Algoritmi e Principi dell'Informatica

Parte I – Modelli e computabilità

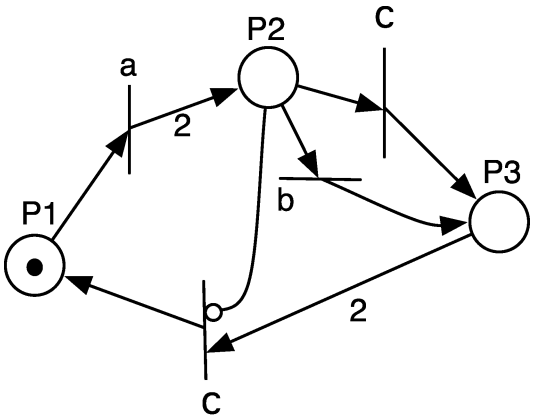
Appello d'esame - 25 Febbraio 2011

Tempo a disposizione: 1h30

NB: Motivare sempre adeguatamente le proprie risposte

Esercizio 1 (11 punti)

Sia data la rete di Petri in figura, con F = {M_F}, $M_F(P_1) = 1$, $M_F(P_2) = 0$, $M_F(P_3) = 0$.



1. Scrivere una grammatica che generi lo stesso linguaggio L della rete di Petri. La grammatica deve avere il minor numero di simboli **nonterminali** possibile.
2. Il tipo della grammatica scritta è a potenza generativa minima tra quelle in grado di generare il linguaggio L in oggetto?
3. La rete di Petri è a potenza minima tra quelle in grado di generare L? Ossia, vi sono tipi di e reti di Petri meno potenti del genere di quella scritta in grado di generare L?

Esercizio 2 (12 punti + Bonus)

Si indichi, come al solito, con f_y la funzione calcolata dalla y-esima Macchina di Turing.

1. E' decidibile il problema di stabilire se f_{10} è definita per ogni $x \leq 10$ (cioè se $\forall x(x \leq 10 \rightarrow f_{10}(x) \neq \perp)$)?
2. E' decidibile il problema di stabilire se, per y generico, f_y è definita per ogni $x \leq 10$?
3. E' semidecidibile il problema del punto 2?
4. E' decidibile il problema di stabilire se, per y generico, f_y è definita per ogni $x > 10$?
5. **Bonus:** E' semidecidibile il problema del punto 4?

Esercizio 3 (7 punti + Bonus)

Si consideri la grammatica G_1 con le seguenti produzioni:

$$T \rightarrow aTbT \mid bTaT \mid \varepsilon$$

e simbolo iniziale T . G_1 genera tutte e sole le stringhe in cui le *a* e le *b* sono in ugual numero.
Si consideri ora la grammatica G_2 con le seguenti produzioni:

$$S \rightarrow TaT$$
$$T \rightarrow aTbT \mid bTaT \mid \varepsilon$$

e simbolo iniziale S.
1. Qual è il linguaggio generato da G_2 ? Motivare la risposta.
2. **Bonus:** fornire una dimostrazione della risposta data al punto 1.

Algoritmi e Principi dell'Informatica

Parte II – Complessità, algoritmi, strutture dati

Appello d'esame - 25 Febbraio 2011

Tempo a disposizione: 1h30

Esercizio 1 (12 punti)

Si consideri il problema di riconoscere il linguaggio L = { ww | w $\in$ {a,b}* }.

1. Si descriva in modo informale, ma sufficientemente preciso, una macchina di Turing deterministica a k nastri di memoria che riconosce il linguaggio L desiderato minimizzando la complessità *spaziale*. Si diano le complessità temporale e spaziale della macchina scritta.
2. Si descriva in modo informale, ma sufficientemente preciso, una macchina di Turing **non**deterministica a k nastri di memoria che riconosce il linguaggio L desiderato minimizzando la complessità *temporale*. Si diano le complessità temporale e spaziale della macchina scritta.

Esercizio 2 (14 punti)

1. Si scriva un algoritmo che, dato in ingresso un array A di numeri naturali (tutti diversi tra di loro) ed un valore x, restituisce *true* se nell'array esistono tre indici i, j, k diversi tra di loro tali che $A[i] < A[j] < A[k]$ e che la somma di $A[j] - A[i]$ con $A[k] - A[j]$ dia x, *false* altrimenti

Per esempio, data in ingresso la sequenza A = [8, 20, 27, 6, 4, 0, 17, 95, 1] ed il valore x = 10, l'algoritmo deve restituire *true*, in quanto presi come indici $i = 7$, $j = 2$, e $k = 3$ (quindi $A[7] = 17$, $A[2] = 20$, $A[3] = 27$) si ha che la somma di $A[2]$ - $A[7]$ (cioè 3) e di $A[3]$ - $A[2]$ (cioè 7) dà 10 come desiderato.

2. Si dia la complessità temporale dell'algoritmo scritto.

3. Supponendo di realizzare l'algoritmo descritto al punto 1 con una macchina RAM (della quale non è necessario scrivere il codice), si diano le complessità temporale e spaziale dell'algoritmo scritto calcolate a criterio di costo logaritmico. Si supponga che la sequenza di numeri non sia già in memoria, ma debba essere letta da standard input.

NB1: E' preferibile che l'algoritmo venga scritto in pseudocodice; tuttavia, verranno accettate anche soluzioni in cui l'algoritmo è descritto in modo informale, purché la descrizione sia sufficientemente precisa per poterne valutare la complessità.
NB2: Il punteggio assegnato sarà tanto più alto quanto migliore sarà la complessità dell'algoritmo proposto.

Esercizio 3 (8 punti)

Si vuole realizzare una applicazione in cui c'è da gestire un insieme di elementi per il quale si sa (o si suppone) che il numero di esecuzioni di operazioni di ricerca sia molto più alto delle esecuzioni delle operazioni di inserimento/cancellazione.
Per questo si vuole realizzare una tabella hash in cui la risoluzione dei conflitti viene fatta memorizzando le chiavi con stesso valore hash non più in una lista, ma in un albero binario di ricerca (BST).

1. Dire come cambiano, se cambiano, le complessità temporali nel caso **medio** delle operazioni di INSERT, DELETE, e SEARCH effettuate sulla tabella basata su BST rispetto alla soluzione basata su lista, sotto l'ipotesi di hashing uniforme semplice.

2. Date le caratteristiche della applicazione, possiamo dire che la scelta del progettista di modificare la struttura dati sia stata azzeccata oppure no? Motivare la risposta.

3. Cambierebbero le complessità del caso medio se, invece di usare dei BST, il progettista avesse usato degli alberi red-black?

Soluzioni – Parte I

Esercizio 1

1. Il linguaggio riconosciuto dalla rete di Petri è: $L = (a(b|c)(b|c)c)^*$. Una grammatica G con il numero minimo di nonterminali (1) che generi L è la seguente:
   $S \rightarrow \varepsilon \mid abbcS \mid abccS \mid acbcS \mid acccS$
2. La grammatica G è di tipo non contestuale. Il linguaggio L è regolare, pertanto G non è a potenza minima in quanto basterebbe una grammatica regolare per generare L.
3. La rete di Petri del testo fa uso di archi inibitori, pertanto non è a potenza minima, in quanto per riconoscere L, che è un linguaggio regolare, sarebbe senz'altro sufficiente una rete di Petri senza archi inibitori (addirittura ne basterebbe una 1-limitata).

Esercizio 2

1. Si tratta di una domanda con risposta chiusa sì/no che non dipende da alcun input, pertanto il problema è decidibile.
2. Non è decidibile per il teorema di Rice, in quanto l'insieme di funzioni descritto non è l'insieme vuoto, né è l'insieme universo.
3. E' semidecidibile, in quanto basta provare le computazioni da 0 a 10 di f_y per un numero crescente di passi, secondo l'usuale tecnica diagonale di simulazione. Se tutte e 11 le esecuzioni terminano, prima o poi lo scopriamo, da cui la semidecidibilità.
4. No, per il teorema di Rice, con ragionamento analogo al punto 2.
5. Non è nemmeno semidecidibile. Se fosse semidecidibile anche questo problema, unitamente al fatto che è semidecidibile stabilire se f_y sia definita per $x \leq 10$ (vedi punto 3), allora sarebbe semidecidibile anche il problema di stabilire se una generica funzione sia definita per ogni x, ovvero sia totale, il che è notoriamente falso.

Esercizio 3

1. Il linguaggio generato da G_2 è il linguaggio di tutte e sole le stringhe in cui il numero di *a* è pari al numero di *b* più uno, ossia $L(G_2) = \{x \mid x \in \{a,b\}^* \wedge \#a(x) = \#b(x) + 1\}$. Come affermato nel testo, le stringhe generate a partire da T sono quelle del linguaggio $L(G_1)$ di tutte e sole le stringhe in cui le *a* e le *b* sono in ugual numero. Le stringhe di $L(G_2)$, tramite la produzione $S \rightarrow TaT$, sono generate dalla giustapposizione di una *a* tra due stringhe di L(G1). Poiché una qualunque stringa *x* tale che $\#a(x) = \#b(x) + 1$ è tale che esiste una *a* senza una corrispondente *b*, la grammatica G_2 genera *tutte* le stringhe con numero di *a* uguale al numero di *b* più 1.

2. Da quanto scritto sopra, le stringhe generate da G_2 hanno la forma $x \cdot a \cdot y$, dove $x,y \in L(G_1)$. E' immediato concludere che G_2 genera soltanto stringhe in cui vi sia una a in più rispetto alle *b*, poiché sia *x* sia *y* sono già bilanciate, appartenendo a $L(G_1)$. Per mostrare che G_2 genera tutte le stringhe in cui vi sia una *a* in più rispetto alle *b*, si consideri una generica stringa *w* tale per cui $\#a(w) = \#b(w) + 1$. Poiché *w* contiene più *a* che *b*, scandendo *w* da sinistra a destra, esiste necessariamente un carattere di *w* che è la prima *a* in soprannumero rispetto alle *b* incontrate (incluso il caso in cui non si siano incontrate *b* e cioè che la *a* sia il primo carattere di *w*). Sia x la sottostringa composta da tutti i caratteri a sinistra di quella *a*, e *y* quella composta da tutti i caratteri a destra della *a*. Per quanto scritto, la *a* che separa *x* da *y* è la prima *a* in soprannumero, quindi in *x* le *a* e le *b* sono in ugual numero. A questo punto segue necessariamente che anche nella *y* le *a* e le *b* sono in ugual numero, altrimenti la stringa $w = x \cdot a \cdot y$ non soddisferebbe più il vincolo $\#a(w) = \#b(w) + 1$.

Soluzioni – Parte II

Esercizio 1

1. Per riconoscere il linguaggio L con una MT deterministica minimizzando la complessità spaziale è sufficiente usare una MT a 2 nastri di memoria, che funzioni nel seguente modo:

1. prima legge tutti i simboli in input, mano a mano incrementando un contatore binario memorizzato nel nastro T1;
2. divide il valore del contatore eliminando lo 0 finale (si bocca in stato non finale se l'ultima cifra binaria non è uno 0);
3. copia il contenuto di T1 nel nastro T2, e quindi si porta sul primo carattere in input
4. memorizza il carattere corrente nello stato, e si porta avanti n/2 celle (con n = |ww| il valore memorizzato in T1), quindi controlla che la cella di arrivo contenga un carattere uguale a quello della cella di partenza;
5. copia il contatore da T1 a T2 e torna indietro n/2-1 celle (sfruttando di nuovo il contatore in T2), quindi di nuovo copia il contatore da T1 a T2 e ripete dal passo 4;
6. la MT accetta se arriva in fondo al nastro di input senza avere incontrato coppie di caratteri diverse.

La complessità spaziale di una tale MT è $\Theta(\log(n))$ (lo spazio occupato dai contatori), mentre quella temporale è $\Theta(n^2\log(n))$ (l'analisi di ognuno degli n/2 caratteri di w richiede di spostarsi n/2 celle in avanti, ed ogni spostamento richiede il decremento di un contatore di lunghezza log(n)).

2. Una MT nondeterministica a 2 nastri di memoria può semplicemente scandire il nastro in ingresso, copiando l'input sul nastro T1, fino a che, nondeterministicamente, decide di cominciare a copiare i caratteri in input sul nastro T2, fino alla fine dell'input. A quel punto deve scandire all'indietro i 2 nastri, e controllare che contengano gli stessi caratteri.

Le complessità temporale e spaziale di una simile macchina di Turing sono entrambe $\Theta(n)$

Esercizio 2

1. Un algoritmo che risolve il problema dato è il seguente:

```
EXACT-DIFF(A, x)
1  MERGE-SORT(A, 1, A.length)
2  i := 1
3  k := 3
4  while k ≤ A.length and i < A.length-1
5    if k = i+1
6       k := k+1
7    elseif A[k] - A[i] = x
8      return true
9    elseif A[k] - A[i] < x
10     k := k+1
11   else i := i+1
12 return false
```

2. La complessità temporale dell'algoritmo definite al punto 1 è determinata dall'ordinamento della riga 1, che ha complessità $\Theta(n \log(n))$. Infatti il ciclo 4-11 viene eseguito al più *n* volte in quanto nessuno degli indici *i*, *k* viene mai decrementato.

3. L'algoritmo realizzato con macchina RAM avrebbe complessità temporale, valutata a criterio di costo logaritmico, $\Theta(n \log^2(n))$ in quanto il MERGE-SORT richiede di accedere alle celle di un array di lunghezza n, ed il costo di tali accessi è $\Theta(\log(n))$ (consideriamo per semplicità i valori contenuti nell'array come delle costanti, altrimenti andrebbe aggiunto un fattore log(v), con v il valore massimo presente nell'array).
La complessità spaziale invece sarebbe $\Theta(n)$ in quanto occorrono *n* celle per memorizzare i valori dell'array A.

Esercizio 3

1. Nel caso medio, il numero di elementi in conflitto è pari a quella che sarebbe la lunghezza della lista, cioè α, con α fattore di carico; di conseguenza, in media il numero di elementi in ogni albero è anch'esso α. Supponendo che gli elementi arrivino in modo casuale, ogni albero ha in media altezza $\log(\alpha)$, quindi le operazioni di INSERT, DELETE e SEARCH hanno tutte complessità $O(\log(\alpha))$ nel caso medio.

2. La scelta è stata opportuna, in quanto l'operazione più usata, la ricerca, risulta essere più efficiente nel caso di alberi (la complessità è $O(\log(\alpha))$ invece che $O(1+ \alpha)$ come per le liste).

3. Nel caso di alberi red-black la complessità media non cambia, in quanto ancora il numero medio di elementi negli alberi sarebbe α, e la loro l'altezza sarebbe $\log(\alpha)$.