Algoritmi per l'Ingegneria

Compito, 31/1/2019 (Durata: 2h30m)

Nome, Cognome: ____________________

Prima Parte: domande a risposta unica (due punti a domanda)

1. Sia $f(n) = n - 1$. Si determini il valore $f^\star(n, n_0)$ per $n = 55$ e $n_0 = 48$. $\boxed{f^\star(55, 48) = 6}$

   (**Solution strategy:**) For $f(n) = n - 1$ and $i \geq 0$, we have $f^{(i)}(n) = n - i$. Recalling that for $n > n_0$ $f^\star(n, n_0) = \max\{i \geq 0 : f^{(i)}(n) > n_0\}$, we solve the inequality $n - i > n_0$, which yields $i < n - n_0$. Thus, the largest value of i satisfying the inequality is $n - n_0 - 1$. It follows that $f^\star(55, 48) = 55 - 48 - 1 = 6$.

2. Utilizzando il Master Theorem, si determini l'ordine di grandezza della seguente ricorrenza:

   $$T(n) = 16T\left(\frac{n}{64}\right) + n^{2/3}. \qquad \boxed{T(n) = \Theta\left(n^{2/3}\log n\right)}$$

   (**Solution strategy:**) The threshold function is $n^{\log_{64}(16)} = n^{(\log_4 16/\log_4 64)} = n^{2/3}$. Since the threshold function and the work function are of the same order, we are in the second case of the Master Theorem, thus $T(n) = \Theta\left(n^{2/3}\log n\right)$.

3. Sia $\boldsymbol{x} = (7, 2, 2, 2, 7, 2, 2, 2)$ e sia $\boldsymbol{y} = F_8(\boldsymbol{x})$. Allora:

   $$\boxed{\boldsymbol{y} = (26, 0, 10, 0, 10, 0, 10, 0)}$$

   (**Solution strategy:**) Observe that $(7, 2, 2, 2, 7, 2, 2, 2) = (5, 0, 0, 0, 5, 0, 0, 0) + (2, 2, 2, 2, 2, 2, 2, 2)$ and recall that DFT_8 is a linear transformation. The first summand is $(8, 4)$-sparse, and the second is a vector with identical components. Thus, the first summand's transform is the four-fold concatenation of $DFT_2(5, 5) = (10, 0)$, while the second summand's transform is the vector with all null components but the first, equal to $2 \cdot 8 = 16$. We obtain $\boldsymbol{y} = (10, 0, 10, 0, 10, 0, 10, 0) + (16, 0, 0, 0, 0, 0, 0, 0) = (26, 0, 10, 0, 10, 0, 10, 0)$. Alternatively, we can simulate the computation performed by the FFT algorithm. The first level of recursion requires computing $DFT_4(7, 2, 7, 2)$ and $DFT_4(2, 2, 2, 2)$. The latter is a special transform, yielding $DFT_4(2, 2, 2, 2) = (8, 0, 0, 0)$ immediately. For the former, from $DFT_2(7, 7) = (14, 0)$ and $DFT_2(2, 2) = (4, 0)$ we obtain $DFT_4(7, 2, 7, 2) = (18, 0, 10, 0)$. The result follows by applying the conquer phase of the first level of recursion.

4. Si consideri il seguente frammento di pseudocodice:

   $a \leftarrow 0$
   for $i \leftarrow 1$ **to** $n - 1$ **do**
       for $j \leftarrow i$ **to** $i + 1$ **do**
           for $k \leftarrow j + 2$ **to** $2j + 1$ **do**
               $a \leftarrow a + 2$
   return a

   Si calcoli il valore di a restituito per $n = 21$. $\boxed{a\,\big|_{n=21} = 880}$

(**Solution strategy:**) For a generic value of n, the returned value $a(n)$ is

$$\begin{aligned}
a(n) &= \sum_{i=1}^{n-1}\sum_{j=i}^{i+1}\sum_{k=j+2}^{2j+1} 2 \\
&= \sum_{i=1}^{n-1}\sum_{j=i}^{i+1} 2j \\
&= \sum_{i=1}^{n-1}(2i+2(i+1)) \\
&= \sum_{i=1}^{n-1}(4i+2) \\
&= 2(n-1)n+2(n-1) \\
&= 2(n-1)(n+1)
\end{aligned}$$

Therefore $a(21) = 2 \cdot 20 \cdot 22 = 880$

5. Dato un file sull'alfabeto $\Sigma = \{\text{'A'}, \text{'B'}, \text{'C'}, \text{'D'}, \text{'E'}, \text{'F'}\}$ con frequenze 'A':9%, 'B':11%, 'C':15%, 'D':25%, 'E':40%, si determinino le 5 codeword ottenute con la codifica di Huffman vista in classe. Nella costruzione del codice, si associ sempre il bit 0 sempre al sottoalbero di frequenza cumulativa minore.

$$\boxed{e(\text{'A'}) = 1110, e(\text{'B'}) = 1111, e(\text{'C'}) = 110, e(\text{'D'}) = 10, e(\text{'E'}) = 0}$$

(**Solution strategy:**) The answer follows from executing the algorithm for the given frequencies.

Seconda Parte: risoluzione di problemi

Esercizio 1 [12 punti]

Punto 1 [7 punti] Sia $n \geq 2$ una potenza di due. Si progetti e si scriva lo pseudocodice di un algoritmo divide-and-conquer TWO_LARGEST(A) per calcolare la coppia ordinata in senso decrescente dei due elementi più grandi di una stringa $A = \langle a_1, a_2, \ldots, a_n \rangle$ di n interi positivi distinti. Ad esempio, se $A = \langle 3, 1, 7, 5, 4, 9, 6, 8 \rangle$, l'algoritmo deve restituire la coppia $(9, 8)$. (**Attenzione**: per essere non banale, l'algoritmo progettato deve eseguire **meno** di $(n-1)+(n-2) = 2n-3$ confronti, complessità banalmente ottenibile calcolando prima il massimo di A per poi eliminarlo dalla stringa e calcolare il massimo della stringa residua.)

Punto 2 [5 punti] Si scriva e si risolva esattamente la ricorrenza relativa al numero di confronti tra gli elementi della stringa effettuati dall'algoritmo. sviluppato al Punto 1.

Answer:

Point 1. For the base case $n = 2$, let $A = \langle a_1, a_2 \rangle$. Then TWO_LARGEST($A$) must return $(\max\{a_1, a_2\}, \min\{a_1, a_2\})$. For $n > 2$, divide $A = \langle a_1, a_2, \ldots, a_n \rangle$ into the two substrings $A_1 = \langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $A_2 = \langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$, and let $(M_1, m_1), (M_2, m_2)$ be the pairs of the two largest elements of A_1 and A_2, respectively, returned by the two recursive calls TWO_LARGEST(A_1) and TWO_LARGEST(A_2). It is immediate to see that if $M_1 > M_2$, then the two largest elements of A will be $(M_1, \max\{m_1, M_2\})$. Otherwise, if $M_1 < M_2$, then the two largest elements of A will be $(M_2, \max\{M_1, m_2\}$. The algorithm follows.

TWO_LARGEST(A)
  $n \leftarrow A.len$
  if $(n = 2)$
    then if $(a_1 > a_2)$
        then return (a_1, a_2)
        else return (a_2, a_1)
  $A_1 \leftarrow \langle a_1, a_2, \ldots, a_{n/2} \rangle$
  $A_2 \leftarrow \langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$
  $(M_1, m_1) \leftarrow$ TWO_LARGEST(A_1)
  $(M_2, m_2) \leftarrow$ TWO_LARGEST(A_2)
  if $(M_1 > M_2)$
    then if $(m_1 > M_2)$
        then return (M_1, m_1)
        else return (M_1, M_2)
    else if $(M_1 > m_2)$
        then return (M_2, M_1)
        else return (M_2, m_2)

Point 2. In the above algorithm, the base case performs a single comparison, the divide step does not perform any comparison, and the conquer step performs two comparisons. Thus, the number of comparisons $T(n)$ obeys to the following recurrence:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + 2, & \text{if } n > 2, \\ 1, & \text{if } n = 2. \end{cases}$$

This is a simple recurrence that can by solved by unfolding as follows:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + 2 \\ &= 2^2 T\left(\frac{n}{2^2}\right) + 2^2 + 2 \\ &\ldots \\ &= 2^k T\left(\frac{n}{2^k}\right) + \sum_{i=1}^{k} 2^i, \ \ 1 \le k \le \log_2 n - 1. \end{aligned}$$

For $k = \log_2 n - 1$ we have that $2^k = n/2$ and $\sum_{i=1}^{\log_2 n - 1} 2^i = 2^{\log_2 n} - 2 = n - 2$. Thus

$$\begin{aligned} T(n) &= \frac{n}{2} T(2) + n - 2 \\ &= \frac{n}{2} + n - 2 \\ &= \frac{3n}{2} - 2. \end{aligned}$$

In fact, a smaller number of comparisons may be obtained with the alternative algorithm sketched in the following. First, we perform the classical divide-and-conquer algorithm seen in class for computing $\max\{A\}$. Observe that algorithm can be thought of as a tournament where each internal node corresponds to a match between the two maximum values M_1 and M_2 associated to its children subinstances. Clearly, the match is won by $\max\{M_1, M_2\}$. When the tournament ends at the root, the global winner is clearly $\max\{A\}$. Once the tournament is over, we can track down all matches that involved $\max\{A\}$. Clearly, the second largest value must be among the opponents of these matches. Since $\max\{A\}$ was involved in exactly $\log_2 n$ matches, we can obtain the second largest element with $\log_2 n - 1$ comparisons, for a total of $n + \log_2 n - 2$ comparisons altogether. It can be shown that this is the least number of comparisons needed to determine the two largest elements of a string. However, note that this is not a divide-and-conquer algorithm and that additional information has to be stored in order to track down the matches involving $\max\{A\}$. □

Esercizio 2 [11 punti] Per $n \geq 0$, si consideri la seguente definizione per ricorrenza della quantità intera $c(i,j)$, con $0 \leq i,j \leq n$:

$$c(i,j) = \begin{cases} 2 \cdot (i+j) & (i=0) \vee (j=0), \\ 2 \cdot (c(i-1,j-1) + c(i-1,j) - c(i,j-1)) & \text{altrimenti.} \end{cases}$$

Punto 1 [7 punti] Si fornisca lo pseudocodice di un algoritmo **memoizzato** per il calcolo di $c(n,n)$. Si presti attenzione alla scelta del valore di default da utilizzare nell'inizalizzazione.

Punto 2 [4 punti] Si determini, analizzando l'albero delle chiamate, il numero **esatto** di operazioni aritmetiche complessive effettuate dalle due routine che costituiscono il codice memoizzato.

Answer:

Point 1. The code is the following:

```
INIT_c(n)
if (n = 0) then return 0
c[0,0] ← 0
for i ← 1 to n do
    c[i,0] ← c[0,i] ← 2 · i
for i ← 1 to n do
    for j ← 1 to n do
        c[i,j] ← 1
return REC_c(n,n)
```

```
REC_c(i,j)
if (c[i,j] = 1) then
  c[i,j] ←2 · (REC_c(i − 1,j − 1)+
              REC_c(i − 1,j) − REC_c(i,j − 1))
return c[i,j]
```

The correctness of the above code follows from the fact that it implements the given recurrence and that the data structure is correctly initialized, since the default value 1 cannot be obtained by the recurrence (whose values are always even).

Point 2 First observe that INIT_c(n) executes n products to initialize the first row and column of table c. Next, the recursion tree of REC_c(n,n) has exactly one internal node for each nonbase case, that is, for each pair of indices i,j, with $1 \leq i,j \leq n$, therefore, there are exactly n^2 internal nodes. For each internal node, we perform one sum, one subtraction and one product. In summary, letting $T_c(n)$ denote the complexity of the memoized code, we obtain: $T_c(n) = 3n^2 + n$.

□