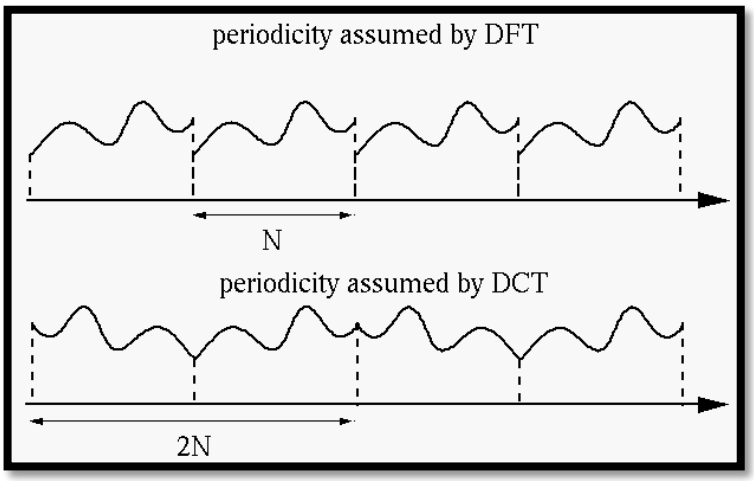
# Discrete Cosine Transform

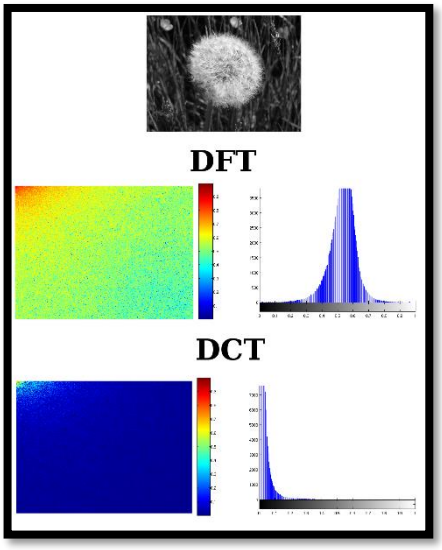
## Introduzione:

La codifica di audio, immagini e video tramite trasformate costituisce una componente fondamentale nelle applicazioni contemporanee. Le codifiche che utilizzano le trasformate si basano sulla premessa che i pixel di un'immagine presentano un certo livello di correlazione con i pixel vicini. Allo stesso modo in un sistema di trasmissione video i pixel adiacenti in fotogrammi consecutivi mostrano una correlazione molto alta e un discorso simile si può fare anche per quanto riguarda le tracce audio. Di conseguenza, queste correlazioni possono essere sfruttate per prevedere il valore di un pixel dai rispettivi pixel vicini. Quindi un utilizzo di queste trasformate è quello di sfruttare le ridondanze nei dati per fornire una compressione. In altre parole, se si prende come esempio le immagini, significa diminuire il numero medio di bit necessari per rappresentare l'immagine.

La **Trasformata discreta del coseno** o **DCT** è strettamente correlata alla Trasformata discreta di Fourier (DFT) con alcune differenze.

- Il DCT è puramente reale, mentre il DFT lavora pure su valori complessi
- Il DCT è più efficiente nel "energy compaction" rispetto al DFT, cioè la capacità di impacchettare l'energia della sequenza nel minor numero possibile di coefficienti di frequenza
- DCT è una trasformata lineare che prende come input un segnale reale x di lunghezza N e fornisce come uscita un segnale reale X di lunghezza N. Può essere anche calcolata prendendo DFT del segnale y di lunghezza 2 * N, che è la concatenazione del segnale con il suo contrario, cioè y = {x[0], ... x[N-1], x[N-1], ..., x[0]}. Poiché y è simmetrica, la sua DFT è reale




*Confronto tra la DFT e la DCT. Lo spettro della DFT è più diffuso dello spettro della DCT. La DCT concentra le informazioni nelle basse frequenze*

## Definizione:

La definizione DCT più comune di una sequenza 1-D di lunghezza N è chiamate DCT-II ed è la seguente:

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) cos\left[\frac{\pi(2x+1)u}{2N}\right] \qquad dove\ \alpha(u) = \begin{cases} \sqrt{1/N}, & se\ u = 0 \\ \sqrt{2/N}, & se\ u \neq 0 \end{cases}$$

È possibile anche definire la DCT in 2-D come diretta estensione di quella in 1-D nel modo seguente:

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) cos\left[\frac{\pi(2x+1)u}{2N}\right] cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

Utilizzata per processare immagini, e questa sarà la formula su cui ci concentreremo maggiormente.

## Proprietà:

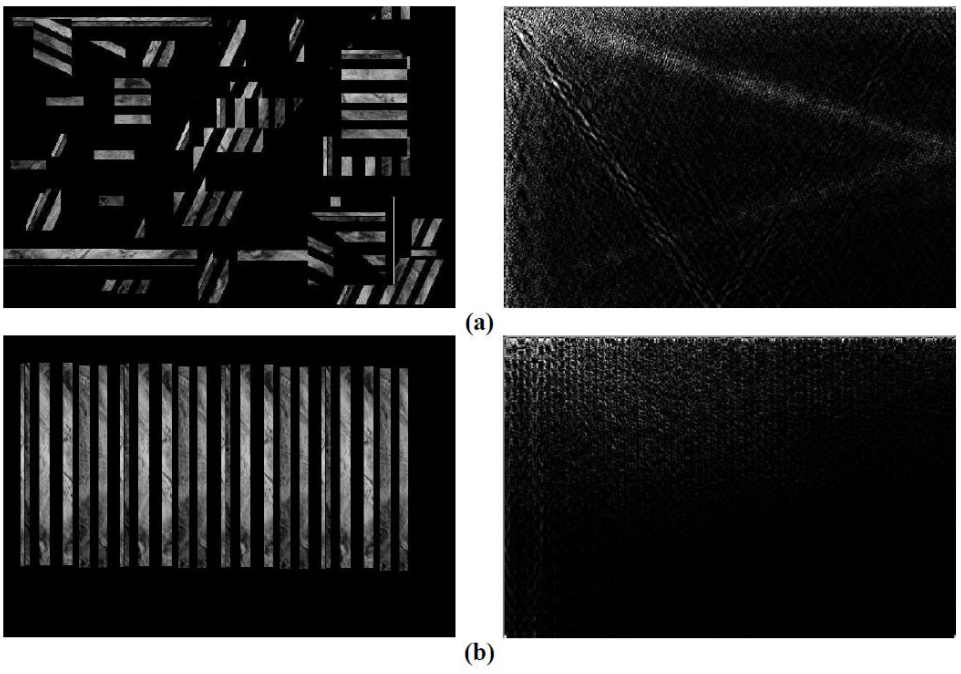
Questa sezione delinea alcune proprietà della DCT in 2-D che hanno un particolare valore per le applicazioni di elaborazione delle immagini.

**Decorrelazione:**

Come discusso in precedenza, il principale vantaggio della trasformazione dell'immagine è la rimozione della ridondanza tra pixel vicini, cioè la somiglianza tra pixel vicini. Ciò porta a coefficienti di trasformazione non correlati che possono essere codificati in modo indipendente. Tutto ciò mira a de-correlare i dati originali e a compattare una grande porzione dei dati dell'immagine in relativamente pochi coefficienti di trasformazione.

**Energy Compaction:**

Se definiamo l'energia di una matrice come la somma dei quadrati dei valori abbiamo che la DCT, pur mantenendo invariata l'energia dell'input, la ridistribuisce in un numero inferiore di coefficienti. Ciò consente di scartare i coefficienti con ampiezze relativamente piccole senza introdurre distorsioni visive nell'immagine ricostruita. DCT esibisce un'eccellente compattazione energetica per immagini altamente correlate.



*Consideriamo le 2 immagini a sinistra con la sua DCT accanto. L'immagine non correlata (a) presenta variazioni di intensità più nette rispetto all'immagine correlata(b). Pertanto, il primo ha più contenuto ad alta frequenza rispetto al secondo. Chiaramente, l'immagine non correlata ha la sua energia distribuita, mentre l'energia dell'immagine correlata è impacchettata nella regione di bassa frequenza (cioè, in alto a sinistra).*

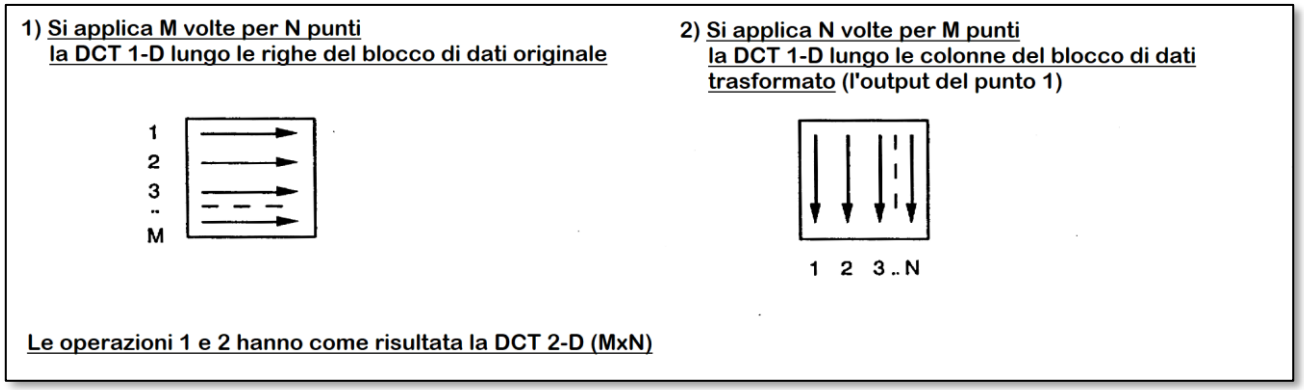
**Separabilità:**

Tutte le DCT multidimensionali possono essere scomposte in successive applicazioni della trasformata unidimensionale per ogni direzione. Nel nostro caso, per la DCT 2-D possiamo riscriverla come il prodotto di due trasformate unidimensionali, la prima che lavora sulle righe mentre la seconda sulle colonne, questa proprietà è detta separabilità ed è utile per ridurre la complessità dell'algoritmo.

Equazione riscritta separando il calcolo delle colonne da quello delle righe:

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\left(cos\left[\frac{\pi(2x+1)u}{2N}\right]\left(\sum_{y=0}^{N-1} f(x,y)\ cos\left[\frac{\pi(2y+1)v}{2N}\right]\right)\right)$$

Quindi un possibile algoritmo per la DCT 2-D dovrebbe prima applicare un algoritmo DCT 1-D alle righe del blocco di input, applicando successivamente all'output un algoritmo DCT 1-D alle colonne del blocco di dati trasformato. Le due applicazioni possono essere invertire e il risultato ottenuto sarà sempre lo stesso.




**Simmetria:**

Un altro sguardo alle operazioni di riga e colonna nella formula data grazie alla separabilità rivela che queste operazioni sono identiche. Tale trasformata è chiamata trasformata simmetrica. Una trasformata separabile e simmetrica può essere espressa nella forma

$$T = AfA',$$

dove A è una matrice di trasformazione simmetrica N × N con valori $a(i,j)$ ottenuti da

$$a(i,j) = \alpha(j)\sum_{j=0}^{N-1} cos\left[\frac{\pi(2j+1)i}{2N}\right],$$

e *f* è la matrice dell'immagine NxN. Questa proprietà è estremamente utile perché è possibile precalcolare la matrice di trasformazione e applicarla poi all'immagine, migliorando così l'efficienza dell'algoritmo.

## Funzione inversa:

La funzione inversa della DCT in 2-D è descritta nel seguente modo:

$$f(x,y) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\alpha(u)\alpha(v)\ C(u,v)\ cos\left[\frac{\pi(2x+1)u}{2N}\right]cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

Ed è possibile definirla anche come moltiplicazioni di matrici. Ed essendo A una matrice ortonormale, la sua inversa è la stessa della sua trasposta($A^{-1}$=A'). Pertanto, la DCT 2-D inversa può essere scritta come:

$$f = A'TA.$$

## Complessità:

La Trasformata discreta del coseno in una dimensione è sostanzialmente una sommatoria da 0 a N-1, quindi sono necessarie N operazioni, ognuna da svolgere per ogni cella dell'array di dimensione N. Questo porta, per come abbiamo definito noi la DCT, ad una complessità quadratica $T(N)=O(N^2)$. Nonostante ciò è possibile calcolare la stessa cosa con la sola complessità di O (N log N) fattorizzando il calcolo in modo simile alla trasformata di Fourier veloce (FFT). Questo è possibile perché si può esprimere la DCT e la sua inversa in termini di una DFT per cui esiste l'algoritmo radix-2 Fast Fourier Transform (FFT), che ottimizzato per trattare con segnali dispari e senza la parte immaginaria, si ottiene una complessità di $O(N \log N)$.Ciò porterebbe, grazie alla proprietà di separabilità della DCT 2-D, a ridurre la complessità da $O(N^4)$ a $O(2N^2 \log N)$.

Un'altra nota che si può fare sulla complessità riguarda la dimensione della matrice trattata. Essendoci delle sommatorie, il numero di operazioni dell'algoritmo cresce rapidamente con le dimensioni dell'immagine. Per questo motivo la DCT non viene applicata all'intera immagine, ma a piccoli blocchi di 8x8 o 16x16 pixel, nei quali viene scomposta l'immagine e elaborati ognuno separatamente.

## Compressione e perdita di informazione (per immagini):

In teoria la trasformata DCT è fedele, ma in pratica poiché si utilizzano numeri in virgola mobile e funzioni trascendenti si introducono errori di rappresentazione che perdono un po' di informazione, ma questa non è la causa principale.
La DCT, come altre trasformate, permette di passare dalla rappresentazione dei pixel nello spazio, e cioè da una rappresentazione contenente i valori della luminosità di ciascun pixel, a una rappresentazione espressa in termini di frequenza. In questa rappresentazione le onde a bassa frequenza sono quelle che tengono conto dell'andamento generale della luminosità dell'immagine i quali in tutta la matrice sono i più significativi, mentre i valori corrispondenti a alle frequenze alte hanno valori molto bassi e quindi poco significativi. Una delle proprietà già citata in precedenza è quella di avere una concentrazione maggiore rispetto alle altre trasformate dei valori significativi nelle basse frequenze, ciò comporta un minor numero di coefficienti significativi.
Inoltre, dividendo ogni coefficiente della matrice per un certo valore è possibile ridurre ulteriormente le componenti meno significative ottenendo una matrice con numerosi valori nulli. Questa operazione è chiamata *quantizzazione* e comporta il fattore maggiore di perdita di qualità e contemporaneamente una compressione elevata.

## Implementazione di un algoritmo:

Di seguito c'è la descrizione e l'implementazione di alcuni algoritmi in Java dove ho trasposto in codice le varie formule della DCT-II 2D riportate in precedenza.

**Implementazione formula diretta:**

Una prima implementazione di un algoritmo che esegua la DCT può essere quella di prendere direttamente la formula base e trasporla in codice, dove i due cicli interni che lavorano su *x* e *y* rappresentano le sommatorie. Mentre quelli esterni si occupano di scorrere la matrice in cui i salvato il nostro blocco di input che corrisponde ad una parte dell'immagine.

```
private static int N = 8;    //BLOCK SIZE

static private double[][] DCT_V1(double[][] dataInput) {
    double[][] DCT = new double[N][N];

    double sum;
    for (int u = 0; u < N; u++) {
        for (int v = 0; v < N; v++) {
            sum = 0;
            for (int x = 0; x < N; x++) {
                for (int y = 0; y < N; y++) {
                    sum += dataInput[x][y] *
                            Math.cos((Math.PI * (2 * x + 1) * u) / (2 * N)) *
                            Math.cos((Math.PI * (2 * y + 1) * v) / (2 * N));
                }
            }
            DCT[u][v] = alpha(u) * alpha(v) * sum;
        }
    }
    return DCT;
}

static private double alpha(int u) {
    if (u == 0)
        return 1.0 / Math.sqrt(N);
    return Math.sqrt(2.0 / N);
}
```

Come detto precedentemente, questo algoritmo ha una complessità di $O(N^4)$. E quindi per una matrice 8x8 sarebbe necessario eseguire 4096 cicli e per file di grandi dimensioni il tempo necessario sarebbe troppo.

**Ottimizzazione con separabilità:**

Un'ottimizzazione del precedente algoritmo è quello di utilizzare la sua proprietà di separabilità che ci permette si separare le sommatorie. Utilizzando questo approccio ci permette di passare da una complessità di $O(N^4)$ a una di $O(N^3)$. In un caso reale, per una matrice 8x8 si passerebbe da 4096 cicli richiesti dell'algoritmo precedente a 1024, un quarto rispetto al caso precedente.

```
private double[][] DCT_V2_1(double[][] dataInput) {
    double[][] DCT  = new double[N][N];
    double[][] DCT2 = new double[N][N];
    /* transform rows */
    double sum;
    for (int u = 0; u < N; u++) {
        for (int v = 0; v < N; v++) {
            sum = 0;
            for (int y = 0; y < N; y++) {
                sum +=  dataInput[y][v] *
                        Math.cos((Math.PI *  (2 * y + 1) *  u) /  (2 * N));
            }
            DCT[u][v] = alpha(u) * sum;
        }
    } . . .
```

```
. . .
    /* transform columns */
    for (int v = 0; v < N; v++) {
        for (int u = 0; u < N; u++) {
            sum = 0.;
            for (int x = 0; x < N; x++) {
                sum += DCT[u][x] *
                        Math.cos((Math.PI *  (2 * x + 1) *  v) /  (2 * N));
            }
            DCT2[u][v] = alpha(v) * sum;
        }
    }
    return DCT2;
}
```

E dato l'algoritmo precedente, è facile ricavarne uno dove per calcolare la DCT in due dimensioni si utilizza quella monodimensionale applicata prima alle colonne e successivamente alle righe.

```
private double[][] DCT_V2_2(double[][] dataInput) {
    double[][] DCT  = new double[N][N];
    double[][] DCT2 = new double[N][N];
    double[] row_in, column_in;
    double[] row_out, column_out;

    for (int j = 0; j < N; j++) {
        column_in = getColumn(dataInput, j);
        column_out = dct_1d(column_in);
        setColumn(DCT, column_out, j);
    }

    for (int j = 0; j < N; j++) {
        row_in = getRow(DCT, j);
        row_out = dct_1d(row_in);
        setRow(DCT2, row_out, j);
    }
    return  DCT2;
}

private double[] dct_1d(double[] in) {
    double[] out = new double[N];
    double sum;
    for (int u = 0; u < N; u++) {
        sum = 0;
        for (int x = 0; x < N; x++) {
            sum += in[x] *
                    Math.cos((Math.PI * (2 * x + 1) *  u) / (2 * N));
        }
        out[u] = alpha(u) * sum;
    }
    return out;
}
```

**Ottimizzazione con la simmetria:**

Infine, è possibile utilizzando la proprietà della simmetria per calcolare la DCT. Questo metodo ci permette di precalcolando la matrice di coseni, detta Kernel Matrix, e moltiplicarla prima all'l'immagine e il risultato va moltiplicato a sua volta per la Kernel Matrix trasposta. Ciò riduce nettamente i calcoli necessari, considerando che se si vuole dividere la nostra immagine in quadrati 8x8 la Kernel Matrix basterà computarla una volta soltanto. E in modo analogo si può ricavare l'inversa

```
static private double[][] DCT_V3(double[][] f) {
    double[][] DCT = new double[N][N];
    double[][] DCT2 = new double[N][N];
    double[][] A = dct_kernelMatrix();
    double[][] A_t = transposeMatrix(A);
    for (int u = 0; u < N; u++) {      //DCT = A*f
        for (int v = 0; v < N; v++) {
            for (int k = 0; k < N; k++) {
                DCT[u][v] += A[u][k] * f[k][v];
            }
        }
    }
    for (int u = 0; u < N; u++) {      //DCT2 = DCT*A'
        for (int v = 0; v < N; v++) {
            for (int k = 0; k < N; k++) {
                DCT2[u][v] += DCT[u][k] * A_t[k][v];
            }
        }
    }
    return DCT2; //DCT2 = A*f*A'
}
static private double[][] iDCT_V3(double[][] T) {//Funzione inversa della DCT
    double[][] _f = new double[N][N];
    double[][] f = new double[N][N];
    double[][] A = dct_kernelMatrix();
    double[][] A_t = transposeMatrix(A);
    for (int u = 0; u < N; u++) {
        for (int v = 0; v < N; v++) {
            for (int k = 0; k < N; k++) {
                _f[u][v] += A_t[u][k] * T[k][v];
            }
        }
    }
    for (int u = 0; u < N; u++) {
        for (int v = 0; v < N; v++) {
            for (int k = 0; k < N; k++) {
                f[u][v] += _f[u][k] * A[k][v];
            }
        }
    }

    return f; // f = A' * T * A
}
static private double[][] dct_kernelMatrix() {
    double[][] KM = new double[N][N];
    double alpha1 = 1 / Math.sqrt(N);
    double alpha2 = Math.sqrt(2.0 / N);
    for (int q = 0; q < N; q++) {
        KM[0][q] = alpha1;
    }
    for (int p = 1; p < N; p++) {
        for (int q = 0; q < N; q++) {
            KM[p][q] = alpha2 *
                    Math.cos((Math.PI * (2 * q + 1) *  p) / (2 * N));
        }
    }
    return KM;
}
```

## Applicazione dell’algoritmo su un’immagine:

Di seguito è riportato ciò che succede applicando l’ultimo algoritmo mostrato a due immagini differenti. Entrambe sono di dimensioni 8x8, ma differiscono per la somiglianza tra un pixel e quelli vicino, la correlazione. La prima è un'immagine molto correlata essendo in una scala di grigi molto simili tra loro, mentre la seconda è stata generata con valori casuali perciò c’è poco correlazione tra i pixel che compongono l’immagine. Si può effettivamente notare che più ci si allontana dalle basse frequenze più i valore tendono a essere nulli per la prima immagine, mentre per la seconda non vale la stessa cosa.

```
IMMAGINE ORIGINALE:
[ 154 152 154 155 160 158 158 150
 151 151 151 150 159 164 160 143
 155 155 159 155 154 155 161 163
 166 158 159 158 159 157 159 160
 157 157 162 154 155 158 159 156
 162 153 156 154 162 165 160 156
 163 156 166 168 168 162 158 163
 163 154 158 158 167 174 171 168]

DCT DELL'IMMAGINE CON COEFICENTI ABBASSATI DI 128:
[244,250 -9,122 -3,415 11,486 -1,750 4,178 5,474 6,757
-23,967 4,007 -1,798 0,229 -6,134 0,320 -6,089 0,204
3,808 -8,683 -7,993 8,034 0,000 0,138 -0,568 -0,119
-7,857 6,051 -6,260 3,185 -0,648 -0,555 0,216 0,839
3,750 -0,294 4,405 0,069 -0,250 0,347 -0,472 0,196
8,199 6,550 0,139 -8,694 10,683 0,212 0,307 -0,121
0,046 -5,994 6,932 -0,179 -0,000 -0,254 0,493 0,093
0,145 0,565 -0,025 -12,148 -0,448 -0,160 0,306 0,097 ]

IMMAGINE RICOSTRUITA DALLA DCT:
[ 153 151 154 154 160 157 158 149
 150 151 151 149 159 163 160 143
 155 155 159 154 154 154 161 162
 165 157 158 157 159 156 159 159
 157 157 162 154 155 158 159 156
 161 152 155 153 161 164 159 155
 163 156 166 168 168 161 158 163
 162 153 157 157 167 173 171 167]

DIFFERENZA TRA L'IMMAGINE ORIGINALE E QUELLA TRASFORMATA:
1 1 0 1 0 1 0 1
1 0 0 1 0 1 0 0
0 0 0 1 0 1 0 1
1 1 1 1 0 1 0 1
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 0 0 0 1 0 0
1 1 1 1 0 1 0 1
```

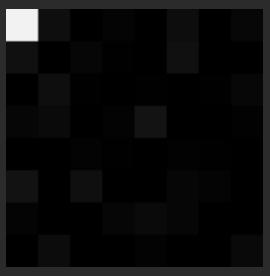


IMMAGINE ORIGINALE:

```
[ 112 159  23  33   2 141 165  13
 101 217  19 112  42  84   0 103
  23 212   0  45   7 174  19   1
  18 103  25 235   0 240  55 162
 167 255  44  24 115  52  30 204
 144  27  46 139  90 138  22 215
 205  34  23 135   6  58 172   0
  41  51  10   0  24 141  58  13]
```

DCT DELL'IMMAGINE CON COEFICENTI ABBASSATI DI 128:

```
[-358,000 29,038 108,205 98,553 -33,250 -47,952 -24,254 -248,231
25,142 47,929 0,456 -17,356 -100,302 -91,536 -64,185 -90,033
-120,738 30,431 3,685 96,244 -98,252 113,397 -40,300 90,344
108,168 8,419 98,764 -14,700 111,311 34,111 1,337 96,570
12,750 -53,628 15,867 1,650 -64,000 -40,979 -0,890 2,322
3,128 -56,750 -59,559 45,530 -54,424 203,060 -1,495 -47,102
-51,733 -58,370 -34,300 37,779 -54,283 -56,459 61,315 -1,396
-34,514 111,386 86,574 60,057 -79,757 -41,176 -98,214 98,711 ]
```

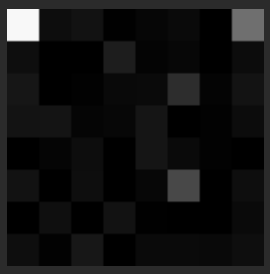


IMMAGINE RICOSTRUITA DALLA DCT:

```
[ 113 158  24  34   2 141 164  14
 101 217  19 113  42  85   0 103
  24 211   0  46   7 174  19   1
  19 104  26 235   1 239  55 161
 166 254  44  24 115  53  31 203
 144  28  47 139  91 138  23 214
 204  35  23 134   6  58 171   0
  42  52  11   1  24 140  58  14]
```

DIFFERENZA TRA L'IMMAGINE ORIGINALE E QUELLA TRASFORMATA:

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