FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

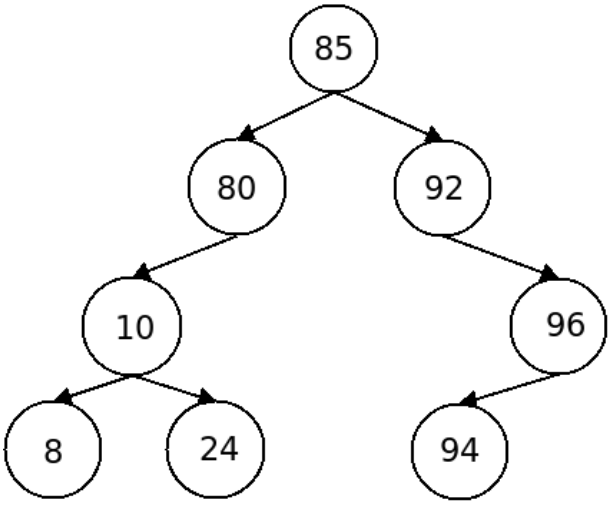
19 febbraio 2010 - ANNO ACCADEMICO 2008/09

NOME: **COGNOME:** **MATRICOLA:**

SCRITTO RIDOTTO: **SI** **NO** **(barrare SI se si intende svolgere lo scritto ridotto)**

Esercizio 1

Sia dato il seguente albero binario di ricerca ad etichette intere:

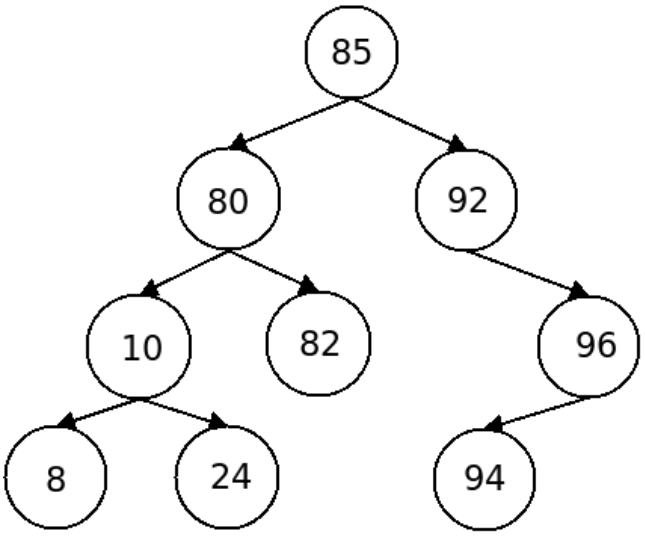


Mostrare l'albero

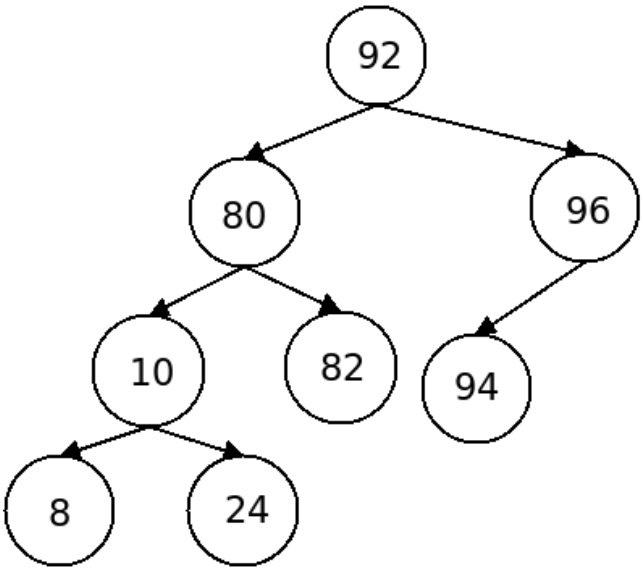
a) dopo l'inserimento del valore 82

b) dopo la cancellazione del valore 85 dall'albero ottenuto al punto a)

a)



b)



Situazione	*Esercizi*	*Tempo*
Senza progetto	**1,2,3,4,5**	80 min
Con progetto	**1,2,3**	40 min

1	*2*	*3*	*4*	*5*

Esercizio 2
Siano date due liste non ordinate contenenti etichette di tipo intero. Le liste possono contenere duplicati. Si supponga che le liste contengono ciascuna n elementi e al massimo m elementi diversi ciascuna. Scrivere **a parole** due algoritmi per controllare se ogni elemento presente nella prima lista è presente anche nella seconda. Il primo algoritmo (a) dovrà essere il più veloce possibile, mentre il secondo (b) dovrà occupare meno memoria possibile. Calcolare la complessità degli algoritmi proposti (in termini di spazio e tempo) dettagliando il costo di ogni operazione e supponendo che le liste contengano ciascuna n elementi.

a) Un algoritmo efficiente in termini di tempo consiste nel memorizzare la seconda lista in un albero binario di ricerca (costo in termini di tempo pari a O(n log n)) e poi cercando ogni elemento della prima lista nell'albero (costo in termini di tempo pari a O(n log n)). Complessivamente il tempo impiegato è pari a O(n log n). La memoria aggiuntiva occupata è pari a O(n). Alternativamente si può utilizzare una tabella hash con funzione hash iniettiva, occupando potenzialmente molta memoria, ma facendo scendere le operazioni da O(n log n) a O(n).

b) se non vogliamo occupare altra memoria aggiuntiva effettuare una ricerca completa nella seconda lista degli elementi della prima. La complessità totale è pari a $O(n^2)$.

Esercizio 3
Calcolare la complessità dell'istruzione

```
for (int i=0; i<=g(f(t)); i++) cout << 3;
```

(indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione del numero di nodi di t. Supporre che t sia un albero binario bilanciato e le funzioni **f** e **g** siano definite come segue:

```
int f(Node* t) {
   if (! t)
     return 1;
   cout >> 2*f(t->left);
   return 2+f(t->left) +f(t->right);
}
```

```
int g(int x) {
  int a=0;
  int n = f(t);
  for (int i=0; i<f(t); i++)
     a+=i;
  return a;
}
```

Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità dell'iterazione singola.

Tf(0) = cost
Tf(n) = 3T(n/2)+cost
Tf è O(n^log2(3)) circa O(n^1.5)

Rf(0) = 1
Rf(n) = cost+2Rf(n/2)
Rf è O(n)

Calcolo di Tg

Numero di iterazioni del for = Rf(n)=O(n)
Complessità singola iterazione = Tf(n) =O(n^1,5)
Complessità del for Tf(n)*Rf(n) = O(n^2.5)

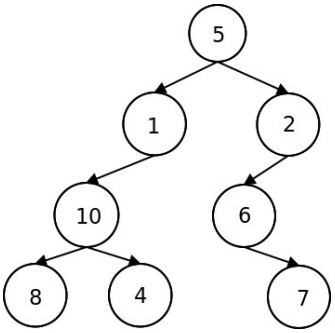
Tg(n) = Tf(n)+complessità del for = O(n^1.5) +O(n^2.5)= O(n^2.5)

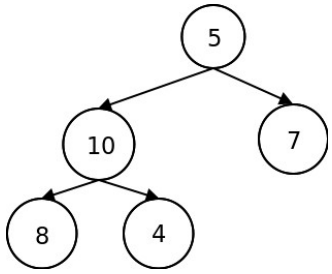
Rg(n) =Rf(n)^2 = O(n^2) (somma dei primi Rf(n) numeri naturali)

complessità istruzione
Numero di iterazioni = Rg(Rf(n)) = O(n^2)
Complessità singola iterazione = Tf(n)+Tg(Rf(n)) = Tf(n)+Tg(n) = O(n^1.5)+O(n^2.5) = O(n^2.5)

Complessità totale for = O(n^2)*O(n^2.5) = O(n^4.5)

Esercizio 4
Sia dato un albero binario ad etichette di tipo intero. Si scriva una funzione compatta che cancella dall'albero i nodi che hanno un solo figlio. Il figlio di ogni nodo da cancellare dovrà essere agganciato al padre del nodo da cancellare. Ad esempio, applicando la funzione compatta all'albero di sinistra, si ottiene l'albero di destra






```
void compatta(Node*& t) {
  if (t) {
    if (t->left && !t->right) {
      Node* tmp = t->left;
      delete t;
      t = tmp;
      compatta(t);
    } else if (!t->left && t->right) {
      Node* tmp = t->right;
      delete t;
      t = tmp;
      compatta(t);
    } else {
      compatta(t->left);
      compatta(t->right);
    }
  }
}
```

Esercizio 5
Sia dato un albero generico ad etichette intere memorizzato figlio fratello. Si scriva una funzione conta che prende in ingresso un intero maggiore o uguale a zero k e restituisce il numero dei sottoalberi che hanno esattamente k foglie.

```
int contafoglie(Node* t) {
  if (!t) return 0;
  return  (!t->left) + contafoglie(t->left)+contafoglie(t->right);
}

int kfoglie(Node* t, int k) {
  if (!t)
    return 0;
  return ((!t->left)+contafoglie(t->left) == k) +
    kfoglie(t->left,k) +
    kfoglie(t->right,k);
}
```