Algoritmi e Principi dell'Informatica

Appello dell'11 Settembre 2013

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 3 ore.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1, 2 e 3, in 1 ora e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 4, 5, 6 in 1 ora e 30 minuti.

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

Esercizio 1 (punti 4)
Sia data la seguente grammatica G:
S → B S a | C
BC → C b b
C → c

1. Si dica quale è il linguaggio generato da G.
2. Si scriva un automa che riconosce il linguaggio generato da G. L'automa deve appartenere alla classe di potenza riconoscitiva minima tra quelle che riconoscono il linguaggio generato da G.

Esercizio 2 (punti 5)

Si consideri il seguente programma (in pseudocodice), il cui scopo è ordinare un generico array A di lunghezza variabile il cui valore è memorizzato nel campo *length* (si supponga che il campo *key* dei vari elementi sia un intero in valore assoluto ≤ 100):

```
INSERTION-SORT(A)
1 for j := 2 to A.length
2   key := A[j]
3   //Inserisce A[j] nella sequenza ordinata A[1..j−1]
4   i := j − 1
5   while i > 0 and A[i] > key
6     A[i + 1] := A[i]
7     i := i − 1
8   A[i + 1] := key
```

1. E' decidibile il problema di stabilire se il programma suddetto ordina correttamente un generico array di lunghezza ≤ 100?
2. E' decidibile il problema di stabilire se il programma suddetto ordina correttamente un generico array di lunghezza qualsiasi?

Esercizio 3

Prima parte (obbligatoria:punti 6)

Si consideri un sistema costituito da un processo che gestisce una risorsa R utilizzata da due processi utenti U1 e U2.

- Ognuno dei due utenti richiede di tanto in tanto l'accesso alla risorsa R, in *mutua esclusione*.
- Né U1 né U2 possono chiedere nuovamente l'uso di R se la loro precedente richiesta non è stata prima soddisfatta.
- Il processo gestore assegna R ai richiedenti, in *mutua esclusione e in alternanza*.
- Al termine dell'uso di R il processo utente rilascia la risorsa.

Si formalizzi questo comportamento del sistema mediante una rete di Petri.

Seconda parte (facoltativa: ulteriori punti 4. NB: la parte facoltativa verrà valutata solo se sarà stata svolta correttamente la parte obbligatoria.)

La versione precedente ha il difetto di imporre una rigida alternanza tra il servizio di U1 e quello di U2, con l'effetto di bloccare entrambi i processi se uno dei due, pur potendo effettuare una richiesta, non la esegue quando è il proprio turno. Ad esempio non sarebbe possibile la seguente sequenza, assumendo che inizialmente né U1 né U2 abbiano richiesto la risorsa:

U1 richiede la risorsa e la ottiene; successivamente la risorsa viene rilasciata e U1 la chiede nuovamente; U2 però nel frattempo non l'ha ancora richiesta e quindi la risorsa viene nuovamente assegnata a U1 (mentre se U2 nel frattempo l'avesse richiesta, essa dovrebbe essere *obbligatoriamente* assegnata ad U2).

Si modifichi la precedente rete di Petri in modo da permettere che la risorsa venga assegnata anche più volte di seguito allo stesso processo ma solo se l'altro non l'ha richiesta.

Esercizio 4 (punti 5)

Si consideri il problema di cercare un elemento x compreso tra 0 e 9 in una sequenza ordinata S di lunghezza n che contiene solo i numeri fra 0 e 9 (ciascun elemento può apparire più volte in S).

Si descriva a grandi linee il comportamento di una macchina di Turing a k nastri che, ricevendo in ingresso la stringa $x\$S$, sia in grado, facendo uso di un algoritmo di ricerca binaria, di stabilire se x appartiene a S e se ne calcolino la complessità temporale e spaziale. Sono preferibili soluzioni che minimizzano la complessità *temporale*.

Esercizio 5 (punti 6)

Si descriva un algoritmo, mediante opportuno pseudocodice, che calcoli il determinante del prodotto tra due matrici A e B $n \times n$, sotto l'ipotesi che A e B siano entrambe triangolari superiori (ossia abbiano tutti 0 al di sotto della diagonale principale) o, simmetricamente, triangolari inferiori. Si valuti la complessità asintotica dell'algoritmo in funzione della dimensione n delle matrici.

Suggerimento (per chi avesse qualche difficoltà a richiamare alla memoria gli elementi basilari di algebra lineare)

Ricavare proprietà e algoritmi generali estrapolandoli da esempi relativi a matrici "piccole": e.g., 2×2, 3×3, ...

Esercizio 6 (punti 5)

Si risponda alle seguenti domande, motivando opportunamente le risposte.

1. Si considerino alberi red-black che ammettono l'esistenza di chiavi duplicate:

   a. è possibile che ci siano due elementi con la stessa chiave che *non sono* uno il padre dell'altro?

   b. è possibile che ci siano due elementi con la stessa chiave che *non sono* uno antenanto dell'altro?

2. Modificare gli algoritmi di `RB-INSERT-RB` e `RB-DELETE` visti a lezione in modo da impedire che ci possano essere in un albero chiavi duplicate.
   Dire come cambia (se cambia) la complessità degli algoritmi modificati rispetto alle versioni originali.

Tracce di soluzioni

Esercizio 1

Il linguaggio generato dalla grammatica è L(G) = {c b^{2n} a^{n} | n >= 0}, che è riconosciuto da un semplice automa a pila deterministico.

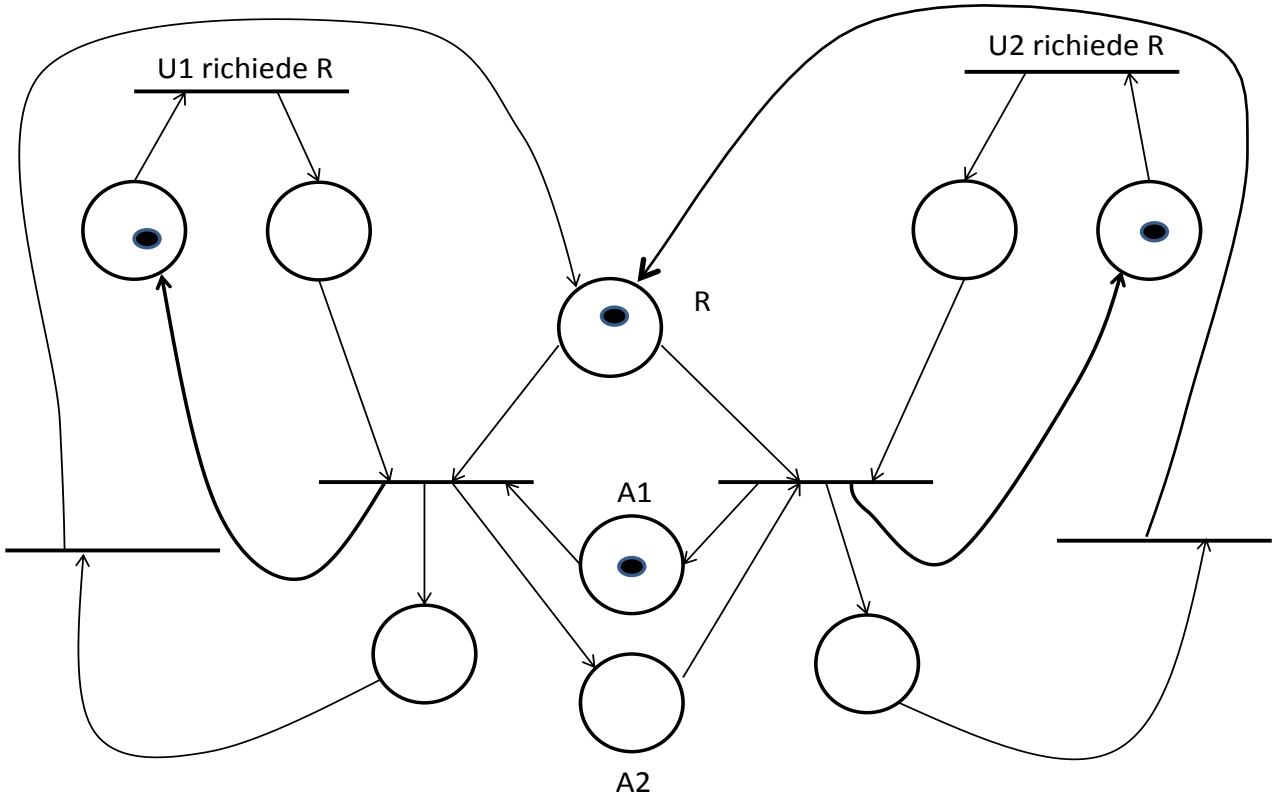
Esercizio 2

La risposta è positiva in entrambi i casi. Infatti il problema può essere *deciso, dimostrando* in effetti la correttezza dell'algoritmo codificato dal programma: in letteratura si trovano diverse dimostrazioni matematiche della sua correttezza, ma anche una semplice analisi informale del codice può portare alla *decisione* che esso effettivamente ordina correttamente un qualsiasi array. Ovviamente, una volta deciso il problema nel caso generale ne consegue la stessa decisione nel caso particolare.

Tuttavia nel primo caso è possibile constatare immediatemente la decidibilità del problema posto, anche senza deciderlo: basta infatti constatare che in tal caso il dominio del problema è finito (l'insieme degli array di lunghezza ≤ 100 e i cui elementi siano a loro volta interi compresi tra -100 e + 100): di conseguenza il problema può essere risolto alla peggio mediante testing esaustivo –posto che a sua volta il problema di stabilire se un array è ordinato è decidibile-.

Esercizio 3

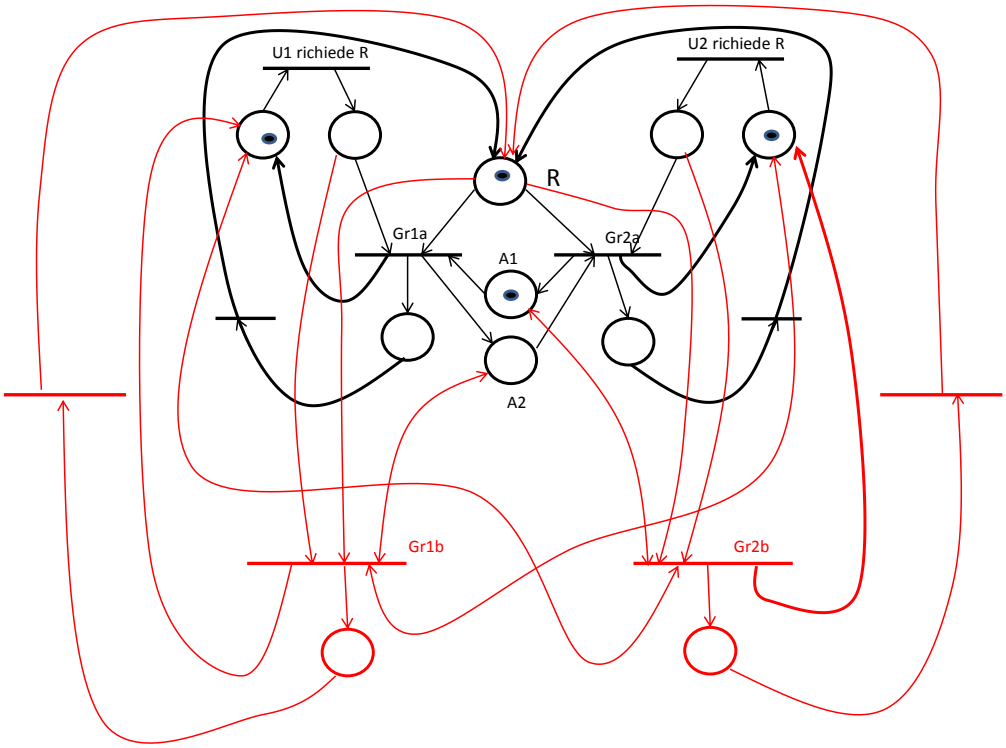
Prima parte (obbligatoria)



A1 e A2 impongono l'alternanza tra il servizio di U1 e quello di U2

Seconda parte (opzionale)

Aggiungendo alla rete precedente le parti in rosso della rete seguente si ottiene l'effetto che, ad esempio, Gr1a (ossia grant di R a U1 avvenga solo se "è il turno di U1"; ma se fosse il turno di U2 e U2 non avesse richiesto la risorsa – e solo in tal caso!-, essa sarebbe ugualmente assegnata ad U1 tramite la transizione Gr1b. In sostanza il comportamento della rete viene reso deterministico in ambo i casi, ma nel primo caso imponendo un'alternanza obbigatoria, nel secondo determinando la scelta tra la concessione di tipo a e quella di tipo b a seconda che sia il proprio turno oppure no ma in assenza di richiesta da parte dell'altro processo.



Esercizo 4

Una soluzione banale consiste nel codificare l'algoritmo tradizionale di ricerca binaria che effettua log(n) accessi alla sequenza S. Tale algoritmo può essere codificato dalla MT con complessità temporale O($n$log$n$); infatti poiché nelle macchine di Turing l'accesso è sequenziale ciascun accesso richiede fino a n passi.

È però possibile ottenere una complessità lineare nel seguente modo, con una macchina di Turing con 3 nastri.

1) Copiare x sul nastro 1 e spostare la testina all'inizio di S.
2) Salvare la lunghezza di S sul nastro 2 (alla fine del procedimento la testina sul nastro di ingresso sarà alla fine di S così come la testina sul nastro 2).
3) Scandire il nastro 2 da sinistra verso destra. Ogni due passi sul nastro 2, muovere di una posizione a sinistra la testina sul nastro di ingresso e scrivere un simbolo sul nastro 3. (alla fine la testina sul nastro in ingresso sarà sul simbolo centrale di S, la testina sul nastro 2 sarà all'inizio del nastro e quella sul nastro 3 sarà alla fine del nastro).

4) Confrontare il simbolo centrale con *x* (salvato nel nastro 1):
    a. Se coincidono, l'esecuzione termina.
    b. Se *x* è minore del simbolo centrale, ripetere i passi 3 e 4 usando il nastro 3 al posto del 2 e il 2 al posto del 3 e muovendosi a destra invece che a sinistra sul nastro di ingresso a partire dalla posizione corrente (si considera come *S* la seconda metà della sequenza corrente).
    c. Se *x* è maggiore del simbolo centrale, ripetere i passi 3 e 4 usando il nastro 3 al posto del 2 e il 2 al posto del 3 (si considera come *S* la prima metà della sequenza corrente).

Con questo procedimento la complessità diventa lineare in quanto la macchina effettua al più log(n) operazioni che costano rispettivamente n, $n/2$, $n/4$, … , cioè $\sum_{i=1}^{logn} 2^i \cong n$.

In entrambe le soluzioni la complessità spaziale è lineare.

Esercizo 5

Si rammenti che il prodotto tra una matrice $n \times m$ e una matrice $m \times p$ è una matrice $n \times p$ il cui generico elemento in osizione *i, j* è datto dalla $\sum_{k=1}^{m} a[i,k] \cdot b[kj]$.

Si constata allora facilmente che il prodotto tra due matrici triangolari (ad esempio, superiori) è a sua volta una matrice triangolare superiore i cui elementi sulla diagonale principale sono il prodotto *a[i, i]*. *b[i,i]*. Notoriamente il determinante di una matrice triangolare è il prodotto degli elementi sulla diagonale principale; esso è quindi dato dalla formula

$$\prod_{i=1}^{n} a[i,i] \cdot b[i,i]$$

che evidentemente può essere calcolata in O(n) a criterio di costo costante.

Esercizo 6

1.a E' possibile, per esempio inserendo le seguenti chiavi: 3, 2, 5, 3

1.b E' possibile, per esempio inserendo 3 volte la stessa chiave (2 sono foglie)

2. C'è da modificare solo l'algoritmo di `RB-NSERT`; la modifica può essere fatta in diverse maniere, per esempio ricercando, prima di inserire l'elemento, se c'è già nell'albero la chiave da inserire. La complessità dell'algoritmo modificato non cambia in termini di comportamento asintotico.

Si può modificare l'algoritmo in modo anche da controllare se esiste già la chiave da inserire mentre si effettua l'inserimento.