Fondamenti dell'Informatica

A.A. 2005/2006

Corso A

Seconda prova di esonero: 8/6/2006 ore 8.30 – 11.00

1. Dare la definizione di insieme ricorsivamente enumerabile. Dato l'insieme delle funzioni ricorsive monotone crescenti M={x | φ_x è monotona crescente} stabilire se esso è un insieme ricorsivamente enumerabile, se lo l'insieme complemento $\overline{M}$ e a quale classe della gerarchia aritmetica (o gerarchia di Kleene) i due insiemi appartengono *(9 punti)*.

Soluzione

> **Definizione 25** *Un insieme $A \subseteq \mathbb{N}$ è detto* ricorsivamente enumerabile *(r.e.) se $A = \emptyset$ o se esiste una funzione ricorsiva totale $f : \mathbb{N} \longmapsto \mathbb{N}$ tale che $A = imm(f)$. In tal caso diciamo che la funzione f* enumera *l'insieme A.*

Similarmente a quanto dimostrato per l'insieme T = M={x | φ_x è totale} si può dimostrare che M non è ricorsivamente enumerabile. Per assurdo supponiamo che lo sia. Allora esiste una funzione *f* ricorsiva totale che lo enumera. Possiamo quindi definire la seguente funzione:

$$h(x) = \sum_{y \le x} \varphi_{f(y)}(y) + 1$$

che è ovviamente monotona crescente (al crescere di x non facciamo altro che aggiungere altri numeri positivi alla somma). Sia $\bar{y}$ un suo indice. Poiché *h* è monotona crescente allora $h \in M$, e quindi ci sarà un $\bar{x}$ tale che $f(\bar{x}) = \bar{y}$. Ora osserviamo che:

$$h(\bar{x}) = \sum_{y \le \bar{x}} \varphi_{f(y)}(y) + 1 = \sum_{y < \bar{x}} \varphi_{f(y)}(y) + \varphi_{f(\bar{x})}(\bar{x}) + 1 \quad \text{per definizione costruttiva di } h$$

$$h(\bar{x}) = \varphi_{\bar{y}}(\bar{x}) \quad \text{per definizione di indice della funzione } h.$$

Ne consegue l'assurdo che:

$$\sum_{y < \bar{x}} \varphi_{f(y)}(y) + \varphi_{f(\bar{x})}(\bar{x}) + 1 = \varphi_{f(\bar{x})}(\bar{x})$$

Ora, se M non è ricorsivamente enumerabile, cioè non appartiene a Σ_1, a quale classe della gerarchia di Kleene appartiene? Ancora una volta, basta rifarsi a quanto detto per l'insieme *T* e riadattarlo a M. Infatti:

M={x | $\forall y\ \exists k$ [$\varphi_x(y)$ richiede meno di *k* passi $\wedge$ $\varphi_x(y+1)$ richiede meno di *k* passi $\wedge$ $\varphi_x(y) \le \varphi_x(y+1)$]}

Il predicato riportato nella condizione esprime la proprietà di computabilità in meno di *k* passi sia per $\varphi_x(y)$ che per $\varphi_x(y+1)$ e che inoltre sia soddisfatta la proprietà di monotonia. Questo predicato è ricorsivo (decidibile), quindi per definizione di classe Π_2 concludiamo che $M \in \Pi_2$.

Negando la condizione, si vede facilmente che $\overline{M}$ appartiene a Σ_2. Infatti possiamo scrivere che:

> $\overline{M}$ ={x | $\exists y\ \forall k$ [$\varphi_x(y)$ richiede più di *k* passi $\vee$ $\varphi_x(y+1)$ richiede più di *k* passi $\vee$ $\varphi_x(y) > \varphi_x(y+1)$]}

dove ancora una volta il predicato nella condizione è ricorsivo (decidibile).

2. Enunciare e dimostrare, ricorrendo alla definizione di enumerazione delle funzioni ricorsive, il teorema della ricorsione (o teorema di Kleene) (*4 punti*). Se t è il seguente programma che trasforma un programma Pascal-like *q* in una altro programma Pascal-like *q'*:

   q'(x) := t(q) (x) := if x = 0 then 0 else x+q(x-1)

   indicare un possibile punto fisso e darne dimostrazione. Il programma *q(x)=x* è un punto fisso per t? (*4 punti*)

Soluzione della seconda parte.

Ogni programma che calcola la somma di Gauss, cioè che calcola:

$$gauss(n) = \sum_{i=0}^{n} i$$

è un punto fisso per *t*. Infatti, se q è un programma che calcola la somma di Gauss, ad esempio il seguente programma Pascal-like:

q(x) := x*(x+1)/2,

nel caso in cui x=0 avremo che q'(x) restituirà 0, cioè gauss(0), cioè proprio quello che calcola q(0), mentre nel caso di x >0 q'(x) restituirà x + q(x-1)=x+gauss(x-1)=gauss(x), cioè proprio quello che calcola q(x).

Al contrario il programma

q(x)=x

non è un punto fisso. Infatti in tal caso q'(x) calcolerà la seguente funzione:

f(0) = 0

f(x) = x+(x-1) se x>0

ed è palese che la funzione f(x) non coincide con la funzione identità calcolata dal programma.

3. Spiegare la tecnica di programmazione del divide-et-impera (*2 punti*) e analizzare in generale la complessità di soluzioni sviluppate con questa tecnica (*4 punti*). Applicare le considerazioni generali riportate in precedenza a un algoritmo sviluppato con la tecnica del divide-et-impera (*3 punti*)

Soluzione della terza parte.

È sufficiente riportare l'algoritmo della ricerca binaria (o del quicksort o del mergesort), spiegare perché rispecchia lo schema tipico definito dalla tecnica di programmazione divide-et-impera, e infine mostrare qual è la relazione di ricorrenza che esprime la complessità dell'algoritmo scelto.

4. Definire le classi *P* e *LOGSPACE*. Spiegare e motivare la relazione esistente fra le due classi. Definire la Karp-riducibilità logspace fra problemi e spiegare come essa sia sfruttabile per definire la P-completezza di un problema. Riportare un esempio di problema P-completo. *(7 punti)*

ATTENZIONE: nella correzione dei compiti, relativamente a questo esercizio sono stati riscontrati frequenti errori relativi alla definizione della classe LOGSPACE e alla modalità di dimostrazione che un problema è P-completo.

Per quanto riguarda LOGSPACE, essa è definita così: LOGSPACE=DTIME(log n)

Mentre per quanto riguarda la dimostrazione della P-completezza di un problema P, si osserva che è sufficiente cercare un problema P-completo P_1 tale che **P_1** è logspace Karp-riducibile **a P** ($P_1 \leq_m^{logspace} P$) **e non il viceversa.**

5. Enunciare e spiegare la tesi della computazione sequenziale. Commentare alla luce di questa tesi il fatto che per il problema della soddisfacibilità (SAT) esistono algoritmi di complessità esponenziale con macchine di Turing deterministiche e algoritmi di complessità polinomiale con macchine di Turing non deterministiche (*4 punti*). [1]

[1] La totalizzazione di un punteggio superiore a 30 punti equivale al *30 con lode*.