FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

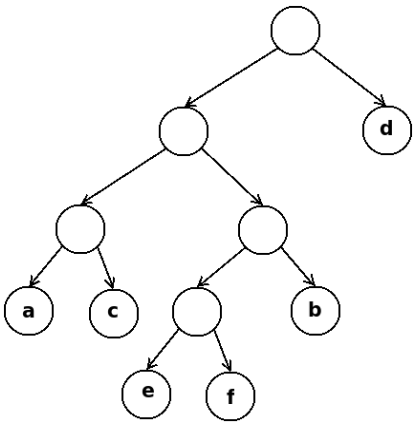
9 giugno 2008 - ANNO ACCADEMICO 2007/08

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

Esercizio 1
Siano date le seguenti coppie (simbolo, nr. di occorrenze). Utilizzando l'algoritmo di Huffman si costruisca il codice ottimo (i sottoalberi sinistri devono corrispondere alla codifica 0 e i destri alla codifica 1, il primo elemento estratto dallo heap ad ogni iterazione deve essere inserito come figlio sinistro del nuovo nodo creato). Riportare solamente il codice nella tabella e l'albero

{ ('a', 40), ('b', 75), ('c', 50), ('d', 250), ('e', 20), ('f', 35) }

Simbolo	Codice
a	000
b	011
c	001
d	1
e	0100
f	0101



Esercizio 2
Scrivere una funzione che, dato un albero generico non vuoto ad etichette intere, memorizzato figlio-fratello, e un intero k >0, restituisce l'etichetta del k-esimo figlio della radice. Se la radice ha meno di k figli la funzione deve restituire il valore -1.

```
int figliok(Node* t, int k) {
 Node* f = t->left;
 for (int i=1;f && (i<k);i++)
     f = f->right;
 if (f) return f->label;
 else return -1;
}
```

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

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

Esercizio 3

Sia **t** la radice di un albero binario quasi bilanciato. Calcolare la complessità del blocco in funzione del numero di nodi di **t** (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione):

```
{
  int a = 0;
  for (int i=0; i <= f(t)*g(t); i++)
     a += 1;
}
```

con le funzioni **f** e **g** definite come segue e assumendo che la funzione **nodes** sia quella definita nel corso:

`int f(Node *tree) {`
`  if (!tree) return 1;`
`  int a = f(tree->left)+`
`  f(tree->right);`
`  int b = 2*f(tree->left);`
`  return 1+a+b;`
`}`
	`int g(Node *tree) {`
	`  if (!tree) return 1;`
	`  int a=0;`
	`  for (int i=0; i<=nodes(tree);i++)`
	`  a+=1;`
	`  cout << g(tree->left) + g(tree->right);`
	`  return a;`
	`}`

Stima del tempo di f

$T_f(0)= d$
$T_f(n)= c + 3T_f(n/2)$

T_f è $O(n^{\log2(3)})$ circa $O(n^{1.5})$

Stima del risultato

$R_f(n) =1 +4R_f(n/2)$
$R_f(n)$è $O(n^{\log2(4)})=O(n^2)$

Stima del tempo di g:
numero iterazioni del for: O(n)
complessità di un'iterazione: O(n)
complessità statement iniziale: O(1)
tempo del for: $O(n^2)$

tempo di g
$T_g(1)= cost$
$T_g(m)= c\cdot n^2 + 2T_g(n/2)$

T_g è $O(n^2)$

Stima del risultato di g
$R_g(n) = O(n)$

Complessità del for del blocco:
numero iterazioni del for: $R_f(n)*R_g(n) = O(n^3)$
complessità di un'iterazione: $T_f(n)+T_g(n) = O(n^{1.5})+O(n^2) = O(n^2)$
complessità statement iniziale: O(1)
Complessità del blocco $= O(n^5)$

Esercizio 4

Siano dati due alberi binari di ricerca non vuoti ad etichette intere A1 e A2. Si supponga che gli alberi abbiano rispettivamente n1 e n2 nodi. Scrivere una funzione che controlla se le etichette di A1 sono tutte minori delle etichette di A2. Calcolarne la complessità in funzione di n1 e n2 nel caso siano alberi quasi bilanciati.

```
int min(Node* t) {
  while (t->left)
    t = t->left;
  return t->label;
}

int max(Node* t) {
  while (t->right)
    t = t->right;
  return t->label;
}

bool isLessThan(Node* a1, Node* a2) {
  return (max(a1)<=min(a2));
}
```

complessita` O(log(n1)+log(n2))

Esercizio 5
Dimostrare per induzione naturale sul livello che in un albero binario completamente bilanciato di livello k le foglie sono 2^k.

Indichiamo con NF(A) il numero di foglie dell'albero A e dimostriamo per induzione sul livello k dell'albero.

Caso Base. Un albero binario completamente bilanciato di livello 0 è composto da un solo nodo che è anche foglia. Poiché $2^0=1$ la tesi è dimostrata.

Passo Induttivo. Sapendo che un albero binario completamente bilanciato di livello k ha 2^k foglie, con k maggiore o uguale a zero, dimostrare che un albero binario completamente bilanciato di livello k+1 ha 2^{k+1} foglie. Un albero binario completamente bilanciato A di livello k+1 è composto dalla radice che ha due sottoalberi A_s e A_d che sono alberi binari bilanciati di livello k. Poiché il numero di foglie di A è dato dal numero di foglie di A_s e più quelle di A_d, allora $NF(A) = NF(A_s) + NF(A_d) = 2^k + 2^k = 2^{k+1}$.