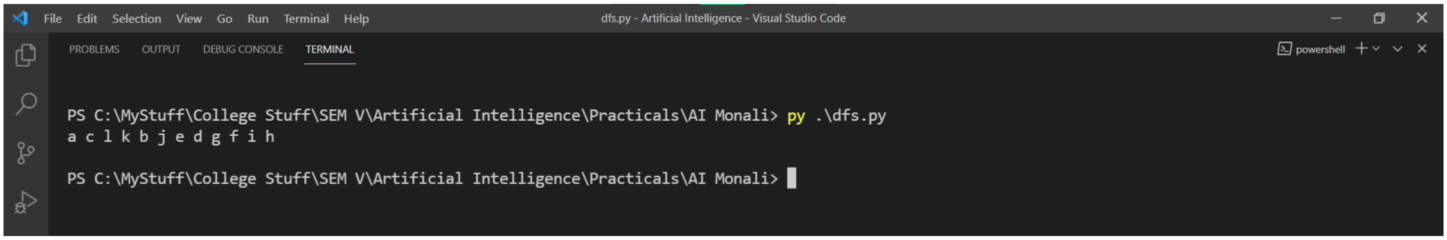
Practical 1

Q1) Demonstrate DFS Algorithm

Ans:

dfs.py

```
"""
dfs.py
Author: Jagrut Gala
Date: 10-07-2021
Practical: 1
Objective: Demonstrate DFS Algorithm
"""

def dfsRecursive(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")
    for next in graph[start] - visited:
        dfsRecursive(graph, next, visited)
    return visited

big_graph= {
    "a": set(["k", "c", "l"]),
    "b": set(["k", "j"]),
    "c": set(["a"]),
    "d": set(["k", "g"]),
    "e": set(["j"]),
    "f": set(["h", "i"]),
    "g": set(["d", "f"]),
    "h": set(["f"]),
    "i": set(["f"]),
    "j": set(["b", "e"]),
    "k": set(["a", "b", "d"]),
    "l": set(["a"]),
}

dfsRecursive(big_graph, 'a')
print("\n")
```

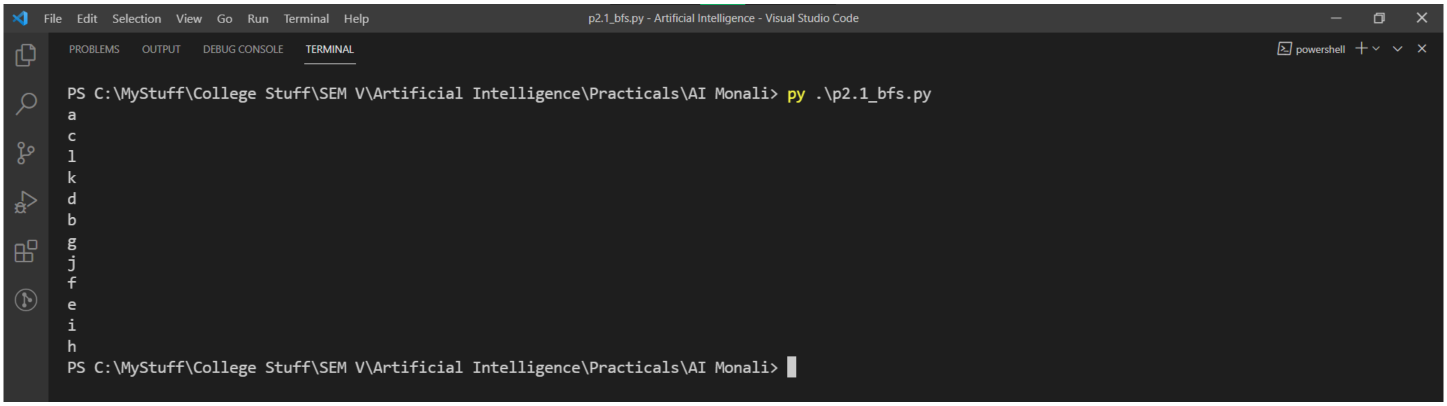

File Edit Selection View Go Run Terminal Help
dfs.py - Artificial Intelligence - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
powershell
PS C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\AI Monali> py .\dfs.py
a c l k b j e d g f i h
PS C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\AI Monali>

Practical 2

Q1) Demonstrate BFS Algorithm.
Ans:

bfs.py

```
"""
bfs.py
Author: Jagrut Gala
Date: 17-07-2021
Practical: 2
Objective: Demonstrate BFS Algorithm
"""
def bfs(visit_complete, graph, current_node):
    visit_complete.append(current_node)
    queue = []
    queue.append(current_node)

    while queue:
        s = queue.pop(0)
        print(s)

        for neighbour in graph[s]:
            if neighbour not in visit_complete:
                visit_complete.append(neighbour)
                queue.append(neighbour)

big_graph= {
    "a": set(["k", "c", "l"]),
    "b": set(["k", "j"]),
    "c": set(["a"]),
    "d": set(["k", "g"]),
    "e": set(["j"]),
    "f": set(["h", "i"]),
    "g": set(["d", "f"]),
    "h": set(["f"]),
    "i": set(["f"]),
    "j": set(["b", "e"]),
    "k": set(["a", "b", "d"]),
    "l": set(["a"]),
}

bfs([], big_graph, 'a')
```


File Edit Selection View Go Run Terminal Help
p2.1_bfs.py - Artificial Intelligence - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
powershell
PS C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\AI Monali> py .\p2.1_bfs.py
a
c
l
k
d
b
g
j
f
e
i
h
PS C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\AI Monali>

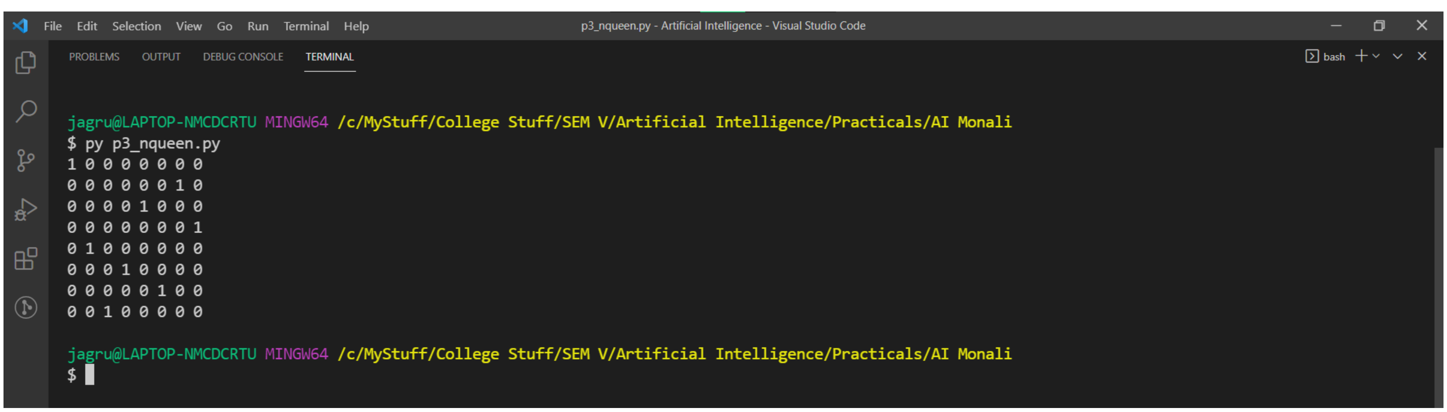
Practical 3

Q1) Demonstrate N Queens Problem and give a solution
Ans:

nqueen.py

```
"""
nqueen.py
Author: Jagrut Gala
Date: 24-07-2021
Practical: 3
Objective: Demonstrate N Queens Problem and give a solution
"""

global N
N = 8

def generateBoard(size: int) -> list:
    board= list()
    for i in range(size):
        l= []
        for j in range(size):
            l.append(0)
        board.append(l)
    return(board)

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print (board[i][j],end = " ")
        print()

def isSafe(board, row, col):
    # Check this row on left side
    for i in range(col):
        if board[row][i] == 1:
            return False
    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1),range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on left side
    for i, j in zip(range(row, N, 1),range(col, -1, -1)):
        if board[i][j] == 1:
            return False
```

```
    return True

def solveNQUtil(board, col):
    if col >= N:
        return True
    for i in range(N):
        if isSafe(board, i, col):
            # Place this queen in board[i][col]
            board[i][col] = 1
            # recur to place rest of the queens
            if solveNQUtil(board, col + 1) == True:
                return True
            board[i][col] = 0
    return False

def solveNQ():
    board = generateBoard(8)

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

# Driver Code
solveNQ()
```



Practical 4

Q1) Demonstrate the Astar Algorithm.
Ans:

p4_astar.py

```
"""
p4_astar.py
Author: Jagrut Gala
Date: 07-08-2021
Practical: 4
Objective: Demonstrate Astar Algorithm
"""

class Node():
    """A node class for A* Pathfinding"""

    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position


def astar(maze, start, end):
    """Returns a list of tuples as a path from the given start to the given end
in the given maze"""

    # Create start and end node
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0

    # Initialize both open and closed list
    open_list = []
    closed_list = []

    # Add the start node
    open_list.append(start_node)
```

```
    # Loop until you find the end
    while len(open_list) > 0:

        # Get the current node
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index

        # Pop current off open list, add to closed list
        open_list.pop(current_index)
        closed_list.append(current_node)

        # Found the goal
        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
            return path[::-1] # Return reversed path

        # Generate children
        children = []
        for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1),
(1, -1), (1, 1)]: # Adjacent squares

            # Get node position
            node_position = (current_node.position[0] + new_position[0],
current_node.position[1] + new_position[1])

            # Make sure within range
            if node_position[0] > (len(maze) - 1)\
            or node_position[0] < 0\
            or node_position[1] > (len(maze[len(maze)-1]) -1)\
            or node_position[1] < 0:
                continue

            # Make sure walkable terrain
            if maze[node_position[0]][node_position[1]] != 0:
                continue

            # Create new node
```

```
            new_node = Node(current_node, node_position)

            # Append
            children.append(new_node)

        # Loop through children
        for child in children:

            # Child is on the closed list
            for closed_child in closed_list:
                if child == closed_child:
                    continue

            # Create the f, g, and h values
            child.g = current_node.g + 1
            child.h = ((child.position[0] - end_node.position[0]) ** 2) +
((child.position[1] - end_node.position[1]) ** 2)
            child.f = child.g + child.h

            # Child is already in the open list
            for open_node in open_list:
                if child == open_node and child.g > open_node.g:
                    continue

            # Add the child to the open list
            open_list.append(child)


def main():

    maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

    start = (0, 0)
    end = (7, 6)

    path = astar(maze, start, end)
```

```
    print(path)


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

```
C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_4>py p4_astar.py
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 4), (6, 5), (7, 6)]

C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_4>
```

Practical 5

Q1) Demonstrate Hill Climbing Technique.

Ans:

p5_hill_climbing.py

```
"""
p5_hill_climbing.py
Author: Jagrut Gala
Date: 14-08-2021
Practical: 5
Objective: Demonstrate Hill Climbing Technique
"""

import math
increment = 0.5
startingPoint = [1, 1]
point1 = [1,7]
point2 = [6,4]
point3 = [5,2]
point4 = [3,1]

def distance(x1, y1, x2, y2):
    dist = math.pow(x2-x1, 2) + math.pow(y2-y1, 2)
    return dist

def sumOfDistances(x1, y1, px1, py1, px2, py2, px3, py3, px4, py4):
    d1 = distance(x1, y1, px1, py1)
    d2 = distance(x1, y1, px2, py2)
    d3 = distance(x1, y1, px3, py3)
    d4 = distance(x1, y1, px4, py4)
    return d1 + d2 + d3 + d4

def newDistance(x1, y1, point1, point2, point3, point4):
    d1 = [x1, y1]
    d1temp = sumOfDistances(x1, y1, point1[0], point1[1], point2[0], point2[1],
point3[0], point3[1], point4[0], point4[1])
    d1.append(d1temp)
    return d1

def newPoints(minimum, d1, d2, d3, d4):
    if d1[2] == minimum:
        return [d1[0], d1[1]]
    elif d2[2] == minimum:
        return [d2[0], d2[1]]
    elif d3[2] == minimum:
```

```
        return [d3[0], d3[1]]
    elif d4[2] == minimum:
        return [d4[0], d4[1]]

minDistance = sumOfDistances(
    startingPoint[0], startingPoint[1],
    point1[0], point1[1], point2[0], point2[1],
    point3[0], point3[1], point4[0], point4[1]
    )
flag = True
i = 1
while flag:
    d1 = newDistance(startingPoint[0]+increment, startingPoint[1],
    point1, point2, point3, point4)
    d2 = newDistance(startingPoint[0]-increment, startingPoint[1],
    point1, point2, point3, point4)
    d3 = newDistance(startingPoint[0], startingPoint[1]+increment,
    point1, point2, point3, point4)
    d4 = newDistance(startingPoint[0], startingPoint[1]-increment,
    point1, point2, point3, point4)
    print (i,' ', round(startingPoint[0], 2), round(startingPoint[1], 2))
    minimum = min(d1[2], d2[2], d3[2], d4[2])
    if minimum < minDistance:
        startingPoint = newPoints(minimum, d1, d2, d3, d4)
        minDistance = minimum
    #print i,' ', round(startingPoint[0], 2), round(startingPoint[1], 2)
        i+=1
    else:
        flag = False
```

```
C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_5>py p5_hill_climbing.py
1   1 1
2   1.5 1
3   1.5 1.5
4   2.0 1.5
5   2.0 2.0
6   2.5 2.0
7   2.5 2.5
8   3.0 2.5
9   3.0 3.0
10   3.5 3.0
11   3.5 3.5

C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_5>
```

Practical 6

Q1) Predict the price of a house using Linear Regression.

Ans:

p6_linear_regression.py

```
"""
p6_linear_regression.py
Author: Jagrut Gala
Date: 28-08-2021
Practical: 6
Objective: Predict the price of a house  using Linear Regression.
"""

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
import pandas as pd
import io
from pathlib import Path

p= Path(__file__).parent/ "Housing.xlsx"
fio= io.open(p, "rb")
df = pd.read_excel(fio)
print(df)
Y = np.array(df['price']).reshape(1, -1)
X = np.array(df['tsft']).reshape(1, -1)
# print(f"Shapes: {X.shape} {Y.shape}")
# # Plot outputs
plt.scatter(X, Y)
plt.title('Test Data')
plt.xlabel('Size')
plt.ylabel('Price')
plt.xticks(())
plt.yticks(())
# # Create linear regression object
regr = linear_model.LinearRegression()
# # Train the model using the training sets
regr.fit(X, Y)
# # Plot outputs
```

```
plt.plot(X, regr.predict(X), color='red',linewidth=3)
plt.show()
```

```
(ai) C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_6>py p6_linear_regression.py
      price  tsft
0   5000000   800
1   6000000   900
2   7000000  1000
3   8000000  1100
4   9000000  1200
5  10000000  1300
6  11000000  1400
7  12000000  1500
8  13000000  1600
9  14000000  1700
```

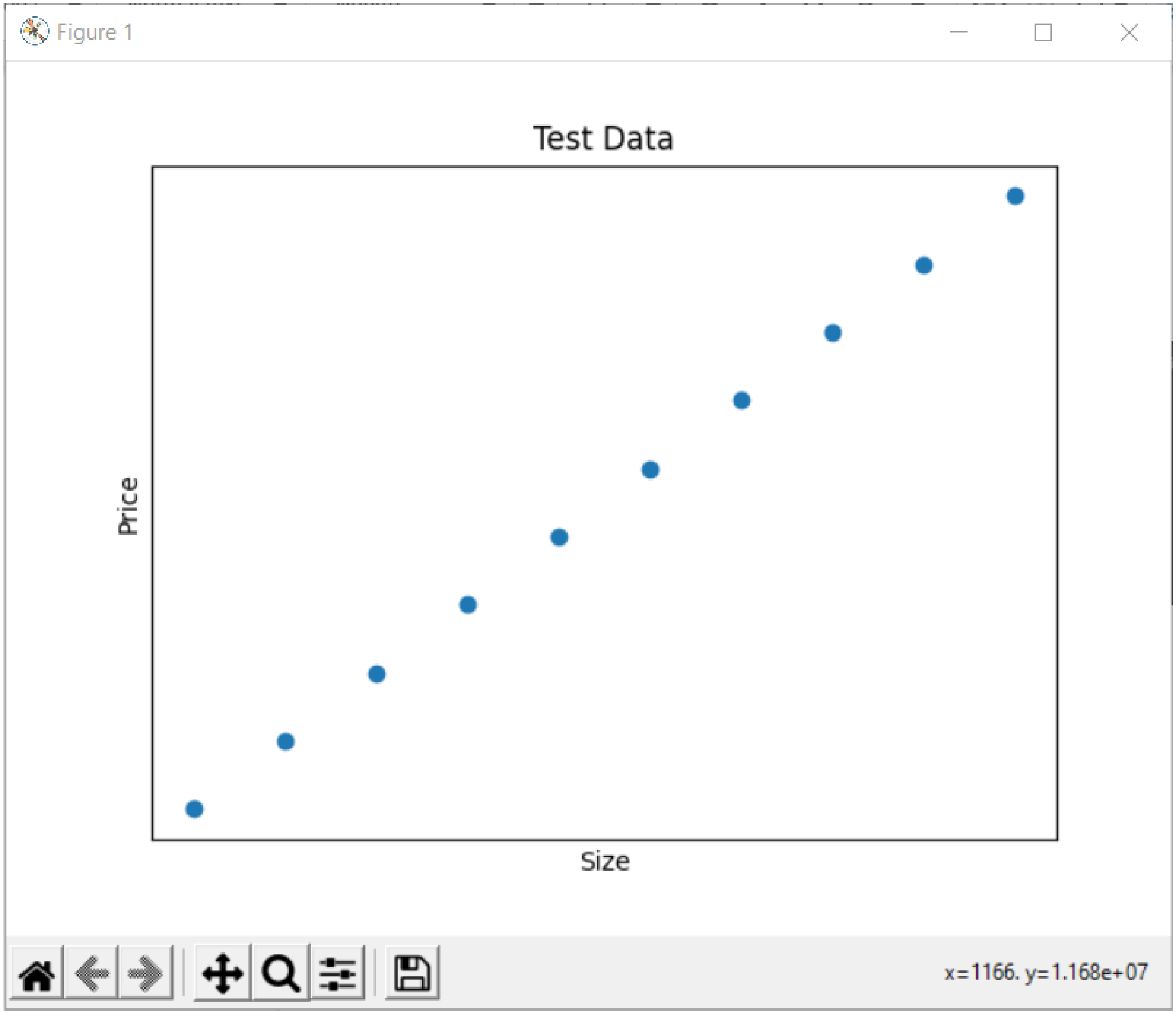


Figure 1
Test Data
Price
Size
x=1166. y=1.168e+07

Practical 7

Q1) Demonstrate Tower of Hanoi Problem.

Ans:

p7_tower_of_hanoi.py

```
"""
p7_tower_of_hanoi.py
Author: Jagrut Gala
Date: 04-09-2021
Practical: 7
Objective: Demonstrate Tower of Hanoi Problem.
"""

def TowerOfHanoi(n, from_rod, to_rod, aux_rod): # f:A t:C x:B
    if n == 1:
        print ("Move disk 1 from rod",from_rod,"to rod",to_rod)
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print ("Move disk",n,"from rod",from_rod,"to rod",to_rod)
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)

# Driver code
n = 4 # n is number of disks
# A B C are rods
print(f"Jagrut Tower of Hanoi")
TowerOfHanoi(n, 'A', 'C', 'B')
```

```
(ai) C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_7>py p7_tower_of_hanoi.py
Jagrut Tower of Hanoi
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C

(ai) C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_7>
```

Practical 8

Q1) Demonstrate Travelling Salesman Problem.
Ans:
p8_travelling_salesman.py

```
"""
p8_travelling_salesman.py
Author: Jagrut Gala
Date: 04-09-2021
Practical: 8
Objective: Demonstrate Travelling Salesman Problem.
"""



# Python3 program to implement traveling salesman
# problem using naive approach.
from sys import maxsize
from itertools import permutations
V = 4
# implementation of traveling Salesman Problem
def travellingSalesmanProblem(graph, s):
# store all vertex apart from source vertex
    vertex = []
    for i in range(V):
        if(i == s): continue
        vertex.append(i)
    # store minimum weight Hamiltonian Cycle
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
        current_pathweight = 0 # store current Path weight(cost)
        k = s # compute current path weight
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight) # update minimum
    return min_path
```

```
# Driver Code
if __name__ == "__main__":
    # matrix representation of graph
    graph = [
        [0, 10, 15, 20],
        [10, 0, 35, 25],
        [15, 35, 0, 30],
        [20, 25, 30, 0],
    ]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

```
(ai) C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_8>py p8_travelling_salesman.py
80

(ai) C:\MyStuff\College Stuff\SEM V\Artificial Intelligence\Practicals\practical_8>
```

Practical 9