Algoritmi e Strutture Dati
07/01/13

Esercizio 1

Un modo semplice è il seguente: si ordina A, si ordina B e poi con una procedura simile alla merge() di mergesort() si verifica se ci sono valori uguali. Il costo è pari a $O(m \log m + n \log n + m + n) = O(m \log m + n \log n)$.

Esercizio 2

E' possibile risolvere il problema in tempo $O(n)$ sommando gli estremi del vettore $S[1]$ e $S[n]$, e notando che se la loro somma $S[1]+S[n]$ è maggiore di x, è possibile scartare l'elemento $S[n]$, perchè sommato a qualunque altro valore nel vettore ordinato S darà un valore ancora più alto di $S[1] + S[n]$. Se invece $S[1] + S[n] < x$, allora è possibile scartare $S[1]$, perchè sommato a qualunque altro valore nel vettore ordinato S darà un valore ancora più basso di $S[1] + S[n]$. Eliminando un estremo alla volta, si giunge o a due valori la cui somma è pari a x, o si esauriscono tutti i valori.

```
boolean searchSum(integer[] S, integer n, integer x)
    integer i ← 1
    integer j ← n
    while i < j do
        if S[i] + S[j] = x then
            return true
        if S[i] + S[j] > x then
            j ← j − 1
        else
            i ← i − 1
    return false
```

Esercizio 3

E' possibile risolvere il problema tramite programmazione dinamica. Sia *count*$[i, j]$ il numero di percorsi che partono dalla casella $[i, j]$ e arrivano a destinazione. Il risultato finale si trova in $[1, 1]$. *count* può essere calcolato come la somma dei percorsi che partono dalla casella $[i + 1, j]$ (se esiste) e dalla casella $[i, j + 1]$ (se esiste), moltiplicato per $B[i, j]$ (in modo che se la casella non può essere attraversata, nessun percorso viene calcolato). Il caso base è in $[m, n]$, il cui valore è pari a 1 se $B[m, n] = 1$, 0 altrimenti.

$$\textit{count}[i,j] = \begin{cases} B[i,j] & i = m \wedge j = n \\ 0 & i = m+1 \vee j = n+1 \\ (\textit{count}[i+1,j] + \textit{count}[i,j+1]) \cdot B[i,j] & \text{otherwise} \end{cases}$$

Il numero di percorsi può essere calcolato nel modo seguente:

```
integer contaPercorsi(integer[][] B, integer m, integer n)
    integer[][] count ← new integer[1 . . . m, 1 . . . n]
    count[m, n] ← B[m, n]
    for i = m − 1 downto 1 do
        count[i, n] ← B[i, n] · count[i + 1, n]
    for j = n − 1 downto 1 do
        count[m, j] ← B[m, j] · count[m, j + 1]
    for i = m − 1 downto 1 do
        for j = n − 1 downto 1 do
            count[i, j] ← B[i, j] · (count[i, j + 1] + count[i + 1, j])
    return count[1, 1]
```

Il costo computazionale dell'algoritmo è $O(mn)$.

Esercizio 4

Bisogna affrontare i due casi (pari e dispari) separatamente; utilizzando i teoremi visti a lezione, è facile vedere che $T(n) = \Theta(n)$ nel caso delle potenze di 2, mentre $T(n) = \Theta(n^2)$ nel caso dei numeri dispari (infatti, nel caso delle potenze di 2 si applica solo la divisione per 2, che produce ancora una potenza di 2, mentre nel caso dei numeri dispari si applica solo la sottrazione di 2, che produce ancora un numero dispari).
Cercando un limite inferiore e superiore, è facile quindi capire che un limite inferiore per la ricorrenza è pari a $\Omega(n)$, mentre un limite superiore è $O(n^2)$.
Dimostriamo per sostituzione il limite superiore: vogliamo dimostrare che $\exists c > 0, \exists m \geq 0$ tale che $T(n) \leq cn^2$ per ogni $n \geq m$.
Se n è pari,

$$\begin{aligned} T(n) &= T(n/2) + n \\ &\leq cn^2/4 + n \\ &\leq cn^2 \end{aligned}$$

che è vera per $c \geq 4/3n$, che è soddisfacibile per $c = 2/3$ con $m = 2$.
Se n è dispari,

$$\begin{aligned} T(n) &= T(n-2) + n \\ &\leq c(n-2)^2 + 1 \\ &\leq cn^2 \end{aligned}$$

che è vera per $c \geq 1/4$ con $m = 2$.
Il caso base è vero per $T(n) = 1 \geq 1 \cdot c$, ovvero per $c \geq 1$. Quindi è banalmente dimostrato che per $c = 1$, $m = 2$, $T(n) \leq cn$ per ogni $n \geq m$.
La dimostrazione per il limite inferiore è analoga.