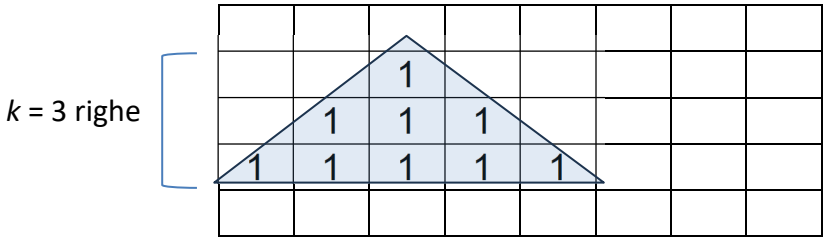
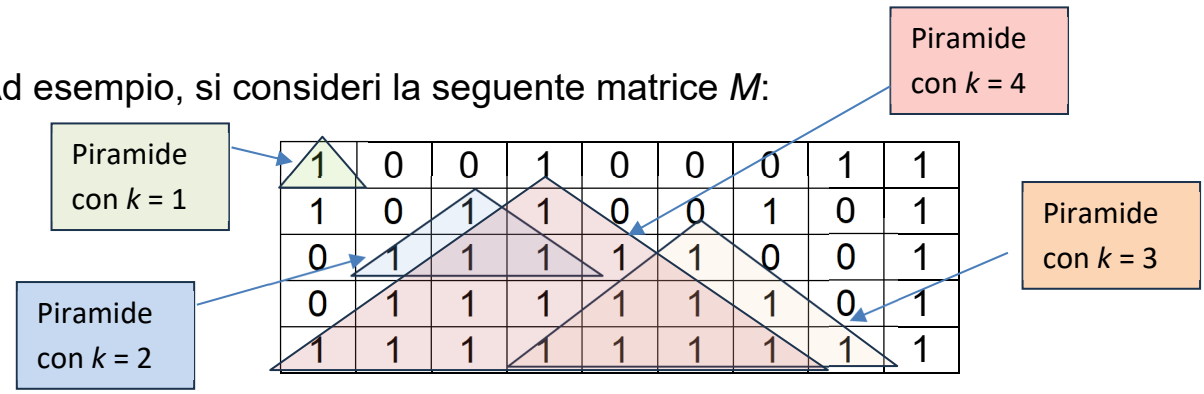
# Esercizio n. 1: Cercare una Piramide

Sia *M* una matrice di interi (di tipo `byte`) di dimensione *m* x *n* (con *m>0* e *n*>0) che può contenere solo valori 0 oppure 1, e sia *k* un valore intero (di tipo `int`) con *k*>0. Una piramide di altezza *k* è definita come una sottomatrice di *M* che contiene tutti valori uguali ad 1 disposti in una forma piramidale di altezza *k* (ovvero formata da *k* righe), come nel seguente esempio (in cui *k*=3):




[**CONSEGNA STANDARD**]: Scrivere un metodo Java-- chiamato `piramide` che, dati in input una matrice *M* di interi ed un intero *k* come precedentemente definiti, restituisca il valore booleano `true` se la matrice *M* contiene almeno una piramide di altezza *k*, e restituisca `false` altrimenti (se nella matrice *M* non c'è nessuna piramide di altezza *k*).

Ad esempio, si consideri la seguente matrice *M*:




- Se invocato con *k*=1, il metodo deve restituire `true` poiché nella matrice c'è almeno una cella con valore uguale ad 1 (caso particolare di piramide di altezza 1).
- Se invocato con *k*=2, il metodo deve restituire `true` poiché nella matrice c'è almeno una piramide di altezza 2 (ce ne sono più di una).
- Se invocato con *k*=3, il metodo deve restituire `true` poiché nella matrice c'è almeno una piramide di altezza 3 (ce ne sono più di una).

- Se invocato con *k*=4, il metodo deve restituire `true` poiché nella matrice c'è almeno una piramide di altezza 4 (ce n'è soltanto una).
- Se invocato con *k*=5, il metodo deve restituire `false` poiché nella matrice non ci sono piramidi di altezza 5.
- Se invocato con valori di *k*≥*6*, il metodo deve restituire `false` poiché nella matrice non ci possono essere piramidi di tale altezza (non c'è spazio sufficiente nella matrice).

[**CONSEGNA RIDOTTA**]: Nella consegna ridotta si assuma che *k* possa assumere solo i seguenti tre valori: *k*=1, *k*=2, *k*=3.

[**CONSEGNA EXTRA - FACOLTATIVA**]: È possibile svolgere la consegna extra solo se si è svolta la consegna standard (non quella ridotta). Scrivere un metodo Java-- chiamato `altezzaMassimaPiramide` che, dati in input una matrice *M* di interi come precedentemente definita, restituisca un valore intero (di tipo `int`) che corrisponde all'altezza massima delle piramidi fra quelle contenute nella matrice *M*. Se non ci sono piramidi, il metodo deve restituire 0.

NOTA BENE:

- Saranno premiate le soluzioni che occuperanno meno memoria dati e che limiteranno il numero di celle da visitare della matrice *M*.
- I Junit Test da superare per la consegna ridotta sono quelli della classe **PiramideRidottaTest** (gli altri test falliranno).
- I Junit Test da superare per la consegna standard sono quelli della classe **PiramideStandardTest** (oltre a quelli della classe PiramideRidottaTest che devono comunque essere superati).
- I Junit Test da superare per la consegna extra (facoltativa) sono quelli della classe **PiramideExtraTest** (oltre a quelli della classe PiramideRidottaTest e PiramideStandardTest che devono comunque essere superati).

- Nello svolgere l'esercizio NON devono essere utilizzati i metodi clone, o arraycopy, o metodi della classe Arrays. L'utilizzo di tali metodi renderà l'esercizio automaticamente insufficiente.

# Esercizio n. 2: Comprimi ed Espandi Intero

Sia *n* un numero intero (di tipo `int`) con *n*≥0, e sia *k* un numero intero (di tipo `byte`) con *k*≥1.

[*Procedura COMPRIMI*] La procedura che *comprime* il numero *n* con chiave *k* produce un nuovo numero intero *m*≥0 le cui cifre si ottengono sommando gruppi di *k* cifre del numero *n* partendo da quelle meno significative. Ad esempio, con *n* = 1002478 e con *k*=2 si ottiene il numero *m* = 10615:

| | | | | |
|---|---|---|---|---|
| *n* = | 1 | 0 0 | 2 4 | 7 8 |
| | 0+1 | 0+0 | 2+4 | 7+8 |
| | \|\| | \|\| | \|\| | \|\| |
| *m* = | 1 | 0 | 6 | 15 |

Infatti, partendo dalle cifre meno significative di *n*, raggruppandole a gruppi di *k*=2 cifre e sommando le cifre di ciascun gruppo, si ha che 7+8 = 15, 2+4=6, 0+0=0, 0+1=1.

Altri esempi partendo dallo stesso *n* = 1002478:

- con k=1, m = 1 0 0 2 4 7 8
- con k=2, m = 1 0 6 1 5
- con k=3, m = 1 2 1 9
- con k=4, m = 1 2 1
- con k=5, m = 1 2 1
- con k=6, m = 1 2 1
- con k≥7, m = 2 2

[*Procedura ESPANDI*] La procedura che *espande* il numero *n* con chiave *k* produce un nuovo numero intero *m*≥0 le cui cifre si ottengono aggiungendo (*k*-1) cifre uguali a zero prima di ogni singola cifra di *n*. Ad esempio, con *n*=408 e con *k*=3 si ottiene il numero *m* = 4000008:

| | | | |
|---|---|---|---|
| *n* = | 4 | 0 | 8 |
| *m* = | 004 | 000 | 008 |

Infatti, si aggiungono due zeri prima della cifra 8, due zeri prima della cifra 0, e due zeri prima della cifra 4.

Altri esempi partendo dallo stesso n = 408:

- con *k*=1, *m* = 4 0 8
- con *k*=2, *m* = 4 00 08
- con *k*=3, *m* = 4 000 008
- con *k*=4, *m* = 4 0000 0008

[**CONSEGNA STANDARD**]: Scrivere due metodi Java-- chiamati `comprimi` ed `espandi` che, dati in input i due numeri interi *n* e *k* come precedentemente definiti, restituiscano il numero intero che viene generato, rispettivamente, dall'applicazione della procedura *comprimi* ed *espandi* sul numero *n* con chiave *k*.

NOTA per la procedura *espandi*: si assuma che il metodo possa essere invocato solo con una combinazione di valori di *n* e *k* tale che il numero prodotto sia sempre rappresentabile come un intero di tipo `int` (no overflow).

[**CONSEGNA RIDOTTA**]: Nella consegna ridotta è possibile svolgere soltanto uno fra i due metodi (`comprimi` oppure `espandi`) a scelta dello studente.

[**CONSEGNA EXTRA**]: È possibile svolgere la consegna extra solo se si è svolta la consegna standard (non quella ridotta). Due numeri interi $n_1$ e $n_2$ sono "connessi" con chiave *k* se uno dei due numeri corrisponde al risultato prodotto dall'applicazione della procedura *comprimi* o *espandi* sull'altro numero. Ad esempio: i numeri $n_1$=1122 e $n_2$=24 sono connessi con chiave *k*=2 (si applica comprimi su $n_1$ e si ottiene $n_2$), ed i numeri $n_1$=124 e $n_2$=1002004 sono connessi con chiave k=3 (si applica espandi su $n_1$ e si ottiene $n_2$). Scrivere un metodo Java-- chiamato `connessi` che, dati in input due numeri interi $n_1$ e $n_2$ e una chiave *k* (con $n_1 \geq 0$, $n_2 \geq 0$ e $k \geq 1$), restituisca `true` se partendo da uno dei due numeri interi ($n_1$ o $n_2$) è possibile produrre l'altro numero attraverso l'applicazione di una delle due procedure *comprimi* o *espandi*, e restituisca `false` altrimenti.

NOTA BENE:

- Saranno premiate le soluzioni che occuperanno meno memoria dati.
- È possibile invocare il metodo `Math.pow` per calcolare la potenza di un numero.
- I Junit Test da superare per la consegna ridotta sono quelli della classe **ComprimiTest** oppure **EspandiTest**, a seconda del metodo che si è sviluppato (comprimi oppure espandi).
- I Junit Test da superare per la consegna standard sono quelli di entrambe le classi **ComprimiTest** e **EspandiTest**.
- I Junit Test da superare per la consegna extra sono quelli della classe **ConnessiTest** (oltre a quelli delle classi ComprimiTest e EspandiTest che devono comunque essere superati).
- Nello svolgere l'esercizio NON devono essere utilizzati i metodi `clone`, o `arraycopy`, o metodi della classe `Arrays`. L'utilizzo di tali metodi renderà l'esercizio automaticamente insufficiente.