Algoritmi e basi di dati – modulo Algoritmi e Strutture dati – a.a. 2011/2012

20 luglio 2012

Cognome	Nome	Matricola

1	2	3	4	5

Esercizio 1

Confrontare le due funzioni F e G dal punto di vista della complessità: dire se una è O dell'altra e viceversa. In caso affermativo, indicare una coppia (n_0,c). In caso negativo, giustificare la risposta.

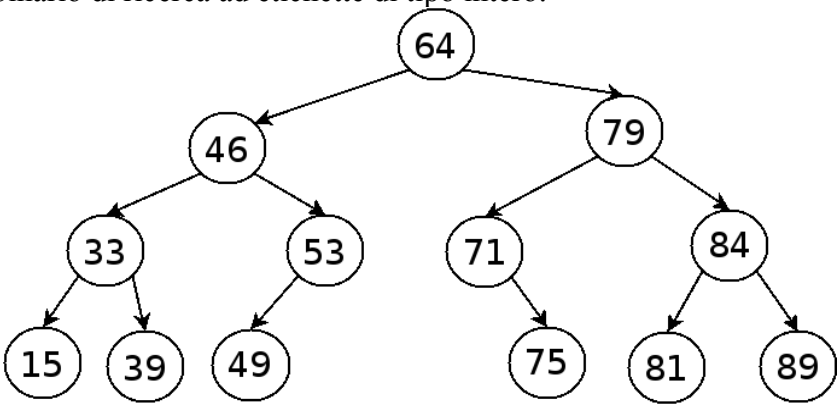
F(x)=	$10x^4$	se x<100	G(x)=	$12x^3$	se x multiplo di 5
	x	se x>=100 e quadrato perfetto			
	4x	altrimenti		3x+2	altrimenti

F è O(G) basta scegliere un n_0 maggiore o uguale a 100 e c=2

G non è O(F) in quanto ci sono infiniti multipli di 5

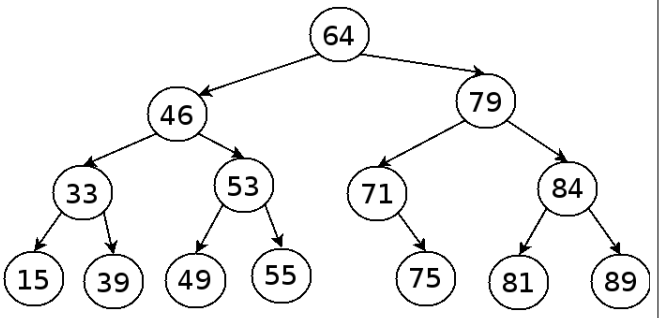
Esercizio 2

Sia dato l'albero binario di ricerca ad etichette di tipo intero:

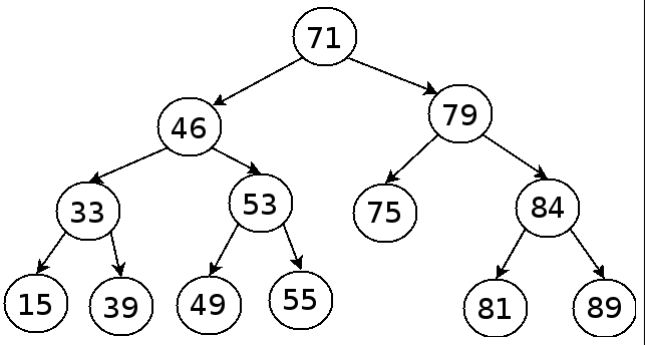


Mostrare l'albero:
a) dopo l'inserimento di un nodo con etichetta 55
b) dopo la cancellazione del nodo con etichetta 71

a)



b)



Esercizio 3

Calcolare la complessità in funzione di **n**>0 dell'istruzione

y=g(g(n))

con le funzioni **f** e **g** definite come segue:

```
int f(int x) {
   if (x<=1) return 1;
   int b=0, i, j, c;
   for (i=1; i<=x; i++) b+=i;
   c = b;
   for (j=1; j<=c; j++) b+=i;
   return b;
}
```

```
int g(int x) {
  if (x<=1) return 10;
  int a=0;
  for (int i=0; i<f(x); i++)
      a++;
   return 10+16*g(x/2);
}
```

Indicare le eventuali relazioni di ricorrenza e spiegare il calcolo della complessità dei cicli.

Stima del tempo di f():

Primo for:
numero iterazioni = O(n)
complessità di un'iterazione = costante
tempo del for = O(n)

Secondo for:
numero iterazioni for = $O(n^2)$
complessità di un'iterazione = costante
tempo del for = $O(n^2)$

T_f è $O(n^2)$

b= $O(n^2+ n^4)$

$R_f(n)$ è $O(n^4)$

Tempo di g(g(n)):

$C[g(n)] + C[g(n^4)] = O(n^6) + O((n^4)^6) = O(n^{24})$

Stima del tempo di g():

Numero iterazioni del for: $R_f(m)= O(m^4)$
complessità di un'iterazione: $T_f(m) = O(m^2)$
tempo del for: $O(m^6)$

Tempo di g:
$T_g(1)$= cost
$T_g(m)= c \cdot m^6 + T_g(m/2)$
T_g è $O(m^6)$

$R_g(1)$= cost
$R_g(m)$= cost + 16*$T_g(m/2)$

$R_g(m)$ è $O(n \text{ ^}(\log_2 16)) = O(n^4)$

Esercizio 4

Sia dato un albero binario ad etichette intere. Scrivere una funzione che, per ogni nodo conta il numero dei discendenti di sinistra e destra e (eventualmente) scambia il sottoalbero destro con il sinistro, in modo che alla fine il sottoalbero sinistro contenga meno nodi del sottoalbero destro. Calcolare la complessità in funzione del numero dei nodi **`N`** dell'albero.

Soluzione 1: complessità O(N)

```
int scambia(Node* t) {
  if (!t)
    return 0;
  int l = scambia(t->left);
  int r = scambia(t->right);
  if (l>r) {
    Node* temp = t->left;
    t->left = t->right;
    t->right = temp;
  }
  return l+r+1;
}
```

Soluzione 2: complessità O(N log N)

```
int nodes(const Node* t) {
  if (!t)
    return 0;
  return nodes(t->left)+
```

```
void scambia(Node* t) {
  if (!t)
    return;
  int l = nodes(t->left);
```

```
          nodes(t->right)+1;
}
```

```
int r = nodes(t->right);
if (l>r) {
  Node* temp = t->left;
  t->left = t->right;
  t->right = temp;
}
scambia(t->left);
scambia(t->right);
}
```

Esercizio 5

Scrivere una funzione che, dato un albero generico non vuoto ad etichette di tipo **`int`**, memorizzato figlio-fratello, restituisce il numero dei nodi dell'albero con etichetta maggiore del livello del nodo stesso. Calcolare la complessità in funzione del numero dei nodi **`N`** dell'albero.

```
int conta_maggiori(const Node* t, int lvl = 0) {
  if (!t)
    return 0;
  return (t->label>lvl)+
    conta_maggiori(t->left,lvl+1) +
    conta_maggiori(t->right,lvl);
}
```

Complessità: O(**`N`**)