B. M. S. COLLEGE OF ENGINEERING

Lab Report
Subject: Computer Networks Cycle 2 (Python)

Name	**Kiran M K**
USN	**1BM19CS073**
Semester&Section	**5B**
Dept.	**CSE**

Write a program for error detecting code using CRC-CCITT (16-bits).

```
import hashlib

def xor(a, b):

    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')

    return ''.join(result)



def mod2div(dividend, divisor):

    pick = len(divisor)
    tmp = dividend[0: pick]

    while pick < len(dividend):

        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + dividend[pick]

        else:
            tmp = xor('0' * pick, tmp) + dividend[pick]
        pick += 1
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0' * pick, tmp)

    checkword = tmp
    return checkword
```

```
def encodeData(data, key):
    l_key = len(key)

    appended_data = data + '0' * (l_key - 1)
    remainder = mod2div(appended_data, key)

    codeword = data + remainder
    return codeword

def decodeData(code, key):
    remainder = mod2div(code, key)
    return remainder

data=input("Enter Data: ")
print("dataword:"+str(data))

key = "10001000000100001"
print("generating polynomial:"+key)
codeword = encodeData(data, key)
print("Checksum: ",codeword)
print("Transmitted Codeword:"+str(codeword))
code = input("enter transmitted codeword:")

recieved_data = int(decodeData(code, key))

if recieved_data == 0:
    print("NO ERROR")
else:
    print("ERROR")
    print(recieved_data)
```

OUTPUT:

```
Enter Data: 1101
dataword:1101
generating polynomial:10001000000100001
Checksum:  11011101000110101101
Transmitted Codeword:1101110100011010 1101
enter transmitted codeword:1101110100011011 1010
ERROR
10111
```

Write a program for distance vector algorithm to find suitable path for transmission.

CODE:

```
class Topology:


    def __init__(self, array_of_points):
        self.nodes = array_of_points
        self.edges = []


    def add_direct_connection(self, p1, p2, cost):
        self.edges.append((p1, p2, cost))
        self.edges.append((p2, p1, cost))


    def distance_vector_routing(self):
        import collections
        for node in self.nodes:
            dist = collections.defaultdict(int)
            next_hop = {node: node}
            for other_node in self.nodes:
                if other_node != node:
                    dist[other_node] = 100000000 # infinity


            # Bellman Ford Algorithm
            for i in range(len(self.nodes)-1):
                for edge in self.edges:
                    src, dest, cost = edge
                    if dist[src] + cost < dist[dest]:
```

```
                    dist[dest] = dist[src] + cost
                    if src == node:
                        next_hop[dest] =dest
                    elif src in next_hop:
                        next_hop[dest] = next_hop[src]

            self.print_routing_table(node, dist, next_hop)
            print()

    def print_routing_table(self, node, dist, next_hop):
        print(f'Routing table for {node}:')
        print('Dest \t Cost \t Next Hop')
        for dest, cost in dist.items():
            print(f'{dest} \t {cost} \t {next_hop[dest]}')

    def start(self):
        pass
nodes = ['A', 'B', 'C', 'D', 'E']

t = Topology(nodes)

t.add_direct_connection('A', 'B', 1)
t.add_direct_connection('A', 'C', 5)
t.add_direct_connection('B', 'C', 3)
t.add_direct_connection('B', 'E', 9)
t.add_direct_connection('C', 'D', 4)
t.add_direct_connection('D', 'E', 2)
```

t.distance_vector_routing()

OUTPUT:

```
Routing table for A:
Dest     Cost    Next Hop
B        1       B
C        4       B
D        8       B
E        10      B
A        0       A

Routing table for B:
Dest     Cost    Next Hop
A        1       A
C        3       C
D        7       C
E        9       E
B        0       B

Routing table for C:
Dest     Cost    Next Hop
A        4       B
B        3       B
D        4       D
E        6       D
C        0       C

Routing table for D:
Dest     Cost    Next Hop
A        8       C
B        7       C
C        4       C
E        2       E
D        0       D

Routing table for E:
Dest     Cost    Next Hop
A        10      B
B        9       B
C        6       D
D        2       D
E        0       E
```

Implement Dijkstra's algorithm to compute the shortest path for a given topology.

CODE:

```
import sys


class Graph:
    def __init__(self,vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]


    def printSolution(self, dist):
        print("Vertex \tDistance from Source")
        for node in range(self.V):
            print(node, "\t", dist[node])


    def minDistance(self, dist, sptSet):


        min = sys.maxsize


        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v


        return min_index
```

```
    def dijkstra(self, src):
        dist = [sys.maxsize] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):

            u = self.minDistance(dist, sptSet)

            sptSet[u] = True

            for v in range(self.V):
                if self.graph[u][v] > 0 and sptSet[v] == False and dist[v] > dist[u] + self.graph[u][v]:
                        dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)
    g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
        [4, 0, 8, 0, 0, 0, 0, 11, 0],
        [0, 8, 0, 7, 0, 4, 0, 0, 2],
        [0, 0, 7, 0, 9, 14, 0, 0, 0],
        [0, 0, 0, 9, 0, 10, 0, 0, 0],
        [0, 0, 4, 14, 10, 0, 2, 0, 0],
        [0, 0, 0, 0, 0, 2, 0, 1, 6],
        [8, 11, 0, 0, 0, 0, 1, 0, 7],
        [0, 0, 2, 0, 0, 0, 6, 7, 0]
        ]
```

g.dijkstra(0)

OUTPUT:

```
Vertex  Distance from Source
0        0
1        4
2        12
3        19
4        21
5        11
6        9
7        8
8        14
```

Write a program for congestion control using Leaky bucket algorithm.

CODE:

```
class LeakyBucket:
    def control_congestion(self, input_stream):
        buffer = 0
        i = 0
        error_text = '\033[93m'
        reset = '\033[0m'
        for packet in input_stream:
            print(f"Packet no {i}\tPacket size {packet}")
            x = self.size - buffer
            if packet < x:
                buffer += packet
                print("\tBucket output successful")
                print(f'\tLast {packet} bytes sent')
            else:
                print(f'{error_text}\tBucket overflow {reset}')
                buffer = self.size
            buffer -= self.flow
            i += 1

        while buffer:
            sent = self.flow if self.flow < buffer else buffer
            print(f'Packet no {i}\tPacket size {sent}')
            print("\tBucket output successful")
            buffer -= sent
            i += 1

        i += 1

    def __init__(self, bucket_size, output_rate):
        self.size = bucket_size
        self.flow = output_rate



input_stream = [int(x) for x in input(
    "Enter input stream of packets: ").split(' ')]
```

```
bucket_size = int(input("Enter bucket size: "))
output_rate = int(input("Enter output data rate: "))
network = LeakyBucket(bucket_size, output_rate)
network.control_congestion(input_stream)
```

OUTPUT:

```
Enter input stream of packets: 100 200 300 200 400 250 400 230
Enter bucket size: 500
Enter output data rate: 100
Packet no 0     Packet size 100
        Bucket output successful
        Last 100 bytes sent
Packet no 1     Packet size 200
        Bucket output successful
        Last 200 bytes sent
Packet no 2     Packet size 300
        Bucket output successful
        Last 300 bytes sent
Packet no 3     Packet size 200
        Bucket overflow
Packet no 4     Packet size 400
        Bucket overflow
Packet no 5     Packet size 250
        Bucket overflow
Packet no 6     Packet size 400
        Bucket overflow
Packet no 7     Packet size 230
        Bucket overflow
Packet no 8     Packet size 100
        Bucket output successful
Packet no 9     Packet size 100
        Bucket output successful
Packet no 10    Packet size 100
        Bucket output successful
Packet no 11    Packet size 100
        Bucket output successful
```

Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

SERVER CODE

```
import socket

HOST = ‘127.0.0.1'

PORT = 65432

print('\033[36m======== SERVER ========\033[0m')


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.bind((HOST, PORT))
    sock.listen(1)
    conn, addr = sock.accept()

    with conn:
        print(f'Connected by: {addr}')
        while True:
            data = conn.recv(1024)
            if not data:
                break

            filename = data.decode('utf-8')
            print(f'Received Filename: {filename}')

            try:
                with open(filename, 'r') as f:
                    data = f.read()
                data = bytes(data, 'utf-8')
            except:
                data = bytes(f'File {filename} not found', 'utf-8')

            conn.sendall(data)
            print(f'Sent: {data}')
            print()
```

CLIENT CODE:

```
import socket
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 65432
print('\033[32m======== CLIENT ========\033[0m')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((SERVER_HOST, SERVER_PORT))
    while True:
        filename = input('Enter file name: ')

        if not filename:
            break

        sock.sendall(bytes(filename, 'utf-8'))
        print(f'Sent: {filename}')

        data = sock.recv(1024)
        contents = data.decode('utf-8')
        print(f'Received: {contents}')
        print()
```

SERVER OUTPUT:

```
======== SERVER ========
Connected by: ('127.0.0.1', 50011)
Received Filename: file.txt
Sent: b'abcdefghijklnmopqrstuvwxyz\nABCDEFGHIJKLNMOPQRSTUVWXYZ\n\n'
```

CLIENT OUTPUT:

```
======== CLIENT ========
Enter file name: file.txt
Sent: file.txt
Received: abcdefghijklnmopqrstuvwxyz
ABCDEFGHIJKLNMOPQRSTUVWXYZ


Enter file name:
```

Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present

SERVER CODE:

```
import socket
HOST = '127.0.0.1'
PORT = 65432
print('\033[36m======== SERVER ========\033[0m')

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    sock.bind((HOST, PORT))
    while True:
        data, addr = sock.recvfrom(1024)
        if not data:
            break

        filename = data.decode('utf-8')
        print(f'Received Filename: {filename} From: {addr}')

        try:
            with open(filename, 'r') as f:
                data = f.read()
            data = bytes(data, 'utf-8')
        except:
            data = bytes(f'File {filename} not found', 'utf-8')

        sock.sendto(data, addr)
        print(f'Sent: {data} To: {addr}')
        print()
```

CLIENT CODE:

```
import socket
HOST = '127.0.0.1'
PORT = 65432
print('\033[32m======== CLIENT ========\033[0m')

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    sock.connect((HOST, PORT))
```

```
while True:
    filename = input('Enter file to request from server: ')

    if not filename:
        break

    sock.sendall(bytes(filename, 'utf-8'))
    print(f'Sent: {filename}')

    data = sock.recv(1024).decode('utf-8')
    print(f'Received: {data}')
    print()
```

SERVER OUTPUT:

```
======== SERVER ========
Received Filename: file.txt From: ('127.0.0.1', 50056)
Sent: b'abcdefghijklnmopqrstuvwxyz\nABCDEFGHIJKLNMOPQRSTUVWXYZ\n\n' To: ('127.0.0.1', 50056)
```

CLIENT OUTPUT:

```
======== CLIENT ========
Enter file name: file.txt
Sent: file.txt
Received: abcdefghijklnmopqrstuvwxyz
ABCDEFGHIJKLNMOPQRSTUVWXYZ


Enter file name:
```