Algoritmi e Principi dell'Informatica

Appello del 1 Settembre 2016

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1 e 2 in 1 ora e 15 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 3 e 4 in 1 ora e 15 minuti.

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

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.

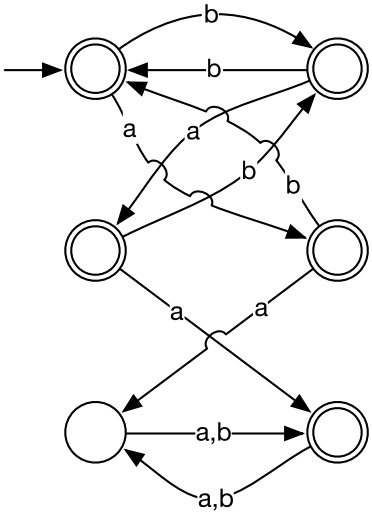
```
QUICKSORT(A, p, r)
1 if p < r
2   q := PARTITION(A, p, r)
3   QUICKSORT(A, p, q–1)
4   QUICKSORT(A, q+1, r)
```

```
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
```

Tracce delle soluzioni

Esercizio 1

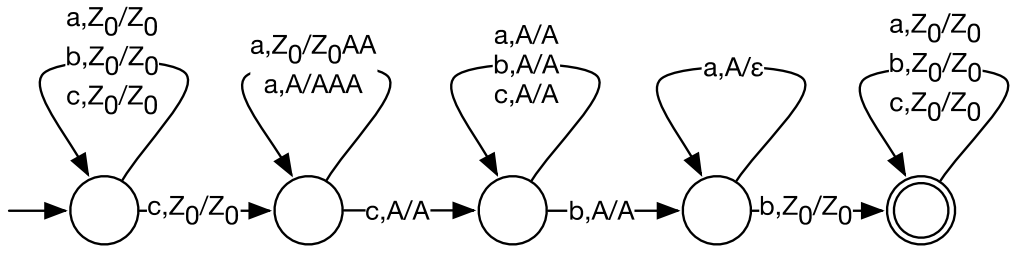
1. Il seguente automa a stati finite riconosce L_1.



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 | aXb
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 nastro 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**]