

Algoritmi e Strutture Dati

7/1/13

Esercizio 1

1. Se si prova a dimostrare che $A(n) = O(n^2 \log n)$ per sostituzione, si ottiene:

$$\begin{aligned} A(n) &= 4A(n/2) + n^2 \log n \\ &\leq 4c \frac{n^2}{2^2} \log n/2 + n^2 \log n \\ &= cn^2 \log n - cn^2 + n^2 \log n \\ &\leq (c+1)n^2 \log n \\ &\leq cn^2 \log n \end{aligned}$$

L'ultima disequazione è il nostro obiettivo, ma non può essere dimostrata perchè si ottiene $c+1 \leq c$, che è falso. Poichè è il termine di ordine superiore a rendere falsa la disequazione, non possiamo usare il trucco di sottrarre un termine di ordine inferiore.

Proviamo quindi un limite più alto, come $A(n) = O(n^3)$; si ottiene

$$\begin{aligned} A(n) &= 4A(n/2) + n^2 \log n \\ &\leq 4c \frac{n^3}{2^3} + n^2 \log n \\ &= \frac{c}{2} n^3 + n^2 \log n \\ &\leq \frac{c}{2} n^3 + n^3 \quad \forall n \geq 2 \\ &= \frac{c+2}{2} n^3 \leq cn^3 \end{aligned}$$

L'ultima disequazione è vera per $c \geq 2$.

Esiste una versione del master theorem (che non abbiamo visto) che dice che $A(n) = \Theta(n^2 \log^2 n)$; tuttavia un limite lasco come quello sopra sarebbe stato sufficiente.

2. Utilizzando il Master Theorem, si ottiene $B(n) = \Theta(n^2 \log n)$.
3. E' facile dimostrare, tramite metodo di sostituzione, che $C(n) = n!$. Mostriamo qui solo la parte relativa al limite superiore; il limite inferiore è simile.
- Caso base ($n = 1$): $C(1) = 1 \leq c \cdot 1$, che è vero per $c \geq 1$.
 - Ipotesi induttiva: $C(n') \leq cn'!$, per ogni $n' < n$.
 - Passo induttivo: $C(n) = nC(n-1) \leq nc(n-1)! \leq cn!$; l'ultima disequazione è vera per ogni c .

Esercizio 2

boolean isComplete(TREE T)

if $T.left = \text{nil}$ **and** $T.right = \text{nil}$ **then**
 return true

if $T.left = \text{nil}$ **or** $T.right = \text{nil}$ **then**
 return false

return isComplete($T.left$) **and** isComplete($T.right$)

Esercizio 3

- E' possibile dimostrare che un cammino monotono è aciclico per assurdo; infatti, se così non fosse, avremmo che il primo elemento del ciclo è sia minore dell'ultimo (per transitività) che maggiore (perchè l'ultimo si trova subito del primo).

- Per quanto riguarda l'algoritmo, è possibile utilizzare una banale variazione dell'algoritmo di visita in profondità come segue.

```
boolean monotono(GRAPH  $G$ , NODE  $s$ , NODE  $d$ )
```

```
  foreach  $v \in G.V$  do
     $v.mark \leftarrow \text{false}$ 
  return monotono-ric( $G, s, d$ )
```

```
boolean monotono-ric(GRAPH  $G$ , NODE  $s$ , NODE  $d$ )
```

```
   $s.mark \leftarrow \text{true}$ 
  if  $s = d$  then
    print  $d$ 
    return true
  else
    foreach  $v \in s.adj()$  do
      if  $u.mark = \text{false}$  and  $w(s) < w(v)$  then
        if monotono-ric( $G, v, d$ ) then
          print  $s$ 
          return true
    return false
```

- La complessità è quella di una visita in profondità, $O(m + n)$.

Esercizio 4

La soluzione si trova nel libro, sezione 19.1. Si risolve tramite programmazione dinamica.