**CYCLE 1**

Question -1:

To implement Insertion sort

Code

def insertion\_sort(array):

length = len(array)

for i in range(1, length):

key = array[i]

j = i

while j > 0 and key < array[j - 1]:

array[j] = array[j - 1]

j -= 1

array[j] = key

return array

length = int(input("How many elements should be inputed?\n"))

print("Enter the elements in array : ")

arr = [int(input("=>")) for i in range(length)]

print(insertion\_sort(arr))

Output:

How many elements should be inputed?

10

Enter the elements in array :

=>10

=>30

=>2

=>7

=>8

=>34

=>23

=>90

=>10

=>22

[2, 7, 8, 10, 10, 22, 23, 30, 34, 90]

Question -2

To implement merge sort

Code

def merge\_sort(array):

if len(array) > 1:

mid = len(array)//2

left = array[:mid]

right = array[mid:]

merge\_sort(left)

merge\_sort(right)

i = k = j = 0

while i < len(left) and j < len(right):

if left[i] < right[j]:

array[k] = left[i]

i += 1

else:

array[k] = right[j]

j += 1

k += 1

while i < len(left):

array[k] = left[i]

i += 1

k += 1

while j < len(right):

array[k] = right[j]

j += 1

k += 1

return array

length = int(input("How many elements should be inputed?\n"))

print("Enter the elements in array : ")

array = [int(input("=>")) for i in range(length)]

print(merge\_sort(array))

Output:

How many elements should be inputed?

5

Enter the elements in array :

=>2

=>4

=>6

=>8

=>22

[2, 4, 5, 6, 8, 22]

Question - 3

To implement String pattern matching using naïve method

Code

string = input("Enter the string: ")

pattern = input("Enter the pattern to be searched: ")

stringArray = []

patternArray = []

for i in string :

stringArray.append(i)

for j in pattern :

patternArray.append(j)

patternIndex = 0

stringIndex = 0

start = stringIndex

flag = 0

while stringIndex < len(stringArray):

if stringArray[stringIndex] == patternArray[patternIndex] :

stringIndex += 1

patternIndex += 1

else :

stringIndex = start + 1

start = stringIndex

patternIndex = 0

# printing location of founded pattern

if patternIndex == len(patternArray):

print(f"\nGiven pattern found in string from position: \n{start} to {stringIndex - 1}")

patternIndex = 0

flag = 1

if flag == 0:

print("\nGiven pattern is not found in the string")

Output:

Enter the string: ABcdcagegorysi

Enter the pattern to be searched: cage

Given pattern found in string from position:

4 to 7

Question – 4

To implement binary search

Code

def binarySearch(array, key):

firstElement = 0

lastElement = len(array) -1 # last element = length of array -1

while lastElement >= firstElement :

middleElement = int((firstElement+lastElement)/2)

if array[middleElement] == key:

return middleElement

elif array[middleElement] > key:

lastElement = middleElement - 1

elif array[middleElement] < key :

firstElement = middleElement + 1

return

#Inputing the elements in array

length = int(input("Enter the Number of elements you want to input in an array : "))

print("Enter the elements in array")

array = [int(input(">")) for i in range(length)]

array = sorted(array)

key = int(input("Enter the element to be searched : "))

searchResult = binarySearch(array, key) + 1 #adding 1 to make the postion of array start from 1

if searchResult == None:

print('element is not found')

else:

print(f'\n {key} is found at {searchResult}th position')

Output:

Enter the Number of elements you want to input in an array : 6

Enter the elements in array

>5

>9

>3

>3

>6

>2

Enter the element to be searched : 6

Question – 5

To implement n-queen problem

Code

import numpy as np

#function to generate chess board as input and then recieve output to print

def Nqueen(n):

def attackingAreas(board, row, column):

#Checking column corresponding to that cell

for i in range(row, -1, -1):

if board[i][column] == 1:

return False

#Checking upper diagonal corresponding to that cell

for i,j in zip(range(row, -1, -1), range(column, -1, -1)):

if board[i][j] == 1:

return False

#Checking lower diagonal corresponding to that cell

for i, j in zip(range(row, -1, -1), range(column, n)):

if board[i][j] == 1:

return False

return True

def SolveNQueen(board, row) :

if row == n:

return True

for i in range(n):

if attackingAreas(board, row, i) is True :

board[row][i] = 1

if SolveNQueen(board, row + 1):

return True

board[row][i] = 0

return False

board = [0 for i in range(n \* n)]

board = np.array(board)

board = board.reshape(n, n)

firstRowIndex = 0

Output = SolveNQueen(board, firstRowIndex)

return board

#read number of queens to be place in n \* n chess board

n = int(input("Enter the n - queens to be placed in a n \* n chess board => "))

if n >= 4:

print(Nqueen(n))

else:

print("N must be more than or equal to 4")

Output:

Enter the n - queens to be placed in a n \* n chess board => 4

[[0 1 0 0]

[0 0 0 1]

[1 0 0 0]

[0 0 1 0]]

Question – 6

To implement All pair shortest path Question

Code

import numpy as np

def printGraph(graph):

for i in graph:

for j in i:

if(j == inf):

print("INF", end="\t")

else:

print(j, end="\t")

print("")

print("\n") #leaves space after printing graph

def allPairShortestPathFinder(graph):

v = len(graph)

# Formula to find the shortest path for all pair of vertices

for k in range(v):

for i in range(v):

for j in range(v):

graph[i][j] = min(graph[i][j], graph[i][k]+graph[k][j])

return graph

v = 4

inf = 99999999999999999999999

graph = [[0, 5, inf, 10],

[inf, 0, 3, inf],

[inf, inf, 0, 1],

[inf, inf, inf, 0]]

graph = np.array(graph)#Converting the normal list to array

print("Input graph : ")

printGraph(graph)

#finding all pair shortest path for input graph

result = allPairShortestPathFinder(graph)

print("Shortest distance between every pair of vertices : ")

printGraph(result)

Output:

Input graph :

0 5 INF 10

INF 0 3 INF

INF INF 0 1

INF INF INF 0

Shortest distance between every pair of vertices :

0 5 8 9

INF 0 3 4

INF INF 0 1

INF INF INF 0

**CYCLE 2**

Question – 1

To implement Breadth first search

Code

def BFS(graph, startVertex):

queue, visited, traversalList = [startVertex], {}, []

for i in graph:

visited.update({i:False})

while queue:

vertex = queue.pop(0)

visited[vertex] = True

traversalList.append(vertex)

for node in graph[vertex]:

if visited[node] is False and node not in queue:

queue.append(node)

return traversalList

graph = { 0:[1, 2, 3], 1:[0, 2], 2:[0, 1, 4], 3:[0], 4:[2]}

print("Input graph : ", graph)

print("Breadth first search :")

for i in BFS(graph,2):

print(i, end=" ")

Output:

Input graph : {0: [1, 2, 3], 1: [0, 2], 2: [0, 1, 4], 3: [0], 4: [2]}

Breadth first search :

2 => 0 => 1 => 4 => 3

Question – 2

To implement Depth first search

Code

def DFS(graph, currentNode):

print(currentNode, end=" ")

for node in graph[currentNode]:

if DFS(graph, node):

return True

return False

graph = {

'A' : ['B', 'C'],

'B' : ['D', 'E'],

'C' : ['G'],

'D' : [],

'E' : ['F'],

'F' : [],

'G' : []

}

print("Input graph : ", graph)

print("Depth first search :")

DFS(graph, 'A')

Output:

Input graph : {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['G'], 'D': [], 'E': ['F'], 'F': [], 'G': []}

Depth first search :

A B D E F C G

Question – 3

To implement Iterative Deepening Depth first search

Code

def DFS(graph, currentNode, depth):

print(currentNode, end=" ")

if depth > 0:

for node in graph[currentNode]:

if DFS(graph, node, depth - 1):

return True

else:

return False

def IDDFS(graph, currentNode, depth):

for i in range(depth):

if DFS(graph, currentNode, i):

return True

return False

graph = {

'A' : ['B', 'C'],

'B' : ['D', 'E'],

'C' : ['G'],

'D' : [],

'E' : ['F'],

'F' : [],

'G' : []

}

print("Input graph : ", graph)

print("Iterative Deepening Depth first search :")

IDDFS(graph, 'A', 4)

Output:

Input graph : {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['G'], 'D': [], 'E': ['F'], 'F': [], 'G': []}

Iterative Deepening Depth first search :

A

A => B => C

A => B => D => E => C => G

A=> B => D => E => F => C => G

Question – 4

To implement Iterative Deepening Depth first search

Code

def uniformCostSearch(graph, cost, start, goal):

opened = []

closed = []

costToChild = 0

opened.append((0, start, []))

while opened:

opened.sort(reverse=True)

selected\_node = opened.pop()

nameOfselected\_node = selected\_node[1]

costOfselected\_node = selected\_node[0]

pathWayOfnode = selected\_node[2]

if nameOfselected\_node == goal:

pathWayOfnode.append(nameOfselected\_node)

return pathWayOfnode, costOfselected\_node

closed.append(selected\_node)

new\_nodes = graph[nameOfselected\_node]

if new\_nodes:

for child in new\_nodes:

costToChild = cost[(nameOfselected\_node, child)]

for i in range(len(opened)):

if child in opened[i] :

if costToChild + costOfselected\_node < opened[i][0]:

opened.pop(i)

path = selected\_node[2].copy()

path.append(nameOfselected\_node)

opened.append((costToChild + costOfselected\_node, child, path))

if i < len(closed) and child in closed[i]:

break

else:

path = selected\_node[2].copy()

path.append(nameOfselected\_node)

opened.append((costToChild + costOfselected\_node, child, path))

# create the graph

# add edge

graph = {

0 : [1, 3],

1 : [6],

2 : [1],

3 : [1, 4, 6],

4 : [2, 4],

5 : [2, 6],

6 : [4]

}

# add the cost

cost = {

(0, 1) : 2,

(0, 3) : 5,

(1, 6) : 1,

(3, 1) : 5,

(3, 6) : 6,

(3, 4) : 2,

(2, 1) : 4,

(4, 2) : 4,

(4, 5) : 3,

(5, 2) : 6,

(5, 6) : 3,

(6, 4) : 7

}

start = 0

goal = 6

ucs = uniformCostSearch(graph, cost, start, goal)

print("\nInput graph and cost : ", graph)

print("\ncost : ")

for i in cost.keys():

print(i[0], " => ", i[1], " : ", cost[i])

if ucs:

print("\nGoal node found....", "pathway :", sep="\n\n")

lengthOfpathway = len(ucs[0]) - 1

for i in ucs[0]:

if i != ucs[0][lengthOfpathway]:

print(i, end=" => ")

else:

print(i)

print(f"\nMinimum Cost of traversal from {start} to {goal} = ", ucs[1])

Output:

input graph and cost : {0: [1, 3], 1: [6], 2: [1], 3: [1, 2, 4], 4: [2, 4], 5: [2, 6], 6: [4]}

cost :

0 => 1 : 2

0 => 3 : 5

1 => 6 : 1

3 => 1 : 5

3 => 6 : 6

3 => 4 : 2

2 => 1 : 4

4 => 2 : 4

4 => 5 : 3

5 => 2 : 6

5 => 6 : 3

6 => 4 : 7

Goal node found....

pathway :

0 => 1 => 6

Minimum Cost of traversal from 0 to 6 = 3

Question – 5

To implement a binary search tree and perform insertion, deletion, searching operations

Code

Main code

from traversal import inorder, preorder, postorder

from BST\_operations import insert, search, delete

class Node:

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

self.left = None

self.right = None

self.value = key

#Driver code

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

r = Node(None)

while True:

print(

"""

Enter any of the number below to perform following operations

1. Insertion

2. Deletion

3. Search a node

4. Traversal

5. Exit

"""

)

choice = int(input("=> "))

if choice:

if choice == 1:

print("Insertion", end="\n\n")

while True:

nodeValue = int(input("Enter the node to be inserted => "))

insert(r, nodeValue, Node)

print(

"\n Enter :",

"X = to Stop insertion",

"Enter key to continue",

sep=" ",

end="\n\n",

)

stop = input()

if stop == "X" or stop == "x":

break

input("Press enter to continue")

elif choice == 2:

print("Deletion", end='\n\n')

nodeValue = int(input("Enter the element to be deleted : "))

if search(r, nodeValue, Node):

delete(r, nodeValue, Node)

print("\nSpecified element has been deleted")

else :

print("\nSpecified element is not in the given tree")

input("\nPress enter to continue")

elif choice == 3:

print("Search a node", end='\n\n')

key = int(input("Enter the element to be searched : "))

if search(r, key, Node):

print("\nElement found in the given tree")

else:

print("\nElement not found in the given tree")

input("\nPress enter to continue")

elif choice == 4:

print("Traversal", end="\n\n")

print(

"Enter the number given below to perform corresponding traversal",

"1. Inorder",

"2. Pre-Order",

"3. Post-Order",

sep= "\n", end="\n\n"

)

opted = int(input("=>"))

if opted == 1:

print("Inorder traversal \n")

inorder(r)

elif opted == 2:

print("Pre-order traversal \n")

preorder(r)

elif opted == 3:

print("Post-order traversal \n")

postorder(r)

else :

print("Invalid Input\n")

print("Try again")

input("\nPress enter to continue")

elif choice == 5:

print("Exiting")

break

else:

print("\nInvalid entry try again..")

input("\npress enter to continue..")

BST\_operations.py

#function to perform insertion

def insert(root, key, Node):

if root is None :

return Node(key)

elif root.value == None:

root.value = key

else:

if root.value == key:

return root

elif root.value > key:

root.left = insert(root.left, key, Node)

else:

root.right = insert(root.right, key, Node)

return root

#function to perform searching

def search(root, key, Node):

if root is None:

return False

#Checking that current node is key or not

if root.value == key:

return True

#If key is less than root.value then move to left node

elif root.value > key:

return search(root.left, key, Node)

#If key is greater than root.value then move to right node

else:

return search(root.right, key, Node)

#function to perform deletion

def delete(root, key, Node):

def minValueNode(node):

current = node

while current.left is not None :

current = current.left

return current

if root is None:

return root

elif root.value > key:

root.left = delete(root.left, key, Node)

elif root.value < key:

root.right = delete(root.right, key, Node)

else :

#if root has no left and right child

if root.left is None and root.right is None :

root = None

#if root has no left child

elif root.left is None :

temp = root.right

root = None

return temp

#if root has no right child

elif root.right is None :

temp = root.left

root = None

return temp

else :

temp = minValueNode(root.right)

root.value = temp.value

root.right = delete(root.right, temp.value, Node)

return root

Traversal.py

# function to perform inorder traversal

def inorder(root):

if root:

inorder(root.left)

print(root.value, end=" ")

inorder(root.right)

# function to perform preorder traversal

def preorder(root):

if root:

print(root.value, end=" ")

inorder(root.left)

inorder(root.right)

# function to perform postorder traversal

def postorder(root):

if root:

inorder(root.left)

inorder(root.right)

print(root.value, end=" ")

Output:

Insertion

Enter the node to be inserted => 100

Enter the node to be inserted => 20

Enter the node to be inserted => 13

Enter the node to be inserted => 50

Enter the node to be inserted => 300

Enter the node to be inserted => 113

Inorder traversal of created binary tree

13 20 50 100 113 300

Deletion

Enter the element to be deleted : 50

Specified element has been deleted

Inorder traversal

13 20 100 113 300

Search a node

Enter the element to be searched : 113

Element found in the given tree

Enter the element to be searched : 12

Element not found in the given tree

**CYCLE 3**

Question – 1

To perform beam search on graph

Code

def beamSearch(graph, heuristics, start, goal, beam):

opened = []

closed = []

heuristicsToGoal = 0

opened.append((heuristicsToGoal, start, []))

while opened:

opened.sort(reverse=True)

selected\_node = opened.pop()

nameOfselected\_node = selected\_node[1]

heuristicsOfselected\_node = selected\_node[0]

pathWayOfnode = selected\_node[2]

if nameOfselected\_node == goal:

pathWayOfnode.append(nameOfselected\_node)

return pathWayOfnode

closed.append(selected\_node)

new\_nodes = graph[nameOfselected\_node]

if new\_nodes:

for child in new\_nodes:

heuristicsToGoal = heuristics[(nameOfselected\_node, goal)]

for i in range(len(opened)):

if child in opened[i] :

if heuristicsToGoal < opened[i][0]:

opened.pop(i)

path = selected\_node[2].copy()

path.append(nameOfselected\_node)

heuristicsOfchild = heuristics[(child, goal)]

opened.append((heuristicsOfchild, child, path))

if i < len(closed) and child in closed[i]:

break

else:

path = selected\_node[2].copy()

path.append(nameOfselected\_node)

heuristicsOfchild = heuristics[(child, goal)]

opened.append((heuristicsOfchild, child, path))

opened.sort()

for i in range(len(opened)):

if len(opened) > beam :

if i >= beam :

a = opened.pop(i)

# create the graph

# add edge

graph = {

0 : [1, 3],

1 : [6],

2 : [1],

3 : [1, 4, 6],

4 : [2, 4],

5 : [2, 6],

6 : [4]

}

cost = {

(0, 1) : 2,

(0, 3) : 5,

(1, 6) : 1,

(3, 1) : 5,

(3, 6) : 6,

(3, 4) : 2,

(2, 1) : 4,

(4, 2) : 4,

(4, 5) : 3,

(5, 2) : 6,

(5, 6) : 3,

(6, 4) : 7

}

start = 0

goal = 6

heuristics = {}

print("Beam Search")

print("\ninput graph : ", graph,"\nStarting node : ", start, "\nGoal node : ", goal, "\n")

print("\ncost : ")

for i in cost.keys():

print(i[0], " => ", i[1], " : ", cost[i])

#getting heuristics from user

for i in graph.keys():

if i is not goal:

heuristicValue = int(input(f"Enter the heuristics from {i} to Goal : "))

heuristics[(i, goal)] = heuristicValue

else:

heuristicValue = 0

heuristics[(i, goal)] = heuristicValue

beam = int(input("Enter the beam width : "))

beam\_search = beamSearch(graph, heuristics, start, goal, beam)

if beam\_search:

#printing path way

print("\nGoal node found....", "\npathway :", sep="\n")

lengthOfpathway = len(beam\_search)

for i in beam\_search:

if i != beam\_search[lengthOfpathway - 1 ]:

print(i, end=' => ')

else:

print(i)

#printing Cost for comparing with other search algorithm

total\_cost = 0

print("\nCost of Traversal : ")

for i in range(1, len(beam\_search)):

total\_cost += cost[(beam\_search[i-1], beam\_search[i])]

print(total\_cost)

else:

print("Goal node not found")

Output:

Beam Search

input graph : {0: [1, 3], 1: [6], 2: [1], 3: [1, 4, 6], 4: [2, 4], 5: [2, 6], 6: [4]}

Starting node : 0

Goal node : 6

cost :

0 => 1 : 2

0 => 3 : 5

1 => 6 : 1

3 => 1 : 5

3 => 6 : 6

3 => 4 : 2

2 => 1 : 4

4 => 2 : 4

4 => 5 : 3

5 => 2 : 6

5 => 6 : 3

6 => 4 : 7

Enter the heuristics from 0 to Goal : 4

Enter the heuristics from 1 to Goal : 3

Enter the heuristics from 2 to Goal : 5

Enter the heuristics from 3 to Goal : 2

Enter the heuristics from 4 to Goal : 5

Enter the heuristics from 5 to Goal : 6

Enter the beam width : 2

Goal node found....

pathway : 0 => 3 => 6

Cost of Traversal :

11

Question – 2

To implement A\* search Question

Code

def a\_star(graph, cost, heuristics, start, goal):

opened = []

closed = []

costToChild = 0

opened.append((0, start, []))

while opened:

opened.sort(reverse=True)

selected\_node = opened.pop()

nameOfselected\_node = selected\_node[1]

costOfselected\_node = selected\_node[0]

pathWayOfnode = selected\_node[2]

if nameOfselected\_node == goal:

pathWayOfnode.append(nameOfselected\_node)

return pathWayOfnode

closed.append(selected\_node)

new\_nodes = graph[nameOfselected\_node]

if new\_nodes:

for child in new\_nodes:

costToChild = cost[(nameOfselected\_node, child)]

costValue = costToChild + costOfselected\_node

for i in range(len(opened)):

if child in opened[i] :

if costValue < opened[i][0]:

opened.pop(i)

path = selected\_node[2].copy()

path.append(nameOfselected\_node)

heuristicsOfchild = heuristics[(child, goal)]

functionValue = heuristicsOfchild + costValue

opened.append((functionValue, child, path))

if i < len(closed) and child in closed[i]:

break

else:

path = selected\_node[2].copy()

path.append(nameOfselected\_node)

heuristicsOfchild = heuristics[(child, goal)]

functionValue = heuristicsOfchild + costValue

opened.append((functionValue, child, path))

# create the graph

# add edge

graph = {

0 : [1, 3],

1 : [6],

2 : [1],

3 : [1, 4, 6],

4 : [2, 4],

5 : [2, 6],

6 : [4]

}

# add the cost

cost = {

(0, 1) : 2,

(0, 3) : 5,

(1, 6) : 1,

(3, 1) : 5,

(3, 6) : 6,

(3, 4) : 2,

(2, 1) : 4,

(4, 2) : 4,

(4, 5) : 3,

(5, 2) : 6,

(5, 6) : 3,

(6, 4) : 7

}

heuristics = {}

start = 0

goal = 6

print("A\* search Algorithm")

print("\nInput graph and cost : ", graph)

print("\ncost : ")

for i in cost.keys():

print(i[0], " => ", i[1], " : ", cost[i])

#getting heuristics from user

for i in graph.keys():

if i is not goal:

heuristicValue = int(input(f"Enter the heuristics from {i} to Goal : "))

heuristics[(i, goal)] = heuristicValue

else:

heuristicValue = 0

heuristics[(i, goal)] = heuristicValue

astar = a\_star(graph, cost, heuristics,start, goal)

if astar:

#printing path way

print("\nGoal node found....", "\npathway :", sep="\n")

for i in astar:

if i != astar[len(astar) - 1 ]:

print(i, end=' => ')

else:

print(i)

#printing Cost for comparing with other search algorithm

total\_cost = 0

print("\nCost of Traversal : ")

for i in range(1, len(astar)):

total\_cost += cost[(astar[i-1], astar[i])]

print(total\_cost)

else:

print("Goal node not found")

Output:

A\* search Question

Input graph and cost : {0: [1, 3], 1: [6], 2: [1], 3: [1, 4, 6], 4: [2, 4], 5: [2, 6], 6: [4]}

cost :

0 => 1 : 2

0 => 3 : 5

1 => 6 : 1

3 => 1 : 5

3 => 6 : 6

3 => 4 : 2

2 => 1 : 4

4 => 2 : 4

4 => 5 : 3

5 => 2 : 6

5 => 6 : 3

6 => 4 : 7

Enter the heuristics from 0 to Goal : 4

Enter the heuristics from 1 to Goal : 3

Enter the heuristics from 2 to Goal : 5

Enter the heuristics from 3 to Goal : 2

Enter the heuristics from 4 to Goal : 5

Enter the heuristics from 5 to Goal : 6

Goal node found....

pathway :

0 => 1 => 6

Cost of Traversal :

3

Question – 3

To implement Beam search Question

Code

def bestFirstSearch(graph, heuristics, start, goal):

opened = []

closed = []

heuristicsToGoal = 0

opened.append((heuristicsToGoal, start, []))

while opened:

opened.sort(reverse=True)

selected\_node = opened.pop()

nameOfselected\_node = selected\_node[1]

heuristicsOfselected\_node = selected\_node[0]

pathWayOfnode = selected\_node[2]

if nameOfselected\_node == goal:

pathWayOfnode.append(nameOfselected\_node)

return pathWayOfnode

closed.append(selected\_node)

new\_nodes = graph[nameOfselected\_node]

if new\_nodes:

for child in new\_nodes:

heuristicsToGoal = heuristics[(nameOfselected\_node, goal)]

for i in range(len(opened)):

if child in opened[i] :

if heuristicsToGoal < opened[i][0]:

opened.pop(i)

path = selected\_node[2].copy()

path.append(nameOfselected\_node)

heuristicsOfchild = heuristics[(child, goal)]

opened.append((heuristicsOfchild, child, path))

if i < len(closed) and child in closed[i]:

break

else:

path = selected\_node[2].copy()

path.append(nameOfselected\_node)

heuristicsOfchild = heuristics[(child, goal)]

opened.append((heuristicsOfchild, child, path))

# create the graph

# add edge

graph = {

0 : [1, 3],

1 : [6],

2 : [1],

3 : [1, 4, 6],

4 : [2, 4],

5 : [2, 6],

6 : [4]

}

cost = {

(0, 1) : 2,

(0, 3) : 5,

(1, 6) : 1,

(3, 1) : 5,

(3, 6) : 6,

(3, 4) : 2,

(2, 1) : 4,

(4, 2) : 4,

(4, 5) : 3,

(5, 2) : 6,

(5, 6) : 3,

(6, 4) : 7

}

start = 0

goal = 6

heuristics = {}

print("Best first Search")

print("\ninput graph : ", graph,"\nStarting node : ", start, "\nGoal node : ", goal, "\n")

print("\ncost : ")

for i in cost.keys():

print(i[0], " => ", i[1], " : ", cost[i])

#getting heuristics from user

for i in graph.keys():

if i is not goal:

heuristicValue = int(input(f"Enter the heuristics from {i} to Goal : "))

heuristics[(i, goal)] = heuristicValue

else:

heuristicValue = 0

heuristics[(i, goal)] = heuristicValue

best\_first\_search = bestFirstSearch(graph, heuristics, start, goal)

if best\_first\_search:

#printing path way

print("\nGoal node found....", "\npathway :", sep="\n")

for i in best\_first\_search:

if i != best\_first\_search[len(best\_first\_search) - 1 ]:

print(i, end=' => ')

else:

print(i)

#printing Cost for comparing with other search algorithm

total\_cost = 0

print("\nCost of Traversal : ")

for i in range(1, len(best\_first\_search)):

total\_cost += cost[(best\_first\_search[i-1], best\_first\_search[i])]

print(total\_cost)

else:

print("Goal node not found")

Output:

Best first Search

input graph : {0: [1, 3], 1: [6], 2: [1], 3: [1, 4, 6], 4: [2, 4], 5: [2, 6], 6: [4]}

Starting node : 0

Goal node : 6

cost :

0 => 1 : 2

0 => 3 : 5

1 => 6 : 1

3 => 1 : 5

3 => 6 : 6

3 => 4 : 2

2 => 1 : 4

4 => 2 : 4

4 => 5 : 3

5 => 2 : 6

5 => 6 : 3

6 => 4 : 7

Enter the heuristics from 0 to Goal : 4

Enter the heuristics from 1 to Goal : 3

Enter the heuristics from 2 to Goal : 5

Enter the heuristics from 3 to Goal : 2

Enter the heuristics from 4 to Goal : 5

Enter the heuristics from 5 to Goal : 6

Goal node found....

pathway :

0 => 3 => 6

Cost of Traversal :

11

Question – 4

To implement Alpha beta pruning

Code

maximum, minimum = 1000, -1000

def alpha\_beta(depth, nodeIndex, maximize,

      value\_list, alpha, beta):

  if depth == 3:

    return value\_list[nodeIndex]

  if maximize:

    best = minimum

    for i in range(0, 2):

      value = alpha\_beta(depth + 1, nodeIndex \* 2 + i,False, value\_list, alpha, beta)

      best = max(best, value)

      alpha = max(alpha, best)

      if beta <= alpha:

        break

    return best

  else:

    best = maximum

    for i in range(0, 2):

      value = alpha\_beta(depth + 1, nodeIndex \* 2 + i,True, value\_list, alpha, beta)

      best = min(best, value)

      beta = min(beta, best)

      if beta <= alpha:

        break

    return best

values\_list = [3, 5, 6, 9, 1, 2, 0, -1]

print("The terminal nodes : ")

for i in values\_list:

  print(i, end=" ")

print("\nThe optimal value is :", alpha\_beta(0, 0, True, values\_list, minimum, maximum))

Output:

The terminal nodes :

3 5 6 9 1 2 0 -1

The optimal value is : 5

**Mathematics Questions**

Question – 1

Calculate L1 Norm of a vector.

Code

import numpy as np

def l1\_norm(vector):

norm = 0

for element in vector:

norm += abs(element)

return norm

print("Enter the values of vector : ")

list1 = [int(input()) for i in range(3)]

vector1 = np.array(list1)

resultVector1 = l1\_norm(list1)

print("The L1 norm of vector is : ", resultVector1)

Output:

Enter the values of vector :

1

2

3

The L1 norm of vector is : 6

Question – 2

Calculate L2 Norm of a vector.

Code

import numpy as np

def l2\_norm(vector):

norm = 0

for element in vector:

norm += element\*\*2

return norm\*\*0.5

print("Enter the values of vector : ")

list1 = [int(input()) for i in range(3)]

vector1 = np.array(list1)

resultVector1 = l2\_norm(list1)

print("The L2 norm of vector is : ", resultVector1)

Output:

Enter the values of vector :

1

2

3

The L2 norm of vector is : 3.7416573867739413

Question – 3

Calculate Max Norm of a vector.

Code

def max\_norm(vector):

norm = 0

for element in vector:

if abs(element) > norm:

norm = abs(element)

return norm

print("Enter the values of vector : ")

list1 = [int(input()) for i in range(3)]

vector1 = np.array(list1)

resultVector1 = max\_norm(list1)

print("The Max norm of vector is : ", resultVector1)

Output:

Enter the values of vector :

1

2

3

The Max norm of vector is : 3

Question – 4

Hadamard product of two matrix.

Code

import numpy as np

def hadamardProduct(matrix1, matrix2) :

resultantMatrix = []

for i in range(len(matrix1)):

row = []

for j in range(len(matrix2)):

row.append(matrix1[i][j] \* matrix2[i][j])

resultantMatrix.append(row)

return resultantMatrix

rows = int(input("Enter the number of rows in Matrices : "))

columns = int(input("Enter the number of columns in Matrices : "))

print(f"Enter the elements of the first {rows}x{columns} matrix")

matrix\_1 = [[int(input("=>")) for j in range(columns)] for i in range(rows)]

print(f"Enter the elements of the second {rows}x{columns} matrix")

matrix\_2 = [[int(input("=>")) for j in range(columns)] for i in range(rows)]

matrix\_1 = np.array(matrix\_1)

matrix\_2 = np.array(matrix\_2)

print("first Matrix \n",matrix\_1)

print("Second Matrix \n", matrix\_2)

result = hadamardProduct(matrix\_1, matrix\_2)

result = np.array(result)

print("Hadamard product of two matrices are:\n", result)

Output:

Enter the number of rows in Matrices : 2

Enter the number of columns in Matrices : 2

Enter the elements of the first 2x2 matrix

=>1

=>2

=>3

=>4

Enter the elements of the second 2x2 matrix

=>5

=>6

=>7

=>8

first Matrix

[[1 2]

[3 4]]

Second Matrix

[[5 6]

[7 8]]

Hadamard product of two matrices are:

[[ 5 12]

[21 32]]

Question – 5

Aim:

Define a 3 \* 3 square matrix. Extract the main diagonal as vector. Create the

diagonal matrix from the extracted vector.

Code

import numpy as np

def MatrixMaker(rows, columns):

a = [int(input("=>")) for i in range(rows \* columns)]

a = np.array(a)

a = a.reshape(rows, columns)

return a

rows = 3

columns = 3

print(f"Enter the elements of the {rows}x{columns} matrix")

matrix\_1 = MatrixMaker(rows, columns)

# extracting vector and making diagonal matrix

diagonalMatrix = []

#extracting vector using loop that only iterates the diagonal elements

vector = [matrix\_1[i][j] for i, j in zip(range(rows) ,range(columns))]

vector = np.array(vector) #converting to vector

#making diagonal matrix using vector

for i in range(rows):

for j in range(columns):

if i == j : diagonalMatrix.append(vector[i])

else : diagonalMatrix.append(0)

#You can use a single line code for making diagonal matrix

#diagonalMatrix = [matrix[i][j] if i==j else 0 for i, j in [(i, j) for i in range(rows) for j in range(columns)]]

diagonalMatrix = np.array(diagonalMatrix)

diagonalMatrix = diagonalMatrix.reshape(rows, columns)

#printing all values

print("Matrix \n", matrix\_1)

print("Vector \n",vector)

print("Diagonal Matrix \n",diagonalMatrix)

Output:

Enter the elements of the 3x3 matrix

=>12

=>3

=>5

=>6

=>7

=>8

=>9

=>4

=>2

Matrix

[[12 3 5]

[ 6 7 8]

[ 9 4 2]]

Vector

[12 7 2]

Diagonal Matrix

[12 0 0]

[ 0 7 0]

[ 0 0 2]

Question – 6

Find determinant of a matrix.

Code

import numpy as np

from math import sqrt

def determinant2D(matrix):

determinant = 0

diagonal1 = 1

diagonal2 = 1

for i in range(2):

for j in range(2):

if i == j :

diagonal1 \*= matrix[i][j]

else :

diagonal2 \*= matrix[i][j]

determinant = diagonal1 - diagonal2

return determinant

def determinantOfMatrix(matrix, dimension):

if(dimension < 3):

determinant = determinant2D(matrix)

else:

determinant = 0

for k in range(len(matrix[0])):

array = []

for i in range(dimension):

for j in range(dimension):

if i == 0 or j == k:

continue

else :

array.append(matrix[i][j])

array = np.array(array)

dimension2 = int(sqrt(len(array)))

array = array.reshape(dimension2, dimension2)

if k % 2 == 0 :

determinant += matrix[0][k] \* determinantOfMatrix(array, dimension2)

else :

determinant -= matrix[0][k] \* determinantOfMatrix(array, dimension2)

return determinant

dimension = int(input("Enter the no.of rows or columns in Matrix : "))

print('Enter the values of matrix')

matrix = [int(input()) for i in range(dimension\*\*2)]

matrix = np.array(matrix)

matrix = matrix.reshape(dimension, dimension)

print("Elements of matrix:\n",matrix)

print("determinant of matrix:\n",determinantOfMatrix(matrix, dimension))

Output:

Elements of matrix:

[[1 2 3]

[4 5 6]

[7 8 9]]

determinant of matrix:

0

Question – 7

Create an orthogonal matrix and check Q ^ Q = Q Q ^ = I.

Code

def product(matrix\_1, matrix\_2):

product = []

row = len(matrix\_1)

column1 = len(matrix\_1[0])

column2 = len(matrix\_2[0])

for i in range(row):

for j in range(column2):

sum = 0

for k in range(column1):

sum += matrix\_1[i][k] \* matrix\_2[k][j]

product.append(sum)

product = np.array(product)

product = product.reshape(row, column2)

return product

def transpose(matrix):

transpose = []

rows = len(matrix)

columns = len(matrix[0])

for i in range(columns):

for j in range(rows):

transpose.append(matrix[j][i])

transpose = np.array(transpose)

transpose = transpose.reshape(columns,rows)

return transpose

matrix = np.array([

[1/3,2/3, -2/3],

[-2/3,2/3,1/3],

[2/3,1/3,2/3]])

print("Matrix :", matrix, sep="\n")

transpose = transpose(matrix)

print("\nTranspose of matrix :", transpose, sep="\n")

identityMatrix = np.array([

[1, 0, 0],

[0, 1, 0],

[0, 0, 1]])

print("\nQ \* Qtranspose : ", product(matrix, transpose), sep='\n')

print("\nQtranspose \* Q : ", product(transpose, matrix), sep='\n')

print("\nHere we can see that Q \* Qtranspose = Qtranspose \* Q = ", identityMatrix, sep='\n')

Output:

Matrix :

[ 0.33333333 0.66666667 -0.66666667]

[-0.66666667 0.66666667 0.33333333]

[ 0.66666667 0.33333333 0.66666667]

Transpose of matrix :

[ 0.33333333 -0.66666667 0.66666667]

[ 0.66666667 0.66666667 0.33333333]

[-0.66666667 0.33333333 0.66666667]

Q \* Qtranspose :

[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]

Qtranspose \* Q :

[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]

Here we can see that Q \* Qtranspose = Qtranspose \* Q =

[1 0 0]

[0 1 0]

[0 0 1]

Question – 8

Find rank of a matrix.

Code

def rank(matrix):

num\_rows, num\_cols = matrix.shape

rank = min(num\_rows, num\_cols)

for col in range(rank):

if matrix[col, col] != 0:

for row in range(col + 1, num\_rows):

factor = matrix[row, col] / matrix[col, col]

matrix[row, col] -= factor \* matrix[col, col]

else:

reduce\_rank = True

for row in range(col + 1, num\_rows):

if matrix[row, col] != 0:

matrix[[col, row]] = matrix[[row, col]]

reduce\_rank = False

break

if reduce\_rank:

rank -= 1

if col < num\_cols - 1:

matrix[:, col] = matrix[:, rank]

return rank

order = 3

print("Enter elements in matrix : ")

matrix = [[int(input("=>")) for j in range(order)] for i in range(order)]

matrix = np.array(matrix)

print("Matrix : ", matrix, sep='\n')

rank = rank(matrix)

print("Rank of matrix is : ", rank)

Output :

Matrix :

[1 2 3]

[4 5 6]

[7 8 9]

Rank of matrix is : 3

Question – 9

Find sparsity of a matrix.

Code:

import numpy as np

def sparcity0fMatrix(matrix):

count = 0

for row in matrix:

for element in row :

if element == 0:

count += 1

rows = len(matrix)

columns = len(matrix[0])

TotalElements = rows \* columns

sparcity = count / TotalElements

return sparcity

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

columns = int(input("Enter the number of columns : "))

print("Enter elements in matrix : ")

matrix = [[int(input("=>")) for j in range(columns)] for i in range(rows)]

matrix = np.array(matrix)

print(matrix)

sparcity = sparcity0fMatrix(matrix)

print("\nSparcity of given matrix is : ", sparcity)

if sparcity > 0.5 :

print("The given Matrix is a sparse matrix")

else :

print("The given matrix is not sparse matrix")

Output:

[[5 0 3]

[0 0 6]

[2 0 0]]

Sparcity of given matrix is : 0.5555555555555556

The given Matrix is a sparse matrix

Question – 10

Find eigen value and eigen vector Of a Matrix

Code:

import numpy as np

from numpy.linalg import eig

order = 3

print("Enter the elements in the matrix")

matrix = np.array([[int(input("=>")) for j in range(order)] for i in range(order)])

print(matrix)

eigenValue, eigenVector = eig(matrix)

eigenValue = np.array(eigenValue)

eigenVector = np.array(eigenVector)

print("Eigen value of a matrix : ", eigenValue, sep='\n')

print("Eigen vector of a matrix : ", eigenVector, sep='\n')

Output:

Input matrix :

[[ 1 2 3]

[ 4 2 1]

[ 3 5 12]]

Eigen value of a matrix :

[13.67489478 -0.93733814 2.26244336]

Eigen vector of a matrix :

[-0.25207554 -0.55072961 -0.17900252]

[-0.16799434 0.81353407 -0.851715 ]

[-0.95301407 -0.18670622 0.49248315]

Question – 11

Print Eigen Values and eigen vectors of a matrix and reconstruct the matrix

Code:

import numpy as np

from numpy import dot, diag

from numpy.linalg import eig, inv

order = 3

print("Enter the elements in the matrix")

matrix = np.array([[int(input("=>")) for j in range(order)] for i in range(order)])

print(matrix)

eigenValue, eigenVector = eig(matrix)

InverseEigen = inv(eigenVector)

vectorDiagonal = diag(eigenValue)

rematrix = eigenVector.dot(vectorDiagonal).dot(InverseEigen)

print("eigen value of matrix is : ", eigenValue, sep="\n")

print("eigen vector of matrix is : ", eigenVector, sep="\n")

print("Reconstructed matrix is :", rematrix, sep="\n")

Output:

Input matrix :

[ 1 2 3]

[ 4 2 1]

[ 3 5 12]

eigen value of matrix is :

[13.67489478 -0.93733814 2.26244336]

eigen vector of matrix is :

[-0.25207554 -0.55072961 -0.17900252]

[-0.16799434 0.81353407 -0.851715 ]

[-0.95301407 -0.18670622 0.49248315]

Reconstructed matrix is :

[ 1. 2. 3.]

[ 4. 2. 1.]

[ 3. 5. 12.]