Programming Assignment 1

Author: Gabriel Hofer

CSC-325

Instructor: Dr. Karlsson

Due: September 25, 2020

Computer Science and Engineering
South Dakota School of Mines and Technology

We randomly create vector p containing either 4 or 11 data bits by calling makeMessage.

$$p = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Listing 1: makeMessage

```
def makeMessage(n):
  return np.random.randint(2, size=(n, 1))
```

Then we encode p by pre-multiplying p by G modulo 2 in the encode function.

$$x = Gp = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Listing 2: encode

```
def encode(G,p):
  return np.matmul(G,p) & 1
```

We sometimes create an error (flip a bit) in p using the makeError function.

Listing 3: makeError

```
def makeError(p):
  if random.randint(0,1): return p
  rdm=random.randint(0,p.shape[0]-1)
  p[rdm,0]=p[rdm,0]^1;
  return p
```

We check to see where errors occured by pre-multiplying x by the parity-check matrix, H. This is done in parityCheck function.

$$z = Hr = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Listing 4: parityCheck

```
def parityCheck(H,r):
    return np.matmul(H,r) & 1
```

Correct the error by flipping the bit that was incorrect accoring to the syndrome vector z.

Listing 5: correctError

```
def correctError(z,r):
    loc=0
    for i in range(0,z.shape[0]):
        loc+=z[i,0]*pow(2,i)
    if loc==0: return r
    r[loc-1,0]=r[loc-1,0]^1;
    return r
```

Finally, we decode the message by pre-multiply the encoded message by a decoding matrix R.

$$p_r = Rr = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Listing 6: decodeMessage

```
def decodeMessage(R, r):
  return np.matmul(R, r)
```

Below is our main function. We hardcoded G, H, and R according to which mode was entered by the user. Then these three matrices are passed to functions encode, parityCheck, and decode, respectively. I created the (15,11) matrices by looking at the bit patterns in the table in the writeup.

Listing 7: main

```
def main():
  # enter mode: either (7,4) or (15,11)
  mode=input("Enter mode: ")
  if mode=="H1511":
    pLen=11
    G=np.array([
        [1,1,0,1,1,0,1,0,1,0,1], \
        [1,0,1,1,0,1,1,0,0,1,1], \
        [1,0,0,0,0,0,0,0,0,0,0], \
        [0,1,1,1,0,0,0,1,1,1,1], \
        [0,1,0,0,0,0,0,0,0,0,0], \
        [0,0,1,0,0,0,0,0,0,0,0], \
        [0,0,0,1,0,0,0,0,0,0,0], \
        [0,0,0,0,1,1,1,1,1,1,1], \
        [0,0,0,0,1,0,0,0,0,0,0], \
        [0,0,0,0,0,1,0,0,0,0,0], \
        [0,0,0,0,0,0,1,0,0,0,0], \
        [0,0,0,0,0,0,0,1,0,0,0], \
        [0,0,0,0,0,0,0,0,1,0,0], \
        [0,0,0,0,0,0,0,0,0,1,0], \
        [0,0,0,0,0,0,0,0,0,0,1]])
    H = np.array([ \
        [1,0,1,0,1,0,1,0,1,0,1,0,1,0,1], \
        [0,1,1,0,0,1,1,0,0,1,1,0,0,1,1], \
        [0,0,0,1,1,1,1,0,0,0,0,1,1,1,1], \
        [0,0,0,0,0,0,0,1,1,1,1,1,1,1,1], \
        ])
    R = np.array([ \
```

```
        [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0], \
        [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0], \
        [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0], \
        [0,0,0,0,0,0,1,0,0,0,0,0,0,0,0], \
        [0,0,0,0,0,0,0,0,1,0,0,0,0,0,0], \
        [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0], \
        [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0], \
        [0,0,0,0,0,0,0,0,0,0,0,1,0,0,0], \
        [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0], \
        [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0], \
        [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]])
else :
    pLen=4
    G=np.array([ \
        [1,1,0,1], \
        [1,0,1,1], \
        [1,0,0,0], \
        [0,1,1,1], \
        [0,1,0,0], \
        [0,0,1,0], \
        [0,0,0,1]])
    H = np.array([ \
        [1,0,1,0,1,0,1], \
        [0,1,1,0,0,1,1], \
        [0,0,0,1,1,1,1]])
    R = np.array([ \
        [0,0,1,0,0,0,0], \
        [0,0,0,0,1,0,0], \
        [0,0,0,0,0,1,0], \
        [0,0,0,0,0,0,1]])

# generate random message vector, p, of length 4 or 11
p=makeMessage(pLen);
print("Message           : "+str(p.transpose()[0]))

# encode (make send vector)
x=encode(G,p)
print("Send Vector       : "+str(x.transpose()[0]))
```

```
# modify the vector to simulate an error or not
r=makeError(x)
print("Recieved Message : "+str(r.transpose()[0]))

# Parity Check
z=parityCheck(H,r)
print("Parity Check      : "+str(z.transpose()[0]));

# Error Correction
corrected=correctError(z,r)
print("Corrected Message: "+str(corrected.transpose()[0]))

# Decode Message
pr=decodeMessage(R,x)
print("Decoded Message  : "+str(pr.transpose()[0]));
```

Usage:

Hamming.py can be run in Ubuntu using the python3 command in the terminal.

Listing 8: main

```
$ python3 Hamming.py
```