FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

14 gennaio 2010 - ANNO ACCADEMICO 2008/09

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

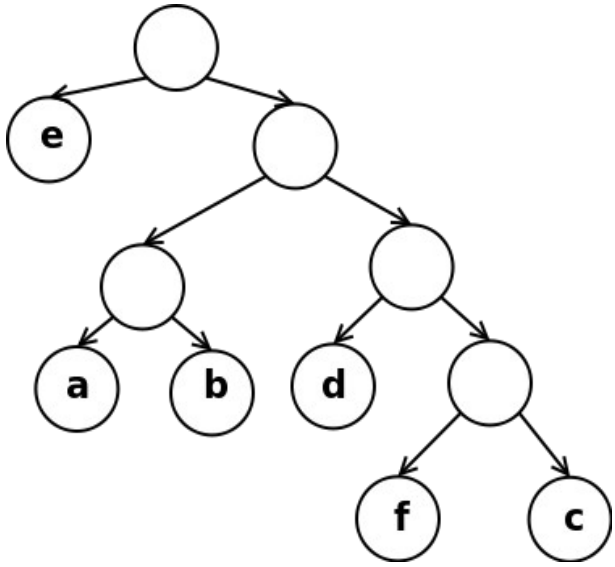
SCRITTO RIDOTTO: **SI** **NO** **(barrare SI se si intende svolgere lo scritto ridotto)**

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', 80), ('b', 100), ('c', 75), ('d',115), ('e', 302), ('f', 70) }

Simbolo	Codice
a	100
b	101
c	1111
d	110
e	0
f	1110



Esercizio 2

Sia dato un albero binario di ricerca contenente etichette di tipo short (2 byte ciascuna). Si supponga che un puntatore occupi 4 byte e che il numero di etichette n sia fissato e non vari nel tempo. Se volessi utilizzare una tabella hash ad indirizzamento aperto per contenere le stesse etichette, quale sarebbe il fattore di carico alfa (n/k) necessario ad avere la stessa occupazione di memoria dell'albero binario di ricerca? Quali proprietà perdo usando una tabella hash rispetto ad un albero binario di ricerca?

Ogni nodo dell'albero contiene l'etichetta più due puntatori per un totale di 10 byte. L'albero occuperà quindi 10n byte. Nella tabella hash invece, ogni casella occupa solo 2 byte, per un totale di 2k byte (dove k è il numero di celle). Quindi, uguagliando le due dimensioni, si ottiene alfa = 0.2.

Utilizzando un albero binario di ricerca abbiamo anche informazioni sull'ordine delle etichette (ad esempio con una visita opportuna otteniamo le etichette in ordine crescente), mentre questo non è più possibile con la tabella hash che ha “mescolato” le etichette.

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 3
Calcolare la complessità dell'istruzione g(t) e del suo risultato (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 con le funzioni **f** e **g** definite come segue:

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

```
int g(Node * t) {
  int a=0,i;
  for (i=0; i <= f(t); i++)
     a++;
  for (i=0; i <= 2*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.

$T_f(0)= k_1$
$T_f(n)= k_2 + 3\ T_f(n/2)$ $T_f(n)$ è $O(n^{\log_2 3})$ circa $O(n^{1,5})$
$R_f(0)= 1$
$R_f(n)= 1 + R_f(n/2)$ $R_f(n)$ è $O(\log n)$

Calcolo $T_g(n)$
Entrambi i for hanno
numero iterazioni: $O(\log n)$ Complessità singola iterazione: $O(n^{1,5})$
complessità dei for: $O(n^{1,5} \log n)$
$T_g(n)$ è $O(n^{1,5} \log n)$

il valore di a alla fine del primo for è $O(\log n)$
nel secondo for viene aggiunto ad a la somma dei primi $(2 \log n)$ numeri naturali che è $O((\log n)^2)$
alla fine della funzione a vale $O(\log n)+O((\log n)^2)=O((\log n)^2)$
$R_g(n)$ è $O((\log n)^2)$

Esercizio 4
Sia dato un albero binario ad etichette di tipo **`string`**, Scrivere una funzione **`contaliv`** che restituisce il numero di nodi dell'albero che hanno un'etichetta di dimensione maggiore del livello in cui in cui si trovano. Calcolare la complessità della soluzione proposta in funzione del numero di nodi n dell'albero.

```
int contaliv(Node* t, int liv = 0) {
  if (!t)
    return 0;
  return (t->label.length()>liv)+
    contaliv(t->left,liv+1)+
    contaliv(t->right,liv+1);
}
```

Complessità O(n)

Esercizio 5
Sia dato un albero generico ad etichette intere memorizzato figlio fratello. Si scriva una funzione **`media`** che prende in ingresso un intero maggiore o uguale a zero k e restituisce la media delle etichette di livello k. Si supponga che l'albero abbia almeno k livelli.

```
int calc_media(Node* t, int liv, int& somma, int k) {
  if (liv>k || !t)
    return 0;
  int nodi = calc_media(t->left,liv+1,somma,k)+
    calc_media(t->right,liv,somma,k);
  if (liv == k) {
    somma+= t->label;
    return 1+nodi;
  }
  return nodi;
}

double media(Node* t, int k) {
  int s = 0;
  int n = calc_media(t,0,s,k);
  return s/((double) n);
}
```