Algoritmi e Strutture Dati
9 febbraio 2022

Note

1. La leggibilità è un prerequisito: parti difficili da leggere potranno essere ignorate.
2. Quando si presenta un algoritmo è fondamentale spiegare l'idea e motivarne la correttezza.
3. L'efficienza e l'aderenza alla traccia sono criteri di valutazione delle soluzioni proposte.
4. Si consegna la scansione dei fogli di bella copia, e come ultima pagina un documento di identità.

Domande

Domanda A (8 punti) Si consideri la seguente funzione ricorsiva con argomento un intero $n \geq 0$

```
val(n)
   if n <= 2
      return 1
   else
      return val(n-1) + val(n-2) + val(n-2)
```

Determinare la ricorrenza che esprime la complessità della funzione e mostrare che la soluzione è $\Omega(2^n)$. La complessità è anche $O(2^n)$? Motivare le risposte.

Soluzione: Nel caso ricorsivo, si effettuano tre chiamate ricorsive, una con argomento $n-1$ e due con argomento $n-2$, e inoltre due somme (operazioni di tempo costante) si ottiene quindi

$$T(n) = T(n-1) + 2\,T(n-2) + c$$

Dimostriamo con il metodo di sostituzione che $T(n) = \Omega(2^n)$, ovvero dimostriamo che esistono $d > 0$ e n_0 tali che $T(n) \geq d \cdot 2^n$, per $n \geq n_0$. Si procede per induzione su n:

$$\begin{aligned}
T(n) &= T(n-1) + 2\,T(n-2) + c && \text{[dalla definizione della ricorrenza]}\\
&\geq d\,2^{n-1} + 2\,d\,2^{n-2} + c && \text{[ipotesi induttiva]}\\
&\geq d\,2^{n-1} + 2\,d\,2^{n-2} && \text{[poiché } c > 0\text{]}\\
&= d(2^{n-1} + 2 \cdot 2^{n-2})\\
&= d\,2^n
\end{aligned}$$

2^(n-1) + 2^(n-2+1)
=2^(n-1) + 2^(n-1)
=2 * 2^(n-1) =
=2^(n-1+1)=2^n --> d2^n

qualunque siano d e n_0.

Vale anche $T(n) = O(2^n)$, ovvero esistono $d > 0$ e n_0 tali che $T(n) \leq d\,2^n$ per un'opportuna costante $d > 0$ e $n \geq n_0$. Ai fini della prova induttiva è opportuno dimostrare che $T(n) \leq d(2^n - 1)$. Infatti:

$$\begin{aligned}
T(n) &= T(n-1) + 2\,T(n-2) + c && \text{[dalla definizione della ricorrenza]}\\
&\leq d(2^{n-1} - 1) + 2\,d(2^{n-2} - 1) + c && \text{[ipotesi induttiva]}\\
&= d(2^{n-1} + 2 \cdot 2^{n-2}) - 3\,d + c\\
&= d\,2^n - 3\,d + c\\
&\leq d\,2^n
\end{aligned}$$

dove l'ultima disuguaglianza vale se $-3d + c \leq 0$, e quindi per $d \geq c/3$.

Domanda B (6 punti) Si consideri un insieme di 6 attività $a_i, 1 \leq i \leq 6$, caratterizzate dai seguenti vettori $\mathbf{s}$ e $\mathbf{f}$ di tempi di inizio e fine:

$$\mathbf{s} = (2, 3, 5, 2, 1, 9) \qquad\qquad \mathbf{f} = (4, 7, 8, 9, 10, 11).$$

Determinare l'insieme di massima cardinalità di attività mutuamente compatibili selezionato dall'algoritmo greedy GREEDY_SEL visto in classe. Motivare il risultato ottenuto descrivendo brevemente l'algoritmo.

Soluzione: $\{a_1, a_3, a_6\}$

Esercizi

Esercizio 1 (10 punti) Un array di interi `A[1..n]` si dice *triangolare* se esiste $q \in [1, n]$ tale che `A[1..q]` è ordinato in modo crescente e `A[q..n]` è ordinato in modo decrescente. Realizzare una funzione `maxTr(A)` che dato un array triangolare `A[1..n]`, senza elementi ripetuti, ne trova il massimo. Valutarne la complessità.

Soluzione: Si realizza una procedura divide et impera che opera sul sotto-array $A[p, r]$, che inizialmente sarà l'intero array. Se $p = r$, ovvero il sotto-array ha dimensione 1, semplicemente si ritorna l'unico elemento $A[p]$. Altrimenti, si considera il punto intermedio $q = \lfloor p + r \rfloor$. Se $A[q] < A[q + 1]$ significa che la parte `A[p..q]` è crescente e la parte `A[q+1..r]` è ancora triangolare, per cui il massimo si troverà in quest'ultima e può essere cercato ricorsivamente. Dualmente se $A[q] > A[q + 1]$ significa che la parte `A[q+1..r]` è decrescente e la parte `A[p..q]` è ancora triangolare, per cui il massimo si troverà in quest'ultima.

```
maxTr(A,p,r)                                12348675
   if p=r
      return A[p]
   else
      q = (p+r)/2
      if A[q]<A[q+1]
         return maxTr(A,q+1,r)
      else
         return maxTr(A,p,q)


maxTr(A)
   return maxTr(A,1,A.length)
```

La complessità è espressa dalla ricorrenza

$$T(n) = T(n/2) + c$$

Si può risolvere utilizzando il master theorem, confrontando $n^{\log_b^a} = n^{\log_2^2} = n$ con $f(n) = c$. Dato che $f(n) = O(n^{\log_b^a - \epsilon}) = O(n^{1-\epsilon})$ per $0 < \epsilon < 1$ si conclude che siamo nel primo caso del teorema che ci da' la soluzione $T(n) = \Theta(\log n)$.

Esercizio 2 (8 punti) Per $n > 0$, siano dati due vettori a componenti intere $\mathbf{a}, \mathbf{b} \in \mathbf{Z}^n$. Si consideri la quantità $c(i,j)$, con $0 \le i \le j \le n-1$, definita come segue:

$$c(i,j) = \begin{cases} a_i & \text{se } 0 < i \le n-1 \text{ e } j = n-1, \\ b_j & \text{se } i = 0 \text{ e } 0 \le j \le n-1, \\ c(i-1,j-1) \cdot c(i,j+1) & 0 < i \le j < n-1. \end{cases}$$

Si vuole calcolare la quantità $m = \min\{c(i,j) : 0 \le i \le j \le n-1\}$.

1. Si fornisca il codice di un algoritmo iterativo bottom-up per il calcolo di m.
2. Si valuti la complessità esatta dell'algoritmo, associando costo unitario ai prodotti tra numeri interi e costo nullo a tutte le altre operazioni.

Soluzione:

(a) Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione in cui calcoliamo gli elementi in ordine crescente di indice di riga e, per ogni riga, in ordine decrescente di indice di colonna. Il codice è il seguente.

```
COMPUTE(a,b)
n <- length(a)
m = +infinito
for i=1 to n-1 do
   C[i,n-1] <- a_i
   m <- MIN(m,C[i,n-1])
for j=0 to n-1 do
   C[0,j] <- b_j
   m <- MIN(m,C[0,j])
for i=1 to n-2 do
   for j=n-2 downto i do
      C[i,j] <- C[i-1,j-1] * C[i,j+1]
      m <- MIN(m,C[i,j])
return m
```

(b)

$$T(n) = \sum_{i=1}^{n-2} \sum_{j=i}^{n-2} 1 = \sum_{i=1}^{n-2} n - 1 - i = \sum_{k=1}^{n-2} k = (n-1)(n-2)/2.$$