Algoritmi e Principi dell'Informatica
Seconda Prova in Itinere

27 Giugno 2019

Avvisi importanti

Il tempo a disposizione è di **1 ora e 30 minuti.**

Esercizio 1 (punti 4/15-esimi)

Si dica, spiegandone brevemente il motivo se le seguenti affermazioni sono vere o false:

a) Dati 2 formalismi riconoscitivi F1 e F2, se F1 è strettamente più potente di F2, allora, preso un qualunque linguaggio L riconoscibile sia mediante F1 che mediante F2, L è sicuramente riconoscibile da F1 con complessità temporale asintoticamente non superiore rispetto ad F2.

b) I linguaggi definibili mediante formule MFO sono riconoscibili da MT a nastro singolo con complessità temporale inferiore rispetto a quella delle macchine RAM.

Il criterio di costo adottato per ogni formalismo deve essere realistico anche in caso di dati "estremi" (numeri o strutture di grandezza illimitata).

Esercizio 2 (punti 6/15-esimi)

Sono dati un array A di n interi distinti e un valore intero X. Si vuole determinare se esistano *due* elementi di A la cui somma sia X. Progettare un algoritmo e una struttura dati che minimizzino il tempo di esecuzione a criterio di costo costante, ***nel caso medio***, senza eccessivo spreco di memoria. Si assuma che i valori interi contenuti nell'array A siano distribuiti in modo uniforme, ossia che all'interno di un certo intervallo, ad esempio [MININT ..MAXINT], abbiano tutti uguale probabilità di comparire in A.

Esercizio 3 (punti 6/15-esimi)

Sono dati in ingresso due array A e B, rispettivamente di n e di m elementi in N (numeri naturali), e una funzione f da N a N monotona crescente.

Si vuole trovare la coppia di elementi (a,b), se esiste, con a in A e b in B, dove a è il più grande elemento di A tale per cui a=f(b).

Descrivere un algoritmo che risolva il problema e determinarne la complessità asintotica. Si può assumere che la funzione f sia calcolabile in tempo O(1), a criterio di costo costante.

Tracce di soluzioni

Esercizio 1

a) Falsa: è ben noto che il linguaggio wcw^R non è riconoscibile da una MT a nastro singolo con complessità temporale inferiore a $\Theta(n^2)$).
b) Falsa: i linguaggi definibili mediante formule MFO sono una sottoclasse dei linguaggi regolari, quindi si possono riconoscere usando memoria finita. Data una MT a nastro singolo che riconosce un linguaggio, ad esempio comportandosi mossa per mossa come un automa a stati finiti, si può scrivere una RAM che esegue un'istruzione per ogni mossa della MT, sempre usando un numero finito di celle; ogni mossa costa un tempo O(1) anche a criterio di costo logaritmico perché i dati in ingresso sono caratteri di un alfabeto finito.

Esercizio 2

Può convenire usare una hash table. In primo luogo, si inseriscano gli n elementi di A in una hash table. Successivamente, per ogni elemento A[i] di A, cerchiamo X-A[i] nella tabella. Se si trova un j tale che A[j] = X-A[i], la coppia <i, j> è una soluzione del problema.
Assumendo una dimensione m della tabella hash non troppo piccola rispetto ad n, ad esempio con riempimento $\alpha = ½$ nel caso di indirizzamento aperto, o anche leggermente superiore nel caso di tabella a liste concatenate, una singola operazione sia di inserimento che di ricerca costa mediamente O(k), con k "vicino" all'unità a seconda dei vari casi. Occorrendo n inserimenti e al più n ricerche (mediamente un po' meno se la probabilità che esista la coppia cercata non è troppo bassa), il tempo medio di esecuzione complessiva sarà dunque O(n).

Esercizio 3

Una soluzione elementare consiste nel provare tutte le n*m coppie di elementi e, per ciascuna di esse, verificare se soddisfino la relazione richiesta, tenendo traccia del massimo elemento di A con queste caratteristiche. La complessità risultante è O(n*m).
Una soluzione più efficiente è la seguente:

- Riordinare A e B per valori decrescenti
- Scandire linearmente gli array ordinati, con elemento corrente a di A e b di B:
  - Se a=f(b) stop
  - Se a>f(b), passa al successivo elemento di A
  - Se a<f(b), passa al successivo elemento di B.
- L'algoritmo termina senza successo se si scandiscono entrambi gli array senza aver trovato la coppia desiderata.

La complessità risultante è O(n log n + m log m + n + m) = O(p log p), dove p è il più grande tra n e m.
Si può ottenere una versione leggermente più efficiente, a parità di ordine di grandezza, nel caso pessimo, osservando che non è necessario ordinare entrambi gli array. Infatti, se n < m, si ordina solo A, con costo O(n log(n)), e successivamente, per ogni elemento b di B, si cerca f(b) in A con una ricerca binaria, tenendo traccia dell'elemento b per cui viene trovato il valore massimo di a, con costo O(m log(n)).
Alternativamente, se m < n, allora si ordina solo B, con costo O(m log(m)), e successivamente, per ogni elemento a di A, si cerca se c'è un b in B tale per cui a = f(b) con una ricerca binaria in cui si confrontano a e f(b) ad ogni iterazione, tenendo traccia del valore a massimo per cui si trova un elemento b in B, con costo O(n log(m)).
In questo caso, la complessità diventa O((m+n) log(q)) = O(p log(q)), dove p = max(n,m) e q=min(n,m).