NATIONAL INSTITUTE OF TECHNOLOGY SRINAGAR

ITT305: PROGRAMMING ASSIGNMENT 1

DEPARTMENT OF INFORMATION TECHNOLOGY

5th SEMESTER - 2022

Submitted To
[Dr. Iqra Altaf Gilani]

Date & Day
NOV. 17, 2022 / THU

[SNEHANKIT BHAGAT] [2020BITE070]

[HARSHIT DUBEY] [2020BITE048]

[ANKIT KUMAR] [2020BITE058]

LINE ENCODING

Objective:

Implement Line coding encoder and scrambler with digital data generator

- Digital data generator: generates completely random data sequence and
  a random sequence with some fixed sub-sequences like eight or four consecutive zeros. It should also return the longest palindromic sequence in
  the generated data.
- Line coding schemes to be implemented: NRZ-L, NRZ-I, Manchester, Differential Manchester, AMI.
- Scrambling schemes: B8ZS, HDB3.

Language used:

Python and its libraries

Import libraries :

```
import numpy as np
import matplotlib.pyplot as plt
from random import *
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
```

Matplotlib.pyplot is used for graphical plotting

User Input Type:

```
print("\n----------------------Line
Encoder----------------------")
print("-------------------------------------By Bhagat
Snehankit.")

print("\nEnter \"1\" for User Input Type")
print("Enter \"2\" for Random Binary Input Type")
x=int(input("Type : "))
```

- **Type 1**

```
if(x==1):
    s=input("\nEnter The Binary Signal : ")
```

- **Type 2**

Using Function...

```
randBinList = lambda n: [randint(0,1) for b in
range(1,n+1)]
```

To generate random signals...

```
else:
    bla = int(input("\nEnter \"1\" for Fixed
Subsequence(size must be greater than 8) Otherwise Enter
\"2\" : "))
    size = int(input("\nEnter Size of Binary Signal : "))
    kla = randint(0,9)
    if(bla==1):
        size = size - 8
```

```
    kla = kla%size
    #print(kla)
    s=[]
    s=randBinList(size)
    ankit = "00000000"
    #print(*s)
    s="".join(str(i) for i in s)
    if(bla==1):
        s=s[:kla]+ankit+s[kla:]
    print("Input is: ")
    print(s)
```

Coding Schemes Types:

```
print("\nCoding Schemes Are...")
print("\n1. NRZ-L \n2. NRZ-I \n3. Manchester \n4. Diff.
Manchester \n5. AMI\n")
n=int(input("Type : "))
```

- **Perform NRZ-L :**

  Define function for input "1" i.e. NRZ-L ...

```
if(n==1):
    # Perform NRZ-L ...
```

Creating a list for positive and negative (1 or -1) form ...

```
    ls=list()
    for i in range(len(s)):
        if(s[i]=='0' or s[i]==0):
            ls.append(-1)
        else:
```

```
            ls.append(1)
```

Define the “x” and “y” position, which is repeated two times to build the block of signal …

```
    xs = np.repeat(range(len(s)), 2)
    ys = np.repeat(ls, 2)
    xs=xs[1:]
    xs=np.append(xs,(xs[len(xs)-1]+1))
    ys=ys[:-1]
    ys=np.append(ys,(ys[len(ys)-1]))
```

Define parameters of plot …

```
    plt.step(xs,ys)
    plt.ylim(-2, 2)
    plt.title('The Binary Signal : {}\n'.format(s),
size=16)
    plt.show()
```

- **Perform NRZ-I :**

```
elif(n==2):
    # NRZ-I ...
    Is=list()
    if(s[0]=='0' or s[0]==0):
        Is.append(-1)
    else:
        Is.append(1)
    k=len(s)
    i=1
    while(i<k):
        if(int(s[i])==0):
            Is.append(Is[i-1])
        else:
```

```
            Is.append(-Is[i-1])
        i=i+1
    xs = np.repeat(range(len(s)), 2)
    ys = np.repeat(Is, 2)
    xs=xs[1:]
    xs=np.append(xs,(xs[len(xs)-1]+1))
    ys=ys[:-1]
    ys=np.append(ys,(ys[len(ys)-1]))

    plt.step(xs,ys)
    plt.ylim(-2, 2)
    plt.title('The Binary Signal : {}\n'.format(s),
size=16)
    plt.show()
```

- **Perform Manchester :**

```
elif(n==3):
    # Manchester ...
    pm=list()
    for j in range(len(s)):
        if(s[j]=='0' or s[j]==0):
            pm.append(-1)
            pm.append(1)
        else:
            pm.append(1)
            pm.append(-1)
    xs=[x*0.5 for x in range(0,(2*len(s)))]
    xs=np.repeat(xs,2)
    ys = np.repeat(pm, 2)
    xs=xs[1:]
    xs=np.append(xs,(xs[len(xs)-1]+0.5))
    ys=ys[:-1]
    ys=np.append(ys,(ys[len(ys)-1]))
```

```
    plt.step(xs,ys)
    plt.ylim(-2, 2)
    plt.title('The Binary Signal : {}\n'.format(s),
size=16)
    plt.show()
```

- **Perform Differential Manchester :**

```
elif(n==4):
    # Differential Manchester ...
    pdm=list()
    pdm.append(1)
    pdm.append(-1)
    i=1
    k=len(s)
    while(i<k):
        if(int(s[i])==1):
            pdm.append(pdm[len(pdm)-1])
            pdm.append(-pdm[len(pdm)-1])
        else:
            pdm.append(-pdm[len(pdm)-1])
            pdm.append(-pdm[len(pdm)-1])
        i=i+1
    print(pdm)
    xs=[x*0.5 for x in range(0,(2*len(s)))]
    xs=np.repeat(xs,2)
    ys = np.repeat(pdm, 2)
    xs=xs[1:]
    xs=np.append(xs,(xs[len(xs)-1]+0.5))
    ys=ys[:-1]
    ys=np.append(ys,(ys[len(ys)-1]))

    plt.step(xs,ys)
```

```
    plt.ylim(-2, 2)
    plt.title('The Binary Signal : {}\n'.format(s), 
size=16)
    plt.show()
```

- **Perform AMI :**

  Ask users for Scrambling ...

```
else:
    # Perform AMI ...
    q=int(input("\nPress \"0\" for Scrambling or \"1\" for 
Not Scrambling : "))
```

  - **With Scrambling :**

```
if(q==0):
        am=list()
        m=1
        for i in range(len(s)):
            if(int(s[i])==0):
                am.append(0)
            else:
                if(m%2==1):
                    am.append(1)
                else:
                    am.append(-1)
                m=m+1
        xs = np.repeat(range(len(s)), 2)
        ys = np.repeat(am, 2)
        xs=xs[1:]
        xs=np.append(xs,(xs[len(xs)-1]+1))
        ys=ys[:-1]
        ys=np.append(ys,(ys[len(ys)-1]))
```

```
        plt.step(xs,ys)
        plt.ylim(-2, 2)
        plt.title('The Binary Signal :
{}\n'.format(s), size=16)
        plt.show()
```

- **Without Scrambling :**

```
else:
        # Scrambling ...
        p=int(input("\nEnter \"1\" for B8ZS ... Enter
\"2\" for HDB3..."))
        q=len(s)
```

- **Performing B8ZS :**

```
if(p==1):
            # B8ZS...
            bz=list()
            m=1
            s1=s.replace("00000000","000vb0vb")
            for i in range(len(s1)):
                if(s1[i]=='0' or s1[i]==0):
                    bz.append(0)
                elif(s1[i]=='1'):
                    if(m%2==1):
                        bz.append(1)
                    else:
                        bz.append(-1)
                    m=m+1
                elif(s1[i]=='v'):
                    if(m%2==1):
                        bz.append(-1)
                    else:
```

```
                    bz.append(1)
                else:
                    if(m%2==1):
                        bz.append(1)
                    else:
                        bz.append(-1)
                    m=m+1
            xs = np.repeat(range(len(s)), 2)
            ys = np.repeat(bz, 2)
            xs=xs[1:]
            xs=np.append(xs,(xs[len(xs)-1]+1))
            ys=ys[:-1]
            ys=np.append(ys,(ys[len(ys)-1]))

            plt.step(xs,ys)
            plt.ylim(-2, 2)
            plt.title('The Binary Signal : {}\n'.format(s),
size=16)
            plt.show()
```

- **Performing HDB3 :**

```
else:
            # HDB3 ...
            m=0
            hd=list()

            f=s.find("0000")
            if(f==-1):
                f=len(s)
            i=0
            k=len(s)
            d=1
            p=0
```

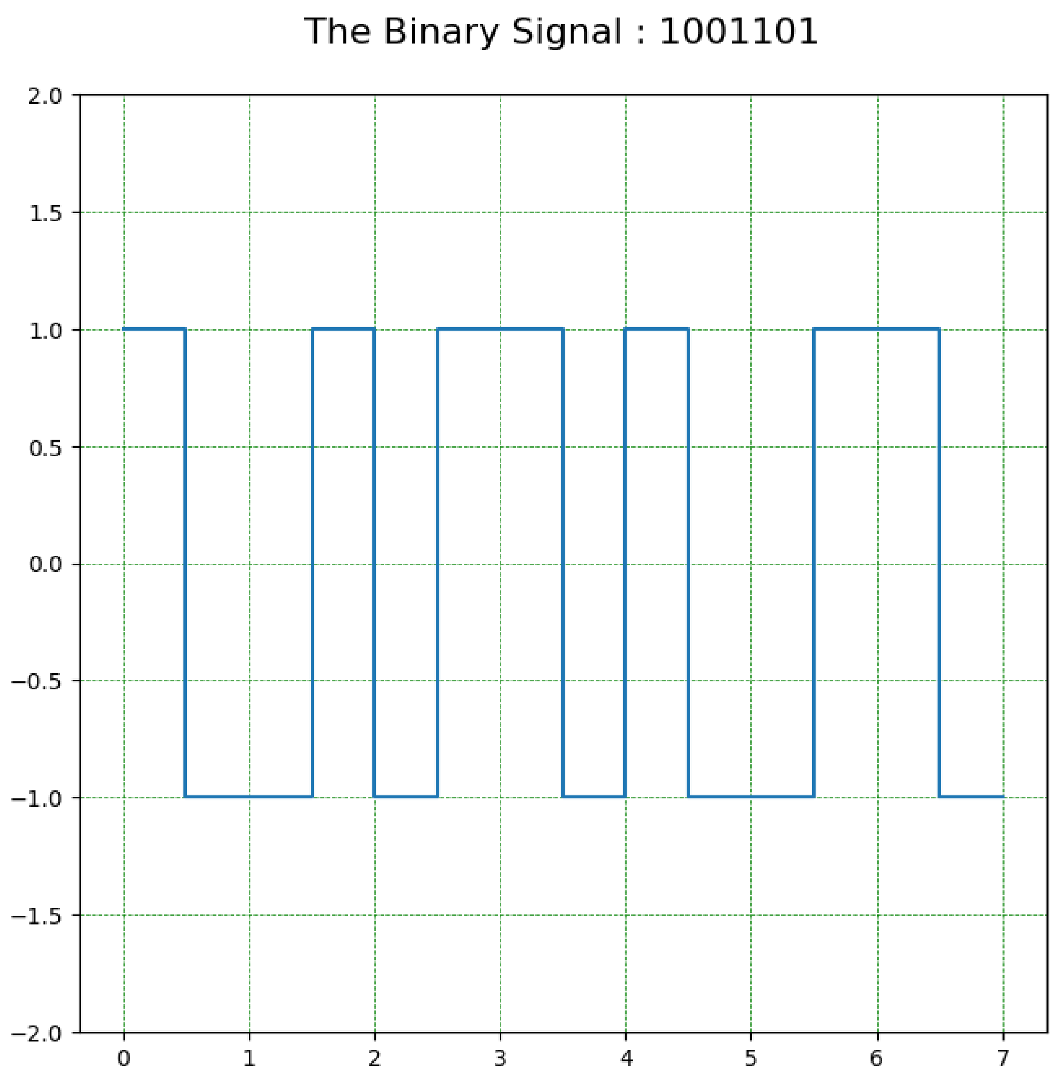
```
            while(i<k):
                if(s[i]=='1' or s[i]==1):
                    m=m+1
                    p=p+1
                    if(m%2==1):
                        hd.append(d)
                        d=1
                    else:
                        hd.append(-d)
                        d=-d
                else:
                    if(i<f):
                        hd.append(0)
                    elif(i==f):
                        i=i+3
                        if(p%2==0):
                            hd.append(-d)
                            hd.append(0)
                            hd.append(0)
                            hd.append(-d)
                            d=-d
                            p=p+2
                            m=m+1
                        else:
                            hd.append(0)
                            hd.append(0)
                            hd.append(0)
                            hd.append(d)
                            p=p+1
                        jk=s[i+1:(i+1)+(k-i-1)]
                        x=jk.find("0000")
                        if(x==-1):
                            f=k
```

```
                    else:
                        f=i+1+x
            i=i+1
        xs = np.repeat(range(len(s)), 2)
        ys = np.repeat(hd, 2)
        xs=xs[1:]
        xs=np.append(xs,(xs[len(xs)-1]+1))
        ys=ys[:-1]
        ys=np.append(ys,(ys[len(ys)-1]))
        plt.step(xs,ys)
        plt.ylim(-1.5,1.5)
        plt.title('The Binary Signal :
{}\nHDB3...'.format(s), size=16)
        plt.show()
        print(hd)
```

Input Sample(User Input) **:**

```
-----------------------Line Encoder-----------------------
-------------------------------------By Bhagat Snehankit.

Enter "1" for User Input Type
Enter "2" for Random Binary Input Type
Type : 1

Enter The Binary Signal : 1001101

Coding Schemes Are...

1. NRZ-L
2. NRZ-I
3. Manchester
4. Diff. Manchester
5. AMI

Type : 3
```

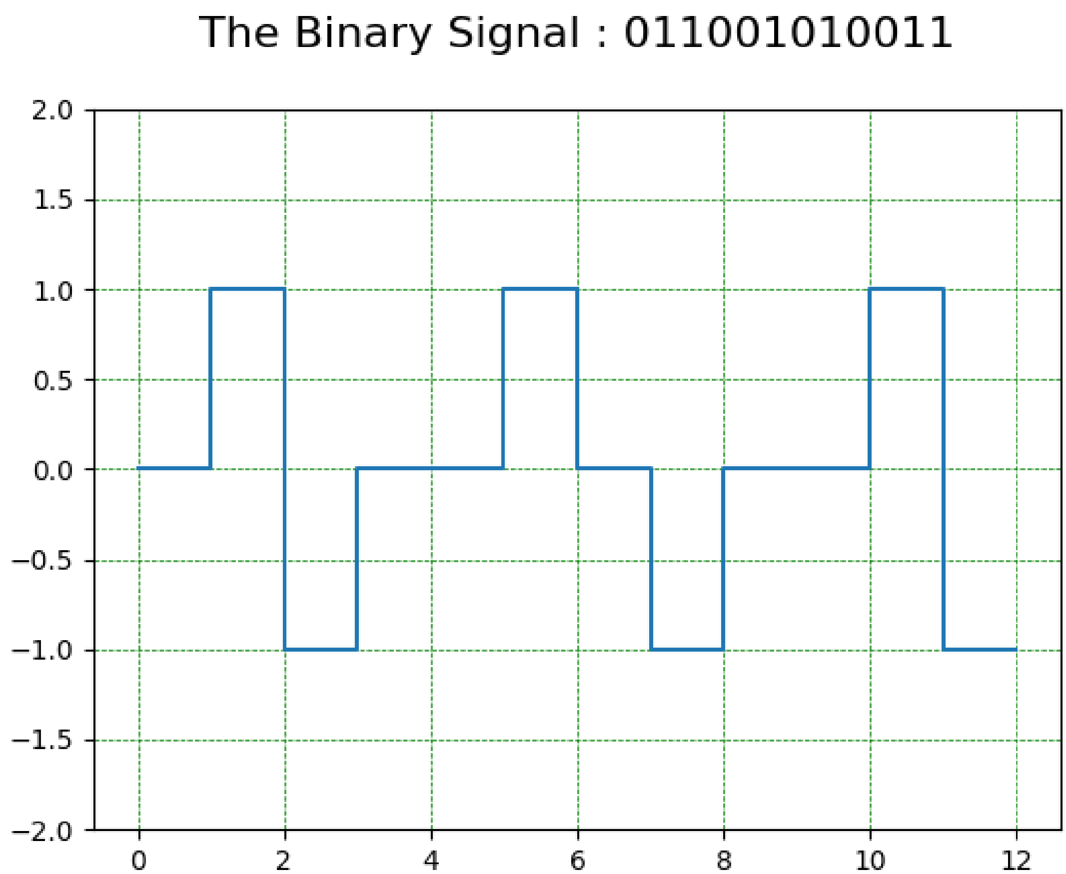
Output :



Input Sample (Random Input) **:**

```
----------------------Line Encoder----------------------
-------------------------------------By Bhagat Snehankit.

Enter "1" for User Input Type
Enter "2" for Random Binary Input Type
Type : 2

Enter "1" for Fixed Subsequence(size must be greater than 8) Otherwise Enter "2" : 2

Enter Size of Binary Signal : 12
Input is:
011001010011

Coding Schemes Are...

1. NRZ-L
2. NRZ-I
3. Manchester
4. Diff. Manchester
5. AMI

Type : 5

Press "0" for Scrambling or "1" for Not Scrambling : 1

Enter "1" for B8ZS ... Enter "2" for HDB3...1
```

Output :


The Binary Signal : 011001010011
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2
4
6
8
10
12