Algoritmi e Principi dell'Informatica

Appello del 2 Settembre 2014

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1, 2, 3 in 1 ora e 15 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 4, 5 e 6 in 1 ora e 15 minuti.

NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1 (punti 6)

Si consideri il linguaggio $L=\{ a^m b^n \mid m \neq n, \quad m, n > 0 \}$. Esempi di stringhe di L sono: *aabbb* e *aaabb*, mentre *ab*, *aabb*, *aba*, ε sono stringhe non appartenenti ad L.

1. Si fornisca una grammatica, preferibilmente a potenza generativa minima tra le classi conosciute, che generi L.
2. Si fornisca un automa, preferibilmente a potenza riconoscitiva minima tra le classi conosciute, che riconosca L.

Esercizio 2 (punti 6 senza la parte opzionale, 8, con la parte opzionale)

Una procedura *P* riceve in ingresso un numero naturale $n > 1$ e due array, *in* e *out*, entrambi contenenti $2 \cdot n$ numeri naturali (l'indice di entrambi gli array è compreso tra 1 e $2 \cdot n$ inclusi).

Preliminarmente si specifichino i predicati *odd*(*n*) e *even*(*n*), facendo uso di formule del prim'ordine contenenti esclusivamente variabili e costanti intere, il predicato di uguaglianza, i simboli delle operazioni di somma e prodotto, in modo tale che i due predicati assumano il loro naturale significato, ossia *odd*(*n*) risulti vero se e solo se *n* è un numero naturale dispari, e *even*(*n*) se e solo se *n* è un numero naturale pari.

Successivamente:

1) Si specifichino le precondizioni seguenti che devono quindi valere all'inizio dell'esecuzione di *P*.
   a) Ogni elemento dell'array *in* è diverso da ogni altro elemento dello stesso array *in*.
   b) Tutti gli elementi dell'array *in* in posizioni dispari sono dispari, e tutti quelli in posizione pari sono pari.

La procedura copia tutti gli elementi dell'array *in* nell'array *out*, piazzando i numeri dispari nelle prime *n* posizioni e quelli pari nelle *n* posizioni seguenti.

2) Si specifichino ora le seguenti postcondizioni per la procedura *P*.
   a) Tutti gli elementi della parte inferiore (ossia quelli nelle posizioni 1 .. *n*) dell'array *out* sono uguali a qualche elemento dispari di *in*.
   b) Tutti gli elementi della parte superiore dell'array *out* sono uguali a qualche elemento pari di *in*.
3) **Opzionalmente** si aggiungano ulteriori post-condizioni, se ritenute necessarie, per rendere la specifica della procedura perfettamente coerente con la sua definizione informale fornita qui sopra.
   NB: la parte opzionale verrà valutata solo se i due punti precedenti saranno risolti in modo soddisfacente.

Esercizio 3 (punti 3)

E' decidibile il problema di stabilire se una qualsiasi procedura Q soddisfa la specifica fornita dalle pre- e post-condizioni dell'esercizio precedente (incluse o escluse eventuali post-condizioni opzionali)? Si fornisca una breve ma precisa spiegazione della risposta.

Esercizio 4 (punti 8)

a) Descrivere a grandi linee ma con precisione una MT che realizzi la procedura dell'esercizio 2 in modo da ottenere la miglior complessità temporale (asintotica) possibile. (Si assuma che gli interi contenuti nell'array siano limitati, in modo che la complessità dipenda solo da *n*.)
b) Descrivere anche una macchina RAM, il cui repertorio istruzioni sia esattamente quello del testo, con lo stesso obiettivo.

   Quale delle due complessità temporali (la seconda a criterio logaritmico) è migliore? Perché?

Esercizio 5 (punti 3)

Si risolva la seguente equazione alle ricorrenze:
$T(n) = 5\ T(n/2) + n^2 \log(n)$

Esercizio 6 (punti 5)

Si consideri come struttura dati un albero generale, in cui cioè ogni nodo può avere un numero arbitrario di figli, anche maggiore di 2.

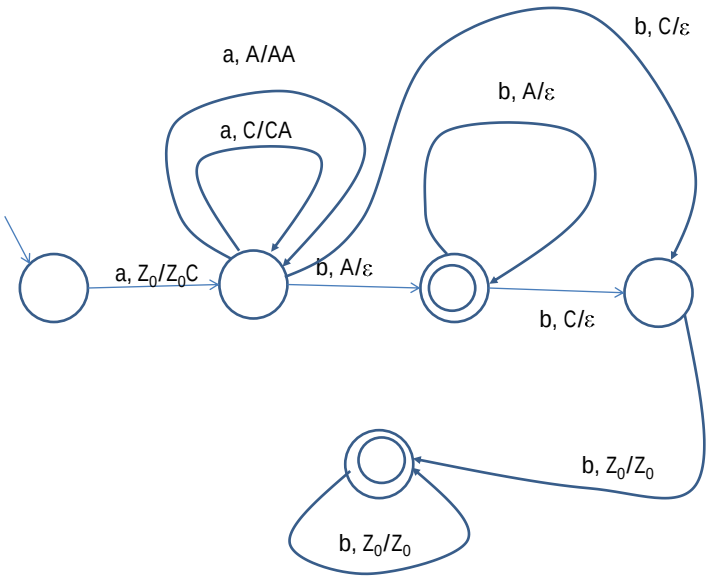
a) Dire come si può rappresentare un albero generale con una struttura a puntatori.
b) Si sviluppi un algoritmo per verificare, per ogni nodo *x* diverso dalla radice, se è soddisfatta la proprietà che la chiave del padre di *x* è > della chiave di *x* stesso. Si dia la complessità dell'algoritmo scritto.

Tracce delle soluzioni

Esercizio 1

1. S → aCb
   C → aCb | A | B
   A → aA | a
   B → bB | b

2.



Esercizio 2

$\forall x\ (\ odd(x) \leftrightarrow \exists k\ (\ x = 2{\cdot}k + 1\ )\ )$
$\forall x\ (\ even(x) \leftrightarrow \exists k\ (\ x = 2{\cdot}k\ )\ )$

1) Precondizioni.
   a) Ogni elemento dell'array *in* è diverso da ogni altro elemento dello stesso array *in*.
      $\forall i\ (1 \le i \le 2n \rightarrow \neg\exists j\ (1 \le j \le 2\text{n} \wedge i \ne j \wedge in[i] = in[j]\ )\ )$
   b) Tutti gli elementi dell'array *in* in posizioni dispari sono dispari, e tutti quelli in posizione pari sono pari.
      $\forall i\ (1 \le i \le 2n \rightarrow (\ (\ odd(i) \rightarrow odd(in[i])\ ) \wedge (even(i) \rightarrow even(in[i])\ )\ )$

2) Postcondizioni.
   a) Tutti gli elementi della parte inferiore (ossia quelli nelle posizioni 1 .. *n*) dell'array *out* sono uguali a qualche elemento dispari di *in*.
      $\forall i\ (1 \le i \le n \rightarrow \exists j\ (1 \le j \le 2n \wedge odd(in[j]) \wedge out[i] = in[j]\ )\ )$
   c) Tutti gli elementi della parte superiore dell'array *out* sono uguali a qualche elemento pari di *in*.
      $\forall i\ (n{+}1 \le i \le 2n \rightarrow \exists j\ (1 \le j \le 2n \wedge even(in[j]) \wedge out[i] = in[j]\ )\ )$

3) Parte opzionale.
   L'array *out* deve contenere tutti gli elementi dell'array *in*.
   $\forall i\ (1 \le i \le 2n \rightarrow \exists j\ (1 \le j \le 2n \wedge out[i] = in[j]\ )\ )$

Esercizio 3

La domanda dell'esercizio è un tipico problema di correttezza: si chiede se un generico algoritmo implementa una particolare funzione o famiglia di funzioni definita mediante le pre- e post-condizioni. In questo caso, evidentemente le funzioni che soddisfano le pre- e post-condizioni, sia includendo che escludendo la parte opzionale, non sono evidentemente né l'insieme vuoto (la specifica non è soddisfacibile) né l'insieme universo (la specifica è soddisfatta da qualsiasi funzione computabile). Quindi per il teorema di Rice il problema è indecidibile.

Esercizio 4

Essendo i numeri memorizzati nell'array limitati, ognuno di essi può essere memorizzato in un numero finito e costante di celle della MT.

In una prima passata la MT copia nel nastro di uscita solo i numeri in posizione dispari; successivamente si riporta all'inizio bel nastro di ingresso e copia a seguire i numeri in posizione pari. La complessità ottenuta è chiaramente O(n).

La RAM invece non ha la possibilità di percorrere a ritroso il nastro di ingresso, né quello di output; essa deve perciò memorizzare preliminarmente il contenuto del nastro di ingresso e procedere poi alla stessa maniera della MT; ogni memorizzazione di un numero però, a criterio di costo logaritmico, costa O(log(n)); quindi la complessità asintotica della RAM è O(n.log(n)).

Esercizio 5

La ricorrenza si risolve semplicemente applicando il Master Theorem. In questo caso $\log_b(a) = \log_2(5)$, che ha valore un po' maggiore di 2 (circa 2.3). $n^{2.3}$ è polinomialmente maggiore di $n^2\log(n)$, quindi la ricorrenza ha soluzione $\Theta(n^{\log_2(5)})$.

Esercizio 6

a) Un albero generale può essere rappresentato con una struttura a puntatori in cui ogni nodo *x* ha 4 attributi: *x.key* è la chiave, *x.p* è il puntatore al padre, *x.fst* è il puntatore al figlio più a sinistra, e *x.sibling* è il puntatore al fratello a destra.

b)

```
test_property(T)
x := T.root
return test(x)
```

```
test(x)
if x = NIL
  return true
if (x.p = NIL OR x.p.key > x.key)
  return test(x.fst) AND test(x.sibling)
else return false
```

La complessità dell'algoritmo scritto è semplicemente $\Theta(n)$, in quanto vengono analizzati tutti i nodi uno ad uno.