Metodi del Calcolo Scientifico
Relazione Progetto 2
Metodi diretti per matrici sparse
1 Luglio 2021

Gusmara Andrea (831141)
Stoianov Oleg (829519)
Villani Alessio (830075)

1

INTRODUZIONE

1. Introduzione

Lo scopo di questo progetto è di utilizzare l'implementazione della DCT2 in un ambiente open source e di studiare gli effetti della compressione tipo jpeg (senza utilizzare una matrice di quantizzazione) sulle immagini in toni di grigio.

JPEG (Joint Photographic Experts Group) è una tecnica di compressione con perdita (lossy compression) per immagini digitali, il grado di compressione può essere selezionato in modo tale da avere un trade off tra dimensione dell'immagine e qualità della stessa.

Alla base del JPEG vi è l'algoritmo DCT (Discrete Cosine Transform) che permette di esprimere una sequenza finita di dati nei termini di una somma di coseni che oscillano a differenti frequenze.

Nel nostro progetto abbiamo utilizzato:

- Java, come ambiente open source.
- Libreria JTransforms (versione 3.1), che implementa la DCT in due dimensioni.
- Framework JFrame, per implementare la componente grafica richiesta nelle specifiche del progetto.
- JFreeChart, per realizzare i grafici di confronto tra DTC2 homemade e della libreria.
- Eclipse IDE come ambiente di sviluppo.

2

PRIMA PARTE

2. Prima parte

Per questa prima parte era richiesta l'implementazione di una DCT bidimensionale, secondo il modello proposto a lezione, e successivamente il confronto con una libreria open source che implementasse la DCT2D nella versione fast (FFT) .

Implementazione DCT2 *homemade*

```
public static double [][] applyDCT2(double[][] f) {
    double[][] dctMatrix = new double[f.length][f[0].length];
    for (int k = 0; k < f.length; k++) {
      for (int l = 0; l < f[0].length; l++) {
        double sum = 0.0;
        for (int i = 0; i < f.length; i++) {
          for (int j = 0; j < f[0].length; j++) {
            sum = sum + ( f[i][j] * Math.cos((((2*i+1)/(2.0 * f.length)) * k * Math.PI)) * Math.cos((((2 * j+1)/(2.0 * f.length) * l * Math.PI))));
          }
        }
        if(k == 0 && l== 0)
            sum = sum / Math.sqrt(f.length*f[0].length);
        else if (k == 0 || l == 0)
            sum = sum * Math.sqrt(2.0/ (f.length *f[0].length));
        else {
            sum = sum * (2.0 / Math.sqrt(f.length*f[0].length));
        }
        dctMatrix[k][l]=sum;

      }
    }
    return dctMatrix;
}

public static double [][] applyIDCT2(double[][]c) {
    double[][] originalMatrix = new double[c.length][c[0].length];
    double a = 0 ;
    for (int i = 0; i < c.length; i++) {
      for (int j = 0; j < c[0].length; j++) {
        double sum = 0.0;
        for (int k = 0; k < c.length; k++) {
          for (int l = 0; l < c[0].length; l++) {
            if(k == 0 && l== 0)
                a = 1 / Math.sqrt(c.length*c[0].length);
            else if (k == 0 || l == 0)
                a =  Math.sqrt(2.0/ (c.length *c[0].length));
             else {
                 a= (2.0 / Math.sqrt(c.length*c[0].length));
            }
            sum += ( c[k][l]* a * Math.cos((((2 * i+1)/(2.0 * c.length) * k * Math.PI))) * Math.cos((((2 * j+1)/(2.0 * c[0].length) * l * Math.PI))));

             originalMatrix[i][j]=sum;

          }
        }

      }
    }
    return originalMatrix;
}
```

Formule proposte a lezione

$$c_{k\ell} = \alpha_{k\ell} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} f_{ij} \cos\left(k\,\pi \frac{2i+1}{2N}\right) \cos\left(\ell\,\pi \frac{2j+1}{2M}\right) \quad \text{(DCT2).}$$

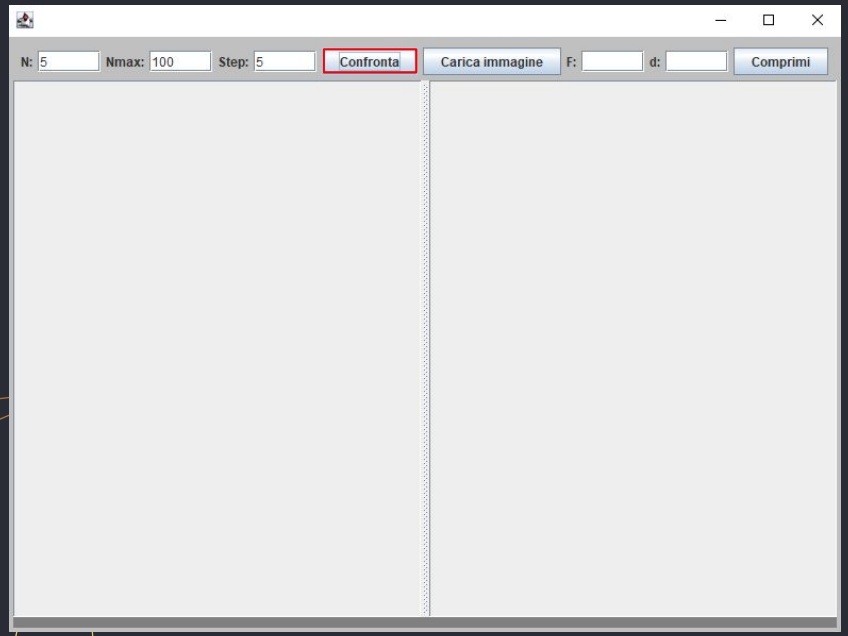
$$\alpha_{00} = \frac{1}{\sqrt{NM}}, \quad \text{e} \quad \alpha_{k0} = \alpha_{0\ell} = \sqrt{\frac{2}{NM}}, \quad \alpha_{k\ell} = \frac{2}{\sqrt{NM}}, \quad k, \ell \geq 1.$$

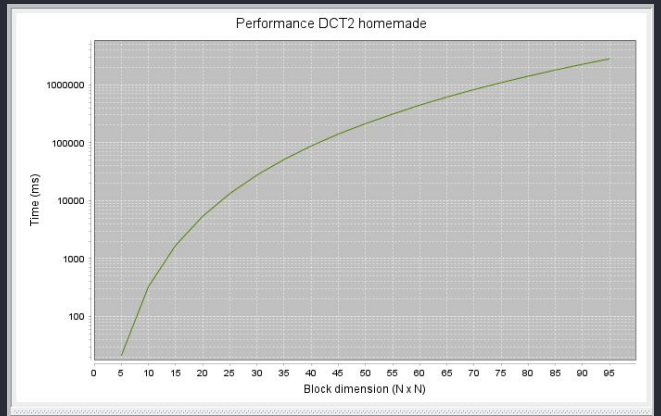
$$f_{ij} = \sum_{k=0}^{N-1} \sum_{\ell=0}^{M-1} c_{k\ell}\, \alpha_{k\ell} \cos\left(k\,\pi \frac{2i+1}{2N}\right) \cos\left(\ell\,\pi \frac{2j+1}{2M}\right) \quad \text{(IDCT2).}$$

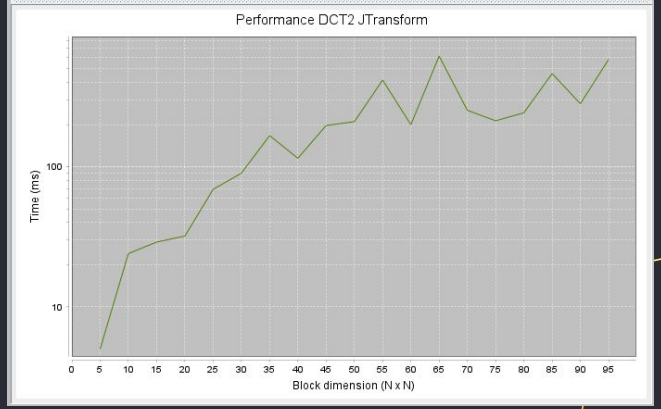
2. Confronto

Come richiesto ci siamo procurati array quadrati di dimensione N x N per simulare un flusso di dati, con N crescente tra N e Nmax per ogni Step
I tempi della libreria JTransform hanno tempistiche irregolari nell'ordine di N^2logN.

La DCT2 homemade invece mantiene un andamento di N^3





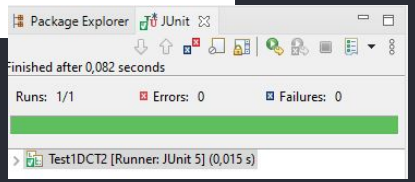


2. Testing

Come richiesto abbiamo testato la nostra DCT2 homemade con la matrice specificata nella consegna del progetto e verificato che i valori corrispondessero.

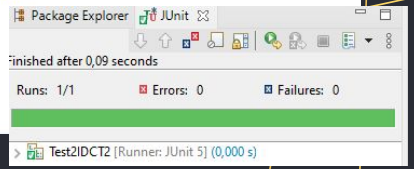
Test1DCT2.java

```java
package test;

import static org.junit.Assert.assertEquals;


class Test1DCT2 {

    public double [][] originalMatrix;
    double [][] originalResult;
    int [][] originalIntMatrix;
    int [][] originalIntResult;


    public Test1DCT2() {
        originalMatrix = new double [][] {
            {231,32, 233, 161, 24, 71, 140, 245},
            {247, 40, 248, 245, 124, 204, 36, 107},
            {234, 202, 245, 167, 9, 217, 239, 173},
            {193, 190, 100, 167, 43, 180, 8, 70},
            {11, 24, 210, 177, 81, 243, 8, 112},
            {97, 195, 203, 47, 125, 114, 165, 181},
            {193, 70, 174, 167, 41, 30, 127, 245},
            {87, 149, 57, 192, 65, 129, 178, 228}
        };

        originalResult = new double [][] {
            {1118 , 44 ,75.9 ,-138, 3.50 ,122 ,195 ,-101},
            {77.1 ,114 ,-21.8 ,41.3 ,8.77 ,99 ,138, 10.9},
            {44.8 ,-62.7, 111, -76.3, 124, 95.5, -39.8, 58.51},
            {-69.9 ,-40.2 ,-23.4 ,-76.7 ,26.6 ,-36.8 ,66.1 ,125},
            {-109 ,-43.3, -55.5 ,8.17, 30.2, -28.6 ,2.44, -94.1},
            {-5.38 ,56.6 ,173 ,-35.4 ,32.3 , 33.4 ,-58.1 , 19.0},
            {78.8 ,-64.5, 118, -15.0, -137, -30.6, -105 ,39.8},
            {19.7, -78.1, 0.972, -72.3 ,-21.5 ,81.3 ,63.7 ,5.90}
        };

        originalIntMatrix = MatrixOperations.doubleMatrixToInt(originalMatrix);
        originalIntResult = MatrixOperations.doubleMatrixToInt(originalResult);
    }

    @SuppressWarnings("deprecation")
    @Test
    void test() {
        double [][] matrix = new double [originalMatrix.length][originalMatrix[0].length];
        int [][] dct2matrix = new int [matrix.length][matrix[0].length];
        matrix = DCT2.applyDCT2(originalMatrix);
        dct2matrix = MatrixOperations.doubleMatrixToInt(matrix);
        assertEquals(originalIntResult, dct2matrix);
    }
}
```



Test2IDCT2.java

```java
package test;

import static org.junit.Assert.assertEquals;
import org.junit.jupiter.api.Test;
import utils.DCT2;
import utils.MatrixOperations;

class Test2IDCT2 {

    double [][] originalMatrix;
    double [][] originalResult;
    int [][] originalIntMatrix;
    int [][] originalIntResult;


    public Test2IDCT2() {
        originalMatrix = new double [][] {
            {231,32, 233, 161, 24, 71, 140, 245},
            {247, 40, 248, 245, 124, 204, 36, 107},
            {234, 202, 245, 167, 9, 217, 239, 173},
            {193, 190, 100, 167, 43, 180, 8, 70},
            {11, 24, 210, 177, 81, 243, 8, 112},
            {97, 195, 203, 47, 125, 114, 165, 181},
            {193, 70, 174, 167, 41, 30, 127, 245},
            {87, 149, 57, 192, 65, 129, 178, 228}
        };

        originalResult = new double [][] {
            {1118 , 44 ,75.9 ,-138, 3.50 ,122 ,195 ,-101},
            {77.1 ,114 ,-21.8 ,41.3 ,8.77 ,99 ,138, 10.9},
            {44.8 ,-62.7, 111, -76.3, 124, 95.5, -39.8, 58.51},
            {-69.9 ,-40.2 ,-23.4 ,-76.7 ,26.6 ,-36.8 ,66.1 ,125},
            {-109 ,-43.3, -55.5 ,8.17, 30.2, -28.6 ,2.44, -94.1},
            {-5.38 ,56.6 ,173 ,-35.4 ,32.3 , 33.4 ,-58.1 , 19.0},
            {78.8 ,-64.5, 118, -15.0, -137, -30.6, -105 ,39.8},
            {19.7, -78.1, 0.972, -72.3 ,-21.5 ,81.3 ,63.7 ,5.90}
        };

        originalIntMatrix = MatrixOperations.doubleMatrixToInt(originalMatrix);
        originalIntResult = MatrixOperations.doubleMatrixToInt(originalResult);
    }
    @SuppressWarnings("deprecation")
    @Test
    void test() {


        double [][] matrixA = new double [originalMatrix.length][originalMatrix[0].length];
        double [][] matrixB = new double [originalMatrix.length][originalMatrix[0].length];
        int [][] matrixInt = new int [matrixA.length][matrixA[0].length];

        matrixA=DCT2.applyDCT2(originalMatrix);
        matrixB=DCT2.applyIDCT2(matrixA);
        matrixInt = MatrixOperations.doubleMatrixToInt(matrixB);


        assertEquals (originalIntMatrix, matrixInt);

    }
}
```

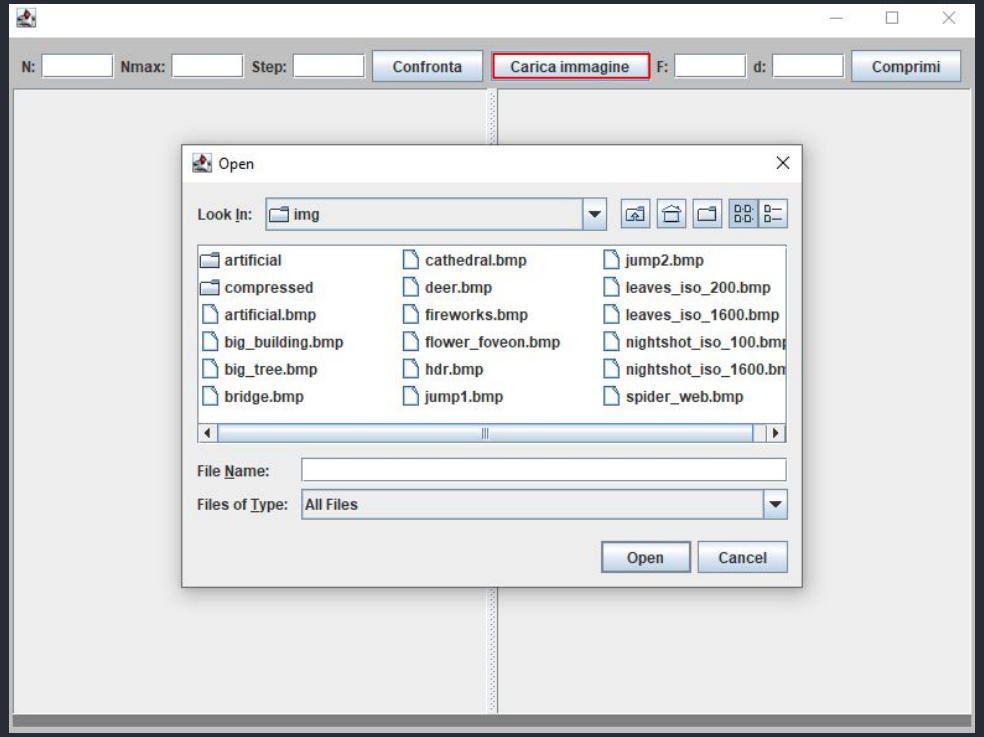


3

SECONDA PARTE

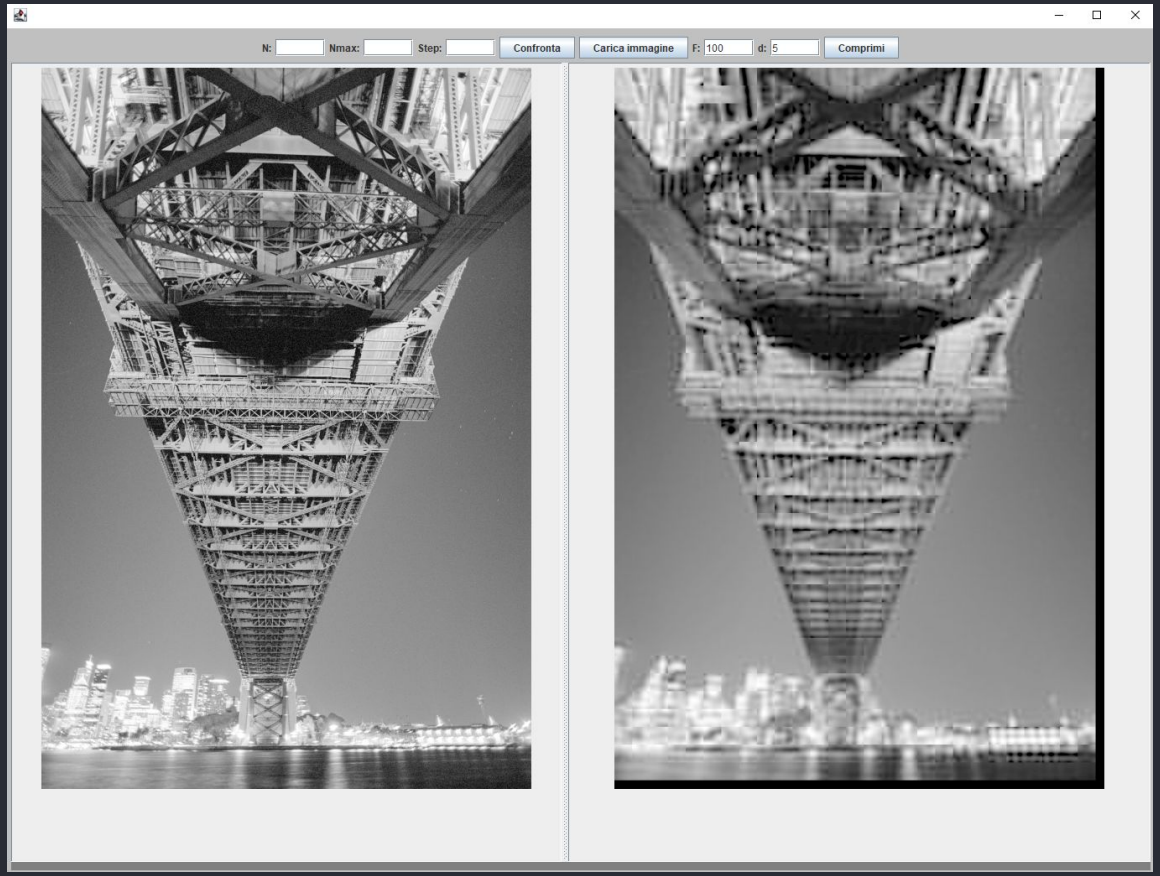
3. Seconda Parte

- Interfaccia che permette all'utente di scegliere dal filesystem un'immagine .bmp in toni di grigio.



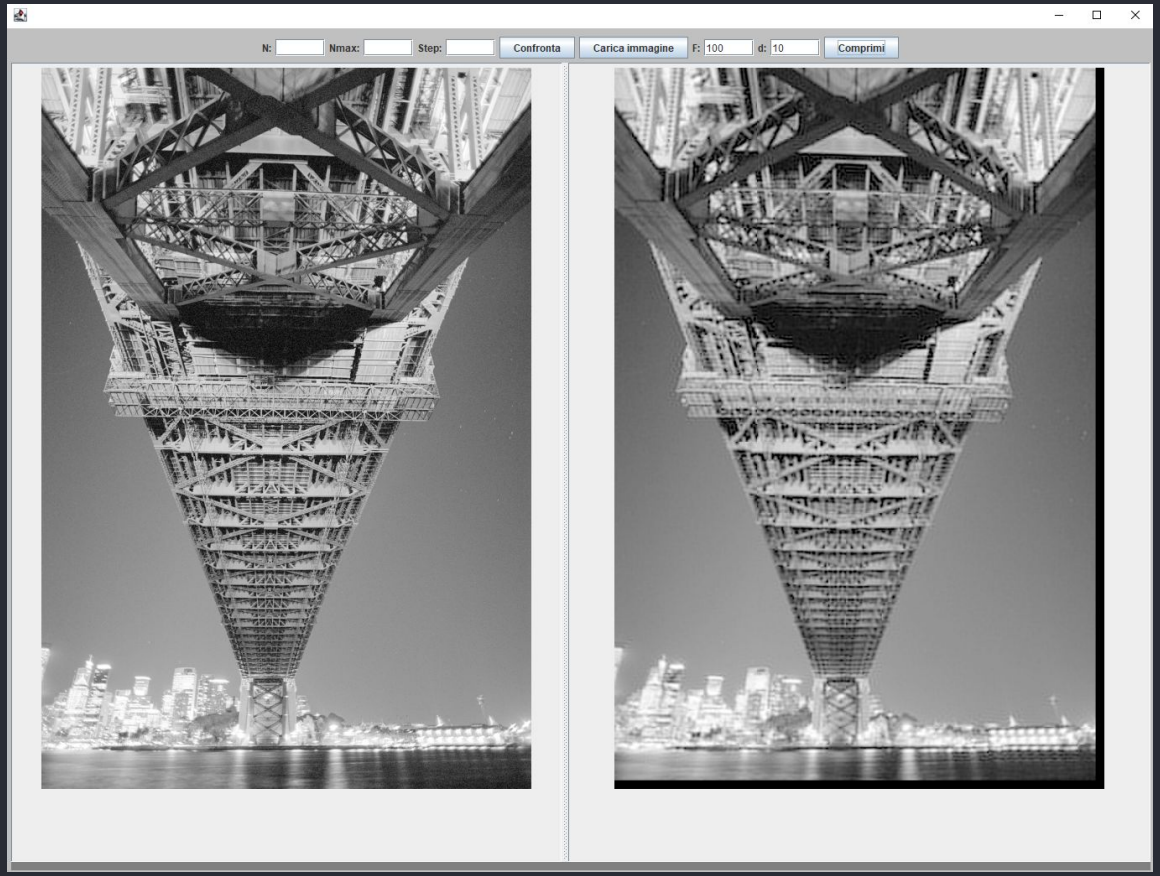
3. Seconda Parte

- Esecuzione della compressione con parametri F = 100 e d = 5 dell'immagine "bridge.bmp"



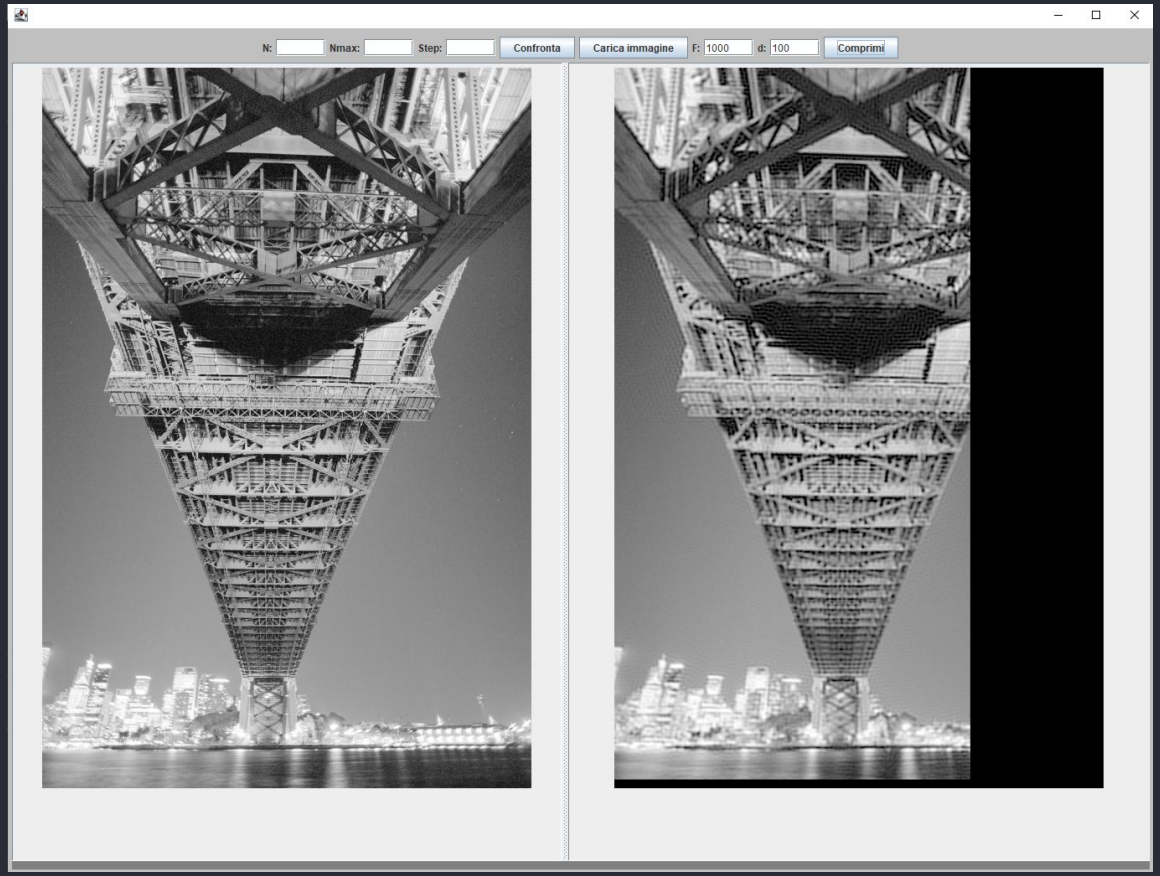
3. Seconda Parte

- Esecuzione della compressione con parametri F = 100 e d = 10 dell'immagine "bridge.bmp"



3. Seconda Parte

- Esecuzione della compressione con parametri F = 1000 e d = 100 dell'immagine "bridge.bmp"



4

CONSIDERAZIONI

4. Considerazioni

Come dimostrato nei 3 esempi precedenti si può evincere che a parità di F (dimensione blocco) ma all'aumentare di d (diminuendo quindi le frequenze tagliate) le immagini compresse risultano essere qualitativamente superiori, mentre nell'ultimo esempio possiamo notare come il mantenimento della proporzione tra F e d mantenga anche la stessa qualità dell'immagine, a patto di avere delle zone nere più evidenti dovute ai blocchi restanti (più piccoli di F) che non vengono considerati dall'algoritmo.