Informatica Teorica

Seconda prova in itinere - 28 Giugno 2010

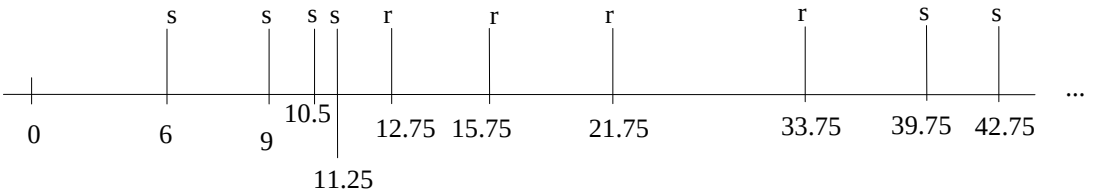
Tempo a disposizione: 2h

NB: il punteggio è espresso in 17-esimi e riflette il peso relativo della prova rispetto all'intero esame il cui punteggio è in 30-esimi. Come già in altre occasioni è tuttavia possibile ottenere un punteggio complessivo superiore a 17/17.

Esercizio 1 (6/17 punti)

Specificare mediante formule di logica del prim'ordine il comportamento di sequenze di segnali *s* e *r* descritte nel seguito.
Le sequenze iniziano con una serie di segnali *s* in cui la distanza tra un segnale *s* ed il successivo si dimezza, fino a che non diventa minore di 1; a quel punto inizia una sequenza di segnali *r*, nella quale la distanza tra due segnali *r* consecutivi si raddoppia, fino a che non diventa maggiore di 10; in seguito c'è una serie di segnali *s* simile alla precedente, a così via infinitamente al futuro.

La figura seguente mostra una esempio di sequenza di segnali.



Si noti che nella transizione da una serie di segnali *s* ad una serie di segnali *r* la distanza tra il primo *r* e l'ultimo *s* è due volte la distanza tra i due ultimi segnali *s*; in modo analogo, nella transizione da una serie di segnali *r* ad una serie di segnali *s* la distanza tra il primo *s* e l'ultimo *r* è la metà della distanza tra i due ultimi segnali *r*.
Si noti inoltre che i segnali *s* ed *r* sono mutuamente esclusivi, nel senso che durante una serie di emissioni di *s*, il segnale *r* non è emesso, e vice versa.

Per descrivere il comportamento di cui sopra si usino solo i predicati $s(t)$ e $r(t)$, in cui $s(t)$ (risp. $r(t)$) è vero esattamente in quegli istanti t in cui *s* (risp. *r*) è emesso.

Suggerimento: si scriva una formula logica per ognuno dei seguenti punti.

- Il segnale *s* è emesso la prima volta in un istante maggiore di 1. La seconda emissione del segnale *s* accade ad una distanza dalla prima che è la metà della distanza tra il primo segnale e 0.
- La distanza tra un'occorrenza del segnale *s* e il segnale precedente (sia esso *s* o *r*) è dimezzata ad ogni emissione, fino a che non diventa meno di 1. Quando la distanza tra un segnale *s* e il precedente è meno di 1, un segnale *r* è emesso ad una distanza dall'ultimo segnale che è due volte la distanza tra i due ultimi segnali *s*.
- (in modo analogo al punto precedente) La distanza tra un'occorrenza del segnale *r* e il segnale precedente (sia esso *r* o *s*) è raddoppiata ad ogni emissione, fino a che non diventa maggiore di 10. Quando la distanza tra un segnale *r* e il precedente è più di 10, un segnale *s* è emesso ad una distanza dall'ultimo segnale che è la metà della distanza tra i due ultimi segnali *r*.

(continua dietro)

Esercizio 2 (6/17 punti)

Si dica se le seguenti funzioni sono computabili, motivando adeguatamente la risposta:

1) $g'(x,y) = 0$ se $f_x(y) \neq \perp$ e $f_x(y) = 2\ f_y(x)$; $g'(x,y) = f_x(y)+f_y(x)$ altrimenti;
2) $g''(x) = 0$ se $\exists y : f_x(y) = f_y(x)$; $g''(x) = 1$ altrimenti.

NB: Si adotta la convenzione che, per una qualunque operazione aritmetica $\oplus$, il risultato dell'espressione $x \oplus y$ è $\perp$ se uno qualunque dei degli operandi è $\perp$.

Esercizio 3 (7/17 punti)

La seguente funzione C calcola il valore del polinomio `c[0]+c[1]·x+c[2]·x`^{2}`+...+c[n]·x`^{n} a partire da un array `c` contenente i coefficienti, dal grado `n` del polinomio, e dal valore della variabile `x`.

```
int evalPolyn (int c[], int n, int x) {
  int pow, val;
  int i, j;

  val = 0;
  for (i = 0; i <= n; i++){
    pow = 1.0;
    for (j=1; j<=i; j++)
        pow = pow * x;
    val = val + c[i]*pow;
  }
  return val;
}
```

1. Si valutino la complessità temporale T(n) e la complessità spaziale S(n) del programma, in funzione del grado del polinomio, usando il criterio di costo costante.
2. Si valutino la complessità temporale T(n, x) e la complessità spaziale S(n, x) in funzione del grado del polinomio e del valore della variabile x, usando il criterio di costo logaritmico. Non è necessario dare l'espressione esatta di T(n, x) e S(n, x), ma è sufficiente fornire una valutazione approssimata che tiene in considerazione i fattori dominanti.

Soluzioni

Esercizio 1

$\exists t\ (t>1 \wedge s(t) \wedge \forall t'\ (0 \le t' < (3/2)t \wedge t' \neq t \rightarrow \neg s(t')) \wedge s((3/2)t)\ \wedge \forall t''\ (0 \le t'' \le (3/2)t \rightarrow \neg r(t'')))$

$\forall t,t'\ (t'>t \wedge s(t') \wedge (s(t) \vee r(t)) \wedge \forall t''\ (t < t'' < t' \rightarrow \neg s(t'') \wedge \neg r(t'')) \rightarrow$
$((t' - t \ge 1) \rightarrow s(t'+(t'-t)/2) \wedge \neg r(t'+(t'-t)/2) \wedge \forall t''\ (t' < t'' < t'+(t'-t)/2 \rightarrow \neg s(t'') \wedge \neg r(t'')))$
$\wedge$
$((t' - t < 1) \rightarrow r(t'+(t'-t)*2) \wedge \neg s(t'+(t'-t)*2) \wedge \forall t''\ (t' < t'' < t'+(t'-t)*2 \rightarrow \neg s(t'') \wedge \neg r(t'')))$

$\forall t,t'\ (t'>t \wedge r(t') \wedge (s(t) \vee r(t)) \wedge \forall t''\ (t < t'' < t' \rightarrow \neg s(t'') \wedge \neg r(t'')) \rightarrow$
$((t' - t \le 10) \rightarrow r(t'+(t'-t)*2) \wedge \neg s(t'+(t'-t)*2) \wedge \forall t''\ (t' < t'' < t'+(t'-t)*2 \rightarrow \neg s(t'') \wedge \neg r(t'')))$
$\wedge$
$((t' - t > 10) \rightarrow s(t'+(t'-t)/2) \wedge \neg s(t'+(t'-t)/2) \wedge \forall t''\ (t' < t'' < t'+(t'-t)/2 \rightarrow \neg s(t'') \wedge \neg r(t'')))$

Esercizio 2

Le funzioni sono entrambe computabili.
Definiamo un algoritmo per computare g'. Utilizzando la usuale enumerazione *E*, otteniamo dapprima due MT M_x e M_y. Poi calcoliamo a := $f_x(y)$, e b := $f_y(x)$. Se a = 2b, viene restituito 0. altrimenti, a+b. Si noti che anche se $f_x(y)$, o $f_y(x)$, o entrambe ($f_x(y)$ e $f_y(x)$) sono indefinite, l'algoritmo rispetta la definizione di g'.
Per g'' è sufficiente notare che possiamo sempre prendere y = x. Quindi g'' è la funzione costante 0, che è ovviamente computabile.

Esercizio 3

1. Il fattore dominante della complessità deriva dall'istruzione `pow=pow*x;` nel ciclo interno. Tale istruzione viene eseguita un numero di volte pari a $\sum_{i=0}^{n}\sum_{j=1}^{i}1$, che è $\Theta(n^2)$. La complessità spaziale è invece dominata dalla dimensione dell'array c, ed è quindi $\Theta(n)$.
2. Con il criterio del costo logaritmico, dobbiamo considerare che l'esecuzione dell'istruzione `pow=pow*x;` corrisponde a eseguire le seguenti 3 istruzioni della macchina RAM:
   ```
   LOAD w;
   MULT y;
   STORE w;
   ```
   dove `w` e `y` sono gli indirizzi delle variabili `pow` e `x`, rispettivamente. Alla j-esima iterazione del ciclo interno, la complessità dell'istruzione `LOAD w` è $l(w)+l(x^{j-1}) \cong (j-1)l(x)$, la complessità dell'istruzione `MULT y` è $l(y)+l(x)+l(x^{j-1}) \cong j \cdot l(x)$, e quella dell'istruzione `STORE w` è $l(w)+l(x^j) \cong j \cdot l(x)$. Quindi la complessità di `pow=pow*x;` è $\cong (3j-1)l(x)$. La complessità dell'intero programma è quindi dominata dal fattore $\sum_{i=0}^{n}\sum_{j=1}^{i}(3j-1)l(x) \cong 3 \cdot l(x)\sum_{i=0}^{n}\sum_{j=1}^{i} j$. Poichè $\sum_{j=1}^{i} j$ è $\Theta(i^2)$ e $\sum_{i=0}^{n} i^2$ è $\Theta(n^3)$, abbiamo che $T(n,x) \cong K \cdot l(x) \cdot n^3$ per qualche valore costante K.
   Per il calcolo della complessità spaziale con il criterio del costo logaritmico dobbiamo invece tenere conto anche dell'occupazione delle celle di memoria. I fattori in gioco sono l'array dei coefficienti, le cui celle contengono valori costanti e quindi la complessità spaziale non cambia rispetto al criterio del costo costante e la cella che conterrà il valore massimo, cioè quella che contiene il risultato. In particolare `val` raggiungerà il valore $\sum_{i=0}^{n} Kx^i$, che è $\Theta(x^n)$ quindi l'occupazione della cella è $\Theta(\log(x^n))$, cioè è $\Theta(n\log(x))$. La complessità spaziale è quindi è $\Theta(n\log(x))$.