EXPERIMENT-01

BFS

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = []
queue = []

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)


print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

OUTPUT:

```
Following is the Breadth-First Search
5 3 7 2 4 8
```

DFS

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = set()

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

OUTPUT:

```
Following is the Depth-First Search

5

3

2

4

8

7
```

EXPERIMENT-02

A* Algorithm

```
from collections import deque

class Graph:

    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list

    def get_neighbors(self, v):
        return self.adjacency_list[v]

    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }

        return H[n]

    def a_star_algorithm(self, start_node, stop_node):
        open_list = set([start_node])
        closed_list = set([])

        g = {}

        g[start_node] = 0

        parents = {}
        parents[start_node] = start_node

        while len(open_list) > 0:
            n = None

            for v in open_list:
                if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                    n = v

            if n == None:
                print('Path does not exist!')
                return None

            if n == stop_node:
                reconst_path = []
```

```
                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]

                reconst_path.append(start_node)

                reconst_path.reverse()

                print('Path found: {}'.format(reconst_path))
                return reconst_path

            for (m, weight) in self.get_neighbors(n):
                if m not in open_list and m not in closed_list:
                    open_list.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight

                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n

                        if m in closed_list:
                            closed_list.remove(m)
                            open_list.add(m)

            open_list.remove(n)
            closed_list.add(n)

        print('Path does not exist!')
        return None

adjacency_list = {
'A': [('B', 1), ('C', 3), ('D', 7)],
'B': [('D', 5)],
'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

OUTPUT:

```
Path found: ['A', 'B', 'D']
```

EXPERIMENT-03

Prims algorithm

```
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 printMST(self, parent):

        print("Edge \tWeight")

        for i in range(1, self.V):

            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

    def minKey(self, key, mstSet):


        # Initialize min value

        min = sys.maxsize


        for v in range(self.V):

            if key[v] < min and mstSet[v] == False:

                min = key[v]

                min_index = v


        return min_index

    def primMST(self):

        key = [sys.maxsize] * self.V

        parent = [None] * self.V # Array to store constructed MST

        key[0] = 0

        mstSet = [False] * self.V


        parent[0] = -1 # First node is always the root of
```

```
        for cout in range(self.V):
            u = self.minKey(key, mstSet)
            mstSet[u] = True

            for v in range(self.V):
                if self.graph[u][v] > 0 and mstSet[v] == False
                and key[v] > self.graph[u][v]:
                    key[v] = self.graph[u][v]
                    parent[v] = u

        self.printMST(parent)

if __name__ == '__main__':
    g = Graph(5)
    g.graph = [[0, 2, 0, 6, 0],
            [2, 0, 3, 8, 5],
            [0, 3, 0, 0, 7],
            [6, 8, 0, 0, 9],
            [0, 5, 7, 9, 0]]

    g.primMST()
```

OUTPUT:

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

Experiment-04

NQUEEN PROBLEMS

```
print ("Enter the number of queens")

N = int(input())

board = [[0]*N for _ in range(N)]

def is_attack(i, j):

    for k in range(0,N):

        if board[i][k]==1 or board[k][j]==1:

            return True


    for k in range(0,N):

        for l in range(0,N):

            if (k+l==i+j) or (k-l==i-j):

                if board[k][l]==1:

                    return True

    return False

def N_queen(n):

    if n==0:

        return True

    for i in range(0,N):

        for j in range(0,N):

            '''checking if we can place a queen here or not

            queen will not be placed if the place is being attacked
```

```
            or already occupied'''
            if (not(is_attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1

                if N_queen(n-1)==True:
                    return True
                board[i][j] = 0
    return False
N_queen(N)
for i in board:
    print (i)
```

Output:

```
Enter the number of queens
8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

Experiment 5

```python
import random

responses = {
    "hi": ["Hello!", "Hi there!", "Hi!"],
    "how are you": ["I'm doing well, thank you!", "I'm fine, thanks for asking.", "I'm good, thanks!"],
    "what's your name": ["My name is Chatbot.", "I'm Chatbot!", "I'm just a simple chatbot without a name."],
    "bye": ["Goodbye!", "See you later!", "Have a nice day!"],
    "thank you": ["You're welcome!", "No problem!", "Anytime!"],
    "default": ["I'm sorry, I don't understand.", "Can you please rephrase that?", "I'm not sure what you mean."]
}

def chatbot():
    print(random.choice(responses["hi"]))

    while True:
        message = input("> ")

        if "hi" in message.lower():
            print(random.choice(responses["hi"]))
        elif "how are you" in message.lower():
            print(random.choice(responses["how are you"]))
        elif "what's your name" in message.lower():
            print(random.choice(responses["what's your name"]))
        elif "bye" in message.lower():
            print(random.choice(responses["bye"]))
            break
        elif "thank" in message.lower():
            print(random.choice(responses["thank you"]))
        else:
            print(random.choice(responses["default"]))

chatbot()
```

Output :-

Hi there!

> hii

Hi there!

> how are you

I'm fine, thanks for asking.

> what is your name

Can you please rephrase that?

> what's your name

My name is Chatbot.

> thank

Anytime!

> bye

Goodbye!

In [3]:

```python
#Informtaion Practical 1 code


import ctypes        #It provides C compatible data types, and allows calling functions in

s = "\Hello World"


and_result = ""
or_result = ""
xor_result = ""
for c in s:

    and_result += chr(ord(c) & 127)
    or_result += chr(ord(c) | 127)
    xor_result += chr(ord(c) ^ 127)




print("AND result:", and_result)
print("OR result :", or_result)
print("XOR result:", xor_result)
```

```
AND result: \Hello World
OR result :
□□R result: #7□□□□_(□
```

In [31]:

```
#IS PRACTICAL 2 CODE

from pycipher import caesar

def caesar_encrypt(plaintext,key):
    c=caesar.Caesar(key)
    ciphertext=c.encipher(plaintext)
    return ciphertext

def caesar_decrypt(ciphertext,key):
    c=caesar.Caesar(key)
    plaintext=c.decipher(ciphertext)
    return plaintext

plaintext=input()

key=int(input())

ciphertext = caesar_encrypt(plaintext, key)
decrypted_plaintext = caesar_decrypt(ciphertext, key)

print('Plaintext: ', plaintext)
print('Ciphertext:', ciphertext)
print('Decrypted plaintext:', decrypted_plaintext)
```

```
HELLOWORLD
4
Plaintext:  HELLOWORLD
Ciphertext: LIPPSASVPH
Decrypted plaintext: HELLOWORLD
```

In [ ]:

In [ ]:

In [1]:

```
!pip install DES
```

```
Requirement already satisfied: DES in c:\users\dell\anaconda3\lib\site-packages
(1.0.6)
```

In [9]:

```
from des import DesKey



key = DesKey(b"secret_k")           # Define the key to be used (64 bits or 8 bytes)


plaintext = input("Enter plaintext: ").encode()      # Prompt the user to enter plaintext


ciphertext = key.encrypt(plaintext, padding=True)    # Encrypt the plaintext using DES


print("Ciphertext:", ciphertext.hex())           # Print the ciphertext


decrypted_plaintext = key.decrypt(ciphertext, padding=True)     # Decrypt the ciphertext using


print("Decrypted plaintext:", decrypted_plaintext.decode())     # Print the decrypted plaintex
```

```
Enter plaintext: Jayawantrao Sawant College of Engineering, Hadapsar
Ciphertext: 87c737e5c73f4d6986d06303f80a4902fc6bdeba1195ac2f6838cfdccd68c56944c0
7d0b02e41b39f740646eb2597a35a09732c507a34702
Decrypted plaintext: Jayawantrao Sawant College of Engineering, Hadapsar
```

In [20]:

```
!pip install AES
!pip install pycryptodome
```

```
Requirement already satisfied: AES in c:\users\dell\anaconda3\lib\site-packa
ges (1.2.0)
Requirement already satisfied: pycryptodome in c:\users\dell\anaconda3\lib\s
ite-packages (3.17)
```

In [22]:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import os

key = os.urandom(16)

plaintext = input("Enter the Plaintext :").encode()

padded_plaintext = pad(plaintext, AES.block_size)

cipher = AES.new(key, AES.MODE_CBC)

ciphertext = cipher.encrypt(padded_plaintext)

print("Ciphertext:", ciphertext.hex())

cipher = AES.new(key, AES.MODE_CBC, iv=cipher.iv)

decrypted_padded_plaintext = cipher.decrypt(ciphertext)

decrypted_plaintext = decrypted_padded_plaintext.rstrip(b"\x00")

print("Decrypted plaintext:", decrypted_plaintext.decode())
```

```
Enter the Plaintext :JSPM
Ciphertext: 5aa76f05de8a34e65d3f11bab0d04ab5
Decrypted plaintext: JSPM
```

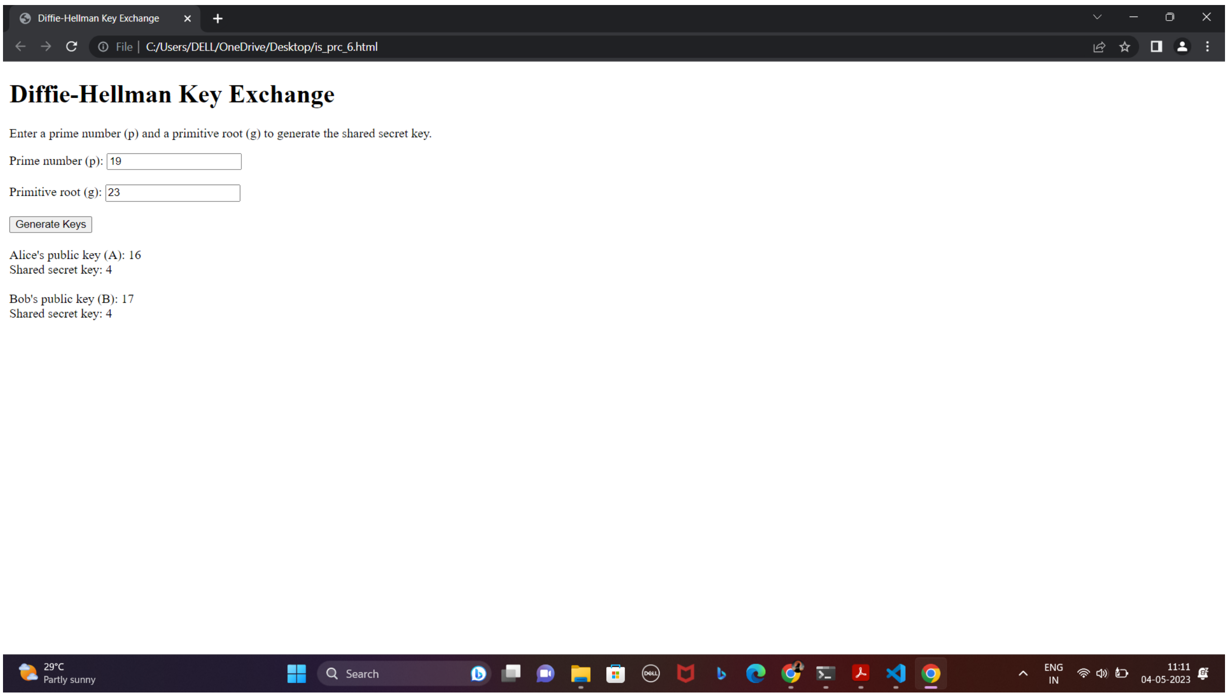
In [ ]:

```

```

In [2]:
```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto import Random

random_generator = Random.new().read
key = RSA.generate(2048, random_generator)


public_key = key.publickey()
private_key = key


print("Public key:")
print(public_key.export_key().decode())

print("Private key:")
print(private_key.export_key().decode())


message = input("Enter the Text :").encode()
cipher = PKCS1_OAEP.new(public_key)
ciphertext = cipher.encrypt(message)

print("Ciphertext:", ciphertext.hex())

# Decrypt the message using the private key
cipher = PKCS1_OAEP.new(private_key)
decrypted_message = cipher.decrypt(ciphertext)

print("Decrypted message:", decrypted_message.decode())
```

```
Public key:
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAhbT5sb1WEIRd/N0FFB+y
FsqQDtMX8a8/qblH6oJ/JWb9YyR4GTzlYr2sa7LQJbZc4Bxn017u+ESHHrpbVcAF
mCU3hmSw7vqhhXd1D0TQQm+tDkqeFoNOf5vjBeTU5frhQgLQVhm3/0xZk+G/1S1x
1Sjd6AlYSXQS6TV5hsWBwXzZsDCti5do5vGSrvqEN/7qjxl+JZFiFY4nu+w7fKQW
VJwhdvQ8/q3uL+2Dqjt5euLgQfqNYeEJTFMbD7f9JWhuMEDOmh0qRv9p2VW8+4Yq
wu5SKTFLPaeP4lR4xdFdgvrAnHGXHg/ssiov8LUTagQXcfH3AZdtoYMcbdn6PR+j
gwIDAQAB
-----END PUBLIC KEY-----
Private key:
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAhbT5sb1WEIRd/N0FFB+yFsqQDtMX8a8/qblH6oJ/JWb9YyR4
GTzlYr2sa7LQJbZc4Bxn017u+ESHHrpbVcAFmCU3hmSw7vqhhXd1D0TQQm+tDkqe
FoNOf5vjBeTU5frhQgLQVhm3/0xZk+G/1S1x1Sjd6AlYSXQS6TV5hsWBwXzZsDCt
i5do5vGSrvqEN/7qjxl+JZFiFY4nu+w7fKQWVJwhdvQ8/q3uL+2Dqjt5euLgQfqN
YeEJTFMbD7f9JWhuMEDOmh0qRv9p2VW8+4Yqwu5SKTFLPaeP4lR4xdFdgvrAnHGX
Hg/ssiov8LUTagQXcfH3AZdtoYMcbdn6PR+jgwIDAQABAoIBADfrTXaRefokce58
PVCCRQgVJZSdomj440ZHcBVbCHQLE7QwH25msoTOUNsmCLmNAGDWYHHdRVJdzqkC
RYqiaXSNNCCn4fvApGbeY/3DyNnQPO86M820ktL5zkO7xXJ9r1HdPWaVEQfb1abs
hj9Q50e/9LbDmcH2oqYYMEA2QCM8SknIyMtrPmJLnAcKRmwo6AvgKwsK4ruRq4LY
l4a/wW6jmAx1q7gs59WW4NY9Q6M9e6LshPX6yXSQL3COjrmNiFCGH6PIxiCQUw6Y
c85lI6uPcP2ekEOCQDNvZfR47v7zvIhffIStvwkUnovKDY/ym/MhlgdGilYv2ffk
Ln2LCXECgYEAuH+QDg85NrWKRf5E/UDS1Imt3rSk6ruhuwb80PxOtwxuQwas0kUs
xaOP0diGZsLnz+Yb5+TZUO4G0PvFXBawsPMSeRsT0JodLa/ThwOY0MjFA8L06CAs
cvU+HENoFHafosaqNb1wyz9Ah6v080jZzROu+nwYW6CCQlGUuUAJm3sCgYEAuYZN
dElTEu+yWk+ur55Tu32YnUWy5+mkxtoDmKWk5oyo6FkLLnuq+5mALuieDRnCkUGh
2BjhNRKejlir527vnMECztxsXUvMsPDdus2noxpTx4d7KZIdLu/DKDKCNZzgT0oS
KrCw4Rt0yhW6M160cHpSx2M0NFvl/4A3aI4v9ZkCgYEAkuj4c0mJ/FdYMF3E34gz
mQu8iD59p3Aa20s4Gs0dfMKINhTlnlV4zjNsfOMWOlNutOv9QozGwpuWj+g7AhYL
Qgt7pTV7dLqTC2MbY0Ho+C82OcWOsBuWaT+A4o6GHO1MNhhNRpd/bkgCgUXaubJD
w/mtdkVEPAm2T0qYy14DiGsCgYEAqhC+95YvnPA1IixhCYOtZfVf7OtzcLvw3IgF
O+y8GMtgGn1ljpq2xiSUKGv8Vi4C+Xyci6di6m/DAOGv01sSMzOLC21ruKo/XQOv
fgn/XbhIjGNZN2ZFcj0/PJ3wVo0T4hsYRCHsQq4UhRsdsPFjnqfDMhtyQ3zqfShf
omyU8iECgYBb8d7P14nII/ElgEDYu8YgCz45FDwWpt1PgE0GESSraDnYq/neT9xS
BtCYcFybrdfFm6yXfhM4wiKCRoHeC+HJ58f33ZJVuiLNF15QjPBHEZOT8xmsB425
u1gLcveXQ+Yh/88aTqipfVyPe10kRfAwI0WgVEGxzqkJ33/TAUwLrw==
-----END RSA PRIVATE KEY-----
Enter the Text :Hello World,Welcome
Ciphertext: 41a29b6e2891226adbaa60e4d8e1915a3a4b5ab823b7e18ad7e6d415d8ec6062456ef7e
db6d5d99ffa554adb9604ec6f0771f747172dffbed7da16143936f6d945594c4dd43f91c108aeef4b74
f14397b0875592bfd2b48fd4ac037680eff4742ef3612e20acafe1c0dfc5b82d76c907c9e65985547a3
c127bfed9efc38669d81939c6246b009ba22c93f2c0bba602c6bb59757f5dd1c915d84146ecacfe6ac7
57a423e5e6b0687676416d0da105212f1e9493cf414e10f768e7891a8d4adf96e58b0ca2c2c1ce19d60
537b26493dae197cbc821ea99eaee9389e7752f08e47f9d1712a0b5584ca4d2a8929848bfacb1310565
12e47506ea5a3c7e2115c129e7
Decrypted message: Hello World,Welcome
```

```html
<!DOCTYPE html>
<html>
<head>
  <title>Diffie-Hellman Key Exchange</title>
</head>
<body>
  <h1>Diffie-Hellman Key Exchange</h1>
  <p>Enter a prime number (p) and a primitive root (g) to generate the shared secret key.</p>
  <form>
    <label for="prime">Prime number (p):</label>
    <input type="text" id="prime" name="prime"><br><br>
    <label for="root">Primitive root (g):</label>
    <input type="text" id="root" name="root"><br><br>
    <button type="button" onclick="generateKeys()">Generate Keys</button>
  </form>
  <br>
  <div id="alice"></div>
  <br>
  <div id="bob"></div>

  <script>
    function generateKeys() {

      var p = parseInt(document.getElementById("prime").value);
      var g = parseInt(document.getElementById("root").value);


      var a = Math.floor(Math.random() * (p - 1)) + 1;


      var A = Math.pow(g, a) % p;


      var aliceDiv = document.getElementById("alice");
      aliceDiv.innerHTML = "Alice's public key (A): " + A;


      var b = Math.floor(Math.random() * (p - 1)) + 1;


      var B = Math.pow(g, b) % p;


      var bobDiv = document.getElementById("bob");
      bobDiv.innerHTML = "Bob's public key (B): " + B;


      var sharedKeyAlice = Math.pow(B, a) % p;
      var sharedKeyBob = Math.pow(A, b) % p;


      aliceDiv.innerHTML += "<br>Shared secret key: " + sharedKeyAlice;
      bobDiv.innerHTML += "<br>Shared secret key: " + sharedKeyBob;
    }
  </script>
</body>
</html>
```



In [1]:

```
import hashlib

message = input("Enter the Text :").encode('utf-8')


md5 = hashlib.md5()


md5.update(message)


digest = md5.digest()


hex_digest = digest.hex()


print("Message:", message.decode('utf-8'))
print("MD5 Digest:", hex_digest)
```

```
Enter the Text :Hello World,Welcome To Pune!
Message: Hello World,Welcome To Pune!
MD5 Digest: 9061742d668c3271de66f9c9d6c80a8d
```

In [ ]: