Trova la matrice

Dati $2n$ interi non negativi $c_1, c_2, \ldots, c_n$, $r_1, r_2, \ldots, r_n$, si vuole determinare una matrice $n \times n$ con elementi interi non negativi tali che la somma degli elementi della colonna i-esima sia uguale a c_i e la somma degli elementi della riga i-esima sia uguale ad r_i. Quali condizioni devono valere su $c_1, \ldots, c_n$, $r_1, \ldots, r_n$ affinché esista una soluzione? Si formuli il problema come un problema di flusso massimo. Si fornisca poi un algoritmo di complessità $O(n^2)$ senza usare il flusso massimo.

Quadrato binario

Sia $A[1 \ldots n, 1 \ldots n]$ una matrice di valori booleani 0/1. Scrivere un algoritmo che restituisce la dimensione del più grande quadrato composto da valori 1. Ad esempio, nella matrice seguente, i quadrati di dimensione massima (ve ne sono due, di cui uno evidenziato in rosso) hanno dimensione pari a 4.

```
1 0 1 0 1 0 0
1 0 1 1 1 1 0
0 1 1 1 1 1 0
0 0 1 1 1 1 0
1 1 1 1 1 1 0
1 1 1 1 1 1 0
```

Costo partizione vettore $V[1 \ldots mn]$

Costo $C(i,j)$ di un sottovettore $V[i \ldots j]$ di V $C(i,j) = \sum_{t=i}^{j} V[t]$
k-partizione di V è una divisione di V in k sottovettori $V[1, j_1], V[j_1 + 1, j_2], V[j_2 + 1, j_3], \ldots, V[j_{k-1} + 1, j_k]$ con $j_k = n$ e $j_t < j_{t+1}, \forall 1 \leq t < k$, ovvero tale per cui i sottovettori coprono totalmente il vettore e non si sovrappongono.
Costo della partizione è il costo massimo dei suoi sottovettori.
Dato un vettore V di n interi e un intero k,con $2 \leq k \leq n$, trovare una k-partizione di V di costo minimo
Esempio: $V = \{2, 3, 7, -7, 15, 2\}$, $k = 3$
1: $\{2, 3, 7\}, \{-7, 15\}, \{2\}$, costo $2 + 3 + 7 = 12$
2: $\{2, 3\}, \{7\}\{-7, 15, 2\}$, costo $-7 + 15 + 2 = 10$

1. Soluzione per $k = 2$ (suggerimento: $O(n)$).
2. Soluzione per $k = 3$ (suggerimento: $O(n^2)$).
3. Soluzione generale (suggerimento: $O(kn^2)$).

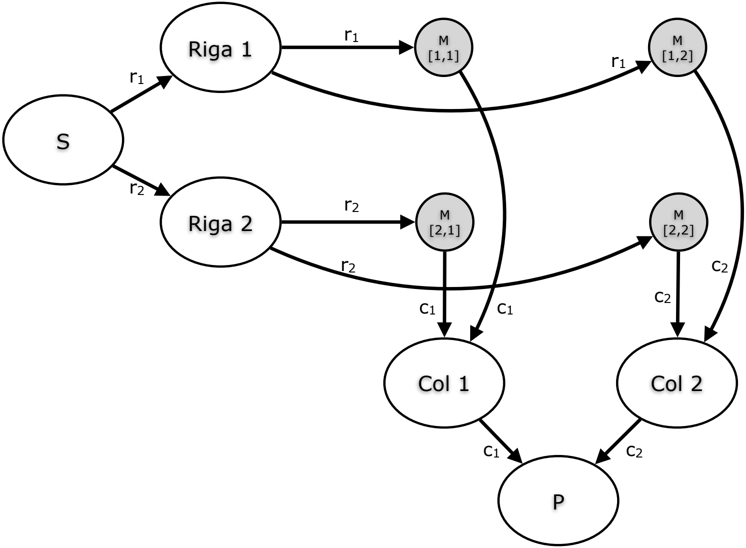
Fiumi e porti

Lungo un fiume ci sono n porti. A ciascuno di questi porti è possibile affittare una barca che può essere restituita ad un altro porto. E' praticamente impossibile andare controcorrente. Il costo dell'affitto di una barca da un punto di partenza i ad un punto di arrivo j, con $i < j$, è denotato con $C[i, j]$. E' possibile che per andare da i a j sia più economico effettuare alcune soste e cambiare la barca piuttosto che affittare un'unica barca. Se si affitta una barca in $k_1, k_2, k_3, \ldots, k_l$ (con $k_1 = 1, k_1 < k_2 < k_3 < k_l$) allora il costo totale è
$C[k_1, k_2] + C[k_2, k_3] + \ldots + C[k_{l-1}, k_l] + C[k_l, n]$.
Scrivere un algoritmo che dato in input i costi $C[i, j]$, determini il costo minimo per recarsi da 1 ad n. Analizzare la complessità computazionale dell'algoritmo proposto. Per un punteggio aggiuntivo, si stampino i porti in cui devono essere noleggiate le barche.

D20

Dati n dadi, con il dado i-esimo dotato di $F[i]$ facce numerate da 1 a $F[i]$, trovare il numero di modi diversi con cui è possibile ottenere una certa somma X sommando i valori di tutti i dadi. Ad esempio, avendo due dadi a quattro facce numerati da 1 a 4, il valore 7 è ottenibile in un solo modo non contando le possibili permutazioni: $3 + 4$. Avendo tre dadi sempre a 4 facce, il valore 6 è ottenibile in tre modi diversi non contando le possibili permutazioni: $1 + 1 + 4$, $1 + 2 + 3$, $2 + 2 + 2$.

Trova la matrice



Trova la matrice

```
integer[][] riempi(integer[] r, integer[] c, integer n)
integer[][]M ← new integer[1...n, 1...n]
for i ← 1 to n do
    for j ← 1 to n do
        M[i, j] ← 0
i ← j ← n
while i > 0 and j > 0 do
    if r[i] < c[j] then
        M[i, j] ← r[i]
        c[j] ← c[j] − r[i]
        i ← i − 1
    else
        M[i, j] ← c[j]
        r[i] ← r[i] − c[j]
        j ← j − 1
    return M
```

Quadrato binario

$$M[i,j] = \begin{cases} 0 & A[i,j] = \textbf{false} \\ 1 & A[i,j] = \textbf{true} \wedge i = n \vee j = n \\ \min\{A[i+1,j], & \\ \qquad A[i+1,j+1], & \\ \qquad A[i,j+1]\} + 1 & \text{altrimenti} \end{cases}$$

Quadrato binario

integer maxSquare(**boolean**[ ][ ] A, **integer** n)

```
integer[][] M ← new integer[1...n][1...n]
for i ← 1 to n do
    M[i, n] ← A[i, n]
    M[n, i] ← A[n, i]
for i ← n − 1 downto 1 do
    for j ← n − 1 downto 1 do
        if A[i, j] = 0 then
            M[i, j] ← 0
        else
            M[i, j] ← min(A[i + 1, j], A[i + 1, j + 1], A[i, j + 1]) + 1
return max(A, n)    % Ritorna il massimo valore della matrice, O(n²)
```

2-partizione

```
2-partition(integer[] V, integer n)
integer tot ← 0
for i ← 1 to n do
    tot ← tot + V[i]
integer sofar ← 0
integer min ← +∞
for i ← 1 to n − 1 do
    sofar ← sofar + V[i]
    min ← min(min, max(sofar, tot − sofar))
return min
```

```
integer 3-partition(integer[] V, integer n)

integer[][] T ← new integer[0...n]
T[0] ← 0
for i ← 1 to n do
    T[i] ← T[i − 1] + V[i]
integer min ← +∞
for i ← 1 to n − 2 do
    for j ← i + 1 to n − 1 do
        integer temp ← max(T[i], T[j] − T[i], T[n] − T[j])
        min ← min(min, temp)
return min
```

k-partizione

Sia $M[i,t]$ il minimo costo associato al sottoproblema di trovare la migliore t-partizione nel vettore $V[1\ldots i]$. Il problema iniziale corrisponde a $M[n,k]$ – ovvero trovare la migliore k-partizione in $V[1\ldots n]$. Sfruttiamo un vettore di appoggio T definito come nel caso $k=3$.

$$M[i,t] = \begin{cases} T[i] & t = 1 \\ +\infty & t > i \\ \min_{1\leq j<i} \max(M[j,t-1], T[i]-T[j]) & \text{altrimenti} \end{cases}$$

k-partizione

```
integer partition-rec(integer[] V, integer[] T, integer[][] M, integer i, integer t)
if t > i then return +∞
if t = 1 then return T[i]
if M[i, t] = ⊥ then
    integer M[i, t] ← +∞
    for j ← 1 to i − 1 do
        integer temp ← max(partition-rec(V, T, M, j, t − 1), T[i] − T[j])
        if temp < M[i, t] then M[i, t] ← temp
return M[i, t]
```

Fiumi e porti (1)

$$best[i,j] = \begin{cases} C[i,j] & j = i+1 \\ \min\{C[i,j], \min_{i<k<j}\{best[i,k] + best[k,j]\}\} \end{cases}$$

```
boat(integer[][] C, integer n, integer[][] best, integer i, j)
if j = i + 1 then
    return C[i, j]
if best[i, j] = ⊥ then
    best[i, j] ← C[i, j]
    for k = i + 1 to j − 1 do
        if best[i, k] + best[k, j] < C[i, j] then
            best[i, j] ← best[i, k] + best[k, j]
return best[i, j]
```

Fiumi e porti (2)

$$best[i] = \begin{cases} 0 & i = n \\ \min_{i<j\leq n}\{C[i,j] + best[j]\} \end{cases}$$

```
integer boat(integer[][] C, integer n)
integer[] best ← new integer[1 . . . n]
best[n] ← 0
for i ← n − 1 downto 1 do
    best[i] ← +∞
    for j ← i + 1 to n do
        if C[i, j] + best[j] < best[i] then
            best[i] ← C[i, j] + best[j]
return best[1]
```