Assignment 4

CN

AIDS-A

Roll 10

**Task:** Write a program for error detection and correction for 7/8 bits ASCII codes using CRC.

Theory:

**What is CRC?**

CRC (Cyclic Redundancy Check) is an error-detecting code used in digital networks and storage devices to detect accidental changes to raw data. It works by generating a checksum (a small, fixed-size binary sequence) based on the data being checked. This checksum is appended to the data and transmitted or stored alongside it. Upon receipt or retrieval, the receiver recalculates the checksum and compares it with the transmitted checksum. If they match, the data is considered intact; if not, an error is detected.

**Why CRC?**

CRC is widely used in digital communication systems because it offers robust error detection capabilities with relatively simple implementation. It helps ensure data integrity by detecting common types of errors, such as single-bit errors, burst errors, and some types of random errors. CRC is particularly useful in environments where data corruption or transmission errors are common, such as in telecommunications, computer networking, and storage systems.

**How to use CRC**

To use CRC, you need to select a CRC polynomial, which determines the algorithm used to generate the checksum. The sender performs CRC calculation by dividing the data (message) by the CRC polynomial using modulo-2 arithmetic. The resulting remainder (checksum) is appended to the original data. At the receiver's end, the same CRC polynomial is applied to the received data (including the checksum). If the calculated checksum matches the received checksum, the data is considered errorfree. Otherwise, an error is detected, and appropriate error handling mechanisms can be implemented to retransmit or correct the data.

**CODE:**

*import* random

def xor(*a*, *b*):

*"""Perform XOR between two binary strings, ensuring they are of the same length."""*

result = []

length = min(len(*a*), len(*b*)) *# Ensure we don't go out of range*

*for* i *in* range(1, length):

result.append('0' *if* *a*[i] == *b*[i] *else* '1')

*return* ''.join(result)

def mod2\_division(*dividend*, *divisor*):

*"""Perform Modulo-2 Division and return the remainder"""*

pick = len(*divisor*)

tmp = *dividend*[:pick] *# Take initial segment of dividend*

*while* pick < len(*dividend*):

*if* tmp[0] == '1': *# Perform XOR*

tmp = xor(tmp, *divisor*) + *dividend*[pick]

*else*: *# If first bit is 0, simply shift*

tmp = xor(tmp, '0' \* len(*divisor*)) + *dividend*[pick]

pick += 1

*# Last step of XOR*

*if* tmp[0] == '1':

tmp = xor(tmp, *divisor*)

*else*:

tmp = xor(tmp, '0' \* len(*divisor*))

*return* tmp.zfill(len(*divisor*) - 1) *# Ensure remainder is of correct length*

def encode\_crc(*data*, *generator*):

*"""Encode data using CRC and return the transmitted message"""*

data\_bits = ''.join(format(ord(c), '08b') *for* c *in* *data*) *# Convert ASCII to binary (8-bit)*

print(f"converted binary value: {data\_bits}")

padded\_data = data\_bits + '0' \* (len(*generator*) - 1) *# Append zeros for CRC computation*

print(f"binary value with padded data: {padded\_data}")

remainder = mod2\_division(padded\_data, *generator*)

transmitted\_message = data\_bits + remainder *# Append CRC to the original data*

*return* transmitted\_message, remainder

def introduce\_error(*data*, *error\_rate*=0.1):

*"""Introduce random errors in the transmitted data"""*

*data* = list(*data*)

*for* i *in* range(len(*data*)):

*if* random.random() < *error\_rate*: *# With some probability, flip a bit*

*data*[i] = '1' *if* *data*[i] == '0' *else* '0'

*return* ''.join(*data*)

def detect\_error(*received\_data*, *generator*):

*"""Check for errors in the received data using CRC"""*

remainder = mod2\_division(*received\_data*, *generator*)

*return* remainder == '0' \* (len(*generator*) - 1), remainder *# If remainder is all 0s, no error*

*# Define the generator polynomial (CRC-8: x^8 + x^2 + x^1 + 1 => 100000111)*

GENERATOR = "100000111"

*# User input ASCII string*

original\_message = "HELLO"

print(f"Original Message: {original\_message}")

*# Encoding*

transmitted\_message, crc = encode\_crc(original\_message, GENERATOR)

print(f"Transmitted Binary Data with CRC: {transmitted\_message}")

print(f"CRC Checksum[appended to the binary value]: {crc}")

corrupted\_message = introduce\_error(transmitted\_message, *error\_rate*=0.1)

print(f"Corrupted Message: {corrupted\_message}")

*# Error Detection*

is\_correct, remainder = detect\_error(corrupted\_message, GENERATOR)

*if* is\_correct:

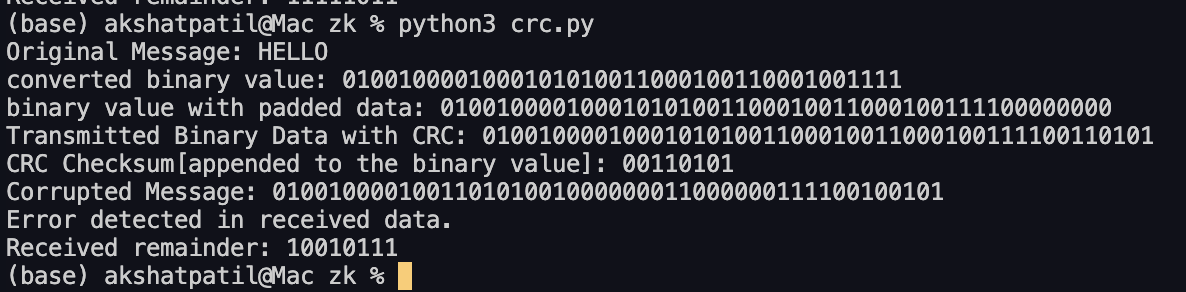
print("No errors detected in received data.")

*else*:

print("Error detected in received data.")

print(f"Received remainder: {remainder}")

**Output:**

****