Algoritmi e Strutture Dati
18 Giugno 2021

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 = 0
      return 0
   else
      return val(n-1) + n + 1
```

Dimostrare induttivamente che la funzione calcola il valore $(n+3)n/2$. Inoltre, determinare la ricorrenza che esprime la complessità della funzione e mostrare che la soluzione è $\Theta(n)$. Motivare le risposte.

Soluzione: Per quanto riguarda la funzione calcolata, procediamo per induzione su n. Se $n = 0$, la funzione ritorna 0 che è in effetti $(0+3)0/2 = 0/2 = 0$. Se $n > 0$, allora per ipotesi induttiva, $val(n-1) = (n+2)(n-1)/2$. Quindi $val(n) = val(n-1)+n+1 = (n+2)(n-1)/2+n+1 = (n^2+n-2+2n+2)/2 = (n^2+3n)/2 = (n+3)n/2$, come desiderato.

Per quanto riguarda la ricorrenza, nel caso ricorsivo, si effettuano una chiamata ricorsiva con argomento $n-1$ e una somma (operazione di tempo costante) si ottiene quindi

$$T(n) = T(n-1) + c$$

È facile dimostrare con il metodo di sostituzione che $T(n) = \Theta(n)$. Dimostriamo, ad esempio, che $T(n) = \Omega(n)$, ovvero che esistono $d > 0$ e n_0 tali che $T(n) \geq dn$, per $n \geq n_0$. Si procede per induzione su n:

$$\begin{aligned} T(n) &= T(n-1) + c && \text{[dalla definizione della ricorrenza]} \\ &\geq d(n-1) + c && \text{[ipotesi induttiva]} \\ &= dn - d + c \\ &\geq dn \end{aligned}$$

per una costante d tale che $0 < d \leq c$ e qualunque n.

Vale anche $T(n) = O(n)$, ovvero esistono $d > 0$ e n_0 tali che $T(n) \leq dn$ per un'opportuna costante $d > 0$ e $n \geq n_0$. Infatti:

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

dove l'ultima disuguaglianza vale per una costante $d \geq c$ e qualunque sia n.

Domanda B (6 punti) Si consideri un insieme di 8 attività $a_i, 1 \leq i \leq 8$, caratterizzate dai seguenti vettori **s** e **f** di tempi di inizio e fine:

$$\mathbf{s} = (1, 1, 2, 4, 2, 5, 6, 9) \qquad \mathbf{f} = (3, 4, 5, 6, 7, 9, 10, 12).$$

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: Si considerano le attività ordinate per tempo di fine, e ad ogni passo si sceglie l'attività che termina prima, rimuovendo quelle incompatibili. Si ottiene così l'insieme di attività $\{a_1, a_4, a_7\}$.

Esercizi

Esercizio 1 (9 punti) Scrivere una funzione `Anc(T,k1,k2)` che dato un albero binario di ricerca `T` nel quale tutte le chiavi sono distinte, e due chiavi `k1`, `k2` presenti in `T`, verifica se il nodo contenente `k1` è antenato del nodo contenente `k2`. Valutare la complessità della funzione.

Soluzione: È sufficiente osservare che, dato che le chiavi sono uniche e `k1`, `k2` sono certamente contenute in `T`, il nodo che contiene `k1` è antenato di quello che contiene `k2` se e solo se cercando `k2` a partire dalla radice di `T` incontro la chiave `k1`. Si assume che un nodo sia antenato di sé stesso.

La funzione può dunque essere realizzata come una semplice variante della ricerca negli alberi binari di ricerca.

```
Anc(T, k1, k2)
   x = T.root

   while (x.key <> k1) and (x.key <> k2)
        if (k2 < x.key)
           x = x.left
        else
           x = x.right

   return (x.key == k1)
```

Se l'albero ha altezza h, nel caso peggiore non trovo `k1` e la chiave `k2` è una foglia a profondità h, che quindi raggiungo in h iterazioni. Pertanto la complessità è $O(h)$.

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 $0 \leq i \leq j \leq n - 1$, definita come segue:

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

Si vuole calcolare la quantità $M = \max\{c(i, j) : 0 \leq i \leq j \leq n - 1\}$.

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:

1. 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 <- MAX(M,C[i,n-1])
for j=0 to n-1 do
   C[0,j] <- b_j
   M <- MAX(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] * C[i,j+1]
      M <- MAX(M,C[i,j])
return M
```

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