**IMPORTANT**

**INSTRUCTIONS**

:



**PAF**

**-**

**Karachi**

**Institute**

**of**

**Economics**

**and**

**Technology**

**College**

**of**

**Computing**

**and**

**Information**

**Sciences**

**Artificial Inte**

**lligence**

**Assignment**

**-**

**1**

**NAME: SARIM AMIR**

**SID: 63686**

Read the following Instructions carefully:

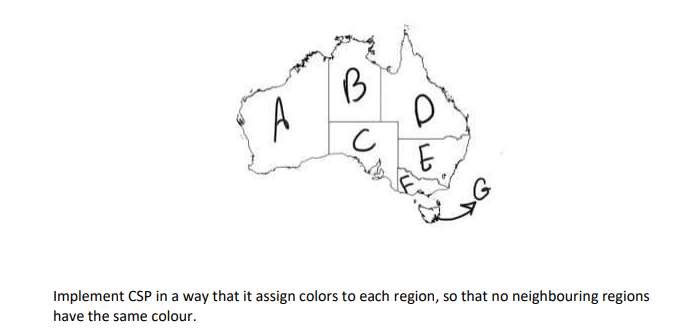
* Attempt all question using python.
* After attempting all question, add screenshot of each question’s solution with output.
* Must attempt all questions by yourself, in case of copied solutions your assignment will marked ZERO.
* Arrange questions and their subsequent parts in sequence.
* Submit on LMS and GCR before deadline.

**Save your word document with name: [Student Name Student ID]**

# Submission Deadline: 28/03/2022

**[5 mark]**

1.



colors = ['Red', 'Green','Blue']

states = ['A','B','C','D','E','F','G']

colors\_of\_states = {}

neighbours = {}

neighbours['A'] = ['B','C']

neighbours['B'] = ['A','C','D']

neighbours['C'] = ['A','B','D','E','F']

neighbours['D'] = ['B','C','E']

neighbours['E'] = ['C','D','F']

neighbours['F'] = ['C','E']

neighbours['G'] = []

def promising(state,color):

for neighbour in neighbours.get(state):

color\_of\_neighbour = colors\_of\_states.get(neighbour)

if color\_of\_neighbour == color:

return False

return True

def colorforstates(state):

for color in colors:

if promising(state,color):

return color

return None

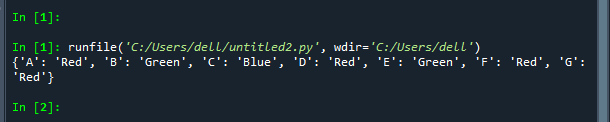
if \_\_name\_\_ == '\_\_main\_\_':

for state in states:

colors\_of\_states[state] = colorforstates(state)

print(colors\_of\_states)

**OUTPUT:**



P a g e 1 | 2 **Save your word document with name: [Student Name Student ID]**

# Submission Deadline: 28/03/2022

[3 mark]

2. Implement simple reflex agent in which , the environment of Lab 1 and Lab 2 will be clean or dirty and the agent will done the action according to the given instruction.

import random

initiallocation={"Lab 1":"0","Lab 2":"0"}

initiallocation["Lab 1"]=random.randint(0,1)

initiallocation["Lab 2"]=random.randint(0,1)

print("Locations are initially:")

print(initiallocation)

vacuumlocation=random.randint(0, 1)

if vacuumlocation==0:

print("Vacuum is initially located at Lab 1")

if initiallocation["Lab 1"]==1:

print("Lab 1 is dirty")

initiallocation["Lab 1"]==0

print("Lab 1 has been cleaned")

if initiallocation["Lab 2"]==1:

print("Lab 2 is dirty")

print("Vacuum moving to Lab 2")

initiallocation["Lab 2"]==0

print("Lab 2 has been cleaned")

else:

print("Lab 2 is dirty")

print("Vacuum moving to Lab 2")

initiallocation["Lab 2"]==0

print("Lab 2 has been cleaned")

else:

print("Vacuum is initially located at Lab 2")

if initiallocation["Lab 2"]==1:

print("Lab 2 is dirty")

initiallocation["Lab 2"]==0

print("Lab 2 has been cleaned")

if initiallocation["Lab 1"]==1:

print("Lab 1 is dirty")

print("Vacuum moving to Lab 1")

initiallocation["Lab 1"]==0

print("Lab 1 has been cleaned")

else:

print("Vacuum moving to Lab 1")

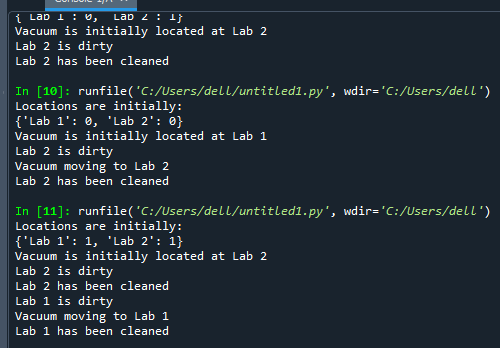
if initiallocation["Lab 1"]==1:

print("Lab 1 is dirty")

initiallocation["Lab 1"]==0

print("Lab 1 has been cleaned")

**OUTPUT:**

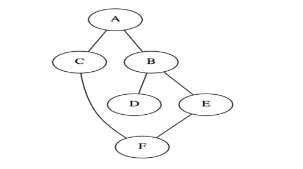


**Save your word document with name: [Student Name Student ID]**

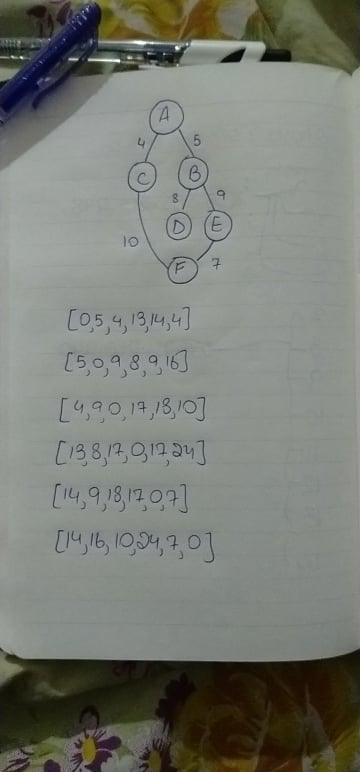
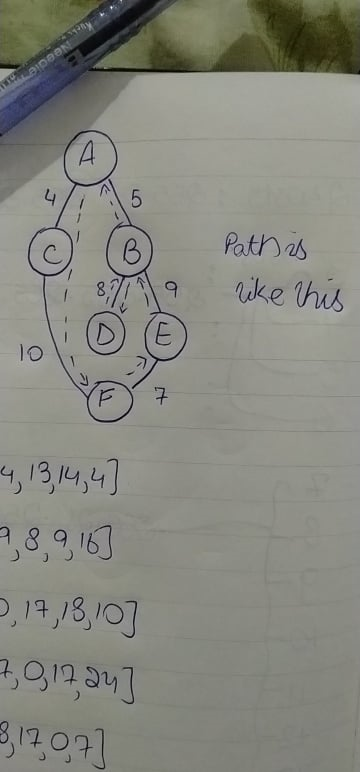
# Submission Deadline: 28/03/2022

3. Implement the following graph.

## Note: [If Student Id last digit is even perform Travelling Sales Man Problem, If Student Id last digit is odd perform Hill Climbing Algorithm.]



[2 mark]

 **Done Using TSP and assume the cost by myself**

from itertools import permutations

from sys import maxsize

graph = [[0,5,4,13,14,14], [5,0,9,8,9,16],

[4,9,0,17,18,10], [13,8,17,0,17,24],[14,9,18,17,0,7],[14,16,10,24,7,0]]

a=0

vertex = []

for i in range(6):

if i != a:

vertex.append(i)

minimumpath = maxsize

next\_permutation=permutations(vertex)

# in above line permuatations(vertex) is returning all possible permutations of length vertex

for i in next\_permutation:

current\_pathweight = 0

k = a

for j in i:

current\_pathweight += graph[k][j]

k = j

current\_pathweight += graph[k][a]

minimumpath = min(minimumpath, current\_pathweight)

print("Cost of shortest route is: ",minimumpath)

**OUTPUT:**

