FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

30 gennaio 2009 - ANNO ACCADEMICO 2007/08

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

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

Esercizio 1

Un albero ternario è un albero in cui ogni nodo può avere fino a 3 figli. Si dimostri per induzione naturale che un albero ternario di livello k ha un numero di nodi compreso fra k+1 e $(3^{k+1}-1)/2$.

Sia m(k) = k+1 e M(k) = $(3^{k+1}-1)/2$.

Caso base k=0

Un albero ternario di livello 0 ha un solo nodo. Poiché m(0) = 0+1 = 1 e M(0) = (3-1)/2=1 il caso base è dimostrato.

Passo induttivo

Dobbiamo dimostrare che m(k+1) = k+2 noto che m(k) = k+1. L'albero ternario di livello k+1 con il minor numero di nodi è formato dalla radice e un solo sottoalbero di livello k con il minor numero di nodi quindi m(k+1) = m(k) + 1 = k+1+1 = k+2. Dobbiamo inoltre dimostrare che M(k+1)=$(3^{k+2}-1)/2$ noto che M(k) = $(3^{k+1}-1)/2$. L'albero ternario di livello k+1 con il maggior numero di nodi è formato dalla radice più tre sottoalberi di livello k, ciascuno con il numero massimo di nodi. Quindi

M(k+1) = 3M(k)+1 = $3(3^{k+1}-1)/2 +1 = (3^{k+2}-3+2)/2 = (3^{k+2} -1)/2$.

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

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

Esercizio 2
Data la lista di interi con puntatore alla testa **`p`**

22 → 67 → 4 → 3 → 45 → 12 → 6

scrivere tutte le chiamate a **`mergeSort`** generate da **`mergeSort(p)`**, specificando, per ogni chiamata, la lista a cui la funzione viene applicata. Non scrivere le chiamate a liste vuote.

Invocazione
mergeSort(22 → 67 → 4 → 3 → 45 → 12 → 6)
mergeSort(22 → 4 → 45 → 6)
mergeSort(22 → 45)
mergeSort(22)
mergeSort(45)
mergeSort(6 → 4)
mergeSort(6)
mergeSort(4)
mergeSort(12 → 3 →67)
mergeSort(12 → 67)
mergeSort(12)
mergeSort(67)
mergeSort(3)

Esercizio 3
Si scriva una funzione che, dato uno heap contenente numeri interi, memorizzato in un array, costruisce uno heap equivalente memorizzato come lista multipla (ogni nodo dell'albero ha due puntatori). Si calcoli la complessità della soluzione in funzione del numero n di elementi nello heap.

```
Node* buildTree(int h[], int n, int i=0) {
  if (i>=n) return NULL;
  Node* t = new Node<int>();
  t->label = h[i];
  t->left = buildTree(h,n,i*2+1);
  t->right = buildTree(h,n,i*2+2);
  return t;
}
```

complessità(n) = O(n)

Esercizio 4
Calcolare la complessità della chiamata

`f(g(n))`

in funzione di **`n`** (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni **`f`** e **`g`** definite come segue:

```
int f(int x) {
   if (x<=1) return 1;
   cout << 2*f(x/2);
   return 1+3*f(x/2);
}
```

```
int g(int x) {
   if (x<=1) return 3;
   for(int i=1;i<=f(x)*f(x);i++)
      cout << i;
   return g(x-1);
}
```

$Tf(1)=a$ $\quad O(n)$
$Tf(n)=b+2Tf(n/2)$

$Rf(1)=1$ $\quad O(n^{\log_2 3}) = O(n^{1,5})$
$Rf(n)=1+3Rf(n/2)$

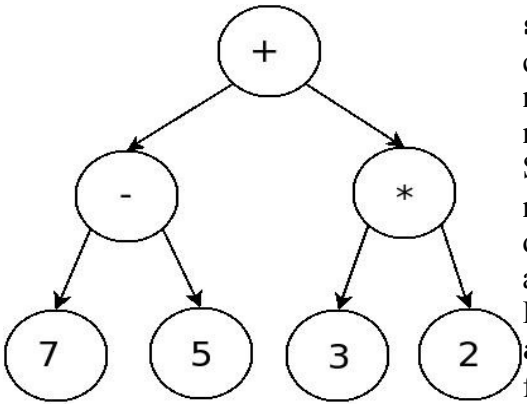
for di g:
Numero di iterazioni: $O(m^3)$
Complessità di una iterazione: $O(m)$
Complessità: $O(m^5)$

$Tg(1)=a$ $\quad O(m^5)$
$Tg(m)= m^4 + Tg(m-1)$

$Rg(m)=3$

$C[f(g(n))] = Tg(n) + Tf(Rg(n)) = Tg(n) + Tf(3) = O(m^5) + O(1) = O(m^5)$

Esercizio 5



Sia dato un albero binario non vuoto ad etichette di tipo **`std::string`**. L'albero rappresenta un'espressione. I nodi che sono foglie contengono stringhe che rappresentano numeri interi, mentre i nodi interni dell'albero sono stringhe rappresentanti le 4 operazioni aritmetiche: "+", "-", "*", "/". Si scriva una funzione **`int valuta(Node* p)`** che restituisce la valutazione dell'espressione rappresentata dall'albero. Ad esempio, la funzione valuta, applicata all'albero a sinistra deve restituire il valore (7-5)+(3*2) = 8. Per convertire una stringa in un intero si usi la funzione **`atoi`**. Si calcoli la complessità della funzione **`valuta`** in funzione del numero n dei nodi dell'albero.

```
int valuta(Node* t) {
  if (!t->left && !t->right)
    // foglia
    return atoi(t->label.c_str());
  if (t->label == "+")
    return valuta(t->left)+
      valuta(t->right);
  if (t->label == "-")
    return valuta(t->left)-
      valuta(t->right);
  if (t->label == "*")
    return valuta(t->left)*
      valuta(t->right);
  if (t->label == "/")
    return valuta(t->left)/
      valuta(t->right);
}
```

complessità(n) = O(n)