Algoritmi e Principi dell'Informatica

Appello del 20 Settembre 2013

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 3 ore.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi1 e 2 in 1 ora e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 3, 4, e 5 in 1 ora e 30 minuti.

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

Esercizio 1 (punti 8)

Il programma P riceve in ingresso due stringhe *x* e *y* costruite sull'alfabeto italiano tali che l'*insieme* dei caratteri di *x* coincida con l'insieme dei caratteri di *y*.

Il programma P verifica se una delle due stringhe in ingresso sia una sottostringa dell'altra. In caso positivo, il programma pone a 1 il valore della variabile ESITO, altrimenti a 0.

Ecco alcuni esempi. Se *x* è basso e *y* è asso, le stringhe *x* e *y* non soddisfano le precondizioni. Se *x* è sasso e *y* è asso, le precondizioni sono soddisfatte e P pone ESITO=1. Infine, se *x* è sasso e *y* è ossa, le precondizioni sono soddisfatte e P pone ESITO=0.

Formulare in logica del prim'ordine le precondizioni e postcondizioni di P servendosi *esclusivamente* dei seguenti predicati e funzioni:

- il predicato unario *char* che è vero se e solo se il suo argomento è una stringa costituita da un unico carattere dell'alfabeto italiano;
- il predicato (binario) = che rappresenta la classica uguaglianza;
- la funzione binaria • che rappresenta la concatenazione di stringhe.

Esercizio 2 (punti 8 escludendo la domanda 4.b, 10 includendo anche essa)

Si consideri il gioco del "forza 4", in cui ci sono 2 giocatori, uno con pedine rosse ed uno con pedine gialle, in cui vince chi per primo mette in fila (orizzontalmente, verticalmente, o diagonalmente) 4 pedine del proprio colore in uno schema di dimensione 7 colonne per 6 righe. Una mossa del gioco consiste nello scegliere una colonna, e nel fare scendere una pedina lungo la colonna: se la colonna al momento della mossa contiene k pedine, in seguito alla mossa la nuova pedina occuperà la posizione k+1 (assumendo che la prima posizione sia la #1); non si possono depositare pedine in una colonna piena e inzialmentetutte le colonne sono vuote.

1. E' decidibile il problema di stabilire se, data una configurazione del gioco (cioè una distribuzione di pedine nello schema), esiste una sequenza di mosse che portano un giocatore a vincere?
2. E' decidibile il problema del punto 1 se lo schema di gioco, invece che essere 6×7, è di dimensione arbitraria h×k?
3. E' decidibile il problema del punto 1 se lo schema di gioco, invece che essere 6×7, non ha limiti né nelle righe né nelle colonne?
4. Si consideri di avere a disposizione una funzione che, date 2 configurazioni del gioco ed un colore, dice quale delle 2 è la "migliore" per quel colore (una configurazione potrebbe essere "migliore" perché, per esempio, rende più probabile al giocatore del colore dato di vincere la partita):

4.a E' decidibile il problema di stabilire se, dato un generico programma che prende in ingresso una configurazione del forza 4 (schema 6×7), ed un colore che deve muovere, questo ritorna sempre la mossa migliore per quel colore?

4.b E' semidecidibile il problema del punto 4.a?

Esercizio 3 (punti 8)
Si consideri la seguente –inutile!- funzione, definita mediante pseudocodice.

```
void    useless (int     n)
{for (i = 1, i <= n, i++) do aux(i)}
```

dove a sua volta la funzione aux(m) è definita nel modo seguente:

```
void    aux(intm)
        {if(m == 1) return
        else
                if(odd (m)) aux(m-1)
                else aux(m/2)
        }
```

Si valuti la complessità asintotica di useless mediante la notazione O,o, preferibilmente, mediante la notazione Θ.

Esercizio 4 (punti 6)

1. Si definisca una struttura dati per rappresentare la rete stadale di una città. La rete stradale è fatta di incroci, collegati da strade, che possono essere a senso unico o a doppio senso; una strada si dice tratto di strada elementare se collega due incroci senza che tra essi ve ne siano altri; è anche possibile che ci siano dei cavalcavia, ossia strade che intersecano i propri tragitti senza per questo dare origine a un incrocio.

2. Si progetti un algoritmo per la struttura dati definita al punto 1 che determini, dato un incrocio della città, un percorso che ritorni al punto di partenza senza passare 2 volte da uno stesso incrocio (ovviamente escluso il primo) o da una stessa strada. Il percorso trovato dovrebbe essere quello di lunghezza minore tra quelli che soddisfano i vincoli, intendendo come lunghezza il numero di tratti di strada elementari che lo compongono.

3. Si valuti la complessità (asintotica) dell'algoritmo progettato.

Esercizio 5 (punti 3)
Si consideri il problema di stabilire se un numero naturale, codificato in binaro, sia pari o dispari. Si dica, giustificando brevemente la risposta, con quale complessità spaziale e temporale possono risolvere questo problema (ovviamente, lavorando al meglio delle proprie possibilità) le seguenti macchine astratte:

1. Una macchina di Turing a k nastri
2. Una macchina di Turing a nastro singolo
3. Una RAM, con criterio di costo logaritmico.

Tracce delle Soluzioni

Esercizio 1

Definisco per convenienza il predicato *substring*:

$\forall a \forall b$ (*substring*(a, b) $\leftrightarrow \exists c \exists d$ c$\bullet$a$\bullet$d = b)

Precondizioni

$\forall c$ (*char*(c) $\rightarrow$ (*substring*(c, x) $\leftrightarrow$*substring*(c, y)))

Postcondizioni

ESITO=1 $\leftrightarrow$ (*substring*(x,y) $\vee$*substring*(y,x)) $\wedge$

ESITO=0 $\leftrightarrow \neg$ESITO=1

Esercizio 2

1. Decidibile: lo schema è finito.
2. Decidibile (è comunque un problema di ricerca in uno spazio limitato).
3. Decidibile, anzi deciso: essendoci un numero illimitato di posizioni, se un giocatore gioca in modo "stupido" è sempre possibile che l'altro vinca.

4.a Indecidibile:anche se la funzioneha un dominio finito, i possibili programmi sono infiniti: sussistono quindi le ipotesi del teorema di Rice.

4.b Semidecidibile: se un programma calcola la funzione in modo corretto, ciò può essere stabilito eseguendo il programma poiché la funzione ha un dominio finito.

Esercizio 3

La funzione aux ha una complessità che è soluzione della seguente equazione alle ricorrenze:

T(1) = 1;

T(n) = **if** odd(n) t**h**en 1 + T(n-1)

else 1 + T(n/2)

Consideriamo due casi estremi:

a) $n = 2^k$: in tal caso, abbiamo

T(n) = 1+ T(n/2) = 1 + 1 + T(n/4) = 1+1+1 … 1+ T(1) = 1*k , quindi T(n) è O(log(n))

b) n è dispari, (n-1)/2 è pure dispari e così via finché la ricorsione si chiude: in tal caso:

T(n) = 1+ T(n-1) = 1 + 1 + T((n-1)/2) = 1+1+1+ T((n-3)/2) = 1 +1 +1+1+ T((n-3)/4) …

che è O(2.log(n)) = O(log(n)).

Quindi aux ha una complessità O(log(n)) (e anche Θ(log(n))).

useless chiama n volte aux con parametro i che "va da 1 a n"; quindi

T(n) per useless è $\leq \sum_{i=1}^{n} c.\log(i)$ che è O(n.log(n)).

T(n) è anche $\geq \sum_{i=n/2}^{n} c.\log(i) \geq n/2.\log(\frac{n}{2})$, che è Ω(n.log(n))

Quindi T(n) è Θ(n.log(n)).

Esercizio 4

1. La struttura è un tipico esempio di grafo orientato. Si noti anche che la presenza dei cavalcavia non obbliga il grafo ad essere planare. Quindi esso può essere rappresentato in uno dei classici modi della letteratura (matrice delle adiacenze, lista dei nodi adiacenti, …).
2. Il problema è un classico problema di raggiungibilità: si può quindi usare uno dei tanti algoritmi dedicati a questo problema con opportuni, semplici adattamenti; ad esempio l'algoritmo BFS standard –descritto nel testo-potrebbe essere adattato al problema semplicemente verificando se il nodo v coincide con s: non appena questo test dà esito positivo, si è anche trovato il (un) percorso a distanza minima, grazie all'impostazione greadth-first dell'algoritmo di visita. Si noti che è impossibile passare due volte per la stessa strada, in un senso qualsiasi, senza anche passare due volte da uno stesso incrocio.

La complessità totale è quindi O(|V| + |E|) (però se si usa la matrice della adiacenze per rappresentare il grafo il ciclo for della linea di codice 10 nell'algoritmo fornito dal testo richiede un tempo O(|V|) e quindi la complessità diventa $O(|V|^2)$ indipendentemente da |E|.

Esercizio 5

Tutte le macchine non devono fare altro che verificare se l'ultimo bit sia 1 o 0:

1. La MT a k nastri deve solo spostare la testina fino all'ultimo bit:
   Complessità spaziale O(1) e temporale O(k) dove k è la lunghezza della stringa binaria che codifica il dato di ingresso n; ossia O(log(n)).
2. La MT a nastro singolo fa esattamente le stesse operazioni ma la sua complessità spaziale include sempre l'input (si tratta però di una convenzione)
3. La RAM può:
   a. leggere il dato di ingresso come intero con un'unica operazione; poi dividerlo per 2 e rimoltiplicare il risultato per due: in tal caso complessità spaziale e temporale sono entrambe O(log(n)).
   b. leggere il dato di ingresso bit a bit e verificare solo il valore dell'ultimo. In tal caso la complessità temporale è O(log(n)) e quella spaziale O(1) perché non deve memorizare tutta la stringa di ingresso ma solo l'ultimo bit letto; inoltre ogni singola operazione costa O(1) anche a criterio di costo logaritmico.