Informatica Teorica, Esame del 1/7/2009
Durata per il tema intero: 2 ore.
Durata per sola II prova in itinere (esercizi 1-3): 1h 20'.

Esercizio 1 (6 punti)
Si consideri un insieme fissato S finito di numeri naturali.
1. L'insieme S' di tutti e soli i numeri che non sono multipli di elementi di S è ricorsivo? Si motivi adeguatamente la risposta.
2. L'insieme dei programmi che stampano tutti e soli i numeri appartenenti al complemento di S è decidibile?
3. L'insieme dei programmi che stampano tutti i numeri di S è semidecidibile?

Esercizio 2 (6 punti)
Si consideri un programma per la macchina RAM che, avendo un certo numero fissato K di valori interi positivi memorizzato in una determinata porzione di memoria (per esempio dalla cella 1000 alla 1000+K-1), legga un valore intero positivo e stampi 1 se il valore letto è multiplo di uno di quelli memorizzati, 0 altrimenti. Si tratteggi il programma in modo sufficientemente completo e preciso per svolgerne l'analisi di complessità temporale, sia col criterio di costo uniforme sia con quello logaritmico. Si preferiscono programmi che non abbiano complessità inutilmente alta.

Si consideri la variante del problema precedente, in cui il programma legge un generico numero N di valori interi positivi, poi legge un valore n intero positivo, e stampi 1 se il valore letto è multiplo di uno di quelli letti, 0 altrimenti. Come cambia in questo caso l'analisi di complessità?

Esercizio 3 (8 punti)
Si descriva con formule di logica del I ordine un garbage collector (GC) real-time, avente il seguente funzionamento:
- GC viene chiamato almeno ogni 5 unità di tempo durante ogni intervallo in cui lo heap sia occupato in modo continuato per meno del 50%;
- GC viene chiamato almeno ogni 3 unità di tempo durante ogni intervallo in cui lo heap sia occupato in modo continuato per il 50% o più;
- GC, una volta chiamato, impiega 1 unità di tempo per liberare un blocco di memoria da 1K, qualora sia possibile farlo;
- GC impiega 2 unità di tempo per accorgersi di non poter liberare memoria.

Ad ogni momento il sistema operative può indicare, su richiesta e con ritardo trascurabile, se la quantità di heap utilizzato sia minore del 50% o no (la quantità di memoria di heap usata cambia dipendentemente dall'allocazione, deallocazione e attivazione del GC, ma questo aspetto non va descritto).
Si consiglia di utilizzare i seguenti predicati:
- CallGC(t): viene chiamato GC all'istante t;
- HighLoad(t): all'istante t l'occupazione dello heap è maggiore o uguale al 50% della memoria allocata;
- Free1K(t): GC segnala all'istante t di aver deallocato dallo heap un blocco di memoria da 1K;
- GCUnable(t): GC segnala all'istante t di non essere in grado di liberare memoria.

Esercizio 4 (7 punti)
Un allocatore di memoria gestisce 10KByte di memoria e può ricevere da un insieme di processi richieste di allocare 2KByte, oppure 3KByte di memoria. Inoltre può ricevere da un garbage collector (in istanti diversi da quelli delle richieste) il segnale che è stato liberato 1 KByte di memoria. Alle richieste di memoria da parte dei processi l'allocatore risponde con un segnale di OK, quando la memoria richiesta è disponibile, o con un segnale outOfMemory altrimenti. Modellare l'allocatore sopra descritto con il più semplice formalismo possibile, che parta dalla configurazione con tutti i 10KByte disponibili e tenga conto dell'evoluzione della memoria disponibile a seconda delle richieste da parte dei processi e della restituzione da parte del garbage collector.

Esercizio 5 (6 punti)
Si considerino i seguenti linguaggi su alfabeto $\Sigma=\{a, b, c, d\}$: $L1=\{a^n b^n c^m d^m \mid m>0, n>0\}$, $L2=\{a^n b^m c^m d^n \mid m>0, n>0\}$.
Si risponda alle seguenti domande, motivando adeguatamente le risposte.
1. Qual è il tipo di grammatica a potenza minima che genera L1? Nel caso di L2 come cambia la risposta?
2. Qual è il tipo di grammatica a potenza minima che genera $L3 = L1 \cap L2$?
3. Si assuma che a=b=c=d, cioè $|\Sigma|=1$: come cambiano le risposte precedenti?

Soluzioni

Esercizio 1

1. Si, perché l'insieme S è finito e per qualsiasi numero intero è facile verificare se si tratta di un multiplo di un numero dato oppure no.
2. No, per il Teorema di Rice: l'insieme degli indici di MT che corrispondono ai programmi con le caratteristiche richieste è un sottinsieme proprio non nullo dei numeri naturali.
3. Si, è dominio della funzione computabile e parziale che, dato in ingresso un indice di MT che corrisponde ad uno dei programmi in oggetto, sia esso P, esegue P un passo alla volta fino a quando ha scritto tutti i numeri di S (finito), poi ne interrompe l'esecuzione e restituisce 1. Nota: P può restituire anche altri numeri (prima, dopo o durante aver elencato quelli in S).

Esercizio 2

This may be a hypothetic implementation with the RAM formalism:

```
        READ   1        // reads the number n from input
        LOAD=  0
        STORE  2        // saves in the 2nd  cell the counter
        LOAD=  K        // K is a constant define somewhere else
        SUM=   1000     // M[0] = 1000 + K
        STORE  3        // saves in the 3rd cell the max value for the counter
loop:   LOAD   2
        ADD=   1000     // computes the address of the cell to check, M[0] = 1000 + M[2]
        STORE  4        // stores the address of the cell to check in M[4] = M[0] = 1000 + M[2]
        LOAD   1        // load n in M[0] = n
        DIV    4        // M[0] = M[1000 + M[2]] / n
        MULT   1        // multiply the result of the division by n, M[0] = M[0] * n
        SUB    1        // check that the result is n, in that case n is a multiple
        JZ     exit-ok  // jump to exit-ok if n is a multiple
        LOAD   2
        ADD=   1
        STORE  2        // increment the counter
        LOAD=  1000
        SUB    2        // check that the counter is less than 1000
        JGZ    loop     // reenter the loop if not entirely scanned
        WRITE=0
        HALT

exit_ok: WRITE=1        // print 1
        HALT
```

To solve the problem, it suffices to have a simple cycle that checks whether *n* is multiple of any of the K stored numbers. To check that n is a multiple of a number m belonging to the array, first we can divide n by m (the result is truncated if the integer division is not exact), then we can multiply the result of the division by m and check if the result is equal to n.

With the uniform criterion, each instruction takes a constant amount of time. Moreover, we take the loop K times in the worst case (where K is fixed and is the length of the array, so the worst case is when the unique divisor is in the last cell of the array), so the temporal cost is $\Theta(K)$ or $\Theta(1)$.

With the logarithmic criterion, we take the loop again K times, but the costs of the MULT and DIV instructions is not constant anymore, but they take log(n) * log(M), where M is the maximum value contained in the array. So the global cost may be expressed as K * log(n) * log(M) but, since the array is fixed (both in its length and for the content), the temporal cost is $\Theta(\log n)$.

The variation of the problem requires light changes on the RAM program. First we need a cycle to read the N integers and then we search for an exact divisor in the array using the same loop used in the first solution.
With the uniform criterion, each instruction takes a constant amount of time but we take the loop N times in the worst case, so the temporal complexity is $\Theta(N)$.

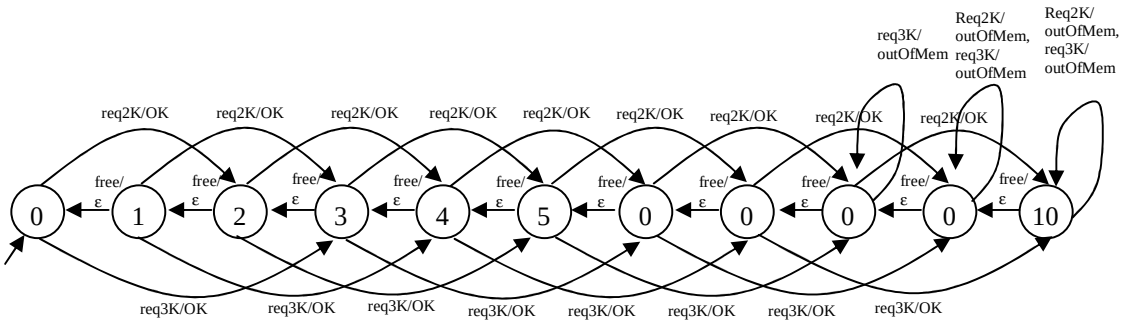
With the logarithmic criterion instead, we need to take into account not only the parameter n, but also the length of the array (which is N) and the maximum value M contained in the array. So, for the first loop (read and load N integers), the cost in the worst case is $\Theta(N*\log(M))$. For the second loop is $\Theta(N(\log(M) + \log(n) + \log(N)))$ ----- N-times the loop * (cost of multiplying and dividing the maximum value by n + cost of incrementing the counter to address array cells) ------.
So the global temporal complexity is $\Theta(N(\log(M) + \log(n) + \log(N)))$.

Esercizio 3

$\forall t'(t<t'<=t+5 \Rightarrow \neg HighLoad(t')) \Rightarrow \exists t''(t<t''<=t+5 \wedge CallGC(t''))$

$\forall t'(t<t'<=t+3 \Rightarrow HighLoad(t')) \Rightarrow \exists t''(t<t''<=t+3 \wedge CallGC(t''))$
$CallGC(t) \Rightarrow (Free1K(t+1) \wedge \neg CGUnable(t+2) \vee \neg Free1K(t+1) \wedge CGUnable(t+2))$
$CGUnable(t) \Rightarrow CallGC(t-2)$
$Free1K(t) \Rightarrow CallGC(t-1)$

Esercizio 4
Un trasduttore finito con 11 stati {0, 1, … 10} corrispondenti alla quantità di memoria occupata.
I={req2k, req3k, free1k}, O={OK, outOfMemory}
Grafo della funzione di transizione:



Esercizio 5
1. For both L1 and L2 a context free grammar suffices.
2. Since L3 = $a^n b^n c^n d^n$ a general grammar is needed.
3. In this case L1 = L2 = $\{a^{2k} \mid k>1\}$, while L3 = $\{a^{4k} \mid k>0\}$; all these languages are regular.