1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

Corso di

Programmazione Logica e Funzionale

Anno accademico

2021-2022

Progetto realizzato da

Barzotti Cristian

290725

Kania Nicholas

291188

Corso tenuto dal professore

Bernardo Marco

Implementazione di DCT e DFT in Haskell e Prolog

Indice

Capitolo 1

Specifica del problema

Si propone di implementare le funzioni denominate DCT (*Discrete Cosine Transform*) e DFT (*Discrete Fourier Transform*), e le relative funzioni inverse IDCT e IDFT.

Nota

Ci teniamo a precisare che la funzione DCT è suddivisa in diversi tipi; in questo progetto siamo andati ad implementare quelle che formalmente sono chiamate DCT tipo II e DCT tipo III[1].
In questo progetto, e relativa documentazione, ci riferiremo alle funzioni rispettivamente come DCT (per la funzione DCT tipo II) e IDCT (per la funzione DCT tipo III).

[1] `https://en.wikipedia.org/wiki/Discrete_cosine_transform`

Capitolo 2

Analisi del problema

2.1 Dati di ingresso del problema

- una lista di numeri reali per il calcolo di DCT e IDCT;
- una lista di numeri complessi per il calcolo di DFT e IDFT.

2.2 Dati di uscita del problema

- una lista di numeri reali calcolati con DCT;
- una lista di numeri reali calcolati con IDCT;
- una lista di numeri complessi calcolato con DFT;
- una lista di numeri complessi calcolato con IDFT.

2.3 Relazioni intercorrenti tra i dati del problema

Tutte le formule che andremo ad utilizzare all'interno del programma seguono la seguente logica: data una sequenza di numeri iniziale di lunghezza N, verrà creata una nuova sequenza di numeri anch'essa di lunghezza N. Tutte le formule sono state prese dagli appunti del corso di Elaborazione Segnali ed Immagini.[1]

2.3.1 DCT

Utilizzando la *Discrete Cosine Transform* su una sequenza di numeri reali, il k-esimo elemento sarà calcolato con la seguente formula:

$$C(k) = 2 \sum_{n=0}^{N-1} x(n) cos \left[\frac{\pi(2n+1)k}{2N} \right]$$

.

[1] `https://blended.uniurb.it/moodle/mod/folder/view.php?id=102652`

2.3.2 IDCT

Utilizzando la *Inverse Discrete Cosine Transform* su una sequenza di numeri reali, l' n-esimo elemento sarà calcolato con la seguente formula:

$$x(n) = \frac{1}{2N}\left\{C(0) + \sum_{k=1}^{N-1} 2C(k)cos\left[\frac{\pi(2n+1)k}{2N}\right]\right\}$$

La sequenza di numeri iniziali con cui calcolare la *IDCT* verrà presa dall'output della *DCT* e, a meno di errori di approssimazione, sarà uguale alla sequenza di numeri utilizzata per il calcolo della *DCT*.

2.3.3 DFT

Utilizzando la *Discrete Fourier Transform* su una sequenza di numeri complessi, il k-esimo elemento sarà calcolato con la seguente formula:

$$X_p(k) = \sum_{n=0}^{N-1} x_p(n)e^{-j\frac{2\pi}{N}nk}$$

.

2.3.4 IDFT

Utilizzando la *Inverse Discrete Fourier Transform* su una sequenza di numeri complessi, l' n-esimo elemento sarà calcolato con la seguente formula:

$$x_p(n) = \frac{1}{N}\sum_{k=0}^{N-1} X_p(k)e^{j\frac{2\pi}{N}nk}$$

La sequenza di numeri iniziali con cui calcolare la *IDFT* verrà presa dall'output della *DFT* e, a meno di errori di approssimazione, sarà uguale alla sequenza di numeri utilizzata per il calcolo della *DFT*.

Capitolo 3

Progettazione dell'algoritmo

3.1 Scelte di progetto

Per rendere più intuitiva l'implementazione, abbiamo deciso di utilizzare la forma cartesiana dei numeri complessi al posto della forma esponenziale, sostituendo $e^{-j\frac{2\pi}{N}nk}$ con la forma cartesiana $cos(\theta)+jsin(\theta)$, dove $\theta = -\frac{2\pi}{N}nk$.
Tutti gli input del programma sono liste di numeri e andranno inseriti in maniera opportuna, in base al linguaggio utilizzato.

3.1.1 Haskell

Per i numeri reali, la lista deve essere inserita nel formato:

$$[< numero >, ...]^1$$

Per i numeri complessi, la lista deve essere inserita nel formato:

$$[< parte\ reale >: + < parte\ immaginaria >, ...]^2$$

3.1.2 Prolog

Per i numeri reali, la lista deve essere inserita nel formato:

$$[< numero >, ...].^3$$

Per i numeri complessi, la lista deve essere inserita nel formato:

$$[(< parte\ reale >, < parte\ immaginaria >), ...].^4$$

[1] es: [1, 0.5, -3]
[2] es: [1 :+ 2, 0.5 :+ -3, 0 :+ 0]
[3] es: [1, 0.5, -3].
[4] es: [(1,2), (0.5,-3), (0,0)].

3.2 Passi dell'algoritmo

I passi dell'algoritmo da eseguire per ogni trasformata sono i seguenti:

- chiama la funzione di generazione della lista sull'elemento di indice k;
  - chiama la funzione di generazione della lista sull'elemento $k+1$ fino al caso base;
  - calcola il k-esimo elemento;
    - somma ricorsivamente i valori della lista;
  - lo aggiunge in testa alla lista;
  - restituisce la lista;

Per semplicità di lettura, rinominiamo i precedenti passi come ***calcolo della trasformata***. I passi dell'algoritmo del nostro programma diventeranno quindi:

- inserimento di una lista di numeri reali da tastiera per il calcolo della DCT;
- ***calcolo della trasformata*** DCT;
- stampa della lista;
- ***calcolo della trasformata*** IDCT, con il precedente output come valori di input;
- stampa della lista;
- inserimento di una lista di numeri complessi da tastiera per il calcolo della DFT;
- ***calcolo della trasformata*** DFT;
- stampa della lista;
- ***calcolo della trasformata*** IDFT, con il precedente output come valori di input;
- stampa della lista.

Capitolo 4

Implementazione dell'algoritmo

4.1 Haskell

```
{- IMPORTAZIONE DELLE LIBRERIE -}
{- Libreria necessaria per utilizzare la funzione 'length' -}
import Data.List
{- Libreria utilizzata per la gestione dei numeri complessi -}
import Data.Complex
{- Libreria necessaria per:
    - utilizzare il tipo 'Either';
    - utilizzare la funzione 'readEither' -}
import Text.Read



{- MAIN -}
{- Il programma accetta in input una lista di numeri reali,
    calcolandone e stampandone i risultati:
      - la trasformata discreta del coseno (DCT);
      - relativa trasformata inversa (IDCT).

    Viene poi richiesta in input una lista di numeri complessi,
    calcolandone e stampandone i risultati:
      - la trasformata discreta di Fourier (DFT);
      - relativa trasformata inversa. -}
main :: IO()
main = do
  putStrLn "Progetto della sessione autunnale del corso Programmazione Logica e Funzionale"
  putStrLn "Anno 2021/2022"
  putStrLn "Corso tenuto dal prof. Marco Bernardo"
  putStrLn "Progetto realizzato da: Barzotti Cristian e Kania Nicholas\n\n"


  real_list <- acquire_real_list
  let val_dct = dct real_list

  putStrLn "\nDCT:"
  putStrLn $ show val_dct
```

```
  putStr "\n"
  putStrLn "IDCT:"
  putStrLn $ show (idct val_dct)

  putStr "\n\n"

  complex_list <- acquire_complex_list

  let val_dft = dft complex_list

  putStrLn "\nDFT:"
  putStrLn $ show val_dft
  putStr "\n"
  putStrLn "IDFT:"
  print (stringify_complex_list (idft val_dft))



{- DCT -}
{- Funzione per il calcolo della DCT -}
dct :: [Double] -> [Double]
dct [] = []
dct xs = generate_dct xs (length xs) 0

{- Genera la trasformata -}
generate_dct :: [Double] -> Int -> Int -> [Double]
generate_dct [] _ _ = []
generate_dct xs size k
  | k == size = []
  | otherwise =
    (sum_terms_dct xs size k 0) :
    (generate_dct xs size (k + 1))

{- Effettua il calcolo del k-esimo elemento della trasformata -}
sum_terms_dct :: [Double] -> Int -> Int -> Int -> Double
sum_terms_dct [] _ _ _ = 0.0
sum_terms_dct (x:xs) size k n =
  2 * x *
  cos (pi *  fromIntegral ((2 * n + 1) * k) / fromIntegral (2 * size)) +
  sum_terms_dct xs size k (n + 1)



{- IDCT -}
{- Funzione per il calcolo della IDCT -}
idct :: [Double] -> [Double]
idct [] = []
idct xs = generate_idct xs (length xs) 0

{- Genera la trasformata inversa (serie originale) -}
generate_idct :: [Double] -> Int -> Int -> [Double]
generate_idct [] _ _ = []
generate_idct (x:xs) size n
```

```
  | n == size = []
  | otherwise =
    ((x + sum_terms_idct xs size n 1) / fromIntegral (2 * size)) :
    (generate_idct (x:xs) size (n + 1))

{- Effettua il calcolo della sommatoria per k-esimo elemento della trasformata inversa -}
sum_terms_idct :: [Double] -> Int -> Int -> Int -> Double
sum_terms_idct [] _ _ _ = 0.0
sum_terms_idct (x:xs) size n k =
  2 * x *
  cos (pi *  fromIntegral ((2 * n + 1) * k) / fromIntegral (2 * size)) +
  sum_terms_idct xs size n (k + 1)



{- DFT -}
{- Funzione per il calcolo della DFT -}
dft :: [Complex Double] -> [Complex Double]
dft [] = []
dft xs = generate_dft xs (length xs) 0

{- Genera la trasformata -}
generate_dft :: [Complex Double] -> Int -> Int -> [Complex Double]
generate_dft [] _ _ = []
generate_dft xs size k
  | k == size = []
  | otherwise =
    (sum_terms_dft xs size k 0) :
    (generate_dft xs size (k + 1))

{- Effettua il calcolo del k-esimo elemento della trasformata -}
sum_terms_dft :: [Complex Double] -> Int -> Int -> Int -> Complex Double
sum_terms_dft [] _ _ _ = 0.0
sum_terms_dft (x:xs) size k n =
  let
    theta = - ((2.0 * pi) / (fromIntegral size)) * (fromIntegral n) * (fromIntegral k)
  in
    (x * (cos theta :+ sin theta)) + sum_terms_dft xs size k (n + 1)



{- IDFT -}
{- Funzione per il calcolo della IDFT -}
idft :: [Complex Double] -> [Complex Double]
idft [] = []
idft xs = generate_idft xs (length xs) 0

{- Genera la trasformata inversa (serie originale) -}
generate_idft :: [Complex Double] -> Int -> Int -> [Complex Double]
generate_idft [] _ _ = []
generate_idft xs size n
  | n == size = []
  | otherwise =
```

```
    (sum_terms_idft xs size n 0 / (fromIntegral size)) :
    (generate_idft xs size (n + 1))

{- Calcolo della sommatoria per il k-esimo elemento -}
sum_terms_idft :: [Complex Double] -> Int -> Int -> Int -> Complex Double
sum_terms_idft [] _ _ _ = 0.0
sum_terms_idft (x:xs) size n k =
  let
    theta = ((2.0 * pi) / (fromIntegral size)) * (fromIntegral n) * (fromIntegral k)
  in
    (x * (cos theta :+ sin theta)) + sum_terms_idft xs size n (k + 1)



{- FUNZIONI AUSILIARIE -}
{- Converte una lista di numeri complessi in lista di stringhe -}
stringify_complex_list :: [Complex Double] -> [String]
stringify_complex_list [] = []
stringify_complex_list (x:xs) = (show x) : (stringify_complex_list xs)


{- Acquisisce una lista di numeri reali da tastiera.
    La funzione non termina fino a quando non verrà inserita una lista valida. -}
acquire_real_list :: IO [Double]
acquire_real_list = do
  putStrLn "Inserisci una lista di numeri reali nel formato:\n"
  putStrLn "\t[<numero>, ...]\n"
  putStrLn "Per esempio: [1, 0.5, -3]"
  line <- getLine

  case readEither line :: Either String [Double] of
    Left err -> do
      putStrLn "\nErrore."
      acquire_real_list
    Right value -> return (value)


{- Acquisisce una lista di numeri complessi da tastiera.
    La funzione non termina fino a quando non verrà inserita una lista valida. -}
acquire_complex_list :: IO [Complex Double]
acquire_complex_list = do
  putStrLn "Inserisci una lista di numeri complessi nel formato:\n"
  putStrLn "\t[<parte reale> :+ <parte immaginaria>, ...]\n"
  putStrLn "Per esempio: [1 :+ 2, 0.5 :+ -3, 0 :+ 0]"
  line <- getLine

  case readEither line :: Either String [Complex Double] of
    Left err -> do
      putStrLn "\nErrore."
      acquire_complex_list
    Right value -> return (value)
```

4.2 Prolog

```
/* MAIN */
/* Il programma accetta in input una lista di numeri reali,
   calcolandone e stampandone i risultati:
     - la trasformata discreta del coseno (DCT);
     - relativa trasformata inversa (IDCT).

   Viene poi richiesta in input una lista di numeri complessi,
   calcolandone e stampandone i risultati:
     - la trasformata discreta di Fourier (DFT);
     - relativa trasformata inversa. */
main :-
  nl,
  write('Progetto della sessione autunnale del corso Programmazione Logica e Funzionale'), nl,
  write('Anno 2021/2022'), nl,
  write('Corso tenuto dal prof. Marco Bernardo'), nl,
  write('Progetto realizzato da: Barzotti Cristian e Kania Nicholas'), nl, nl, nl,

  acquire_real_list(RealList), nl,
  dct(RealList, C),
  write('DCT:'), nl,
  write(C), nl, nl,
  idct(C, DctOriginal),
  write('IDCT:'), nl,
  write(DctOriginal), nl, nl,nl,

  acquire_complex_list(ComplexList), nl,
  dft(ComplexList, X),
  write('DFT:'), nl,
  write(X), nl, nl,
  idft(X, DftOriginal),
  write('IDFT:'), nl,
  write(DftOriginal), nl.



/* DCT */
/* Funzione per il calcolo della DCT */
dct([], []).
dct([X | Y], C) :-
  size([X | Y], S),
  generate_dct([X | Y], S, 0, C).

/* Genera la trasformata */
generate_dct([], _, _, []).
generate_dct(L, S, K, C) :-
  ((K == S) ->
    C = [];
    K1 is K + 1,
    generate_dct(L, S, K1, C1),
    sum_terms_dct(L, S, K, 0, E),
    append([E], C1, C)).
```

```
/* Effettua il calcolo del k-esimo elemento della trasformata */
sum_terms_dct([], _, _, _, 0.0).
sum_terms_dct([H | T], S, K, N, C) :-
  N1 is N + 1,
  sum_terms_dct(T, S, K, N1, C1),
  C is C1 + (H * 2 * cos(pi * (2 * N + 1) * K / (2 * S))).



/* IDCT */
/* Funzione per il calcolo della IDCT */
idct([], []).
idct([X | Y], C) :-
  size([X | Y], S),
  generate_idct([X | Y], S, 0, C).

/* Genera la trasformata inversa (serie originale) */
generate_idct([], _, _, []).
generate_idct([H | T], S, N, C) :-
  ((N == S) ->
    C = [];
    N1 is N + 1,
    generate_idct([H | T], S, N1, C1),
    sum_terms_idct(T, S, N, 1, E),
    X is (H + E) / (2 * S),
    append([X], C1, C)).

/* Effettua il calcolo della sommatoria per k-esimo elemento della trasformata inversa */
sum_terms_idct([], _, _, _, 0.0).
sum_terms_idct([H | T], S, N, K, C) :-
  K1 is K + 1,
  sum_terms_idct(T, S, N, K1, C1),
  C is C1 + (H * 2 * cos(pi * (2 * N + 1) * K / (2 * S))).



/* DFT */
/* Funzione per il calcolo della DFT */
dft([], []).
dft([X | Y], C) :-
  size([X | Y], S),
  generate_dft([X | Y], S, 0, C).

/* Genera la trasformata */
generate_dft([], _, _, []).
generate_dft(L, S, K, C) :-
  ((K == S) ->
    C = [];
    K1 is K + 1,
    generate_dft(L, S, K1, C1),
    sum_terms_dft(L, S, K, 0, E),
    append([E], C1, C)).
```

```
/* Effettua il calcolo del k-esimo elemento della trasformata */
sum_terms_dft([], _, _, _, (0,0)).
sum_terms_dft([H | T], S, K, N, C) :-
  N1 is N + 1,
  THETA is (-((2 * pi) / S) * N * K),
  sum_terms_dft(T, S, K, N1, C1),
  complex_prod(H, (cos(THETA),sin(THETA)), P),
  complex_sum(P, C1, C).



/* IDFT */
/* Funzione per il calcolo della IDFT */
idft([], []).
idft([X | Y], C) :-
  size([X | Y], S),
  generate_idft([X | Y], S, 0, C).

/* Genera la trasformata inversa (serie originale) */
generate_idft([], _, _, []).
generate_idft(L, S, N, C) :-
  ((N == S) -> C = [];
  N1 is N + 1,
  generate_idft(L, S, N1, C1),
  sum_terms_idft(L, S, N, 0, E),
  complex_div_real(E, S, P),
  append([P], C1, C)).

/* Calcolo della sommatoria per il k-esimo elemento */
sum_terms_idft([], _, _, _, (0,0)).
sum_terms_idft([H | T], S, N, K, C) :-
  K1 is K + 1,
  THETA is (((2 * pi) / S) * N * K),
  sum_terms_idft(T, S, N, K1, C1),
  complex_prod(H, (cos(THETA),sin(THETA)), P),
  complex_sum(P, C1, C).



/* FUNZIONI AUSILIARIE */
/* Funzione per calcolare il numero di elementi in una lista */
size([], 0).
size([_ | Y], S) :-
  size(Y, S1),
  S is S1 + 1.

/* Funzione che definisce il prodotto fra numeri complessi */
complex_prod((ZR, ZI), (WR, WI), (R,I)) :-
  R is (ZR * WR - ZI * WI),
  I is (ZR * WI + ZI * WR).

/* Funzione che definisce la somma fra numeri complessi */
```

```
complex_sum((ZR, ZI), (WR, WI), (R,I)) :-
  R is ZR + WR,
  I is ZI + WI.

/* Funzione che definisce la divisione fra un numero complesso ed un numero reale */
complex_div_real((ZR, ZI), N, (R, I)) :-
  R is ZR / N,
  I is ZI / N.

/* Acquisisce una lista di numeri reali da tastiera.
   La funzione non termina fino a quando non verrà inserita una lista valida. */
acquire_real_list(List) :-
  repeat,
  write('Inserisci una lista di numeri reali nel formato:'), nl, nl,
  write('\t[<numero>, ...].'), nl, nl,
  write('Per esempio: [1, 0.5, -3].'), nl,
  catch(read(List), Error, true),
  ((check_real_list(List), var(Error)) ->
    !;
    nl, write('Errore.'), nl, fail).

/* Acquisisce una lista di numeri complessi da tastiera.
   La funzione non termina fino a quando non verrà inserita una lista valida. */
acquire_complex_list(List) :-
  repeat,
  write('Inserire una lista di numeri complessi nel formato:'), nl, nl,
  write('\t[(<parte reale>, <parte immaginaria>), ...].'), nl, nl,
  write('Per esempio: [(1,2), (0.5,-3), (0,0)].'), nl,
  catch(read(List), Error, true),
  ((check_complex_list(List), var(Error)) ->
    !;
    nl, write('Errore.'), nl, fail).

/* Controlla che una lista sia composta solo da numeri reali */
check_real_list([]).
check_real_list([A | L]) :-
  ((number(A)) ->
    check_real_list(L);
    fail).

/* Controlla che una lista sia composta solo tuple rappresentanti numeri complessi */
check_complex_list([]).
check_complex_list([(A, B) | L]) :-
  ((number(A), number(B)) ->
    check_complex_list(L);
    fail).
```

Capitolo 5

Testing

5.1 Haskell

Figura 5.1: Input corretto durante l'inserimento per DCT

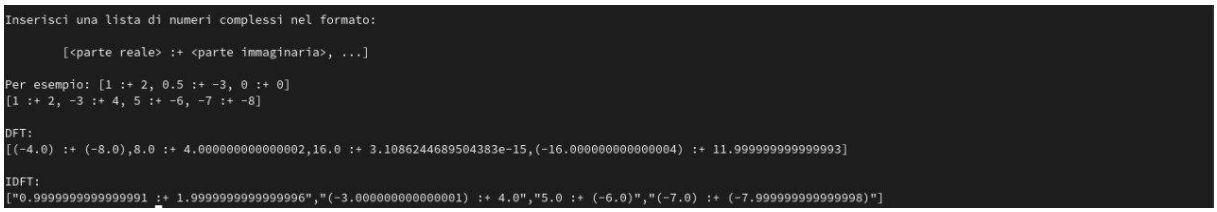

Figura 5.2: Input corretto durante l'inserimento per DFT

Figura 5.3: Input vuoto durante l'inserimento per DCT

Figura 5.4: Input vuoto durante l'inserimento per DFT

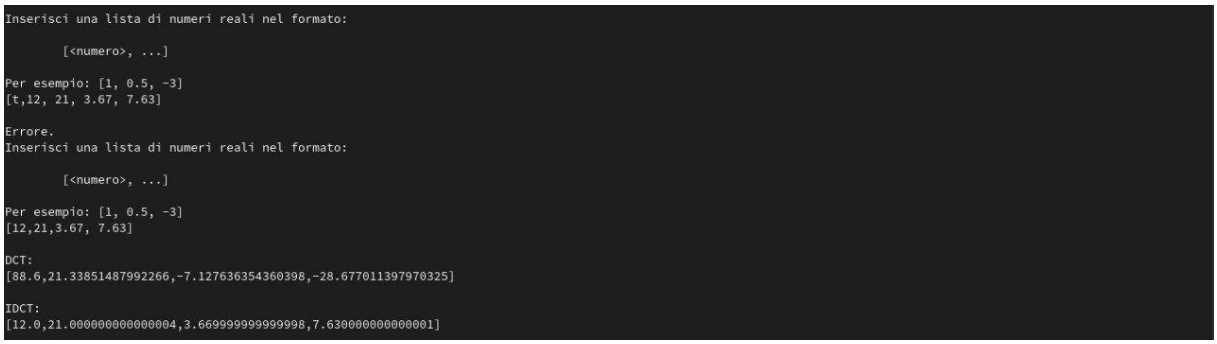

Figura 5.5: Tipo in input errato durante l'inserimento per DCT

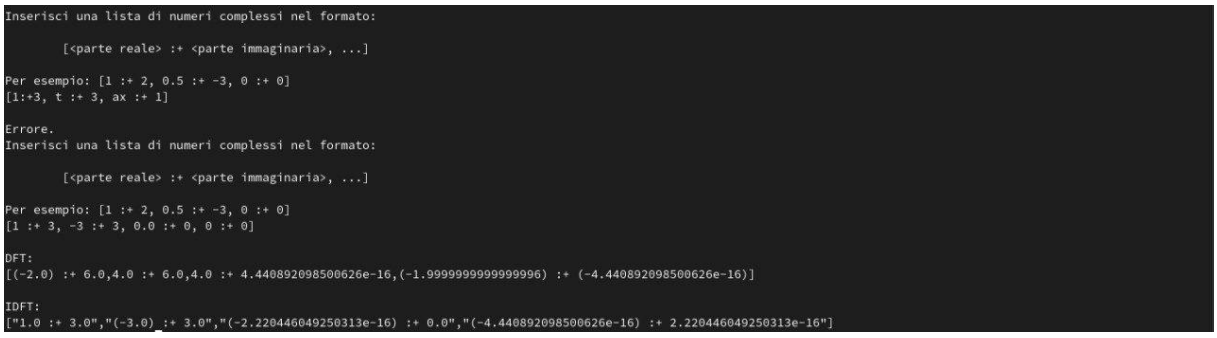

Figura 5.6: Tipo in input errato durante l'inserimento per DFT

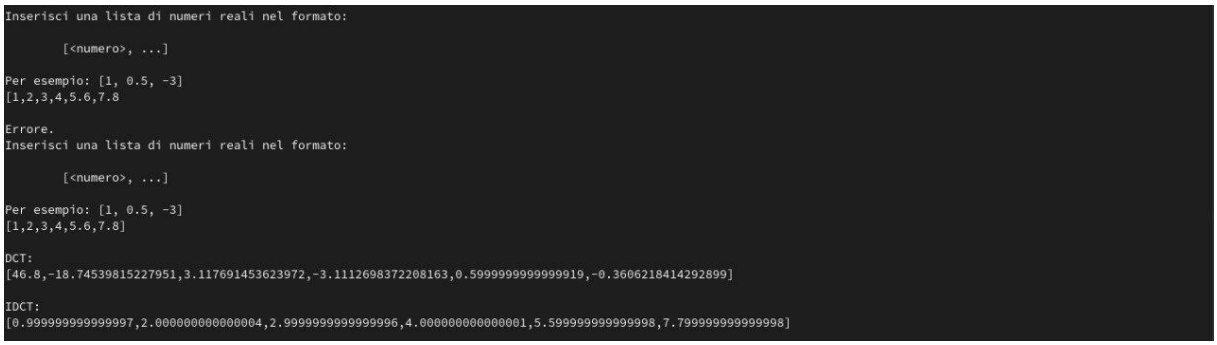

Figura 5.7: Formattazione errata durante l'inserimento per DCT

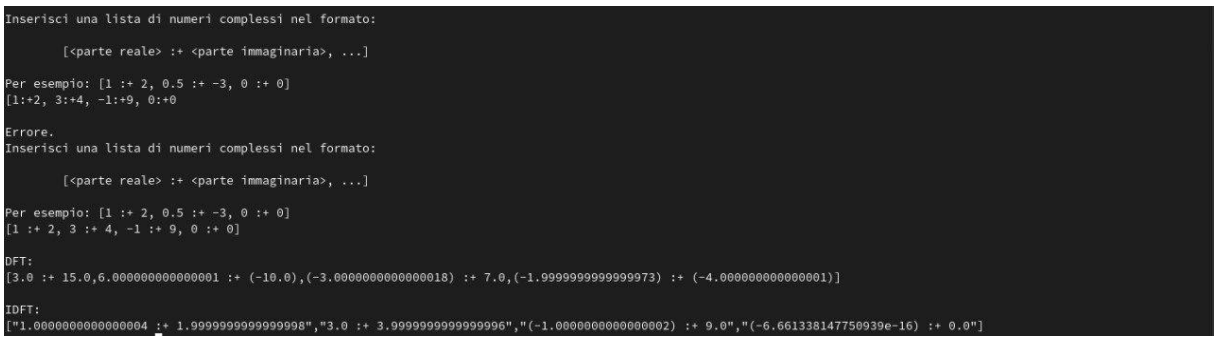

Figura 5.8: Formattazione errata durante l'inserimento per DFT

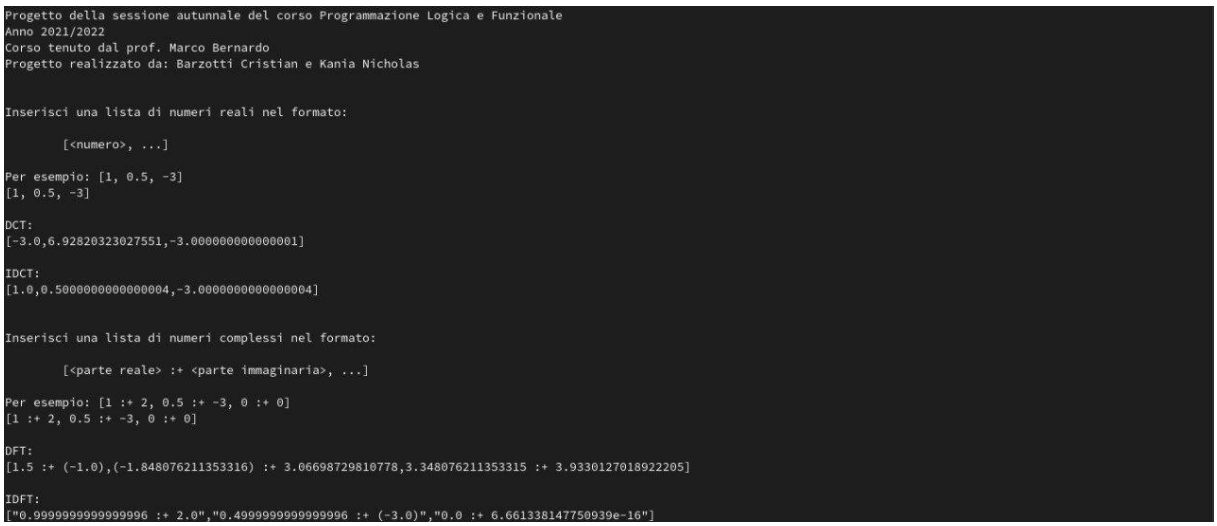

Figura 5.9: Funzionamento dell'intero programma con input corretti

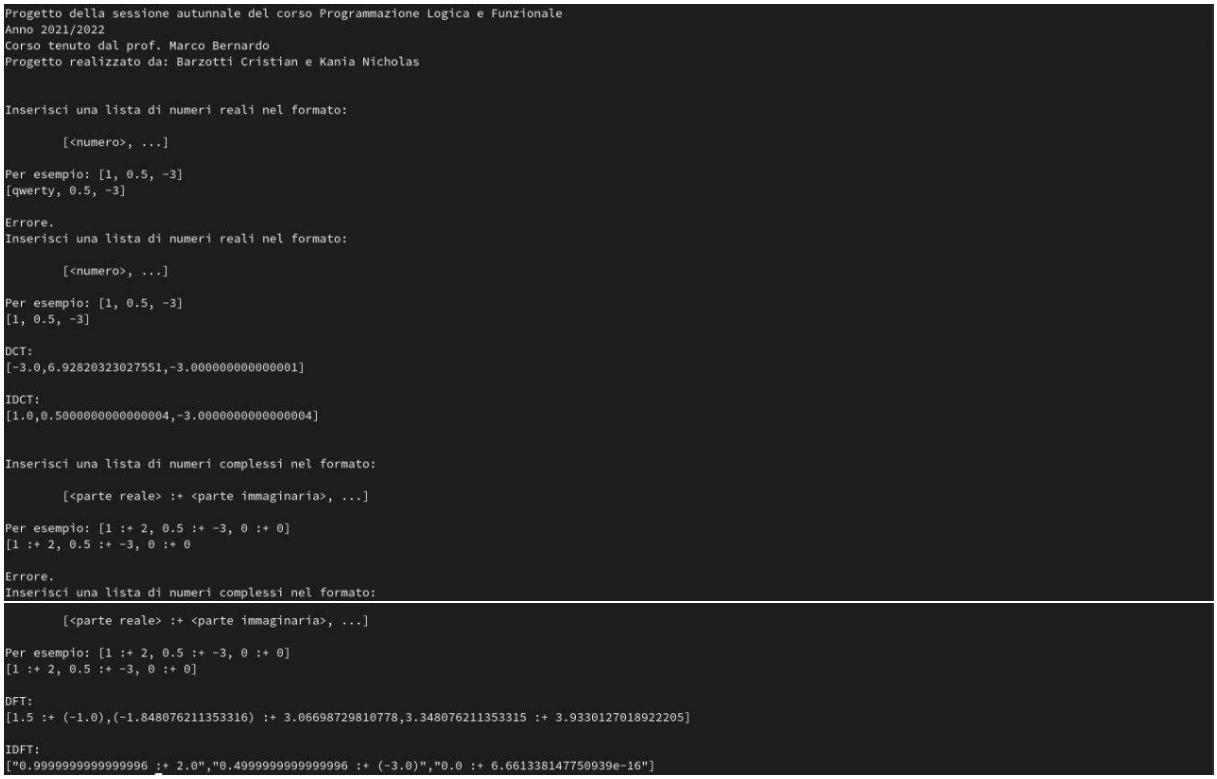

Figura 5.10: Funzionamento dell'intero programma con alcuni input errati

5.2 Prolog

```
| ?- main.

Progetto della sessione autunnale del corso Programmazione Logica e Funzionale
Anno 2021/2022
Corso tenuto dal prof. Marco Bernardo
Progetto realizzato da: Barzotti Cristian e Kania Nicholas


Inserisci una lista di numeri reali nel formato:

        [<numero>, ...].

Per esempio: [1, 0.5, -3].
[1.2, 3.4, 5.6, 7.8, 9.0].

DCT:
[53.999999999999993,-20.008991874378157,-1.6180339887498958,-0.80015259236522684,-0.6180339887498999]

IDCT:
[1.1999999999999993,3.3999999999999999,5.5999999999999988,7.7999999999999998,8.9999999999999982]
```

Figura 5.11: Input corretto durante l'inserimento per DCT

```
Inserire una lista di numeri complessi nel formato:

        [(<parte reale>, <parte immaginaria>), ...].

Per esempio: [(1,2), (0.5,-3), (0,0)].
[(1.2,3.4), (5.6,7.8), (9.0, 1.1)].

DFT:
[(15.799999999999999,12.300000000000001),(-0.29762979464426276,1.8944863728670893),(-11.902370205355732,-3.9944863728671014)]

IDFT:
[(1.2000000000000017,3.3999999999999964),(5.5999999999999988,7.7999999999999998),(8.9999999999999982,1.1000000000000014)]
```

Figura 5.12: Input corretto durante l'inserimento per DFT

Figura 5.13: Input vuoto durante l'inserimento per DCT

Figura 5.14: Input vuoto durante l'inserimento per DFT

Figura 5.15: Tipo in input errato durante l'inserimento per DCT

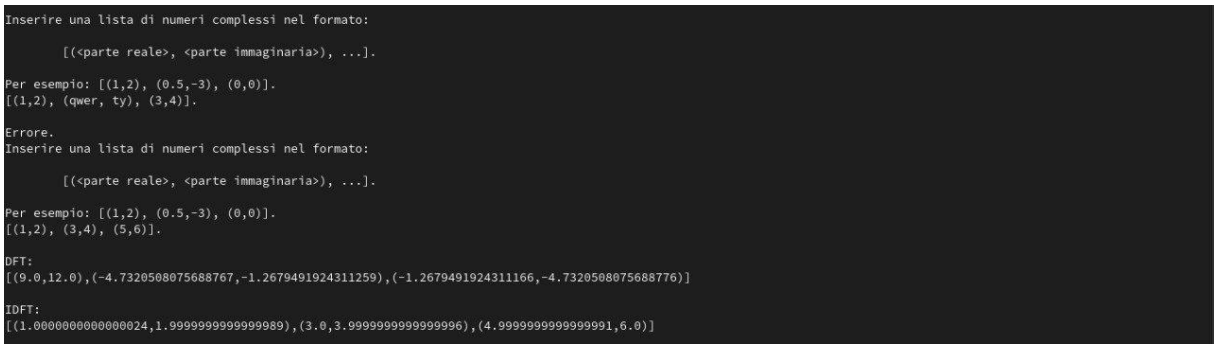

Figura 5.16: Tipo in input errato durante l'inserimento per DFT

Figura 5.17: Formattazione errata durante l'inserimento per DCT

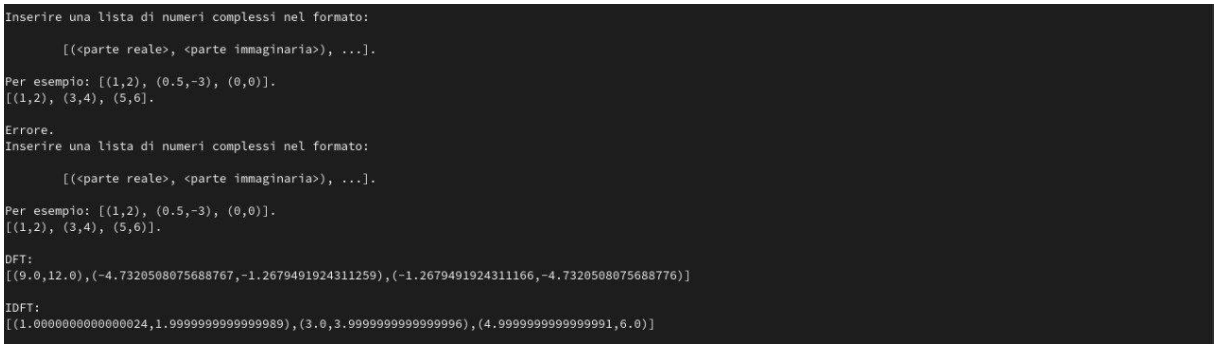

Figura 5.18: Formattazione errata durante l'inserimento per DFT

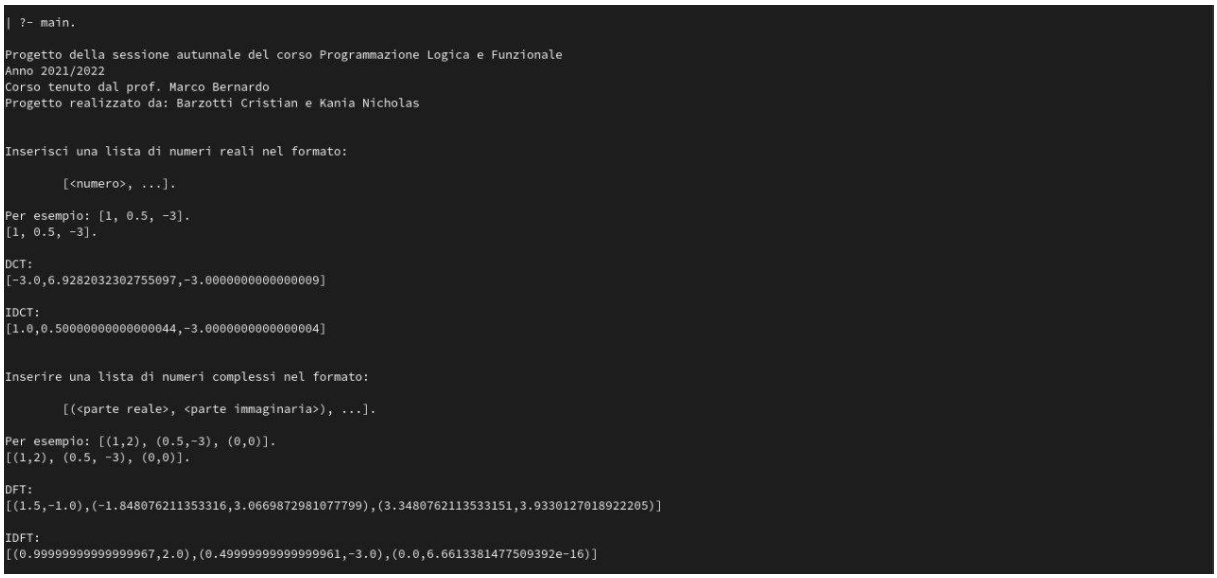

Figura 5.19: Funzionamento dell'intero programma con input corretti

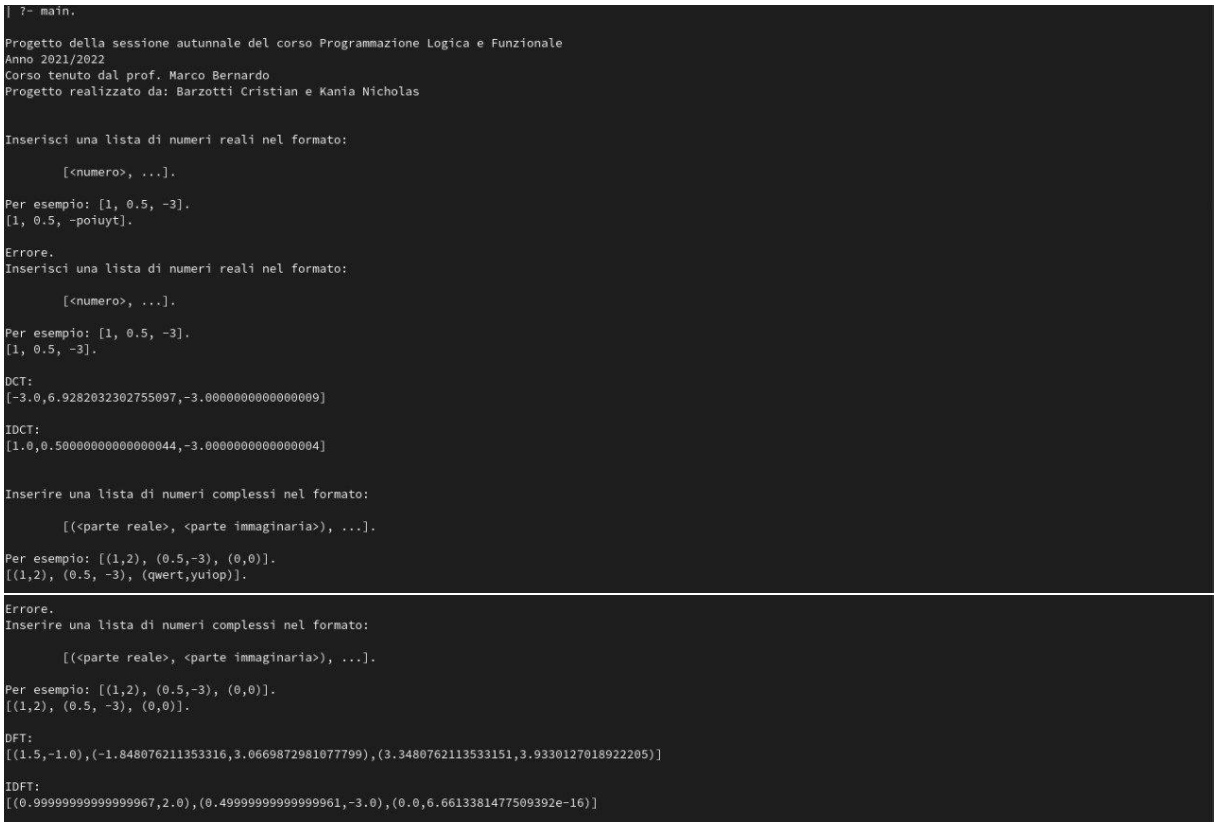

Figura 5.20: Funzionamento dell'intero programma con alcuni input errati