NULL) contaFigli++
if(v.left != NULL) contaFigli++
if(contaFigli == 1)
     AGGIUNGI-IN-CODA(L,v.info)
else
     AGGIUNGI-IN-TESTA(L,v.info)
FUNZ-RIC(v.left,L)
FUNZ-RIC(v.right,L)
```

Assumi che AGGIUNGI-IN-TESTA faccia un numero di operazioni costante, mentre AGGIUNGI-IN-CODA faccia un numero di operazioni proporzionali alla lunghezza della lista corrente. Fai un esempio di un albero in cui si ha il caso peggiore di complessità asintotica.

Soluzione esercizio 1

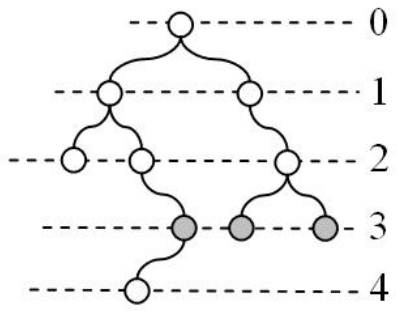
La funzione FUNZIONE(T), oltre a richiamare la funzione FUNZ-RIC(v,L), fa solamente operazioni O(1), quindi le due funzioni hanno la stessa complessità asintotica.

La funzione FUNZ-RIC(v,L) visita l'intero albero. Nel caso peggiore l'albero ha solo nodi interni con un figlio ed un'unica foglia, è cioè un cammino. In questo caso FUNZ-RIC(v,L) esegue aggiungi-in-coda O(n) volte, con un costo complessivo Theta(n^2).

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 una profondità dell'albero per cui il numero di nodi a quella profondità è esattamente uguale alla profondità stessa. Altrimenti ritorna 0.

Per esempio nell'albero in figura la funzione verifica() ritorna 1 perché a profondità 3 ci sono 3 nodi (quelli grigi).



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

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

int altezza_albero(nodo* a){
     if(a == NULL) return -1;
     int l = altezza_albero(a->left);
     int r = altezza_albero(a->right);
     if(l > r) return l + 1;
     return r + 1;
}
```

```
int verifica(nodo* a){
    int h = altezza_albero(a);
    int i;
    for(i = 0;  i <= h; i++){
        if(i == nodi_a_profondita(a,i)
        return 1;
    }
    return 0;
}
```