

# Algoritmi e Strutture Dati

17/06/2013

## Esercizio 1

La ricorrenza associata a questa procedura ricorsiva è la seguente:

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + T(\lfloor \sqrt{n} \rfloor) + n & n > 1 \end{cases}$$

E' possibile dimostrare che  $T(n) = O(n^2)$  tramite sostituzione:

- Caso base:  $n = 1, T(1) = 1 \leq cn^2 = c$ , che è vero per  $c \geq 1$ .
- Ipotesi induttiva:  $T(n') \leq cn^2$  per ogni  $n' < n$
- Passo induttivo:

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

L'ultima disequazione è vera se  $c \geq \frac{n}{n-1}$ . Si noti che la funzione  $\frac{n}{n-1}$  tende a 1 per  $n$  che tende all'infinito, decrescendo dal valore 2 per  $n = 2$ . Quindi per tutti gli  $n \geq 2$ ,  $c$  deve essere maggiore di 2.

Quindi, per soddisfare sia il caso base che il passo induttivo,  $c \geq 2$ .

Per quanto riguarda  $T(n) = \Omega(n^2)$ , è sufficiente notare che  $T(n) = T(n-1) + T(\lfloor \sqrt{n} \rfloor) + n \geq T(n-1) + n$ , che per il teorema delle ricorrenze lineari di ordine costante ha complessità  $\Omega(n^2)$ .

## Esercizio 2

E' possibile identificare l'intervallo dei valori che devono essere compresi in ognuno dei sottovettori utilizzando l'algoritmo di selezione e sfruttando il fatto che gli interi sono distinti. Infatti, gli elementi che si troverebbero in posizione  $n, 2n, 3n, 4n$  se il vettore fosse ordinato, identificano gli elementi confine di ognuno dei sottovettori. Visto che è possibile utilizzare un algoritmo di selezione di costo lineare, è possibile ottenere un algoritmo di complessità  $O(n)$  come segue:

---

```
splitfour(integer[] V, integer n, integer[][] B)
```

---

```
integer bounds ← new bounds[1...4]
for k ← 1 to 4 do
    bounds[k] ← select(A, k · n)
integer pos ← new pos[1...4]
for k ← 1 to 4 do pos[k] = 1

for i ← 1 to 4n do
    for k ← 1 to 4 do
        if V[i] ≤ bounds[k] then
            B[k, pos[k]] ← V[i]
            pos[k] ← pos[k] + 1
            break
```

---

## Esercizio 3

Il problema può essere facilmente risolto tramite programmazione dinamica, avendo l'accortezza di evitare di selezionare due vicini consecutivi. Detto  $M[i]$  la quantità massima che può essere raccolta dai primi  $i$  abitanti, è possibile esprimere  $M[i]$  in maniera ricorsiva come segue:

$$M[i] = \begin{cases} D[1] & i = 1 \\ \max\{D[1], D[2]\} & i = 2 \\ \max\{M[i-1], M[i-2] + D[i]\} & i > 2 \end{cases}$$

Una versione basata su programmazione dinamica può essere scritta come segue:

---

```

integer fundraising(integer[] D, integer n)
integer[] M ← new integer[1 ... n]                                % Calcola il vettore M
M[1] ← D[1]
M[2] ← max(D[1], D[2])
for i ← 3 to n do
    M[i] ← max(D[i-1], D[i-2] + D[i])
integer i ← n                                              % Stampa gli indici selezionati

while i > 2 do
    if M[i] = M[i-2] + D[i] then
        print i
        i ← i - 2
    else
        i ← i - 1
if i > 0 then
    print i
return M[n]                                              % Ritorna la quantità massima raccoglitibile

```

---

La complessità dell'algoritmo è banalmente  $\Theta(n)$

#### Esercizio 4

Per risolvere questo problema, è sufficiente utilizzare il grafo  $G$  come una rete di flusso, utilizzando  $u$  come sorgente e  $v$  come pozzo, e associando il peso 1 a tutti gli archi. Non appena si raggiunge un flusso grande almeno  $k$ , si ritorna true. Se il flusso totale è inferiore a  $k$ , si ritorna false.

La complessità di questo algoritmo è pari a  $O(k(m + n))$ , con  $k = O(n)$ .