

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :

Week 6 – Socket Programming using TCP Iterative and concurrent

1. Explain the purpose of the socket module in Python and how it facilitates communication between processes.

Answer: The socket module in Python provides a set of tools for working with sockets, allowing processes to communicate over a network. Sockets offer a standard mechanism for processes on different devices to establish a connection and exchange data. This module abstracts the complexities of network communication and provides a simple interface for creating servers and clients.

2. Describe the role of the while loop in the iterative server code. How does it handle multiple client connections?

Answer: The while loop in the iterative server code is responsible for repeatedly accepting client connections. It creates a loop that waits for clients to connect, handles the communication with each client sequentially, and closes the connection before returning to the loop to wait for the next client. This makes the server iterative because it processes one client at a time, ensuring that each client is served sequentially.

3. Explain why the concurrent server uses threads (threading module). How does this approach enable the server to handle multiple client connections simultaneously?

Answer: The concurrent server uses threads to handle multiple client connections simultaneously. Each new client connection triggers the creation of a new thread, allowing the server to serve multiple clients concurrently. Threads enable parallel execution, ensuring that the server can handle multiple tasks concurrently without blocking other connections. This approach improves the server's responsiveness and efficiency.

4. Elaborate on the line `client_socket.send(message.encode())` in the client code. What is the purpose of encoding the message before sending it to the server?

Answer: The line `client_socket.send(message.encode())` sends the message to the server after encoding it using the `encode()` method. The `encode()` method converts the string message into a byte-like object, which is the format that can be sent over the network. Encoding is necessary because network communication deals with binary data, and the `encode()` method ensures that the string is represented in a form suitable for transmission.

5. Discuss whether there are any modifications needed in the client code when connecting to a concurrent server compared to an iterative server.

Answer: No modifications are required in the client code when connecting to either an iterative or concurrent server. The client code remains the same because the client is generally unaware of the server's implementation details. As long as the client correctly establishes a connection, sends messages, and receives responses, it can interact with both types of servers seamlessly. The differences in server implementation (iterative or concurrent) are transparent to the client.

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

Lab Programs -6

iterserver.py
import socket

```
def iterative_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('127.0.0.1', 8080))
    server_socket.listen(1)
    print("Iterative Server is listening on port 8080...")
    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Connection from {client_address}")
        data = client_socket.recv(1024)
        while data:
            print(f"Received: {data.decode()}")
            client_socket.send(data)
            data = client_socket.recv(1024)
        print(f"Connection from {client_address} closed.")
        client_socket.close()
if __name__ == "__main__":
    iterative_server()
```

Output:

```
lws/Python/cn/iterserver.py
Iterative Server is listening on port 8080...
Connection from ('127.0.0.1', 55208)
Received: Hello, server!
Connection from ('127.0.0.1', 55208) closed.
```

iterclient.py
import socket

```
def iterative_client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('127.0.0.1', 8080))

    message = "Hello, server!"
    client_socket.send(message.encode())

    data = client_socket.recv(1024)
    print(f"Received from server: {data.decode()}")
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

```
client_socket.close()
```

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

Output:

```
op/My Filws/Python/cn/iterclient.py"  
Received from server: Hello, server!  
PS C:\Users\hp\OneDrive - Vasavi College Of Engineer  
ing\Desktop\My Filws\Python\cn>
```

Concurrent server.py

```
import socket  
import threading
```

```
def handle_client(client_socket, address):  
    print(f"Connection from {address}")
```

```
    data = client_socket.recv(1024)  
    while data:  
        print(f"Received: {data.decode()}")  
        client_socket.send(data)  
        data = client_socket.recv(1024)
```

```
    print(f"Connection from {address} closed.")  
    client_socket.close()
```

```
def concurrent_server():  
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    server_socket.bind(('127.0.0.1', 8080))  
    server_socket.listen(5)
```

```
    print("Concurrent Server is listening on port 8080...")
```

```
    while True:  
        client_socket, client_address = server_socket.accept()  
        client_handler = threading.Thread(target=handle_client, args=(client_socket,  
client_address))
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

```
client_handler.start()
if __name__ == "__main__":
    concurrent_server()
```

Output:

```
Concurrent Server is listening on port 8080...
Connection from ('127.0.0.1', 54912)
Received: Hello, server!
Connection from ('127.0.0.1', 54912) closed.
```

Concurrent client.py

```
import socket
import threading
def handle_client(client_socket, address):
    print(f"Connection from {address}")

    data = client_socket.recv(1024)
    while data:
        print(f"Received: {data.decode()}")
        client_socket.send(data)
        data = client_socket.recv(1024)
    print(f"Connection from {address} closed.")
    client_socket.close()
def concurrent_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('127.0.0.1', 8080))
    server_socket.listen(5)
    print("Concurrent Server is listening on port 8080...")
    while True:
        client_socket, client_address = server_socket.accept()
        client_handler = threading.Thread(target=handle_client, args=(client_socket,
client_address))
        client_handler.start()
if __name__ == "__main__":
    concurrent_server()
```

Output:

```
Received from server: Hello, server!
PS C:\Users\hp\OneDrive - Vasavi College Of Engineer
ing\Desktop\My Filws\Python\cn>
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :

WEEK - 7 Prelab questions

Socket programming using UDP concurrent and iterative

1. Describe the main differences between UDP and TCP protocols in the context of socket programming. Highlight scenarios where you might choose one over the other.

Ans: UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) are both transport layer protocols, but they differ in several ways. TCP provides reliable, connection-oriented communication with error-checking and data integrity, making it suitable for applications where data accuracy is crucial. UDP, on the other hand, is connectionless and offers faster, less reliable communication, making it suitable for real-time applications such as video streaming or online gaming.

2. Explain the key features of the iterative UDP server code. How does it handle incoming messages from multiple clients in a sequential manner?

Ans: The iterative UDP server uses a single socket to receive and send messages. It enters a loop where it waits for incoming messages using `recvfrom()`, processes each message sequentially, and then sends a response back to the client using `sendto()`. Since UDP is connectionless, the server does not maintain state between clients, and each message is processed independently.

3. Discuss the purpose of using threads in the concurrent UDP server code. How does this approach allow the server to handle multiple clients simultaneously?

Ans: The concurrent UDP server uses threads to handle multiple clients simultaneously. Each thread is responsible for receiving messages from a specific client using `recvfrom()` and sending responses using `sendto()`. By creating a new thread for each client, the server can process multiple clients concurrently without blocking other clients. This improves the server's responsiveness and efficiency.

4. Elaborate on the line `client_socket.sendto(message.encode(), ('127.0.0.1', 8080))` in both the iterative and concurrent UDP client codes. Why is encoding necessary, and what does the server address represent?

Ans: The line sends the encoded message to the server's address `('127.0.0.1', 8080)` using `sendto()`. Encoding is necessary because UDP deals with raw bytes, and the `encode()` method converts the string message into a byte-like object. The server address represents the destination where the message will be sent—specifically, the IP address `('127.0.0.1')` and port `('8080')` on the local machine in this case.

5. Discuss whether there are any modifications needed in the client code when connecting to a concurrent UDP server compared to an iterative UDP server.

Ans: No modifications are required in the client code when connecting to either an iterative or concurrent UDP server. The client code remains the same because the client is generally unaware of the server's implementation details. As long as the client correctly sends and receives messages, it can interact with both types of servers seamlessly. The differences in server implementation (iterative or concurrent) are transparent to the client.

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

Iterative UDP server.py

```
import socket
def iterative_udp_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind(('127.0.0.1', 8080))

    print("Iterative UDP Server is listening on port 8080...")

    while True:
        data, client_address = server_socket.recvfrom(1024)
        print(f"Received from {client_address}: {data.decode()}")
        server_socket.sendto(data, client_address)

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

Output:

```
lws/Python/cn/udp/iterserver.py"
Iterative UDP Server is listening on port 8080...
Received from ('127.0.0.1', 55434): Hello, server!
```

client.py

```
import socket

def iterative_udp_client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    message = "Hello, server!"
    client_socket.sendto(message.encode(), ('127.0.0.1', 8080))

    data, server_address = client_socket.recvfrom(1024)
    print(f"Received from server: {data.decode()}")

    client_socket.close()

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

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

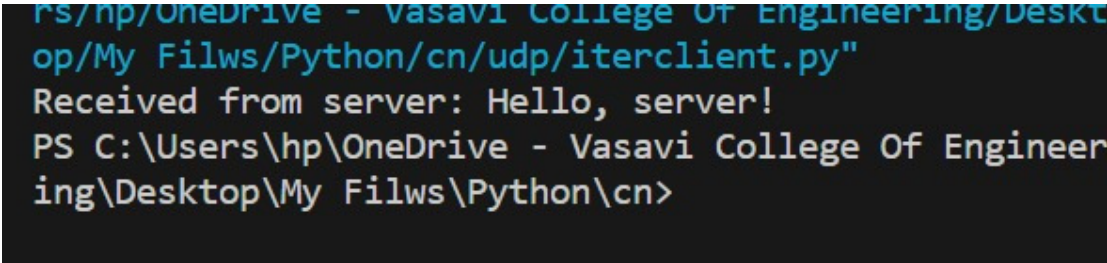
Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

Output:



```
PS C:\Users\hp\OneDrive - Vasavi College Of Engineering\Desktop\My Filws\Python\cn> python udp/iterclient.py
Received from server: Hello, server!
PS C:\Users\hp\OneDrive - Vasavi College Of Engineering\Desktop\My Filws\Python\cn>
```

Concurrent UDP server.py

```
import socket
import threading
```

```
def handle_client(server_socket):
```

```
    while True:
```

```
        data, client_address = server_socket.recvfrom(1024)
```

```
        print(f"Received from {client_address}: {data.decode()}")
```

```
        server_socket.sendto(data, client_address)
```

```
def concurrent_udp_server():
```

```
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
    server_socket.bind(('127.0.0.1', 8080))
```

```
    print("Concurrent UDP Server is listening on port 8080...")
```

```
    for _ in range(5): # Limit to 5 concurrent threads for simplicity
```

```
        client_handler = threading.Thread(target=handle_client, args=(server_socket,))
```

```
        client_handler.start()
```

```
if __name__ == "__main__":
```

```
    concurrent_udp_server()
```

Output:

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

```
OneDrive - Vasavi College Of Engineering\Desktop\My Filws/Python/cn/udp/conserver.py"
Concurrent UDP Server is listening on port 8080...
Received from ('127.0.0.1', 62589): Hello, server!
```

Client.py

```
import socket
```

```
def concurrent_udp_client():
```

```
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
    message = "Hello, server!"
```

```
    client_socket.sendto(message.encode(), ('127.0.0.1', 8080))
```

```
    data, server_address = client_socket.recvfrom(1024)
```

```
    print(f"Received from server: {data.decode()}")
```

```
    client_socket.close()
```

```
if __name__ == "__main__":
```

```
    concurrent_udp_client()
```

Output:

```
op/My Filws/Python/cn/udp/conclient.py"
Received from server: Hello, server!
PS C:\Users\hp\OneDrive - Vasavi College Of Engineering\Desktop\My Filws\Python\cn>
```


VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :

WEEK 8 - SHORTEST PATH (dijkstra algorithm)

PRELAB QUESTIONS - 8

1. Provide a detailed explanation of Dijkstra's algorithm. How does it work, and what problem does it aim to solve?

Ans: Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge weights. It starts from a source node and iteratively selects the vertex with the smallest known distance to update its neighbors' distances. The algorithm maintains a set of vertices whose shortest distance from the source is known and continually expands this set until it reaches the destination or all vertices have been explored.

2. Explain the role of a priority queue in the implementation of Dijkstra's algorithm. Why is it crucial for the algorithm's efficiency?

Ans: A priority queue is used in Dijkstra's algorithm to efficiently select the vertex with the smallest known distance in each iteration. It ensures that the algorithm always considers the most promising vertex first. The priority queue allows for fast retrieval of the vertex with the minimum distance, improving the overall efficiency of the algorithm.

3. Describe the initial setup steps in Dijkstra's algorithm. What values are assigned to the distances and predecessors before the algorithm begins?

Ans: In the initialization step of Dijkstra's algorithm, all distances are set to infinity, except for the source node, whose distance is set to 0. Predecessor values are typically set to null or an undefined state. This setup ensures that the algorithm starts with a clean slate and progressively updates the shortest distances as it explores the graph.

4. Elaborate on the relaxation step in Dijkstra's algorithm. How does it update the distances and predecessors of neighboring vertices?

Ans: The relaxation step in Dijkstra's algorithm involves comparing the current shortest distance to a vertex with the sum of the distance to the current vertex and the weight of the edge connecting them. If the sum is smaller, the distance to the neighboring vertex is updated, and the predecessor is set to the current vertex. This process continues until all adjacent vertices have been considered.

5. Discuss the impact of negative edge weights on Dijkstra's algorithm. How does the algorithm behave when negative weights are present in the graph?

Ans: Dijkstra's algorithm assumes non-negative edge weights. When negative edge weights are present, the algorithm may produce incorrect results because it can be misled by a negative-weight cycle. In the presence of negative weights, it is more appropriate to use algorithms like Bellman-Ford that can handle negative cycles and provide accurate shortest paths even in such scenarios.

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

LAB PROGRAM – 8

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

        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):
            x = self.minDistance(dist, sptSet)
            sptSet[x] = True
            for y in range(self.V):
                if self.graph[x][y] > 0 and sptSet[y] == False and \
                   dist[y] > dist[x] + self.graph[x][y]:
                    dist[y] = dist[x] + self.graph[x][y]

        self.printSolution(dist)

if __name__ == "__main__":
    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],
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

```
[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

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :

Week 9 - RSA algorithm

Prelab questions - 9

1. Provide a detailed explanation of the RSA algorithm. How does it achieve secure communication, and what are the key components of the algorithm?

Ans: The RSA algorithm is a widely used public-key cryptosystem that enables secure communication over insecure channels. It involves the use of a public key for encryption and a private key for decryption. The key components include the generation of a public and private key pair, the use of modular arithmetic, and the security of the algorithm relying on the difficulty of factoring large composite numbers.

2. Explain the process of key generation in the RSA algorithm. What steps are involved in generating a public-private key pair?

Ans: Key generation in RSA involves the following steps:

1. Select two large prime numbers, p and q .
2. Compute $n = p * q$, where n is used as the modulus for both the public and private keys.
3. Compute the totient $\phi(n) = (p-1)(q-1)$.
4. Choose a public exponent e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
5. Calculate the private exponent d such that $d \equiv e^{-1} \pmod{\phi(n)}$.
6. The public key is (n, e) , and the private key is (n, d) .

3. Describe the process of encryption in the RSA algorithm. How is plaintext transformed into ciphertext using the public key?

Ans: Encryption in RSA involves the following steps:

1. Represent the plaintext message as an integer m , where $0 < m < n$.
2. Use the recipient's public key (n, e) to compute the ciphertext $c \equiv m^e \pmod{n}$.
3. The ciphertext c is then sent to the recipient for decryption.

4. Elaborate on the process of decryption in the RSA algorithm. How is ciphertext transformed back into plaintext using the private key?

Ans: Decryption in RSA involves the following steps:

1. Use the recipient's private key (n, d) to compute the original message $m \equiv c^d \pmod{n}$.
2. The computed m represents the original plaintext message.

5. Discuss the security considerations in the RSA algorithm. What factors contribute to the algorithm's security, and what vulnerabilities should be avoided?

Ans: The security of RSA relies on the difficulty of factoring the product of two large prime numbers. The key length is crucial for security, with longer keys providing higher resistance to attacks. It's essential to avoid using small or related primes for key generation.

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :

Lab program - 9

```
import random
import math
prime = set()
public_key = None
private_key = None
n = None
def primefiller():
    seive = [True] * 250
    seive[0] = False
    seive[1] = False
    for i in range(2, 250):
        for j in range(i * 2, 250, i):
            seive[j] = False
    for i in range(len(seive)):
        if seive[i]:
            prime.add(i)
def pickrandomprime():
    global prime
    k = random.randint(0, len(prime) - 1)
    it = iter(prime)
    for _ in range(k):
        next(it)
    ret = next(it)
    prime.remove(ret)
    return ret

def setkeys():
    global public_key, private_key, n
    prime1 = pickrandomprime()
    prime2 = pickrandomprime()

    n = prime1 * prime2
    fi = (prime1 - 1) * (prime2 - 1)
    e = 2
    while True:
        if math.gcd(e, fi) == 1:
            break
        e += 1

    # d = (k*Φ(n) + 1) / e for some integer k
    public_key = e

    d = 2
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :

```
while True:
    if (d * e) % fi == 1:
        break
    d += 1

private_key = d
# To encrypt the given number
def encrypt(message):
    global public_key, n
    e = public_key
    encrypted_text = 1
    while e > 0:
        encrypted_text *= message
        encrypted_text %= n
        e -= 1
    return encrypted_text
# To decrypt the given number
def decrypt(encrypted_text):
    global private_key, n
    d = private_key
    decrypted = 1
    while d > 0:
        decrypted *= encrypted_text
        decrypted %= n
        d -= 1
    return decrypted
def encoder(message):
    encoded = []
    # Calling the encrypting function in encoding function
    for letter in message:
        encoded.append(encrypt(ord(letter)))
    return encoded

def decoder(encoded):
    s = ""
    # Calling the decrypting function decoding function
    for num in encoded:
        s += chr(decrypt(num))
    return s

if __name__ == '__main__':
    primefiller()
    setkeys()
    message = "Test Message"
    # Uncomment below for manual input
    # message = input("Enter the message\n")
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad – 500 031.

DEPARTMENT OF: COMPUTER SCIENCE AND ENGINEERING

NAME OF THE LABORATORY: COMPUTER NETWORKSLAB

Name: G.Bhupathi Roll No: 1602-21-733-011 Page No. :__

```
coded = encoder(message)
```

```
print("Initial message:")
```

```
print(message)
```

```
print("\n\nThe encoded message(encrypted by public key)\n")
```

```
print(".join(str(p) for p in coded))
```

```
print("\n\nThe decoded message(decrypted by public key)\n")
```

```
print(".join(str(p) for p in decoder(coded)))
```

Output:

Initial message:

Test Message

The encoded message(encrypted by public key)

863312887135951593413927434912887135951359583051879012887

The decoded message(decrypted by private key)

Test Message