# Text, logo, company name Description automatically generatedLab File

# Artificial Intelligence [CSE401]

# Department of Computer Science and Engineering

# Logo Description automatically generatedASET

**Submitted by:**

**Janya Verma**

**B. Tech. CSE AI**

**A023119820023**

**4CSE-11 AI**

**Submitted to:**

**Dr. Soumya Ranjan Nayak**

**Experiment 1**

Write a program to implement A\* algorithm in python

**Program:**

class Node():

"""A node class for A\* Pathfinding"""

def init (self, parent=None, position=None): self.parent = parent

self.position = position self.g = 0

self.h = 0

self.f = 0

def eq (self, other):

return self.position == other.position def astar(maze, start, end):

"""Returns a list of tuples as a path from the given start to the given end in the given maze"""

# Create start and end node start\_node = Node(None, start)

start\_node.g = start\_node.h = start\_node.f = 0 end\_node = Node(None, end)

end\_node.g = end\_node.h = end\_node.f = 0 # Initialize both open and closed list open\_list = []

closed\_list = []

# Add the start node open\_list.append(start\_node) # Loop until you find the end while len(open\_list) > 0:

# Get the current node current\_node = open\_list[0] current\_index = 0

for index, item in enumerate(open\_list): if item.f < current\_node.f:

current\_node = item current\_index = index

# Pop current off open list, add to closed list open\_list.pop(current\_index)

closed\_list.append(current\_node)

# Found the goal

if current\_node == end\_node: path = []

current = current\_node while current is not None:

path.append(current.position) current = current.parent

return path[::-1] # Return reversed path

# Generate children children = []

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1),

(1, -1), (1, 1)]: # Adjacent squares

# Get node position

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

# Make sure within range

if node\_position[0] > (len(maze) - 1) or node\_position[0] < 0 or node\_position[1] > (len(maze[len(maze)-1]) -1) or node\_position[1] < 0:

continue

# Make sure walkable terrain

if maze[node\_position[0]][node\_position[1]] != 0: continue

# Create new node

new\_node = Node(current\_node, node\_position)

# Append children.append(new\_node)

# Loop through children for child in children:

# Child is on the closed list for closed\_child in closed\_list:

if child == closed\_child:

continue

# Create the f, g, and h values child.g = current\_node.g + 1

child.h = ((child.position[0] - end\_node.position[0]) \*\* 2) + ((child.position[1] - end\_node.position[1]) \*\* 2)

child.f = child.g + child.h

# Child is already in the open list for open\_node in open\_list:

if child == open\_node and child.g > open\_node.g: continue

# Add the child to the open list open\_list.append(child)

def main():

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| maze = [[0, | 0, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 1, | 0, | 0, | 0, | 0, | 0], |
| [0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0]] |

start = (0, 0)

end = (7, 6)

path = astar(maze, start, end) print(path)

if name == ' main ': main()

**Output**



# **Experiment 2**

Write a program to implement Single Player Game

## Program

**Output**

**Experiment 3**

Write a program to implement Tic-Tac-Toe game problem

## Program:

# Tic-Tac-Toe Program using # random number in Python

# importing all necessary libraries 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

for x in range(len(board)): if board[x, x] != 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

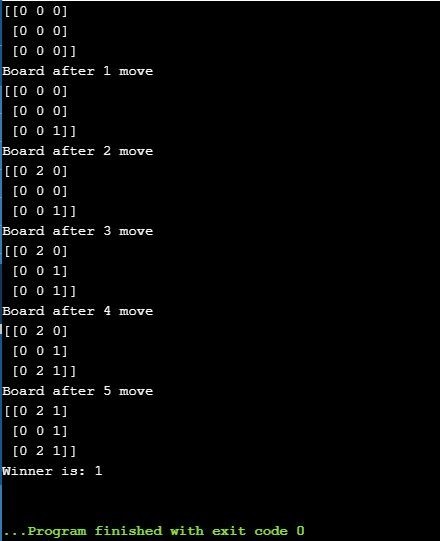
winner = evaluate(board) if winner != 0:

break return(winner)

# Driver Code

print("Winner is: " + str(play\_game()))

## Output



**Experiment 4**

Implement Brute force solution to the Knapsack problem in Python

## Program:

*class Bounty:*

*def init (self, value, weight, volume):*

*self.value, self.weight, self.volume = value, weight, volume*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *panacea* | *=* | *Bounty(3000,* | *0.3,* | *0.025)* |
| *ichor =* |  | *Bounty(1800,* | *0.2,* | *0.015)* |
| *gold =* |  | *Bounty(2500,* | *2.0,* | *0.002)* |
| *sack =* |  | *Bounty( 0,* | *25.0,* | *0.25)* |
| *best =* |  | *Bounty( 0,* | *0,* | *0)* |
| *current* | *=* | *Bounty( 0,* | *0,* | *0)* |

*best\_amounts = (0, 0, 0)*

*max\_panacea = int(min(sack.weight // panacea.weight, sack.volume // panacea.volume))*

*max\_ichor = int(min(sack.weight // ichor.weight, sack.volume // ichor.volume))*

*max\_gold = int(min(sack.weight // gold.weight, sack.volume // gold.volume))*

*for npanacea in xrange(max\_panacea): for nichor in xrange(max\_ichor):*

*for ngold in xrange(max\_gold):*

*current.value = npanacea \* panacea.value + nichor \* ichor.value*

*+ ngold \* gold.value*

*current.weight = npanacea \* panacea.weight + nichor \* ichor.weight + ngold \* gold.weight*

*current.volume = npanacea \* panacea.volume + nichor \* ichor.volume + ngold \* gold.volume*

*and \*

*if current.value > best.value and current.weight <= sack.weight*

*current.volume <= sack.volume:*

*best = Bounty(current.value, current.weight,*

*current.volume)*

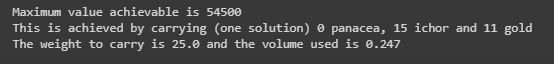
*best\_amounts = (npanacea, nichor, ngold)*

*print "Maximum value achievable is", best.value*

*print "This is achieved by carrying (one solution) %d panacea,%d ichor and*

*%d gold" % \(best\_amounts[0], best\_amounts[1], best\_amounts[2]) print "The weight to carry is %4.1f and the volume used is %5.3f" % (best.weight, best.volume)*

**Output**



# **Experiment 5**

Implement Graph colouring problem using python

**Program:**

def color\_nodes(graph):

# Order nodes in descending degree

nodes = sorted(list(graph.keys()), key=lambda x: len(graph[x]), reverse=True)

color\_map = {}

for node in nodes:

available\_colors = [True] \* len(nodes) for neighbor in graph[node]:

if neighbor in color\_map: color = color\_map[neighbor]

available\_colors[color] = False

for color, available in enumerate(available\_colors): if available:

color\_map[node] = color break

return color\_map

if name == ' main ': graph = {

'a': list('bcd'),

'b': list('ac'),

'c': list('abdef'),

'd': list('ace'),

'e': list('cdf'),

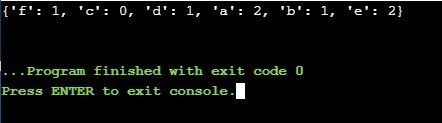
'f': list('ce')

}

print(color\_nodes(graph))

# {'c': 0, 'a': 1, 'd': 2, 'e': 1, 'b': 2, 'f': 2}

## Output



**Experiment 6**

Write a program to implement BFS for water jug problem using Python

## Program:

def pour(jug1, jug2):

max1, max2, fill = 5, 7, 4 #Change maximum capacity and final capacity 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)

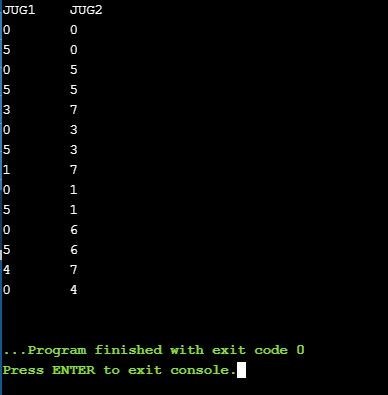
elif jug1 < (max2-jug2): pour(0, (jug1+jug2))

else:

pour(jug1-(max2-jug2), (max2-jug2)+jug2)

print("JUG1\tJUG2") pour(0, 0)

## Output



**Experiment 7**

Write a program to implement DFS using Python

## Program:

# Python3 program to print DFS traversal # from a given given graph

from collections import defaultdict

# This class represents a directed graph using # adjacency list representation

class Graph:

# Constructor

def init (self):

# default dictionary to store graph self.graph = defaultdict(list)

# function to add an edge to graph def addEdge(self, u, v):

self.graph[u].append(v)

# A function used by DFS

def DFSUtil(self, v, visited):

# Mark the current node as visited # and print it

visited[v] = True print(v, end = ' ')

# Recur for all the vertices # adjacent to this vertex for i in self.graph[v]:

if visited[i] == False: self.DFSUtil(i, visited)

# The function to do DFS traversal. It uses # recursive DFSUtil()

def DFS(self, v):

# Mark all the vertices as not visited visited = [False] \* (len(self.graph))

# Call the recursive helper function # to print DFS traversal self.DFSUtil(v, visited)

# Driver code

# Create a graph given # in the above diagram g = Graph() g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

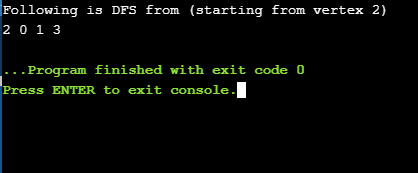
g.addEdge(2, 0)

g.addEdge(2, 3)

g.addEdge(3, 3)

print("Following is DFS from (starting from vertex 2)") g.DFS(2)

**Output**



# **Experiment 8**

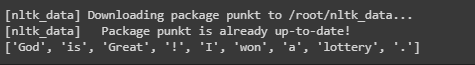
Tokenization of word and Sentences with the help of NLTK package

## Program:

Import nltk nltk.download('punkt')

from nltk.tokenize import word\_tokenize text = "God is Great! I won a lottery." print(word\_tokenize(text))

## Output



**Experiment 9**

Design an XOR truth table using

**Program:**

# Python3 program to illustrate # working of Xor gate

def XOR (a, b): if a != b:

return 1 else:

return 0

# Driver code

if name ==' main ': print(XOR(5, 5))

print("+---------------+ +")

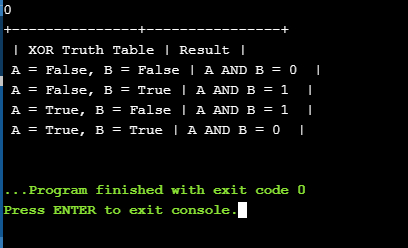
print(" | XOR Truth Table | Result |")

print(" A = False, B = False | A AND B =",XOR(False,False)," |

")

print(" A = False, B = True | A AND B =",XOR(False,True)," | ") print(" A = True, B = False | A AND B =",XOR(True,False)," | ") print(" A = True, B = True | A AND B =",XOR(True,True)," | ")

**Output**



# **Experiment 10**

# Study of SCIKIT fuzzy

## Theory:

scikit-fuzzy (a.k.a. skfuzzy): Fuzzy logic toolbox for Python.

This package implements many useful tools for projects involving fuzzy logic, also known as grey logic.

Scikit-fuzzy is a robust set of foundational tools for problems involving fuzzy logic and fuzzy systems. This area has been a challenge for the scientific Python community, largely because the common first exposure to this topic is through the MATLAB® Fuzzy Logic Toolbox™.

The current capabilities of scikit-fuzzy include: fuzzy membership function generation; fuzzy set operations; lambda-cuts; fuzzy mathematics including Zadeh's extension principle, the vertex method, and the DSW method; fuzzy implication given an IF THEN system of fuzzy rules (via Mamdani [min] or Larsen [product] implication); various defuzzification algorithms; fuzzy c-means clustering; and Fuzzy Inference Ruled by Else-action (FIRE) denoising of 1d or 2d signals.

The goals of scikit-fuzzy are to provide the community with a robust toolkit of independently developed and implemented fuzzy logic algorithms, filling a void in the capabilities of scientific and numerical Python, and to increase the attractiveness of scientific Python as a valid alternative to closed-source options. Scikit-fuzzy is structured similarly to scikit-learn and scikit-image, current source code is available on GitHub, and pull requests are welcome.