NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY
GREATER NOIDA-201306
(An Autonomous Institute)
School of Computer Sciences & Engineering in Emerging Technologies

Introduction to Artificial Intelligence Lab (ACSAI-0351)

INDEX

S. No.	PRACTICAL CONDUCTED	DATE	PAGE NO	SIGNATURE
1	Write a Program to implement an AI chatbot	08/09/21		
2	Write a program to perform the TIK TAK TOE Problem	15/09/21		
3	Write a Program to perform Breadth first search	22/09/21		
4	Write a Program to perform Depth first search	22/09/21		
5	Write a Program to perform Water Jug Problem	01/10/21		
6	Write a Program to perform Simple Calculator	08/10/21		
7	Write a program to perform the N Queen problem	29/10/21		
8	Write a Program to perform Best first search	26/11/21		
9	Write a Program to Implement Alpha Beta Pruning graphically	03/12/21		
10	Write Program to implement Hill-Climbing Algorithm Using Heuristic Search Techniques	10/12/21		
11	Write a program to implement Hangman game using python.			
12	Write a program to solve the Monkey Banana problem			
13	Solve Robot (traversal) problem using means End Analysis.			
14	Implementation of Image features Processing using OPENCV AND OPEN VINO			
15	Write a program to implement Naïve Bayes Algorithm			

Program No-1

AIM: Write a Program to implement an AI chatbot

```
IMPORT NLTK

from nltk.chat.util import Chat,reflections

reflections = {
   "i am" : "you are",
   "i was":"you were",
   "i":"you",
   "i'm ":"you are",
   "i'd ":"you would",
   "i've ":"you have",
   "i'll ":"you will",
   "my":"your",
   "you are":"I am",
   "you were":"I was",
   "you've ":"I have",
   "you'll ":"I will",
   "your":"my",
   "yours":"mine",
   "you":"me",
   "me":"you",
}

pairs = [
   [
      r"my name is (.*)",
      ["Hello %1, How are you today ?"]
   ],
   [
      r"hi|hey|hello",
      ["Hello, Hey there"]
   ],
   [
      r"what is your name ?",
      ["I am bot created by HARSH, you can call me a Genius bot"]
   ],
   [
      r"sorry (.*) ",
      ["Ita alright, Its Ok, never mind"]
   ],
   [
      r"I am fine",
      ["Great to hear that, How can i help you ?"]
   ],
   [
```

```
        r"i'am (.*)doing good",
        ["Nice to know that , How can I help you ?"]
    ],
    [
        r" ",
        [" "]
    ],
    [
        r" ",
        [" "]
    ],
    [
        r" ",
        [" "]
    ],
    [
        r" ",
        [" "]
    ],
]
def chat():
  print("Hi! I am a chatbot created by Harsh for you service")
  chat = Chat(pairs, reflections)
  chat.converse()
if __name__ == "__main__":
  chat()
```

OUTPUT:

```
Hi! I am a chatbot created by Harsh for you service
>hi
Hello, Hey there
>hey
Hello, Hey there
>how are you
None
>what is your name
I am bot created by Harsh, you can call me a Genius bot
>sorry
None
>quit
None
```

Program No-2

AIM: Write a program to perform the TIK TAK TOE Problem

```
# Tic-Tac-Toe Program using
# random number in Python
import numpy as np
import random
from time import sleep

# Creates an empty board
def create_board():
 return(np.array([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]]))

# Check for empty places on board
def possibilities(board):
 l = []

 for i in range(len(board)):
  for j in range(len(board)):

   if board[i][j] == 0:
    l.append((i, j))
 return(l)

# Select a random place for the player
def random_place(board, player):
 selection = possibilities(board)
 current_loc = random.choice(selection)
 board[current_loc] = player
 return(board)

# Checks whether the player has three
# of their marks in a horizontal row
def row_win(board, player):
 for x in range(len(board)):
  win = True

  for y in range(len(board)):
   if board[x, y] != player:
    win = False
    continue

  if win == True:
   return(win)
 return(win)
```

```
# Checks whether the player has three
# of their marks in a vertical row
def col_win(board, player):
  for x in range(len(board)):
    win = True

    for y in range(len(board)):
      if board[y][x] != player:
        win = False
        continue

    if win == True:
      return(win)
  return(win)

# Checks whether the player has three
# of their marks in a diagonal row
def diag_win(board, player):
  win = True
  y = 0
  for x in range(len(board)):
    if board[x, x] != player:
      win = False
  if win:
    return win
  win = True
  if win:
    for x in range(len(board)):
      y = len(board) - 1 - x
      if board[x, y] != player:
        win = False
  return win

# Evaluates whether there is
# a winner or a tie
def evaluate(board):
  winner = 0

  for player in [1, 2]:
    if (row_win(board, player) or
      col_win(board,player) or
      diag_win(board,player)):

      winner = player

  if np.all(board != 0) and winner == 0:
    winner = -1
  return winner

# Main function to start the game
```

```
def play_game():
  board, winner, counter = create_board(), 0, 1
  print(board)
  sleep(2)

  while winner == 0:
    for player in [1, 2]:
      board = random_place(board, player)
      print("Board after " + str(counter) + " move")
      print(board)
      sleep(2)
      counter += 1
      winner = evaluate(board)
      if winner != 0:
        break
  return(winner)

# Driver Code
print("Winner is: " + str(play_game()))
```

OUTPUT:

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 0 0]
 [0 0 0]
 [1 0 0]]
Board after 2 move
[[2 0 0]
 [0 0 0]
 [1 0 0]]
Board after 3 move
[[2 0 0]
 [0 0 0]
 [1 1 0]]
Board after 4 move
[[2 0 0]
 [0 2 0]
 [1 1 0]]
Board after 5 move
[[2 0 1]
 [0 2 0]
 [1 1 0]]
Board after 6 move
[[2 0 1]
 [0 2 0]
 [1 1 2]]
Winner is: 2
```

Program No-3

AIM: Write a Program to perform Breadth first search

```
# BREADTH FIRST SEARCH
graph = {
   'A' : ['B','C'],
   'B' : ['D','E'],
   'C' : ['F'],
   'D' : [],
   'E' : ['F'],
   'F' : []
 }
visited =[]
queue = []
def bfs(visited,graph,node):
  visited.append(node)
  queue.append(node)
  while queue:
    s = queue.pop(0)
    print(s,end=" ")
    for neighbour in graph[s]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
bfs(visited,graph,'A')
```

OUTPUT:

A B C D E F

Program No-4

AIM: Write a Program to perform Depth first search

```
#Depth First Search
graph = {
   'A' : ['B','C'],
   'B' : ['D','E'],
   'C' : ['F'],
   'D' : [],
   'E' : ['G'],
   'F' : [],
   'G' : [],
}

visited =[]
queue = []
def dfs(visited,graph,node):
  visited.append(node)
  queue.insert(0,node)
  while(queue):
    s = queue.pop(0)
    print(s,end=" ")
    for next in graph[s]:
      if next not in visited:
        queue.insert(0,next)
        visited.append(next)
dfs(visited,graph,'A')
```

OUTPUT:

A C F B E G D

Program No-5

AIM: Write a Program to perform Water Jug Problem

```
# WATER JUG PROBLEM
def pour(jug1,jug2):

  max1,max2, fill= 3, 4, 2
  print("%d\t%d" % (jug1,jug2))
  if jug2 is fill:
    return
  elif jug2 is max2:
    pour(0,jug1)
  elif jug1 !=0 and jug2 is 0:
    pour(0,jug1)
  elif jug1 is fill:
    pour(jug1,0)
  elif jug1 < max1:
    pour(max1,jug2)
  elif jug1 < (max2-jug2):
    pour(0,(jug1+jug2))
  else:
    pour(jug1-(max2-jug2),(max2-jug2)+jug2)
print("JUG1\tJUG2")
pour(0,0)
```

OUTPUT:

```
JUG1	JUG2
0	0
3	0
0	3
3	3
2	4
0	2
```

Program No-6

AIM: Write a Program to perform Simple Calculator

```
# Python program for simple calculator
# Function to add two numbers
def add(num1, num2):
    return num1 + num2

# Function to subtract two numbers
def subtract(num1, num2):
    return num1 - num2

# Function to multiply two numbers
def multiply(num1, num2):
    return num1 * num2

# Function to divide two numbers
def divide(num1, num2):
    return num1 / num2

print("Please select operation -\n" \
        "1. Add\n" \
        "2. Subtract\n" \
        "3. Multiply\n" \
        "4. Divide\n")


# Take input from the user
select = int(input("Select operations form 1, 2, 3, 4 :"))

number_1 = int(input("Enter first number: "))
number_2 = int(input("Enter second number: "))

if select == 1:
    print(number_1, "+", number_2, "=",
                    add(number_1, number_2))

elif select == 2:
    print(number_1, "-", number_2, "=",
                    subtract(number_1, number_2))

elif select == 3:
    print(number_1, "*", number_2, "=",
                    multiply(number_1, number_2))

elif select == 4:
    print(number_1, "/", number_2, "=",
                    divide(number_1, number_2))
```

```
else:
   print("Invalid input")
```

OUTPUT:

```
Please select operation -
1. Add
2. Subtract
3. Multiply
4. Divide

Select operations form 1, 2, 3, 4 :1
Enter first number: 23
Enter second number: 45
23 + 45 = 68
```

Program No-7

AIM: Write a program to perform the N Queen problem.

```
#queen n problem
# Function to check if two queens threaten each other or not
def isSafe(mat, r, c):
  # return false if two queens share the same column
  for i in range(r):
    if mat[i][c] == 'Q':
      return False
  # return false if two queens share the same `\` diagonal
  (i, j) = (r, c)
  while i >= 0 and j >= 0:
    if mat[i][j] == 'Q':
      return False
    i = i - 1
    j = j - 1
  # return false if two queens share the same `/` diagonal
  (i, j) = (r, c)
  while i >= 0 and j < len(mat):
    if mat[i][j] == 'Q':
     return False
    i = i - 1
    j = j + 1
  return True
def printSolution(mat):
  for r in mat:
    print(str(r).replace(',', '').replace('\'', ''))
  print()
def nQueen(mat, r):
  # if `N` queens are placed successfully, print the solution
  if r == len(mat):
    printSolution(mat)
    return
  # place queen at every square in the current row `r`
  # and recur for each valid movement
  for i in range(len(mat)):
    # if no two queens threaten each other
    if isSafe(mat, r, i):
      # place queen on the current square
      mat[r][i] = 'Q'
      # recur for the next row
      nQueen(mat, r + 1)
      # backtrack and remove the queen from the current square
      mat[r][i] = '–'
if __name__ == '__main__':
  # `N × N` chessboard
  N = 4
```

```
# `mat[][]` keeps track of the position of queens in
# the current configuration
mat = [['–' for x in range(N)] for y in range(N)]
nQueen(mat, 0)
```

OUTPUT:

```
[– Q – –]
[– – – Q]
[Q – – –]
[– – Q –]

[– – Q –]
[Q – – –]
[– – – Q]
[– Q – –]
```

Program No-8

AIM: Write a Program to perform Best first search

```
graph={
"A":[["B",3],["C",2]],
"B":[["D",2],["E",3]],
"C":[],
"D":[],
"E":[]
}
l=[]
def BTS(graph ,l,node):
  q=[]
  print(node,end=" ")
  l.append(node)
  for i in graph[node]:
    q.append(tuple(i))
    q=sorted(q,key=lambda x:x[1])
    while q:
      s=q.pop(0)
      print(s[0],end=" ")
      for i in graph[s[0]]:
        if i[0] not in l:
          l.append(i[0])
          q.append(i)
          q=sorted(q,key=lambda x:x[1])
BTS(graph,l,"A")
```

OUTPUT:

A B D E C

Program No-9

AIM: Write a Program to Implement Alpha Beta Pruning graphically

```
# Python3 program to demonstrate
# working of Alpha-Beta Pruning
MAX, MIN = 1000, -1000
def minimax(depth, nodeIndex, maximizingPlayer,
			values, alpha, beta):
	if depth == 3:
		return values[nodeIndex]
	if maximizingPlayer:
		best = MIN
		for i in range(0, 2):
			val = minimax(depth + 1, nodeIndex * 2 + i,
						False, values, alpha, beta)
			best = max(best, val)
			alpha = max(alpha, best)
			if beta <= alpha:
				break
		return best
	else:
		best = MAX
		for i in range(0, 2):
			val = minimax(depth + 1, nodeIndex * 2 + i,
							True, values, alpha, beta)
			best = min(best, val)
			beta = min(beta, best)
			if beta <= alpha:
				break
		return best
if __name__ == "__main__":
	values = [3, 5, 6, 9, 1, 2, 0, -1]
	print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))
```

OUTPUT

The optimal value is : 5

Program No-10

AIM: Write Program to implement Hill-Climbing Algorithm Using Heuristic Search Techniques

```
import math
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp
def create_data_model():
   """Stores the data for the problem."""
   data = {}
   # Locations in block units
   data['locations'] = [
      (288, 149), (288, 129), (270, 133), (256, 141), (256, 157), (246, 157),
      (236, 169), (228, 169), (228, 161), (220, 169), (212, 169), (204, 169),
      (196, 169), (188, 169), (196, 161), (188, 145), (172, 145), (164, 145),
      (156, 145), (148, 145), (140, 145), (148, 169), (164, 169), (172, 169),
      (156, 169), (140, 169), (132, 169), (124, 169), (116, 161), (104, 153),
      (104, 161), (104, 169), (90, 165), (80, 157), (64, 157), (64, 165),
      (56, 169), (56, 161), (56, 153), (56, 145), (56, 137), (56, 129),
      (56, 121), (40, 121), (40, 129), (40, 137), (40, 145), (40, 153),
      (40, 161), (40, 169), (32, 169), (32, 161), (32, 153), (32, 145),
      (32, 137), (32, 129), (32, 121), (32, 113), (40, 113), (56, 113),
      (56, 105), (48, 99), (40, 99), (32, 97), (32, 89), (24, 89),
      (16, 97), (16, 109), (8, 109), (8, 97), (8, 89), (8, 81),
      (8, 73), (8, 65), (8, 57), (16, 57), (8, 49), (8, 41),
      (24, 45), (32, 41), (32, 49), (32, 57), (32, 65), (32, 73),
      (32, 81), (40, 83), (40, 73), (40, 63), (40, 51), (44, 43),
      (44, 35), (44, 27), (32, 25), (24, 25), (16, 25), (16, 17),
      (24, 17), (32, 17), (44, 11), (56, 9), (56, 17), (56, 25),
      (56, 33), (56, 41), (64, 41), (72, 41), (72, 49), (56, 49),
      (48, 51), (56, 57), (56, 65), (48, 63), (48, 73), (56, 73),
      (56, 81), (48, 83), (56, 89), (56, 97), (104, 97), (104, 105),
      (104, 113), (104, 121), (104, 129), (104, 137), (104, 145), (116, 145),
      (124, 145), (132, 145), (132, 137), (140, 137), (148, 137), (156, 137),
      (164, 137), (172, 125), (172, 117), (172, 109), (172, 101), (172, 93),
      (172, 85), (180, 85), (180, 77), (180, 69), (180, 61), (180, 53),
      (172, 53), (172, 61), (172, 69), (172, 77), (164, 81), (148, 85),
      (124, 85), (124, 93), (124, 109), (124, 125), (124, 117), (124, 101),
      (104, 89), (104, 81), (104, 73), (104, 65), (104, 49), (104, 41),
      (104, 33), (104, 25), (104, 17), (92, 9), (80, 9), (72, 9),
      (64, 21), (72, 25), (80, 25), (80, 25), (80, 41), (88, 49),
```

```
        (104, 57), (124, 69), (124, 77), (132, 81), (140, 65), (132, 61),
        (124, 61), (124, 53), (124, 45), (124, 37), (124, 29), (132, 21),
        (124, 21), (120, 9), (128, 9), (136, 9), (148, 9), (162, 9),
        (156, 25), (172, 21), (180, 21), (180, 29), (172, 29), (172, 37),
        (172, 45), (180, 45), (180, 37), (188, 41), (196, 49), (204, 57),
        (212, 65), (220, 73), (228, 69), (228, 77), (236, 77), (236, 69),
        (236, 61), (228, 61), (228, 53), (236, 53), (236, 45), (228, 45),
        (228, 37), (236, 37), (236, 29), (228, 29), (228, 21), (236, 21),
        (252, 21), (260, 29), (260, 37), (260, 45), (260, 53), (260, 61),
        (260, 69), (260, 77), (276, 77), (276, 69), (276, 61), (276, 53),
        (284, 53), (284, 61), (284, 69), (284, 77), (284, 85), (284, 93),
        (284, 101), (288, 109), (280, 109), (276, 101), (276, 93), (276, 85),
        (268, 97), (260, 109), (252, 101), (260, 93), (260, 85), (236, 85),
        (228, 85), (228, 93), (236, 93), (236, 101), (228, 101), (228, 109),
        (228, 117), (228, 125), (220, 125), (212, 117), (204, 109), (196, 101),
        (188, 93), (180, 93), (180, 101), (180, 109), (180, 117), (180, 125),
        (196, 145), (204, 145), (212, 145), (220, 145), (228, 145), (236, 145),
        (246, 141), (252, 125), (260, 129), (280, 133)
    ]
 # yapf: disable
    data['num_vehicles'] = 1
    data['depot'] = 0
    return data
def compute_euclidean_distance_matrix(locations):
    """Creates callback to return distance between points."""
    distances = {}
    for from_counter, from_node in enumerate(locations):
        distances[from_counter] = {}
        for to_counter, to_node in enumerate(locations):
            if from_counter == to_counter:
                distances[from_counter][to_counter] = 0
            else:
                # Euclidean distance
                distances[from_counter][to_counter] = (int(
                    math.hypot((from_node[0] - to_node[0]),
                               (from_node[1] - to_node[1]))))
    return distances
def print_solution(manager, routing, solution):
    """Prints solution on console."""
    print('Objective: {}'.format(solution.ObjectiveValue()))
    index = routing.Start(0)
    plan_output = 'Route:\n'
    route_distance = 0
```

```
    while not routing.IsEnd(index):
        plan_output += ' {} ->'.format(manager.IndexToNode(index))
        previous_index = index
        index = solution.Value(routing.NextVar(index))
        route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
    plan_output += ' {}\n'.format(manager.IndexToNode(index))
    print(plan_output)
    plan_output += 'Objective: {}m\n'.format(route_distance)

def main():
    """Entry point of the program."""
    # Instantiate the data problem.
    data = create_data_model()

    # Create the routing index manager.
    manager = pywrapcp.RoutingIndexManager(len(data['locations']),
    data['num_vehicles'], data['depot'])

    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)

    distance_matrix = compute_euclidean_distance_matrix(data['locations'])
    def distance_callback(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return distance_matrix[from_node][to_node]
      transit_callback_index = routing.RegisterTransitCallback(distance_callback)

    # Define cost of each arc.
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

    # Solve the problem.
    solution = routing.SolveWithParameters(search_parameters)

    # Print solution on console.
    if solution:
```

```
        print_solution(manager, routing, solution)
if __name__ == '__main__':
    main()
```

OUTPUT:

Objective: 2790

Route:

0 -> 1 -> 279 -> 2 -> 278 -> 277 -> 248 -> 247 -> 243 -> 242 -> 241 -> 240 -> 239 -> 238 -> 245 -> 244 -> 246 -> 249 -> 250 -> 229 -> 228 -> 231 -> 230 -> 237 -> 236 -> 235 -> 234 -> 233 -> 232 -> 227 -> 226 -> 225 -> 224 -> 223 -> 222 -> 218 -> 221 -> 220 -> 219 -> 202 -> 203 -> 204 -> 205 -> 207 -> 206 -> 211 -> 212 -> 215 -> 216 -> 217 -> 214 -> 213 -> 210 -> 209 -> 208 -> 251 -> 254 -> 255 -> 257 -> 256 -> 253 -> 252 -> 139 -> 140 -> 141 -> 142 -> 143 -> 199 -> 201 -> 200 -> 195 -> 194 -> 193 -> 191 -> 190 -> 189 -> 188 -> 187 -> 163 -> 164 -> 165 -> 166 -> 167 -> 168 -> 169 -> 171 -> 170 -> 172 -> 105 -> 106 -> 104 -> 103 -> 107 -> 109 -> 110 -> 113 -> 114 -> 116 -> 117 -> 61 -> 62 -> 63 -> 65 -> 64 -> 84 -> 85 -> 115 -> 112 -> 86 -> 83 -> 82 -> 87 -> 111 -> 108 -> 89 -> 90 -> 91 -> 102 -> 101 -> 100 -> 99 -> 98 -> 97 -> 96 -> 95 -> 94 -> 93 -> 92 -> 79 -> 88 -> 81 -> 80 -> 78 -> 77 -> 76 -> 74 -> 75 -> 73 -> 72 -> 71 -> 70 -> 69 -> 66 -> 68 -> 67 -> 57 -> 56 -> 55 -> 54 -> 53 -> 52 -> 51 -> 50 -> 49 -> 48 -> 47 -> 46 -> 45 -> 44 -> 43 -> 58 -> 60 -> 59 -> 42 -> 41 -> 40 -> 39 -> 38 -> 37 -> 36 -> 35 -> 34 -> 33 -> 32 -> 31 -> 30 -> 29 -> 124 -> 123 -> 122 -> 121 -> 120 -> 119 -> 118 -> 156 -> 157 -> 158 -> 173 -> 162 -> 161 -> 160 -> 174 -> 159 -> 150 -> 151 -> 155 -> 152 -> 154 -> 153 -> 128 -> 129 -> 130 -> 131 -> 18 -> 19 -> 20 -> 127 -> 126 -> 125 -> 28 -> 27 -> 26 -> 25 -> 21 -> 24 -> 22 -> 23 -> 13 -> 12 -> 14 -> 11 -> 10 -> 9 -> 7 -> 8 -> 6 -> 5 -> 275 -> 274 -> 273 -> 272 -> 271 -> 270 -> 15 -> 16 -> 17 -> 132 -> 149 -> 177 -> 176 -> 175 -> 178 -> 179 -> 180 -> 181 -> 182 -> 183 -> 184 -> 186 -> 185 -> 192 -> 196 -> 197 -> 198 -> 144 -> 145 -> 146 -> 147 -> 148 -> 138 -> 137 -> 136 -> 135 -> 134 -> 133 -> 269 -> 268 -> 267 -> 266 -> 265 -> 264 -> 263 -> 262 -> 261 -> 260 -> 258 -> 259 -> 276 -> 3 -> 4 -> 0