Algoritmi e Strutture Dati
10 Luglio 2020

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) Definire formalmente la classe $\Theta(g(n))$. Dimostrare le seguenti affermazioni o fornire un controesempio:

i. se $f(n), f'(n) \in \Theta(g(n))$ allora $f(n) + f'(n) \in \Theta(g(n))$;

ii. $f(n), f'(n) \in \Theta(g(n))$ allora $f(n) * f'(n) \in \Theta(g(n))$;

Soluzione: Per la definizione di $\Theta(f(n))$, consultare il libro.

Per (i), siano $f(n), f'(n) \in \Theta(g(n))$. Per definizione, esistono $c_1, c_2 > 0$ e n_0 tali che per ogni $n \geq n_0$ vale:

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

e, analogamente esistono $c'_1, c'_2 > 0$ e n'_0 tali che per ogni $n \geq n'_0$ vale:

$$0 \leq c'_1 g(n) \leq f'(n) \leq c'_2 g(n)$$

Quindi, per ogni $n \geq \max\{n_0, n'_0\}$ abbiamo:

$$(c_1 + c'_1)g(n) = c_1 g(n) + c'_1 g(n \leq f(n) + f'(n) \leq c_2 g(n) + c'_2 g(n) = (c_2 + c'_2)g(n)$$

dato che $c_1 + c'_1, c_2 + c'_1 > 0$ questo conclude la prova che $f(n) + f'(n) \in \Theta(g(n))$.

Per la parte (ii), l'affermazione è falsa. Basta considerare $f(n) = f'(n) = n \in \Theta(n)$, ma ovviamente $f(n) * f'(n) = n^2 \notin \Theta(n)$.

Domanda B (6 punti) Si consideri un insieme di 7 attività $a_i, 1 \leq i \leq 7$, caratterizzate dai seguenti vettori $\mathbf{s}$ e $\mathbf{f}$ di tempi di inizio e fine:

$$\mathbf{s} = (1, 4, 2, 3, 7, 8, 11) \qquad \mathbf{f} = (3, 6, 9, 10, 11, 12, 13).$$

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 $\{a_1, a_2, a_5, a_7\}$.

Esercizi

Esercizio 1 (9 punti) Realizzare una funzione booleana `checkSum(A,B,C,n)` che dati tre array `A[1..n]`, `B[1..n]` e `C[1..n]` con valori numerici, verifica se esistono tre indici i, j, k con $1 \leq i, j, k \leq n$ tali che $A[i] + B[j] = C[k]$. Valutarne la complessità.

Soluzione: L'idea consiste nell'ordinare i vettori B e C e a questo punto verificare per ogni elemento $A[i]$ del primo array se ne esiste uno del secondo $B[j]$ la cui somma produce un elemento $C[k]$ del terzo. Il fatto che gli array B e C siano ordinati permette di scorrerli una sola volta per ogni elemento del primo array.

```
checkSum(A,B,C,n):
    # ordina B e C
    Sort(B)
    Sort(C)

    i = 1
    found = false

    while (i <= n) and not found
        # per ogni A[i] verifica se per qualche elemento B[j] vale che
        # A[i]+B[j]=C[k] scorrendo B e C

        j = k = 1
        while (j <= n) and (k <= n) and (A[i] + B[j] <> C[k])
            if (A[i] + B[j] < C[k]):
               j++
            else
               k++
        if (j<=n and k<=n)
           found = true
        else
           i++

    return found
```

L'invariante del ciclo esterno è che se `found` è falsa, allora per ogni $i' < i$, e qualunque siano j' e k', vale $A[i'] + B[j'] \neq C[k']$. Se invece `found` è vera, allora $i, j, n \leq n$ e $A[i] + A[j] = A[k]$.

Per il ciclo interno, abbiamo che per ogni $j' < j$ e per ogni $k' < k$ vale $A[i] + B[j'] \neq C[k']$. Per il mantenimento dell'invariante, unico fatto da osservare è che quando $A[i]+B[j] < C[k]$ posso incrementare j mantenendo l'invariante. Infatti so che per ogni $j' < j$, dato che B è ordinato, vale $B[j'] \leq B[j]$ e quindi $A[i] + B[j'] \leq A[i] + B[j] < C[k]$. Il ragionamento quando $A[i] + B[j] > C[k]$ e incremento k è duale.

La complessità è $O(n^2)$ dato che ho due cicli annidati. In quello esterno, quando non esco i aumenta, quindi il numero di iterazioni è limitato da n. In quello interno, quando non esco, j oppure k aumentano, quindi il numero di iterazioni è limitato da $2n$. Complessivamente, nel caso peggiore, ho $n * 2n = 2n^2 = \Theta(n^2)$ iterazioni, ciascuna di costo costante.

A questo si somma il costo degli ordinamenti, che però posso realizzare in tempo $\Theta(n \log n)$ di modo che non incida sulla complessità asintotica.

Esercizio 2 (9 punti) Data una stringa di numeri interi $A = (a_1, a_2, \ldots, a_n)$, si consideri la seguente ricorrenza $c(i,j)$ definita per ogni coppia di valori (i,j) con $1 \le i, j \le n$:

$$c(i,j) = \begin{cases} a_j & \text{if } i = 1, 1 \le j \le n, \\ a_{n+1-i} & \text{if } j = n, 1 < i \le n, \\ c(i-1,j) \cdot c(i,j+1) \cdot c(i-1,j+1) & \text{altrimenti.} \end{cases}$$

1. Si fornisca il codice di un algoritmo iterativo bottom-up COMPUTE_C(A) che, data in input la stringa A restituisca in uscita il valore $c(n,1)$.
2. Si valuti il numero esatto $T_{CC}(n)$ di moltiplicazioni tra interi eseguite dall'algoritmo sviluppato al punto (1).

Soluzione:

1. Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione "reverse column-major", in cui calcoliamo gli elementi della tabella in ordine decrescente di indice di colonna e, all'interno della stessa colonna, in ordine crescente di indice di riga. Il codice è il seguente.

   ```
   COMPUTE_C(A)
   n = length(A)
   for i=1 to n do
      c[1,i] = a_i
      c[i,n] = a_{n+1-i}
   for j=n-1 downto 1 do
      for i=2 to n do
         c[i,j] = c[i-1,j] * c[i,j+1] * c[i-1,j+1]
   return c[n,1]
   ```

   Si osservi che un altro modo corretto di riempire la tabella è attraverso una scansione "reverse diagonal", che scansiona per diagonali parallele alla diagonale principale partendo da quella contenente solo $c[1,n]$.

2. Ogni iterazione del doppio ciclo dell'algoritmo esegue due operazioni tra interi, e quindi

$$\begin{aligned} T_{CC}(n) &= \sum_{j=1}^{n-1} \sum_{i=2}^{n} 2 \\ &= \sum_{j=1}^{n-1} 2(n-1) \\ &= 2(n-1)^2. \end{aligned}$$

   Equivalentemente, basta osservare che l'algoritmo esegue due moltiplicazioni per ogni elemento di una tabella $(n-1) \times (n-1)$.