Informatica Teorica

Appello dell'8 Febbraio 2007

Il tempo a disposizione per lo svolgimento del tema d'esame è 1h e 30'.

Esercizio 1 (punti 10)

Sia dato il linguaggio L sull'alfabeto {a, b, c} contenenti tutte e sole le stringhe che hanno le seguenti caratteristiche:

- la stringa inizia e termina con *esattamente* lo stesso numero di 'a'
- la stringa contiene almeno un carattere diverso da 'a'

Per esempio, le seguenti stringhe appartengono a L:

aabcaa, aca, aacabbabaa, aaaacaaaa, ...

Le seguenti stringhe, invece, **non** appartengono ad L:

aaabaa, aaaa, aaacaccaaaaaa, ecc.

Si scriva un automa *oppure* una grammatica che riconosca o generi il linguaggio L.

NB1: il punteggio massimo verrà dato se il formalismo scelto è a potenza minima tra quelli che riconoscono/generano L.

NB2: si scelga *una sola* tra le due possibilità: si scriva o un automa, oppure una grammatica, ma *non* entrambi.

Esercizio 2 (11)

L'algoritmo di ordinamento per conteggio serve ad ordinare un array di valori interi fornito in ingresso. Esso per funzionare si basa sulla ipotesi fondamentale che i valori da ordinare siano degli interi appartenenti all'insieme [0..k], con k prefissato. L'algoritmo sfrutta per funzionare un array ausiliario C di k+1 elementi, che serve a contenere il conteggio effettuato dall'algoritmo: in pratica C[j] contiene il numero di elementi presenti nell'array in ingresso con valore pari a j.

Si implementi un algoritmo di ordinamento per conteggio, sapendo che ogni singolo dato da ordinare è sempre un intero non negativo memorizzabile in 8 bit. Quali sono le sue complessità temporale e spaziale valutate a criterio di costo costante e a criterio di costo logaritmico? Giustificare brevemente la risposta.

NB: Si consiglia per comodità di descrivere l'algoritmo in pseudocodice, o mediante un linguaggio di alto livello.

Esercizio 3 (punti 11)

Il professor Touring ha preparato, per il prossimo appello del suo esame di Informatica 1, i 3 seguenti esercizi:

1. Sia dato il seguente frammento di codice C:

```
1.  void DivisoreComune (int i1, int i2){
2.    int min, max;
3.
4.    if(i1 <= i2){
5.      min = i1;
6.      max = i2;
7.    } else {
8.      min = i2;
9.      max = i1;
10.   }
11.
12.   for(d = 1; d <= min; d++){
13.     if(min % d == 0 && max % d == 0)
14.       return d;
15.   }
16.
17.   return 0;
18. }
```

   Dire se ci sono errori di sintassi e, se ce ne sono, indicare quali.

2. Si scriva un sottoprogramma che prende in ingresso due parole `p1` e `p2` di al massimo 20 caratteri l'una, e ritorna `true` se sono una l'anagramma dell'altra, `false` altrimenti.

3. Siano P, Q, R, tre puntatori a interi e x, y due variabili intere. Si dica quanto valgono rispettivamente x, y, *P, *Q, *R dopo l'esecuzione della seguente sequenza di istruzioni.

```
x = 3; y = 5;
P = &x; Q = &y; R = P;
*R = 10; y = x + *Q; x = x + *P;
Q = R; P = Q
```

Per ognuno di questi esercizi dire, motivando adeguatamente la risposta, se è decidibile il problema di stabilire se la soluzione proposta da uno studente è corretta oppure no.

Soluzioni

Esercizio 1

La seguente grammatica noncontestuale genera il linguaggio L.

S → aSa | BXB | B

X → aX | bX | cX | ε

B → b | c

Esercizio 2

L'algoritmo puo' operare come segue. Si mantengono due puntatori a, c rispettivamente all'elemento corrente dell'array A da ordinare e all'elemento corrente dell'array C dei contatori.

1. Fase di input, che supponiamo venga terminata dalla lettura di -1, e di scrittura dei contatori:

```
for (c = 0 to 255)
   C[c] := 0
c := read();
a := 0;
while( c ≠ -1 ) {
   A[a] := c;
   C[c] := C[c]+1;
   c := read();
   a := a+1;
}
A[a] := -1;
```

2. Fase di scrittura dell'output, che scandisce C e sovrascrive A:

```
a := 0
for (c = 0 to 255) {
  while ( C[c] > 0 ) {
   A[a] := c;
   C[c] := C[c]-1;
   a := a+1; }}
```

Sia *n* il numero di elementi dell'array da ordinare.

- A **costo costante**, la *complessita' spaziale* e' $\Theta(n)$ in quanto memorizzo un array di dimensione n e uno di dimensione costante (infatti considero interi a 8 bit, quindi 256 possibili valori).
  La *complessita' temporale* e' pure $\Theta(n)$. Infatti il passo 1 consta di 1 ciclo eseguito $\Theta(n)$ volte. Nel passo 2, invece, il ciclo piu' interno e' eseguito un numero costante di volte, mentre quello esterno e' eseguito al piu' *n* volte. Quindi tutti i passi sono di complessita' temporale $\Theta(n)$, e lo stesso e' la loro composizione sequenziale.
- A **costo logaritmico**, il costo di memorizzazione dell'array da ordinare e' sempre $\Theta(n)$, in quanto in ogni cella ho un valore limitato da 256. L'array di contatori consta invece di 256 celle, ognuna delle quali costa $\Theta(\log(n))$. Quindi in tutto la *complessita' spaziale* e' ancora limitata superiormente da $\Theta(n)$.
  Per quanto riguarda la *complessita' temporale*, si manipolano dei contatori che sono limitati

superiormente da n. Dunque, in costo logaritmico, questo implica in fattore log(n) in ciascuna esecuzione dei cicli, che si traduce in una complessita' temporale complessiva di $\Theta(n \log(n))$.

Esercizio 3

1. La risposta si riduce a una lista, sicuramente finita visto che il frammento di programma e' finito, di potenziali errori di sintassi. Pertanto verificare la risposta corrisponde semplicemente a confrontare due liste di dimensioni finita, problema chiaramente decidibile.
2. Verificare la correttezza della risposta corrisponde a decidere se il programma/soluzione fornito dallo studente implementa la funzione richiesta oppure no. Come noto, decidere se un dato programma implementa una certa funzione e' un problema non decidibile. Si noti che il fatto che l'input del programma e' ristretto ad un numero finito di elementi (le stringhe di 20 caratteri) non cambia la risposta, dal momento che il formalismo usato per descrivere la soluzione (il codice) e' comunque Turing-completo.
3. In questo caso la risposta consiste semplicemente in 3 valori interi per *P, *Q, e *R. Controllare la validita' della risposta e' pertanto decidibile, in quanto si riduce al confrontro tra 3 coppie di interi.