**Dati e Algoritmi 2 – CdL Magistrale in Ingegneria Informatica**

**Compito, 23/1/2019 (Durata: 3h)**

**Nome, Cognome, Matricola:** _____

**Corso di studio:** _____

# Prima Parte: domande teoriche

Si forniscano risposte il più possibile **rigorose e succinte** ai seguenti quesiti riguardanti argomenti trattati nel corso. Risposte ingiustificate, approssimative, prolisse o in cattivo italiano saranno **fortemente penalizzate**. Ai fini del superamento dell'esame, è necessario conseguire una valutazione sufficiente alla prima parte, che richiede che almeno due esercizi su tre siano svolti in modo sufficiente.

**T1** Si specifichi formalmente il problema del **calcolo delle radici cubiche di un numero complesso diverso da zero** come una relazione $\Pi \subseteq \mathcal{I} \times \mathcal{S}$. In particolare, si specifichino gli insiemi $\mathcal{I}$, $\mathcal{S}$ e la relazione $\Pi$ tra i loro elementi.

**Solution:** Recall that any nonzero complex number has three distinct nonzero square roots. Hence we can set

$$\begin{aligned}\mathcal{I} &= \boldsymbol{C} - \{0\} \\ \mathcal{S} &= \{(w_1, w_2, w_3) \in \boldsymbol{C} \times \boldsymbol{C} \times \boldsymbol{C} : ((w_i \neq w_j) \wedge (w_i^3 = w_j^3), 1 \leq i \neq j \leq 3\}\end{aligned}$$

Then we have that:

$$\forall z \in \mathcal{I}, \forall s = (w_1, w_2, w_3) \in \mathcal{S} : z \Pi (w_1, w_2, w_3) \Leftrightarrow (w_1^3 = z).$$

**T2** Data la ricorrenza $T(n) = 16T(n/64) + \sqrt[3]{n^2}$, si discuta se esiste un caso di applicabilità del Master Theorem e, se sì, si indichi quale e si fornisca l'ordine di grandezza della soluzione.

**Solution:** For $a = 16$, $b = 64$, we have that $\log_b a = \log_{64} 16 = \log_4 16 / \log_4 64 = 2/3$, hence $w(n) = \sqrt[3]{n^2} = n^{2/3} = n^{\log_b a}$ is of the same order of the threshold function. We are therefore in the second case of the Master Theorem, whence $T(n) = \Theta(w(n) \log n) = \Theta(n^{2/3} \log n)$.

**T3** Si provi che la riducibilità $<_P$ è una relazione transitiva tra linguaggi.

**Solution:** We have to prove that

$$\forall L_1, L_2, L_3 \subseteq \{0,1\}^\star : (L_1 <_P L_2) \wedge (L_2 <_P L_3) \Rightarrow L_1 <_P L_3$$

Since $(L_1 <_P L_2)$, there exists $f : \{0,1\}^\star \to \{0,1\}^\star$ polynomial-time computable such that $\forall x \in \{0,1\}^\star : (x \in L_1) \Leftrightarrow (f(x) \in L_2)$. Analogously, since $(L_2 <_P L_3)$, there exists $g : \{0,1\}^\star \to \{0,1\}^\star$ polynomial-time computable such that $\forall y \in \{0,1\}^\star : (y \in L_2) \Leftrightarrow (g(y) \in L_3)$. Let $A_f(x)$ and $A_g(y)$ be the algorithms that compute $f(x)$ and $g(y)$ in time $T_{A_f}(|x|) = O(|x|^{k_f})$ and $T_{A_g}(|y|) = O(|y|^{k_g})$, respectively, and observe that function $h(x) = (g \circ f)(x)$ is computable by Algorithm $A_g(A_f(x))$ in time $T_h(x) = O(|x|^{\max\{k_f, k_f k_g\}})$, which is still polynomial. Moreover:

$$x \in L_1 \Leftrightarrow y = f(x) \in L_2 \Leftrightarrow g(y) = g(f(x)) = (g \circ f)(x) = h(x) \in L_3,$$

whence $L_1 <_P L_3$.

# Seconda Parte: risoluzione di problemi

Si forniscano soluzioni esaurienti e rigorose ai tre problemi seguenti. Gli algoritmi vanno codificati utilizzando lo **pseudocodice** usato in classe. **Attenzione:** Risposte immotivate e prive di prova di correttezza non saranno considerate.

**Esercizio 1 [10 punti]** Si consideri la seguente ricorrenza, definita sul parametro intero $n > 0$:

$$T(n) = \begin{cases} 47 & n \leq 16, \\ T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + T\left(\lfloor \sqrt{n} \rfloor\right) + 4n & n > 16. \end{cases}$$

Utilizzando l'induzione parametrica, si determini un'adeguata costante $c > 0$ per cui $T(n) \leq cn$ per ogni $n > 0$. (em Suggerimento: Può tornare utile la maggiorazione $\sqrt{n} < n/4$ che vale per tutti i valori di $n > 16$.

**Answer:** Let us collect constraints on $c$ by "simulating" an inductive argument. For the base case, we have that for $0 < n \leq 16$, $T(n) = 47 \leq 47n \overset{?}{\leq} cn \Leftrightarrow c \geq 47$. Assume now that $T(k) \leq ck$ for $0 < k < n$, with $n > 16$, and observe that for $n > 16$ we have that $\sqrt{n} < n/4$. For $k = n$ we have:

$$\begin{aligned} T(n) &\leq c\lfloor 2n/3 \rfloor + c\left\lfloor \sqrt{n} \right\rfloor + 4n \\ &\leq 2cn/3 + c\sqrt{n} + 4n \\ &\leq 2cn/3 + cn/4 + 4n \\ &= (11/12)cn + 4n \overset{?}{\leq} cn \\ &\Leftrightarrow 4n \overset{?}{\leq} cn/12 \\ &\Leftrightarrow c \geq 48 \end{aligned}$$

It follows that by choosing $c = \max\{47, 48\} = 48$ we obtain $T(n) \leq 48n$. □

**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 e si eviti di utilizzare un numero negativo.

**Punto 2 [4 punti]** Si determini, analizzando l'albero delle chiamate, il numero **esatto** di operazioni aritmetiche tra interi effettuate dall'algoritmo.

**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: $T_c(n) = 3n^2 + n$.

□

**Esercizio 3 [11 punti]** Si ricorda che il problema decisionale HAMILTON ha come istanza la codifica di un grafo non orientato $\langle G = (V, E)\rangle$, per cui si vuole determinare l'esistenza di un ciclo che tocca tutti i vertici una volta sola. Si supponga di avere un algoritmo $A_H(\langle G\rangle)$ che decida HAMILTON in tempo lineare. **Si fornisca lo pseudocodice e si analizzi correttezza e complessità** di un algoritmo FIND_CYCLE($\langle G\rangle$) che utilizza $A_H$ come subroutine e restituisce la stringa **"no cycle"** se $\langle G = (V, E)\rangle$ è un'istanza negativa di HAMILTON e l'insieme degli archi che formano un ciclo hamiltoniano altrimenti.

**Answer:** On input $\langle G = (V, E)\rangle$, Algorithm FIND_CYCLE($\langle G\rangle$) first checks whether $G$ contains a hamiltonian cycle with a call to $A_H(\langle G\rangle)$. If the instance is negative, FIND_CYCLE outputs the string **"no cycle"** and returns. Otherwise,

it enters a loop where, for each edge in $e \in E$, it checks whether the graph remains hamiltonian when $e$ is removed, and, if so, it removes $e$ from $E$. At the end of the loop, FIND_CYCLE returns the residual edge set. The code follows.

FIND_CYCLE($\langle G = (V, E)\rangle$)
**if** $(A_H(\langle G\rangle) = 0)$
  **then return "no cycle"**
$\star$ Let $E = \{e_1, e_2, \ldots, e_m\}\star$
$C \leftarrow E$
**for** $i \leftarrow 1$ **to** $m$ **do**
  **if** $(A_H(\langle (V, C - \{e_i\})\rangle) = 1)$
    **then** $C \leftarrow C - \{e_i\}$
**return** $C$

Let us anayze the correctness of FIND_CYCLE. Clearly, if $\langle G = (V, E)\rangle$ is a negative instance, the algorithm correctly returns the string **"no cycle"**, as prescribed. Otherwise, we can show by induction that for $0 \leq i \leq |E| = m$, at the end of Iteration $i$, instance $\langle (V, C)\rangle$ is a positive instance (conventionally, the end of Iteration 0 is considered to be the instant before entering the **for** loop). This is clearly true for $i = 0$ since we reach the loop only if $\langle G\rangle$ is a positive instance. Moreover, assuming that $(C, E)$ is hamiltonian at the end of Iteration $i-1$, $i \geq 1$, during Iteration $i$ we remove edge $e_i$ only if the instance $(C, E - \{e_i\})$ stays positive and otherwise we keep the previous instance, which is positive by the inductive hypothesis.

Now, it is easy to argue that after the last iteration, set $C$ only contains $|V|$ edges which form a hamiltonian cycle. Indeed, if this were not the case, then $C$ should contain a subset of edges $C'$ that form a hamiltonian cycle (by the property proven above) and at least an edge $e_k \notin C'$. Observe that at the beginning of iteration $k$, $C'$ is clearly still contained in $C$, hence the subsequent call $(A_H(\langle G_C = (V, C - \{e_k\})\rangle)$ would return 1, and $e_k$ would be removed, a contradiction.

The running time of FIND_CYCLE is dominated, in the worst case, by the $m = O(|\langle G\rangle|)$ calls to $A_H$, each requiring time $O(|\langle G\rangle|)$ time, therefore the algorithm has complexity at most quadratic in the instance size:

$$T_{\text{FIND\_CYCLE}}(|\langle G\rangle|) = O(|\langle G\rangle|^2).$$

□