SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-2

REGISTRATION NUMBER	**RA1811028010049**
NAME	**SHUSHRUT KUMAR**

B.Tech-CSE-CC, Third Year (Section: J2)

Faculty Incharge: Vaishnavi Moorthy, Asst Prof/Dept of CSE

Year 2020-2021

INDEX

Ex No	DATE	Title	Page No	Marks
1	21-01-2020	Toy Problem:Selling movie tickets problem	3	
2	29-01-2020	Graph Coloring Problem	7	

Exercise: 1

Date : 21-01-2020

TOY PROBLEM

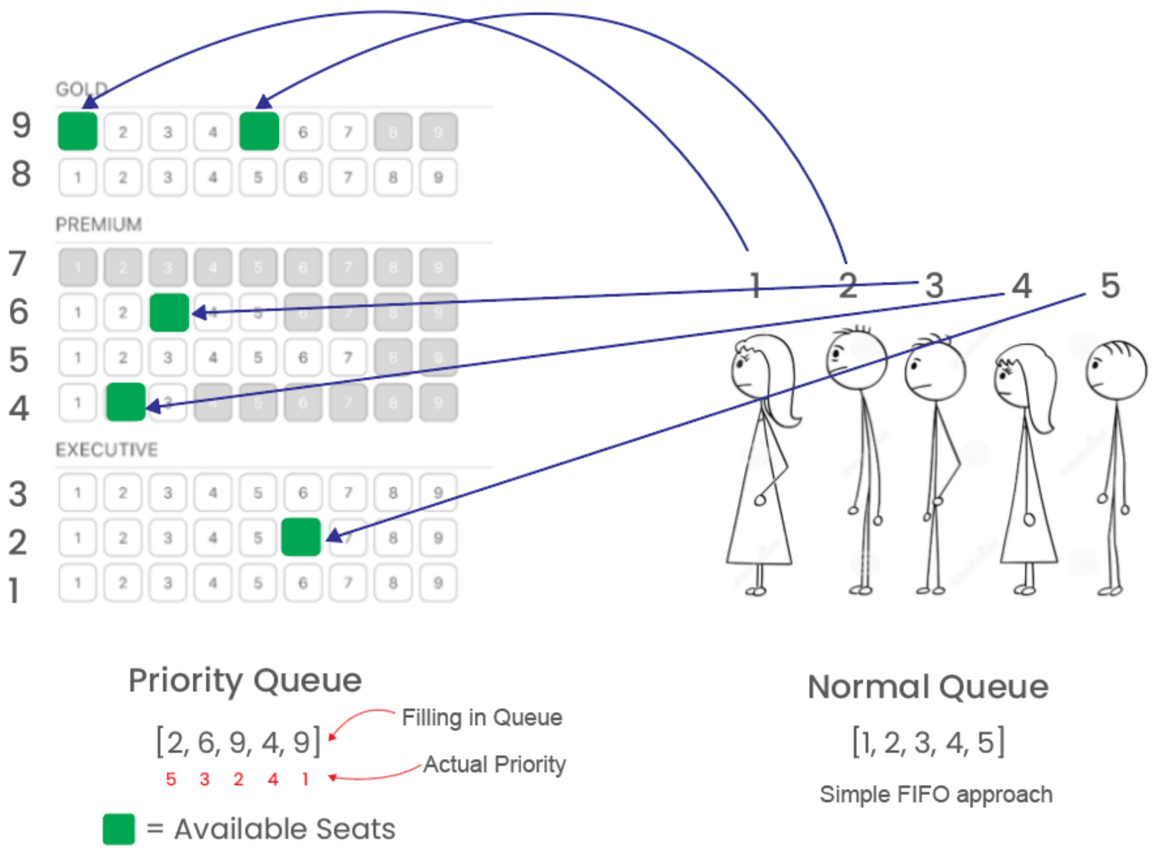
Problem Statement : Given an integer N and an array of seats[] where N is the number of people standing in a line to buy a movie ticket and seat[i] is the number of empty seats in the ith row of the movie theater. The task is to find the maximum amount a theater owner can make by selling movie tickets to N people. Price of a ticket is equal to the maximum number of empty seats among all the rows.

Algorithm :

1. Initialize queue q insert all seats array elements to the queue.
2. Tickets sold and the amount generated to be set to 0.
3. If tickets sold < N (People in the queue) and q top > 0
4. Then remove top element from queue and update total amount
5. Repeat step 3 and 4 until tickets sold = number of people in the queue.

Optimization technique : This problem can be solved by using a priority queue that will store the count of empty seats for every row and the maximum among them will be available at the top.

1. Create an empty priority_queue q and traverse the seats[] array and insert all elements into the priority_queue.
2. Initialize two integer variable ticketSold = 0 and ans = 0 that will store the number of tickets sold and the total collection of the amount so far.
3. Now check while ticketSold < N and q.top() > 0 then remove the top element from the priority_queue and update ans by adding top element of the priority queue. Also store this top value in a variable temp and insert temp – 1 back to the priority_queue.
4. Repeat these steps until all the people have been sold the tickets and print the final result.

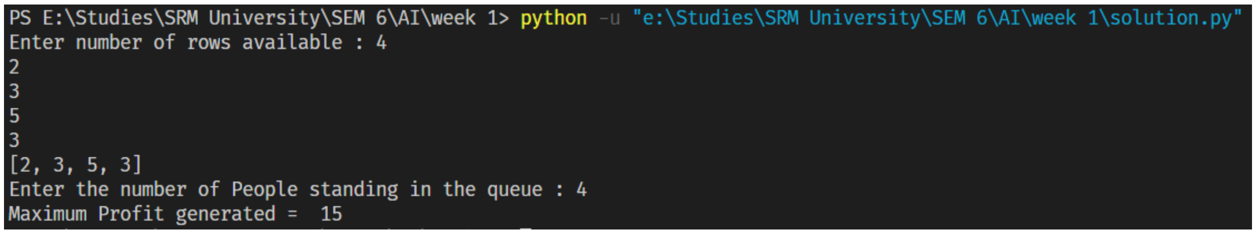


Tool : VS Code and Python 3.9.0

Programming code :

```
def maxAmount(M, N, seats):

    q = []

    for i in range(M):
        q.append(seats[i])
    ticketSold = 0

    ans = 0

    q.sort(reverse = True)
    while (ticketSold < N and q[0] > 0):
        ans = ans + q[0]
        temp = q[0]
        q = q[1:]
        q.append(temp - 1)
        q.sort(reverse = True)
        ticketSold += 1

    return ans

if __name__ == '__main__':

    seats = []

    rows = int(input("Enter number of rows available : "))

    for i in range(0, rows):
        empty = int(input())
        seats.append(empty)


    print(seats)

    M = len(seats)

    N = int(input("Enter the number of People standing in the queue : "))

    print("Maximum Profit generated = ", maxAmount(N, M, seats))
```

Output screen shots :



```
PS E:\Studies\SRM University\SEM 6\AI\week 1> python -u "e:\Studies\SRM University\SEM 6\AI\week 1\solution.py"
Enter number of rows available : 4
2
3
5
3
[2, 3, 5, 3]
Enter the number of People standing in the queue : 4
Maximum Profit generated =  15
```

Result : Successfully found out the maximum amount the theater owner can make by selling movie tickets to N people for a movie.

Exercise: 2

Date : 29-01-2020

GRAPH COLORING PROBLEM

PROBLEM STATEMENT : Given a graph color its edges such that no two adjacent have the same color using minimum number of colors and return the Chromatic number.

ALGORITHM :

Initialize:

1. Color first vertex with first color.

Loop for remaining V-1 vertices.:

1. Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it.

2. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

3. Repeat the following for all edges.

4. Index of color used is the chromatic number.

OPTIMIZATION TECHNIQUE:

Graph coloring problem is to assign colors to certain elements of a graph subject to certain constraints.

Vertex coloring is the most common graph coloring problem. The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using the same color. The other graph coloring problems like Edge Coloring (No vertex is incident to two edges of same color) and Face Coloring (Geographical Map Coloring) can be transformed into vertex coloring.

Chromatic Number: The smallest number of colors needed to color a graph G is called its chromatic number. For example, the following can be colored at least 2 colors.

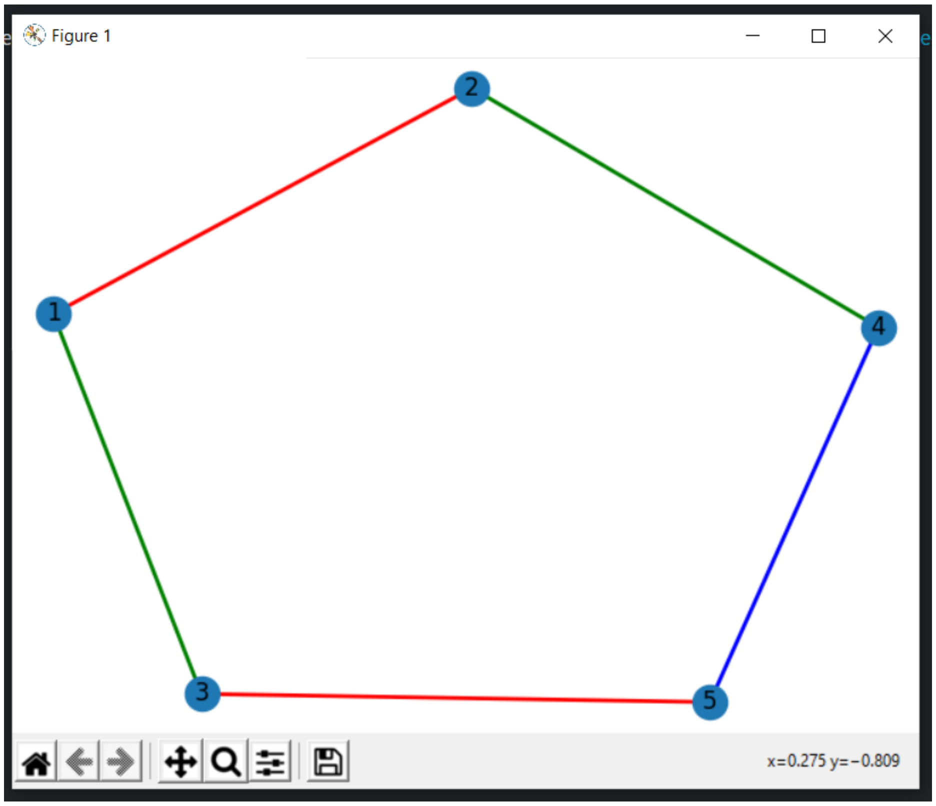
CODE - EDGE COLORING :

```
import matplotlib.pyplot as plt

import networkx as nx

from matplotlib.patches import Polygon

import numpy as np


G = nx.Graph()


colors = {0:"red", 1:"green", 2:"blue", 3:"yellow"}


G.add_nodes_from([1,2,3,4,5])

G.add_edges_from([(1,2), (1,3), (2,4), (3,5), (4,5)])


nodes = list(G.nodes)

edges = list(G.edges)
```

```
color_lists = []

color_of_edge = []

some_colors = ['red','green','blue','yellow']


for i in range(len(nodes) + 1):

    color_lists.append([])

    color_of_edge.append(-1)


def getSmallestColor(ls1,ls2):

    i = 1

    while(i in ls1 or i in ls2):

        i = i + 1

    return i


#iterate over edges

i = 0

for ed in edges:

    newColor = getSmallestColor(color_lists[ed[0]],color_lists[ed[1]])

    color_lists[ed[0]].append(newColor)

    color_lists[ed[1]].append(newColor)

    color_of_edge[i] = newColor

    i = i + 1
```

```
# Makin graph again

G = nx.Graph()


for i in range(len(edges)):

    G.add_edge(edges[i][0],edges[i][1],color=some_colors[color_of_edge[i]-1])


colors = nx.get_edge_attributes(G,'color').values()

nx.draw(G, edge_color=colors, with_labels=True, width=2)


plt.show()
```

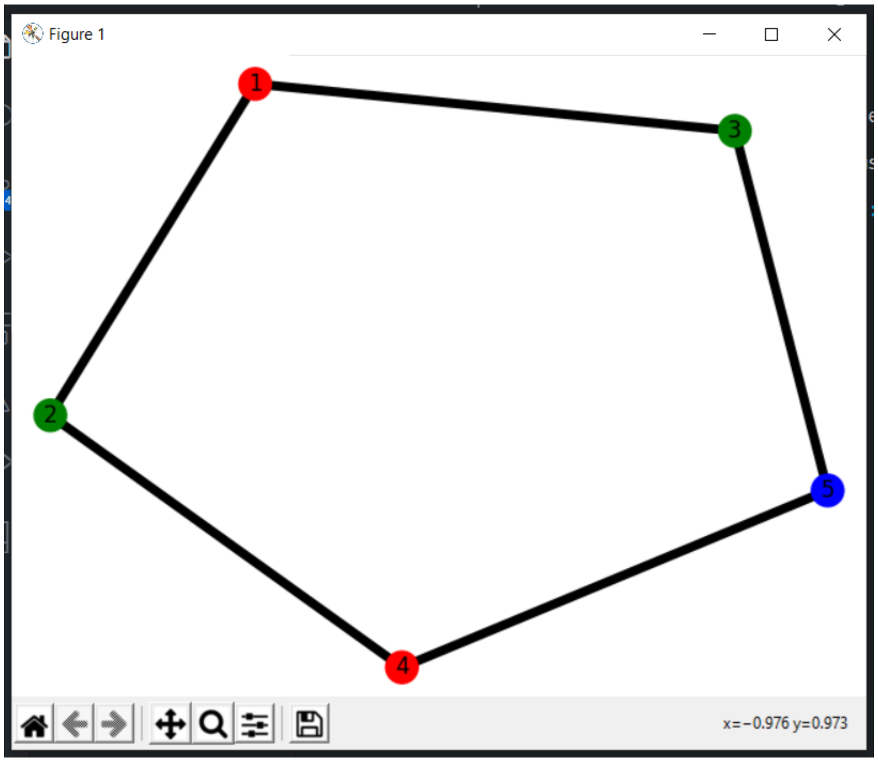
OUTPUT :



CODE - VERTEX COLORING :

```
import matplotlib.pyplot as plt

import networkx as nx


G = nx.Graph()


colors = {0:"red", 1:"green", 2:"blue"}


G.add_nodes_from([1,2,3,4,5])

G.add_edges_from([(1,2), (1,3), (2,4), (3,5), (4,5)])


d = nx.coloring.greedy_color(G, strategy = "largest_first")

node_colors = []

for i in sorted (d.keys()):

    node_colors.append(colors[d[i]])


nx.draw(G, node_color = node_colors, with_labels = True, width = 5)


plt.show()
```

OUTPUT :

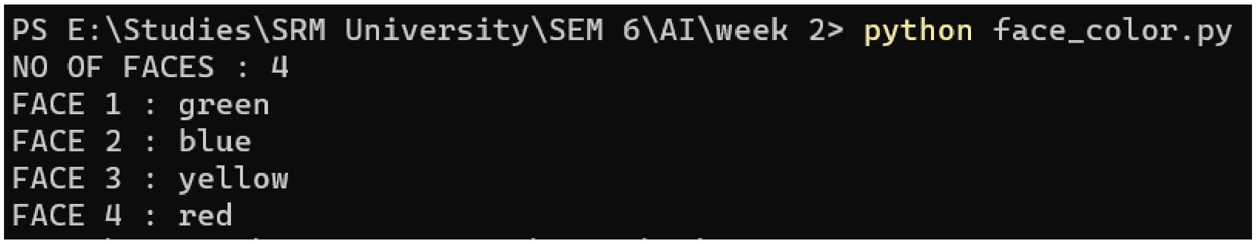


CODE - FACE COLORING :

```
import networkx as nx

G = nx.Graph()

colors = {0:"red", 1:"green", 2:"blue", 3:"yellow"}

G.add_nodes_from([1,2,3,4,5])

G.add_edges_from([(1,2), (1,3), (2,4), (3,4), (4,5)])

nodes = list(G.nodes)

edges = list(G.edges)

some_colors = ['red','green','blue','yellow']

no_of_faces = len(edges)+2-len(nodes)-1
```

```
def regionColour(regions):
    print("NO OF FACES : "+str(regions))
    for i in range(1,regions+1):
        print("FACE 1 : "+some_colors[i%4])
regionColour(no_of_faces)
```

OUTPUT :



```
PS E:\Studies\SRM University\SEM 6\AI\week 2> python face_color.py
NO OF FACES : 4
FACE 1 : green
FACE 2 : blue
FACE 3 : yellow
FACE 4 : red
```

RESULT :

Edge, vertex and face coloring problem which are together known as graph coloring problem solved and visualized in an optimized way using greedy approach.