MAHARISHI DAYANAND UNIVERSITY

Logo

Description automatically generated with medium confidence

*Delhi Global Institute of Technology*

Artificial Intelligence Lab

*Submitted By: Bazgha Razi*

*Subject Code* : LC-CSE-326G

*Subject Name* : Artificial Intelligence Lab using Python

*Registration Number* : *191380214*

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Program** | **Date** | **Remaks** |
| 1. | Write a program to implement Breadth First Search using Python. | 30/03/22 |  |
| 2. | Write a program to implement Depth First Search using Python. | 05/04/22 |  |
| 3. | Write a program to implement Tic Tac Toe using Python. | 08/04/22 |  |
| 4. | Write a program to implement 8-Puzzle problem using Python. | 22/04/22 |  |
| 5. | Write a program to implement Water-Jug problem using Python. | 29/04/22 |  |
| 6. | Write a program to implement Travelling Salesman problem using Python. | 05/05/22 |  |
| 7. | Write a program to implement Tower of Hanoi using Python. | 13/05/22 |  |
| 8. | Write a program to implement Monkey Banana problem using Python. | 03/06/22 |  |
| 9. | Write a program to implement Missionaries-Cannibals problem using Python. | 03/06/22 |  |
| 10. | Write a program to implement N-Queens problem using Python. | 10/06/22 |  |

**Program 1:** Write a program to implement Breadth First Search using python.

**Code:**

from collections import defaultdict

class Graph:

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

self.graph = defaultdict(list)

def addEdge(self,u,v):

self.graph[u].append(v)

def BFS(self, s):

visited = [False] \* (len(self.graph))

queue = []

queue.append(s)

visited[s] = True

while queue:

s = queue.pop(0)

print (s, end = " ")

for i in self.graph[s]:

if visited[i] == False:

queue.append(i)

visited[i] = True

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 Breadth First Traversal"" (starting from vertex 1)")

g.BFS(1)

**Output:**

**Text

Description automatically generated**

**Program 2:** Write a program to implement Depth First Search using python.

**Code:**

def dfs(visited, graph, node):

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

dfs(visited, graph, neighbour)

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

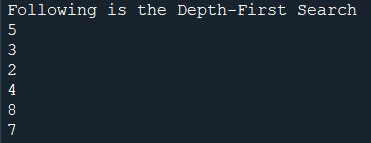
}

visited = set()

print("Following is the Depth-First Search")

dfs(visited, graph, '5')

**Output:**

****

**Program 3:** Write a program to implement Tic-Tac-Toe using python.

**Code:**

import numpy as np

import random

from time import sleep

def create\_board():

return(np.array([

[0, 0, 0],

[0, 0, 0],

[0, 0, 0]

]))

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)

def random\_place(board, player):

selection = possibilities(board)

current\_loc = random.choice(selection)

board[current\_loc] = player

return(board)

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)

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)

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

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

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)

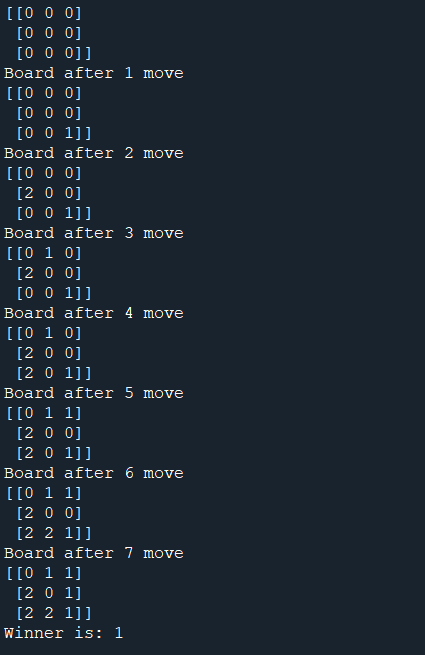
if winner != 0:

break

return(winner)

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

**Output:**



**Program 4:** Write a program to implement 8-Puzzle problem using python.

**Code:**

import copy

from heapq import heappush, heappop

n = 3

row = [ 1, 0, -1, 0 ]

col = [ 0, -1, 0, 1 ]

class priorityQueue:

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

self.heap = []

def push(self, k):

heappush(self.heap, k)

def pop(self):

return heappop(self.heap)

def empty(self):

if not self.heap:

return True

else:

return False

class node:

def \_\_init\_\_(self, parent, mat, empty\_tile\_pos,

cost, level):

self.parent = parent

self.mat = mat

self.empty\_tile\_pos = empty\_tile\_pos

self.cost = cost

self.level = level

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

return self.cost < nxt.cost

def calculateCost(mat, final) -> int:

count = 0

for i in range(n):

for j in range(n):

if ((mat[i][j]) and

(mat[i][j] != final[i][j])):

count += 1

return count

def newNode(mat, empty\_tile\_pos, new\_empty\_tile\_pos,

level, parent, final) -> node:

new\_mat = copy.deepcopy(mat)

x1 = empty\_tile\_pos[0]

y1 = empty\_tile\_pos[1]

x2 = new\_empty\_tile\_pos[0]

y2 = new\_empty\_tile\_pos[1]

new\_mat[x1][y1], new\_mat[x2][y2] = new\_mat[x2][y2], new\_mat[x1][y1]

cost = calculateCost(new\_mat, final)

new\_node = node(parent, new\_mat, new\_empty\_tile\_pos,

cost, level)

return new\_node

def printMatrix(mat):

for i in range(n):

for j in range(n):

print("%d " % (mat[i][j]), end = " ")

print()

def isSafe(x, y):

return x >= 0 and x < n and y >= 0 and y < n

def printPath(root):

if root == None:

return

printPath(root.parent)

printMatrix(root.mat)

print()

def solve(initial, empty\_tile\_pos, final):

pq = priorityQueue()

cost = calculateCost(initial, final)

root = node(None, initial,

empty\_tile\_pos, cost, 0)

pq.push(root)

while not pq.empty():

minimum = pq.pop()

if minimum.cost == 0:

printPath(minimum)

return

for i in range(n):

new\_tile\_pos = [

minimum.empty\_tile\_pos[0] + row[i],

minimum.empty\_tile\_pos[1] + col[i], ]

if isSafe(new\_tile\_pos[0], new\_tile\_pos[1]):

child = newNode(minimum.mat,

minimum.empty\_tile\_pos,

new\_tile\_pos,

minimum.level + 1,

minimum, final,)

pq.push(child)

initial = [ [ 1, 2, 3 ],

[ 5, 6, 0 ],

[ 7, 8, 4 ] ]

final = [ [ 1, 2, 3 ],

[ 5, 8, 6 ],

[ 0, 7, 4 ] ]

empty\_tile\_pos = [ 1, 2 ]

solve(initial, empty\_tile\_pos, final)

**Output:**

A picture containing remote, electronics, control, remote control

Description automatically generated