

**COMPETITIVE PROFESSIONAL**

**SKILLS-III**

Subject Code: (18CSC203J)

B. TECH III Year / VI Semester

Name: SURAJ SINGH

Reg. No.: RA2011026030119



**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

SRM INSTITUTE OF SCIENCE & TECHNOLOGY,

Delhi NCR CAMPUS, MODINAGAR

SIKRI KALAN, DELHI MEERUT ROAD, DIST. – GHAZIABAD – 201204

Even Semester (Jan – May -2023)

[**www.srmimt.net**](http://www.srmimt.net)

**BONAFIDE CERTIFICATE**

Registration no.\_\_\_\_\_\_\_\_\_\_\_

*Certified to be the bonafide record of work done by* ***Suraj Singh*** *of 6th semester 3rd year B. TECH degree course in SRM INSTITUTE OF SCIENCE & TECHNOLOGY, DELHI-NCR Campus for the Department of* ***Computer Science & Engineering****, in Competitive Professional Skills-III during the academic year 2022-2023.*

**Lab In charge**  **Head of the department**

*Submitted for End Semester examination held on \_\_/\_\_\_\_/\_\_\_ at SRM INSTITUTE OF SCIENCE & TECHNOLOGY, DELHI-NCR Campus.*

*Internal Examiner-I Internal Examiner-II*

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Exp. No.** | **Title of Experiment** | **Date of Experiment** | **Signature** |
| 1. | Program on Statistics mean, median and mode. |  |  |
| 2. | Program on Sums and Function. |  |  |
| 3. | Program on graph. |  |  |
| 4. | Coding problems on graphs. |  |  |
| 5. | Problems on Huffman. |  |  |
| 6. | Program on Greedy Methods. |  |  |
| 7. | Program on Job Sequencing |  |  |
| 8. | Program on MST (Minimum Spanning Tree). |  |  |
| 9. | Program on problem solving applying Dynamic programming strategies. |  |  |
| 10. | Program on problem solving applying Dynamic programming strategies. |  |  |
| 11. | Program on Solutions to Classical grid problems of Dynamic programming |  |  |
| 12. | Program on Solutions to problems on Divide and Conquer approach |  |  |
| 13. | Program on backtracking implementing iterative and loop free approaches |  |  |
| 14. | Coding implementing R Programming. |  |  |
| 15. | Problem Solving using R Programming. |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Lab – 01**

**Aim – Program on Statistics mean, median and mode.**

**Code –**

**Mean**

n\_num = [1, 2, 3, 4, 5]

n = len(n\_num)

get\_sum = sum(n\_num)

mean = get\_sum / n

print("Mean / Average is: " + str(mean))

**Output**

Mean / Average is: 3.0

**Median**

n\_num = [1, 2, 3, 4, 5]

n = len(n\_num)

n\_num.sort()

if n % 2 == 0:

    median1 = n\_num[n//2]

    median2 = n\_num[n//2 - 1]

    median = (median1 + median2)/2

else:

    median = n\_num[n//2]

print("Median is: " + str(median))

**Output**

Median is: 3

**Mode**

from collections import Counter

# list of elements to calculate mode

n\_num = [1, 2, 3, 4, 5, 5]

n = len(n\_num)

data = Counter(n\_num)

get\_mode = dict(data)

mode = [k for k, v in get\_mode.items() if v == max(list(data.values()))]

if len(mode) == n:

    get\_mode = "No mode found"

else:

    get\_mode = "Mode is / are: " + ', '.join(map(str, mode))

print(get\_mode)

**Output**

Mode is / are: 5

**Lab – 02**

**Aim – Program on Sums and Function.**

**Code -**

numbers = [1,2,3,4,5,1,4,5]

# start parameter is not provided

Sum = sum(numbers)

print(Sum)

# start = 10

Sum = sum(numbers, 10)

print(Sum)

**Output**

25

35

**Lab – 03**

**Aim – Program on graph.**

**Code –**

class AdjNode:

  def \_\_init\_\_(self, data):

    self.vertex = data

    self.next = None

class Graph:

  def \_\_init\_\_(self, vertices):

    self.V = vertices

    self.graph = [None] \* self.V

  # Function to add an edge in an undirected graph

  def add\_edge(self, src, dest):

    # Adding the node to the source node

    node = AdjNode(dest)

    node.next = self.graph[src]

    self.graph[src] = node

    # Adding the source node to the destination as

    # it is the undirected graph

    node = AdjNode(src)

    node.next = self.graph[dest]

    self.graph[dest] = node

  # Function to print the graph

  def print\_graph(self):

    for i in range(self.V):

      print("Adjacency list of vertex {}\n head".format(i), end="")

      temp = self.graph[i]

      while temp:

        print(" -> {}".format(temp.vertex), end="")

        temp = temp.next

      print(" \n")

# Driver program to the above graph class

if \_\_name\_\_ == "\_\_main\_\_":

  V = 5

  graph = Graph(V)

  graph.add\_edge(0, 1)

  graph.add\_edge(0, 4)

  graph.add\_edge(1, 2)

  graph.add\_edge(1, 3)

  graph.add\_edge(1, 4)

  graph.add\_edge(2, 3)

  graph.add\_edge(3, 4)

  graph.print\_graph()

**Output**

Adjacency list of vertex 0

head -> 4 -> 1

Adjacency list of vertex 1

head -> 4 -> 3 -> 2 -> 0

Adjacency list of vertex 2

head -> 3 -> 1

Adjacency list of vertex 3

head -> 4 -> 2 -> 1

Adjacency list of vertex 4

head -> 3 -> 1 -> 0

**Lab – 04**

**Aim – Coding problems on graphs.**

**Code -**

"""

A Python program to demonstrate the adjacency

list representation of the graph

"""

# A class to represent the adjacency list of the node

class AdjNode:

def \_\_init\_\_(self, data):

self.vertex = data

self.next = None

# A class to represent a graph. A graph

# is the list of the adjacency lists.

# Size of the array will be the no. of the

# vertices "V"

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = [None] \* self.V

# Function to add an edge in an undirected graph

def add\_edge(self, src, dest):

# Adding the node to the source node

node = AdjNode(dest)

node.next = self.graph[src]

self.graph[src] = node

# Adding the source node to the destination as

# it is the undirected graph

node = AdjNode(src)

node.next = self.graph[dest]

self.graph[dest] = node

# Function to print the graph

def print\_graph(self):

for i in range(self.V):

print("Adjacency list of vertex {}\n head".format(i), end="")

temp = self.graph[i]

while temp:

print(" -> {}".format(temp.vertex), end="")

temp = temp.next

print(" \n")

# Driver program to the above graph class

if \_\_name\_\_ == "\_\_main\_\_":

V = 5

graph = Graph(V)

graph.add\_edge(0, 1)

graph.add\_edge(0, 4)

graph.add\_edge(1, 2)

graph.add\_edge(1, 3)

graph.add\_edge(1, 4)

graph.add\_edge(2, 3)

graph.add\_edge(3, 4)

graph.print\_graph()

**Output**

Adjacency list of vertex 0

head -> 1-> 4

Adjacency list of vertex 1

head -> 0-> 2-> 3-> 4

Adjacency list of vertex 2

head -> 1-> 3

Adjacency list of vertex 3

head -> 1-> 2-> 4

Adjacency list of vertex 4

head -> 0-> 1-> 3

**Lab – 05**

**Aim – Problems on Huffman.**

**Code -**

# A Huffman Tree Node

import heapq

class node:

def \_\_init\_\_(self, freq, symbol, left=None, right=None):

# frequency of symbol

self.freq = freq

# symbol name (character)

self.symbol = symbol

# node left of current node

self.left = left

# node right of current node

self.right = right

# tree direction (0/1)

self.huff = ''

def \_\_lt\_\_(self, nxt):

return self.freq < nxt.freq

# utility function to print huffman

# codes for all symbols in the newly

# created Huffman tree

def printNodes(node, val=''):

# huffman code for current node

newVal = val + str(node.huff)

# if node is not an edge node

# then traverse inside it

if(node.left):

printNodes(node.left, newVal)

if(node.right):

printNodes(node.right, newVal)

# if node is edge node then

# display its huffman code

if(not node.left and not node.right):

print(f"{node.symbol} -> {newVal}")

# characters for huffman tree

chars = ['a', 'b', 'c', 'd', 'e', 'f']

# frequency of characters

freq = [5, 9, 12, 13, 16, 45]

# list containing unused nodes

nodes = []

# converting characters and frequencies

# into huffman tree nodes

for x in range(len(chars)):

heapq.heappush(nodes, node(freq[x], chars[x]))

while len(nodes) > 1:

# sort all the nodes in ascending order

# based on their frequency

left = heapq.heappop(nodes)

right = heapq.heappop(nodes)

# assign directional value to these nodes

left.huff = 0

right.huff = 1

# combine the 2 smallest nodes to create

# new node as their parent

newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

heapq.heappush(nodes, newNode)

# Huffman Tree is ready!

printNodes(nodes[0])

**Output**

f: 0

c: 100

d: 101

a: 1100

b: 1101

e: 111

**Lab – 06**

**Aim – Program on Greedy Methods.**

**Code -**

# Python3 program to find efficient

# solution for the network

# number of houses and number

# of pipes

n = 0

p = 0

# Array rd stores the

# ending vertex of pipe

rd = [0]\*1100

# Array wd stores the value

# of diameters between two pipes

wt = [0]\*1100

# Array cd stores the

# starting end of pipe

cd = [0]\*1100

# List a, b, c are used

# to store the final output

a = []

b = []

c = []

ans = 0

def dfs(w):

global ans

if (cd[w] == 0):

return w

if (wt[w] < ans):

ans = wt[w]

return dfs(cd[w])

# Function performing calculations.

def solve(arr):

global ans

i = 0

while (i < p):

q = arr[i][0]

h = arr[i][1]

t = arr[i][2]

cd[q] = h

wt[q] = t

rd[h] = q

i += 1

a = []

b = []

c = []

'''If a pipe has no ending vertex

but has starting vertex i.e is

an outgoing pipe then we need

to start DFS with this vertex.'''

for j in range(1, n + 1):

if (rd[j] == 0 and cd[j]):

ans = 1000000000

w = dfs(j)

# We put the details of component

# in final output array

a.append(j)

b.append(w)

c.append(ans)

print(len(a))

for j in range(len(a)):

print(a[j], b[j], c[j])

# Driver function

n = 9

p = 6

arr = [[7, 4, 98], [5, 9, 72], [4, 6, 10 ],

[2, 8, 22 ], [9, 7, 17], [3, 1, 66]]

solve(arr)

**Output**

3

2 8 22

3 1 66

5 6 10

**Lab – 07**

**Aim – Program on Job Sequencing.**

**Code -**

# Python3 code for the above approach

# function to schedule the jobs take 2

# arguments array and no of jobs to schedule

def printJobScheduling(arr, t):

# length of array

n = len(arr)

# Sort all jobs according to

# decreasing order of profit

for i in range(n):

for j in range(n - 1 - i):

if arr[j][2] < arr[j + 1][2]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

# To keep track of free time slots

result = [False] \* t

# To store result (Sequence of jobs)

job = ['-1'] \* t

# Iterate through all given jobs

for i in range(len(arr)):

# Find a free slot for this job

# (Note that we start from the

# last possible slot)

for j in range(min(t - 1, arr[i][1] - 1), -1, -1):

# Free slot found

if result[j] is False:

result[j] = True

job[j] = arr[i][0]

break

# print the sequence

print(job)

# Driver's Code

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

arr = [['a', 2, 100], # Job Array

['b', 1, 19],

['c', 2, 27],

['d', 1, 25],

['e', 3, 15]]

print("Following is maximum profit sequence of jobs")

# Function Call

printJobScheduling(arr, 3)

**Output**

Following is maximum profit sequence of jobs

c a e

**Lab – 08**

**Aim – Program on MST (Minimum Spanning Tree).**

**Code -**

# A Python3 program for getting minimum

# product spanning tree The program is

# for adjacency matrix representation

# of the graph

import math

# Number of vertices in the graph

V = 5

# A utility function to find the vertex

# with minimum key value, from the set

# of vertices not yet included in MST

def minKey(key, mstSet):

# Initialize min value

min = 10000000

min\_index = 0

for v in range(V):

if (mstSet[v] == False and

key[v] < min):

min = key[v]

min\_index = v

return min\_index

# A utility function to print the constructed

# MST stored in parent[] and print Minimum

# Obtainable product

def printMST(parent, n, graph):

print("Edge Weight")

minProduct = 1

for i in range(1, V):

print("{} - {} {} ".format(parent[i], i,

graph[i][parent[i]]))

minProduct \*= graph[i][parent[i]]

print("Minimum Obtainable product is {}".format(

minProduct))

# Function to construct and print MST for

# a graph represented using adjacency

# matrix representation inputGraph is

# sent for printing actual edges and

# logGraph is sent for actual MST

# operations

def primMST(inputGraph, logGraph):

# Array to store constructed MST

parent = [0 for i in range(V)]

# Key values used to pick minimum

key = [10000000 for i in range(V)]

# weight edge in cut

# To represent set of vertices not

mstSet = [False for i in range(V)]

# Yet included in MST

# Always include first 1st vertex in MST

# Make key 0 so that this vertex is

key[0] = 0

# Picked as first vertex

# First node is always root of MST

parent[0] = -1

# The MST will have V vertices

for count in range(0, V - 1):

# Pick the minimum key vertex from

# the set of vertices not yet

# included in MST

u = minKey(key, mstSet)

# Add the picked vertex to the MST Set

mstSet[u] = True

# Update key value and parent index

# of the adjacent vertices of the

# picked vertex. Consider only those

# vertices which are not yet

# included in MST

for v in range(V):

# logGraph[u][v] is non zero only

# for adjacent vertices of m

# mstSet[v] is false for vertices

# not yet included in MST. Update

# the key only if logGraph[u][v] is

# smaller than key[v]

if (logGraph[u][v] > 0 and

mstSet[v] == False and

logGraph[u][v] < key[v]):

parent[v] = u

key[v] = logGraph[u][v]

# Print the constructed MST

printMST(parent, V, inputGraph)

# Method to get minimum product spanning tree

def minimumProductMST(graph):

logGraph = [[0 for j in range(V)]

for i in range(V)]

# Constructing logGraph from

# original graph

for i in range(V):

for j in range(V):

if (graph[i][j] > 0):

logGraph[i][j] = math.log(graph[i][j])

else:

logGraph[i][j] = 0

# Applying standard Prim's MST algorithm

# on Log graph.

primMST(graph, logGraph)

# Driver code

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

''' Let us create the following graph

2 3

(0)--(1)--(2)

| / \ |

6| 8/ \5 |7

| / \ |

(3)-------(4)

9 '''

graph = [ [ 0, 2, 0, 6, 0 ],

[ 2, 0, 3, 8, 5 ],

[ 0, 3, 0, 0, 7 ],

[ 6, 8, 0, 0, 9 ],

[ 0, 5, 7, 9, 0 ], ]

# Print the solution

minimumProductMST(graph)

**Output**

Edge Weight

0 - 1 2

1 - 2 3

0 - 3 6

1 - 4 5

Minimum Obtainable product is 180

**Lab – 09**

**Aim – Program on problem solving applying Dynamic programming strategies.**

**Code -**

# Dynamic Programming implementation

# of Box Stacking problem

class Box:

# Representation of a box

def \_\_init\_\_(self, h, w, d):

self.h = h

self.w = w

self.d = d

def \_\_lt\_\_(self, other):

return self.d \* self.w < other.d \* other.w

def maxStackHeight(arr, n):

# Create an array of all rotations of

# given boxes. For example, for a box {1, 2, 3},

# we consider three instances{{1, 2, 3},

# {2, 1, 3}, {3, 1, 2}}

rot = [Box(0, 0, 0) for \_ in range(3 \* n)]

index = 0

for i in range(n):

# Copy the original box

rot[index].h = arr[i].h

rot[index].d = max(arr[i].d, arr[i].w)

rot[index].w = min(arr[i].d, arr[i].w)

index += 1

# First rotation of the box

rot[index].h = arr[i].w

rot[index].d = max(arr[i].h, arr[i].d)

rot[index].w = min(arr[i].h, arr[i].d)

index += 1

# Second rotation of the box

rot[index].h = arr[i].d

rot[index].d = max(arr[i].h, arr[i].w)

rot[index].w = min(arr[i].h, arr[i].w)

index += 1

# Now the number of boxes is 3n

n \*= 3

# Sort the array 'rot[]' in non-increasing

# order of base area

rot.sort(reverse = True)

# Uncomment following two lines to print

# all rotations

# for i in range(n):

# print(rot[i].h, 'x', rot[i].w, 'x', rot[i].d)

# Initialize msh values for all indexes

# msh[i] --> Maximum possible Stack Height

# with box i on top

msh = [0] \* n

for i in range(n):

msh[i] = rot[i].h

# Compute optimized msh values

# in bottom up manner

for i in range(1, n):

for j in range(0, i):

if (rot[i].w < rot[j].w and

rot[i].d < rot[j].d):

if msh[i] < msh[j] + rot[i].h:

msh[i] = msh[j] + rot[i].h

maxm = -1

for i in range(n):

maxm = max(maxm, msh[i])

return maxm

# Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

arr = [Box(4, 6, 7), Box(1, 2, 3),

Box(4, 5, 6), Box(10, 12, 32)]

n = len(arr)

print("The maximum possible height of stack is",

maxStackHeight(arr, n))

**Output**

The maximum possible height of stack is 60.

**Lab – 10**

**Aim – Program on problem solving applying Dynamic programming strategies.**

**Code -**

# Python code to implement the above approach

def knapSack(W, wt, val, n):

# Making the dp array

dp = [0 for i in range(W+1)]

# Taking first i elements

for i in range(1, n+1):

# Starting from back,

# so that we also have data of

# previous computation when taking i-1 items

for w in range(W, 0, -1):

if wt[i-1] <= w:

# Finding the maximum value

dp[w] = max(dp[w], dp[w-wt[i-1]]+val[i-1])

# Returning the maximum value of knapsack

return dp[W]

# Driver code

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

profit = [60, 100, 120]

weight = [10, 20, 30]

W = 50

n = len(profit)

print(knapSack(W, weight, profit, n))

**Output**

220

**Lab – 11**

**Aim – Program on Solutions to Classical grid problems of Dynamic programming.**

**Code -**

# Python 3 program to find the longest repeated

# non-overlapping substring

# Returns the longest repeating non-overlapping

# substring in str

def longestRepeatedSubstring(str):

n = len(str)

LCSRe = [[0 for x in range(n + 1)]

for y in range(n + 1)]

res = "" # To store result

res\_length = 0 # To store length of result

# building table in bottom-up manner

index = 0

for i in range(1, n + 1):

for j in range(i + 1, n + 1):

# (j-i) > LCSRe[i-1][j-1] to remove

# overlapping

if (str[i - 1] == str[j - 1] and

LCSRe[i - 1][j - 1] < (j - i)):

LCSRe[i][j] = LCSRe[i - 1][j - 1] + 1

# updating maximum length of the

# substring and updating the finishing

# index of the suffix

if (LCSRe[i][j] > res\_length):

res\_length = LCSRe[i][j]

index = max(i, index)

else:

LCSRe[i][j] = 0

# If we have non-empty result, then insert

# all characters from first character to

# last character of string

if (res\_length > 0):

for i in range(index - res\_length + 1,

index + 1):

res = res + str[i - 1]

return res

# Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

str = "geeksforgeeks"

print(longestRepeatedSubstring(str))

**Output**

geeks

**Lab – 12**

**Aim – Program on Solutions to problems on Divide and Conquer approach**

**Code -**

# Python3 code to demonstrate Divide and

# Conquer Algorithm

# Function to find the maximum no.

# in a given array.

def DAC\_Max(a, index, l):

max = -1

if(l - 1 == 0):

return arr[index]

if (index >= l - 2):

if (a[index] > a[index + 1]):

return a[index]

else:

return a[index + 1]

# Logic to find the Maximum element

# in the given array.

max = DAC\_Max(a, index + 1, l)

if (a[index] > max):

return a[index]

else:

return max

# Function to find the minimum no.

# in a given array.

def DAC\_Min(a, index, l):

min = 0

if(l - 1 == 0):

return arr[index]

if (index >= l - 2):

if (a[index] < a[index + 1]):

return a[index]

else:

return a[index + 1]

# Logic to find the Minimum element

# in the given array.

min = DAC\_Min(a, index + 1, l)

if (a[index] < min):

return a[index]

else:

return min

# Driver Code

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

# Defining the variables

min, max = 0, -1

# Initializing the array

a = [70, 250, 50, 80, 140, 12, 14]

# Recursion - DAC\_Max function called

max = DAC\_Max(a, 0, 7)

# Recursion - DAC\_Max function called

min = DAC\_Min(a, 0, 7)

print("The minimum number in a given array is : ", min)

print("The maximum number in a given array is : ", max)

**Output**

Maximum: 120

Minimum: 11

**Lab – 13**

**Aim – Program on backtracking implementing iterative and loop free approaches**

**Code -**

# Python program for above approach

import math

result = []

# Program to solve N-Queens Problem

def solveBoard(board, row, rowmask,

ldmask, rdmask):

n = len(board)

all\_rows\_filled = (1 << n) - 1

if (rowmask == all\_rows\_filled):

v = []

for i in board:

for j in range(len(i)):

if i[j] == 'Q':

v.append(j+1)

result.append(v)

# We extract a bit mask(safe) by rowmask,

# ldmask and rdmask. all set bits of 'safe'

# indicates the safe column index for queen

# placement of this iteration for row index(row).

safe = all\_rows\_filled & (~(rowmask |

ldmask | rdmask))

while (safe > 0):

# Extracts the right-most set bit

# (safe column index) where queen

# can be placed for this row

p = safe & (-safe)

col = (int)(math.log(p)/math.log(2))

board[row][col] = 'Q'

# these bit masks will keep updated in each

# iteration for next row

solveBoard(board, row+1, rowmask | p,

(ldmask | p) << 1, (rdmask | p) >> 1)

# Reset right-most set bit to 0 so, next

# iteration will continue by placing the queen

# at another safe column index of this row

safe = safe & (safe-1)

# Backtracking, replace 'Q' by ' '

board[row][col] = ' '

# Program to print board

def printBoard(board):

for row in board:

print("|" + "|".join(row) + "|")

# Driver Code

def main():

n = 4 # board size

board = []

for i in range(n):

row = []

for j in range(n):

row.append(' ')

board.append(row)

rowmask = 0

ldmask = 0

rdmask = 0

row = 0

# Function Call

result.clear()

solveBoard(board, row, rowmask, ldmask, rdmask)

result.sort()

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output**

[2 4 1 3 ][3 1 4 2 ]

**Lab – 14**

**Aim – Coding implementing R Programming.**

**Code -**

# Create two vectors of different lengths.

vector1 <- c(5,9,3)

vector2 <- c(10,11,12,13,14,15)

column.names <- c("COL1","COL2","COL3")

row.names <- c("ROW1","ROW2","ROW3")

matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.

result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,

column.names, matrix.names))

# Print the third row of the second matrix of the array.

print(result[3,,2])

# Print the element in the 1st row and 3rd column of the 1st matrix.

print(result[1,3,1])

# Print the 2nd Matrix.

print(result[,,2])

**Output**

COL1 COL2 COL3

3 12 15

[1] 13

COL1 COL2 COL3

ROW1 5 10 13

ROW2 9 11 14

ROW3 3 12 15

**Lab – 15**

**Aim – Problem Solving using R Programming.**

**Code -**

# declaring matrix

mat = matrix(c(1, NA, 2, 3, NA, 4), ncol = 2)

# replacing matrix NA with 0s

mat[is.na(mat)] = 0

# printing original matrix

print ("Original Matrix")

print (mat)

# calculating transpose of the

# matrix

transmat = t(mat)

print ("Transpose Matrix")

print (transmat)

# calculating product of matrices

prod = mat%\*%transmat

print ("Product Matrix")

print (prod)

**Output**

[1] "Original Matrix"

[,1] [,2]

[1,] 1 3

[2,] 0 0

[3,] 2 4

[1] "Transpose Matrix"

[,1] [,2] [,3]

[1,] 1 0 2

[2,] 3 0 4

[1] "Product Matrix"

[,1] [,2] [,3]

[1,] 10 0 14

[2,] 0 0 0

[3,] 14 0 20