Algoritmi e Principi dell'Informatica - Modulo 2 - Informatica 3

Prof. Giovanni Agosta
Prof. Alessandro Campi
Prof. Maristella Matera

Prova scritta – Appello del 04/02/2011[1]

COGNOME e NOME:..... Matricola:.....
SEZIONE: Agosta ○ Campi ○ Matera ○

[1]Tempo: 2 ore. Possono essere consultati libri di testo e slide del corso. È consentito scrivere a matita. Scrivere il proprio nome sugli eventuali fogli aggiuntivi. *Rispondere punto per punto alle domande!*

Esercizio 1 (10 punti)

Dato un albero binario, definiamo altezza minimale di un nodo v la minima distanza di v da una delle foglie del suo sottoalbero, definiamo invece altezza massimale di un nodo v la massima distanza di v da una delle foglie del suo sottoalbero. Supponiamo di avere un albero T che contiene in ogni nodo anche un numero intero m. Diciamo che l'albero è k-equilibrato se tutti i nodi (eccetto al più k) contengono un valore m che è compreso tra l'altezza minimale e l'altezza massimale del nodo stesso. Definire un algoritmo che riceve in input l'albero T e un intero k e restituisce vero se l'albero T è k-equilibrato, falso altrimenti. Calcolare la complessità dell'algoritmo proposto.

```
count_unbalanced(v):
   if v = NIL then 0, MAXINT, 0
   if leaf[v] then
      if m[v] != 0 then return 1, 0, 0
      return 0, 0, 0
   kl, minl, maxl <- count_unbalanced(left[v])
   kr, minr, maxr <- count_unbalanced(right[v])
   minv <- min(minl,minr)+1
   maxv <- max(maxl,maxr)+1
   if m[v] > maxv or m[v] < minv then
       return kr+kl+1, minv, maxv
   return kr+kl, minv, maxv

is_balanced(T,k):
   if count_unbalanced(root[T])[0] > k then
      return FALSE
   return TRUE
```

La funzione `count_unbalanced` riceve in ingresso un nodo v, e restituisce come risultati il numero di nodi non equilibrati nel sottoalbero radicato in v, l'altezza minimale di v e l'altezza massimale di v.

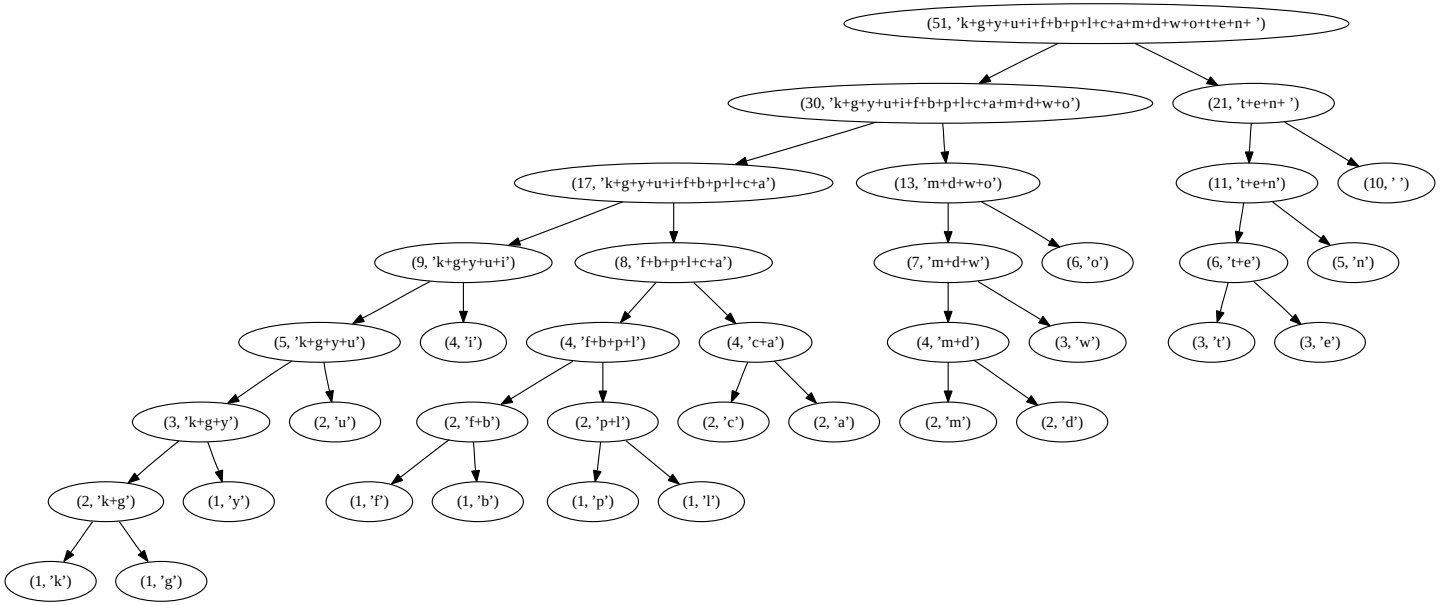
La complessità è quella della visita in postordine. Ogni nodo viene visitato una ed una sola volta, quindi $T(n) = \Theta(n)$, dove n è il numero dei nodi di T.

Esercizio 2 (10 punti)

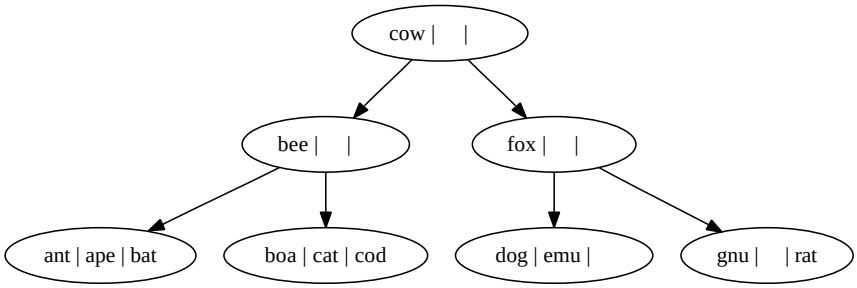
1. Determinare la codifica di Huffman della seguente frase: *it would be nice to know an optimum way of encoding*

   Frequenze:

   ```
   (1, b), (1, f), (1, g), (1, k), (1, l), (1, p), (1, y),
   (2, a), (2, c), (2, d), (2, m), (2, u), (3, e), (3, t),
   (3, w), (4, i), (5, n), (6, o), (10, ' ' )
   ```


(51, 'k+g+y+u+i+f+b+p+l+c+a+m+d+w+o+t+e+n+ ')
(30, 'k+g+y+u+i+f+b+p+l+c+a+m+d+w+o')
(21, 't+e+n+ ')
(17, 'k+g+y+u+i+f+b+p+l+c+a')
(13, 'm+d+w+o')
(11, 't+e+n')
(10, ' ')
(9, 'k+g+y+u+i')
(8, 'f+b+p+l+c+a')
(7, 'm+d+w')
(6, 'o')
(6, 't+e')
(5, 'n')
(5, 'k+g+y+u')
(4, 'i')
(4, 'f+b+p+l')
(4, 'c+a')
(4, 'm+d')
(3, 'w')
(3, 't')
(3, 'e')
(3, 'k+g+y')
(2, 'u')
(2, 'f+b')
(2, 'p+l')
(2, 'c')
(2, 'a')
(2, 'm')
(2, 'd')
(2, 'k+g')
(1, 'y')
(1, 'f')
(1, 'b')
(1, 'p')
(1, 'l')
(1, 'k')
(1, 'g')

2. Inserire le seguenti stringhe in un 2-3-4 Tree (ovvero un BTree con 1-3 chiavi e 2-4 figli per nodo) *oppure* in un albero Red-Black (indicare la scelta!) inizialmente vuoto: *cod, bee, ape, cow, rat, fox, bat, cat, dog, gnu, emu, boa, ant.*



Esercizio 3 (10 punti)

1. Determinare la complessità computazionale del seguente frammento di codice C in funzione della lunghezza della stringa A.

```
int foo(char A[], int n, int m){
   int i, a=0;
 if (n>=m) return 0;
   for(i=n; i<m; i++)
     a+=A[i];
   return a + foo(A, n*2, m/2);
}

int bar(char A[]){
   return foo(A,1,strlen(A));
}
```

   La funzione *bar* invoca la funzione foo, passando come parametri la stringa A, e due indici $n = 1$ ed $m = |A|$. Quindi, all'interno della prima esecuzione di foo troviamo un ciclo con $m - 1$ iterazioni, equivalenti ad una complessità asintotica $\Theta(|A|)$.

   foo viene poi invocata ricorsivamente. Ciascuna invocazione ricorsiva opera su un frammento della stringa delimitato dagli indici n ed m, che rispettivamente raddoppiano e dimezzano il proprio valore ad ogni invocazione. Quindi, nella seconda invocazione il ciclo opererà su $|A|/2 - 2$ elementi della stringa, e in generale nell'invocazione i-esima, il ciclo opererà su $|A|/2^i - 2^i$ elementi. Quindi, il numero totale di operazioni sarà al più a $\Sigma_{i=0}^{(log_2|A|)/2}|A|/2^i - 2^i$.

   La soluzione di questa serie può risultare tediosa da calcolare. Ricorriamo allora ad una semplice considerazione: maggioriamo la serie con $\Sigma_{i=0}^{log_2|A|}|A|/2^i$. Otterremo così un limite di complessità asintotica O invece di Θ. La soluzione della seconda serie è però elementare, in quanto la sommatoria di $1/2^i$ tende a 2. Quindi, $T(|A|) = O(|A|)$.

   Se invece si risolve la serie in modo preciso, $T(|A|) = 2|A| - 3\sqrt{(}|A|) + 1$.

2. Determinare i bound di complessità asintotica più stretti per le seguenti ricorrenze:

   (a) $T(n) = T(n-1) + T(n-3) + \Theta(nlogn)$

Questa ricorrenza è analoga a quella relativa al calcolo della serie di Fibonacci. Dato che le chiamate ricorsive esplorano un albero binario completo fino ad una altezza di $n/3$, la complessità non può essere inferiore a $\Omega(2^{n/3})$. D'altra parte, l'unico altro contributo significativo è dato dal passo di ricombinazione $\Theta(nlogn)$, e l'albero esplorato ha meno di 2^n nodi, $T(n) = O(2^n nlogn)$.

(b) $T(n) = 4T(n/3) + n^5$
$\Theta(n^5)$, per il Teorema Maestro.

(c) $T(n) = T(n-5) + n^2 + 5n$
$\Theta(n^3)$.

(d) $T(n) = 2T(n/4) + n^2 logn$
$\Theta(n^2 logn)$, per il Teorema Maestro.

(e) $T(n) = 3T(n/2) + \sqrt{n}$
$\Theta(n^{log_2 3})$, per il Teorema Maestro.