# Algoritmi e Principi dell'Informatica

Prima prova in itinere - 15 Novembre 2010

**Tempo a disposizione: 1h30**

**Esercizio 1 (10 punti)**
Si scriva un automa (a stati finiti, a pila, macchina di Turing oppure rete di Petri) che riconosce il linguaggio L fatto di tutte e sole le stringhe della forma $a^{n_1}b^{n_2}c^{n_3}$ in cui $0 < n_1 < n_2 < n_3$.

**NB**: Il punteggio massimo verrà dato solo se l'automa scritto è a *potenza minima* tra quelli in grado di riconoscere L.

**Esercizio 2 (11 punti)**
Formalizzare mediante formule di logica del prim'ordine le seguenti affermazioni:

2.1. L'insieme dei numeri naturali primi è illimitato.
2.2. C'è una infinità di macchine di Turing che calcolano funzioni totali.
2.3. Esistono macchine di Turing che calcolano funzioni con dominio finito.

Si indichi come al solito con la funzione a due argomenti $f_y(x)$ il valore calcolato dalla y-esima macchina di Turing con ingresso x.

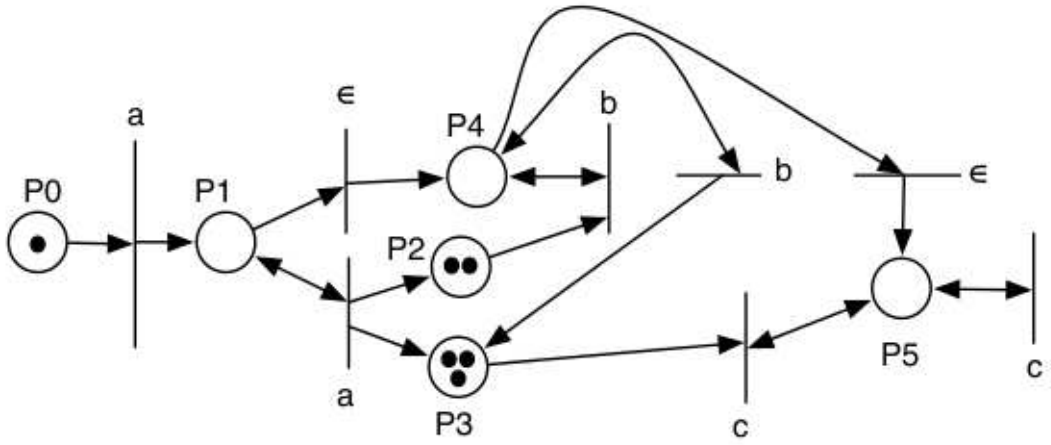
**Esercizio 3 (11 punti)**
Il professor Rice dà ai suoi studenti il seguente esercizio:

> "Scrivere un automa (a stati finiti, a pila, o macchina di Turing) che riconosca il linguaggio L(G) generato dalla seguente grammatica G:
> $S \rightarrow ABCS \mid cABCABC$
> $ABC \rightarrow a \mid b \mid c$
> L'automa scritto deve essere a potenza riconoscitiva minima tra quelli che riconoscono L(G), o la soluzione proposta sarà considerata sbagliata, e riceverà 0 punti."

3.1. E' decidibile il problema di stabilire se uno studente, in risposta all'esercizio, scrive una macchina di Turing, un automa a pila, o un automa a stati finiti?
3.2. E' possibile scrivere un programma che faccia in automatico la correzione delle soluzioni proposte dagli studenti, e cioè che, data una qualunque soluzione proposta, dica se essa è corretta oppure no?

**Soluzioni**

**Esercizio 1**
L'automa a potenza minima tra quelli in grado di riconoscere il linguaggio desiderato è una rete di Petri senza archi inibitori (il linguaggio non è invece riconosciuto da nessun automa a pila, neanche nondeterministico).
Una rete di Petri che riconosce il linguaggio L desiderato è la seguente:




L'insieme F delle marcature finali contiene la sola $M_F$, che è tale che $M_F(P5) = 1$, e $M_F(P_i) = 0$ per tutti gli altri posti diversi da P5.

**Esercizio 2**

1. $\forall x \exists y (y>x \wedge \neg\exists z,q (1<z<y \wedge y=zq))$
2. $\forall x (\exists z (z>x \wedge \forall y (f_z(y) \neq \perp)))$
3. $\exists x (\exists z (\forall y(y > z \rightarrow f_x(z) = \perp)))$

**Esercizio 3**
La grammatica G, nonostante sia non ristretta, genera un linguaggio regolare; più precisamente, essa genera il linguaggio L(G) fatto di stringhe costruite sull'alfabeto {a,b,c} in cui il terz'ultimo carattere è una c.

1. Il problema è decidibile, in quanto si tratta di fare una semplice analisi sintattica dell'automa scritto dall'utente (non è difficile immaginare di codificare mediante opportune stringhe le funzioni di transizioni corrispondenti agli automi scritti dagli studenti): se sugli archi compaiono solo simboli dell'alfabeto di input, l'automa è a stati finiti; se le etichette sugli archi sono del tipo a,A/β, l'automa è a pila (e si può capire se è deterministico o no analizzando, per ogni stato, le transizioni che escono da esso); se le etichette sono del tipo a,A/B,<$M_0$,$M_1$> (con $M_i \in \{R,L,S\}$), l'automa è una macchina di Turing (si può capire quanti nastri ha la macchina di Turing da quanti simboli compaiono prima della "/").

2. Il problema è decidibile. Infatti, poiché L(G) è un linguaggio regolare, si danno 2 casi: o lo studente scrive un automa più potente di un FSA, nel qual caso il programma se ne può accorgere (si veda la soluzione del punto 1) e segnalare che l'esercizio è sbagliato; oppure lo studente scrive un FSA, ed in questo caso il problema da risolvere è una semplice equivalenza di FSA (quello scritto dallo studente e quello che riconosce L(G), che è decidibile.

# Algoritmi e Principi dell'Informatica

Seconda prova in itinere - 4 Febbraio 2011

**Tempo a disposizione: 1h30**

## Esercizio 1 (12 punti)

Si consideri il problema di calcolare, a partire da un valore *n* in input, il valore $n2^n$.

1. Si descriva in modo informale, ma sufficientemente preciso, una macchina di Turing a k nastri con alfabeto binario {0,1} (sia per il nastro di input che per i nastri di memoria) che calcola la funzione desiderata, supponendo che il valore *n* si trovi codificato in binario nel nastro di input. Si diano le complessità temporale e spaziale della macchina scritta.
2. Si descriva (in pseudocodice) una macchina RAM che calcola la funzione desiderata. Si diano le complessità temporale e spaziale della macchina scritta, calcolate con criterio di costo logaritmico.

Quale dei 2 meccanismi proposti è il più efficiente per eseguire il calcolo in questione?

## Esercizio 2 (14 punti)

1. L'algoritmo di `MERGE-SORT` visto a lezione non ordina "sul posto", in quanto usa una quantità di memoria aggiuntiva (di dimensione non costante) al momento di fare il merge dei 2 sottoarray (che vengono ricopiati ad ogni chiamata di `MERGE` in 2 array di supporto `L` ed `R`).
   1a. Scrivere una variante `MERGE-SORT-IN-PLACE` dell'algoritmo di `MERGE-SORT`, che però ordini sul posto.
   1b. Dire se e come cambia la complessità temporale di `MERGE-SORT-IN-PLACE` rispetto a quulla di `MERGE-SORT`.

2. Definire un algoritmo `MERGE-SORT-LIST-IN-PLACE` che ordini sul posto liste doppiamente concatenate, e darne la complessità temporale.

**NB1:** E' preferibile che gli algoritmi vengano scritti in pseudocodice; tuttavia, verranno accettate anche soluzioni in cui gli algoritmi siano descritti in modo informale, purché la descrizione sia sufficientemente precisa per poter valutare la complessità degli algoritmi.
**NB2:** Il punteggio assegnato sarà tanto più alto quanto migliore sarà la complessità degli algoritmi proposti.

## Esercizio 3 (6 punti)

Dire, giustificando opportunamente le risposte, se le seguenti affermazioni sono vere o false.

1. Ogni volta che un nuovo nodo è aggiunto ad un BST mediante l'algoritmo `TREE-INSERT` visto a lezione, il nodo è aggiunto come foglia dell'albero.

2. Ogni volta che un nuovo nodo è aggiunto ad un albero red-black mediante l'algoritmo `RB-INSERT` visto a lezione, il nodo è aggiunto come foglia dell'albero (più precisamente, per come sono stati realizzati gli alberi red-black a lezione, viene inserito come nodo interno in cui entrambi i suoi figli sono il nodo sentinella `T.nil`).

## Soluzioni

## Esercizio 1

**1.**

Per risolvere in modo efficiente il problema dato con una MT, basta considerare che, per numeri codificati in binario, moltiplicare per 2 corrisponde ad aggiungere uno 0 come cifra meno significativa. Per questo, per ottenere il valore $n2^n$ è sufficiente aggiungere $n$ volte 0 alla fine della stringa binaria che rappresenta $n$ (supponendo che la cifra più significativa sia a sinistra).
Quindi, una MT (traduttrice) che esegue il calcolo desiderato può avere 1 nastro di memoria e comportarsi come segue.

- Innanzi tutto copia $n$ (in binario) sul nastro $T_1$. Contemporaneamente scrive $n$ (sempre in binario) in uscita.
- Con un ciclo, decrementa via via il contatore in $T_1$, e ad ogni decremento appende uno 0 in fondo al nastro di uscita.

Una tale macchina di Turing deve ripetere il ciclo $n$ volte, ed ogni ripetizione costa $\log(n)$ per decrementare il contatore in $T_1$. In totale, quindi, la complessità temporale è $n \cdot \log(n)$, con $n$ valore del numero in input. La complessità spaziale è invece $\log(n)$ (vengono contati solo i nastri di memoria).
Si noti che se, come è abitudine, per la MT consideriamo la lunghezza $m$ della stringa in input come "dimensione del dato", essendo come noto $m = \log(n)$ (cioè $n = 2^m$) allora la complessità temporale diventa $m2^m$, e quella spaziale $m$.

**2.**

Il seguente pseudocodice risolve il problema in questione.

```
1 read(n)
2 ris := 1
3 for i := 1 to n
4   ris := ris*2
5 ris := n*ris
6 write(ris)
```

Il ciclo 3-4 viene eseguito $n$ volte, e ad ogni iterazione *ris* vale $2^i$. Il suo costo, a criterio di costo logaritmico, è quindi $\Sigma^n_{i=1} \log(2^i)$ cioè $\Sigma^n_{i=1} i$, che è $\Theta(n^2)$.

Dei 2 meccanismi di calcolo presentati, quindi, quello realizzato dalla MT è il più efficiente.
Tuttavia, è possibile scrivere una macchina RAM che realizza il calcolo desiderato, ma con complessità migliore.
L'algoritmo è il seguente

```
1  read(n)
2  m := n
3  ris := 1
4  temp := 1
5  while m > 0
6    if m mod 2 = 1
7      ris := ris * temp
8    temp := temp * temp
9    m := m div 2
10 ris := ris*n
11 write(ris)
```

In questo caso il ciclo 5-9 viene eseguito solo $\log(n)$ volte, e l'operazione più onerosa del ciclo è la 7 (che nel caso pessimo, quello in cui $n$ sia una potenza di 2 meno 1, per esempio 15 viene eseguita ad ogni iterazione del ciclo), che costa $\log(2^{2^i} \cdot 2^{2^i})$, cioè $\log(2^{2^{i+1}})$, cioè $2^{i+1}$. Quindi, il costo del ciclo 5-9 è $\Sigma^{\log n}_{i=1} 2^{(i+1)}$ che è $2^{(\log(n)+2)}$, che è $\Theta(n)$.
In realtà, con un meccanismo per certi versi analogo a quello descritto ora per la macchina RAM, è possibile ottenere una complessità $\Theta(n)$ anche per la MT.

## Esercizio 2

**1.**

Per fare in modo che MERGE-SORT ordini sul posto occorre modificare l'algoritmi di MERGE, evitando di usare gli array L ed R. Ciò si ottiene "shiftando" la parte del sottoarray di sinistra ancora da sistemare quando viene preso un elemento da destra.
Lo pseudocodice per il nuovo algoritmo di MERGE, chiamato ora MERGE-IN-PLACE. è il seguente:

```
MERGE-IN-PLACE (A, p, q, r)
1 j := q+1
2 k := p
3 while k < j and j <= r
4   if A[k] > A[j]
5     temp := A[j]
6      // ora spostiamo la parte del sottoarray di sinistra ancora da
sistemare una posizione a destra
7     for l := j downto k+1
8       A[j] := A[j-1]
9     A[k] := temp
10    j := j + 1
11  k := k+1
12  // non c'e' ramo else, perche' se A[k] <= A[j], allora A[k] e' gia' a
posto
```

In questo caso l'algoritmo di MERGE-IN-PLACE costa $\Theta(n^2)$ nel caso pessimo, in quanto, ad ogni ripetizione del ciclo 3-12, può richiedere di spostare $j$-$k$+1 celle che è un numero $\Theta(n)$ (per esempio nel caso in cui tutto il sottoarray di destra sia più piccolo del sottoarray di sinistra, ogni ripetizione del ciclo 7-8 costa $n/2$, con $n=r$-$p$+1 la lunghezza del sottoarray che si sta ordinando).
Quindi, la ricorrenza di MERGE-SORT-IN-PLACE nel caso pessimo diventa
$T(n) = 2T(n/2) + n^2$
la cui soluzione è facile vedere essere, usando il metodo dell'esperto, $\Theta(n^2)$, con un peggioramento quindi rispetto a MERGE-SORT.

**2.**

Nel caso di una lista, invece, è possibile fare il MERGE senza dovere ricorrere ad array ausiliari, né di dover spostare interi sottoarray: è sufficiente modificare un numero costante di puntatori per ottenere l'effetto di posizionare i nodi nel punto giusto durante il MERGE, come nel seguente algoritmo (la lista L va aggiunta come parametro per poter modificare l'attributo head nel momento in cui un nodo va messo in testa alla lista):

```
SUBLIST-MERGE-IN-PLACE(L, h, m, t)
1 j := m.next
2 k := h
3 while k ≠ j and j ≠ r.next
4   if k.key > j.key
5     temp := j.next
6     // ora stacchiamo il nodo j, per riattaccarlo prima di k
7     j.prev.next := j.next
8     if j.next /= NIL
9       j.next.prev := j.prev
10    if k.prev /= NIL
11      k.prev.next := j
12    else L.head := j
13    j.prev := k.prev
14    j.next := k
15    k.prev := j
```

```
15    j := temp
17  else k := k.next // k va spostato solo se j non è stato staccato e attaccato prima di esso
```

In questo caso c'è un solo ciclo, quello delle linee 3-17, ed esso viene ripetuto al più *n* volte.
Quindi, la complessità di SUBLIST-MERGE-IN-PLACE è ancora Θ(n), la ricorrenza di LIST-MERGE-SORT-IN-PLACE è la solita (LIST-MERGE-SORT-IN-PLACE è lo stesso di LIST-MERGE-SORT visto a esercitazione, salvo che la chiamata a SUBLIST-MERGE in SUBLIST-MERGE-SORT va sostituita con quella a SUBLIST-MERGE-IN-PLACE):
$T(n) = 2T(n/2) + n$
e la complessità è ancora Θ(n log(n)).

## Esercizio 3

Entrambe le affermazioni sono vere.
E' banale vedere che l'affermazione 1 è vera: l'algoritmo di TREE-INSERT scende nell'albero fino ad arrivare ad un nodo NIL, e attacca il nuovo nodo al posto del NIL, non attaccando niente altro sotto al nuovo nodo, che quindi è una foglia.

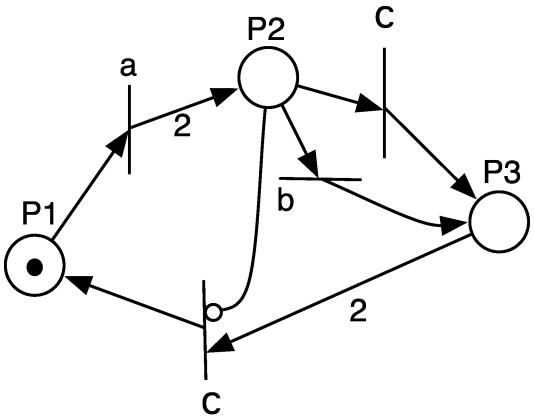
Per vedere che anche l'affermazione 2 è vera basta notare che RB-INSERT si comporta esattamente come TREE-INSERT fino a che il nuovo nodo viene attaccato come foglia; quindi viene invocato RB-INSERT-FIXUP sul nodo *z* appena inserito. Si noti che i figli di un nodo su cui viene invocato RB-INSERT-FIXUP non cambiano nei casi 1 e 3, e possono cambiare solo nel caso 2 (in cui il fratello di *z* diventa suo figlio).
Tuttavia, il caso 2 non può verificarsi per il nodo appena attaccato ad un albero da RB-INSERT. Infatti, se il caso 2 si verificasse per il nodo appena inserito, siccome il padre *x* di *z* è rosso, esso prima dell'inserimento di *z* poteva solo avere entrambi i figli uguali a T.nil, o le proprietà degli alberi RB sarebbero state violate. Tuttavia, per avere il caso 2 il fratello *y* di *x* dovrebbe essere nero, e quindi le altezze nera del sottoalbero di radice *x* e di quello di radice *y* sarebbero diverse (quella di *x* è 0, quella di *y* è almeno 1), il che di nuovo violerebbe le proprietà degli alberi RB.
Di conseguenza il nodo *z* rimarrà una foglia dell'albero anche dopo RB-INSERT-FIXUP.

# 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 $L = \{ ww \mid w \in \{a,b\}^* \}$.

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 → ε | abbcS | abccS | acbcS | 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 → 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 $\alpha$, con $\alpha$ fattore di carico; di conseguenza, in media il numero di elementi in ogni albero è anch'esso $\alpha$. 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 $\alpha$, e la loro l'altezza sarebbe $\log(\alpha)$.

# Algoritmi e Principi dell'Informatica

## Parte I – Modelli e computabilità

Appello del 6 luglio 2011

**Tempo a disposizione: 1h30'**

**Esercizio 1 (11 punti)**

Si consideri il linguaggio $L \subseteq \{a,b,c\}^*$ composto dalle stringhe di lunghezza $\geq 2$ che finiscono con 'bc', e una funzione di traduzione $\tau$ che, per ogni stringa $x \in L$, cancella i simboli 'a', raddoppia le 'b', e mantiene inalterate le 'c'. Per esempio

$\tau(abc)=bbc$, $\tau(cbabc)= cbbbbc$, $\tau(c)=\perp$, $\tau(cbab)= \perp$.

- Si scriva il trasduttore finito che calcoli la traduzione descritta.
- Si scriva l'automa a stati finiti che accetti il linguaggio $\tau(L)$; se possibile si mostri come l'automa che accetta $\tau(L)$ si possa derivare dal trasduttore definito precedentemente.
- Basandosi sulla domanda precedente, si delinei una procedura sistematica per ottenere da un trasduttore a stati finiti, l'automa che accetti il linguaggio di uscita del trasduttore stesso.

**Esercizio 2 (10 punti)**

Si consideri un sistema composto da una lampada dotata di $N>1$ bottoni per accenderla e spegnerla. La lampada viene accesa premendo simultaneamente almeno due bottoni, mentre viene spenta premendo esattamente un bottone.

Si descriva il sistema attraverso una rete di Petri oppure formule logiche, usando i seguenti predicati:

$L(t)$: la lampada è accesa al tempo $t$;

$P(b, t)$: il bottone $b$ viene premuto al tempo $t$.

**Esercizio 3 (11 punti)**

Si consideri l'insieme di macchine di Turing (MT) a nastro singolo con n stati definite su un alfabeto prefissato. Come noto, c'è un numero finito di tali MT a n stati. Iniziando l'esecuzione con il nastro con tutti blank, alcune di queste MT terminano, mentre altre non terminano.

1. Dire se è decidibile il problema di stabilire se, data una generica MT dell'insieme considerato, questa termina partendo con il nastro con tutti blank.

Si definisce S(n) il numero di passi eseguiti dalla MT che termina più tardi, tra quelle a n stati che terminano (iniziando l'esecuzione con il nastro con tutti blank).

Tale numero, noto anche come l'n-esimo numero "alacre castoro", è definito in modo univoco e, intuitivamente, rappresenta la durata dell'esecuzione della MT più "indaffarata" tra quelle che terminano. La funzione S(n) cresce molto velocemente al crescere di n.

2. Dimostrare che la funzione S(n) non è computabile.

# Algoritmi e Principi dell'Informatica

## Parte II – Complessità, algoritmi, strutture dati

Appello del 6 luglio 2011

**Tempo a disposizione: 1h30**

### Esercizio 1 (10 punti)

Si consideri una macchina RAM che simula l'esecuzione di una Macchina di Turing Nondeterministica (NTM). La RAM scrive sul nastro di output l'output risultante dall'esecuzione della NTM che termina prima, tra le tante esecuzioni possibili della NTM simulate.

1. Dire quale è la complessità spaziale della macchina RAM, sotto l'ipotesi che la complessità temporale della NTM simulate (in tutti i casi in cui termina) sia $\Theta(n)$.

2. Dire, motivando opportunamente la risposta, se l'utilizzo di un criterio di costo logaritmico per valutare la complessità (temporale e spaziale) della macchina RAM darebbe significativi vantaggi in termini della precisione della analisi di complessità.

### Esercizio 2 (11 punti)

1. Si consideri il seguente frammento di pseudocodice:

```
1 for i := 1 to n
2   p1(n)
```

dove `p1(n)` è definito nel seguente modo:

```
p1(n)
1 if n < 1
2   print(n)
3 for j := 1 to 3
4   p1(n/2)
5 for j := 1 to n
6   for k := 1 to n
7     print(j,k)
```

Dire quale è la complessità temporale del frammento di codice in funzione del parametro *n*.

2. Si consideri il seguente frammento di pseudocodice:

```
1 i := n
2 j := 1
3 while (i > 0)
4   p2(j)
5   i := i /3
6   j := j+1
```

dove `p2(n)` è definito nel seguente modo:

```
p2(n)
1 if n < 1
2   print(n)
3 for l := 1 to 4
4   p2(n/2)
5 for k := 1 to n
6   for j := k+1 to n
7     print(k,j)
```

Dire quale è la complessità temporale del frammento di codice in funzione del parametro *n*.

3. Sia dia la complessità del seguente algoritmo:

```
p3(n)
1 if n < 1
2   print(n)
3 for i := 1 to 9
4   p3(n/3)
5 p2(n)
```

Dove `p2(n)` è definito come al punto 2.

**NB:** Tutte le variabili nei frammenti di codice precedenti sono da intendersi di tipo intero, quindi anche le operazioni su di esse sono da intendersi come operazioni sugli interi.

## Esercizio 3 (11 punti)

Si vogliono descrivere le relazioni di parentela tra diverse persone viventi. Più precisamente, si vogliono descrivere le relazioni "è figlio di", "è padre di", "è madre di", "è marito di", "è moglie di". Non è detto che ogni persona abbia tutte le relazioni; per esempio, un orfano non ha i genitori (si ricordi che interessano solo le relazioni tra persone viventi); inoltre, non tutte le persone hanno figli o sono sposate. Ogni persona può avere più figli, ma può essere sposata con al massimo un'altra persona.

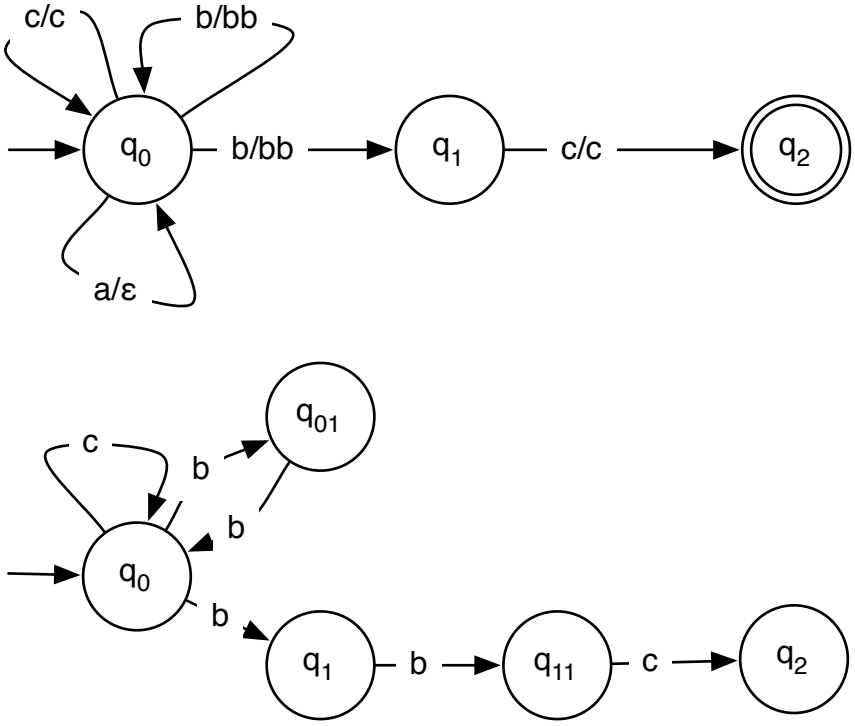
1. Si descriva un'opportuna struttura dati per rappresentare le relazioni descritte sopra.

In un caso anomalo, ma non impossibile, è possibile che una persona sia "il proprio nonno"; il caso accade quando un uomo: (i) sposa una signora che ha una figlia; (ii) la figlia sposa il padre dell'uomo.

2. Scrivere un algoritmo in pseudocodice che determina se esiste una persona tra quelle rappresentate nella relazione che è il proprio nonno; dare la complessità dell'algoritmo ideato.

**Soluzioni – Parte I**

**Esercizio 1**




Costruzione: in pratica basta "buttar via" la parte di input, mentre si trasformano in transizioni le stringhe di output: se una stringa è vuota, si perde la transizione; se è fatta da più di un carattere si mette una catena di stati con una "maglia" per ogni carattere.

**Esercizio 2**

Versione a tempo discreto
$\exists b_1 \exists b_2 (b_1 \neq b2 \wedge P(b_1, t) \wedge P(b_2, t)) \Rightarrow L(t+1)$
$\exists b_1 (P(b_1,t) \wedge \neg \exists b_2 (b_1 \neq b_2 \wedge P(b_2, t)) \Rightarrow \neg L(t+1)$
$\neg \exists b\, P(b, t) \Rightarrow (L(t) \Leftrightarrow L(t+1))$

**Esercizio 3**

1.
Il problema di stabilire se, data una generica MT, questa termina partendo con il nastro con tutti blank è indecidibile. Questo si può vedere ad esempio usando il teorema di Rice. Infatti, l'insieme delle MT che terminano come descritto non è né l'insieme vuoto, né l'insieme di tutte le funzioni calcolabili.

2.
Per dimostrare che la funzione S(n) non è computabile procediamo per assurdo.

Supponiamo di essere in grado di calcolare S(n) per ogni n. Allora possiamo decidere se una qualunque MT con n stati termina iniziando la sua esecuzione con il nastro d'ingresso con tutti blank. Infatti basta mettere in esecuzione la macchina: se si ferma, sappiamo che termina; se non si ferma entro S(n) passi, allora sappiamo che non terminerà mai, poiché S(n) è il numero massimo di passi eseguibili da una MT che termina.

Ma stabilire se una MT termina iniziando la sua esecuzione con il nastro d'ingresso vuoto è indecidibile, come dimostrato in precedenza. Assurdo.

**Soluzioni – Parte II**

**Esercizio 1**

1. La macchina RAM, per simulare la NTM, deve esplorare l'albero delle computazioni nondeterministiche della MT effettuando una visita in ampiezza dell'albero (in cui ogni nodo corrisponde ad una configurazione $\langle q, \{\langle p_i, T_i\rangle\}_i\rangle$, con $p_i$ la posizione della testina sul nastro i-esimo, e $T_i$ il contenuto del nastro i-esimo, codificato per esempio come un numero binario se l'alfabeto della NTM è $\{0,1\}$). Data una profondità *n*, il numero di nodi in tutti i livelli fino ad *n* è al massimo $\Theta(2^n)$. Nel caso pessimo tutti questi nodi vanno memorizzati (per esempio se la computazione che termina è l'ultima in basso a destra nell'albero), quindi la complessità spaziale (a costo costante) della RAM è $\Theta(2^n)$.

2. Se si usasse il criterio di costo logaritmico, il costo di memorizzare una singola configurazione è, nel caso pessimo, $\Theta(\log(n))$, in quanto il contenuto di ogni nastro è lungo al massimo *n* (per ipotesi sulla complessità della NTM). Di conseguenza la complessità spaziale sarebbe $\Theta(\ 2^n \log(n)\ )$. La complessità temporale, invece, sarebbe come minimo $\Theta(\ 2^n \log(2^n)) = \Theta(n2^n)$ in quanto l'indirizzo più grande a cui la macchina RAM dovrebbe accedere sarebbe appunto $2^n$. Usando il criterio di costo logaritmico l'analisi sarebbe quindi più precisa, ma il termine dominante (sia pure con un fatto moltiplicativo davanti) sarebbe comunque $2^n$.

**Esercizio 2**

1. `p1` è un algoritmo ricorsivo la cui ricorrenza è $T(n) = 3T(n/2) + n^2$, la cui soluzione è, applicando il Master theorem, $T(n) = \Theta(n^2)$. Di conseguenza la complessità globale del frammento di pseudocodice è $\Theta(n*n^2) = \Theta(n^3)$, in quanto `p1` viene invocata *n* volte con parametro *n*.

2. In questo secondo caso la ricorrenza di `p2` è $T(n) = 4T(n/2) + n^2$, che ha soluzione (sempre applicando il Master theorem) $T(n) = \Theta(n^2\log(n))$. `p2` viene invocata un numero $\log_3(n)$ volte con valori crescenti del parametro, quindi il costo totale del frammento di pseudocodice è $\sum_{j=1}^{\log(n)} j^2 \log(j)$, che è $\Theta(n^2\log^2(n))$ (si può vedere che $\sum_{j=1}^{k} j^2 \log(j)$ è $\leq n^2\log^2(n)$ , ed inoltre che è $\geq (n/2)^2\log^2(n/2)$).

3. In questo caso la ricorrenza di `p3` è $T(n) = 9T(n/3) + \Theta(n^2\log(n))$, che non può essere risolta con il Master Theorem ($n^2\log(n)$ non è polinomialmente più grande di $n^2$. Tuttavia, tramite il metodo di sostituzione, è possibile mostrare che in questo caso $T(n) = \Theta(n^2\log^2(n))$: è facile vedere che $T(n) \geq c\ n^2\log^2(n)$, in quanto $T(n) = 9T(n/3) + dn^2\log(n) \geq 9c(n/3)^2\log^2(n/3) + dn^2\log(n) \geq c\ n^2\log^2(n)$; si può inoltre mostrare che esiste un $d_1$ tale che $T(n) \leq c\ n^2\log^2(n) - d_1\ n^2\log(n)$, cioè che $T(n) = O(n^2\log^2(n))$.

**Esercizio 3**

1. Una maniera possibile per rappresentare le relazioni desiderate è tramite una struttura a puntatori (di fatto un grafo) in cui ogni nodo N ha gli attributi N.p per indicare il padre, N.m per indicare il nodo corrispondente alla madre, N.h (per "husband") per indicare il marito, N.w (per "wife") per indicare la moglie, e un array N.f per indicare i figli. Se qualche relazione è mancante, il corrispondente attributo avrà un puntatore NIL.

2. Data una struttura dati *R* fatta come descritto al punto 1, con un attributo *R.N* che è un array con i nodi della struttura, un algoritmo possibile è il seguente:

```
MyOwnGrandPa(R)
1  for each u ∈ R.N
2    if u.p ≠ NIL and u.p.w ≠ NIL and u.p.w.m ≠ NIL
        and u.p.w.m.h ≠ NIL and u.p.w.m.h = u
3      return true
```

L'algoritmo ha complessità $\Theta(n)$, in quanto ogni iterazione del ciclo ha costo costante, ed il ciclo viene eseguito al massimo n volte (con n numero delle persone rappresentate nella relazione, cioè numero dei nodi in R).

# Algoritmi e Principi dell'Informatica

## Parte I – Modelli e computabilità

Appello dell'8 settembre 2011

**Tempo a disposizione: 1h30'**

### Esercizio 1 (12 punti)

Si consideri il linguaggio L fatto di tutte e sole le stringhe $x \in \{a, b\}^*$ in cui *ogni prefisso y* di *x* è tale che $\#b(y) \geq \#a(y)$, laddove con #a(y) e #b(y) si indica, rispettivamente, il numero di *a* e di *b* nella stringa *y*.

1. Si scriva una grammatica che generi L. La grammatica scritta è a potenza minima tra quelle che generano L? Se no, quale è la classe di grammatiche a potenza minima tra quelle che generano L?
2. Si scriva una rete di Petri che riconosca L. La RdP scritta è a potenza minima tra quelle che riconoscono L?

### Esercizio 2 (10 punti)

Si consideri la funzione (parziale) $car_x : N \rightarrow \{s, e, b\}$ che descrive i caratteri di una stringa *x*. Più precisamente, $car_x(i)$ restituisce il carattere in posizione i-esima della stringa *x* (ed è indefinito se *x* non ha carattere in posizione i-esima).
Si consideri la seguente specifica logica di un linguaggio L (le stringhe del linguaggio L, cioè, sono tutte e sole quelle che soddisfano le condizioni scritte sotto):

$$
\begin{aligned}
\exists\, n \,(\quad & n > 0 \\
& \forall j\, (j \geq n \Rightarrow car_x(j) = \perp) \wedge \\
& \forall j\, (0 \leq j < n \Rightarrow car_x(j) \neq \perp) \wedge \\
& car_x(0) \neq e \wedge \\
& \forall j\, (\quad car_x(j) = e \wedge car_x(j+1) = e \wedge \\
& \qquad car_x(j-1) \neq e \wedge car_x(j+2) \neq e \\
& \qquad \Rightarrow \\
& \qquad car_x(j-1) = s\ ))
\end{aligned}
$$

1. Descrivere a parole come è fatto L, e dare almeno 2 esempi di stringhe che appartengono ad L e almeno 2 esempi di stringhe che non appartengono ad L. Gli esempi devono essere tutti significativi, nel senso che devono soddisfare o violare la specifica in modi diversi.
2. Scrivere un automa che riconosce L. L'automa deve essere a potenza minima tra quelli che riconoscono L.

### Esercizio 3 (11 punti)

Il `datalog` è un linguaggio di interrogazione per basi di dati. Data una query `datalog` Q, si indica con Q(D) l'insieme di risposte a Q ottenute sul database D.
Si parta dalle seguenti definizioni e fatti noti su `datalog`:

- Si dice che Q1 è *contenuta* in Q2 se $Q1(D) \subseteq Q2(D)$ per ogni database D.
- Due query datalog Q1 e Q2 si dicono invece *equivalenti* se $Q1(D) \subseteq Q2(D)$ e $Q2(D) \subseteq Q1(D)$ per ogni database D.
- Il problema di determinare se due generiche query `datalog` siano equivalenti è indecidibile.

A partire da questi fatti (e solo da questi), è possibile concludere la decidibilità o indecidibilità dei seguenti problemi?

1. Determinare se una generica query `datalog` è contenuta in un'altra generica query `datalog`.
2. Determinare se una generica query `datalog` è equivalente a una query `datalog` prefissata.
3. Determinare se una query `datalog` prefissata è equivalente a un'altra query `datalog` prefissata.

**NB:** per questo esercizio non è importante conoscere la struttura delle query `datalog`, né il meccanismo di calcolo delle risposte alle query.

# Algoritmi e Principi dell'Informatica

## Parte II – Complessità, algoritmi, strutture dati

Appello dell'8 settembre 2011

**Tempo a disposizione: 1h30**

### Esercizio 1 (12 punti)

Si consideri la seguente funzione f(x) definita su stringhe $x \in \{a, b\}^*$:

f(x) = if #a(x)≥#b(x) then a else b

dove #a(x) e #b(x) denotano rispettivamente il numero di *a* ed il numero di *b* nella stringa x. Per esempio, f(aabab)=a, f(abbab)=b, f(baabba)=a, f(ε)=a.

1. Si definisca in dettaglio una macchina di Turing (la più semplice, indipendentemente dalla sua complessità computazionale) che calcola la funzione f.
2. Si delinei una macchina RAM che calcola f (si assuma che la macchina RAM possa gestire caratteri, oltre che numeri).
3. Si valutino le funzioni $T_M(|x|)$, $S_M(|x|)$, $T_{RAM}(|x|)$ e $S_{RAM}(|x|)$, cioè le complessità temporali e spaziali della macchina di Turing e della macchina RAM in funzione della lunghezza della stringa *x* (per la macchina RAM si adotti il criterio di costo uniforme). Si confrontino le due funzioni di complessità e le rispettive classi di complessità $\Theta(T_M)$ e $\Theta(T_{RAM})$: quale è la migliore? E per quel che riguarda $\Theta(S_M)$ e $\Theta(S_{RAM})$?
4. Cosa cambia nella risposta al punto 3 se per la valutazione di $T_{RAM}(|x|)$ e $S_{RAM}(|x|)$ viene usato il criterio di costo logaritmico?

### Esercizio 2 (12 punti)

1. Si consideri il seguente algoritmo `p(n)`:

```
p(n)
1 if n < 2
2   return 1
3 return p(n/2) + p(n/2) + p(n/2)
```

   a. Quale è la complessità temporale dell'algoritmo `p` in funzione del parametro *n*?

   b. Cambierebbe il risultato prodotto dall'algoritmo se la terza istruzione fosse
   `return 3*p(n/2)`
   invece di quella scritta sopra?
   Cambierebbe la complessità?

2. Si consideri il seguente frammento di pseudocodice:

```
1 for i := 1 to n
2   j := i+1
3   while (j / n <= n)
4     k := 1
5     while (k <= n)
6       p(4)
7       k := k + 3
8     j := j +1
```

laddove `p` è l'algoritmo definito al punto 1.

Dire quale è la complessità temporale del frammento di codice in funzione del parametro *n*.

3. Si consideri il seguente frammento di pseudocodice:

```
1 i := n
2 while (i > 0)
3   p2(n)
4   i := i/2
```

dove `p2(n)` è definito nel seguente modo:

```
p2(n)
1 if n < 1
2   print(n)
3 else p2(n-1)
```

Dire quale è la complessità temporale del frammento di codice in funzione del parametro *n*.

**NB:** Tutte le variabili nei frammenti di codice precedenti sono da intendersi di tipo intero, quindi anche le operazioni su di esse sono da intendersi come operazioni sugli interi.

## Esercizio 3 (8 punti)

Si consideri il problema di ritornare, dato in ingresso un albero binario *T* contenente numeri interi, la radice del sottoalbero *S* di *T* la cui somma degli elementi è massima (tale cioè per cui non esiste nessun altro sottoalbero di *T* la cui somma dei valori è maggiore di quella degli elementi di *S*).

Scrivere in pesudocodice un algoritmo che risolve il problema desiderato, e darne la complessità.

**Soluzioni – Parte I**

**Esercizio 1**

1. Una grammatica a potenza minima che genera il linguaggio desiderato è la seguente:

$S \rightarrow HBS \mid \varepsilon$

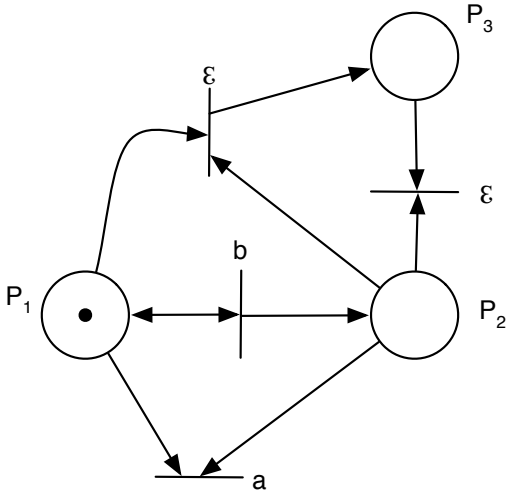
$B \rightarrow Bb \mid b$

$H \rightarrow bHa \mid HH \mid \varepsilon$

Una grammatica alternativa (sempre a potenza minima) e con meno produzioni, è la seguente:

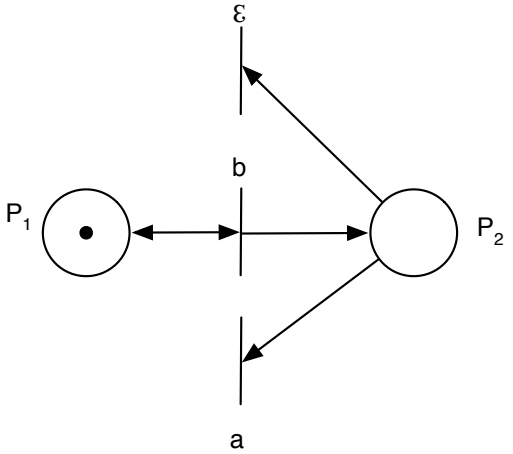
$S \rightarrow bSaS \mid bB \mid \varepsilon$

2. Una rete di Petri che riconosce il linguaggio desiderato è la seguente:




In cui le marcature finali sono 2, $M_1$, ed $M_3$, definite nel seguente modo: $M_1(P_1) = 1$, $M_1(P_2) = M_1(P_3) = 0$ e $M_3(P_3) = 1$, $M_3(P_2) = M_3(P_1) = 0$.

In alternativa, un'altra rete di Petri, più compatta, che risolve il problema è la seguente:




In questo caso la marcatura finale è una sola, e coincide con quella iniziale.

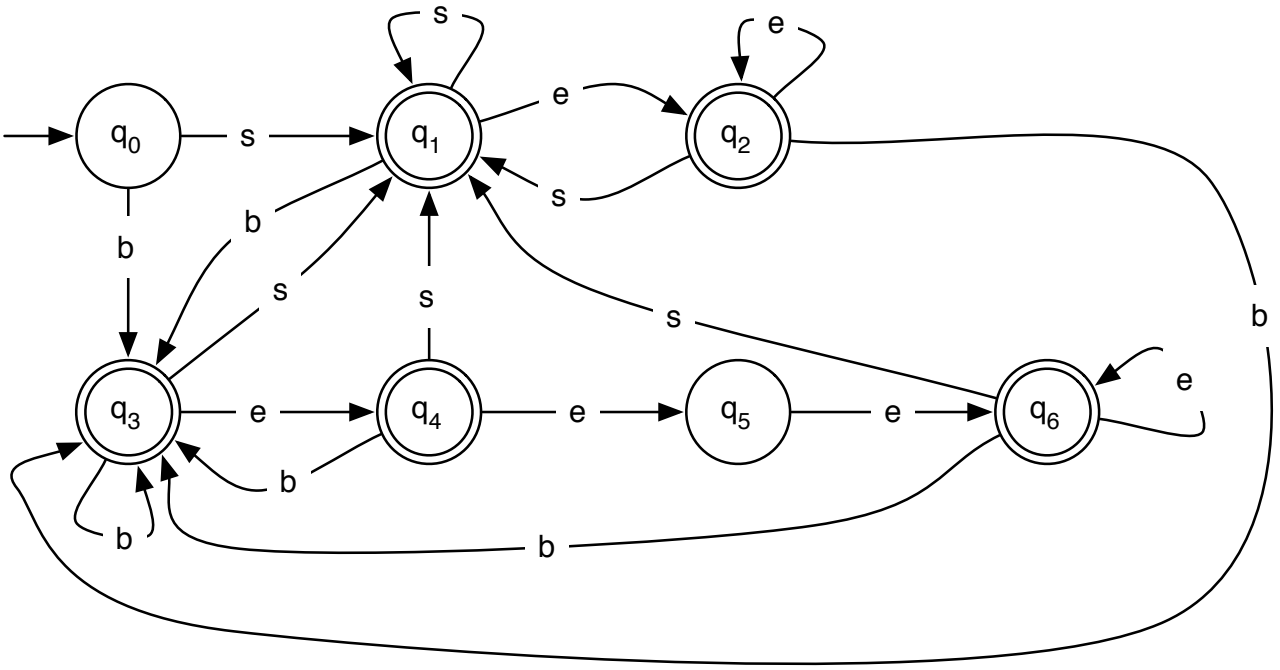
**Esercizio 2**

Il linguaggio specificato è fatto di tutte e sole le stringhe di almeno 1 carattere che non iniziano per *e*, e tali che ogni volta che si incontrano esattamente 2 *e* consecutive (cioè 2 e precedute e seguite da un simbolo diverso da *e*, eventualmente da $\perp$) il simbolo precedente le *e* è una *s*.

2 stringhe che appartengono: bbbbsee, sebbbeeebessseeb

2 stringhe che non appartegono: essss, bsbbeebb

Un automa a stati finiti che riconosce il linguaggio L è il seguente:




**Esercizio 3**

Il problema è indecidibile. Infatti, se fosse possibile decidere il contenimento tra due generiche query Q1 e Q2 si potrebbe immediatamente decidere anche la loro equivalenza, il che è assurdo. A tal fine si noti che, poiché Q1 è equivalente a Q2 se e solo se Q1 è contenuta in Q2 e Q2 è contenuta in Q1, c'è una riduzione dal problema dell'equivalenza (che è noto essere indecidibile) a quello del contenimento (che è quindi più generale e pertanto indecidibile).

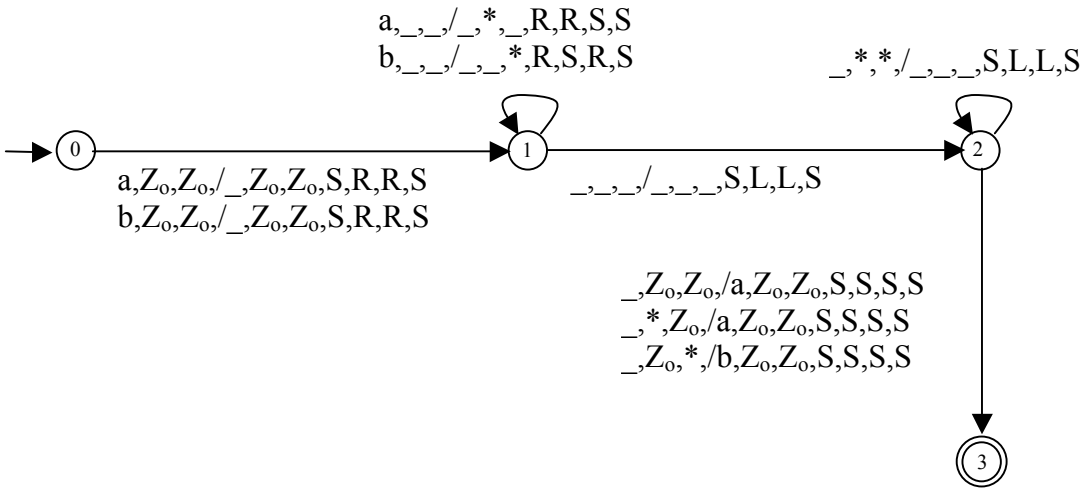
2. In assenza di altre informazioni non è possibile concludere nulla sulla decidibilità di questo problema. Si tratta infatti di un caso specifico di un problema indecidibile (quello dell'equivalenza tra due query generiche).

3. In questo caso il problema è banalmente decidibile in quanto si tratta di una domanda con risposta chiusa sì/no che non dipende da alcun ingresso.

**Soluzioni – Parte II**

**Esercizio 1**

1. Una macchina di Turing (a 2 nastri) che calcola la funzione desiderata è la seguente:




2. La macchina RAM legge semplicemente tutti i simboli in input, conta il numero di *a* e *b* in 2 contatori *na* e *nb*, confronta *na* con *nb* ed infine scrive *a* se *na*≥*nb*, *b* altrimenti.
3. La MT percorre l'autoanello sullo stato 1 |x| volte, e quello sullo stato 2 min(#a(x),#b(x)) volte; il caso pessimo è quello in cui #a(x)=#b(x); in questo caso la complessità è $T_M(|x|)=3+(3/2)|x|$; $T_{RAM}(|x|)=k+|x|$ per una (piccola) costante k. $T_M$ è $\Theta(n)$, idem per $T_{RAM}$. Quindi la complessità $T_{RAM}$ è migliore di $T_M$, ma la classe $\Theta$ è la stessa.
   Per quel che riguarda la complessità **spaziale**, $S_M(|x|)=|x|$, e $S_{RAM}(|x|)=2$,quindi la complessità della macchina RAM è migliore, anche considerando le classi $\Theta$.
4. Poiché il valore dei contatori *na* e *nb* può crescere indefinitamente, i costi delle operazioni di incremento e del confronto finale sono moltiplicati di un fattore log |x|, quindi $T_{RAM}$ è $\Theta(|x| \log |x|)$: la classe di complessità $\Theta(T_{RAM})$ è quindi peggiore di $\Theta(T_M)$. Inoltre, $T_{RAM}(|x|)$ può essere peggiore di $T_M(|x|)$, ma ciò accade per valori abbastanza grandi di |x|.
   Per quel che riguarda la complessità **spaziale**, $S_{RAM}(|x|)=2\cdot\log(|x|)$, quindi la complessità spaziale della macchina RAM è ancora migliore, anche considerando le classi $\Theta$.

**Esercizio 2**

1a. `p` è un algoritmo ricorsivo la cui ricorrenza è $T(n) = 3T(n/2)+ \Theta(1)$, la cui soluzione è, applicando il Master theorem, $T(n) = \Theta(n^{\log_2 3})$.

1b. In questo secondo caso la ricorrenza diventa $T(n) = T(n/2)+ \Theta(1)$, in quanto vi è una sola chiamata ricorsiva; la soluzione della ricorrenza è quindi, applicando il Master theorem, $T(n) = \Theta(\log(n))$.

2. Poiché la chiamata all'algoritmo `p` viene sempre fatta con parametro costante, il suo costo è altrettanto costante. Il costo del frammento di codice dipende quindi solo dai cicli. Quello più interno viene eseguito sempre n/3 volte. Quello intermedio viene eseguito ogni volta $(n-i)^2$ volte, per *i* che va da 1 a n. Il costo totale del frammento è quindi $\sum_{i=1}^{n} i^2 \frac{n}{3}$, che è $\Theta(n^4)$.

3. In questo caso la ricorrenza di `p2` è $T(n) = T(n-1) + \Theta(1)$, che non può essere risolta con il Master Theorem. Tuttavia, come visto a lezione, è noto che questa ricorrenza ha soluzione $\Theta(n)$, come si può anche calcolare notando che vengono fatte *n* chiamate ricorsive, ed ognuna ha costo costante. `p2` viene invocato $log(n)$ volte (tante quante sono le iterazioni del ciclo), sempre con parametro *n*, quindi il costo totale del frammento di codice è $\Theta(n \log(n))$.

**Esercizio 3**
Un algoritmo che risolve il problema desiderato è il seguente:

```
MaxSubtree(T)
1  [S, sumS, sumT] = MaxSubtreeWalk(T)
2  return S
```

Dove `MaxSubtreeWalk` è il seguente (dove `-inf` è un valore convenzionale che indica un numero più piccolo di tutti quelli possibili):

```
MaxSubtreeWalk(S)
1  if S = NIL
2    return S,-inf,-inf
3  else
4    [SL, sumSL, sumL] = MaxSubtreeWalk(S.left)
5    [SR, sumSR, sumR] = MaxSubtreeWalk(S.right)
6    sumS := sumSL+sumSR+S.key
7    if sumS >= sumSL and sumS >= sumSR
8      return S, sumS, sumS
9    else if sumSL >= sumS and sumSL >= sumSR
10     return SL, sumSL, sumS
11   else return SR, sumSR, sumS
```

L'algoritmo è essenzialmente un post-order tree walk, e la sua complessità è $\Theta(n)$.

# Algoritmi e Principi dell'Informatica
## Semiunità di Informatica Teorica
Appello dell'8 settembre 2011

**Tempo a disposizione: 2h**

### Esercizio 1 (10 punti)
Si consideri il linguaggio L fatto di tutte e sole le stringhe $x \in \{a, b\}^*$ in cui *ogni prefisso y* di *x* è tale che $\#b(y) \geq \#a(y)$, laddove con #a(y) e #b(y) si indica, rispettivamente, il numero di *a* e di *b* nella stringa *y*.

1. Si scriva un automa che generi L. L'automa scritto è a potenza minima tra quelli che generano L? Se no, quale è la classe di automi a potenza minima tra quelli che generano L?
2. Si scriva una grammatica che generi L. La grammatica scritta è a potenza minima tra quelle che generano L? Se no, quale è la classe di grammatiche a potenza minima tra quelle che generano L?

### Esercizio 2 (5 punti)
Si consideri la funzione (parziale) $car_x : N \rightarrow \{s, e, b\}$ che descrive i caratteri di una stringa *x*. Più precisamente, $car_x(i)$ restituisce il carattere in posizione i-esima della stringa *x* (ed è indefinito se *x* non ha carattere in posizione i-esima).
Si consideri la seguente specifica logica di un linguaggio L (le stringhe del linguaggio L, cioè, sono tutte e sole quelle che soddisfano le condizioni scritte sotto):

$$
\begin{aligned}
\exists\, n \,(\quad & n > 0 \\
& \forall j\, (j \geq n \Rightarrow car_x(j) = \perp) \wedge \\
& \forall j\, (0 \leq j < n \Rightarrow car_x(j) \neq \perp) \wedge \\
& car_x(0) \neq e \wedge \\
& \forall j\, (\quad car_x(j) = e \wedge car_x(j+1) = e \wedge \\
& \qquad car_x(j-1) \neq e \wedge car_x(j+2) \neq e \\
& \qquad \Rightarrow \\
& \qquad car_x(j-1) = s\ ))
\end{aligned}
$$

Descrivere a parole come è fatto L, e dare almeno 2 esempi di stringhe che appartengono ad L e almeno 2 esempi di stringhe che non appartengono ad L. Gli esempi devono essere tutti significativi, nel senso che devono soddisfare o violare la specifica in modi diversi.

### Esercizio 3 (9 punti)

Il `datalog` è un linguaggio di interrogazione per basi di dati. Data una query `datalog` Q, si indica con Q(D) l'insieme di risposte a Q ottenute sul database D.
Si parta dalle seguenti definizioni e fatti noti su `datalog`:

- Si dice che Q1 è *contenuta* in Q2 se $Q1(D) \subseteq Q2(D)$ per ogni database D.
- Due query datalog Q1 e Q2 si dicono invece *equivalenti* se $Q1(D) \subseteq Q2(D)$ e $Q2(D) \subseteq Q1(D)$ per ogni database D.
- Il problema di determinare se due generiche query `datalog` siano equivalenti è indecidibile.

A partire da questi fatti (e solo da questi), è possibile concludere la decidibilità o indecidibilità dei seguenti problemi?

1. Determinare se una generica query `datalog` è contenuta in un'altra generica query `datalog`.
2. Determinare se una generica query `datalog` è equivalente a una query `datalog` prefissata.
3. Determinare se una query `datalog` prefissata è equivalente a un'altra query `datalog` prefissata.

**NB:** per questo esercizio non è importante conoscere la struttura delle query `datalog`, né il meccanismo di calcolo delle risposte alle query.

**Esercizio 4 (9 punti)**

Si consideri la seguente funzione f(x) definita su stringhe $x \in \{a, b\}^*$:

$f(x) = \text{if } \#a(x) \geq \#b(x) \text{ then a else b}$

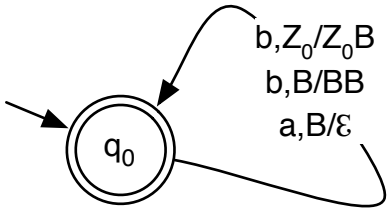
dove #a(x) e #b(x) denotano rispettivamente il numero di *a* ed il numero di *b* nella stringa x. Per esempio, f(aabab)=a, f(abbab)=b, f(baabba)=a, f(ε)=a.

1. Si definisca in dettaglio una macchina di Turing (la più semplice, indipendentemente dalla sua complessità computazionale) che calcola la funzione f.
2. Si delinei una macchina RAM che calcola f (si assuma che la macchina RAM possa gestire caratteri, oltre che numeri).
3. Si valutino le funzioni $T_M(|x|)$, $S_M(|x|)$, $T_{RAM}(|x|)$ e $S_{RAM}(|x|)$, cioè le complessità temporali e spaziali della macchina di Turing e della macchina RAM in funzione della lunghezza della stringa *x* (per la macchina RAM si adotti il criterio di costo uniforme). Si confrontino le due funzioni di complessità e le rispettive classi di complessità $\Theta(T_M)$ e $\Theta(T_{RAM})$: quale è la migliore? E per quel che riguarda $\Theta(S_M)$ e $\Theta(S_{RAM})$?
4. Cosa cambia nella risposta al punto 3 se per la valutazione di $T_{RAM}(|x|)$ e $S_{RAM}(|x|)$ viene usato il criterio di costo logaritmico?

**Soluzioni**

**Esercizio 1**

1. Un automa (a potenza minima) che riconosce il linguaggio desiderato è il seguente:




2. Una grammatica a potenza minima che genera il linguaggio desiderato è la seguente:
S → HBS | ε
B → Bb | b
H → bHa | HH | ε

Una grammatica alternativa (sempre a potenza minima) e con meno produzioni, è la seguente:
S → bSaS | bB | ε

**Esercizio 2**

Il linguaggio specificato è fatto di tutte e sole le stringhe di almeno 1 carattere che non iniziano per *e*, e tali che ogni volta che si incontrano esattamente 2 *e* consecutive (cioè 2 e precedute e seguite da un simbolo diverso da *e*, eventualmente da ⊥) il simbolo precedente le *e* è una *s*.

2 stringhe che appartengono: bbbbsee, sebbbeeebessseeb
2 stringhe che non appartegono: essss, bsbbeebb

**Esercizio 3**

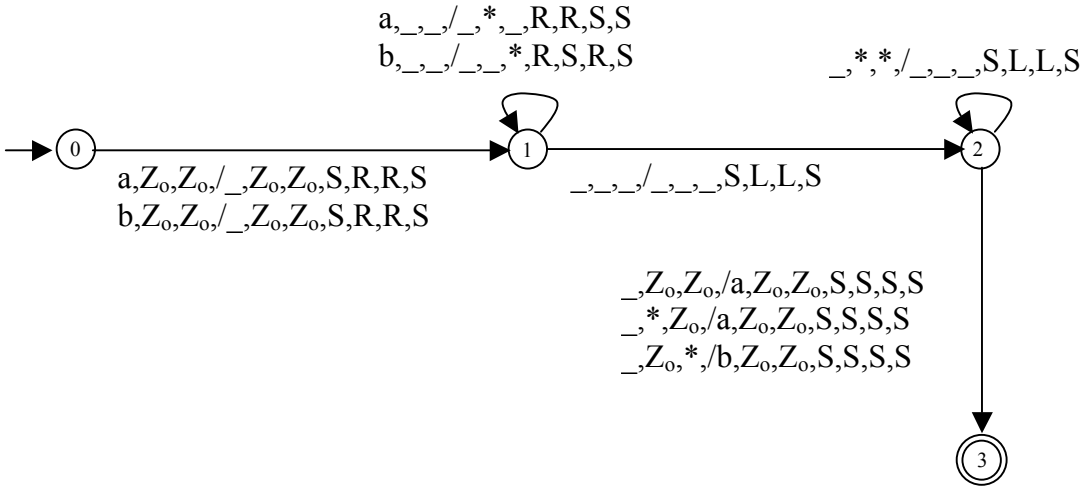
Il problema è indecidibile. Infatti, se fosse possibile decidere il contenimento tra due generiche query Q1 e Q2 si potrebbe immediatamente decidere anche la loro equivalenza, il che è assurdo. A tal fine si noti che, poiché Q1 è equivalente a Q2 se e solo se Q1 è contenuta in Q2 e Q2 è contenuta in Q1, c'è una riduzione dal problema dell'equivalenza (che è noto essere indecidibile) a quello del contenimento (che è quindi più generale e pertanto indecidibile).

2. In assenza di altre informazioni non è possibile concludere nulla sulla decidibilità di questo problema. Si tratta infatti di un caso specifico di un problema indecidibile (quello dell'equivalenza tra due query generiche).

3. In questo caso il problema è banalmente decidibile in quanto si tratta di una domanda con risposta chiusa sì/no che non dipende da alcun ingresso.

**Esercizio 4**

1. Una macchina di Turing (a 2 nastri) che calcola la funzione desiderata è la seguente:




2. La macchina RAM legge semplicemente tutti i simboli in input, conta il numero di *a* e *b* in 2 contatori *na* e *nb*, confronta *na* con *nb* ed infine scrive *a* se *na*≥*nb*, *b* altrimenti.
3. La MT percorre l'autoanello sullo stato 1 |x| volte, e quello sullo stato 2 min(#a(x),#b(x)) volte; il caso pessimo è quello in cui #a(x)=#b(x); in questo caso la complessità è $T_M(|x|)=3+(3/2)|x|$; $T_{RAM}(|x|)=k+|x|$ per una (piccola) costante k. $T_M$ è $\Theta(n)$, idem per $T_{RAM}$. Quindi la complessità $T_{RAM}$ è migliore di $T_M$, ma la classe $\Theta$ è la stessa.
   Per quel che riguarda la complessità **spaziale**, $S_M(|x|)=|x|$, e $S_{RAM}(|x|)=2$,quindi la complessità della macchina RAM è migliore, anche considerando le classi $\Theta$.
4. Poiché il valore dei contatori *na* e *nb* può crescere indefinitamente, i costi delle operazioni di incremento e del confronto finale sono moltiplicati di un fattore log |x|, quindi $T_{RAM}$ è $\Theta(|x| \log |x|)$: la classe di complessità $\Theta(T_{RAM})$ è quindi peggiore di $\Theta(T_M)$. Inoltre, $T_{RAM}(|x|)$ può essere peggiore di $T_M(|x|)$, ma ciò accade per valori abbastanza grandi di |x|.
   Per quel che riguarda la complessità **spaziale**, $S_{RAM}(|x|)=2\cdot\log(|x|)$, quindi la complessità spaziale della macchina RAM è ancora migliore, anche considerando le classi $\Theta$.