22/8/24

Experiment -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 data is transmitted from sender to the receiver. It is a technique developed by R.W Hamming for error correction.

Create Sender program with below features.

1. Input to sender file should be a text of any lenght. Program should convert the text to binary
2. Apply hamming code concept on the binary data and add redundant bits to it
3. Save this output in a file called channel.

Create a receiver program with below features.

1. Receive program should read the input from channel file.
2. Apply hamming code on the binary data to check errors.
3. If there is an error, display position of the error.
4. Else remove the redundant bits and convert the binary data to ascii and display the output.

Student Observation:

```
def char_to_binary(ch)
    return [int(bit) for bit in format(ord(ch), '08b')]
def calculate_parity_bits(hamming_code, r):
    n = len(hamming_code) - 1
    for i in range(r):
    Parity_pos = 2**i
    Parity = 0
    for j in range(Parity_pos, n+1, 2*parity_pos):
    for k in range(j, min(j+parity_pos, n+1)):
    Parity ^= hamming_code[k]
    hamming_code[parity_pos] = parity
def generate_hamming_code(data_bits):
    m = len(data_bits)
    r = 0
    n = m
    while n+r+1 > 2**r:
        r += 1
        n = m+r
        hamming_code[i] = 0
        j += 1
else:
    hamming_code[i] = data_bits[i-j-1]
calculate_parity_bits(hamming_code, r)
    returning hamming_code
def detect_and_correct_error(hamming_code):
    n = len(hamming_code) - 1
```

```
    r = int(math.log2(n+1))
    error_pos = 0
    for i in range(r):
        parity_pos = 2**i
        parity = 0
    for j in range(parity_pos, n+1, 2*parity_pos):
        for k in range(j, min(j+parity_pos, n+1)):
            parity ^= hamming_code[k]
        if parity != 0:
            error_pos += parity_pos
    return error_pos
def binary_to_chars(binary):
    chars = []
    for i in range(0, len(binary), 8):
        byte = binary[i:i+8]
        chars.append(chr(int(''.join(map(str, byte)), 2)))
    return ''.join(chars)
def main():
    input_string = input("Enter the input string: ")
    binary = []
    for ch in input_string:
        binary.extend(char_to_binary(ch))
    hamming_code = generate_hamming_code(binary)
    print("Generated Hamming code:",
          hamming_code[1:])
    error_pos = int(input("Enter the position to
    simulate error (0 for no error): "))
```

```
if 1 <= error_pos <= len(hamming_code):
    hamming_code[error_pos] ^= 1
    print("Hamming code with error:", hamming_code)
    if detected_error_pos == 0:
        print("No error detected.")
    else:
        print(f"Error detected at position: {detected_error_pos}")
        hamming_code[detected_error_pos] ^= 1
        print("Corrected Hamming code:", hamming_code)
        print(f"Corrected bit at position {detected_error_pos}. {hamming_code[detected_error_pos]}")
corrected_data_bits = [hamming_code[i] for i in range(1, len(hamming_code)) if not (i & (i-1))]
corrected_string = binary_to_char(corrected_data_bits)
print("Corrected string:", corrected_string)

if __name__ == "__main__":
    main()
```

Result:

Thus an program to implement error detection an correction using hamming code concept has been successfully executed.

18/9/24