Algoritmi per l'Ingegneria – Ingegneria dell'Informazione

Compito Tipo (Durata: 2h30m)

Nome, Cognome, Matricola: ______________________________

Prima Parte: domande a risposta unica

1. Sia $\Pi = \mathcal{I} \times \mathcal{S}$ il problema della risoluzione di equazioni di primo grado $ax + b = 0$, con a e b coefficienti reali **positivi** (ovvero, $a, b \in \boldsymbol{R}^+$). Si definiscano gli insiemi $\mathcal{I}$ e $\mathcal{S}$.

   $\boxed{\mathcal{I} = \boldsymbol{R}^+ \times \boldsymbol{R}^+,\ \mathcal{S} = \boldsymbol{R}^-}$

   (**Solution strategy:**) The generic instance is a pair of positive reals (a, b), which are the coefficients of the equation to be solved. The solution to such equation is $x = -b/a < 0$. Therefore, $\mathcal{I} = \boldsymbol{R}^+ \times \boldsymbol{R}^+$, and $\mathcal{S} = \boldsymbol{R}^-$.

2. Si determini l'ordine di grandezza stretto (Θ) della seguente ricorrenza utilizzando il Master Theorem: $T(n) = 4T(n/16) + \sqrt{n}$.

   $\boxed{T(n) = \Theta(\sqrt{n} \log n)}$

   (**Solution strategy:**) The threshold function is $n^{\log_{16} 4} = \sqrt{n}$, which is of the same order of the work function. We are thus in the second case of the Master Theorem, whence $T(n) = \Theta(\sqrt{n} \log n)$.

3. Sia $\boldsymbol{y} = (8, 0, 0, 0, 16, 0, 0, 0)$ e sia $\boldsymbol{x} = DFT_8^{-1}(\boldsymbol{y})$. Allora

   $\boxed{\boldsymbol{x} = (3, -1, 3, -1, 3, -1, 3, -1)}$

   (**Solution strategy:**) We know that $DFT_8^{-1}(\boldsymbol{y}) = (1/8)\,(DFT_8(\boldsymbol{y}))^{\mathbf{rev}}$. Note that $\boldsymbol{y}$ is an $(8, 4)$-sparse vector, hence its transform is made by four repetitions of

   $$DFT_2((y_0, y_4)) = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 8 \\ 16 \end{pmatrix} = (24, -8).$$

   The results follows by dividing the two components by 8 and replicating the vector four times. Note that no reversing is necessary since, in this case $DFT_8(\boldsymbol{y}) = (DFT_8(\boldsymbol{y}))^{\mathbf{rev}}$.

4. Sia $\boldsymbol{p} = (1, 10, 5, 4)$ un vettore di dimensioni. Il minimo numero di prodotti tra scalari necessario a calcolare il prodotto di tre matrici A_i di dimensioni $p_{i-1} \times p_i$, $1 \leq i \leq 3$ è:

$$\boxed{m(1,3) = 70}$$

(**Solution strategy:**) With three matrices A_1, A_2, A_3, we have only two parenthesizations: $((A_1 \times A_2) \times A_3)$ and $(A_1 \times (A_2 \times A_3))$. The first has a cost of $1 \times 10 \times 5 + 1 \times 5 \times 4 = 70$. The second has a cost of $10 \times 5 \times 4 + 1 \times 10 \times 4 = 240$.

5. Si calcoli il numero esatto $T(n)$ di esecuzioni del comando $a \leftarrow a + 1$ nel seguente frammento di codice:

   for $i \leftarrow 1$ **to** $n - 1$ **do**
       for $j \leftarrow i$ **to** $i + 1$ **do**
           for $k \leftarrow j$ **to** $2j - 1$ **do**
               $a \leftarrow a + 1$

$$\boxed{T(n) = n^2 - 1}$$

(**Solution strategy:**) We have

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n-1} \sum_{j=i}^{i+1} \sum_{k=j}^{2j-1} 1 \\
&= \sum_{i=1}^{n-1} \sum_{j=i}^{i+1} j \\
&= \sum_{i=1}^{n-1} (i + (i+1)) \\
&= \sum_{i=1}^{n-1} (2i + 1) \\
&= 2(n-1)n/2 + (n-1) \\
&= (n-1)(n+1) \\
&= n^2 - 1
\end{aligned}$$

Seconda Parte: risoluzione di problemi

Esercizio 1 ([11 Punti]) Sia n una potenza di due. Per un dato problema computazionale Π, si supponga di avere un algoritmo iterativo A_1 la cui complessità su istanze di taglia n sia $T_1(n) = n^2$, e un algoritmo ricorsivo A_2 la cui complessità sia regolata dalla seguente ricorrenza:

$$T(n) = \begin{cases} 1 & n = 1, \\ 2T(n/2) + 7n & n > 1. \end{cases}$$

1. ([7 Punti]) Analizzando l'albero delle chiamate, si determini la complessità $T_{n_0}(n)$ del generico algoritmo ibrido A_H che utilizza la strategia ricorsiva di A_2 per taglie dell'istanza $n > n_0$ e invoca A_1 per taglie $n \leq n_0$, dove n_0 è una potenza di due.

2. ([4 Punti]) Si determini il valore $\bar{n}_0$ che fornisce il miglior algoritmo ibrido per $n > \bar{n}_0$. (*Attenzione:* si ricordi che $\frac{d}{dx} \log_2 x = \frac{\log_2 e}{x}$)

Answer: Point 1 The recurrence associated to the generic hybrid algorithm is:

$$T_{n_0}(n) = \begin{cases} n^2 & n \leq n_0, \\ 2T_{n_0}(n/2) + 7n & n > n_0 \end{cases}$$

For $n \geq n_0$, the recursion tree associated to the above recurrence has $\log(n/n_0)$ levels. On level i, $0 \leq i < \log(n/n_0) - 1$, there are 2^i nodes, each referring to an instance of size $n/2^i$, hence contributing work $7(n/2^i)$, while on level $\log(n/n_0)$ there are $2^{\log(n/n_0)} = n/n_0$ nodes each contributing work ${n_0}^2$. The total work is then

$$\sum_{i=0}^{\log(n/n_0)-1} 2^i \cdot 7\left(\frac{n}{2^i}\right) + \left(\frac{n}{n_0}\right){n_0}^2 = 7n\log(n/n_0) + nn_0.$$

Therefore, the running time of the generic hybrid algorithm is

$$T_{n_0}(n) = \begin{cases} n_0^2 & n \leq n_0, \\ 7n\log(n/n_0) + nn_0 & n > n_0. \end{cases}$$

Point 2 Let us now take the partial derivative of $T_{n_0}(n)$ with respect to n_0:

$$\frac{\partial T_{n_0}(n)}{\partial n_0} = -\frac{7n\log e}{n_0} + n = n\left(1 - \frac{7\log e}{n_0}\right)$$

which is nonnegative when

$$1 - \frac{7 \log e}{n_0} \geq 0 \Leftrightarrow n_0 \geq 7 \log e.$$

Therefore, the optimal choice $\bar{n}_0$ is one of the two powers of two surrounding $7 \log e$, i.e., either 8 or 16. By comparing the analytic expressions of $T_{\bar{n}_0}(n)$ for $n > \bar{n}_0$ and $\bar{n}_0 = 8, 16$, we discover that the best running time is obtained for $\bar{n}_0 = 8$. $\square$

Esercizio 2 [12 Punti] Per $n > 1$, dato un array bidimensionale $A[1..n, 1..n]$ di interi e una coppia di indici $[s,t]$ con $1 \leq s, t \leq n$, si definisce $[s,t]$-*cammino in* A una sequenza $\Pi_{[s,t]} = \langle A[i_0, j_0], A[i_1, j_1], \ldots, A[i_m, j_m] \rangle$ con $1 = i_0 \leq i_1 \leq \ldots \leq i_m = s$, $1 = j_0 \leq j_1 \leq \ldots \leq j_m = t$ e tale che per $0 \leq k < m$, $(i_{k+1} - i_k) + (j_{k+1} - j_k) = 1$. In parole, un $[s,t]$-cammino è una sequenza di elementi dell'array a partire da $[1,1]$ sino a $[s,t]$, con il vincolo che elementi successivi nel cammino si trovano o sulla stessa riga e in colonne successive oppure sulla stessa colonna e in righe successive. Il *costo* di $\Pi_{[s,t]}$ è dato da $\sum_{k=0}^{m} A[i_k, j_k]$. Dato in ingresso un array $A[1..n, 1..n]$ di interi positivi, si consideri ora il problema di determinare il costo minimo di un $[n,n]$-cammino in A.

1. ([7 Punti]) Si enunci e si dimostri una proprietà di sottostruttura ottima per il problema e si derivi una ricorrenza dalla proprietà enunciata.

2. ([5 Punti]) Si fornisca lo pseudocodice del risultante algoritmo **iterativo** di programmazione dinamica. Per avere punteggio pieno, l'algoritmo deve eseguire $O(n^2)$ operazioni aritmetiche tra interi.

Answer: Point 1 Our subproblem space is finding optimal $[s,t]$-paths, for all $1 \leq s, t \leq n$. The optimal substructure property is the following. Let $\Pi^\star_{[s,t]}$ be an optimal $[s,t]$-path. Then

$$\Pi^\star_{[s,t]} = \begin{cases} \langle A[1,1], A[1,2], \ldots, A[1,t] \rangle & s = 1, 1 \leq t \leq n, \\ \langle A[1,1], A[2,1], \ldots, A[s,1] \rangle & t = 1, 1 < s \leq n, \\ \text{either} \langle \Pi^\star_{[s-1,t]}, A[s,t] \rangle \text{ or } \langle \Pi^\star_{[s,t-1]}, A[s,t] \rangle & \text{otherwise.} \end{cases}$$

The first two cases are trivial, since if either s or t is equal to 1, there is a single $[s,t]$-path in A. For the third case, note that $\Pi^\star_{[s,t]}$ must contain either $[s-1,t]$ or $[s,t-1]$. Consider the first case, that is $\Pi^\star_{[s,t]} = \langle A[i_0, j_0], \ldots, A[i_{m-1}, j_{m-1}], A[s,t] \rangle$, with $1 = i_0 \leq i_1 \leq \ldots \leq i_{m-1} = s-1$, $1 = j_0 \leq j_1 \leq \ldots \leq j_{m-1} = t$. Then, we claim that $\langle A[i_0, j_0], \ldots, A[i_{m-1}, j_{m-1}] \rangle$ is an optimal $[s-1,t]$-path, since otherwise we could obtain an $[s,t]$-path of cost smaller than the one of $\Pi^\star_{[s,t]}$, a contradiction. The second case can be proved in the same fashion.

Let $c(s,t)$ be the cost of an optimal $[s,t]$-path, for $1 \leq s,t \leq n$. The above substructure property immediately yields the following recurrence:

$$c(s,t) = \begin{cases} \sum_{j=1}^{t} A[1,j] & s = 1, 1 \leq t \leq n, \\ \sum_{i=1}^{s} A[i,1] & t = 1, 1 < s \leq n, \\ \min\{c(s-1,t), c(s,t-1)\} + A[s,t] & \text{otherwise.} \end{cases}$$

Clearly, we are interested in computing $c(n,n)$.

Point 2 We compute $c(s,t)$ in a bottom up fashion according to the above recurrence, storing the computed values in a bidimensional array $C[1..n,1..n]$. Observe that the first row and column of C can be computed directly in $O(n)$ time. As for the other entries, in order to compute $c[s,t]$ we need the values $C[s-1,t]$ and $C[s,t-1]$, therefore a simple row-major scan suffices. Let $\text{MIN}(x,y)$ be a simple routine returning $\min\{x,y\}$. The algorithm is the following.

```
COMPUTE_C(A)
n ← rows(A)
C[1,1] ← s_r ← s_c ← A[1,1]
for i ← 2 to n do
    C[1,i] ← s_r ← s_r + A[1,i]
    C[i,1] ← s_c ← s_c + A[i,1]
for s ← 2 to n do
    for t ← 2 to n do
        C[s,t] ← MIN(C[s-1,t], C[s,t-1]) + A[s,t]
return C[n,n]
```

The correctness of the above algorithm follows from the correctness of the substructure property and the fact that each entry of array C is computed prior to its use. The running time of the algorithm is dominated by the time required by the two nested loops, with each iteration requiring a single addition, for a total of $O(n^2)$ additions overall. □