Laboratorio 10

Esercizio 1

Creare una classe **EserciziMatrici** con i seguenti metodi:

- un metodo *initAlt*(int[] matp, int[] matd, int numr) che ritorna in output un nuovo array di arrays con numr righe, dove le righe pari sono una copia di matp, e le righe dispari una copia di matd.
  HINT: definire un metodo ausiliario clonaArray per duplicare un array.
- un metodo *contaElementi* che prende in input un array di arrays, e ritorna in output la somma totale delle lunghezze di tutte le righe. Le righe nulle contano come lunghezza zero.
- un metodo *linearizzaRighe* che prende in input un array di arrays di interi mat e restituisce in output un array, i cui elementi sono la concatenazione delle righe di mat. Le righe null sono ignorate. Se mat è null, il metodo deve ritornare null a sua volta. L'array di arrays in input può essere irregolare (ragged).
- un metodo *toString*(int[][] m) che inserisca tutti gli elementi dell'array di arrays m in una stringa. Usando il carattere di escape '\n', sia dia la forma «da matrice» alla stringa, cioè gli elementi di m su righe diverse devono risultare su righe diverse anche nella stringa.
  HINT: '\n' è il carattere di "ritorno a capo", e permette di costruire una stringa su più righe.

Definire i seguenti array:
int[] a1 = {3, 5, 7}, a2 = {2, 10, 8, 9}, a3 = {8};
e le seguenti matrici:
int[][] m1 = *initAlt*(a1, a2, 6), m2 = *initAlt*(a3, null, 5), m3 = *initAlt*(null, a2, 4);
e stampare in output il risultato di *linearizzaRighe* per m1, m2 e m3.

Esercizio 2

Aggiungere i seguenti metodi alla classe **EserciziMatrici**:

- un metodo *maxRowLen* che prende in input un array di arrays, e ritorna in output la massima lunghezza delle righe. Le righe nulle contano come lunghezza 0. Se l'input è null, il metodo deve ritornare 0.
- un metodo *sommaRighe* che prende in input un array di arrays a, e ritorna in output un nuovo array contenente la somma delle righe di a, cioè ogni elemento i-esimo dell'array in output è la somma degli elementi i-esimi delle righe della matrice (se esistono). L' array di arrays in input può essere irregolare.

Definire la matrice:
int[][] m4 = { {2,5,6}, {3,7,8,9,1}, {0,2}, {0,3,9,1} };
e stampare la *sommaRighe* delle matrici m1, m2, m3 e m4.

Esercizio 3

Aggiungere un metodo *azzeraColonnaMax*(int[] imat) che prende in input un array di arrays ed azzera la colonna la cui somma degli elementi è la più grande. Usare il precedente metodo *sommaRighe* per trovare la colonna con somma massima. Verificare che il metodo, se chiamato ripetutamente, azzera progressivamente tutte le colonne dell'array di arrays in input.
NOTA: la colonna j da azzerare potrebbe non comparire in tutte le righe, in quanto l'array di arrays in input può essere ragged. Per queste righe, non fare nessuna operazione quando viene azzerata una colonna j >= length.

Esercizio 4

Data una matrice rettangolare mat di numeri interi positivi di dimensione m*n, si dice che una posizione j < n è dominante per la riga i < m se il numero in posizione mat[i][j] divide tutti i numeri della riga mat[i]. Si scriva un metodo iterativo *domMat*() che restituisca true sse ogni riga ha almeno una posizione dominante.
Implementare un metodo ausiliario *domRiga*() che, dato array mat[i] e intero j restituisce true se il numero in posizione j divide tutti i numeri dell'array mat[i]. Richiamare *domRiga*() per ogni riga della matrice per vedere se la proprietà vale su tutte le righe della matrice.
Provare il metodo *domMat* sulle seguenti matrici:
int[][] m5 = { {1, 5, 10, 7}, {3, 12, 21, 30}, {5, 10, 20, 30} }; // true
int[][] m6 = { {4, 7, 2, 5}, {7, 9, 20, 12}, {5, 8, 11, 21} }; // false

Esercizio 5 (*copia con trasformazione)*

Nella classe **EserciziMatrici**, creare un metodo ricorsivo *incrementa*(int[][] imat) che che ritorna in output un nuovo array di arrays omat i cui elementi si ottengono incrementando di uno i corrispondenti elementi di imat.
Implementare un metodo ricorsivo **covariante** *incrementaRiga*(int[] in, int[] out, int k) che prende in input l'array sorgente e l'array destinazione, che si assumono entrambi allocati nello heap, e un indice di ricorsione k, ed effetua la copia dell'array con incremento degli elementi.
Implementare un secondo metodo ricorsivo **contro-variante** *incrementaRic* che si occupa di allocare gli array per l'output omat, e chiama *incrementaRiga* su ciascuna riga dell'input.
Gestire opportunamente le matrici irregolari e i possibili valori null.
Stampare il risultato di *incrementa* sulle matrici m1, m2, m3 e m4.

Esercizio 6 (*conteggio dicotomico)*

Scrivere un metodo ricorsivo **dicotomico** *conteggio*(int[] a, int k) che prenda un array a e un intero k come parametri e restituisca il numero di occorrenze di k in a. Il metodo involucro deve richiamare un metodo *conteggioDicotomico* che abbia 2 parametri aggiuntivi (indice iniziale e finale porzione di array considerata) e effettuare la ricerca in modo dicotomico: identificando in a parti di lunghezza via via piu' corte, ma di lunghezza esenzialmente identica. Quando la parte contiene un singolo elemento verifica se esso sia pari a k. In tal caso il conteggio è 1. Altrimenti è zero.

Esercizio 7

Risolvere i due esercizi riportati nel file BTestoEsame.java, nell'archivio EsempioEsame.zip. L'archivio contiene anche due classi di test, EsameTestE1 e EsameTestE2. L'esecuzione di queste classi permette di sperimentare alcuni input/output attesi daglie sercizi richiesti.