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"]),

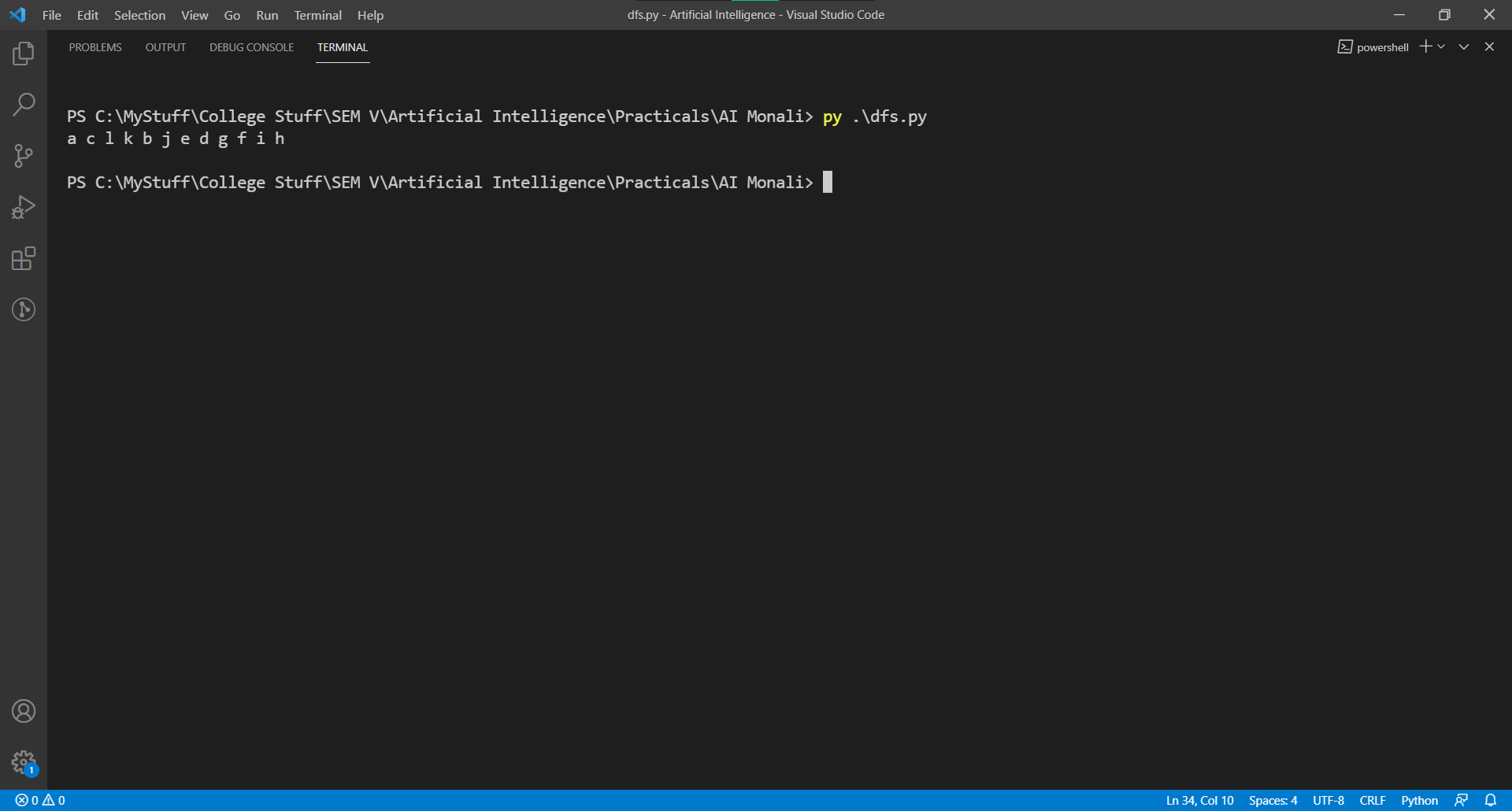
"k": set(["a", "b", "d"]),

"l": set(["a"]),

}

dfsRecursive(big\_graph, 'a')

print("\n")



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"]),

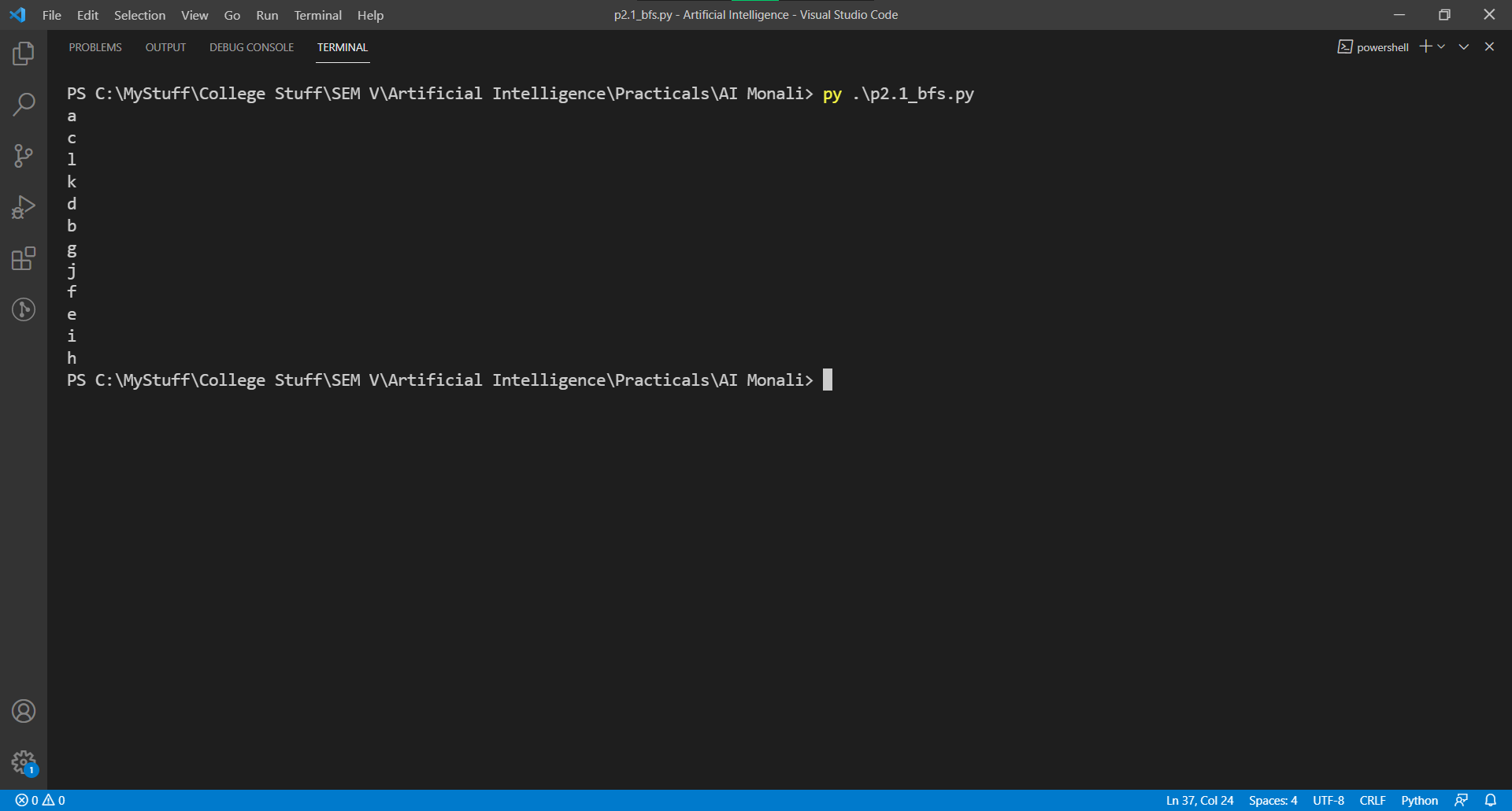
"j": set(["b", "e"]),

"k": set(["a", "b", "d"]),

"l": set(["a"]),

}

bfs([], big\_graph, 'a')



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")

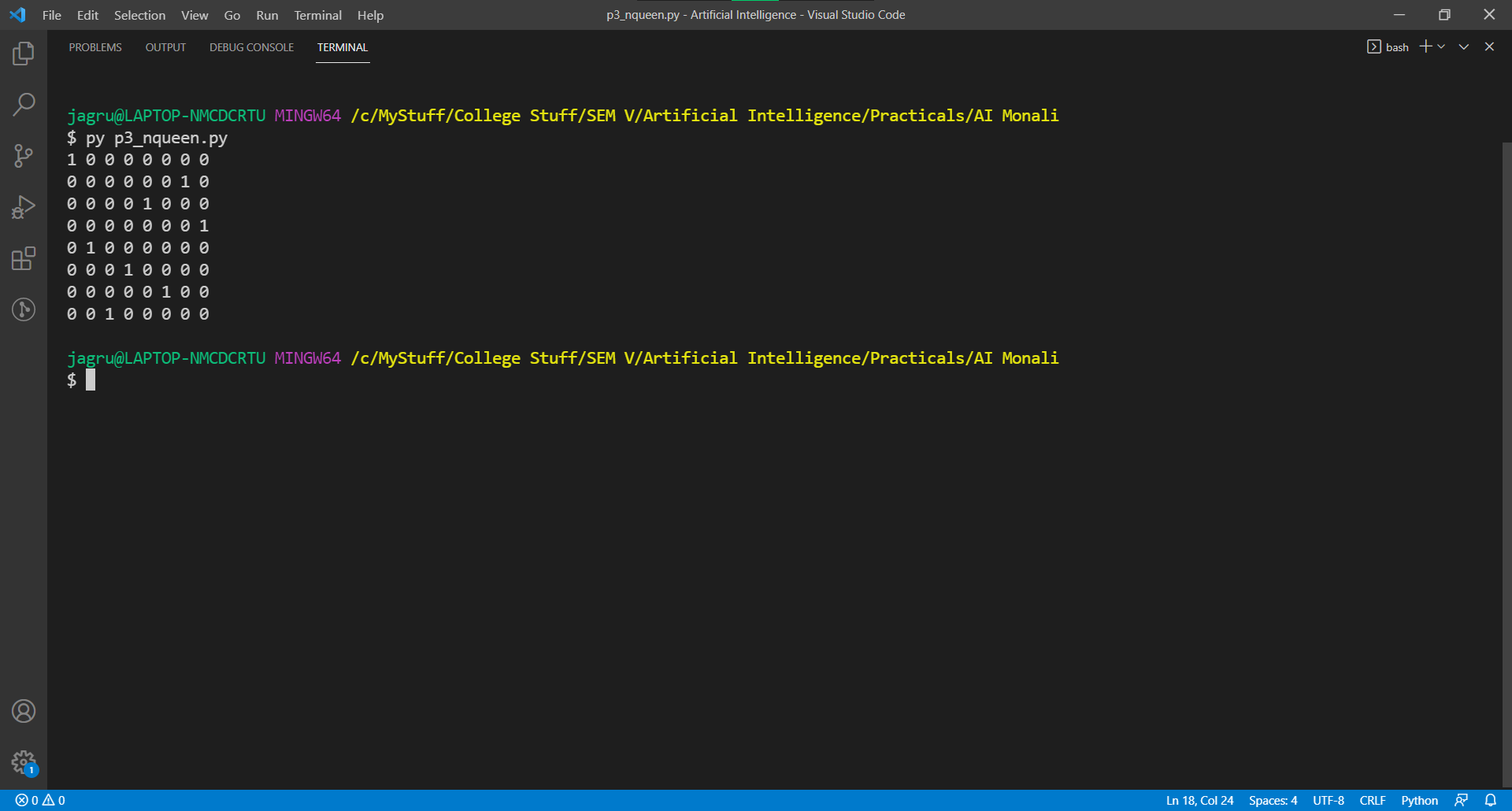
return False

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)

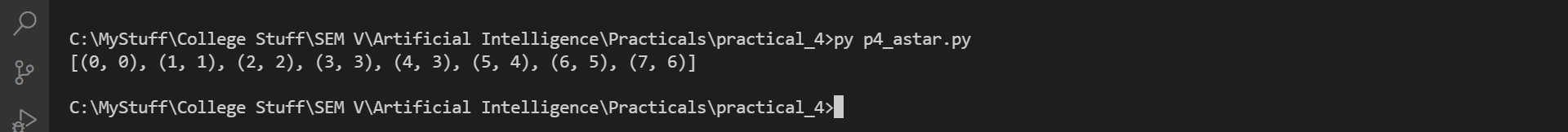
end = (7, 6)

path = astar(maze, start, end)

print(path)

if \_\_name\_\_ == '\_\_main\_\_':

main()



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)

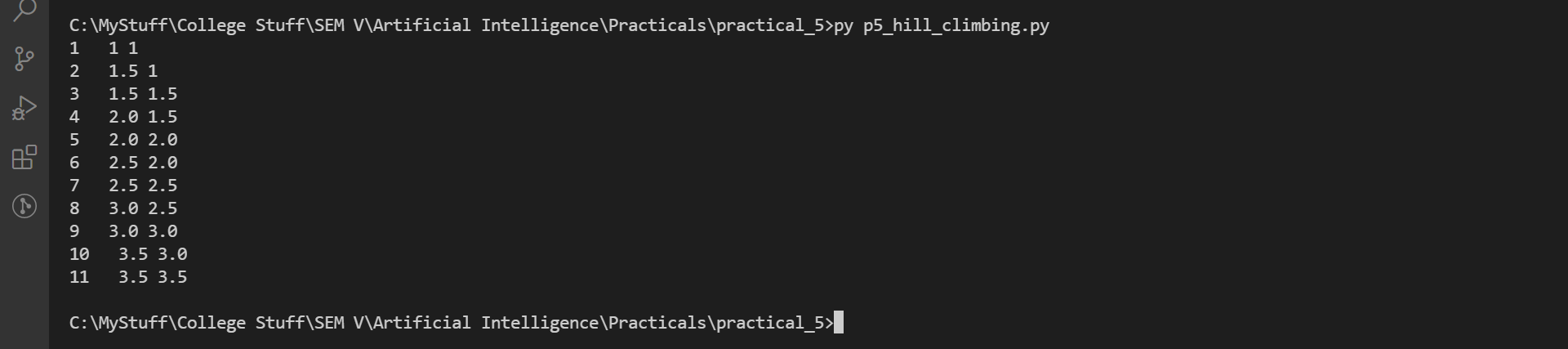
minDistance = minimum

#print i,' ', round(startingPoint[0], 2), round(startingPoint[1], 2)

i+=1

else:

flag = False



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()

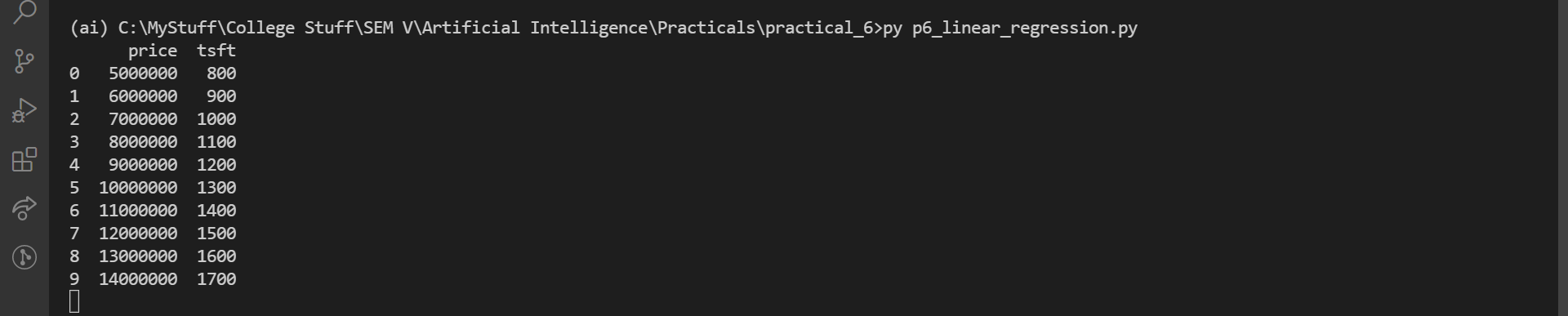
# # Train the model using the training sets

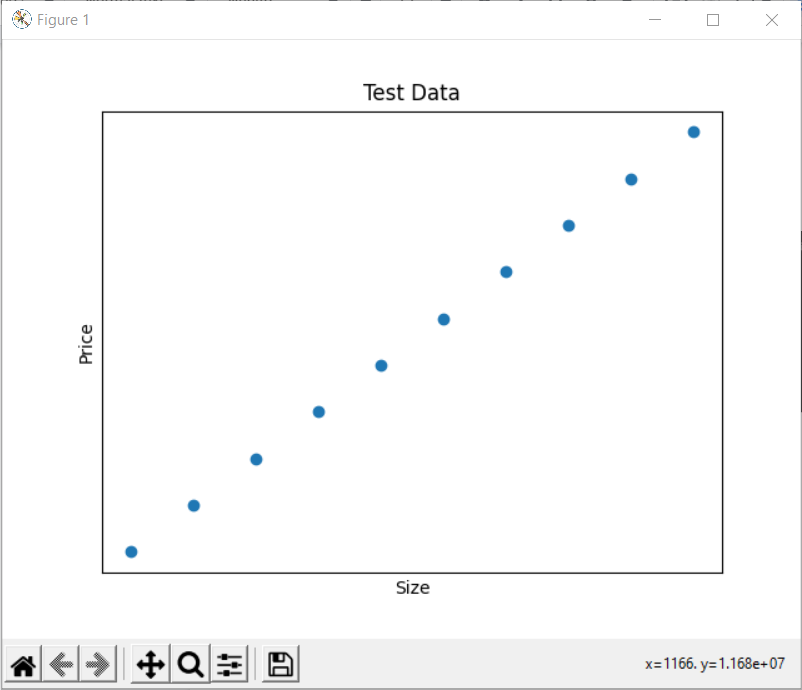
regr.fit(X, Y)

# # Plot outputs

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

plt.show()





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)

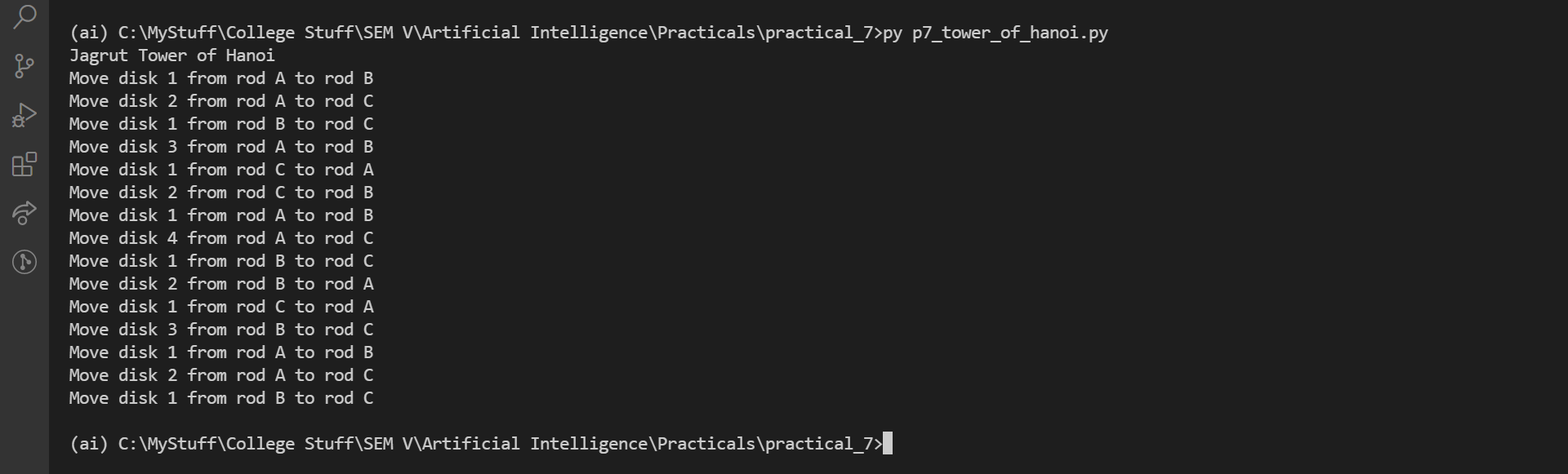
# Driver code

n = 4 # n is number of disks

# A B C are rods

print(f"Jagrut Tower of Hanoi")

TowerOfHanoi(n, 'A', 'C', 'B')



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],

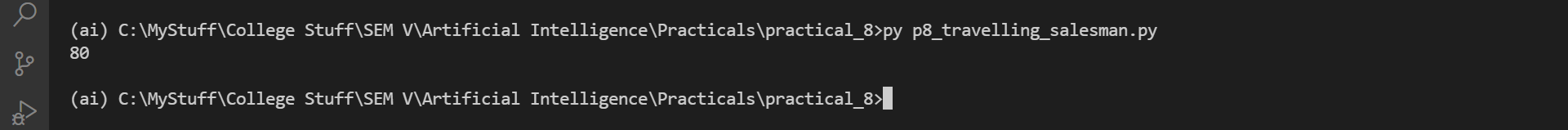
[15, 35, 0, 30],

[20, 25, 30, 0],

]

s = 0

print(travellingSalesmanProblem(graph, s))



Practical 9