Index

S. No.	Name of the Experiment	Page No.	Date of Experiment	Date of Submission	Remarks
1.	Solve "Tower of Hanoi" with only 3 disks.		14/11/22		
2.	Solve "8-Queens" puzzle.		21/11/22		
3.	Solve Travelling Salesman Problem		5/12/22		
4.	Solve "4-color Map" problem		12/12/22		

Expt. No. 1

Page No. 01

Date

Experiment - 1.

* Solve "Tower of Hanoi" with only 3 disks.

```
# Recursive function to Solve tower of Hanoi.
def TowerOfHanoi ( n, from_rod, to_rod, aux_rod ):
    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)

TowerOfHanoi (3, 'A', 'C', 'B')
```

Teacher's Signature : .....

Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C.

2
Expt. No.

02
Page No.

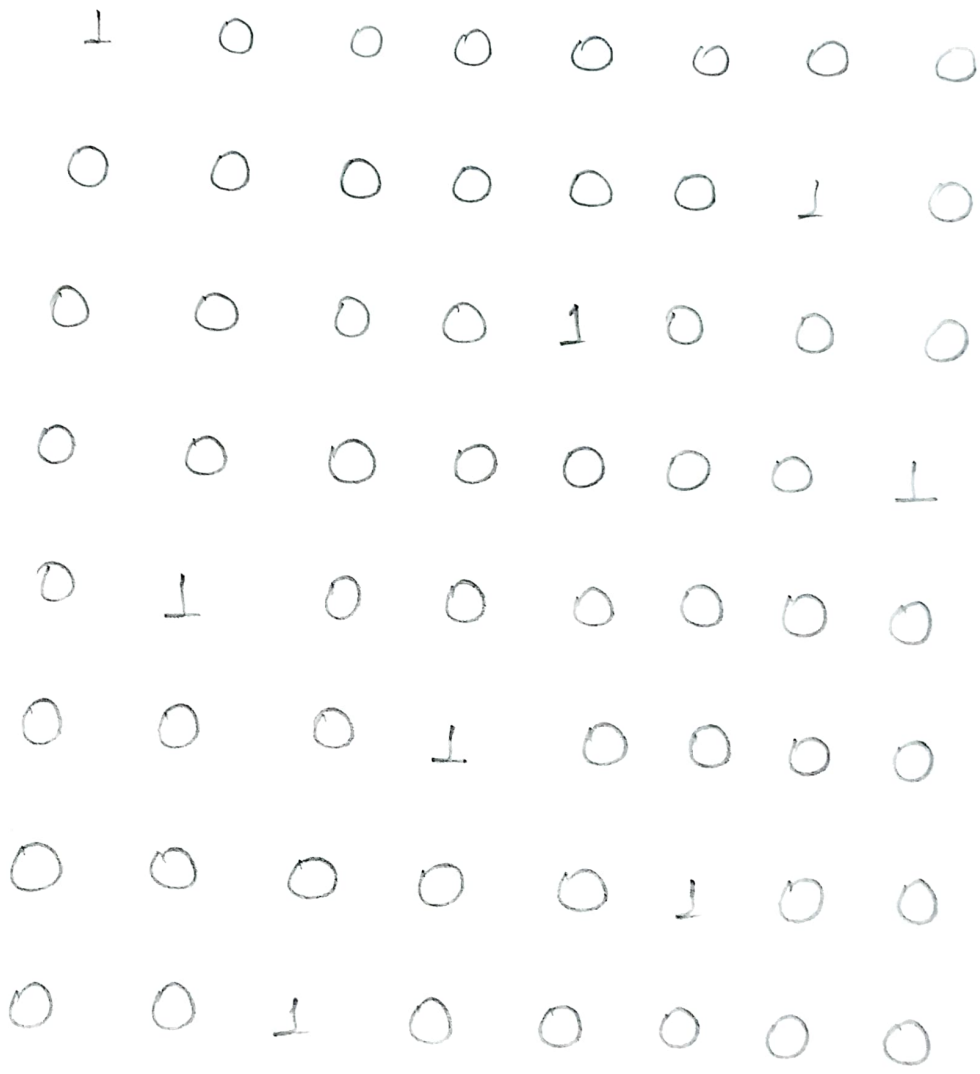
Date

Experiment - 2

* Solve "8-Queens" puzzle.

```
global N
N=8

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

def isSafe(board, row, col):

    for i in range(col):
        if(board[row][i] == 1):
            return False

    for i,j in zip(range(row,-1,-1), range(col,-1,-1)):
        if(board[i][j] == 1):
            return False

    for i,j in zip(range(row,N,1), range(col,-1,-1)):
        if(board[i][j] == 1):
            return False

    return True
```

Teacher's Signature : .....



Expt. No. .....

Page No. 3

Date .....

```
def solveNQUtil(board, col):
    if (col >= N):
        return True

    for i in range(N):
        if (isSafe(board, i, col)):
            board[i][col] = 1
        if(solveNQUtil(board, col+1) == True):
            return True
        board[i][col] = 0

    return False

def solveNQ():

    board = [ [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0] ]
```

Teacher's Signature : .....

Expt. No. .....

Page No. 4

Date .....

```
if (solveNQUntil(board, 0) == False):
        Print ("Solution doesnot exist")
        return False

printSolution(board)
return True

SolveNQ()
```

3
Expt. No.

4
Page No.

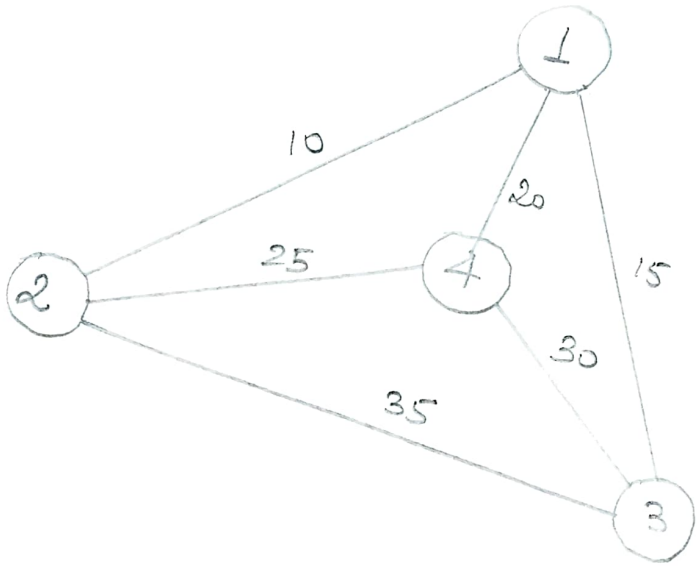
Date

Experiment - 3

* Solve Travelling Salesman Problem

```
from sys import maxsize
from itertools import permutations
V = 4   # vertex

def travellingSalesmanProblem(graph, s):
    vertex = []
    for i in range(V):
        if (i != s):
            vertex.append(i)
    min_path = maxsize
    next_permutation = permutations(vertex)

    for i in next_permutation:
        current_pathweight = 0
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathWeight)
    return min_path
```

Teacher's Signature : .....



Output: 80

```
graph = [ [0,10,15,20],
          [10,0,35,25],
          [15,35,0,30],
          [20,25,30,0] ]

s=0

print(travellingSalesmanProblem(graph, s))
```

4 Expt. No.

7 Page No.

Date

Experiment - 4

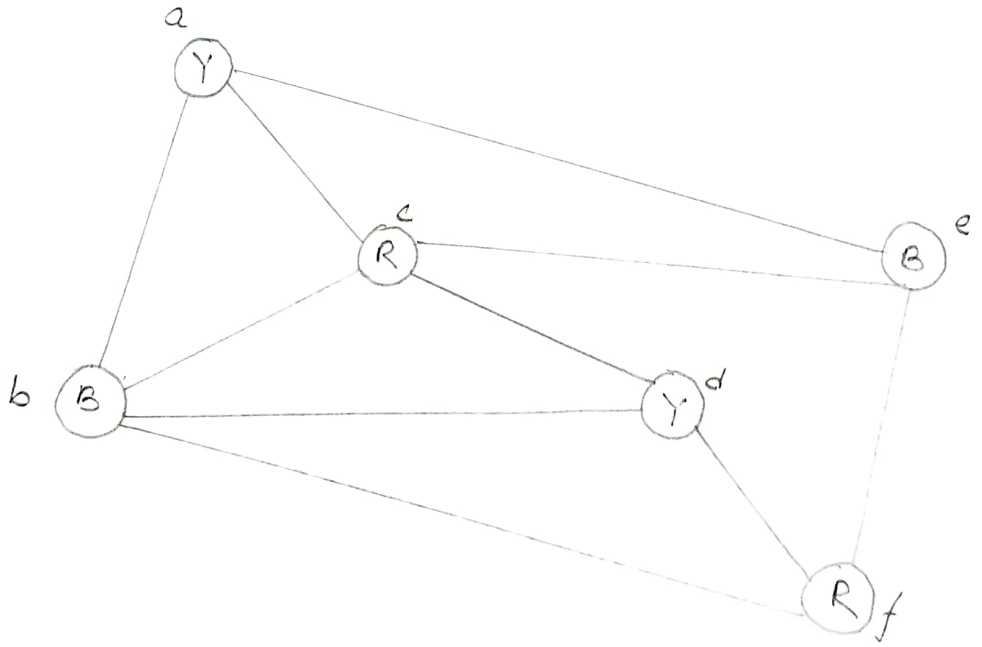
* Solve "4 - color map" problem

#Adjacent Matrix

```
G = [ [ 0, 1, 1, 0, 1, 0 ],
      [ 1, 0, 1, 1, 0, 1 ],
      [ 1, 1, 0, 1, 1, 0 ],
      [ 0, 1, 1, 0, 0, 1 ],
      [ 1, 0, 1, 0, 0, 1 ],
      [ 0, 1, 0, 1, 1, 0 ] ]

node = "abcdef"
t_ = { }
for i in range(len(G)):
        t_[node[i]] = i

degree = [ ]
for i in range(len(G)):
        degree.append(sum(G[i]))

ColorDict = { }
for i in range[ len(G)):
        ColorDict[node[i]] = ["Blue", "Red", "Yellow", "Green" ]
```

Teacher's Signature : .....

Node a = Yellow
Node b = Blue
Node c = Red
Node d = Yellow
Node e = Blue
Node f = Red



Expt. No. ..... Page No. 8 Date .....

```
sortedNode = []
indeks = []

for i in range (len (degree)):
    _max = 0
    j = 0
    for j in range (len(degree)):
        if j not in indeks:
            if degree[j] > _max:
                _max = degree [j]
                idx = j
    indeks. append (idx)
    sortedNode. append ( node[idx])

solution = {}
for n in sortedNode:
    setTheColor = colorsDict[n]
    solution [n] = setTheColor [0]
    adjacentNode = G [t_[n]]
    for j in range (len (adjacentNode)):
        if (adjacentNode [j] == 1 and (setTheColor [0] in colorDict [node[j]]):
            colorDict [node[j]]. remove (setTheColor [0])

for t, w in sorted (solution.items ()):
    print ("Node", t, " = ", w)
```

Teacher's Signature : .....