TOLANI COLLEGEOFCOMMERCE

***(Affiliated toUniversity of Mumbai)***

**Sher E Punjab Colony, Andheri East**

**MUMBAI-MAHARASHTRA-400093**

DEPARTMENT OF B.S.c(INFORMATION TECHNOLOGY)



**CERTIFICATE**

This is to certify that the project entitled, “**Artificial Intelligence Journal**”,is bonafied work of **Yuvraj Vijay Achrekar** bearing roll no: **03** submitted in partial fulfillment of the requirements for the award of degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

**Internal Examiner Co-ordinator**

**External Examiner**

**Date: College Seal**

Artificial Intelligence Practical

|  |  |
| --- | --- |
| **Practical No** | **Details** |
| **1a** | **Write a program to implement depth first search algorithm.** |
| **1b** | **Write a program to implement breadth first search algorithm.** |
| **2a** | **Write a program to simulate 4-Queen / N-Queen problem.** |
| **2b** | **Write a program to solve tower of Hanoi problem.** |
| **3a** | **Write a program to implement alpha beta search.** |
| **3b** | **Write a program for Hill climbing problem.** |
| **4a** | **Write a program to implement A\* algorithm.** |
| **4b** | **Write a program to implement AO\* algorithm.** |
| **5a** | **Write a program to solve water jug problem.** |
| **5b** | **Design the simulation of tic – tac – toe game using min-max algorithm.** |
| **6a** | **Write a program to solve Missionaries and Cannibals problem.** |
| **6b** | **Design an application to simulate number puzzle problem.** |
| **7a** | **Write a program to shuffle Deck of cards.** |
| **7b** | **Solve traveling salesman problem using artificial intelligence technique.** |
| **8a** | **Solve the block of World problem.** |
| **8b** | **Solve constraint satisfaction problem** |
| **9a** | **Derive the expressions based on Associative law** |
| **9b** | **Derive the expressions based on Distributive law** |
| **10a** | **Write a program to derive the predicate.** |
| **10b** | **Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather , grandmother, brother, sister, uncle, aunt, nephew and niece, cousin. Question: i. Draw Family Tree ii. Define: Clauses, Facts, Predicates and Rules with conjunction and disjunction** |

**1A - Write a program to implement depth first search algorithm.**

**Code:**

graph1 = {

    'A':set(['B','C']),

    'B':set(['A','D','E']),

    'C':set(['A','F']),

    'D':set(['B']),

    'E':set(['B','F']),

    'F':set(['C','E'])

}

def dfs(graph,node, visited):

    if node not in visited:

        visited.append(node)

        for n in graph[node]:

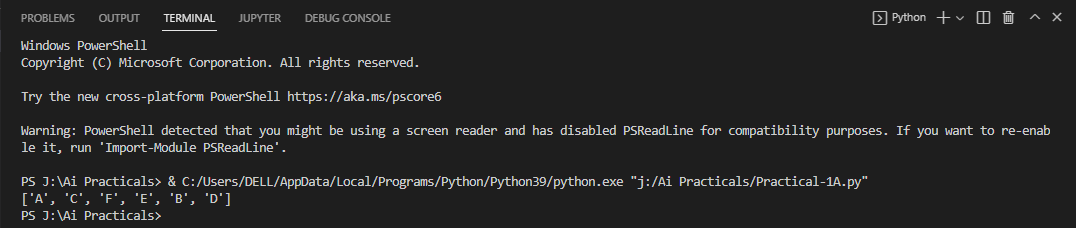
            dfs(graph,n, visited)

        return visited

visited = dfs(graph1,'A',[])

print(visited)

**Output:**



**1B - Write a program to implement breadth first search algorithm.**

**Code:**

import collections

import queue

def bfs(graph,root):

    visited = set()

    queue = collections.deque([root])

    while queue:

        vertex = queue.popleft()

        visited.add(vertex)

        for i in graph[vertex]:

            if i not in visited:

                queue.append(i)

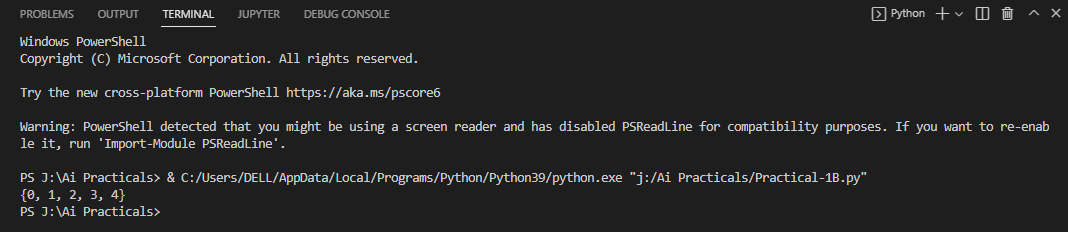
    print(visited)

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

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

        bfs(graph,0)

**Output :**

****

**2A - Write a program to simulate the 4-Queen / N-Queen problem.**

**Code:**

def is\_safe(board,row,col,n):

    for c in range (col,-1,-1):

        if board[row][c] == 1:

            return False

    i = row

    j = col

    while i >= 0 and j>=0:

        if board[i][j] == 1:

            return False

        i -= 14

        j -= 1

    i = row

    j = col

    while i < n and j>=0:

        if board[i][j] == 1:

            return False

        i += 1

        j -= 1

    return True

def nQueens(board,col,n):

    if col >= n:

        return True

    for i in range (n):

        if is\_safe(board,i,col,n):

            board[i][col] = 1

            if nQueens(board,col+1,n):

                return True

        board[i][col] = 0

    return False

n = int(input('Enter the size of the board :'))

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

if nQueens(board,0,n) == True:

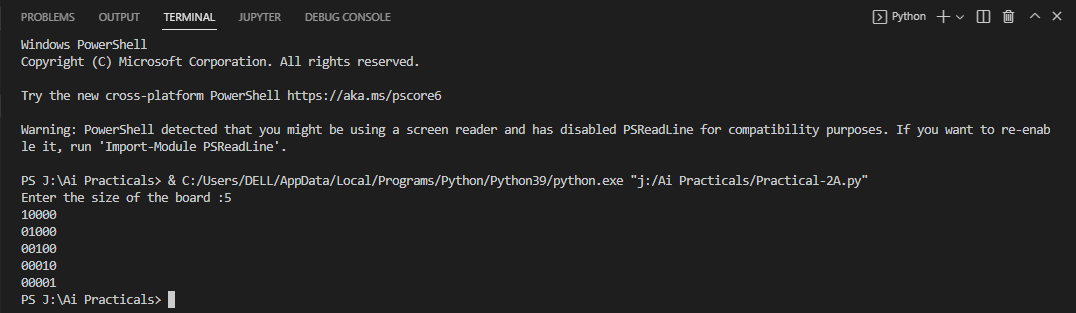
    for i in range(n):

        for j in range(n):

            print(board[i][j], end='')

        print()

**Output :**



**2B - Write a program to solve the tower of Hanoi problem**

**Code:**

def tower\_of\_hanoi(disks,source,auxilary,target):

    if(disks == 1):

        print('Move disk 1 from rod {} to rod {}.'.format(source,target))

        return

#Function call itself

    tower\_of\_hanoi(disks - 1,source,target,auxilary)

    print('Move disk {} from rod {} to rod {}.'.format(disks,source,target))

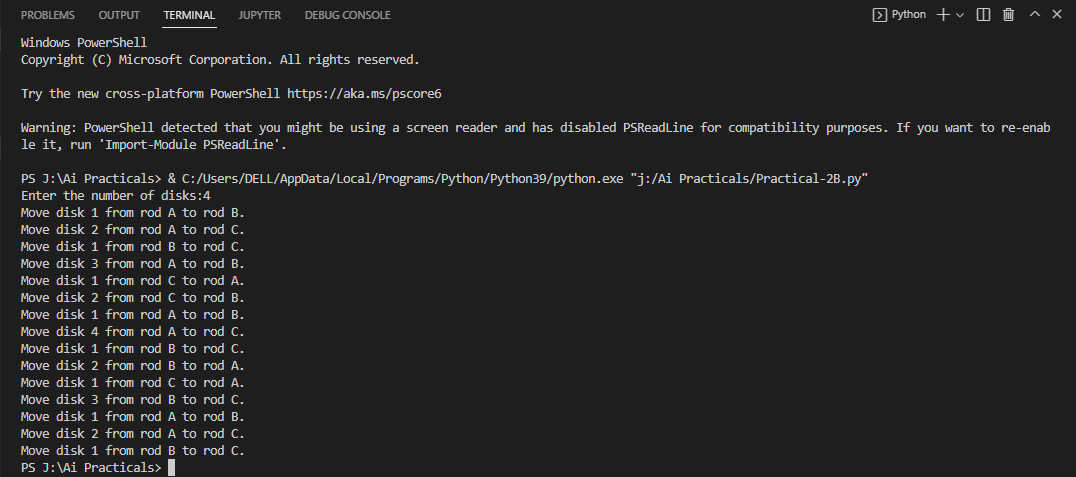
    tower\_of\_hanoi(disks - 1,auxilary,source,target)

disks = int(input('Enter the number of disks:'))

#We are referring source as A, auxilary as B, and target as C

tower\_of\_hanoi(disks,'A','B','C') #Calling the function

**Output :**

****

**3A - #Write a program to implement alpha beta search.**

**Code :**

maximum, minimum = 1000, -1000

def fun\_alphabeta(d, node, maxP, v, A, B):

    if d==3:

        return v[node]

    if maxP:

        best = minimum

        for i in range(0,2):

            value = fun\_alphabeta(d+1, node\*2+i, False, v, A, B)

            best = max(best, value)

            A = max(A, best)

            if B <= A:

                break

        return best

    else:

        best = maximum

        for i in range(0,2):

            value = fun\_alphabeta(d+1, node\*2+i, True, v, A, B)

            best = min(best, value)

            B = min(B, best)

            if B <= A:

                break

        return best

scr=[]

x=int(input("Enter total number of leaf node: "))

for i in range(x):

    y = int(input("Enter node values: "))

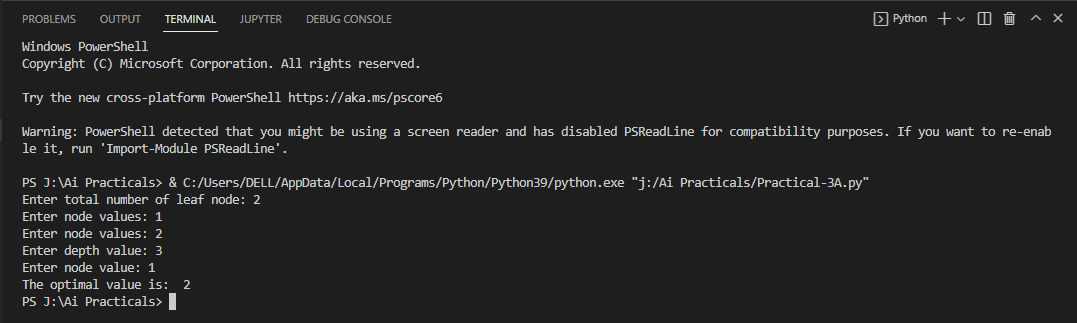
    scr.append(y)

d=int(input("Enter depth value: "))

node=int(input("Enter node value: "))

print("The optimal value is: ", fun\_alphabeta(d, node, True, scr, minimum, maximum))

**Output :**

****

**3B – Write a program for Hill climbing problem.**

**Code :**

import math

from sys import flags

increment = 0.1

startingPoint = [1, 1]

point1 = [1, 5]

point2 = [6, 4]

point3 = [5, 2]

point4 = [2, 1]

def distance(x1, y1, x2, y2):

    dist = math.pow(x2-x1, 2) + math.pow(y2-y1, 2)

    return dist

def sumOfDistances(x1,y1,px1,py1,px2,py2,px3,py3,px4,py4):

    d1 = distance(x1,y1,px1, py1)

    d2 = distance(x1, y1, px2, py2)

    d3 = distance(x1, y1, px3, py3)

    d4 = distance(x1, y1, px4, py4)

    return d1 + d2 + d3 + d4

def newDistance(x1, y1, point1, point2, point3, point4):

    d1 = [x1, y1]

    d1temp = sumOfDistances(x1, y1, point1[0], point2[1], point2[0], point2[1],

                            point3[0], point3[1], point4[0], point4[1])

    d1.append(d1temp)

    return d1

minDistance = sumOfDistances(startingPoint[0], startingPoint[1],

                    point1[0], point1[1], point2[0], point2[1], point3[0],

                    point3[1], point4[0], point4[1])

flag = True

def newPoints(minimum, d1, d2, d3, d4):

    if d1[2] == minimum:

        return [d1[0], d1[1]]

    elif d2[2] == minimum:

        return [d2[0], d2[1]]

    elif d3[2] == minimum:

        return [d3[0], d3[1]]

    elif d4[2] == minimum:

        return [d4[0], d4[1]]

i = 1

while flag:

    d1 = newDistance(startingPoint[0] +increment, startingPoint[1], point1, point2, point3, point4)

    d2 = newDistance(startingPoint[0] -increment, startingPoint[1], point1, point2, point3, point4)

    d3 = newDistance(startingPoint[0], startingPoint[1] +increment, point1, point2, point3, point4)

    d4 = newDistance(startingPoint[0], startingPoint[1] -increment, point1, point2, point3, point4)

    print(i, ' ', round(startingPoint[0], 2), round(startingPoint[1], 2))

    minimum = min(d1[2], d2[2], d3[2], d4[2])

    if minimum <minDistance:

        startingPoint = newPoints(minimum, d1, d2, d3, d4)

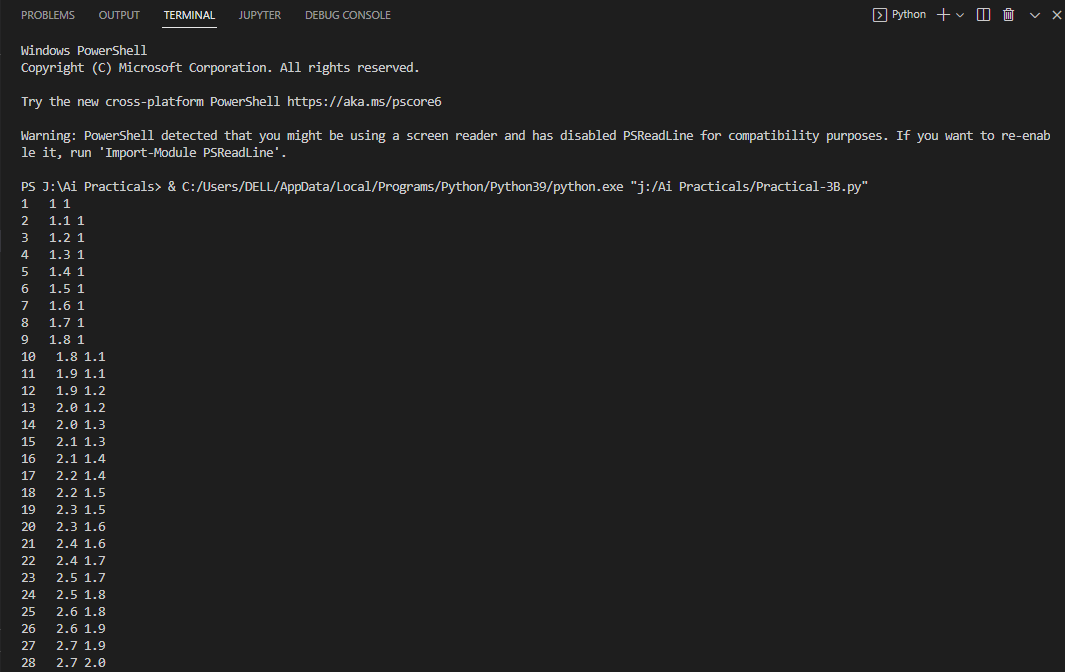
        minDistance = minimum

        i += 1

    else:

        flag = False

**Output :**

****

**4A - #Write a program to implement A\* algorithm.**

**Code :**

def aStarAlgo(start\_node, stop\_node):

    open\_set = set(start\_node)

    closed\_set = set()

    g = {}

    parents = {}

    g[start\_node] = 0

    parents[start\_node] = start\_node

    while len(open\_set) >0:

        n = None

        for v in open\_set:

            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

                n = v

        if n == stop\_node or Graph\_nodes[n] == None:

            pass

        else:

            for (m, weight) in get\_neighbors(n):

                if m not in open\_set and m not in closed\_set:

                    open\_set.add(m)

                    parents[m] = n

                    g[m] = g[n] + weight

                else:

                    if g[m] > g[n] + weight:

                        g[m] = g[n] + weight

                        parents[m] = n

                        if m in closed\_set:

                            closed\_set.remove(m)

                            open\_set.add(m)

        if n == None:

            print('Path does not exist!')

            return None

        if n == stop\_node:

            path = []

            while parents[n] != n:

                path.append(n)

                n = parents[n]

            path.append(start\_node)

            path.reverse()

            print('Path found: {}'.format(path))

            return path

        open\_set.remove(n)

        closed\_set.add(n)

    print('Path does not exist!')

    return None

def get\_neighbors(v):

    if v in Graph\_nodes:

        return Graph\_nodes[v]

    else:

        return None

def heuristic(n):

    H\_dist = {

        'A': 11,

        'B': 6,

        'C': 99,

        'D': 1,

        'E': 7,

        'G': 0,

    }

    return H\_dist[n]

Graph\_nodes = {

        'A': [('B', 2), ('E', 3)],

        'B': [('C', 1), ('G', 9)],

        'C': None,

        'E': [('D', 6)],

        'D': [('G', 1)],

    }

aStarAlgo('A', 'G')

**Output :**

****

**4B - Write a program to implement AO\* algorithm.**

**Code :**

def recAOStar(n):

    global finalPath

    print("Expanding Node:", n)

    and\_nodes = []

    or\_nodes = []

    if (n in allNodes):

        if 'AND' in allNodes[n]:

            and\_nodes = allNodes[n]['AND']

        if 'OR' in allNodes[n]:

            or\_nodes = allNodes[n]['OR']

    if len(and\_nodes) == 0 and len(or\_nodes) == 0:

        return

    solvable = False

    marked = {}

    while not solvable:

        if len(marked) == len(and\_nodes) + len(or\_nodes):

            min\_cost\_least, min\_cost\_group\_least = least\_cost\_group(and\_nodes, or\_nodes, {})

            solvable = True

            change\_heuristic(n, min\_cost\_least)

            optimal\_child\_group[n] = min\_cost\_group\_least

            continue

        min\_cost, min\_cost\_group = least\_cost\_group(and\_nodes, or\_nodes, marked)

        is\_expanded = False

        if len(min\_cost\_group) > 1:

            if (min\_cost\_group[0] in allNodes):

                is\_expanded = True

                recAOStar(min\_cost\_group[0])

            if (min\_cost\_group[1] in allNodes):

                is\_expanded = True

                recAOStar(min\_cost\_group[1])

        else:

            if (min\_cost\_group in allNodes):

                is\_expanded = True

                recAOStar(min\_cost\_group)

        if is\_expanded:

            min\_cost\_verify, min\_cost\_group\_verify = least\_cost\_group(and\_nodes, or\_nodes, {})

            if min\_cost\_group == min\_cost\_group\_verify:

                solvable = True

                change\_heuristic(n, min\_cost\_verify)

                optimal\_child\_group[n] = min\_cost\_group

        else:

            solvable = True

            change\_heuristic(n, min\_cost)

            optimal\_child\_group[n] = min\_cost\_group

        marked[min\_cost\_group] = 1

    return heuristic(n)

def least\_cost\_group(and\_nodes, or\_nodes, marked):

    node\_wise\_cost = {}

    for node\_pair in and\_nodes:

        if not node\_pair[0] + node\_pair[1] in marked:

            cost = 0

            cost = cost + heuristic(node\_pair[0]) + heuristic(node\_pair[1]) + 2

            node\_wise\_cost[node\_pair[0] + node\_pair[1]] = cost

    for node in or\_nodes:

        if not node in marked:

            cost = 0

            cost = cost + heuristic(node) + 1

            node\_wise\_cost[node] = cost

    min\_cost = 999999

    min\_cost\_group = None

    for costKey in node\_wise\_cost:

        if node\_wise\_cost[costKey] < min\_cost:

            min\_cost = node\_wise\_cost[costKey]

            min\_cost\_group = costKey

    return [min\_cost, min\_cost\_group]

def heuristic(n):

    return H\_dist[n]

def change\_heuristic(n, cost):

    H\_dist[n] = cost

    return

def print\_path(node):

    print(optimal\_child\_group[node], end="")

    node = optimal\_child\_group[node]

    if len(node) > 1:

        if node[0] in optimal\_child\_group:

            print("->", end="")

            print\_path(node[0])

        if node[1] in optimal\_child\_group:

            print("->", end="")

            print\_path(node[1])

    else:

        if node in optimal\_child\_group:

            print("->", end="")

            print\_path(node)

H\_dist = {

    'A': -1,

    'B': 4,

    'C': 2,

    'D': 3,

    'E': 6,

    'F': 8,

    'G': 2,

    'H': 0,

    'I': 0,

    'J': 0

}

allNodes = {

    'A': {'AND': [('C', 'D')], 'OR': ['B']},

    'B': {'OR': ['E', 'F']},

    'C': {'OR': ['G'], 'AND': [('H', 'I')]},

    'D': {'OR': ['J']}

}

optimal\_child\_group = {}

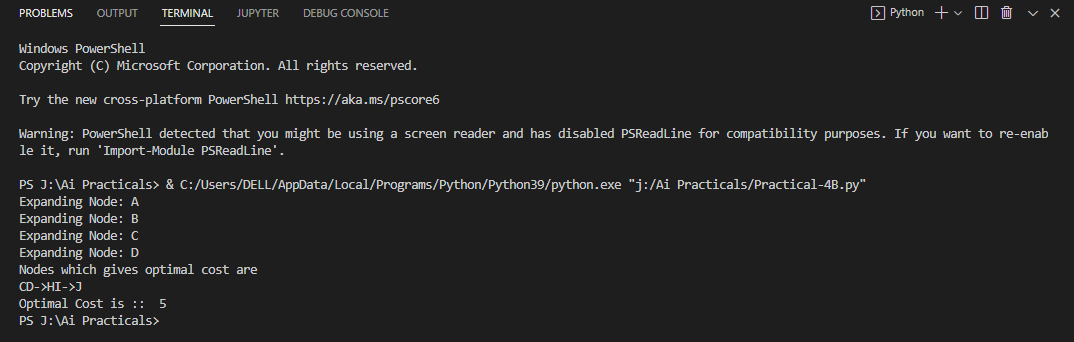
optimal\_cost = recAOStar('A')

print('Nodes which gives optimal cost are')

print\_path('A')

print('\nOptimal Cost is :: ', optimal\_cost)

**Output :**

****

**5A – Write a program to solve water jug problem.**

**Code :**

from operator import ge

capacity = (12,8,5)

x=capacity[0]

y=capacity[1]

z=capacity[2]

memory = {}

ans = []

def get\_all\_states(state):

    a=state[0]

    b=state[1]

    c=state[2]

    if(a==6 and b==6):

        ans.append(state)

        return True

    if ((a,b,c) in memory):

        return False

    memory[(a,b,c)]=1

    if (a>0):

        if(a+b<=y):

            if(get\_all\_states((0,a+b,c))):

                ans.append(state)

        else:

            if(get\_all\_states((a-(y-b),y,c))):

                ans.append(state)

                return True

        if(a+c<=z):

            if(get\_all\_states((0,b,a+c))):

                ans.append(state)

                return True

        else:

            if (get\_all\_states((a-(z-c),b,z))):

                ans.append(state)

                return True

    if(b>0):

        if(a+b<=x):

            if(get\_all\_states((a+b,0,c))):

                ans.append(state)

        else:

            if(get\_all\_states((x,b-(x-a),c))):

                ans.append(state)

                return True

        if(b+c<=z):

            if(get\_all\_states((a,0,b+c))):

                ans.append(state)

                return True

        else:

            if(get\_all\_states((a,b-(z-c),z))):

                return True

    if(c>0):

        if(a+c<=x):

            if(get\_all\_states((a+c,b,0))):

                ans.append(state)

                return True

        else:

            if(get\_all\_states((x,b,c-(x-a)))):

                ans.append(state)

                return True

        if(b+c<=y):

            if(get\_all\_states((a,b+c,0))):

                ans.append(state)

                return True

        else:

            if(get\_all\_states((a,y,c-(y- b)))):

                ans.append(state)

                return True

    return False

initial\_state = (12,0,0)

print("Starting work....\n")

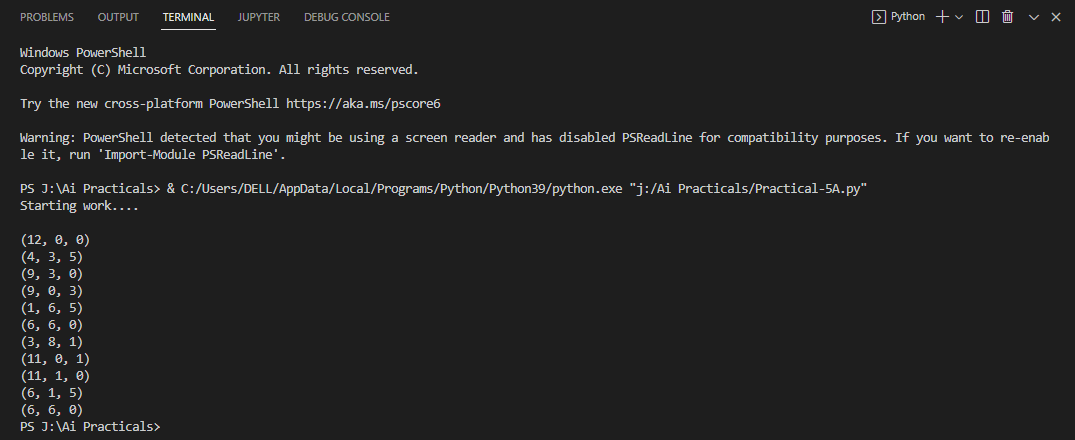
get\_all\_states(initial\_state)

ans.reverse()

for i in ans:

    print(i)

**Output :**

****

**5B - #Design the simulation of tic – tac – toe game using min-max algorithm.**

**Code :**

import os

import time

board = [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']

player = 1

########win Flags##########

Win = 1

Draw = -1

Running = 0

Stop = 1

###########################

Game = Running

Mark = 'X'

def DrawBoard():

    print(" %c | %c | %c " % (board[1], board[2], board[3]))

    print("\_\_\_|\_\_\_|\_\_\_")

    print(" %c | %c | %c " % (board[4], board[5], board[6]))

    print("\_\_\_|\_\_\_|\_\_\_")

    print(" %c | %c | %c " % (board[7], board[8], board[9]))

    print("   |   |   ")

def CheckPosition(x):

    if (board[x] == ' '):

        return True

    else:

        return False

def CheckWin():

    global Game

    if (board[1] == board[2] and board[2] == board[3] and board[1] != ' '):

        Game = Win

    elif (board[4] == board[5] and board[5] == board[6] and board[4] != ' '):

        Game = Win

    elif (board[7] == board[8] and board[8] == board[9] and board[7] != ' '):

        Game = Win

        # Vertical Winning Condition

    elif (board[1] == board[4] and board[4] == board[7] and board[1] != ' '):

        Game = Win

    elif (board[2] == board[5] and board[5] == board[8] and board[2] != ' '):

        Game = Win

    elif (board[3] == board[6] and board[6] == board[9] and board[3] != ' '):

        Game = Win

        # Diagonal Winning Condition

    elif (board[1] == board[5] and board[5] == board[9] and board[5] != ' '):

        Game = Win

    elif (board[3] == board[5] and board[5] == board[7] and board[5] != ' '):

        Game = Win

        # Match Tie or Draw Condition

    elif (board[1] != ' ' and board[2] != ' ' and board[3] != ' ' and board[4] != ' ' and board[5] != ' ' and board[

        6] != ' ' and board[7] != ' ' and board[8] != ' ' and board[9] != ' '):

        Game = Draw

    else:

        Game = Running

print("Tic-Tac-Toe Game Designed By Sourabh Somani")

print("Player 1 [X] --- Player 2 [O]\n")

print()

print()

print("Please Wait...")

time.sleep(3)

while (Game == Running):

    os.system('cls')

    DrawBoard()

    if (player % 2 != 0):

        print("Player 1's chance")

        Mark = 'X'

    else:

        print("Player 2's chance")

        Mark = 'O'

    choice = int(input("Enter the position between [1-9] where you want to mark : "))

    if (CheckPosition(choice)):

        board[choice] = Mark

        player += 1

        CheckWin()

os.system('cls')

DrawBoard()

if (Game == Draw):

    print("Game Draw")

elif (Game == Win):

    player -= 1

    if (player % 2 != 0):

        print("Player 1 Won")

    else:

        print("Player 2 Won")

**Output :**

****

**6A – #Write a program to solve Missionaries and Cannibals problem.**

**Code :**

print("\n")

print("\tGame Start\nNow the task is to move all of them to right side of the river")

print(

    "rules:\n1. The boat can carry at most two people\n2. If cannibals num greater then missionaries then the cannibals would eat the missionaries\n3. The boat cannot cross the river by itself with no people on board")

lM = 3  # lM = Left side Missionaries number

lC = 3  # lC = Laft side Cannibals number

rM = 0  # rM = Right side Missionaries number

rC = 0  # rC = Right side cannibals number

userM = 0  # userM = User input for number of missionaries for right to left side travel

userC = 0  # userC = User input for number of cannibals for right to left travel

k = 0

print("\nM M M C C C |     --- | \n")

try:

    while (True):

        while (True):

            print("Left side -> right side river travel")

            # uM = user input for number of missionaries for left to right travel

            # uC = user input for  number of cannibals for left to right travel

            uM = int(input("Enter number of Missionaries travel => "))

            uC = int(input("Enter number of Cannibals travel => "))

            if ((uM == 0) and (uC == 0)):

                print("Empty travel not possible")

                print("Re-enter : ")

            elif (((uM + uC) <= 2) and ((lM - uM) >= 0) and ((lC - uC) >= 0)):

                break

            else:

                print("Wrong input re-enter : ")

        lM = (lM - uM)

        lC = (lC - uC)

        rM += uM

        rC += uC

        print("\n")

        for i in range(0, lM):

            print("M ", end="")

        for i in range(0, lC):

            print("C ", end="")

        print("| --> | ", end="")

        for i in range(0, rM):

            print("M ", end="")

        for i in range(0, rC):

            print("C ", end="")

        print("\n")

        k += 1

        if (((lC == 3) and (lM == 1)) or ((lC == 3) and (lM == 2)) or ((lC == 2) and (lM == 1)) or (

                (rC == 3) and (rM == 1)) or ((rC == 3) and (rM == 2)) or ((rC == 2) and (rM == 1))):

            print("Cannibals eat missionaries:\nYou lost the game")

            break

        if ((rM + rC) == 6):

            print("You won the game : \n\tCongrats")

            print("Total attempt")

            print(k)

            break

        while (True):

            print("Right side -> Left side river travel")

            userM = int(input("Enter number of Missionaries travel => "))

            userC = int(input("Enter number of Cannibals travel => "))

            if ((userM == 0) and (userC == 0)):

                print("Empty travel not possible")

                print("Re-enter : ")

            elif (((userM + userC) <= 2) and ((rM - userM) >= 0) and ((rC - userC) >= 0)):

                break

            else:

                print("Wrong input re-enter : ")

        lM += userM

        lC += userC

        rM -= userM

        rC -= userC

        k += 1

        print("\n")

        for i in range(0, lM):

            print("M ", end="")

        for i in range(0, lC):

            print("C ", end="")

        print("| <-- | ", end="")

        for i in range(0, rM):

            print("M ", end="")

        for i in range(0, rC):

            print("C ", end="")

        print("\n")

        if (((lC == 3) and (lM == 1)) or ((lC == 3) and (lM == 2)) or ((lC == 2) and (lM == 1)) or (

                (rC == 3) and (rM == 1)) or ((rC == 3) and (rM == 2)) or ((rC == 2) and (rM == 1))):

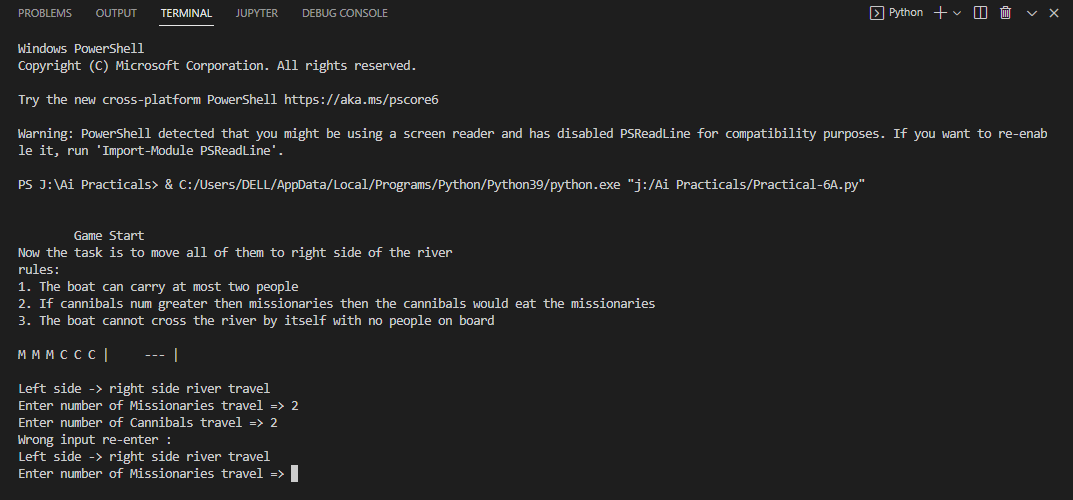
            print("Cannibals eat missionaries:\nYou lost the game")

            break

except EOFError as e:

    print("\nInvalid input please retry !!")

**Output :**

****

**6B - Design an application to simulate number puzzle problem.**

**Code:**

from \_future\_ import print\_function

from simpleai.search import astar, SearchProblem

from simpleai.search.viewers import WebViewer

GOAL = '''1-2-3

4-5-6

7-8-e'''

INITIAL = '''4-1-2

7-e-3

8-5-6'''

def list\_to\_string(list\_):

return '\n'.join(['-'.join(row) for row in list\_])

def string\_to\_list(string\_):

return [row.split('-') for row in string\_.split('\n')]

def find\_location(rows, element\_to\_find):

'''Find the location of a piece in the puzzle.

Returns a tuple: row, column'''

for ir, row in enumerate(rows):

for ic, element in enumerate(row):

if element == element\_to\_find:

return ir, ic

goal\_positions = {}

rows\_goal = string\_to\_list(GOAL)

for number in '12345678e':

goal\_positions[number] = find\_location(rows\_goal, number)

class EigthPuzzleProblem(SearchProblem):

def actions(self, state):

'''Returns a list of the pieces we can move to the empty space.'''

rows = string\_to\_list(state)

row\_e, col\_e = find\_location(rows, 'e')

actions = []

if row\_e > 0:

actions.append(rows[row\_e - 1][col\_e])

if row\_e < 2:

actions.append(rows[row\_e + 1][col\_e])

if col\_e > 0:

actions.append(rows[row\_e][col\_e - 1])

if col\_e < 2:

actions.append(rows[row\_e][col\_e + 1])

return actions

def result(self, state, action):

'''Return the resulting state after moving a piece to the empty space.

(the "action" parameter contains the piece to move)

'''

rows = string\_to\_list(state)

row\_e, col\_e = find\_location(rows, 'e')

row\_n, col\_n = find\_location(rows, action)

rows[row\_e][col\_e], rows[row\_n][col\_n] = rows[row\_n][col\_n], rows[row\_e][col\_e]

return list\_to\_string(rows)

def is\_goal(self, state):

'''Returns true if a state is the goal state.'''

return state == GOAL

def cost(self, state1, action, state2):

'''Returns the cost of performing an action. No useful on this problem, i

but needed.

'''

return 1

def heuristic(self, state):

'''Returns an estimation of the distance from a state to the goal.

We are using the manhattan distance.

'''

rows = string\_to\_list(state)

distance = 0

for number in '12345678e':

row\_n, col\_n = find\_location(rows, number)

row\_n\_goal, col\_n\_goal = goal\_positions[number]

distance += abs(row\_n - row\_n\_goal) + abs(col\_n - col\_n\_goal)

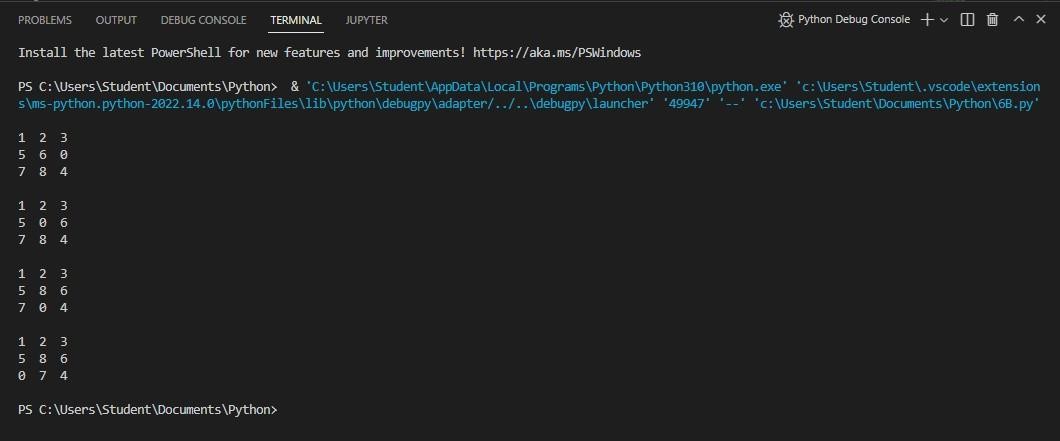
return distance

result = astar(EigthPuzzleProblem(INITIAL))

for action, state in result.path():

print('Move number', action)

print(state)

**Output :**

**7A - #Write a program to shuffle Deck of cards.**

**Code:**

import itertools, random

deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))

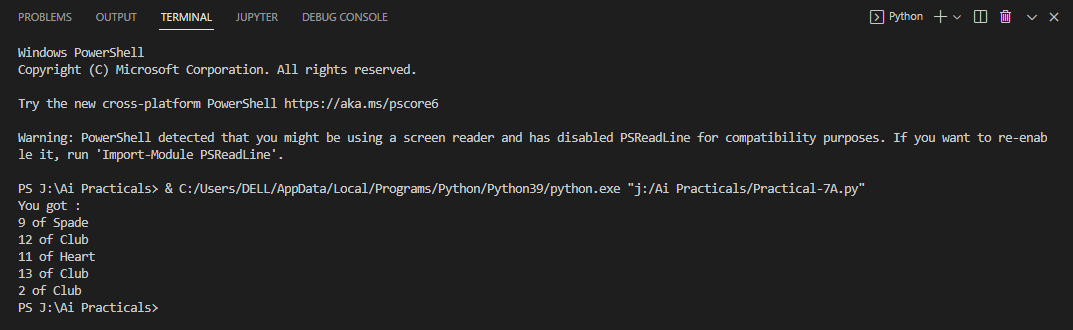
random.shuffle(deck)

print("You got : ")

for i in range(5):

    print(deck[i][0], "of", deck[i][1])

**Output :**

****

**7B – Solve traveling salesman problem using artificial intelligence technique.**

**Code :**

from sys import maxsize

from itertools import permutations

V=4

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

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

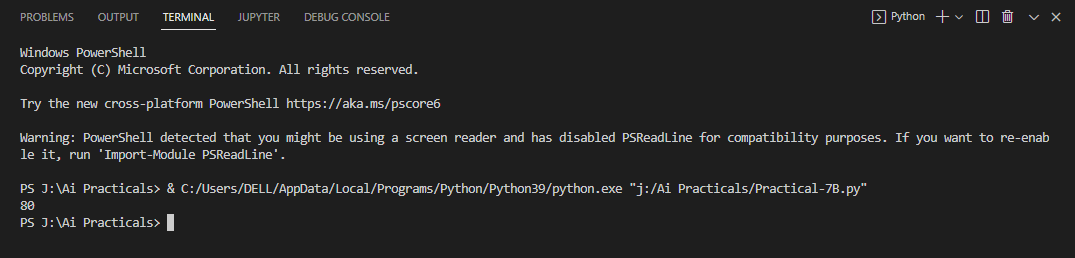
    graph = [[0,10,15,20], [10,0,35,25],

             [15,35,0,30],[20,25,30,0]]

    s = 0

    print(travellingSalesmanProblem(graph, s))

**Output :**

****

**8A – #Solve the block of World problem.**

**Code :**

class  point:

    def \_\_init\_\_(self, x, y):

        self.x = x

        self.y = y

class line:

    def \_\_init\_\_(self, a, b, c):

        self.a = a

        self.b = b

        self.c = c

def evalPointOnline(p, curLine):

    eval = curLine.a\*p.x + curLine.b\*p.y + curLine.c

    if (eval > 0):

        return 1

    return -1

def minJumpToReachDestination(start, dest, lines, N):

    jumps = 0

    for i in range(N):

        signStart = evalPointOnline(start, lines[i])

        signDest = evalPointOnline(dest, lines[i])

        if (signStart \* signDest < 0):

            jumps = jumps+1

    return jumps

start = point(1, 1)

dest = point(-2, -1)

lines = []

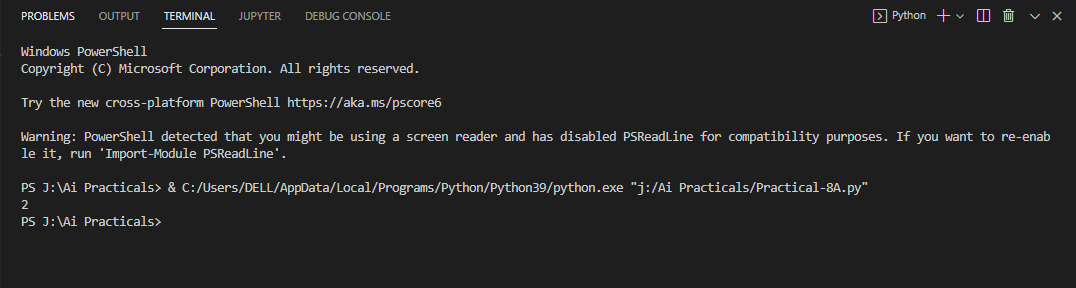
lines.append(line(1,0,0))

lines.append(line(0,1,0))

lines.append(line(1,1,-2))

print(minJumpToReachDestination(start, dest, lines, 3))

**Output :**

****

**8B – #Solve constraint satisfaction problem**

**Code :**

import constraint

problem = constraint.Problem()

problem.addVariable('x', [1,2,3])

problem.addVariable('y', range(10))

def our\_constraint(x,y):

    if x+y>=5:

        return True

problem.addConstriant(our\_constraint, ['x','y'])

solutions = problem.getSolutions()

length = len(solutions)

print("(x,y) ∈ {", end = "")

for index, solution in enumerate(solutions):

    if index == length - 1:

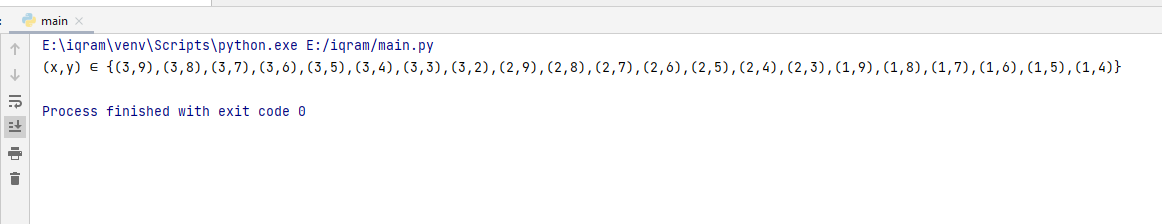
        print("({},{})".format(solution['x'], solution['y']), end = "")

    else:

        print("({},{})".format(solution['x'], solution['y']), end = "")

print("}")

**Output :**

****

**9A – Derive the expressions based on Associative law**

**Code :**

meal = "fruit"

money = 0

if meal == "fruit" or meal == "sandwich" and money >= 2:

    print("Lunch being delivered")

else:

    print("Can't deliver lunch")

meal = "fruit"

money = 0

if (meal == "fruit" or meal == "sandwich") and money >= 2:

    print("Lunch being delivered")

else:

    print("Can't deliver lunch")

print(5\*2//3)

print(5 \*(2 // 3))

expr = 10+20\*30

print(expr)

name = "Alex"

age = 0

if name == "Alex" or name == "John" and age >= 2:

    print("Hello Welcome !!!")

else:

    print("Good Bye !!!")

name = "Alex"

age = 0

if (name == "Alex" or name == "John") and age >= 2:

    print("Hello Welcome !!!")

else:

    print("Good Bye !!!")

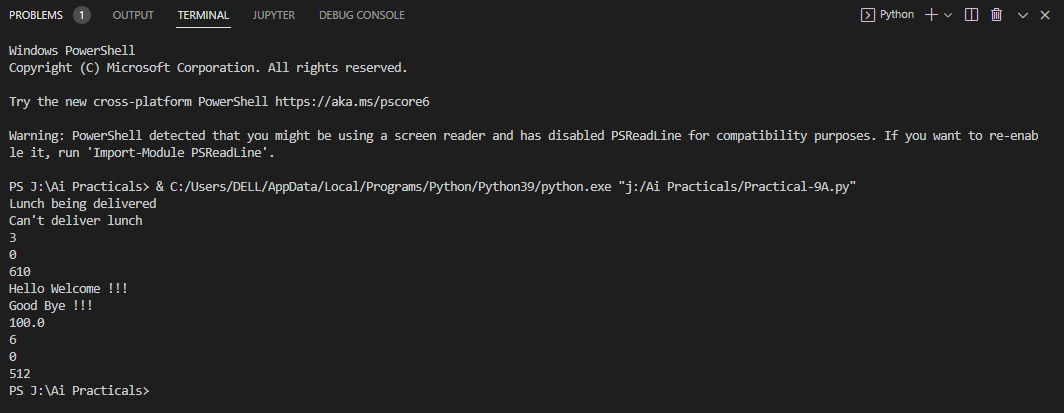
print(100/10\*10)

print(5-2+3)

print(5-(2+3))

print(2\*\*3\*\*2)

**Output :**



**10B - #Write a program to derive the predicate.**

**Code :**

import heapq as hq

my\_dict = {'z': 'zebra', 'b': 'ball', 'w': 'whale',

           'a': 'apple', 'm': 'monkey', 'c': 'cat'}

my\_list = [(k, v) for k, v in my\_dict.items()]

print("Before organizing as heap :", my\_list)

hq.heapify(my\_list)

print("After organizing as heap :", my\_list)

my\_dict = dict(my\_list)

print("Resultant dictionary :", my\_dict)

import heapq as hq

my\_dict = {'z': 'zebra', 'b': 'ball', 'w': 'whale',

           'a': 'apple', 'm': 'monkey', 'c': 'cat'}

my\_list = [(k, v) for k, v in my\_dict.items()]

print("Before organizing as heap :", my\_list)

hq.heapify(my\_list)

print("After organizing as heap :", my\_list)

my\_dict = dict(my\_list)

print("Resultant dictionary :", my\_dict)

import heapq as hq

my\_dict = [{'z': 'zebra'}, {'b': 'ball'}, {'w': 'whale'},

           {'a': 'apple'}, {'m': 'monkey'}, {'c': 'cat'}]

my\_list = [(k, v) for i in my\_dict for k, v in i.items()]

print("Before organizing as heap :", my\_list)

hq.heapify(my\_list)

print("After organizing as heap :", my\_list)

my\_dict = dict(my\_list)

print("Resultant dictionary :", my\_dict)

import heapq as hq

class employee:

    def \_init\_(self, n, d, yos, s):

        self.name = n

        self.des = d

        self.yos = yos

        self.sal = s

    def print\_me(self):

        print("Name :", self.name)

        print("Designation :", self.des)

        print("Years of service :", str(self.yos))

        print("salary :", str(self.sal))

    def \_lt\_(self, nxt):

        return self.yos < nxt.yos

e1 = employee('Yuvraj', 'manager', 3, 24000)

e2 = employee('Shaban', 'programmer', 2, 15000)

e3 = employee('Vinith', 'Analyst', 5, 30000)

e4 = employee('Iqram', 'programmer', 1, 10000)

emp = [e1, e2, e3, e4]

hq.heapify(emp)

for i in range(0, len(emp)):

    emp[i].print\_me()

    print()

**Output :**

