Computer Networks Assignment 4: Implement CDMA with Walsh code

Name: **Sayantan Biswas**

Class: **BCSE 3rd year 1st sem**

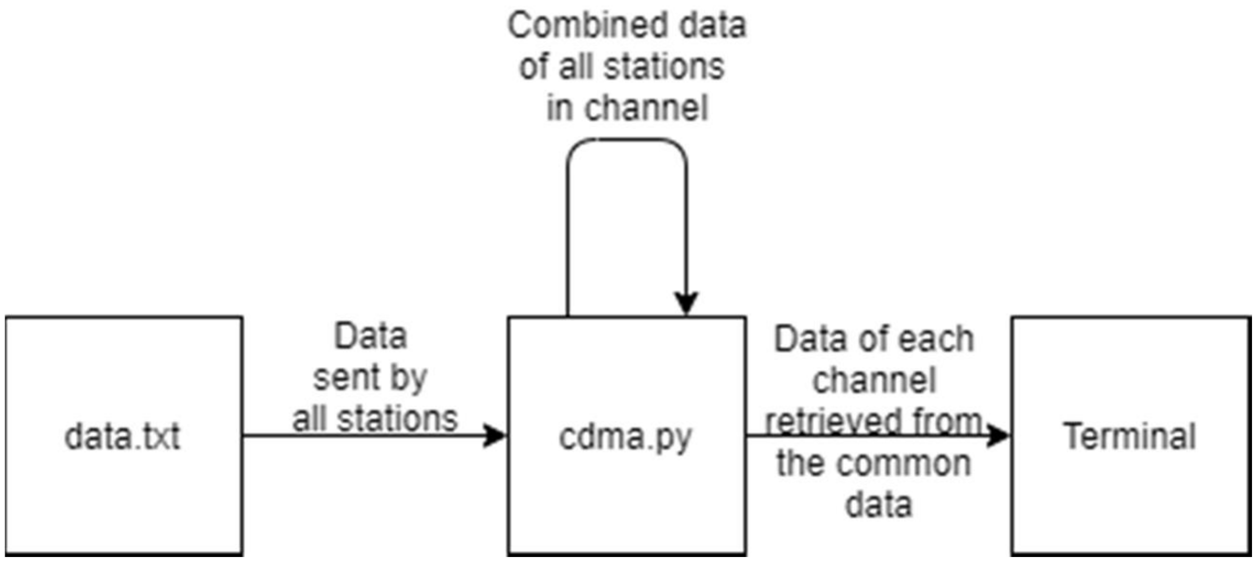
Roll: **001910501057**

Group: **A2**

Problem Statement: In this assignment you have to implement CDMA for multiple access of a common channel by n stations. Each sender uses a unique code word, given by the Walsh set, to encode its data, send it across the channel, and then perfectly reconstruct the data at n stations.

DESIGN: The code has been written in python. This program aims at implementing Code Division Multiple Access, in short CDMA protocol in Computer Networking. This has been achieved by the following program:
cdma.py : This program performs the various functions which read the bit stream from a text file named *data1.txt* and converts all the 0's to (-1)'s and the i's to 0's. Then it creates a Walsh table depending on the number of stations mentioned in the data file. Then it takes one bit from each station at a time and creates a single information for the channel using the Walsh table. Finally, it decomposes the signal in the channel using the same Walsh table and shows whether the data sent by the stations and the data displayed on the terminal are identical or not.

STRUCTURE DIAGRAM:


Combined data of all stations in channel
data.txt
Data sent by all stations
cdma.py
Data of each channel retrieved from the common data
Terminal

Input Format: The input is taken from a text file named *data1.txt* . It contains a stream of 0s, 1s and i's. Here, i means that the station is idle at that instance of time. These characters are treated as bits and are used to create datawords.
Output Format: The message that is sent is printed on the terminal.

IMPLEMENTATION:

In this problem statement, the entire program is implemented in Python 3.
The detailed method description is written below with a suitable code snippets along with comments for better understanding code overview.

In this method, it usually returns the walsh code for the n number of stations. In other words, it is a method which returns the walsh matrix depeding on the number of stations provided by the data file.

```
#Returns the walsh code for n stations
def get_walsh(n):
 return [[int(bin(i&j),13)%2 or -1 for j in range(n)] for i in
range(n)]
```

In this method, it returns the value 0, 1 and -1 which totally depends on the data character as i, 1 and 0 respectively.

```
#Returns 0, 1 or -1, depending on given character is i, 1 or 0 respectively
def get_bin(chr):
    switcher = {
    '0':-1,
    '1':1,
    'i':0
    }
    return switcher.get(chr,2)
```

In this function, it returns the char as 0, 1 and i, depending on the converted data as -1, 1 and 0 respectively.

```

#Returns 0, 1 or i, depending on character is -1, 1 or 0 respectively
def get_char(n):

    switcher = {
    -1:'0',
    1:'1',
    0:'i'
    }
    return switcher.get(n,'na')
```

It reads the input file that is provided by the user,here, data.txt is the input file which contains a streams of 0,1 and i in a matrix form. It helps to find out the number of stations that has been created.

```
#To create the walsh code table
#filea = open("data.txt", 'r') #Test Case 1
filea = open("data1.txt",'r') #Test Case 2
data = (filea.read()).split('\n')
print(data)
la = len(data)
num_stn = la
max_num_stn = 2**ceil(log(num_stn,2))
data_fin = [['i' for i in range(max_num_stn)] for i in range(max_num_stn)]

for i in range(num_stn):
    if len(data[i]) > 0:
        for j in range(len(data[i])):
            data_fin[i][j] = data[i][j]
```

```
num_fin = 0
i = 0
w = np.array(get_walsh(max_num_stn))
print('Walsh code\n'+str(w))
```

Finally, the summary of results. The program reads *data.txt* and gets the number of stations from it and that is mentioned the previous code snippets. Then it calls the *get_walsh(n)* function to get the Walsh matrix depending on the number of stations. Then it calls the *get_bin(chr)* function for each character in the data. Then it uses this converted data and the Walsh code to get a single data and send it to the channel. Then it decomposes the same data into a data for every station. These data are then converted to character using the *get_char(n)* function to check the similarity of the input and output data.

```
#Results
for i in range(max_num_stn):

    chara = [x[i] for x in data_fin]
    print('Data input = '+str(chara))
    data_enc = [get_bin(x) for x in chara]
    print('Data encoded = '+str(data_enc))
    data = np.matmul(w,data_enc)
    print('Data on channel = '+str(data))
    data_dec = np.matmul(w,data)/max_num_stn
    data_rcv = [get_char(int(x)) for x in data_dec]
    print('Data received = '+str(data_rcv)+'\n')
    time.sleep(1)

print('END')
```

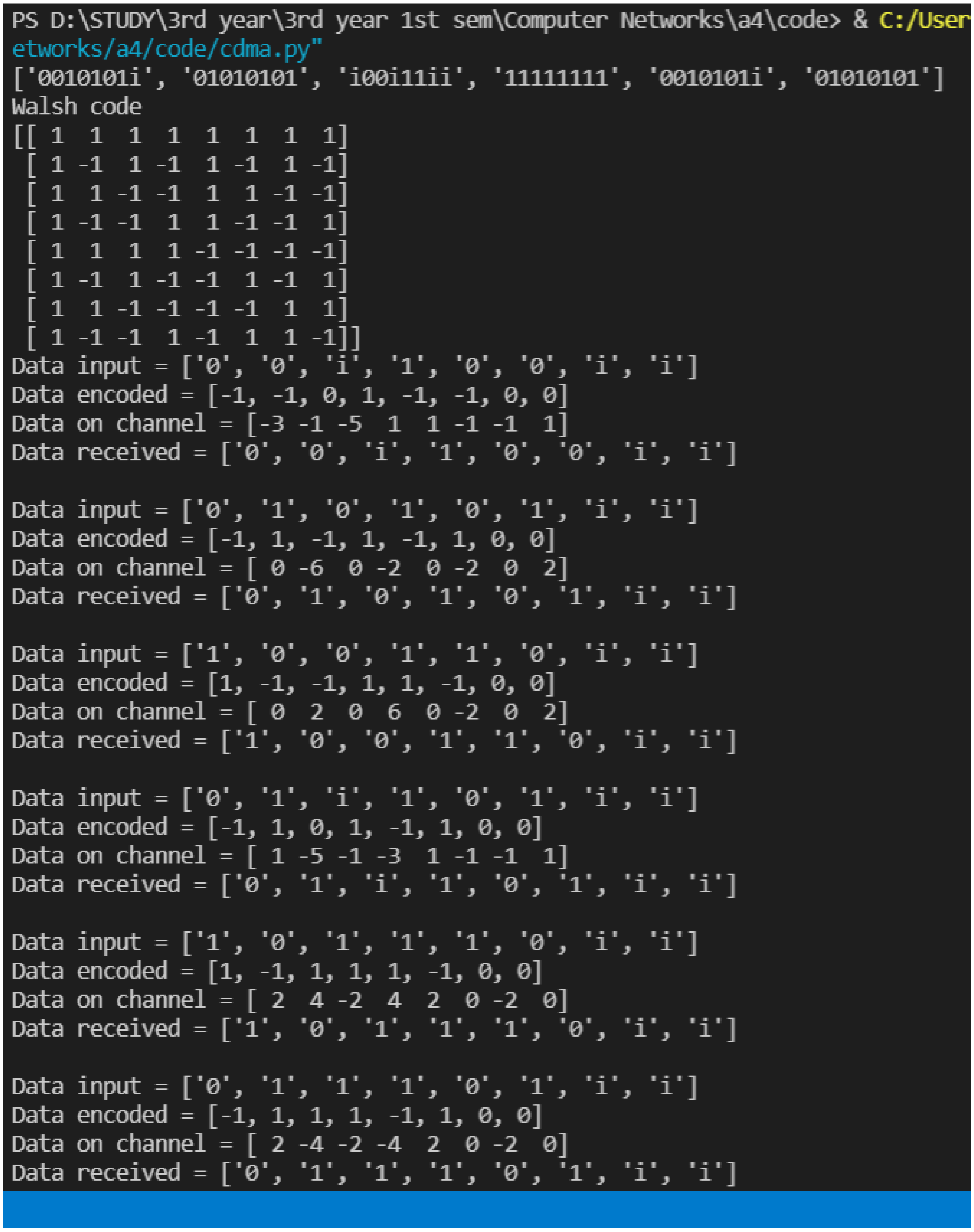
Result:

The CDMA protocol has been demonstrated with a binary-like text file. To show that the Walsh code is generated smoothly even when the number of stations is not the power of 2, the screenshots have been demonstrated with 6 stations. 2 extra imaginary stations have been taken into consideration that are always silent to make the total number a power of 2 for smooth working of the protocol. The protocols have been tested with various number of stations, each of them having different length of messages. The program runs on all such cases.

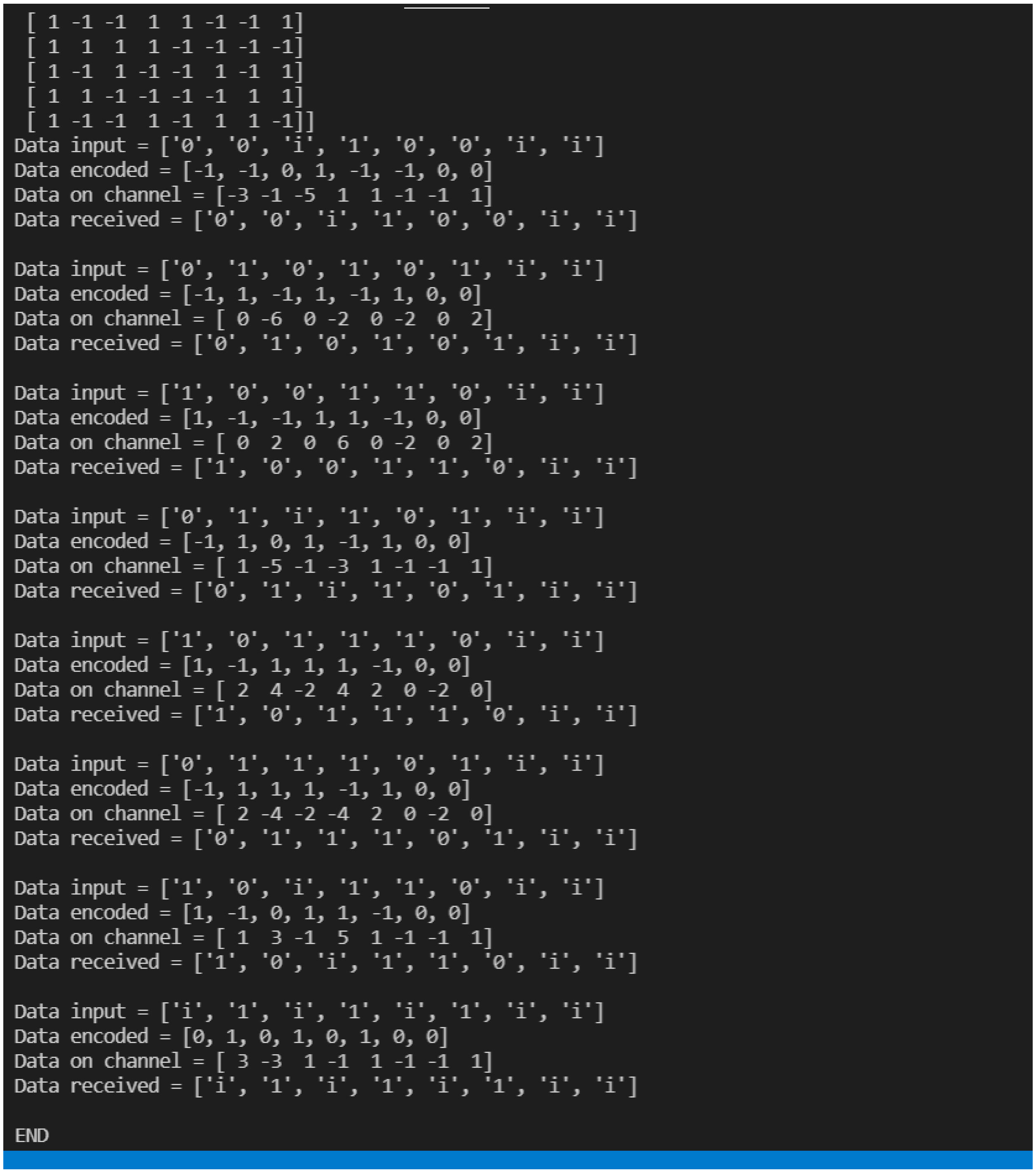
OUTPUT:

To demonstrate the CDMA protocol given in the problem statement, a *data1.txt* file has been loaded with the following stream of 0s, 1s and i's. The frames are read from this stream, are converted to bits, then it accumulate into a single signal. This signal is then decomposed and displayed on the screen finally.

Output screenshots->



```
PS D:\STUDY\3rd year\3rd year 1st sem\Computer Networks\a4\code> & C:/User
etworks/a4/code/cdma.py"
['0010101i', '01010101', 'i00i11ii', '11111111', '0010101i', '01010101']
Walsh code
[[ 1  1  1  1  1  1  1  1]
 [ 1 -1  1 -1  1 -1  1 -1]
 [ 1  1 -1 -1  1  1 -1 -1]
 [ 1 -1 -1  1  1 -1 -1  1]
 [ 1  1  1  1 -1 -1 -1 -1]
 [ 1 -1  1 -1 -1  1 -1  1]
 [ 1  1 -1 -1 -1 -1  1  1]
 [ 1 -1 -1  1 -1  1  1 -1]]
Data input = ['0', '0', 'i', '1', '0', '0', 'i', 'i']
Data encoded = [-1, -1, 0, 1, -1, -1, 0, 0]
Data on channel = [-3 -1 -5  1  1 -1 -1  1]
Data received = ['0', '0', 'i', '1', '0', '0', 'i', 'i']

Data input = ['0', '1', '0', '1', '0', '1', 'i', 'i']
Data encoded = [-1, 1, -1, 1, -1, 1, 0, 0]
Data on channel = [ 0 -6  0 -2  0 -2  0  2]
Data received = ['0', '1', '0', '1', '0', '1', 'i', 'i']

Data input = ['1', '0', '0', '1', '1', '0', 'i', 'i']
Data encoded = [1, -1, -1, 1, 1, -1, 0, 0]
Data on channel = [ 0  2  0  6  0 -2  0  2]
Data received = ['1', '0', '0', '1', '1', '0', 'i', 'i']

Data input = ['0', '1', 'i', '1', '0', '1', 'i', 'i']
Data encoded = [-1, 1, 0, 1, -1, 1, 0, 0]
Data on channel = [ 1 -5 -1 -3  1 -1 -1  1]
Data received = ['0', '1', 'i', '1', '0', '1', 'i', 'i']

Data input = ['1', '0', '1', '1', '1', '0', 'i', 'i']
Data encoded = [1, -1, 1, 1, 1, -1, 0, 0]
Data on channel = [ 2  4 -2  4  2  0 -2  0]
Data received = ['1', '0', '1', '1', '1', '0', 'i', 'i']

Data input = ['0', '1', '1', '1', '0', '1', 'i', 'i']
Data encoded = [-1, 1, 1, 1, -1, 1, 0, 0]
Data on channel = [ 2 -4 -2 -4  2  0 -2  0]
Data received = ['0', '1', '1', '1', '0', '1', 'i', 'i']
```



```
 [ 1 -1 -1  1  1 -1 -1  1]
 [ 1  1  1  1 -1 -1 -1 -1]
 [ 1 -1  1 -1 -1  1 -1  1]
 [ 1  1 -1 -1 -1 -1  1  1]
 [ 1 -1 -1  1 -1  1  1 -1]]
Data input = ['0', '0', 'i', '1', '0', '0', 'i', 'i']
Data encoded = [-1, -1, 0, 1, -1, -1, 0, 0]
Data on channel = [-3 -1 -5  1  1 -1 -1  1]
Data received = ['0', '0', 'i', '1', '0', '0', 'i', 'i']

Data input = ['0', '1', '0', '1', '0', '1', 'i', 'i']
Data encoded = [-1, 1, -1, 1, -1, 1, 0, 0]
Data on channel = [ 0 -6  0 -2  0 -2  0  2]
Data received = ['0', '1', '0', '1', '0', '1', 'i', 'i']

Data input = ['1', '0', '0', '1', '1', '0', 'i', 'i']
Data encoded = [1, -1, -1, 1, 1, -1, 0, 0]
Data on channel = [ 0  2  0  6  0 -2  0  2]
Data received = ['1', '0', '0', '1', '1', '0', 'i', 'i']

Data input = ['0', '1', 'i', '1', '0', '1', 'i', 'i']
Data encoded = [-1, 1, 0, 1, -1, 1, 0, 0]
Data on channel = [ 1 -5 -1 -3  1 -1 -1  1]
Data received = ['0', '1', 'i', '1', '0', '1', 'i', 'i']

Data input = ['1', '0', '1', '1', '1', '0', 'i', 'i']
Data encoded = [1, -1, 1, 1, 1, -1, 0, 0]
Data on channel = [ 2  4 -2  4  2  0 -2  0]
Data received = ['1', '0', '1', '1', '1', '0', 'i', 'i']

Data input = ['0', '1', '1', '1', '0', '1', 'i', 'i']
Data encoded = [-1, 1, 1, 1, -1, 1, 0, 0]
Data on channel = [ 2 -4 -2 -4  2  0 -2  0]
Data received = ['0', '1', '1', '1', '0', '1', 'i', 'i']

Data input = ['1', '0', 'i', '1', '1', '0', 'i', 'i']
Data encoded = [1, -1, 0, 1, 1, -1, 0, 0]
Data on channel = [ 1  3 -1  5  1 -1 -1  1]
Data received = ['1', '0', 'i', '1', '1', '0', 'i', 'i']

Data input = ['i', '1', 'i', '1', 'i', '1', 'i', 'i']
Data encoded = [0, 1, 0, 1, 0, 1, 0, 0]
Data on channel = [ 3 -3  1 -1  1 -1 -1  1]
Data received = ['i', '1', 'i', '1', 'i', '1', 'i', 'i']

END
```
