

7/8/25

PRACTICAL - 6

AIM:

Write a Program to implement error detection and correction using Hamming Code concept. Make a test run to input data stream and verify error correction feature.

Error Correction at Data link layer:

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is transmitted from the sender to the receiver. It is a technique developed by R.W Hamming for error correction.

Create sender problem with the below feature:

- 1) Input to sender file should be a text of any length. Program should convert the text to binary.
- 2) Apply Hamming code concept on the binary data length. Program should convert the text to binary.
- 3) save this output in a file

Create a receiver program with below features :

- 1) Receiver program should read the input from channel file.
- 2) apply hamming code on the binary data to check error.
- 3) If there is an error
- 4) Else remove the redundant bits and convert the binary data to ascii and display the output.

Student Observation

Code:

```
def calc_parity(data):
```

```
C = [0, 0, 0, data[0], 0, data[1],
```

```
data[2], data[3]]
```

```
C[1] = C[3] ^ C[5] ^ C[7]
```

```
C[2] = C[3] ^ C[6] ^ C[7]
```

```
C[4] = C[5] ^ C[6] ^ C[7]
```

```
return C[1], C[2], C[4]
```

def detect_correct(r):

$r[0] + r[1]$ (start of ref (i) diff = 0)

$s_1, s_2, s_4 = r[1] + r[3] \wedge r[5] \wedge r[7],$
 $(r[2] \wedge r[3] \wedge r[6]) \wedge r[7],$

((0110) place $r[4] \wedge r[5] \wedge r[6] \wedge r[7]$)

def detect_correct(r):

(($r = [0] + r[1..3]$) diff = ?)

$s_1, s_2, s_4 = r[1] + r[3] \wedge r[5] \wedge r[2]$

err_pos = $s_4 * 1 + s_2 * r + s_1$

if err_pos:

print("Error at bit " + str(err_pos))

else: print("No error detected")

print("Corrected = ", r[1:])

else:

print("No error detected")

return r[1:]

def extract_data(c): return [c[2], c[4], c[5],
c[6]]

if __name__ == "__main__":

data = input("Enter 4-bit data: ")

if len(data) != 4 or any(b not in "01" for

b in data):

print("Error: Invalid data")

exist ("invalid input")
data = [int(b) for b in data]
enc = calc_Parity(data)
print ("Encoded:", enc)
if input ("Introduced error (y/n):")

lower() == 'y':

P = int(input ("Enter position 1-7:"))

if 1 <= P <= 7:

enc[P-1] ^= 1

print ("Received:", enc)

corr = detect_correct(enc)

print ("Original - data:", extra_data

between") print (corr))

Output: Enter one bit

Enter 4-bit data: 1011

Encoded: [0, 1, 1, 0, 0, 1, 1]

Introduced error: (y/n): y

Enter position 1-7: 3

Received: [0, 1, 0, 0, 0, 1, 1]

Error at bit 3: 1 (calculated)

Corrected: [0, 1, 0, 1, 0, 1, 1]

original data: [1, 0, 1, 1, 1]

a given DNA sequence has subsp
 seq in other sequence has. At least
 . else
 part in target
 matches first character of each DNA
 seq. so have seen. because a at which
 character it must be target. pinkish
 pinkish. easier work of read
 DNA sequence by hand without. also
 introduced target a scrub several
 more reads. read of 1000
 bases. which is required to find
 a target. and is called
 part of file at last

Result: the no. of reads has not increased
 since the code for Hamming problem
 was successfully executed
 26/9/2020