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 ≥ 2 che finiscono con 'bc', e una funzione di traduzione τ 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:

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

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

```
p1(n)
if n < 1
  print(n)
for j := 1 to 3
  p1(n/2)
for j := 1 to n
  for k := 1 to n
    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:

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

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

```
p2(n)
if n < 1
  print(n)
for l := 1 to 4
  p2(n/2)
for k := 1 to n
  for j := k+1 to n
    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.

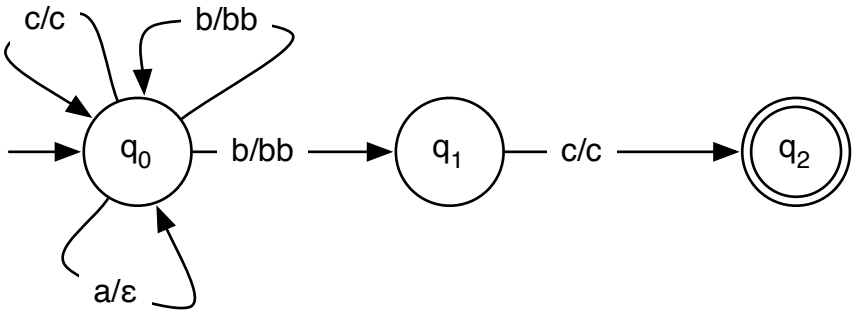
1. Si descriva un'opportuna struttura dati per rappresentare le relazioni descritte sopra.

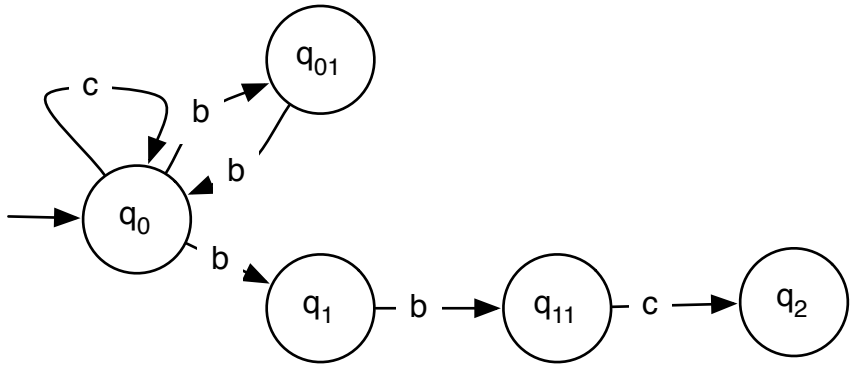
In un caso anomalo, ma non impossibile, è possibile che una persona sia "il proprio nonno"; il caso accade quando un uomo: (i) sposa una signora che ha una figlia; (ii) la figlia sposa il padre dell'uomo.

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)
for each u ∈ R.N
  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
    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).