

PIERLUIGI CRESCENZI

# 50 ESERCIZI JAVA--

# Istruzioni



## *Come svolgere un esercizio*

Facciamo riferimento all'esercizio Party.

- Scaricare il file `party.zip` disponibile sul sito web del corso, senza decomprimerlo.
- Avviare Eclipse JavaMM SDK e, se necessario, creare un workspace.
- Selezionare la voce `Import...` dal menu `File`.
- Selezionare la voce `Existing Projects into Workspace` all'interno della cartella `General`.
- Premere `Next>`.
- Selezionare il file `party.zip` (precedentemente scaricato) all'interno della casella `Select archive file:`.
- Premere `Finish`.
- Il progetto `Party.student` appare nel `Package Explorer`.
- Modificare il metodo `party` che è incluso nella cartella `javamm` all'interno della cartella `src` del progetto `Party.student`, in modo da risolvere l'esercizio come specificato nel testo dell'esercizio stesso (altri metodi possono, e probabilmente devono, essere dichiarati e definiti).
- Una volta finita la scrittura del metodo `party` e degli eventuali altri metodi e verificata la sua correttezza con gli esempi forniti nel testo ed eventualmente con altri esempi, premere con il tasto destro sul progetto `Party.student` e selezionare la voce `JUnit Test` all'interno della voce `Run As`.
- Se il test viene eseguito senza errori, appare una barra verde.
- Se il test viene eseguito con errori, appare una barra rossa e l'elenco di tutti gli esempi in cui il test ha fallito.

# Party

## Definizione del problema

Quando gli studenti vanno a una festa, gli piace bere qualche *drink*. Per le leggi in vigore, è possibile bere a una festa non più di 40 centilitri di alcol. Se la festa si svolge il sabato sera, allora è possibile bere fino a 100 centilitri di alcol. Il problema consiste nel decidere se sia possibile bere una certa quantità di alcol a una festa, a seconda del giorno in cui si svolge la festa.



## Esercizio

Scrivere un metodo chiamato `party` che, dati in input un numero `c` di tipo `int` e un valore logico `s` di tipo `boolean`, restituisca il valore `true` se e solo se sia possibile bere `c` centilitri di alcol a una festa che si svolge di sabato se e solo se `s` è vero.

## Esempi

- `c = 30` e `s = false`: `true`
- `c = 50` e `s = false`: `false`
- `c = 70` e `s = true`: `true`

**Parte I**

# **Espressioni Booleane**



# *Six*

## *Esercizio*

Scrivere un metodo chiamato `six` che, dati in input due numeri `a` e `b` di tipo `int`, restituisca il valore `true` se e solo uno dei due numeri sia uguale a 6 o se la loro somma o la loro differenza sia uguale a 6.

## *Esempi*

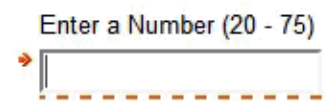
- `a = 6 e b = 4: true`
- `a = 4 e b = 5: false`
- `a = 1 e b = 5: true`



# Range

## Definizione del problema

Capita spesso di dover chiedere a un utente di inserire un numero all'interno di uno specifico intervallo. Dato un intervallo  $[l, r]$  dove  $l$  e  $r$  sono due numeri interi, vogliamo sapere se un numero  $n$  appartiene a tale intervallo. Se siamo in modalità *out*, invece, vogliamo sapere se  $n$  è fuori dell'intervallo.



## Esercizio

Scrivere un metodo chiamato `range` che, dati in input tre numeri  $n$ ,  $l$  e  $r$  di tipo `int` e dato un valore logico  $o$  di tipo `boolean`, restituisca il valore `true` se e solo se  $n$  è incluso (rispettivamente, non incluso) nell'intervallo  $[l, r]$  se  $o = \text{false}$  (rispettivamente,  $o = \text{true}$ ).

## Esempi

- $n = 5, l = 1, r = 10$  e  $o = \text{false}$ : `true`
- $n = 11, l = 1, r = 10$  e  $o = \text{false}$ : `false`
- $n = 11, l = 1, r = 10$  e  $o = \text{true}$ : `true`

# *Last digit*

## *Esercizio*

Scrivere un metodo chiamato `lastDigit` che, dati in input tre numeri non negativi `a`, `b` e `c` di tipo `int`, restituisca il valore `true` se e solo se almeno due di essi terminano con la stessa cifra decimale.

## *Esempi*

- `a = 23, b = 19 e c = 13: true`
- `a = 23, b = 19 e c = 12: false`
- `a = 23, b = 19 e c = 3: true`

## *Is sum*

### *Esercizio*

Scrivere un metodo chiamato `isSum` che, dati in input tre numeri `a`, `b` e `c` di tipo `int`, restituisca il valore `true` se e solo se uno dei tre numeri sia uguale alla somma degli altri due.

### *Esempi*

- `a = 1, b = 2 e c = 3: true`
- `a = 3, b = 1 e c = 2: true`
- `a = 3, b = 2 e c = 2: false`

# *Order*

## *Esercizio*

Scrivere un metodo chiamato `order` che, dati in input tre numeri `a`, `b` e `c` di tipo `int` e dato un valore logico `noA` di tipo `boolean`, restituisca il valore `true` se e solo se `a` è più piccolo di `b` e `b` è più piccolo di `c`. Nel caso in cui `noA` sia vero, il metodo deve restituire il valore `true` se e solo se `b` è più piccolo di `c`.

## *Esempi*

- `a = 1, b = 2, c = 4` e `noA = false`: `true`
- `a = 1, b = 2, c = 1` e `noA = false`: `false`
- `a = 1, b = 1, c = 2` e `noA = true`: `true`

## **Parte II**

### **Istruzioni `if` e `if-else`**



# *Lone sum*

## *Esercizio*

Scrivere un metodo chiamato `loneSum` che, dati in input tre numeri positivi `a`, `b` e `c` di tipo `int`, restituisca la somma dei numeri tra di essi che sono diversi dagli altri due.

## *Esempi*

- `a = 1, b = 2 e c = 3: 6`
- `a = 3, b = 2 e c = 3: 2`
- `a = 3, b = 3 e c = 3: 0`

# *No teen sum*

## *Esercizio*

Scrivere un metodo chiamato `noTeenSum` che, dati in input tre numeri positivi `a`, `b` e `c` di tipo `int`, restituisca la somma dei numeri tra di essi che non sono numeri *teen*: un numero è *teen* se è incluso nell'intervallo `[13, 19]` ed è diverso da 15 e da 16. Per evitare la duplicazione di codice, scrivere un metodo `fixTeen` che, dato in input un numero positivo `a` di tipo `int`, restituisca lo stesso numero se non è *teen*, altrimenti restituisca 0. Utilizzare poi il metodo `fixTeen` all'interno del metodo `noTeenSum`.

## *Esempi*

- `a = 1, b = 2 e c = 3: 6`
- `a = 2, b = 13 e c = 1: 3`
- `a = 2, b = 1 e c = 14: 3`
- `a = 2, b = 1 e c = 15: 18`



# *Round sum*

## *Esercizio*

Scrivere un metodo chiamato `roundSum` che, dati in input tre numeri positivi `a`, `b` e `c` di tipo `int`, restituisca la somma dei tre numeri arrotondati al multiplo di 10 più vicino: l'arrotondamento di un numero positivo è per difetto se l'ultima cifra è minore di 5, per eccesso altrimenti. Per evitare la duplicazione di codice, scrivere un metodo `round` che, dato in input un numero positivo `a` di tipo `int`, restituisca il suo arrotondamento. Utilizzare poi il metodo `round` all'interno del metodo `roundSum`.

## *Esempi*

- `a = 16, b = 17 e c = 18: 60`
- `a = 12, b = 13 e c = 14: 30`
- `a = 6, b = 4 e c = 4: 10`
- `a = 4, b = 4 e c = 4: 0`

## *Evenly spaced*

### *Esercizio*

Scrivere un metodo chiamato `evenlySpaced` che, dati in input tre numeri positivi `a`, `b` e `c` di tipo `int`, restituisca `true` se e solo se la differenza tra il valore di mezzo e quello minimo sia uguale alla differenza tra il valore massimo e quello di mezzo. Per semplificare la scrittura del codice, scrivere un metodo `min` (rispettivamente, `mid` e `max`) che, dati in input tre numeri positivi `a`, `b` e `c` di tipo `int`, restituisca il valore minimo (rispettivamente, di mezzo e massimo). Utilizzare poi i metodi `min`, `mid` e `max` all'interno del metodo `evenlySpaced`.

### *Esempi*

- `a = 2, b = 4 e c = 6: true`
- `a = 4, b = 6 e c = 2: true`
- `a = 4, b = 6 e c = 3: false`

# Make wall

## Esercizio

Scrivere un metodo chiamato `isWallDoable` che, dati in input tre numeri positivi  $s$ ,  $b$  e  $g$  di tipo `int`, restituisca il valore `true` se e solo se esistono due numeri  $0 \leq s' \leq s$  e  $0 \leq b' \leq b$  tali che  $s' + 5b' = g$ . Il metodo non deve fare uso di strutture di ripetizione.

## Esempi

- $s = 3, b = 1$  e  $g = 8$ : `true`
- $s = 3, b = 1$  e  $g = 9$ : `false`
- $s = 3, b = 2$  e  $g = 10$ : `true`
- $s = 3, b = 3$  e  $g = 14$ : `false`
- $s = 2, b = 10$  e  $g = 48$ : `false`

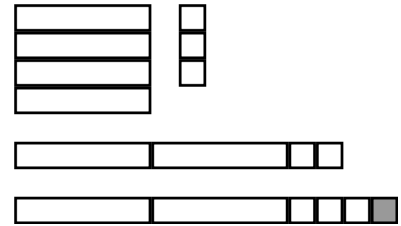


Figura 1: Se  $b = 4$  e  $s = 3$ , possiamo costruire un muro lungo  $g = 12$  (scegliendo  $b' = s' = 2$ ), ma non un muro lungo  $g = 14$  (in quanto  $b' \leq 2$  e  $s' \leq s = 3$ ).

## **Parte III**

### **Istruzione while**



# *Integer square root*

## *Esercizio*

Scrivere un metodo chiamato `integerSquareRoot` che, dato in input un numero positivo  $n$  di tipo `int`, restituisca la sua radice quadrata intera  $x$ , ovvero il massimo valore intero  $x$  tale che  $x \cdot x \leq n$ .

## *Esempi*

- $n = 1$ : 1
- $n = 2$ : 1
- $n = 15$ : 3
- $n = 16$ : 4

## *First last digit sum*

### *Esercizio*

Scrivere un metodo chiamato `firstLastDigitSum` che, dato in input un numero `n` di tipo `int` maggiore o uguale a 10, restituisca la somma della sua cifra più significativa e della sua cifra meno significativa.

### *Esempi*

- `n = 10`: 1
- `n = 11`: 2
- `n = 23456`: 8
- `n = 95643`: 12

## *Invert number*

### *Esercizio*

Scrivere un metodo chiamato `invertNumber` che, dato in input un numero positivo `n` di tipo `int` la cui cifra meno significativa sia maggiore di 0, restituisca il numero intero ottenuto invertendo l'ordine delle sue cifre.

### *Esempi*

- `n = 1`: 1
- `n = 11`: 11
- `n = 23456`: 65432
- `n = 95643`: 34659



# *Harmonic sum*

## *Esercizio*

Per ogni  $k \geq 1$ , il numero armonico  $H(k)$  è definito come

$$H(k) = \sum_{i=1}^k \frac{1}{i}.$$

Scrivere un metodo chiamato `harmonicSum` che, dato in input un numero positivo  $x$  di tipo `int`, restituisca il più piccolo numero intero  $n$  tale che

$$H(1) + H(2) + \cdots + H(n) \geq x.$$

## *Esempi*

- $x = 1$ : 1
- $x = 2$ : 2
- $x = 3$ : 3
- $x = 5$ : 4
- $x = 7$ : 5
- $x = 1000$ : 204

# *Duplicate digit*

## *Esercizio*

Scrivere un metodo chiamato `duplicateDigit` che, dato in input un numero positivo `n` di tipo `int`, restituisca il numero intero (di tipo `long`) ottenuto duplicando ogni sua cifra (ovvero scrivendo ogni sua cifra due volte).

## *Esempi*

- `n = 1`: 11
- `n = 12`: 1122
- `n = 21`: 2211
- `n = 210`: 221100
- `n = 95043`: 9955004433

**Parte IV**

**Istruzione for**



# *Scalar product*

## *Esercizio*

Il prodotto scalare di due array  $a$  e  $b$  di  $n$  numeri interi ciascuno è definito come

$$a \cdot b = \sum_{i=0}^{n-1} a[i]b[i].$$

Scrivere un metodo chiamato `scalarProduct` che, dati in input due array non vuoti  $a$  e  $b$  con lo stesso numero di elementi di tipo `int`, restituisca il loro prodotto scalare.

## *Esempi*

- $a = \{3, 4, -2\}$  e  $b = \{-1, 5, 3\}$ : 11
- $a = \{1, 2, 3, 4, 5\}$  e  $b = \{1, 1, 1, 1, 1\}$ : 15
- $a = \{1, 2, 3, 4, 5\}$  e  $b = \{0, 0, 0, 0, 0\}$ : 0
- $a = \{7\}$  e  $b = \{-4\}$ : -28

# *Array sum*

## *Esercizio*

La somma di due array  $a$  e  $b$  di  $n$  numeri interi ciascuno è definito come l'array  $c$  tale che, per ogni  $i$  con  $0 \leq i \leq n - 1$ ,

$$c[i] = a[i] + b[i].$$

Scrivere un metodo chiamato `arraySum` che, dati in input due array non vuoti  $a$  e  $b$  con lo stesso numero di elementi di tipo `int`, restituisca la loro somma.

## *Esempi*

- $a = \{3, 4, -2\}$  e  $b = \{-1, 5, 3\}$ :  $\{2, 9, 1\}$
- $a = \{1, -2, 3, -4\}$  e  $b = \{1, -2, 3, -4\}$ :  $\{2, -4, 6, -8\}$
- $a = \{1, 2, 3, 4, 5\}$  e  $b = \{0, 0, 0, 0, 0\}$ :  $\{1, 2, 3, 4, 5\}$

# *Binary to decimal*

## *Esercizio*

Dato un array  $a$  di  $n$  cifre binarie, il corrispondente numero decimale  $d$  è definito come

$$d = a[n-1] + 2a[n-2] + 2^2a[n-3] + \dots + 2^{n-2}a[1] + 2^{n-1}a[0].$$

Scrivere un metodo chiamato `decimal` che, dato in input un array non vuoto `a` di cifre binarie di tipo `int`, restituisca il corrispondente numero decimale di tipo `long`.

## *Esempi*

- `a = {0,0,0,0}`: 0
- `a = {0,1,1,1}`: 7
- `a = {1,0,0,0}`: 8
- `a = {1,1,1,1}`: 15

## *To upper case*

### *Esercizio*

Scrivere un metodo chiamato `upperCase` che, dato in input un array non vuoto `a` di caratteri alfabetici di tipo `char`, restituisca il corrispondente array con tutti i caratteri in maiuscolo.

### *Esempi*

- `a = {'r','o','m','a'}: {'R','O','M','A'}`
- `a = {'r','O','m','a'}: {'R','O','M','A'}`
- `a = {'R','O','M','A'}: {'R','O','M','A'}`



## *Palindrome array*

### *Esercizio*

Un array di numeri interi  $a$  è palindromo se letto da sinistra verso destra è uguale a letto da destra verso sinistra. Scrivere un metodo chiamato `palindrome` che, dato in input un array non vuoto  $a$  di numeri interi di tipo `int`, restituisca il valore `true` se e solo se  $a$  è palindromo.

### *Esempi*

- $a = \{4, 2, 5, 9, 9, 5, 2, 4\}$ : `true`
- $a = \{3, -8, 1, -8, 3\}$ : `true`
- $a = \{1, -9, 4, 3, 3, 4, 9, 1\}$ : `false`
- $a = \{1, 2, 3, 3, 2, 1, 1\}$ : `false`

**Parte V**

**Programmazione  
procedurale**



# Over average

## Esercizio

Scrivere un metodo, chiamato `overAverage`, che dato in input un array `a` di interi positivi restituisca l'array di tutti gli elementi di `a` maggiori della media degli elementi di `a`, nello stesso ordine con cui appaiono in `a`.

$a = \{1, 6, 5, 8, 6\}$

Media: 5.2

Array degli elementi sopra la media

$\{6, 8, 6\}$

## Esempi

- $a = \{1, 6, 5, 8, 6\}$ :  $\{6, 8, 6\}$
- $a = \{2, 5, 4, 3, 6\}$ :  $\{5, 6\}$
- $a = \{1, 1, 2, 1, 1\}$ :  $\{2\}$
- $a = \{1, 1, 1, 1, 1\}$ :  $\{\}$

# Maximum number

## Esercizio

Scrivere un metodo, chiamato `maximumNumber`, che, dato in input un array  $a$  di  $n$  cifre decimali positive e un numero  $k$  positivo minore o uguale di  $n$ , restituisca il massimo numero intero di  $k$  cifre prese dall'array  $a$  nell'ordine in cui esse appaiono in  $a$ .

$a = \{4, 3, 1, 9, 5\}$  e  $k = 3$   
Massimo numero: 495

## Esempi

- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 1$ : 9
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 2$ : 98
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 3$ : 987
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 4$ : 9875
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 5$ : 98175
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 6$ : 958175
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 7$ : 93581759
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 8$ : 79358175
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 9$ : 749358175
- $a = \{7, 4, 2, 9, 3, 5, 8, 1, 7, 5\}$  e  $k = 10$ : 7429358175

# Anagrams

## Esercizio

Due array  $s$  e  $t$  di numeri interi positivi sono uno un *anagramma* dell'altro se contengono gli stessi numeri, con le stesse molteplicità.

Per esempio, l'array  $\{12, 1, 45, 5, 3, 3, 8, 1\}$  è un anagramma dell'array  $\{3, 3, 12, 1, 1, 8, 45, 5\}$ , mentre non è un anagramma dell'array  $\{3, 3, 12, 1, 8, 45, 5\}$  (in quanto manca, nel secondo array, un 1).

Un algoritmo efficiente per verificare se due array  $s$  e  $t$  sono uno un anagramma dell'altro consiste nel copiare i due array, ordinare le copie in ordine non decrescente e verificare se i due array ordinati sono uguali.

Scrivere un metodo, chiamato `anagram`, che dati in input due array  $s$  e  $t$  di numeri interi positivi, restituisca il valore `true` se e solo se  $s$  è un anagramma di  $t$ , facendo uso dell'algoritmo appena descritto.

$$s = \{12, 1, 45, 5, 3, 3, 8, 1\}$$
$$t = \{3, 3, 12, 1, 1, 8, 45, 5\}$$

Copie degli array ordinate

$$s = \{1, 1, 3, 3, 5, 8, 12, 45\}$$
$$t = \{1, 1, 3, 3, 5, 8, 12, 45\}$$

Le copie ordinate sono uguali

## Esempi

- $s = \{12, 1, 45, 5, 3, 3, 8, 1\}$  e  $t = \{3, 3, 12, 1, 1, 8, 45, 5\}$ : `true`
- $s = \{12, 1, 45, 5, 3, 3, 8, 1\}$  e  $t = \{3, 3, 12, 1, 8, 45, 5\}$ : `false`
- $s = \{12, 1, 45, 5, 3, 3, 8, 1\}$  e  $t = \{3, 3, 12, 1, 8, 45, 5, 8\}$ : `false`

# Armstrong

## Esercizio

Un numero intero positivo  $n$  è detto un *numero di Armstrong* se la somma delle sue cifre, ciascuna elevata alla potenza del numero di cifre di  $n$ , è uguale al numero stesso. Per esempio, il numero 153 è un numero di Armstrong in quanto  $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$ . Analogamente, il numero 1634 è un numero di Armstrong in quanto  $1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634$ . Invece, 372 non è un numero di Armstrong in quanto  $3^3 + 7^3 + 2^3 = 27 + 343 + 8 = 378 \neq 372$ .

Scrivere un metodo, chiamato `armstrong`, che dato in input un numero intero positivo  $n$ , restituisca il valore `true` se e solo se  $n$  è un numero di Armstrong.

## Esempi

- $n = 153$ : `true`
- $n = 1634$ : `true`
- $n = 372$ : `false`

# Amicable

## Esercizio

Un *divisore proprio* di un numero intero positivo  $n$  è un divisore di  $n$  diverso da  $n$  stesso: per esempio, i divisori propri di 6 sono 1, 2 e 3. Due numeri interi positivi  $n$  e  $m$  si dicono *amicabili* se la somma dei divisori propri di  $n$  è uguale a  $m$  e se la somma dei divisori propri di  $m$  è uguale a  $n$ .

Scrivere un metodo, chiamato `amicable`, che, dati in input due numeri  $a$  e  $b$  interi positivi, restituisca il valore `true` se e solo se  $a$  e  $b$  sono amicali.

## Esempi

- $a = 284$  e  $b = 220$ : `true`
- $a = 9$  e  $b = 15$ : `false`
- $a = 1184$  e  $b = 1210$ : `true`

$$n = 220 \text{ e } m = 284$$

Divisori propri di  $n$

1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110

Divisori propri di  $m$

1, 2, 4, 71, 142

Somma divisori propri di  $n$ : 284 Somma  
divisori propri di  $m$ : 220  $n$  e  $m$  sono  
amicabili

$$n = 15 \text{ e } m = 9$$

Divisori propri di  $n$

1, 3, 5

Divisori propri di  $m$

1, 3

Somma divisori propri di  $n$ : 9 Somma  
divisori propri di  $m$ : 4  $n$  e  $m$  non sono  
amicabili



# *Frequencies*

## *Esercizio*

Scrivere un metodo, chiamato `frequencies`, che, dato in input un array `a` di numeri interi non vuoto, restituisca l'array delle frequenze degli elementi distinti di `a`, in ordine crescente dei valori degli elementi stessi. Ad esempio, con input l'array `a`  $\{1,6,5,8,6,6,8,5,3,1,6,8\}$ , il metodo deve restituire l'array  $\{2,1,2,4,3\}$ , in quanto gli elementi distinti di `a` in ordine crescente sono 1, 3, 5, 6 e 8 e il valore 1 appare due volte, il 3 una volta, il 5 due volte, il 6 quattro volte e l'8 tre volte. L'array in input non deve essere modificato dall'esecuzione del metodo.

## *Esempi*

- `a = {1,6,5,8,6,6,8,5,3,1,6,8}`:  $\{2,1,2,4,3\}$
- `a = {6,6,6,6}`:  $\{4\}$
- `a = {3,5,6,2}`:  $\{1,1,1,1\}$

# Lexical comparator

## Esercizio

Dato un numero positivo  $x$ , indichiamo con  $d(x)$  il numero di cifre di  $x$ . Un numero intero positivo  $a$  è *lessicograficamente minore* di un numero intero positivo  $b$  se una delle seguenti due condizioni è verificata.

1. Esiste un  $0 \leq k < d(a)$  tale che le prime  $k$  cifre più significative di  $a$  sono uguali alle prime  $k$  cifre più significative di  $b$  e la  $(k + 1)$ -esima cifra più significativa di  $a$  è minore della  $(k + 1)$ -esima cifra più significativa di  $b$ .
2. Le prime  $d(a)$  cifre più significative di  $a$  sono uguali alle prime  $d(a)$  cifre più significative di  $b$  e  $d(b)$  è maggiore di  $d(a)$ .

Scrivere un metodo, chiamato `lexCompare`, che dati in input due numeri interi positivi  $a$  e  $b$ , restituisca il valore `true` se e solo se  $a$  è lessicograficamente minore di  $b$ .

## Esempi

- $a = 21$  e  $b = 22$ : `true`
- $a = 200$  e  $b = 22$ : `true`
- $a = 223$  e  $b = 22$ : `false`
- $a = 223$  e  $b = 223$ : `false`
- $a = 23471$  e  $b = 2358$ : `true`

156 è minore di 157

- $k = 2$ . Le prime 2 cifre più significative di 156 (ovvero 1 e 5) sono uguali alle prime 2 cifre più significative di 157 e la terza cifra più significativa di 156 (ovvero 6) è minore della terza cifra più significativa di 157 (ovvero 7).

23471 è minore di 234712

- Le prime 5 cifre più significative di 23471 sono uguali alle prime 5 cifre più significative di 234712 e 234712 ha 6 cifre mentre 23471 ha 5 cifre.

# H-index

## Esercizio

Dato un array  $a$  di  $n$  numeri interi non negativi, l'indice  $H$  di  $a$  è definito come il massimo valore  $h$ , compreso tra 1 ed  $n$ , tale che l' $h$ -esimo elemento di  $a$  in ordine non crescente sia non inferiore ad  $h$ . Ad esempio, l'indice  $H$  di  $\{2, 0, 3, 5, 6, 5, 4, 3, 2, 0, 1, 7, 0, 9\}$  è 5, in quanto il quinto elemento in ordine non crescente è 5 (che è uguale a 5), mentre il sesto elemento in ordine non crescente è 4 (che è minore di 6). Analogamente, l'indice  $H$  di  $\{23, 1, 3, 14, 6, 15, 4, 13, 2, 10, 1, 17, 20, 19\}$  è 8, in quanto l'ottavo elemento in ordine non crescente è 10 (che è maggiore di 8), mentre il nono elemento in ordine non crescente è 6 (che è minore di 9). Infine, l'indice  $H$  di un array di elementi tutti uguali a 0 è, per convenzione, uguale a 0.

Scrivere un metodo, chiamato `hIndex`, che dato in input un array  $a$  di numeri interi non negativi, restituisca il suo indice  $H$ .

$$a = \{2, 0, 3, 5, 6, 5, 4, 3, 2, 0, 1, 7, 0, 9\}$$

Ordiniamo  $a$

$$b = \{9, 7, 6, 5, 5, 4, 3, 3, 2, 2, 1, 0, 0, 0\}$$

Il primo elemento di  $b$  è  $9 \geq 1$   
Il secondo elemento di  $b$  è  $7 \geq 2$   
Il terzo elemento di  $b$  è  $6 \geq 3$   
Il quarto elemento di  $b$  è  $5 \geq 4$   
Il quinto elemento di  $b$  è  $5 \geq 5$   
Il sesto elemento di  $b$  è  $4 < 6$   
Quindi, l'indice  $H$  è uguale a 5.

## Esempi

- $a = \{0, 3, 0, 3, 1, 1\}$ : 2
- $a = \{5, 8, 4, 10, 3\}$ : 4
- $a = \{1, 1, 2, 1, 1\}$ : 1

# Strike and ball

## Esercizio

Dati due numeri interi positivi  $n$  e  $m$  con lo stesso numero di cifre, si ha uno *strike* tra  $n$  e  $m$  per ogni cifra di  $n$  che appare anche in  $m$  nella stessa posizione, mentre si ha un *ball* tra  $n$  e  $m$  per ogni cifra di  $n$  che appare anche in  $m$  ma in posizione diversa. Ad esempio, se  $n = 180712$  e  $m = 785104$ , si ha uno strike in corrispondenza dell'8 e tre ball in corrispondenza di un 1, dello 0 e del 7.

Scrivere un metodo, chiamato `strikeBall`, che, dati in input due numeri interi positivi  $n$  e  $m$  con lo stesso numero di cifre, restituisca un array di due elementi uguali al numero di strike e al numero di ball tra  $n$  e  $m$ , rispettivamente.

Per calcolare il numero di strike è sufficiente calcolare il numero di cifre uguali in  $n$  e in  $m$  che appaiono nella stessa posizione. Per calcolare il numero di ball si usi il seguente algoritmo. Si costruiscano due array di 10 elementi che indichino le frequenze delle 10 cifre decimali in  $n$  e in  $m$  (senza però contare gli strike): il numero di ball è la somma, per  $i$  che va da 0 a 9, dei minimi tra i due valori contenuti nei due array in posizione  $i$ .

## Esempi

- $n = 180712$  e  $m = 781104$ :  $\{1, 4\}$
- $n = 180712$  e  $m = 180712$ :  $\{6, 0\}$
- $n = 180712$  e  $m = 211087$ :  $\{0, 6\}$
- $n = 123321$  e  $m = 456654$ :  $\{0, 0\}$

$$n = 180712 \text{ e } m = 785104$$

Uno strike

8 in  $n$  e  $m$  nella stessa posizione

Tre ball

Uno 0 in  $n$  e  $m$  in posizioni diverse

Un 1 in  $n$  e  $m$  in posizioni diverse

Un 7 in  $n$  e  $m$  in posizioni diverse

$$n = 180712 \text{ e } m = 781108$$

Uno strike

8 in  $n$  e  $m$  nella stessa posizione

Quattro ball

Uno 0 in  $n$  e  $m$  in posizioni diverse

Due 1 in  $n$  e  $m$  in posizioni diverse

Un 7 in  $n$  e  $m$  in posizioni diverse

$$n = 180712 \text{ e } m = 785104$$

Gli array delle frequenze (senza strike)

$$\{1, 2, 0, 0, 0, 0, 0, 1, 0, 0\}$$

$$\{1, 1, 0, 0, 1, 1, 0, 1, 0, 0\}$$

L'array dei minimi

$$\{1, 1, 0, 0, 0, 0, 0, 1, 0, 0\}$$

La somma dei minimi è 3

# Reverse word

## Esercizio

Una *parola* è una sequenza di lettere alfabetiche minuscole. Una frase è una sequenza di parole separate da uno spazio. Data una frase  $f$ , la sua inversa per parole è la frase costituita dalle parole di  $f$  in ordine inverso rispetto a  $f$ .

Scrivere un metodo chiamato `reverseWord` che, dato in input un array  $f$  di caratteri rappresentante una frase, restituisca l'array di caratteri rappresentante l'inversa per parole della frase rappresentata da  $f$ . Si può assumere che l'array in input sia non vuoto, che il suo primo carattere non sia uno spazio, che il suo ultimo carattere non sia uno spazio, che contenga solamente lettere alfabetiche minuscole e spazi e che sia la corretta rappresentazione di una frase.

"oggi ieri e domani"  
Frase con 4 parole

Inversa di  $f$ :  
"domani e ieri oggi"

## Esempi

- $f = \{'o', 'g', 'g', 'i', ' ', 'i', 'e', 'r', 'i', ' ', 'e', ' ', 'd', 'o', 'm', 'a', 'n', 'i'\}$ :  
 $\{'d', 'o', 'm', 'a', 'n', 'i', ' ', 'e', ' ', 'i', 'e', 'r', 'i', ' ', 'o', 'g', 'g', 'i'\}$

## Lexical permutations

### Esercizio

Dato un insieme di  $n$  caratteri, consideriamo il problema di generare tutte le permutazioni di quest'insieme in ordine lessicografico. La generazione di tali permutazioni può essere eseguita nel modo seguente.

1. Generare la permutazione dei caratteri ordinati in modo crescente.
2. A partire dalla permutazione corrente generare la prossima nel modo seguente.
  - (a) Determinare il carattere  $c_1$  più destra seguito da un carattere più grande.
  - (b) Determinare il minimo carattere  $c_2$  alla destra di  $c_1$  e più grande di  $c_1$ .
  - (c) Scambiare  $c_1$  e  $c_2$ .
  - (d) Ordinare la sotto-sequenza alla destra della posizione originale di  $c_1$ .
3. Ripetere il passo precedente  $n! - 1$  volte.

Scrivere un metodo, chiamato `lexPermutation`, che dati in input un array  $a$  di  $n$  caratteri distinti e un intero positivo  $p \leq n!$ , restituisca un array rappresentante la permutazione in posizione  $p$  nella generazione lessicografica.

### Esempi

- $a = \{'m', 'b', 't'\}$  e  $p = 1$ :  $\{'b', 'm', 't'\}$
- $a = \{'m', 'b', 't'\}$  e  $p = 4$ :  $\{'m', 't', 'b'\}$
- $a = \{'m', 'b', 't'\}$  e  $p = 6$ :  $\{'t', 'm', 'b'\}$
- $a = \{'a', 'x', 'o', 'f'\}$  e  $p = 12$ :  $\{'f', 'x', 'o', 'a'\}$
- $a = \{'a', 'x', 'o', 'f'\}$  e  $p = 16$ :  $\{'o', 'f', 'x', 'a'\}$

Insieme  $\{'m', 'b', 't'\}$

1.  $\{'b', 'm', 't'\}$
2.  $\{'b', 't', 'm'\}$
3.  $\{'m', 'b', 't'\}$
4.  $\{'m', 't', 'b'\}$
5.  $\{'t', 'b', 'm'\}$
6.  $\{'t', 'm', 'b'\}$

Insieme  $\{'a', 'x', 'o', 'f'\}$

1.  $\{'a', 'f', 'o', 'x'\}$
2.  $\{'a', 'f', 'x', 'o'\}$
3.  $\{'a', 'o', 'f', 'x'\}$
4.  $\{'a', 'o', 'x', 'f'\}$
5.  $\{'a', 'x', 'f', 'o'\}$
6.  $\{'a', 'x', 'o', 'f'\}$
- ...
24.  $\{'x', 'o', 'f', 'a'\}$

Sequenza iniziale:  $\{'a', 'f', 'o', 'x'\}$

Se sequenza corrente è  $\{'o', 'a', 'x', 'f'\}$

$c_1 = 'a'$

$c_2 = 'f'$

Sequenza diventa  $\{'o', 'f', 'x', 'a'\}$

Sequenza diventa  $\{'o', 'f', 'a', 'x'\}$

## *Palindrome pair*

### *Esercizio*

Un numero intero positivo  $n$  è *palindromo* se e solo se leggerlo da sinistra verso destra è equivalente a leggerlo da destra verso sinistra. Per esempio, 42599524 e 38183 sono due numeri palindromi, mentre 19434491 e 1233211 non lo sono. La *concatenazione* di due numeri interi positivi  $n$  e  $m$  è il numero ottenuto concatenando  $m$  ad  $n$ . Per esempio, la concatenazione di 42599 e 524 è il numero 42599524, quella di 38 e 183 è 38183, quella di 12 e 33211 è 1233211, e quella di 1943449 e 1 è 19434491.

Scrivere un metodo, chiamato `palindromePair`, che, dato in input un array `a` di numeri interi positivi non multipli di 10, restituisca un array bidimensionale contenente in ciascuna riga due elementi  $x \leq y$  uguali alle posizioni (eventualmente coincidenti) all'interno di `a` di due suoi elementi la cui concatenazione fornisca un numero palindromo. Le righe devono essere ordinate in modo che  $\{x, y\}$  venga prima di  $\{w, z\}$  se e solo se  $x < w$  oppure  $x = w$  e  $y < z$ . Se non esistono coppie di elementi di `a` la cui concatenazione fornisca un numero palindromo, il metodo deve restituire un array con nessuna riga, ovvero l'array creato con l'istruzione `new int[0][ ]`.

### *Esempi*

- `a = {42599, 38, 183, 524}`: `{{0, 3}, {1, 2}}`
- `a = {42597, 38, 183, 524}`: `{{1, 2}}`
- `a = {134, 33211, 1943449, 25}`: `{}`
- `a = {134, 33211, 1943449, 121}`: `{{3, 3}}`
- `a = {111, 33211, 1943449, 1}`: `{{0, 0}, {0, 3}, {3, 3}}`

# Interval merge

## Esercizio

Dati 2 intervalli chiusi  $[a, b]$  e  $[c, d]$  che si sovrappongono (ovvero tali che  $a \leq c \leq b$ ), la loro *fusione* consiste dell'intervallo  $[a, \max(b, d)]$ . Dati  $n$  intervalli chiusi, specificati dai due estremi sinistro e destro e ordinati in modo non decrescente rispetto agli estremi sinistri, la fusione di questi intervalli consiste nel fondere tutti gli intervalli che si sovrappongono. In particolare, partendo dal primo intervallo, se questo si sovrappone al secondo, allora i due intervalli vengono fusi, ottenendo un nuovo intervallo. Se questo nuovo intervallo si sovrappone al terzo, i due intervalli vengono fusi ottenendo un nuovo intervallo. Si procede in questo modo fino a quando il nuovo intervallo non si sovrappone a quello successivo da esaminare. In tal caso, il nuovo intervallo viene memorizzato e si ricomincia con l'intervallo da esaminare fino ad avere esaminato tutti gli intervalli.

Scrivere un metodo, chiamato `intervalMerge`, che data in input una matrice `m` di interi positivi di dimensione  $n \times 2$ , rappresentante  $n$  intervalli chiusi ordinati in modo non decrescente rispetto alla prima colonna, restituisca la matrice degli intervalli ottenuti dalla fusione degli  $n$  intervalli.

## Esempi

- `m = {1, 3}, {2, 6}, {8, 10}, {15, 18}`: `{1, 6}, {8, 10}, {15, 18}`
- `m = {1, 3}, {2, 4}, {5, 7}, {6, 8}`: `{1, 4}, {5, 8}`
- `m = {3, 6}, {5, 7}, {6, 12}, {10, 12}, {11, 16}`: `{3, 16}`

Intervalli

`[1, 3], [2, 6], [8, 10], [15, 18]`

Primo intervallo fuso con secondo  
Nuovo intervallo

`[1, 6]`

`[1, 6]` non fuso con terzo  
`[8, 10]` non fuso con quarto  
Intervalli finali

`[1, 6], [8, 10], [15, 18]`

Intervalli

`[1, 3], [2, 4], [5, 7], [6, 8]`

Primo intervallo fuso con secondo  
Nuovo intervallo

`[1, 4]`

`[1, 4]` non fuso con terzo  
`[5, 7]` fuso con quarto  
Nuovo intervallo

`[5, 8]`

Intervalli finali

`[1, 4], [5, 8]`

Intervalli

`[3, 6], [5, 7], [6, 12], [10, 12], [11, 16]`

Primo intervallo fuso con secondo  
Nuovo intervallo

`[3, 7]`

`[3, 7]` fuso con terzo  
Nuovo intervallo

`[3, 12]`

`[3, 12]` fuso con quarto  
Nuovo intervallo

`[3, 12]`

`[3, 12]` fuso con quinto  
Nuovo intervallo

`[3, 16]`

Intervalli finali

`[3, 16]`



# Traveller

## Esercizio

Consideriamo una matrice  $m$  di  $n$  righe e 2 colonne contenente numeri interi positivi e tale che la somma dei numeri nella prima colonna sia uguale alla somma dei numeri nella seconda colonna. Una riga della matrice è un *punto di partenza* se, partendo da un “budget” uguale a zero, da essa è possibile “visitare” tutte le righe della matrice, una dopo l’altra ciclicamente, senza che il budget divenga mai negativo applicando la regola seguente di aggiornamento del budget. Quando si visita una riga il budget aumenta del valore incluso nella prima colonna della riga e diminuisce del valore incluso nella seconda colonna. Chiaramente, se una riga è un punto di partenza, alla fine della visita il budget deve essere uguale a 0.

Scrivere un metodo, chiamato `traveller`, che, data in input una matrice  $m$  di  $n$  righe e 2 colonne contenente numeri interi positivi e tale che la somma dei numeri nella prima colonna sia uguale alla somma dei numeri nella seconda colonna, restituisca l’array contenente gli indici delle righe che sono punti di partenza.

## Esempi

- $m = \{\{5, 10\}, \{30, 20\}, \{25, 30\}, \{10, 10\}, \{10, 5\}, \{50, 45\}, \{20, 30\}\}$ :  
 $\{1\}$
- $m = \{\{5, 10\}, \{30, 20\}, \{25, 30\}, \{10, 15\}, \{10, 10\}, \{60, 45\}, \{20, 30\}\}$ :  
 $\{1, 4, 5\}$

$$m = \begin{array}{|c|c|} \hline 5 & 10 \\ \hline 30 & 20 \\ \hline 25 & 30 \\ \hline 10 & 10 \\ \hline 10 & 5 \\ \hline 50 & 45 \\ \hline 20 & 30 \\ \hline \end{array}$$

Somma prima e seconda colonna = 150

Partendo dalla riga 0

- Riga 0:  $0 + 5 - 10 = -5$

La riga 0 non è punto di partenza

Partendo dalla riga 1

- Riga 1:  $0 + 30 - 20 = 10$
- Riga 2:  $10 + 25 - 30 = 5$
- Riga 3:  $5 + 10 - 10 = 5$
- Riga 4:  $5 + 10 - 5 = 10$
- Riga 5:  $10 + 50 - 45 = 15$
- Riga 6:  $15 + 20 - 30 = 5$
- Riga 0:  $5 + 5 - 10 = 0$

La riga 1 è punto di partenza

Partendo dalla riga 3

- Riga 3:  $0 + 10 - 10 = 0$
- Riga 4:  $0 + 10 - 5 = 5$
- Riga 5:  $5 + 50 - 45 = 10$
- Riga 6:  $10 + 20 - 30 = 0$
- Riga 0:  $0 + 5 - 10 = -5$

La riga 3 non è punto di partenza

# Isomorphism

## Esercizio

Sia  $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$  l'insieme delle cifre decimali. Due array  $s$  e  $t$  di uguale lunghezza  $n$ , contenenti solo elementi in  $D$ , si dicono isomorfi se esiste una corrispondenza biunivoca  $f$  da  $D$  a  $D$  tale che, per ogni  $i$  compreso tra 0 e  $n - 1$ ,  $t[i]$  è uguale a  $f(s[i])$ .

Definire un metodo, chiamato `isomorph`, che dati in input due array  $s$  e  $t$  di uguale lunghezza, contenenti solo cifre decimali, restituisca il valore `true` se e solo se  $s$  e  $t$  sono isomorfi.

## Esempi

- $s = \{2, 0, 1, 2, 2\}$  e  $t = \{1, 3, 0, 1, 1\}$ : `true`
- $s = \{3, 0, 4, 3, 2\}$  e  $t = \{4, 1, 2, 4, 3\}$ : `true`
- $s = \{2, 3, 0, 2, 0\}$  e  $t = \{4, 1, 3, 0, 3\}$ : `false`
- $s = \{2, 3, 0, 1, 4\}$  e  $t = \{4, 1, 3, 0, 4\}$ : `false`

$$s = \{2, 0, 1, 2, 2\}$$

$$t = \{1, 3, 0, 1, 1\}$$

$s$  e  $t$  isomorfi

- $f(0) = 3$
- $f(1) = 0$
- $f(2) = 1$

Valore di  $f$  per altre cifre decimali non importa, in quanto queste non appaiono in  $s$

## **Parte VI**

# **Algoritmi golosi**



# Change

## Esercizio

Dato un sistema monetario in cui siano disponibili  $n$  tipi distinti di moneta tali che la moneta da 1 sia presente, e dato un resto  $r$ , un modo possibile di determinare quali e quante monete usare per dare il resto è il seguente algoritmo goloso. Ordiniamo i tipi di moneta in ordine decrescente. Esaminiamo i tipi di moneta uno dopo l'altro e prendiamo tante copie del tipo attuale quante ne possono entrare nel resto rimasto, eventualmente sottraendo al resto stesso la quantità ottenuta. Terminiamo quando il resto diviene uguale a 0.

Scrivere un metodo, chiamato `change`, che, dato in input un array  $c$  di  $n$  numeri interi positivi distinti (che indichino i tipi di monete), tale che esista un indice  $k$  per cui  $c[k] = 1$ , e un numero intero positivo  $r$  (che indichi il resto), restituisca il numero di monete da utilizzare per dare il resto ottenuto mediante l'algoritmo goloso appena descritto.

## Esempi

- $c = \{1, 2, 5, 10, 20, 50, 100, 200\}$  e  $r = 453$ : 5
- $c = \{1, 5, 10, 25, 50, 100\}$  e  $r = 453$ : 8
- $c = \{1, 2, 4, 8, 16, 32, 64\}$  e  $r = 453$ : 9

Tipi di moneta

$\{4, 2, 1, 30, 12, 5\}$

Resto: 117

Tipi di moneta ordinati

$\{30, 12, 5, 4, 2, 1\}$

Prendiamo 3 monete da 30: resto 27

Prendiamo 2 monete da 12: resto 3

Prendiamo 0 monete da 5: resto 3

Prendiamo 0 monete da 4: resto 3

Prendiamo 1 moneta da 2: resto 1

Prendiamo 1 moneta da 1: resto 0

# Bin packing

## Esercizio

Dato un numero intero positivo  $s$  e dato un array  $a$  di numeri interi positivi minori o uguali a  $s$ , un impacchettamento di  $a$  consiste in una suddivisione degli elementi di  $a$  in gruppi, tali che la somma degli elementi in ciascun gruppo non sia maggiore di  $s$ .

Un impacchettamento può essere costruito in modo *goloso* facendo uso del seguente algoritmo.

1. Ordiniamo gli elementi di  $a$  in ordine non crescente.
2. Creiamo tanti gruppi vuoti quanti sono gli elementi di  $a$ .
3. Scandiamo gli elementi ordinati e inseriamo ciascun elemento nel primo gruppo che lo può contenere (ovvero, tale che inserendo l'elemento nel gruppo, la somma del gruppo non superi  $s$ ).
4. Eliminiamo i gruppi rimasti vuoti.

Scrivere un metodo, chiamato `ffd`, che dato un numero intero positivo  $s$  e dato un array  $a$  di numeri interi positivi minori o uguali a  $s$ , restituisca il numero di gruppi nell'impacchettamento calcolato dall'algoritmo goloso appena descritto.

Per implementare il metodo, è sufficiente memorizzare in un array la somma attuale di ciascun gruppo (senza ricordarsi quali siano gli elementi nel gruppo). Alla fine il metodo può restituire il numero di elementi diversi da zero di tale array.

## Esempi

- $a = \{4, 1, 2, 5, 3, 2, 3, 6, 3\}$  e  $s = 6$ : 5
- $a = \{1, 6, 8, 10, 6, 3, 8, 2, 6, 4\}$  e  $s = 10$ : 6

Se  $s=4$  e  $a=\{2, 3, 1, 4, 2, 1, 1\}$ , allora  $\{3, 1\}$ ,  $\{4\}$ ,  $\{2, 2\}$ ,  $\{1, 1\}$  è un impacchettamento di  $a$ . Infatti,  $3 + 1 \leq 4$ ,  $4 \leq 4$ ,  $2 + 2 \leq 4$  e  $1 + 1 \leq 4$ . Anche  $\{3\}$ ,  $\{4\}$ ,  $\{2, 1, 1\}$ ,  $\{2, 1\}$  è un impacchettamento di  $a$ .

$a$  ordinato:  $\{4, 3, 2, 2, 1, 1, 1\}$

7 gruppi:  $\{\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\}$

$\{4\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\}$   
 $\{4\}, \{3\}, \{\}, \{\}, \{\}, \{\}, \{\}$   
 $\{4\}, \{3\}, \{2\}, \{\}, \{\}, \{\}, \{\}$   
 $\{4\}, \{3\}, \{2, 2\}, \{\}, \{\}, \{\}, \{\}$   
 $\{4\}, \{3, 1\}, \{2, 2\}, \{\}, \{\}, \{\}, \{\}$   
 $\{4\}, \{3, 1\}, \{2, 2\}, \{1\}, \{\}, \{\}, \{\}$   
 $\{4\}, \{3, 1\}, \{2, 2\}, \{1, 1\}, \{\}, \{\}, \{\}$

`ffd`( $\{2, 3, 1, 4, 2, 1, 1\}, 4$ ) restituisce 4

$\{4, 0, 0, 0, 0, 0, 0\}$   
 $\{4, 3, 0, 0, 0, 0, 0\}$   
 $\{4, 3, 2, 0, 0, 0, 0\}$   
 $\{4, 3, 4, 0, 0, 0, 0\}$   
 $\{4, 4, 4, 0, 0, 0, 0\}$   
 $\{4, 4, 4, 1, 0, 0, 0\}$   
 $\{4, 4, 4, 2, 0, 0, 0\}$

# Independent set

## Esercizio

Sia  $m$  una matrice  $n \times n$  contenente solo 0 e 1, simmetrica (ovvero  $m[i][j] = m[j][i]$  per ogni coppia di indici  $i$  e  $j$ , con  $0 \leq i, j < n$ ) e tale che la diagonale contenga solo 0 (ovvero  $m[i][i] = 0$  per ogni indice  $i$ , con  $0 \leq i < n$ ). Due indici  $i$  e  $j$ , con  $0 \leq i, j < n$ , sono *indipendenti* se  $m[i][j] = 0$ . Un insieme di indici  $I$  è un *insieme indipendente* se, per ogni coppia di indici  $i$  e  $j$  in  $I$ ,  $i$  e  $j$  sono indipendenti.

Un insieme indipendente può essere costruito mediante il seguente approccio goloso. Ordiniamo gli indici in ordine non decrescente rispetto al numero di 1 contenuti nella riga corrispondente di  $m$  (se due righe hanno lo stesso numero di 1, viene prima quella con indice più piccolo). Scandiamo gli indici uno dopo l'altro e inseriamo un indice  $i$  nell'insieme indipendente  $I$  se  $i$  è indipendente dagli indici già inclusi in  $I$ .

Scrivere un metodo, chiamato `independentSet`, che, data in input una matrice quadrata  $m$  contenente solo 0 e 1, simmetrica e tale che la diagonale contenga solo 0, restituisca la cardinalità dell'insieme indipendente costruito dall'algoritmo goloso appena descritto (per rappresentare l'insieme indipendente può essere utile fare uso di un array di valori Booleani).

0	1	1	1
1	0	1	0
1	1	0	0
1	0	0	0

1 e 3 sono indipendenti  
0 e 1 non sono indipendenti

Indici e corrispondenti numeri di 1

0	1	2	3
3	2	2	1

Indici ordinati

3	1	2	0
1	2	2	3

- 3 viene selezionato
- 1 viene selezionato ( $m[1][3] = 0$ )
- 2 non viene selezionato ( $m[2][1] = 1$ )
- 0 non viene selezionato ( $m[0][3] = 1$ )

Insieme indipendente

false	true	false	true
-------	------	-------	------

## Esempi

- $m = \{\{0, 1, 1, 1\}, \{1, 0, 1, 0\}, \{1, 1, 0, 0\}, \{1, 0, 0, 0\}\}$ : 2
- $m = \{\{0, 1, 0, 1, 0, 1\}, \{1, 0, 1, 0, 1, 0\}, \{0, 1, 0, 1, 0, 1\}, \{1, 0, 1, 0, 1, 0\}, \{0, 1, 0, 1, 0, 1\}, \{1, 0, 1, 0, 1, 0\}\}$ : 3
- $m = \{\{0, 1, 1, 1, 1, 1\}, \{1, 0, 1, 0, 0, 1\}, \{1, 1, 0, 1, 0, 0\}, \{1, 0, 1, 0, 1, 0\}, \{1, 0, 0, 1, 0, 1\}, \{1, 1, 0, 0, 1, 0\}\}$ : 2
- $m = \{\{0, 0, 0, 0, 0\}, \{0, 0, 0, 0, 0\}, \{0, 0, 0, 0, 0\}, \{0, 0, 0, 0, 0\}, \{0, 0, 0, 0, 0\}\}$ : 5
- $m = \{\{0, 1, 1, 1, 1\}, \{1, 0, 1, 1, 1\}, \{1, 1, 0, 1, 1\}, \{1, 1, 1, 0, 1\}, \{1, 1, 1, 1, 0\}\}$ : 1

# Greedy code

## Esercizio

Il *codice goloso* corrispondente a un numero intero  $n$  associa a ogni cifra decimale un carattere alfabetico secondo la seguente regola. Alla cifra più frequente in  $n$  viene assegnato il carattere a, a quella più frequente fra le rimanenti il carattere b, a quella più frequente fra le rimanenti il carattere c e così via (a parità di frequenze, si dà la precedenza alla cifra più piccola). Un algoritmo per calcolare il codice goloso corrispondente a un numero intero  $n$  opera nel modo seguente. Calcola la matrice delle frequenze delle cifre decimali, ordina tale matrice e la utilizza per decidere il carattere da associare a ciascuna cifra. Vediamo questi passi più in dettaglio.

*Matrice delle frequenze.* Dato un numero intero  $n$ , la matrice  $m$  delle frequenze corrispondente a  $n$  è un array bidimensionale di due righe e dieci colonne. Per ogni  $i$  con  $0 \leq i < 10$ , l'elemento in prima riga e colonna  $i$  è uguale a  $i$ , mentre l'elemento in seconda riga e colonna  $i$  è uguale al numero di volte che la cifra  $i$  appare in  $n$ .

*Ordinamento della matrice delle frequenze.* La matrice  $m$  delle frequenze deve essere ordinata in base alle frequenze, cioè in base ai valori della seconda riga (ricordando che a parità di frequenza, si dà la precedenza alla cifra più piccola).

*Costruzione del codice goloso.* Il codice è rappresentato mediante un array  $c$  di dieci caratteri. Per ogni colonna  $i$  della matrice  $m$  delle frequenze ordinata, il carattere di  $c$  nella posizione specificata dalla prima riga della colonna è uguale al carattere che segue alfabeticamente il carattere a di  $i$  posizioni.

Scrivere un metodo, chiamato `greedyCode`, che dato un numero intero positivo  $n$  di tipo `long`, restituisca l'array rappresentante il codice goloso corrispondente a  $n$ .

## Esempi

- $n = 122721$ : [d, b, a, e, f, g, h, c, i, j]
- $n = 346360844766$ : [d, g, h, c, b, i, a, e, f, j]

$n = 2136353418482150401$

0	⇒	d
1	⇒	a
2	⇒	e
3	⇒	b
4	⇒	c
5	⇒	f
6	⇒	h
7	⇒	i
8	⇒	g
9	⇒	j

$n = 2136353418482150401$

Matrice  $m$  delle frequenze

0	1	2	3	4	5	6	7	8	9
2	4	2	3	3	2	1	0	2	0

Matrice  $m$  delle frequenze ordinate

1	3	4	0	2	5	8	6	7	9
4	3	3	2	2	2	2	1	0	0

L'array rappresentante il codice

d	a	e	b	c	f	h	i	g	j
---	---	---	---	---	---	---	---	---	---



- $n = 2097563043614052353$ : [b, g, d, a, e, c, f, h, j, i]
- $n = 333333$ : [b, c, d, a, e, f, g, h, i, j]
- $n = 0$ : [a, b, c, d, e, f, g, h, i, j]

# Knapsack

## Esercizio

Siano dati  $n$  oggetti  $o_1, \dots, o_n$ , ciascuno con uno specifico *valore*  $v_i$  e uno specifico costo  $c_i$ , e sia dato un *limite superiore*  $b$  (assumiamo che i valori, i pesi e il limite superiore siano tutti numeri interi positivi). Una soluzione al problema della *bisaccia* con tale input è un sottoinsieme degli oggetti la somma dei cui pesi non sia maggiore di  $b$ : il valore della soluzione è uguale alla somma dei valori degli oggetti inclusi nella soluzione stessa. Una soluzione al problema della bisaccia può essere calcolata mediante il seguente algoritmo goloso. Ordiniamo gli oggetti in ordine non crescente della loro *densità di valore*, ovvero del rapporto  $\frac{v_i}{p_i}$  (a parità di densità, l'oggetto con valore maggiore viene prima). Esaminiamo tutti gli oggetti in tale ordine, e aggiungiamo un oggetto alla soluzione se il peso degli oggetti già inclusi nella soluzione sommato al peso dell'oggetto attualmente esaminato non è maggiore di  $b$ .

Scrivere un metodo, chiamato `knapsack`, che data un input una matrice `m` di interi positivi di dimensione  $n \times 2$  (rappresentante  $n$  oggetti il cui valore sia indicato nella prima colonna e il cui peso sia indicato nella seconda colonna) e un intero positivo `b`, restituisca il valore della soluzione calcolata dall'algoritmo goloso sopra descritto.

## Esempi

- `m = {{10,5},{5,6},{12,8},{7,2},{9,7}}` e `b = 14`: 26
- `m = {{15,10},{20,10},{33,11}}` e `b = 20`: 33
- `m = {{50,31},{65,39},{35,26},{16,21},{18,25},{55,28},{45,29},{40,27},{25,23}}` e `b = 100`: 170

Valore e peso degli oggetti

`[[10,5],[5,6],[12,8],[7,2],[9,7]]`

Limite superiore

`b = 14`

Densità di peso degli oggetti

- `[10,5]`: 2.0
- `[5,6]`: 0.8333...
- `[12,8]`: 1.5
- `[7,2]`: 3.0
- `[9,7]`: 1.2857...

Oggetti ordinati

`[[7,2],[10,5],[12,8],[9,7],[5,6]]`

Costruzione della soluzione

- Oggetto `[7,2]`: incluso  
Peso soluzione: 2
- Oggetto `[10,5]`: incluso  
Peso soluzione: 7
- Oggetto `[12,8]`: non incluso
- Oggetto `[9,7]`: incluso  
Peso soluzione: 14
- Oggetto `[5,6]`: non incluso

Valore soluzione:  $7 + 10 + 9 = 26$ .

# Knight path

## Esercizio

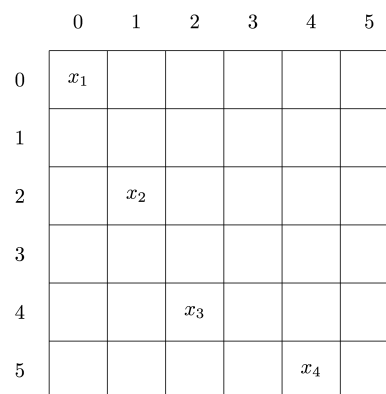
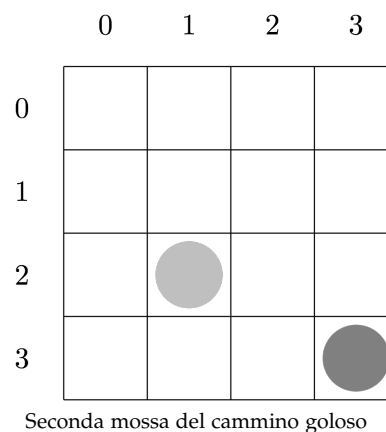
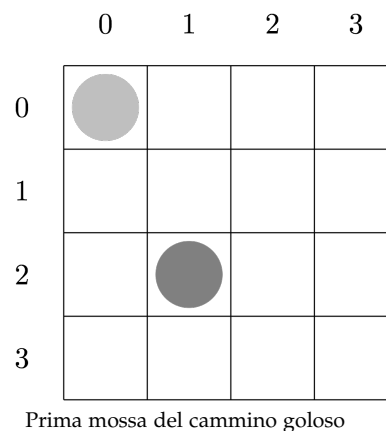
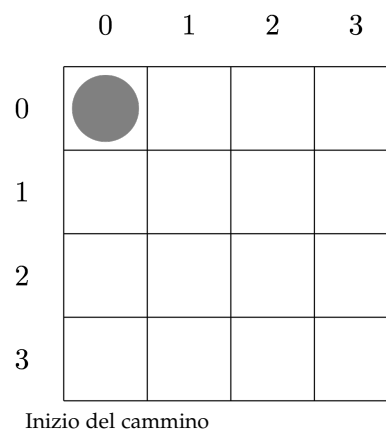
Nel gioco degli scacchi il cavallo è un pezzo che può fare due soli tipi di mosse che consistono nel muoversi di 2 caselle in orizzontale e 1 in verticale oppure di 1 casella in orizzontale e 2 in verticale.

Consideriamo una scacchiera di dimensione  $n \times n$  con  $n > 2$ , le cui righe siano numerate dall'alto in basso da 0 a  $n - 1$  e le cui colonne siano numerate da sinistra a destra da 0 a  $n - 1$ . All'inizio, il cavallo è posizionato nella casella  $s = (0,0)$ . L'obiettivo è quello di arrivare alla casella  $t = (n - 1, n - 1)$  eseguendo, in ogni momento, la seguente mossa golosa: il cavallo va sempre nella casella che lo avvicina di più alla meta. Più precisamente, trovandosi nella casella  $x$ , il cavallo va nella casella  $y = (r, c)$  raggiungibile da  $x$  che minimizza la distanza di Manhattan tra  $y$  e  $t$ , ovvero il valore  $|r - (n - 1)| + |c - (n - 1)|$ . In caso di parità, il cavallo si sposta nella casella  $d$  che minimizza  $|r - (n - 1)|$ . Infine, se nessuna casella raggiungibile da  $c$  ha una distanza di Manhattan minore di quella di  $c$  da  $t$ , allora il cavallo si ferma.

Ad esempio, se  $n = 4$ , allora dalla casella  $x = (0,0)$ , il cavallo si sposta nella casella  $y_1 = (2,1)$ . Infatti, l'altra casella dove potrebbe andare è la casella  $y_2 = (1,2)$ . La distanza di Manhattan di  $y_1$  da  $t$  è uguale a  $|2 - 3| + |1 - 3| = 1 + 2 = 3$  e la distanza di Manhattan di  $y_2$  da  $t$  è uguale a  $|1 - 3| + |2 - 3| = 2 + 1 = 3$ . Le due distanze sono uguali, ma il cavallo si sposta in  $y_1$  in quanto  $|2 - 3| = 1 < 2 = |1 - 3|$ .

Non sempre il cammino goloso permette al cavallo di raggiungere la casella  $t$ . Nel caso  $n = 4$ , ciò è vero, in quanto il cavallo va dalla casella  $(0,0)$  alla casella  $(2,1)$  e da questa va in  $t = (3,3)$ . Ma se  $n = 6$ , il cavallo, a partire dalla casella  $x_1 = (0,0)$ , passa per le caselle  $x_2 = (2,1)$ ,  $x_3 = (4,2)$  e  $x_4 = (5,4)$ . Ogni casella raggiungibile da  $x_4$  allontana il cavallo da  $t = (5,5)$ : in questo caso, quindi il cavallo si ferma.

Scrivere un metodo, chiamato `knightPath`, che dato un numero intero positivo  $n$  maggiore di 2, restituisca il valore `true` se e solo se il



Se  $n = 6$  il cavallo non arriva a  $t$

cammino goloso del cavallo a partire dalla casella  $(0,0)$  raggiunge la casella  $(n-1, n-1)$ .

### *Esempi*

Per  $2 < n < 100$ , i soli valori per cui `knightPath` restituisce il valore `true` sono i seguenti: 4, 9, 12, 17, 20, 25, 28, 33, 36, 41, 44, 49, 52, 57, 60, 65, 68, 73, 76, 81, 84, 89, 92, 97.

# Planner

## Esercizio

Due intervalli chiusi a sinistra e aperti a destra  $[a, b)$  e  $[c, d)$  non si sovrappongono se  $b \leq c$ . Dati  $n$  intervalli chiusi a sinistra e aperti a destra, specificati dai due estremi sinistro e destro, il massimo numero di intervalli che non si sovrappongono può essere ottenuto mediante il seguente algoritmo goloso. Ordiniamo gli intervalli in ordine non decrescente del loro estremo destro. Selezioniamo il primo intervallo come ultimo. Esaminiamo gli altri intervalli nell'ordine sopra indicato, e selezioniamo un intervallo se il suo estremo sinistro non è minore dell'estremo destro dell'ultimo (in tal caso, il nuovo intervallo selezionato diviene l'ultimo). Restituiamo il numero di intervalli selezionati.

Scrivere un metodo, chiamato `plan`, che data un input una matrice `m` di interi positivi di dimensione  $n \times 2$ , rappresentante  $n$  intervalli chiusi a sinistra e aperti a destra  $[a, b)$  con  $a < b$ , restituisca il massimo numero di intervalli che non si sovrappongono.

## Esempi

- `m = {{1,2},{8,9},{5,7},{0,6},{3,4},{5,9}}`: 4
- `m = {{1,6},{2,9},{5,7},{0,8},{3,9},{5,10}}`: 1
- `m = {{1,2},{7,8},{5,6},{9,10},{3,4},{11,12}}`: 6

## Intervalli

`[1,2], [8,9], [5,7], [0,6], [3,4], [5,9]`

## Intervalli ordinati

`[1,2], [3,4], [0,6], [5,7], [8,9], [5,9]`

- Ultimo: `[1,2]`
- `3 > 2`. Ultimo: `[3,4]`
- `0 < 4`.
- `5 > 4`. Ultimo: `[5,7]`
- `8 > 7`. Ultimo: `[8,9]`
- `5 < 9`.

4 intervalli non si sovrappongono.

# Shortest job first

## Esercizio

Siano dati  $n$  compiti  $c_i$  da eseguire, ciascun con un proprio tempo di esecuzione  $t_i$ . Il *tempo medio di attesa* corrispondente a uno specifico ordine di esecuzione dei compiti è uguale alla somma dei tempi di attesa di tutti i compiti diviso  $n$ . Ad esempio, se  $n = 4$  e i quattro compiti hanno tempi di esecuzione pari a  $t_1 = 21$ ,  $t_2 = 3$ ,  $t_3 = 1$  e  $t_4 = 2$  e i compiti sono eseguiti nell'ordine  $c_1, c_2, c_3$  e  $c_4$ , allora il tempo di attesa del primo compito eseguito è 0, quello del secondo è 21, quello del terzo è  $21 + 3 = 24$  e quello del quarto è  $21 + 3 + 1 = 25$ . Quindi, in quest'ordine il tempo medio di attesa è uguale a  $\frac{0+21+24+25}{4} = 17.5$ . Se invece i 4 compiti sono eseguiti nell'ordine  $c_2, c_4, c_1$  e  $c_3$ , allora i tempi di attesa sono 0, 3,  $3 + 2 = 5$  e  $3 + 2 + 21 = 26$ . Quindi, in quest'ordine il tempo medio di attesa è uguale a  $\frac{0+3+5+26}{4} = 8.5$ .

Un algoritmo goloso per calcolare un ordine di esecuzione dei compiti che minimizzi il tempo medio di attesa consiste nell'ordinare i compiti in ordine non decrescente rispettivamente ai loro tempi di esecuzione.

Scrivere un metodo, chiamato `sjf`, che dato un array di numeri interi positivi, restituisca il tempo medio di attesa ottenuto applicando l'algoritmo goloso.

## Esempi

- $a = \{7, 5, 1, 4\}$ : 4.0
- $a = \{80, 20, 10, 20, 50\}$ : 38.0
- $a = \{13\}$ : 0.0
- $a = \{13, 13, 13, 13, 13, 13, 13\}$ : 39.0

Con l'ordine  $\{21, 3, 1, 2\}$

$$\begin{aligned}\tau &= \frac{0 + 21 + (21 + 3) + (21 + 3 + 1)}{4} \\ &= \frac{0 + 21 + 24 + 25}{4} = 17.5\end{aligned}$$

Con l'ordine  $\{3, 2, 21, 1\}$

$$\begin{aligned}\tau &= \frac{0 + 3 + (3 + 2) + (3 + 2 + 21)}{4} \\ &= \frac{0 + 3 + 5 + 26}{4} = 8.5\end{aligned}$$

Con l'ordine  $\{1, 2, 3, 21\}$

$$\begin{aligned}\tau &= \frac{0 + 1 + (1 + 2) + (1 + 2 + 3)}{4} \\ &= \frac{0 + 1 + 3 + 6}{4} = 2.5\end{aligned}$$

# Deepest downhill length

## Esercizio

Data una matrice quadrata  $m$  di numeri interi positivi, una *discesa golosa* a partire da un elemento  $a$  di  $m$  è una sequenza di elementi della matrice, che inizia da  $a$  tale che ogni elemento nella sequenza (oltre ad  $a$ ) è più piccolo di quello precedente e due elementi consecutivi nella sequenza appartengono alla stessa riga e a due colonne adiacenti oppure appartengono alla stessa colonna e a due righe adiacenti.

La *discesa golosa ripida* che parte da un elemento  $a$  della matrice  $m$  è una discesa golosa che parte da  $a$ , nella quale ogni elemento  $x$  della discesa è seguito dall'elemento più piccolo della matrice che appartiene alla stessa riga di  $x$  oppure alla stessa colonna di  $x$  (in caso di parità, l'elemento a nord è preferito a quello a ovest, il quale è preferito a quello a sud, il quale è preferito a quello a est). La *lunghezza della discesa golosa ripida* (in breve, *ddl*) di una matrice è il massimo numero di elementi inclusi nella discesa golosa ripida a partire da uno qualunque degli elementi della matrice. Per esempio, consideriamo la seguente matrice.

4	8	7	4
2	5	9	3
1	3	1	2
4	4	1	6

La sua *ddl* è uguale a 5, in quanto la discesa golosa ripida che parte dall'elemento 7 è 7, 4, 3, 2, 1, e, quindi, ha lunghezza 5. Tutte le altre discese golose ripide non sono più lunghe di 5. Per esempio, quella partire dal primo 4 è 4, 2, 1 e ha lunghezza 3, quella a partire dall'8 è 8, 4, 2, 1 e ha lunghezza 4, e così via.

Scrivere un metodo, chiamato *ddl* che, data in input una matrice  $m$  di numeri interi positivi, restituisca la lunghezza della discesa golosa ripida di  $m$ .

4	8	7	4
2	5	9	3
1	3	1	2
4	4	1	6

5, 3, 1 è una discesa golosa

4	8	7	4
2	5	9	3
1	3	1	2
4	4	1	6

8, 4, 2, 1 è una discesa golosa ripida a partire da 8

### *Esempi*

- $m = \{\{4, 8, 7, 4\}, \{2, 5, 9, 3\}, \{1, 3, 1, 2\}, \{4, 4, 1, 6\}\}$ : 5
- $m = \{\{1, 3, 10, 19, 7\}, \{2, 14, 13, 18, 7\}, \{16, 9, 20, 7, 18\}, \{16, 10, 19, 10, 11\}, \{4, 13, 2, 4, 19\}\}$ :  
4
- $m = \{\{16, 15, 14, 13\}, \{12, 11, 10, 9\}, \{8, 7, 6, 5\}, \{4, 3, 2, 1\}\}$ : 7
- $m = \{\{1, 1, 1\}, \{1, 1, 1\}, \{1, 1, 1\}, \{1, 1, 1\}\}$ : 1



## **Parte VII**

# **Algoritmi ricorsivi**



# *Partitioner*

## *Esercizio*

Facendo uso del metodo di generazione delle stringhe binarie (e, quindi, dei sottoinsiemi di un dato insieme), scrivere un metodo, chiamato `partition`, che, dato in input un array `a` di numeri interi, restituisca il valore `true` se e solo se esiste un sottoinsieme degli elementi di `a` la cui somma sia uguale alla metà della somma di tutti gli elementi di `a`.

$$a = \{1, 6, 2, 3, 4, 2, 1, 3\}$$

Somma totale: 22

Sottoinsieme  $S$ :  $\{1, 6, 4\}$

Somma  $S$ : 11

## *Esempi*

- $a = \{1, 6, 2, 3, 4, 2, 1, 3\}$ : `true`
- $a = \{3, 3, 4, 12, 5, 2\}$ : `false`
- $a = \{-1, 6, -2, 3, 6, -2, 2, -6\}$ : `true`
- $a = \{-1, 5, -2, 3, 6, -2, 2, -6, 0\}$ : `false`

# Strange partitioner

## Esercizio

Facendo uso del metodo di generazione delle stringhe binarie (e, quindi, dei sottoinsiemi di un dato insieme), scrivere un metodo, chiamato `strangePartition`, che, dato in input un array `a` di numeri interi positivi, restituisca il valore `true` se e solo se esiste un sottoinsieme `X` degli elementi di `a` la cui somma sia un multiplo di 10 e tale che la somma degli elementi di `a` non in `X` sia dispari.

## Esempi

- $a = \{5, 5, 5\}$ : `true`
- $a = \{5, 5, 6\}$ : `false`
- $a = \{5, 5, 6, 1\}$ : `true`
- $a = \{6, 5, 5, 1, 10\}$ : `true`
- $a = \{6, 5, 5, 5, 1\}$ : `false`
- $a = \{10, 7, 5, 5, 2\}$ : `true`
- $a = \{10, 7, 5, 5, 1\}$ : `false`

$$a = \{5, 5, 5\}$$

---

Sottoinsieme `X`:  $\{5, 5\}$   
Somma elementi in `X`:  $5 + 5 = 10$   
Somma elementi non in `X`: 5

---

Metodo deve restituire valore `true`

$$a = \{5, 5, 6\}$$

---

Sottoinsieme `X`:  $\{\}$   
Somma elementi in `X`: 0  
Somma elementi non in `X`: 16

---

Sottoinsieme `X`:  $\{5\}$   
Somma elementi in `X`: 5

---

Sottoinsieme `X`:  $\{5\}$   
Somma elementi in `X`: 5

---

Sottoinsieme `X`:  $\{6\}$   
Somma elementi in `X`: 6

---

Sottoinsieme `X`:  $\{5, 5\}$   
Somma elementi in `X`: 10  
Somma elementi non in `X`: 6

---

Sottoinsieme `X`:  $\{5, 6\}$   
Somma elementi in `X`: 11

---

Sottoinsieme `X`:  $\{5, 6\}$   
Somma elementi in `X`: 11

---

Sottoinsieme `X`:  $\{5, 5, 6\}$   
Somma elementi in `X`: 16

---

Metodo deve restituire valore `false`

# Set packing

## Esercizio

Sia  $a$  una matrice  $n \times m$  contenente solo 0 e 1. Due indici di riga  $i$  e  $j$ , con  $0 \leq i, j < n$ , sono *disgiunti* se, per ogni  $k$  con  $0 \leq k < m$ ,  $a[i][k] \cdot a[j][k] = 0$ . Un insieme di indici di riga  $I$  e' un *impacchettamento* di  $a$  se, per ogni coppia  $i$  e  $j$  in  $I$ ,  $i$  e  $j$  sono disgiunti.

Facendo uso del metodo di generazione delle stringhe binarie (e, quindi, dei sottoinsiemi di un dato insieme), scrivere un metodo, chiamato `setPacking`, che, data in input una matrice  $a$  contenente solo 0 e 1, restituisca la cardinalità dell'impacchettamento di massima cardinalità.

## Esempi

- $a = \{\{0, 1, 1, 1, 0\}, \{1, 0, 1, 0, 0\}, \{1, 1, 0, 0, 0\}, \{0, 1, 0, 0, 1\}\}$ : 2
- $a = \{\{1, 1, 1, 1, 0, 0, 0, 0, 0\}, \{0, 0, 0, 1, 1, 1, 0, 0, 0\}, \{0, 0, 0, 0, 1, 1, 1, 1, 0, 0\}, \{0, 0, 0, 0, 0, 0, 0, 1, 1\}\}$ : 3
- $a = \{\{1, 1, 1, 1, 0, 0, 0, 0, 0\}, \{0, 0, 0, 1, 1, 1, 0, 0, 0\}, \{0, 0, 0, 0, 1, 1, 1, 1, 0\}, \{0, 0, 0, 0, 0, 0, 0, 1, 1\}, \{1, 1, 1, 0, 0, 0, 0, 0, 0\}\}$ : 3
- $a = \{\{1, 1, 1, 1, 0, 0, 0, 0, 0\}, \{0, 0, 0, 1, 1, 1, 0, 0, 0\}, \{0, 0, 0, 0, 1, 1, 1, 1, 0\}, \{0, 0, 0, 0, 0, 0, 0, 1, 1\}, \{1, 1, 1, 0, 0, 0, 0, 0, 0\}, \{0, 0, 0, 1, 0, 0, 0, 0, 0\}\}$ : 4
- $a = \{\{1, 1, 1, 1, 0, 0, 0, 0, 0\}, \{0, 0, 0, 1, 1, 1, 0, 0, 0\}, \{0, 0, 0, 0, 1, 1, 1, 1, 0\}, \{0, 0, 0, 0, 0, 0, 0, 1, 1\}, \{1, 1, 1, 0, 0, 0, 0, 0, 0\}, \{0, 0, 0, 1, 0, 0, 0, 0, 0\}, \{0, 1, 0, 0, 1, 0, 0, 0, 0\}\}$ : 4

0	1	1	1	0
1	0	1	0	0
1	1	0	0	0
0	1	0	0	1

1 e 3 disgiunti

$$a[1][0] \cdot a[3][0] = 1 \cdot 0 = 0$$

$$a[1][1] \cdot a[3][1] = 0 \cdot 1 = 0$$

$$a[1][2] \cdot a[3][2] = 1 \cdot 0 = 0$$

$$a[1][3] \cdot a[3][3] = 0 \cdot 0 = 0$$

$$a[1][4] \cdot a[3][4] = 0 \cdot 1 = 0$$

0 e 1 non disgiunti

$$a[0][2] \cdot a[1][2] = 1 \cdot 1 = 1$$

## *k-Subset sum*

### *Esercizio*

Gli array binari di lunghezza  $n$  e con esattamente  $k$  elementi uguali a 1 possono essere generati in modo simile a come vengono generati tutti gli array binari di lunghezza  $n$ . Infatti, è sufficiente apportare le seguenti modifiche al metodo ricorsivo di generazione delle stringhe binarie descritto nel libro di testo. Il metodo ha bisogno di un ulteriore parametro  $r$  che indichi quanti 1 devono ancora essere generati (inizialmente  $r = k$ ). Successivamente, la prima chiamata ricorsiva (quella in cui il bit in posizione  $b - 1$  ha valore 0) viene effettuata solo se le rimanenti posizioni da occupare sono almeno pari a  $r$  (ovvero solo se  $r < b$ ), mentre la seconda chiamata ricorsiva (quella in cui il bit in posizione  $b - 1$  ha valore 1) viene effettuata solo se ci sono ancora 1 da inserire (ovvero solo se  $r > 0$ ).

Usando questo metodo per generare gli array binari di lunghezza  $n$  e con esattamente  $k$  elementi uguali a 1, scrivere un metodo, chiamato `kSubsetSum`, che, dati in input un array  $a$  di numeri interi positivi, un intero positivo  $k$  non maggiore della lunghezza di  $a$  e un intero positivo  $s$ , restituisca il valore `true` se e solo se esistono esattamente  $k$  elementi in  $a$  la cui somma sia uguale a  $s$ .

### *Esempi*

- $a = \{1, 2, 2, 6, 7, 7, 20\}$ ,  $k = 1$  e  $s = 20$ : `true`
- $a = \{1, 2, 2, 6, 7, 7, 20\}$ ,  $k = 2$  e  $s = 20$ : `false`
- $a = \{1, 2, 2, 6, 7, 7, 20\}$ ,  $k = 3$  e  $s = 20$ : `true`
- $a = \{1, 2, 2, 6, 7, 7, 20\}$ ,  $k = 4$  e  $s = 20$ : `false`
- $a = \{1, 2, 2, 6, 7, 7, 20\}$ ,  $k = 5$  e  $s = 20$ : `false`
- $a = \{1, 2, 2, 6, 7, 7, 20\}$ ,  $k = 6$  e  $s = 20$ : `false`
- $a = \{1, 2, 2, 6, 7, 7, 20\}$ ,  $k = 7$  e  $s = 45$ : `true`

# Bursting balloons

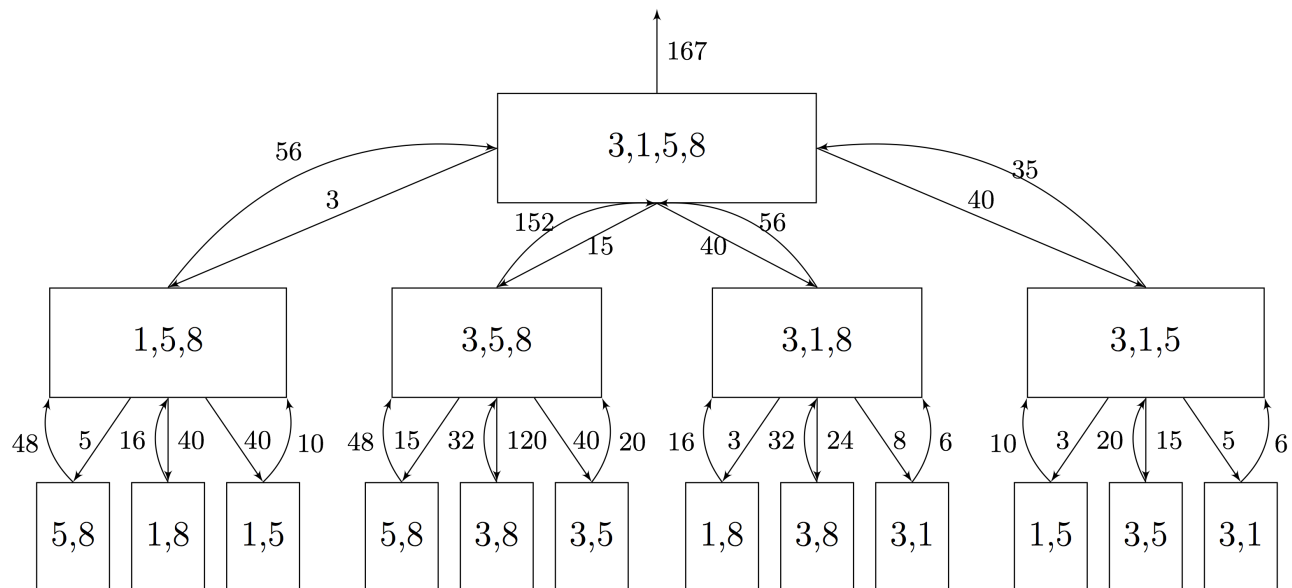
## Esercizio

Sia dato un array di  $n$  numeri interi positivi. Quando si “spara” a un elemento dell’array, si guadagnano tante monetine quanto è il prodotto dell’elemento colpito e dei suoi due elementi adiacenti (se un elemento adiacente non esiste, allora il suo valore si assume uguale a 1). Dopo lo sparo, l’elemento colpito sparisce dall’array la cui dimensione si riduce di uno. L’obiettivo è trovare l’ordine con cui sparare agli  $n$  elementi in modo da ottenere il massimo numero di monetine.

Con l’array  $\{3, 1, 5, 8\}$

Se si spara su 3 si guadagna  $1 \cdot 3 \cdot 1 = 3$   
 Se si spara su 1 si guadagna  $3 \cdot 1 \cdot 5 = 15$   
 Se si spara su 5 si guadagna  $1 \cdot 5 \cdot 8 = 40$   
 Se si spara su 8 si guadagna  $5 \cdot 8 \cdot 1 = 40$

Se si spara su 3, il nuovo array:  $\{1, 5, 8\}$   
 Se si spara su 1, nuovo array:  $\{3, 5, 8\}$   
 Se si spara su 5, nuovo array:  $\{3, 1, 8\}$   
 Se si spara su 8, nuovo array:  $\{3, 1, 5\}$



Un algoritmo ricorsivo per calcolare il massimo numero di monetine ottenibile, opera nel modo seguente (si veda la figura precedente che mostra l’esecuzione dell’algoritmo con input l’array  $\{3, 1, 5, 8\}$  e in cui le etichette degli archi verso il basso denotano il valore  $x$ , le altre il valore  $y$ ). Per ogni possibile posizione dell’array, (1) calcola il numero  $x$  di monetine guadagnate sparando all’elemento in quella posizione ed elimina l’elemento, (2) calcola ricorsivamente il massimo numero  $y$  di monetine ottenibile a partire dall’array ridotto, e (3) se  $x + y$  è maggiore del massimo finora trovato, aggiorna il massimo. Alla fine del ciclo re-

stituisce il massimo trovato. Il caso base dell'algoritmo è quando l'array ha due elementi  $a$  e  $b$ : in tal caso, l'algoritmo restituisce  $a \cdot b + \max(a, b)$ .

Scrivere un metodo, chiamato `maxCoins`, che dato un array di  $n$  cifre decimali positive con  $1 < n < 10$ , restituisca il massimo numero di centesimi che si possono guadagnare con  $n$  spari, facendo uso dell'algoritmo ricorsivo precedentemente descritto.

### *Esempi*

- $a = \{4, 3\}$ : 16
- $a = \{2, 4, 2, 3\}$ : 57
- $a = \{3, 1, 5, 8, 2, 4, 2, 3, 1\}$ : 406
- $a = \{1, 1, 1, 1, 1, 1, 1, 1\}$ : 9
- $a = \{9, 9, 9, 9, 9, 9, 9, 9, 9\}$ : 5193