Algoritmi e Principi dell'Informatica

Prima prova in itinere - 27 novembre 2015

Tempo a disposizione: 1h30

Esercizio 1 (7 punti)

1. Scrivere un automa che riconosca il linguaggio L fatto di tutte e sole le stringhe della forma $\{a,b,c\}^*a^n\{a,b,c\}^*b^n\{a,b,c\}^*c^n\{a,b,c\}^*$, con $n \geq 1$. L'automa deve essere a potenza minima tra quelli che riconoscono L.

2. Qual è l'automa a potenza minima (tra quelli visti a lezione) che riconosce il linguaggio L' fatto di tutte e sole le stringhe della forma $\{b,c\}^*a^nc^*b^na^*c^n\{a,b\}^*$, con $n \geq 1$?

Esercizio 2 (5 punti)

Scrivere una formula MFO o MSO che definisca il linguaggio fatto di tutte e sole le stringhe in cui compaiono esattamente 2 *b* in seconda e penultima posizione della stringa.

Esercizio 3 (5 punti)

Dire, motivando opportunamente le risposte, se sono decidibili i seguenti problemi:

1. Data una qualunque macchina di Turing M, essa riconosce il linguaggio L definito nell'esercizio 1.1?

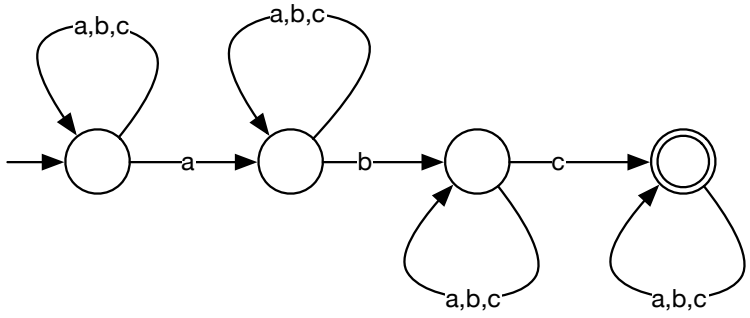
2. Data una qualunque macchina di Turing M, essa riconosce il linguaggio L' definito nell'esercizio 1.2?

3. Dato un qualunque automa a stati finiti A, esso riconosce il linguaggio L' definito nell'esercizio 1.2?

Soluzioni

Esercizio 1

1.
Il linguaggio L è riconoscibile da un FSA, in quanto si tratta solo di verificare che nella stringa *x* in input ci sia almeno una *a*, seguita (non necessariamente subito) da almeno una b, seguita (non necessariamente subito) da almeno una *c*.



2.
Per riconoscere il linguaggio L', invece, occorre contare la lunghezza della prima sequenza di *a*, della successiva sequenza di *b*, e della successiva ancora sequenza di *c*, e verificare che esse sono uguali. Per questo, il formalismo a potenza minima tra quelli visti a lezione che è in grado di riconoscere il linguaggio L' sono le reti di Petri senza archi inibitori.

Esercizio 2

$$\exists x, y \, (\; x < y \wedge \mathrm{b}(x) \wedge \mathrm{b}(y) \wedge \forall z \, (z \neq x \wedge z \neq y \rightarrow \neg \mathrm{b}(z)) \wedge x = 1 \wedge \exists u \, (u = y + 1 \wedge \mathrm{last}(u)) \;)$$

Esercizio 3

1.
Il problema è indecidibile per il teorema di Rice, in quanto l'insieme delle MT che riconoscono il linguaggio L non è l'insieme vuoto (il linguaggio è riconoscibile da una MT), e non è l'insieme universo (per esempio una MT che riconosce il linguaggio vuoto non riconosce L).

2.
Analogamente al punto 1, il problema è indecidibile per il teorema di Rice.

3.
Il linguaggio L' non è riconoscibile da nessun FSA, quindi il problema è banalmente decidibile (e deciso), in quanto, dato un qualunque FSA A, la risposta è che esso non riconosce L'.

Algoritmi e Principi dell'Informatica

Seconda prova in itinere - 1 febbraio 2016

Tempo a disposizione: 1h45

Esercizio 1 (7 punti)

a.
Si descriva a grandi linee, ma in modo sufficientemente preciso per calcolarne la complessità, una macchina di Turing (deterministica) a k nastri che riconosca il linguaggio $L = \{a^{2^n} | \, n \geq 0\}$.
Dire quali sono, nel caso pessimo, le complessità temporale e spaziale della MT ideata.

b.
Si descriva a grandi linee una macchina RAM che riconosce il linguaggio *L* del punto a.
Darne le complessità temporale e spaziale, calcolate sia a criterio di costo costante che a criterio di costo logaritmico.

Esercizio 2 (5 punti)

Si calcoli la complessità del seguente frammento di codice:

```
PROC(n)
  if n < 5
    return 0
  s := PROC(2*n/5)
  s := s+PROC(3*n/5)
  i := 1
  while i ≤ 3*n
    s := s+1
    i := i+2
  return s
```

Esercizio 3 (6 punti)

Si vuole aumentare un albero red-black aggiungendo, ad ogni nodo *x* dell'albero, un campo *maxval* che tenga traccia del valore massimo presente nel sottoalbero di radice *x*.
Dire come devono essere modificate le operazioni di `INSERT` e `DELETE` per mantenere i campi *maxval* aggiornati correttamente.
Dire qual è la complessità temporale degli algoritmi modificati.

Soluzioni

Esercizio 1

a.

Per riconoscere il linguaggio desiderato, è sufficiente una MT con 2 nastri di memoria. la macchina scandisce la sequenza di *a* in input, e per ogni *a* letta incrementa un contatore binario. Alla fine, la macchina controlla che il contatore sia della forma 10000... (cioè un 1 seguito da un numero arbitrario di 0). La complessità spaziale della MT così ideata è $\log(2^n)$, cioè $O(n)$. La complessità temporale è $O(n2^n)$. Si noti che in questo caso n non è la lunghezza della stringa in input, ma vale la relazione $|x| = 2^n$, cioè $n = \log(|x|)$.
Tuttavia, si può vedere che incrementare m volte un contatore binario ha complessità $O(m)$. Infatti, la complessità di m incrementi di un contatore binario è $O\left(m \sum_{h=0}^{\lfloor \log(m) \rfloor} \frac{h+1}{2^h}\right) = O(m)$ (poiché si può dimostrare che $\sum_{h=0}^{\infty} \frac{h+1}{2^h} \leq 5$). Questo perché ogni cifra in posizione h viene modificata un numero di volte pari a $\frac{m}{2^h}$; per esempio, la cifra in posizione 0, cioè quella meno significativa, è modificata sempre; quella in posizione 1 è modificata solo se la prima è 1, quindi la metà delle volte; quella in posizione 2 viene modificata solo se le prime 2 sono entrambe 1, quindi un quarto delle volte, ecc.. Quindi, in realtà, la complessità temporale della macchina è $O(2^n)$.

b.

La macchina RAM si comporta in maniera simile alla MT: incrementa un contatore di 1 per ogni *a* letta. Alla fine, si continua a dividere il contatore per 2, fino a che o non si ottiene 0, o si trova un numero dispari.
A criterio di costo costante la MT ha complessità spaziale $O(1)$ e complessità temporale $O(2^n)$. A criterio di costo logaritmico la complessità spaziale è $O(\log(2^n))$, cioè $O(n)$, mentre la complessità temporale è $O(n2^n)$.

Esercizio 3

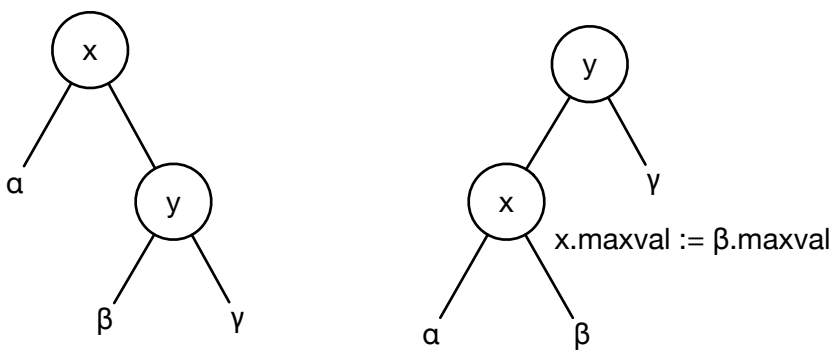
Lo pseudocodice ricorsivo dell'esercizio ha una funzione di complessità definita dalla seguente ricorrenza:

$$T(n) = T\left(\frac{2}{5}n\right) + T\left(\frac{3}{5}n\right) + \Theta(n)$$

che si risolve mediante il metodo di sostituzione (in modo analogo a quanto visto a lezione per la ricorrenza $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2}{3}n\right) + \Theta(n)$), dopo avere fatto l'ipotesi che la soluzione sia $\Theta(n \log(n))$.

Esercizio 3

Per mantenere aggiornato il campo *maxval* è sufficiente, in fase di inserimento, ogni volta che si scende nell'albero verso destra, controllare se il valore dell'elemento da inserire è maggiore di *maxval*, e, se lo è, aggiornare il valore di *maxval* a quello dell'elemento da inserire. Ovviamente, il valore di *maxval* del nodo inserito è pari alla propria chiave.
In fase di cancellazione, una volta eliminato il nodo *y*, se questo non ha figlio destro occorre risalire l'albero e, tutte le volte che si risale verso sinistra, aggiornare il valore di *maxval* del padre a quello del suo figlio destro (a quello della sua chiave se il figlio destro è NIL), fino a che non si risale la prima volta verso destra.
In fase di rotazione verso sinistra, è sufficiente assegnare a *x.maxval* il valore di *β.maxval*, come indicato in figura (se *β* è NIL, `x.maxval := x.key`). La rotazione destra è analoga.



In tutti i casi la complessità degli algoritmi rimane $\Theta(\log(n))$, in quanto al massimo le nuove operazioni richiedono di risalire (nel caso della `DELETE`) una sola volta l'albero (tutte le altre si eseguono in tempo costante).

Algoritmi e Principi dell'Informatica

Appello d'esame - 25 febbraio 2016

Tempo a disposizione: 2h30

Esercizio 1 (9 punti)

1.
Si definisca formalmente il seguente modello di automa, chiamato *macchina di Turing monodirezionale ad un nastro* (MTM1).
Una MTM1 è un accettore nondeterministico (non effettua quindi traduzioni) dotato di un nastro di ingresso e un nastro di memoria. Il funzionamento è analogo a quello di una MT tradizionale, a parte il movimento delle due testine dei nastri: esso può essere solamente S (stop) o R (right).

2.
Si confronti la capacità espressiva delle MTM1 con i modelli visti a lezione.

3.
Si consideri una variante "a nastro singolo" e deterministica della MTM1 e se ne valuti l'eventuale differenza di capacità espressiva.

Esercizio 2 (8 punti)

Si consideri il seguente insieme.

$S = \{ i \mid$ La macchina di Turing M_i termina la sua esecuzione in meno di 100 passi per qualche ingresso $\}$

S è ricorsivamente enumerabile? S è ricorsivo? Giustificare brevemente le risposte.

Esercizio 3 (7 punti)

Si descriva un algoritmo che, dato un array A di n interi distinti ed un valore $k \leq n$, restituisce il k-esimo elemento più piccolo di A.
Si calcoli la complessità dell'algoritmo descritto.

NB1: Il punteggio dato sarà tanto maggiore quanto migliore è la complessità dell'algoritmo scritto.

NB2: Per definire l'algoritmo è possibile far riferimento ad altri algoritmi noti, specificandone con precisione funzionalità e complessità.

Esercizio 4 (9 punti)

Sia T un albero binario. Sia x un suo nodo e siano $x.left$ e $x.right$ il sottoalbero sinistro e destro di x. Denotiamo con $size(x)$ il numero di nodi del sottoalbero radicato in x.
Un albero binario T si dice ***perfettamente bilanciato*** quando, per ogni suo nodo x, la differenza tra il numero dei nodi dei due sottoalberi di x è in valore assoluto al più 1:

$$| size(x.left) - size(x.right) | \leq 1.$$

Si definisca un algoritmo che, dato in input un nodo x, verifica se l'albero radicato in x è perfettamente bilanciato.
Si calcoli la complessità dell'algoritmo definito.

Soluzioni

Esercizio 1

1.

$A = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, stato iniziale $q_0 \in Q$, stati finali $F \subseteq Q$

Funzione di transizione $\delta: Q \times \Sigma \times \Gamma \rightarrow \wp(Q \times \Gamma \times \{S, R\}^2)$

Configurazione: per comodità usiamo come seconda componente la stringa in ingresso ancora da leggere (cioè la parte destra della stringa, testina inclusa); per la terza componente, che rappresenta il nastro, indichiamo la parte scritta e visitata dalla macchina (cioè la parte sinistra, testina inclusa).

$\langle q, a\text{-}x, y\text{-}b \rangle \vdash \langle q', a'\text{-}x, y\text{-}b'\text{-}c \rangle$, se $(q', b', m_1, m_2) \in \delta(q, a, b)$ dove

$a' = \varepsilon$ se $m_1 = R$ altrimenti $a' = a$

$c = \varepsilon$ se $m_2 = S$ altrimenti $c = \_$ (blank)

$x \in L(A)$ sse $\langle q_0, x, Z_0 \rangle \vdash^* \langle q_F, x', y \rangle$, $q_F \in F$ e si ferma (cioè $\delta(q_F,a,b) = \perp$, per ogni a, b).

2.

Una MTM1 accede solo ad una cella di memoria (quella corrente) e non può in alcun modo ritornare sulla parte sinistra del nastro di memoria. Questa informazione è finita ($\in \Gamma$) e può essere facilmente codificata nello stato dell'organo di controllo. Per questo motivo le MTM1 definiscono tutti e soli i linguaggi Regolari.

3.

L'uso di un unico nastro e del determinismo non cambia nulla.

Esercizio 2

S è ricorsivo (e quindi anche ricorsivamente enumerabile). Infatti basta verificare se la macchina si arresta entro 99 passi per qualche stringa di lunghezza <100. Tali stringhe sono in numero finito, quindi la verifica avviene in un numero finito di passi. Non occorre considerare stringhe più lunghe perché, se anche ci fosse una stringa più lunga di 100 caratteri tale per cui la macchina si arresti in meno di 100 mosse, e quindi avendo letto un prefisso di lunghezza minore di 100, necessariamente vi sarebbe anche una stringa con lo stesso prefisso ma lunga meno di 100 che farebbe arrestare la macchina nello stesso numero di mosse.

Si noti che in questo caso non è applicabile il teorema di Rice, in quanto la proprietà delle macchine di Turing considerate per l'insieme S (terminare in meno di 100 passi per qualche ingresso) non è una proprietà della funzione calcolata, bensì una proprietà strutturale della macchina.

Esercizio 3

Questo problema prevede diverse soluzioni. Ne riportiamo alcune.

- E' possibile ordinare l'insieme usando uno degli algoritmi classici di ordinamento ($O(n \log(n))$, non potendo fare alcuna assunzione sui valori dell'insieme), quindi restituire i primi k elementi.
- E' possibile far uso di una min-heap. Si rimuove il minimo dalla min-heap k volte, invocando dopo ogni rimozione MIN-HEAPIFY. Si restituisce in output il valore trovato con l'ultima chiamata. Occorre inizialmente invocare BUILD-MIN-HEAP, che ha complessità $O(n)$. Il costo delle k rimozioni del minimo e conseguente invocazione di MIN-HEAPIFY è $O(k \log(n))$. Si ottiene quindi complessità totale in $O(n + k \log(n))$.

Esercizio 4

E' sufficiente effettuare una visita dell'albero in ordine posticipato, contando i nodi nel sottoalbero sinistro (contemporaneamente verificando se è bilanciato), quindi fare la stessa cosa sul sottoalbero destro, quindi verificare il bilanciamento del sottoalbero corrente, e restituire una coppia di valori contente il numero di nodi totali del sottoalbero, e l'indicazione se l'albero è bilanciato o meno.

```
CHECK-BALANCE(x)
  if x = NIL
    return (true, 0)
  (lres, lsize) := CHECK-BALANCE (x.left)
  (rres, rsize) := CHECK-BALANCE (x.right)
  if (not lres) or (not rres) or |lsize - rsize| > 1
    return (false, lsize+rsize+1)
  return (true, lsize+rsize+1)
```

La complessità è ovviamente O(n), con n numero dei nodi nel sottoalbero di radice x.

Algoritmi e Principi dell'Informatica

Appello d'esame - 30 giugno 2016

Tempo a disposizione: 2h30

Esercizio 1 (7 punti)

Definire mediante una formula MFO il linguaggio fatto di tutte e sole le stringhe di simboli *w* (per "warning"), *a* (per "alarm"), *c* (per "cancel"), e *n* (per "nothing") tali che, dopo ogni *w*, entro 10 posizioni nella stringa c'è un simbolo *a*, a meno che non ci sia nel frattempo un simbolo *c*, nel qual caso non ci sono simboli *a* per le 10 posizioni successive alla *w*.
Inoltre, tra 2 consecutivi simboli *w* ci devono essere almeno 10 simboli diversi da *w*.

Esercizio 2 (9 punti)

Sia $k > 1$, e siano $L_1, L_2, \ldots L_k$, k linguaggi definiti su un alfabeto Σ, tali che valgano le seguenti proprietà:

- $\forall i \neq j, L_i \cap L_j = \emptyset$,
- $L_1 \cup L_2 \cup \ldots \cup L_k = \Sigma^*$,
- $\forall i, L_i$ è semidecidibile.

Si dica, giustificando brevemente ogni risposta, se le seguenti affermazioni sono vere o false.

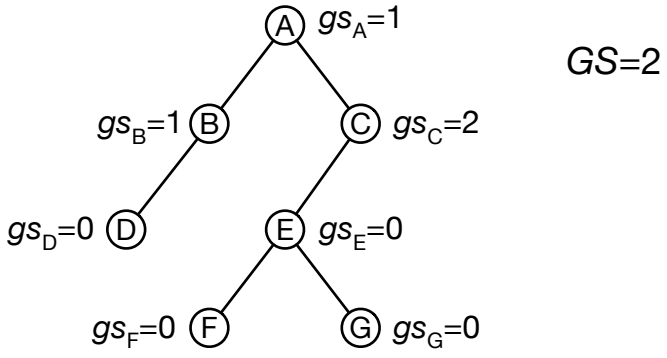
- Ognuno dei linguaggi $L_1, L_2, \ldots L_k$ è ricorsivo.
- E' possibile che alcuni di essi siano regolari.
- Ognuno dei linguaggi $L_1, L_2, \ldots L_k$ è necessariamente regolare.

Esercizio 3 (7 punti)

Si definisca una MT a k nastri che riconosca il linguaggio L = {www | w ∈ $\{0,1\}^+$}, e se ne valutino le complessità spaziale e temporale.

Esercizio 4 (8 punti)

In un albero binario, il ***grado di sbilanciamento di un nodo*** può essere calcolato come valore assoluto della differenza fra il numero di foglie presenti nei suoi due sottoalberi. A partire da tale valore calcolato per ogni nodo è possibile ricavare un ***grado di sbilanciamento di un albero*** come il massimo tra i gradi di sbilanciamento calcolati per tutti i suoi nodi.
Per esempio, si consideri il seguente albero binario, per il quale si riportano il grado di sbilanciamento (gs_n) di ogni nodo *n* dell'albero ed il grado di sbilanciamento (*GS*) di tutto l'albero:



Il grado di sbilanciamento *per l'albero* è *2*, poiché questo è il massimo valore di gs_n calcolato per i nodi dell'albero (che in questo caso è il valore calcolato per il nodo C, il cui sottoalbero sinistro ha 2 foglie, mentre quello destro non ne ha alcuna, essendo vuoto).
Si definisca un algoritmo per il calcolo del grado di sbilanciamento di un albero binario, e se ne dia la complessità.

Soluzioni

Esercizio 1

$$\forall x, z\ (\ \ w(x) \wedge z = x + 10 \rightarrow \exists y\ (x < y \wedge y \leq z \wedge a(y) \wedge \forall u\ (\ x < u \wedge u < y \rightarrow\ \neg c(u)))$$
$$\vee$$
$$\exists y\ (x < y \wedge y \leq z \wedge c(y) \wedge \forall u\ (\ x < u \wedge u < z \rightarrow\ \neg a(u))))$$
$$\wedge$$
$$\forall x, z\ (\ \ w(x) \wedge z = x + 10 \rightarrow \forall u\ (\ x < u \wedge u \leq z \rightarrow\ \neg w(u)))$$

Esercizio 2

- Poiché i linguaggi sono tra loro disgiunti e la loro unione è l'intero Σ^*, enumerando tutte stringhe di L_1, L_2, .. L_k nell'ordine 1, 2, …k, (ossia, una stringa di L_1, una stringa di L_2, …) prima o poi una qualsiasi stringa x deve comparire in una e una sola delle sequenze L_i, e quindi si può decidere a quale linguaggio appartiene.
- E' certamente possibile che qualche o anche tutti gli L_i, siano regolari, ma non necessario (per esempio, il linguaggio $a^n b^n c^n$ e il suo complemento soddisfano i requisiti, ma non sono regolari).

Esercizio 3

La MT fa una prima passata sulla stringa in ingresso (che supponiamo di indicare con x) e sul nastro 1 memorizza un simbolo X ogni 3 caratteri letti–in questo modo viene calcolata la lunghezza della sottostringa w in unario. Alla seconda passata la macchina copia la sottostringa w (cioè la prima sottostringa di x di lunghezza $|x|/3$) sul nastro 2, sfruttando il nastro 1 per vedere quando esso termina. Continua poi la lettura, controllando che la seconda e la terza sottostringa di lunghezza $|x|/3$ di x siano identiche a quanto salvato nel nastro 2.

Complessità spaziale: vengono memorizzate 2 stringhe di lunghezza $n/3$ (con $n = |x|$), quindi $S(n) = \Theta(n)$.
Complessità temporale: la macchina effettua 2 scansioni lineari di x, quindi $T(n) = \Theta(n)$.

Esercizio 4

E' sufficiente effettuare una visita dell'albero in ordine posticipato, contando i nodi nel sottoalbero sinistro (contemporaneamente verificando se è bilanciato), quindi fare la stessa cosa sul sottoalbero destro, quindi verificare il bilanciamento del sottoalbero corrente, e restituire una coppia di valori contente il numero di nodi totali del sottoalbero, e l'indicazione se l'albero è bilanciato o meno.

```
CHECK-BALANCE(x)
  if x = NIL
    return (true, 0)
  (lres, lsize) := CHECK-BALANCE (x.left)
  (rres, rsize) := CHECK-BALANCE (x.right)
  if (not lres) or (not rres) or |lsize - rsize| > 1
    return (false, lsize+rsize+1)
  return (true, lsize+rsize+1)
```

La complessità è ovviamente $O(n)$, con n numero dei nodi nel sottoalbero di radice x.

Algoritmi e Principi dell'Informatica

Appello d'esame - 1 settembre 2016

Tempo a disposizione: 2h30

Esercizio 1 (9 punti)

1. Si costruisca un automa, preferibilmente a minima potenza riconoscitiva, che accetti il linguaggio $L_1 \subseteq \{a,b\}^*$ costituito da tutte e sole le stringhe w tali che, se w contiene la sottostringa aa, allora la sua lunghezza è dispari.
2. Si determini la famiglia di automi a potenza riconoscitiva minima–o minimale–tale che un suo membro riconosca il linguaggio $L_2 \subseteq \{a,b,c\}^*$, costituito da tutte e sole le stringhe w tali che w includa la stringa ca^nc (per qualche $n > 0$) e, successivamente, la stringa $ba^{2n}b$ (la stringa w può contenere altre sottostringhe oltre alle suddette ca^nc e $ba^{2n}b$).
    a. Si costruisca un automa in tale famiglia che riconosca L_2.
    b. Si costruisca una grammatica, preferibilmente a minima potenza generativa, che generi L_2.

Esercizio 2 (8 punti)

Per entrambi i punti seguenti si assuma che L sia un linguaggio infinito costruito su un alfabeto A.

1. Sia L un linguaggio ricorsivo.
   Esiste necessariamente una enumerazione algoritmica E che enumera tutte le stringhe di L in ordine lessicografico? Motivare brevemente la risposta.
2. Sia E una enumerazione algoritmica E che enumera tutte le stringhe di un linguaggio L in ordine lessicografico.
   L è necessariamente ricorsivo? Motivare brevemente la risposta.

Esercizio 3 (7 punti)

Si descriva una MT a k nastri che, dato in input un valore n naturale maggiore di 0 codificato in *unario*, produce in output le prime n potenze di 4, cioè 4^0, 4^1, 4^2, ... 4^{n-1}, ognuna codificata in *binario* e seguita dal simbolo #.

Dare le complessità temporale e spaziale della MT ideata.

Esercizio 4 (8 punti)

E' noto che il Quicksort ha un comportamento nel caso pessimo $O(n^2)$; ad esempio la versione presentata nel corso (e fornita dal testo) ha complessità $O(n^2)$ quando l'array in input è già totalmente ordinato.

1. Si modifichi il codice della versione suddetta, riportata qui sotto per comodità, in modo che la complessità di esecuzione, se A è già ordinato, risulti $O(n \cdot \log(n))$.
   NB: la nuova versione deve essere esclusivamente una modifica dell'algoritmo originario, non includere altri (sotto)algoritmi più adatti al caso specifico.
2. Qual è la complessità asintotica della nuova versione nel caso pessimo? Descrivere come è fatto il caso pessimo, illustrandolo con un esempio.

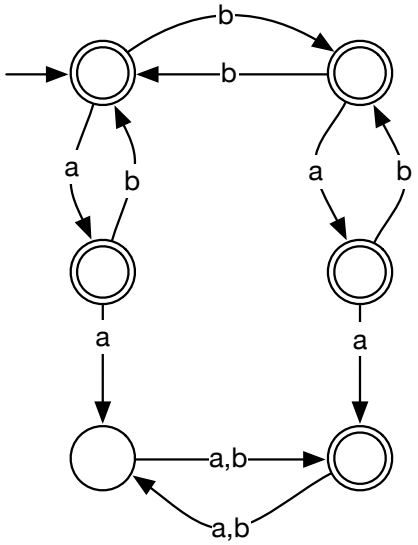
```
QUICKSORT(A, p, r)
1 if p < r
2   q := PARTITION(A, p, r)
3   QUICKSORT(A, p, q-1)
4   QUICKSORT(A, q+1, r)
```

```
PARTITION(A, p, r)
1 x := A[r]
2 i := p - 1
3 for j := p to r - 1
4   if A[j] ≤ x
5     i := i + 1
6     swap A[i] ↔ A[j]
7 swap A[i+1] ↔ A[r]
8 return i + 1
```

Soluzioni

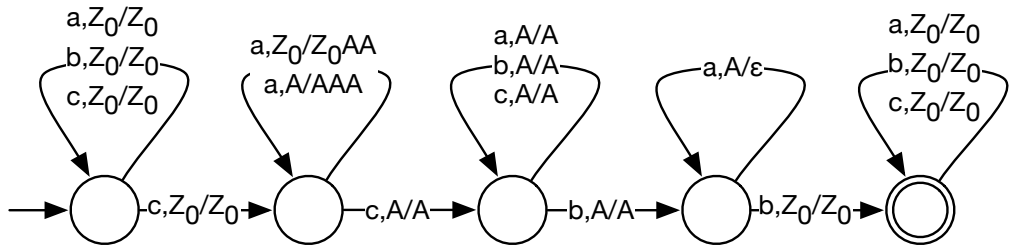
Esercizio 1

1. Il seguente automa a stati finite riconosce L_1.



2. Un automa a stati finiti non è evidentemente in grado di riconoscere L_2 a causa della necessità di un conteggio illimitato. Neanche un automa a pila deterministico è in grado di operare il riconoscimento perché una stringa del linguaggio può contenere diverse sottostringhe del tipo ca^nc, di cui solo una abbia una corrispondente sottostringa $ba^{2n}b$ come ad esempio nella stringa *acaaacbbcaccbaabc*.
E' anche possibile costruire una rete di Petri che riconosca L_2; quindi, siccome PDA e reti di Petri hanno potenza riconoscitiva incomparabile, vi sono almeno due famiglie di automi, tra quelle classiche, a potenza riconoscitiva minimale, in grado di formalizzare L_2.

a.



b. S → XDX
D → caEaab
E → aEaa | cXb
X → aX | bX | cX | ε

Esercizio 2

1) L è ricorsivo, quindi esiste un "decisore" per L, cioè una macchina di Turing M che termina sempre e scrive 1 in uscita se la stringa in ingresso appartiene a L, 0 altrimenti. L'enumerazione algoritmica E richiesta si ottiene come segue: basta enumerare tutte le stringhe di A* in ordine lessicografico (cosa sempre possibile) e, per ciascuna di esse, testarne l'appartenenza ad L tramite M (che è disponibile per ipotesi), stampandola in uscita solo se appartiene a L.

2) L è enumerabile algoritmicamente mediante E in ordine lessicografico. Si può quindi costruire un decider M per L come segue: quando riceve una stringa x in ingresso, M esegue l'enumerazione E e legge la lista di tutte le stringhe di L in ordine lessicografico fino a che non compare una stringa che lessicograficamente stia

dopo x (tale stringa deve necessariamente esistere, perché L è infinito). Se x è apparso nell'enumerazione, allora M stampa 1 in uscita, altrimenti stampa 0.

Esercizio 3

Per produrre i valori desiderati è sufficiente una MT a 1 nastro di memoria, che funziona nel seguente modo:

- Legge un simbolo dal nastro in ingresso.
- Scrive 1 sul santro di output.
- Copia tutti gli 0 (leggendoli da sinistra a destra) dal nastro di memoria al nastro di output, ed aggiunge in fondo al nastro di output un #.
- Mette due 0 sul nastro di memoria (la testina è ora in fondo al nastro di memoria).
- Torna all'inizio del nastro di memoria, leggendo gli 0 da destra a sinistra, e poi ricomincia da capo.

Per il calcolo della complessità spaziale conta solo il nastro di memoria, non il nastro di input, né quello di output, quindi la complessità è $\Theta(n)$.
Per il calcolo della complessità temporale, il costo della i-esima iterazione del ciclo è $\Theta(2i)$, quindi il costo totale è $\sum_{i=\{1..n\}} 2i$, che è $\Theta(n^2)$.

Esercizio 4

1. Il comportamento dell'algoritmo è determinato dalla scelta del "pivot" che nel nostro caso è l'ultimo elemento dell'array. Perciò se l'array è già ordinato la partizione è totalmente sbilanciata. In questo caso perciò conviene scegliere come pivot l'elemento mediano, in modo che la partizione lasci inalterato l'array e così via ricorsivamente ottenendo una complessità $O(n \cdot \log(n))$. La modifica dell'algoritmo in tal senso può semplicemente consistere nel premettere al codice di PARTITION l'istruzione
   ```
   k = (r + p)/2
   ```
   e usare A[k] come pivot, ossia scambiare A[k] con A[r]. Alla fine della partizione, l'array tornerà nella permutazione originaria già ordinata e così via ricorsivamente.
2. Ovviamente anche la versione così modificata ha un comprtamento $O(n^2)$ nel caso pessimo che si verifica quando l'array è tale che il pivot è sempre il massimo dell'array che rimane da ordinare. Quindi, prendendo come esempio un array di lunghezza pari, l'elemento di mezzo è il massimo, il secondo elemento più grande è l'ultimo, ecc. Un esempio di tale array è [2, 4, 6, 8, 10, 5, 3, 7, 1, 9], che dà luogo alla seguente sequenza di chiamate (la parte non in grassetto è l'array di sinistra in ogni successiva chiamata di Quicksort, la parte in grassetto sono i pivot, e l'array di destra è sempre vuoto):
   [2, 4, 6, 8, 10, 5, 3, 7, 1, 9]
   [2, 4, 6, 8, 9, 5, 3, 7, 1, **10**]
   [2, 4, 6, 8, 1, 5, 3, 7, **9, 10**]
   [2, 4, 6, 7, 1, 5, 3, **8, 9, 10**]
   [2, 4, 6, 3, 1, 5, **7, 8, 9, 10**]
   [2, 4, 5, 3, 1, **6, 7, 8, 9, 10**]
   [2, 4, 1, 3, **5, 6, 7, 8, 9, 10**]
   [2, 3, 1, **4, 5, 6, 7, 8, 9, 10**]
   [2, 1, **3, 4, 5, 6, 7, 8, 9, 10**]
   [1, **2, 3, 4, 5, 6, 7, 8, 9, 10**]
   [**1, 2, 3, 4, 5, 6, 7, 8, 9, 10**]

Algoritmi e Principi dell'Informatica

Appello d'esame - 15 settembre 2016

Tempo a disposizione: 2h30

Esercizio 1 (8 punti)

Descrivere mediante formule di logica MFO/MSO il linguaggio L fatto di tutte e sole le stringhe tali che in ogni suffisso di lunghezza maggiore o uguale di 5 caratteri ci sono almeno 2 simboli 'a' a distanza 4 uno dall'altro. Inoltre, ogni stringa del linguaggio deve contenere almeno 2 simboli 'b' (in posizione qualunque).

Si preferiscono formule scritte nella logica a potenza espressiva minore tra quelle in grado di esprimere il linguaggio.

Esercizio 2 (8 punti)

In matematica, si definiscono *numeri primi gemelli* due numeri primi che differiscano tra loro di 2. La *congettura dei numeri primi gemelli* afferma che esistono infinite coppie di numeri primi gemelli.

1. Ad oggi, la più grande coppia di numeri primi gemelli trovata è $3756801695685 \cdot 2^{666669} \pm 1$. Si tratta di numeri con ben 200700 cifre, ma non è ancora stato dimostrato l'enunciato della congettura, che potrebbe quindi anche essere falsa.
   Che cosa si può concludere in merito alla decidibilità dell'enunciato della congettura?
2. Sia f una funzione definita come segue:
   $f(x) =$ 1 se esiste una coppia di numeri primi gemelli maggiori di x;
   0 altrimenti.
   La funzione f è calcolabile?

Esercizio 3 (9 punti)

Si consideri il linguaggio L_q di figure definito sull' alfabeto {a, b}, composto da tutte e sole le figure *quadrate* fatte da righe di soli simboli 'a' alternate a righe di soli simboli 'b', partendo con le 'a'.

Es.
```
aaaa
bbbb
aaaa
bbbb
```

Si descrivano esaurientemente due macchine di Turing che accettino L_q, una a k-nastri e una a nastro singolo, sapendo che la figura è fornita sul nastro in ingresso riga per riga e senza separatori, e se ne studino le complessità *spaziali e temporali*.

Esercizio 4 (7 punti)

Sia dato un vettore di interi, A, ordinato in ordine non decrescente. Sia dato inoltre un valore intero v. Si definisca un algoritmo che verifichi se A contiene due elementi a_i e a_j tali che $a_i - a_j = v$.

Il punteggio dell'esercizio sarà tanto più alto quanto migliore sarà la complessità temporale dell'algoritmo.

Soluzioni

Esercizio 1

E' sufficiente una formula MFO, come per esempio la seguente:

$\forall x, z$ (last(z) $\wedge$ $\exists y$ ($x \leq y \wedge z = y + 4$) $\rightarrow$ $\exists t,w$ ($x \leq t \wedge w = t+4 \wedge w \leq z \wedge a(t) \wedge a(w)$))

$\wedge$

$\exists x, z$ ($x < z \wedge b(x) \wedge b(z)$)

Esercizio 2

1.

Si tratta di una questione certamente decidibile poiché la congettura o è vera, o è falsa, quindi si tratta di un problema in forma chiusa.

2.

Se la congettura dei numeri primi gemelli è vera, allora la funzione *f* vale costantemente 1, ed è pertanto calcolabile.

Se la congettura è falsa, allora esiste un numero *k* oltre il quale non è più vero che esistono due numeri primi gemelli (e prima del quale esistono). Pertanto la funzione *f* sarà fatta a "scalino", e sarà pari a 1 per ingressi minori di *k*, e pari a 0 per ingressi maggiori o uguali a *k*. Anche in questo caso la funzione è certamente calcolabile.

Esercizio 3

Macchina a k-nastri

Si legge la prima sequenza di 'a' copiandola su 2 nastri di memoria; poi il primo nastro viene utilizzato per contare il numero di elementi di ogni riga, mentre il secondo serve per contare il numero di righe.

Chiaramente basta una passata sulla stringa in ingresso. Siano m gli elementi di una riga, la stringa è lunga $n = m^2$, mentre la memoria occupata risulta $2m$.

Quindi: $S(n) = \Theta(\sqrt{n})$, $T(n) = \Theta(n)$.

Macchina a nastro singolo

L'idea è controllare la lunghezza di ogni riga, cambiando di volta in volta le 'a' in 'A' e le 'b' in 'B' riga per riga, da sinistra verso destra, a parte i primi elementi di ogni riga che verranno cambiati in 'X'. In questo modo possiamo controllare che tutte le righe siano lunghe uguali, facendo $\Theta(nm)$ mosse.

Per controllare che il numero di righe sia corretto, la macchina può controllare che il numero delle 'X' sia uguale al numero di 'A' della prima riga + 1, facendo ulteriori $\Theta(nm)$ mosse (ad esempio cambiandoli uno alla volta in altri simboli).

Quindi: $S(n) = \Theta(n)$, $T(n) = \Theta(n\sqrt{n})$.

Esercizio 4

Una soluzione banale, ma poco efficiente, controlla tutte le coppie, $A[i]$, $A[j]$, con $1 \leq i < j \leq A.length$, per verificare se $A[j] - A[i] = v$. La complessità di tale soluzione è $O(n^2)$.

È possibile poi definire una soluzione con complessità $O(n\log(n))$. Per ognuno degli n elementi dell'array, $A[i]$, si applica una *ricerca binaria* agli elementi successivi per cercare un elemento pari a $v+A[i]$.

```
CercaCoppia-RicercaBinaria (A,v)
for i := 1 to A.length
  if BIN-SEARCH(A,i+1,A.length,v+A[i])
    return true
return false
```

È infine possibile ottenere una soluzione più efficiente, di complessità $O(n)$, scandendo gli elementi in modo lineare tramite un opportuno incremento di due indici, i e j:

```
CercaCoppia-Lineare(A,v)
i := 1
j := 2
while (j ≤ A.length)
  if (A[j]-A[i] < v)
    j := j + 1
  else
    if (A[j]-A[i] > v)
      i := i + 1
    else
      return true;
return false
```

A ogni iterazione, si incrementa j oppure i, oppure si esce dalla funzione. A ogni iterazione vale la condizione $j \geq i$; quindi si eseguono $O(n)$ iterazioni.

Algoritmi e Principi dell'Informatica

Appello d'esame straordinario - 27 settembre 2016

Tempo a disposizione: 2h30

Esercizio 1 (12 punti)

Si consideri il linguaggio L $\subseteq$ {a,b}* fatto di tutte e sole le stringhe che iniziano e terminano con una sequenza di *esattamente n* simboli 'a' (cioè di *n* simboli 'a' seguiti/preceduti da una 'b'), con $1 \leq n \leq 4$.

1. Definire un automa (a stati finiti, a pila, ecc.) che riconosca il linguaggio L. L'automa deve avere potere espressivo minimo tra quelli che riconoscono il linguaggio L.
2. Scrivere una formula MFO/MSO che definisce il linguaggio L. La formula deve appartenere alla logica di potere espressivo minimo tra quelle che permettono di definire L.
3. Scrivere una grammatica che genera il linguaggio L. La grammatica deve avere il minor numero possibile di simboli nonterminali.

Esercizio 2 (6 punti)

Si consideri di nuovo il linguaggio L definito nell'esercizio 1.

1. Dire se è decidibile il problema di stabilire se una stringa *x* appartiene al linguaggio L.
2. Dire se è decidibile il problema di stabilire se, data una generica MT *M*, questa riconosce il linguaggio L.
3. Dire se è decidibile il problema di stabilire se, data una generica MT *M*, questa riconosce il linguaggio L ed è un automa a potere espressivo minimo tra quelli che riconoscono L.

Esercizio 3 (4 punti)

1. Dire quali sono le complessità temporali e spaziali minime per una MT a *k* nastri che riconosce il linguaggio L definito nell'esercizio 1.
2. Dire quali sono le complessità temporali e spaziali minime per una MT *a nastro singolo* che riconosce il linguaggio L definito nell'esercizio 1.

Esercizio 4 (4 punti)

Risolvere la seguente ricorrenza: $T(n) = 9T\left(\frac{n}{10}\right) + n \log n$

Esercizio 5 (7 punti)

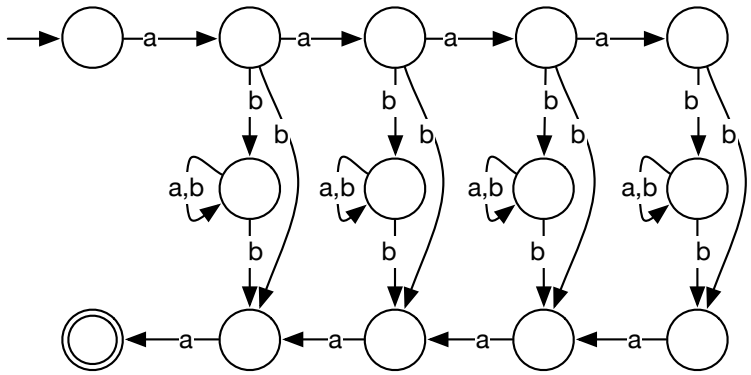
Si vuole progettare una struttura dati per memorizzare le relazioni di "amicizia" di un social network. La relazione di amicizia è simmetrica (se A è amico di B, B è amico di A), ed il numero di amici di una persona può essere molto grande. Si vuole progettare una struttura dati in cui la verifica del fatto che una persona sia amica di un'altra, l'aggiunta e la cancellazione di una persona, l'aggiunta e la cancellazione di una relazione di amicizia siano operazioni efficienti.

1. Descrivere una struttura dati appropriata per il problema.
2. Scrivere un algoritmo che, data una persona, restituisce la chiusura transitiva dei suoi amici (cioè che restituisce tutti gli amici, gli amici degli amici, gli amici degli amici degli amici, ecc. della persona data), e darne la complessità.

Soluzioni

Esercizio 1

1. E' sufficiente un automa a stati finiti.



2. E' sufficiente una formula MFO, come per esempio la seguente:
$\exists i \in [1,4](\forall j \in [0,i\text{-}1]$ a(i) $\wedge$ b(i) $\wedge \forall$ z (last(z) $\rightarrow \forall j \in [0,i\text{-}1]$ a(z-j) $\wedge$ b(z-i))

3. Una grammatica è la seguente:
S → abXba | aabXbaa | aaabXbaaa | aaaabXbaaaa | aba | aabaa | aaabaaa | aaaabaaaa
X → aX | bX | ε

Si noti che la grammatica non è a potere espressivo minimo tra quelli che generano L (per generare L basta una grammatica regolare, questa è noncontestuale).

Esercizio 2

1. Banalmente decidibile, per deciderlo basa usare l'automa definito al punto 1.1.

2. Indecidibile, si tratta del problema di stabilire se una MT calcola una certa funzione fissata oppure no.

3. Banalmente decidibile (e deciso) perché la MT non è l'automa a potere espressivo minimo che riconosce il linguaggio, quindi la risposta è "no" qualunque sia la MT in input.

Esercizio 3

Essendo il linguaggio riconoscibile da un FSA, è possibile scrivere sia una MT a k nastri, che una MT a nastro singolo che riconoscono L in tempo $\Theta(n)$; la MT a k nastri ha complessità spaziale $\Theta(1)$, in quanto non serve memoria aggiuntiva per simulare un FSA; la MT a nastro singolo, invece, ha complessità spaziale $\Theta(n)$, in quanto la stringa in input conta per lo spazio occupato.

Esercizio 4

La ricorrenza si risolve applicando il Master Theorem. Siamo nel "caso 3" (si vede facilmente che la condizione aggiuntiva $a\ \mathrm{f}(n/b) \leq c\ \mathrm{f}(n)$ è verificata, in quanto diventa $\frac{9n}{10}\log\frac{n}{10} \leq c \log n$, cioè $\frac{9n}{10}\log n - \frac{9n}{10}\log 10 \leq c \log n$, che è banalmente verificata prendendo, per esempio, c=99/100), per cui la soluzione è $T(n) = n \log n$.

Esercizio 5

1.

Per rappresentare la relazione desiderata basta usare un grafo, in cui però la rappresentazione a liste di adiacenza è sostituita da una in cui i nodi sono memorizzati in un albero bilanciato (per esempio red-black) invece che in un array, e al posto di liste di adiacenza si usano degli alberi (di nuovo bilanciati) di adiacenza.

2.

L'algoritmo è un normale algoritmo di search del grafo. Può essere per esempio realizzato mediante un BFS, con complessità temporale $O(N + V)$.

Algoritmi e Principi dell'Informatica

Appello d'esame - 8 febbraio 2017

Tempo a disposizione: 2h30

Esercizio 1 (12 punti)

Si consideri il linguaggio L fatto di tutte e sole le stringhe della forma $a^{2k+1}ba^{2k}$, con k un numero naturale tale che $k \geq 0$.

1. Definire un automa (a stati finiti, a pila, macchina di Turing, rete di Petri) che riconosca il linguaggio L. L'automa deve essere a potenza riconoscitiva minima tra quelli che riconoscono L. Dire se l'automa a potenza riconoscitiva minima che riconosce il complemento di L, cioè $\neg L$, è della stessa classe di quello definito.
2. Definire una grammatica che genera il linguaggio L. La grammatica deve avere il minor numero possibile di produzioni tra quelle che generano L. La grammatica scritta è a potere generativo minimo tra quelle viste a lezione che sono in grado di generare L?.
3. Definire, se possibile, una formula MFO o MSO che riconosce il linguaggio L. Se non è possibile, spiegare perché, e definire una formula MFO/MSO che riconosce il linguaggio L' contenente tutte e sole le stringhe della forma a^+ba^*. La formula deve essere della logica a potere espressivo minimo tra quelle che sono in grado di definire L'. Definire, se possibile, una formula logica che definisce il linguaggio complemento di L', cioè $\neg L$'.

Esercizio 2 (6 punti)

Si consideri la seguente congettura: esiste un numero naturale B tale che, indicato con p_n l'n-esimo numero primo (quindi $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 7$, e così via) si che per ogni n, $p_{n+1} - p_n < B$.

1. E' decidibile il problema di stabilire se la congettura è vera?
2. Si consideri la funzione $f(n) = p_{n+1} - p_n$. Dire se la funzione $f(n)$ è calcolabile oppure no.

Esercizio 3 (6 punti)

Si consideri di nuovo il linguaggio L definito nell'esercizio 1. Si delinei (senza necessariamente darne tutti i dettagli) una macchina a Turing a nastro *singolo* che riconosce il linguaggio L. Dare le complessità temporale e spaziale della MT ideata.

E' possibile risolvere il problema del riconoscimento del linguaggio L in modo più efficiente (dal punto di vista della complessità temporale) rispetto alla MT ideata, ma mediante un automa di potere espressivo inferire alla MT? E dal punto di vista della complessità spaziale?

Esercizio 4 (3 punti)

Si risolva la seguente ricorrenza: $8T\left(\frac{n}{6}\right) + n^{\frac{3}{2}}(\log n)^2$

Esercizio 5 (6 punti)

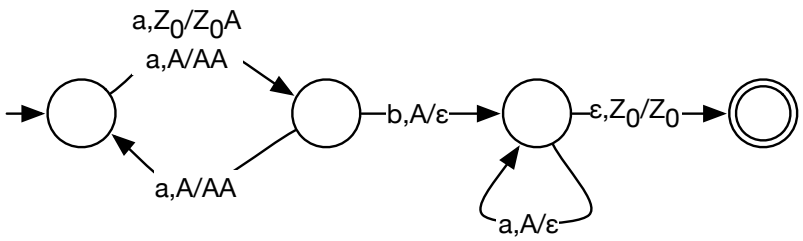
Sia dato un vettore di *interi* A. Si definisca un algoritmo che determini qual è la più lunga sottosequenza di valori di A in cui i valori sono tutti strettamente positivi (quindi anche diversi da 0) ed in ordine strettamente decrescente (se ci sono più sequenze della stessa lunghezza, l'algoritmo ne ritorna una di quelle).

Il punteggio dell'esercizio sarà tanto più alto quanto migliore sarà la complessità temporale dell'algoritmo.

Soluzioni

Esercizio 1

1.
L'automa a potere espressivo minimo che riconosce il linguaggio *L* desiderato è un AP deterministico, per esempio il seguente:



Essendo la classe dei problemi riconoscibili mediante APD chiusa rispetto al complemento, anche $\neg L$ è riconoscibile mediante un APD. Inoltre, $\neg L$ non è riconoscibile mediante un FSA, in quanto se lo fosse, anche *L* lo sarebbe, quindi la classe a potere espressivo minimo che riconosce $\neg L$ sono gli APD.

2.
Una grammatica (con solo 2 produzioni, che è il minimo) che genera il linguaggio desiderato è la seguente:
$S \rightarrow aaSaa \mid ab$
La grammatica è a potere generativo minimo (tra quelle viste a lezione) in quanto il linguaggio è riconoscibile da un APD.

3.
Essendo MSO equivalente agli FSA, e non potendo un FSA riconoscere il linguaggio *L*, non esiste una formula MSO in grado di definire *L* (e, a maggior ragione, non esiste una formula MFO). Una formula MFO che definisce il linguaggio *L'* è la seguente:
$\exists x, z\ (\ x < z \wedge a(x) \wedge b(z) \wedge \forall y\ (y \neq z \rightarrow a(y)))$
Una formula MFO che definisce il linguaggio $\neg L$' è semplicemente la negazione della formula precedente:
$\neg\exists x, z\ (x < z \wedge a(x) \wedge b(z) \wedge \forall y\ (y \neq z \rightarrow a(y)))$

Esercizio 2

1.
Si tratta di una questione certamente decidibile poiché la congettura o è vera, o è falsa, quindi si tratta di un problema in forma chiusa.

2.
La funzione è calcolabile; un algoritmo che la calcola è fatto nel modo seguente: si calcolano i numeri primi *n*-esimo e *n*+1-esimo (basta enumerare tutti i numeri naturali, testare se sono primi, e selezionare l'*n*-esimo e l'*n*+1-esimo che superano il test), quindi farne la differenza.

Esercizio 3

Una macchina a nastro singolo che riconosce il linguaggio *L* si può comportare in modo simile a quella che riconosce il linguaggio $a^n b^n$: è sufficiente scorrere più volte avanti e indietro il nastro, cancellando (marcando co un simbolo opportuno, per esempio X) due simboli *a* all'inizio della stringa e due simboli *a* alla fine, fino a che non rimane solo *ab*. La complessità spaziale di una simile MT è chiaramente $S(n) = \Theta(n)$, mentre per la complessità temporale vale $T(n) = \Theta(n^2)$.

Siccome il linguaggio è riconoscibile mediante APD, che sono strettamente meno potenti delle MT, ma le cui complessità temporale e spaziale sono $O(n)$, la risposta al secondo quesito è positiva per quel che riguarda la complessità temporale. Non è invece possibile riconosce il linguaggio con una complessità spaziale inferiore, in quanto il linguaggio non è riconoscibile mediante FSA (che sono il formalismo meno

potente delle MT che possono avere complessità spaziale meno che lineare, addirittura in questo caso costante).

Esercizio 4

La ricorrenza si risolve applicando semplicemente il metodo dell'esperto.

Si ha che $\log_b a = \log_6 8$, quindi $n^{\log_6 8}$, che vale circa $n^{1.16}$ è polinomialmente più piccolo di $n^{\frac{3}{2}}(\log n)^2$. Siamo quindi nel caso 3 del teorema (la condizione $af(n/b) \leq cf(n)$ è verificata già per $c = 1$, è si ha che $T(n) = \Theta(n^{\frac{3}{2}}(\log n)^2)$.

Esercizio 5

Una soluzione semplice, ma poco efficiente, controlla tutte le coppie, $A[i]$, $A[j]$, con $1 \leq i < j \leq A.length$, per verificare se tra di loro ci sono solo numeri positivi in ordine decrescente. La complessità di tale soluzione è $O(n^2)$.

È possibile definire una soluzione con complessità $O(n)$. Partendo dal primo elemento, si prosegue fino a che non si trova un numero negativo, o più grande di quello corrente, e, quando si trova un tale numero, si guarda se la sequenza che termina è più lunga di quella fino ad ora identificata.

```
CercaSequenza(A)
if (A.length > 0 and A[1] > 0)
  cur := 1
  first := 1
  last := 1
else
  cur := 0
  first := 0
  last := 0
i := 2
for i := 2 to A.length
  if A[i] > 0 and A[i] < A[i-1]
    if i-cur > last-first
      first := cur
      last := i
  else
    if A[i] > 0
      cur := i
    else
      cur := 0
return (first,last)
```

Si noti che, se A è vuoto o contiene solo numeri negative, l'algoritmo restituisce la coppia (0,0).