Title:

Implementation of Hamming Code for error detection

Sravani Tangeda (CSE-C)
AP20110010174
sravani_tangeda@srmap.edu.in

Objective:

Whenever a message is transmitted, it is possible that the data may be corrupted due to interference. This could be because of the bit flips, where a binary 1 becomes a 0 or vice versa. Error-correcting codes detect the error in the sent message. This can be done by adding parity or redundant bits. Hamming code is one of these error correction techniques that can detect single-bit errors or burst errors and correct single-bit errors. In this document, we implemented this method to detect and correct the error in the received message.

Problem Statement:

The problem is to detect the error in the generated code word and correct it using Hamming code. For this method, the programmer is provided with the message which is in a stream of bytes. Extra bits known as parity bits are inserted in the message to enable error detection. When the transmitted message is sent to the receiver, it should find and print whether the message code is with error or without error. If there is an error, then the error position should be indicated and finally show the corrected code word.

Algorithm:

- Input the message of the 4-bit code word.
- The r redundant bits are placed at the power of 2(1,2,4,..) positions, and the general equation for parity bit position is calculated as 2r>=n+m+1.
- All other positions are data bits.
- Parity bits are calculated as:
  C1=parity(1,3,5,7,9,…)
  C2=parity(2,3,6,7,10,11..)
  C3=parity(4-7,12-15,20-23…)
- If we check for even parity, set parity bit to 1 if the total no of 1's is odd and set to 0 if the total no of 1's is even.
- After data is received, the receiver calculates parity bits again.
- If the value is 0, there is no error in the transmitted message.
- If the value is not 0, there is an error in the transmitted position, and the position at which the error occurred is displayed.
- The data is corrected by flipping the bits and finally corrected code is displayed.

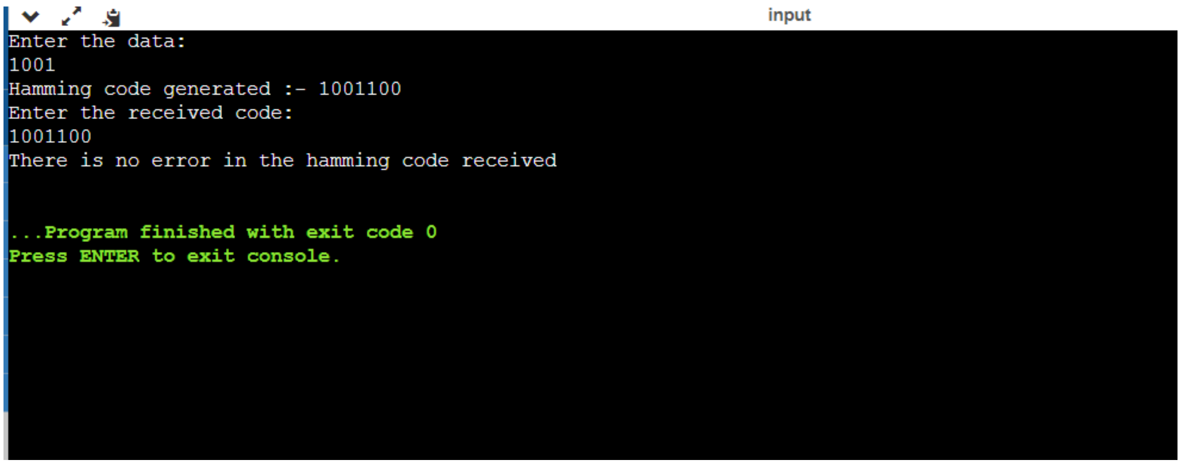
Code:

```
print('Enter the data:')
d=input()         # Input the message
data=list(d)
data.reverse()
c,ch,j,r,h=0,0,0,0,[]
# parity bit calculation and generating hamming code
while ((len(d)+r+1)>(pow(2,r))):
    r=r+1
for i in range(0,(r+len(data))):
    p=(2**c)
    if(p==(i+1)):
        h.append(0)
        c=c+1
    else:
        h.append(int(data[j]))
        j=j+1

for parity in range(0,(len(h))):
    ph=(2**ch)
    if(ph==(parity+1)):
        startIndex=ph-1
        i=startIndex
        toXor=[]
        while(i<len(h)):
            block=h[i:i+ph]
            toXor.extend(block)
            i+=2*ph
        for z in range(1,len(toXor)):
            h[startIndex]=h[startIndex]^toXor[z]
        ch+=1
h.reverse()
print('Hamming code generated :- ', end="") # Hamming code is generated
print(int(''.join(map(str, h))))
print('Enter the received code: ') # Input the message received
d=input()
data=list(d)
data.reverse()
c,ch,j,r,error,h,parity_list,h_copy=0,0,0,0,0,[],[],[]
for k in range(0,len(data)):
    p=(2**c)
    h.append(int(data[k]))
    h_copy.append(data[k])
    if(p==(k+1)):
        c=c+1

for parity in range(0,(len(h))):
    ph=(2**ch)
    if(ph==(parity+1)):
        startIndex=ph-1
        i=startIndex
        toXor=[]
        while(i<len(h)):
            block=h[i:i+ph]
            toXor.extend(block)
            i+=2*ph
        for z in range(1,len(toXor)):
            h[startIndex]=h[startIndex]^toXor[z]
        parity_list.append(h[parity])
        ch+=1
```

```
parity_list.reverse()
error=sum(int(parity_list) * (2 ** i) for i, parity_list in enumerate(parity_list[::-1]))
if((error)==0):
    print('There is no error in the hamming code received')  # As value is 0 there is no error
else:
    print('Error is in',error,'bit from right')    # Error position is printed
    if(h_copy[error-1]=='0'):
        h_copy[error-1]='1'
    elif(h_copy[error-1]=='1'):
        h_copy[error-1]='0'
        print('After correction hamming code is:- ')
    h_copy.reverse()
    print(int(''.join(map(str, h_copy))))        # Corrected code is displayed
```

Output:

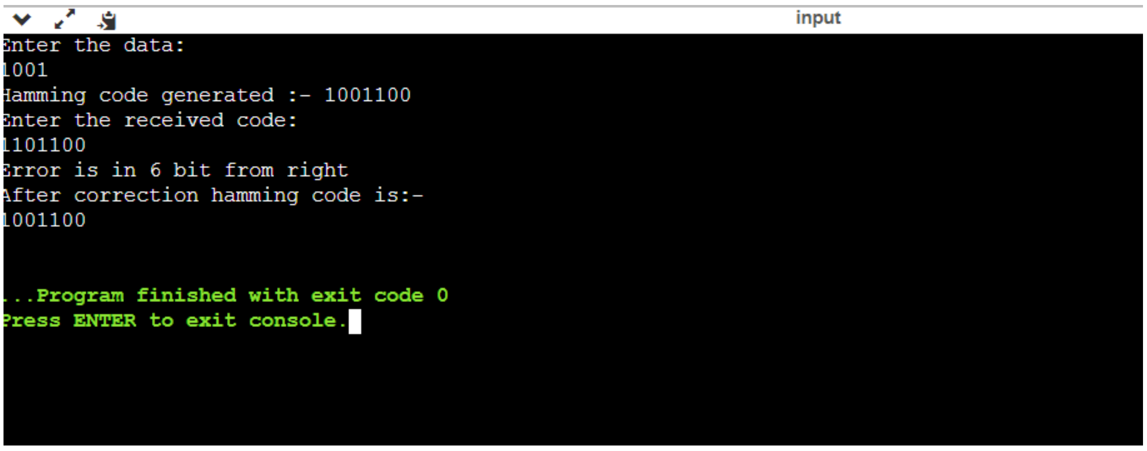
CASE 1:



```
Enter the data:
1001
Hamming code generated :- 1001100
Enter the received code:
1001100
There is no error in the hamming code received


...Program finished with exit code 0
Press ENTER to exit console.
```

Here the message is 1001 and the code generated after performing Hamming code technique is 1001100. Here the received code is same as the original data. So , no error is detected.

CASE 2:



```
Enter the data:
1001
Hamming code generated :- 1001100
Enter the received code:
1101100
Error is in 6 bit from right
After correction hamming code is:-
1001100


...Program finished with exit code 0
Press ENTER to exit console.
```

Here the data is 1001 and the hamming code generated is 1001100. The received message is flipped by a single bit and the data now is 1101100. So the error is shown in the 6th bit from the right. The error is also corrected and the corrected code is displayed as 1001100.

Problems Faced:

Initially, I found it difficult to understand the process to generate the hamming code. Detecting the error position and correction challenged me. But I finally tried to solve the problems.

Conclusion:

With the help of this problem, I understood the concept of Hamming code method for error detection and correction and I learned how to implement it in the Python language.