SETHU INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)
PULLOOR, KARIAPATTI - 626 115

<u>DEPARTMENT OF INFORMATION TECHNOLOGY</u>

MASTER RECORD

BRANCH :	**IT**
YEAR / SEMESTER :	**III / VI**
SUB CODE :	**19UIT608**
SUB NAME :	**ARTIFICIAL INTELLIGENCE LABORATORY**

FACULTY INCHARGE

Mr.S.Parameswaran

Assistant Professor

1. **Write a Program to Implement Breadth First Search using Python.**

Aim:

To write a program to implement a breadth first search using python

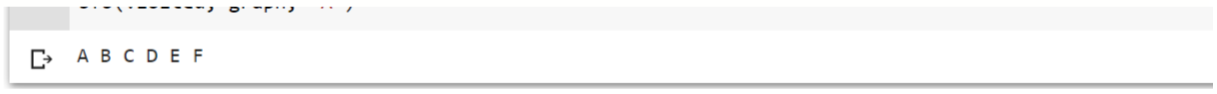
Algorithm:

```
Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until QUEUE is empty
Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).
Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and
set their STATUS = 2 (waiting state) [END OF LOOP]
```

Program:

```
from collections import defaultdict
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)

# Driver Code
bfs(visited, graph, 'A')
```

Output:



```
bfs(visited, graph, 'A')
```

```
A B C D E F
```

Result:
Hence the implementation of breadth first search program is verified successfully.

2. Write a Program to Implement Depth First Search using Python.

Aim:

To write a program to implement a depth first search using python

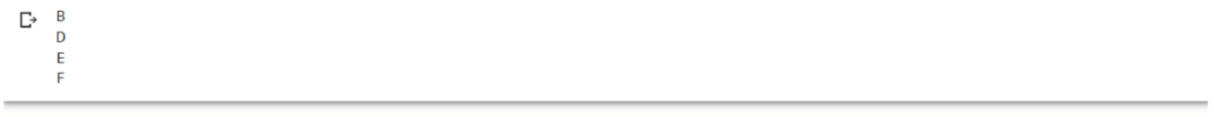
Algorithm:

Step 1: Create a stack with the total number of vertices in the graph as the size.
Step 2: Choose any vertex as the traversal's beginning point. Push a visit to that vertex and add it to the stack.
Step 3 - Push any non-visited adjacent vertices of a vertex at the top of the stack to the top of the stack.
Step 4 - Repeat steps 3 and 4 until there are no more vertices to visit from the vertex at the top of the stack.
Step 5 - If there are no new vertices to visit, go back and pop one from the stack using backtracking.
Step 6 - Continue using steps 3, 4, and 5 until the stack is empty.
Step 7 - When the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges.

Program:

```
from collections import defaultdict
graph = {
'A' : ['B','C'],
'B' : ['D', 'E'],
'C' : ['F'],
'D' : [],
'E' : ['F'],
'F' : []
}
visited = set()
def dfs(visited, graph, node):
if node not in visited:
print (node)
visited.add(node)
for neighbour in graph[node]:
dfs(visited, graph, neighbour)
dfs(visited, graph, 'B')
```

Output:



```
B
D
E
F
```

Result:

Hence the implementation of depth first search program is verified successfully.

3. Write a Program to Implement A* Search using Python.

Aim:

To write a program to implement A* Search using python

Algorithm:

```
Step 1: Make an open list containing starting node
Step 2: If it reaches the destination node:
        Make a closed empty list
        If it does not reach the destination node,
        then consider a node with the lowest f-score in the open list
        Else :
        Put the current node in the list and check its neighbors
Step 3: For each neighbor of the current node:
        If the neighbor has a lower g value than the current node and is in the closed list:
        Replace neighbor with this new node as the neighbor's parent
        Else If (current g is lower and neighbor is in the open list):
        Replace neighbor with the lower g value and change the neighbor's parent to the
        current node.
Step 4: Else If the neighbor is not in both lists:
Step 5: Add it to the open list and set its g
```

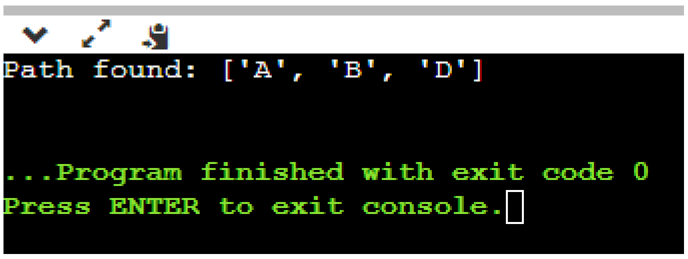
Program:

```
from collections import deque

class Graph:
   def __init__(self, adjac_lis):
      self.adjac_lis = adjac_lis

   def get_neighbors(self, v):
      return self.adjac_lis[v]

   # This is heuristic function which is having equal values for all nodes
   def h(self, n):
      H = {
         'A': 1,
         'B': 1,
         'C': 1,
         'D': 1
      }

      return H[n]
```

```
  def a_star_algorithm(self, start, stop):
     # In this open_lst is a lisy of nodes which have been visited, but who's
     # neighbours haven't all been always inspected, It starts off with the start
#node
     # And closed_lst is a list of nodes which have been visited
     # and who's neighbors have been always inspected
     open_lst = set([start])
     closed_lst = set([])

     # poo has present distances from start to all other nodes
     # the default value is +infinity
     poo = {}
     poo[start] = 0

     # par contains an adjac mapping of all nodes
     par = {}
     par[start] = start

     while len(open_lst) > 0:
        n = None

        # it will find a node with the lowest value of f() -
        for v in open_lst:
           if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
              n = v;

        if n == None:
           print('Path does not exist!')
           return None

        # if the current node is the stop
        # then we start again from start
        if n == stop:
           reconst_path = []

           while par[n] != n:
              reconst_path.append(n)
              n = par[n]

           reconst_path.append(start)

           reconst_path.reverse()

           print('Path found: {}'.format(reconst_path))
```

```
                return reconst_path

            # for all the neighbors of the current node do
            for (m, weight) in self.get_neighbors(n):
              # if the current node is not presentin both open_lst and closed_lst
                # add it to open_lst and note n as it's par
                if m not in open_lst and m not in closed_lst:
                    open_lst.add(m)
                    par[m] = n
                    poo[m] = poo[n] + weight

                # otherwise, check if it's quicker to first visit n, then m
                # and if it is, update par data and poo data
                # and if the node was in the closed_lst, move it to open_lst
                else:
                    if poo[m] > poo[n] + weight:
                        poo[m] = poo[n] + weight
                        par[m] = n

                        if m in closed_lst:
                            closed_lst.remove(m)
                            open_lst.add(m)

            # remove n from the open_lst, and add it to closed_lst
            # because all of his neighbors were inspected
            open_lst.remove(n)
            closed_lst.add(n)

        print('Path does not exist!')
        return None

adjac_lis = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
graph1.a_star_algorithm('A', 'D')
```

Output:



```
Path found: ['A', 'B', 'D']


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:

Hence the implementation of A* Search program is verified successfully.

4. Write a program to implement Best First Search.

Aim:

To write a program to implement a best first search using python

Algorithm:

Step 1: Create 2 empty lists: OPEN and CLOSED

Step 2: Start from the initial node (say N) and put it in the 'ordered' OPEN list

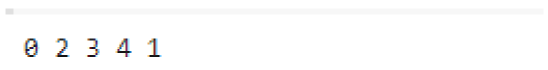
Step 3: Repeat the next steps until the GOAL node is reached

1. If the OPEN list is empty, then EXIT the loop returning 'False'
2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node
3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
4. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

Program:

```
from queue import PriorityQueue
v = 5
graph = [[] for i in range(v)]
def best_first_search(source, target, n):
  visited = [0] * n
  visited[0] = True
  pq = PriorityQueue()
  pq.put((0, source))
  while pq.empty() == False:
    u = pq.get()[1]
    print(u, end=" ")
    if u == target:
      break
    for v, c in graph[u]:
      if visited[v] == False:
        visited[v] = True
        pq.put((c, v))
  print()
def addedge(x, y, cost):
  graph[x].append((y, cost))
  graph[y].append((x, cost))
addedge(0, 1, 5)
addedge(0, 2, 1)
addedge(2, 3, 2)
addedge(1, 4, 1)
addedge(3, 4, 2)
source = 0
target = 4
best_first_search(0,5,7)
```

Output:



```
0 2 3 4 1
```

Result:
Hence the implementation of Best First Search program is verified successfully.

5. Write a Program to Implement a Tic-Tac-Toe game using Python.

Aim:

To write a program to implement a tic-tac-toe game using python

Algorithm:

To make a move, do the following:

Step 1: View the vector board as a ternary number (base three). Convert it to a decimal number.

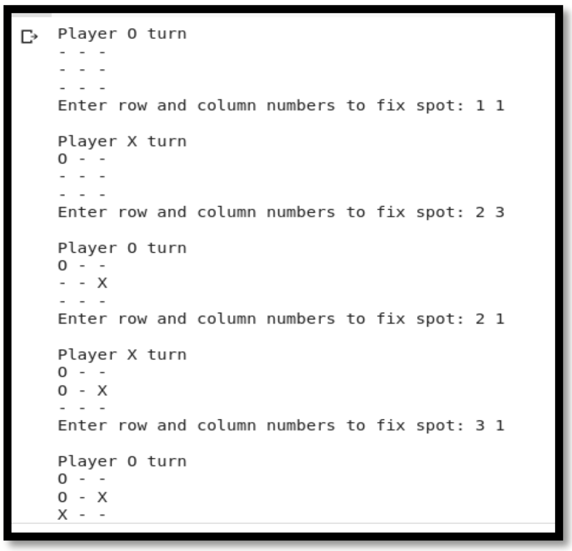
Step 2: Use the number computed in step 1 as index into Movetable and access the vector stored there.

Step 3: The vector selected in step 2 represents the way the board will look after the move that should be made. So set Board equal to that vector.

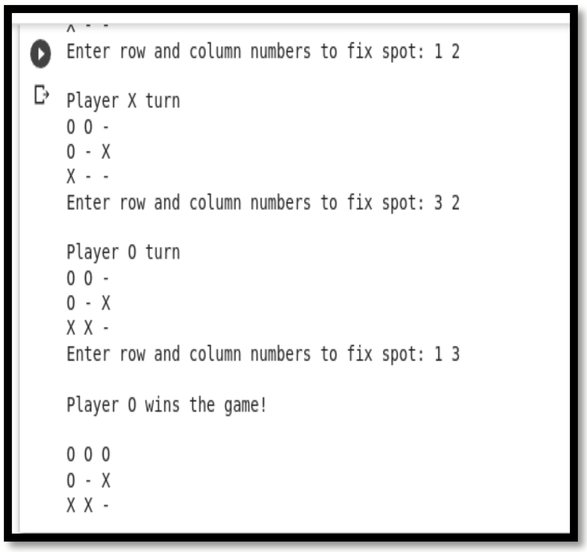
Program:

```
import random
class TicTacToe:
  def __init__(self):
     self.board = []
  def create_board(self):
     for i in range(3):
        row = []
        for j in range(3):
           row.append('-')
        self.board.append(row)
  def get_random_first_player(self):
     return random.randint(0, 1)
  def fix_spot(self, row, col, player):
     self.board[row][col] = player
  def is_player_win(self, player):
     win = None
     n = len(self.board)
     for i in range(n):
        win = True
        for j in range(n):
           if self.board[i][j] != player:
              win = False
              break
        if win:
            return win
    for i in range(n):
        win = True
        for j in range(n):
            if self.board[j][i] != player:
                win = False
                break
        if win:
            return win
    win = True
    for i in range(n):
        if self.board[i][i] != player:
            win = False
            break
    if win:
        return win
    win = True
    for i in range(n):
        if self.board[i][n - 1 - i] != player:
            win = False
            break
    if win:
        return win
    return False
    for row in self.board:
        for item in row:
            if item == '-':
                return False
    return True
def is_board_filled(self):
    for row in self.board:
        for item in row:
            if item == '-':
                return False
    return True
def swap_player_turn(self, player):
    return 'X' if player == 'O' else 'O'
    def show_board(self):
        for row in self.board:
            for item in row:
                print(item, end=" ")
            print()
    def start(self):
        self.create_board()
        player = 'X' if self.get_random_first_player() == 1 else 'O'
        while True:
            print(f"Player {player} turn")
            self.show_board()
            row, col = list(
                map(int, input("Enter row and column numbers to fix spot: ").split()))
            print()
            self.fix_spot(row - 1, col - 1, player)
            if self.is_player_win(player):
                print(f"Player {player} wins the game!")
                break
            if self.is_board_filled():
                print("Match Draw!")
                break
            player = self.swap_player_turn(player)

        print()
        self.show_board()
tic_tac_toe = TicTacToe()
tic_tac_toe.start()
```

Output:



```
Player O turn
- - -
- - -
- - -
Enter row and column numbers to fix spot: 1 1

Player X turn
O - -
- - -
- - -
Enter row and column numbers to fix spot: 2 3

Player O turn
O - -
- - X
- - -
Enter row and column numbers to fix spot: 2 1

Player X turn
O - -
O - X
- - -
Enter row and column numbers to fix spot: 3 1

Player O turn
O - -
O - X
X - -
```



```
Enter row and column numbers to fix spot: 1 2

Player X turn
O O -
O - X
X - -
Enter row and column numbers to fix spot: 3 2

Player O turn
O O -
O - X
X X -
Enter row and column numbers to fix spot: 1 3

Player O wins the game!

O O O
O - X
X X -
```

Result:

Hence the implementation of tic-tac-toe game is verified successfully.

6. Write a Program to implement 8-Puzzle problem using game search algorithm

Aim:

To write a program to implement a 8-Puzzle using python

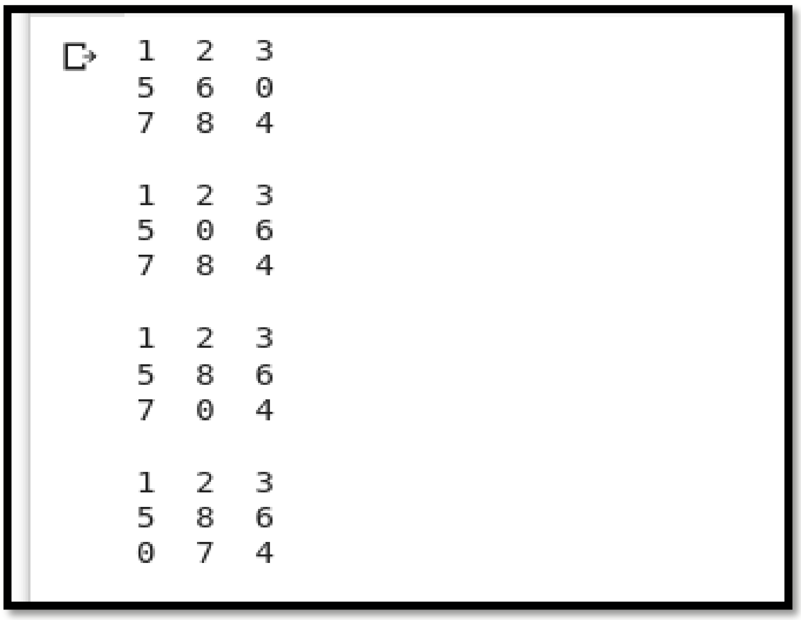
Algorithm:

```
Step 1: (i) Define a list OPEN.
        (ii)Initially, OPEN consists solely of a single node, the start node S.
Step 2: If the list is empty, return failure and exit
Step 3: (i) Remove node n with the smallest value of f(n) from OPEN and move it to list
            CLOSED.
        (ii)If node n is a goal state, return success and exit.
Step 4: Expand node n.
Step 5: (i) If any successor to n is the goal node, return success and the solution by tracing the
            path from goal node to S.
        (ii)Otherwise, go to Step-06.
Step 6: (i)For each successor node,
        (ii) Apply the evaluation function f to the node.
        (iii) If the node has not been in either list, add it to OPEN.
Step 7:Go back to Step-02.
```

Program:

```
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:



```
1  2  3
5  6  0
7  8  4

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

1  2  3
5  8  6
0  7  4
```

Result:
Hence the implementation of 8 Puzzle problem using python is verified successfully.

7. Write a program to construct a Bayesian network from given data.
Scenario:
Alarm Problem:

A person has installed a new alarm system that can be triggered by a burglary or an earthquake. These people also have two neighbors (John and Mary) that are asked to make a call if they hear the alarm. This problem is modeled in a bayesian network with probabilities attached to each edge. Alarm has burglary and earthquake as parents, John Calls has Alarm as parent and Mary Calls has Alarm as parent. We can ask the network: what is the probability for a burglary if both John and Mary calls.

Aim:

To write a program to construct a Bayesian network for the given data.

Algorithm:

Step 1: The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off.
Step 2: The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
Step 3: The conditional distributions for each node are given as conditional probabilities table or CPT.
Step 4: Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
Step 5: In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

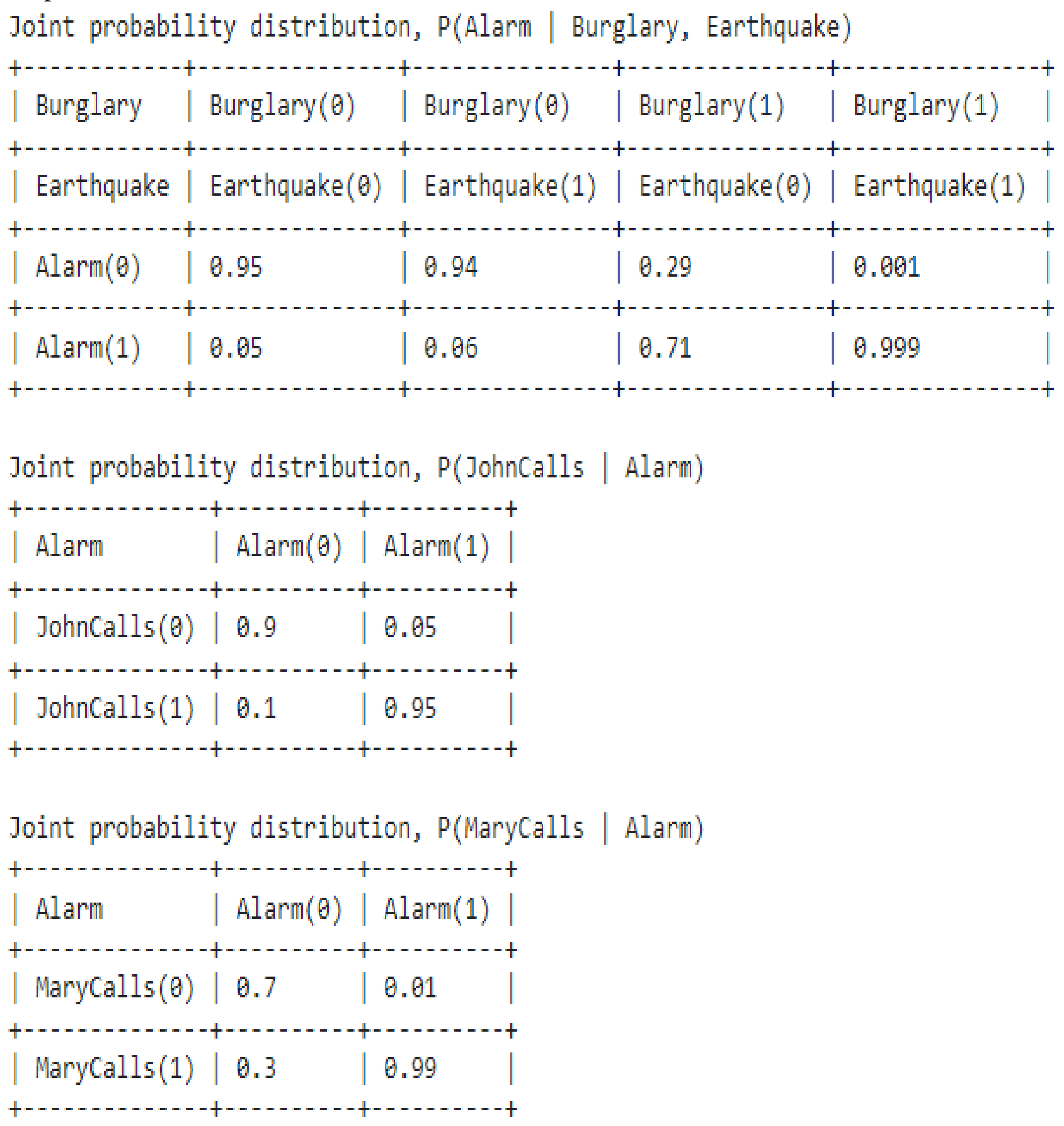
Program:

```
!pip install pgmpy
!pip install pandas
!pip install numpy

import pgmpy.models
import pgmpy.inference
import networkx as nx
import pylab as plt
# Create a bayesian network
model = pgmpy.models.BayesianModel([('Burglary', 'Alarm'),
                          ('Earthquake', 'Alarm'),
                          ('Alarm', 'JohnCalls'),
                          ('Alarm', 'MaryCalls')])
# Define conditional probability distributions (CPD)
# Probability of burglary (True, False)
cpd_burglary = pgmpy.factors.discrete.TabularCPD('Burglary', 2, [[0.001], [0.999]])
# Probability of earthquake (True, False)
```

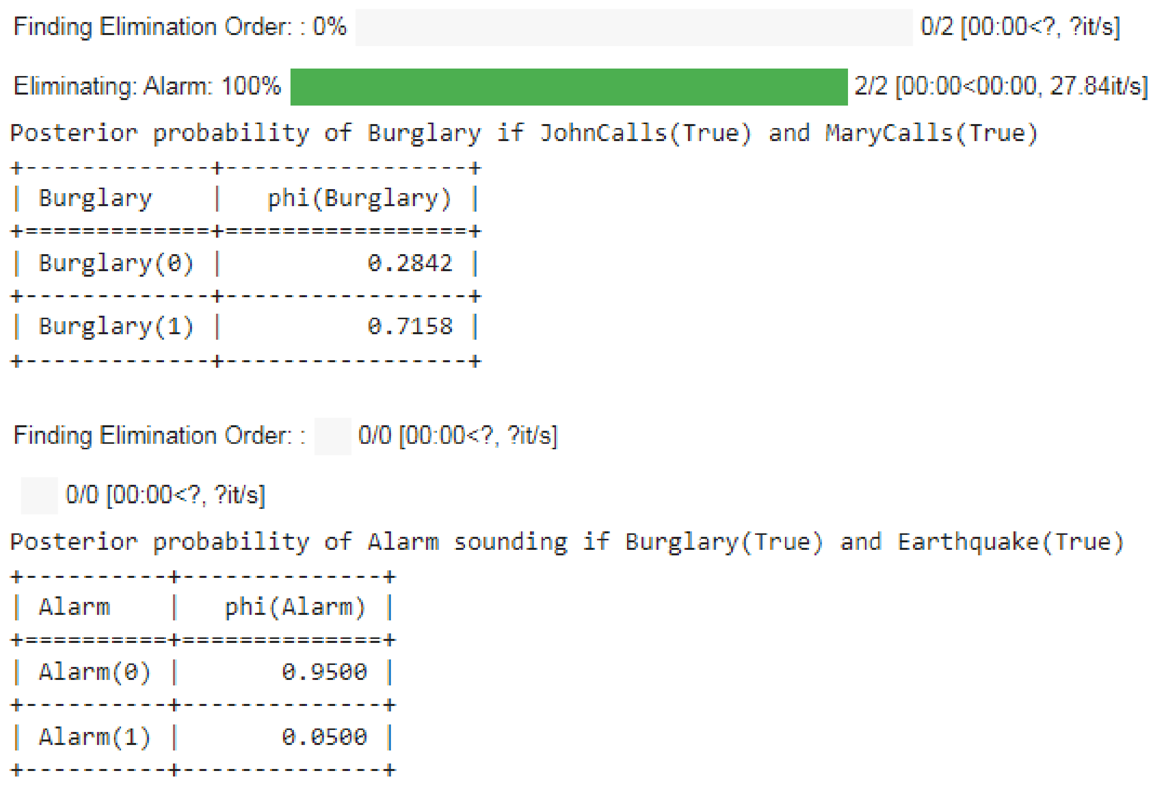
```
cpd_earthquake = pgmpy.factors.discrete.TabularCPD('Earthquake', 2, [[0.002], [0.998]])
# Probability of alarm going of (True, False) given a burglary and/or earthquake
cpd_alarm = pgmpy.factors.discrete.TabularCPD('Alarm', 2, [[0.95, 0.94, 0.29, 0.001],
                                                           [0.05, 0.06, 0.71, 0.999]],
                                              evidence=['Burglary', 'Earthquake'],
                                              evidence_card=[2, 2])
# Probability that John calls (True, False) given that the alarm has sounded
cpd_john = pgmpy.factors.discrete.TabularCPD('JohnCalls', 2, [[0.90, 0.05],
                                                           [0.10, 0.95]],
                                              evidence=['Alarm'],
                                              evidence_card=[2])
# Probability that Mary calls (True, False) given that the alarm has sounded
cpd_mary = pgmpy.factors.discrete.TabularCPD('MaryCalls', 2, [[0.70, 0.01],
                                                           [0.30, 0.99]],
                                              evidence=['Alarm'],
                                              evidence_card=[2])
# Add CPDs to the network structure
model.add_cpds(cpd_burglary, cpd_earthquake, cpd_alarm, cpd_john, cpd_mary)
# Check if the model is valid, throw an exception otherwise
model.check_model()
# Print probability distributions
print('Probability distribution, P(Burglary)')
print(cpd_burglary)
print()
print('Probability distribution, P(Earthquake)')
print(cpd_earthquake)
print()
print('Joint probability distribution, P(Alarm | Burglary, Earthquake)')
print(cpd_alarm)
print()
print('Joint probability distribution, P(JohnCalls | Alarm)')
print(cpd_john)
print()
print('Joint probability distribution, P(MaryCalls | Alarm)')
print(cpd_mary)
print()
# Plot the model
nx.draw(model, with_labels=True)
plt.savefig('C:\\DATA\\Python-data\\bayesian-networks\\alarm.png')
plt.close()
# Perform variable elimination for inference
# Variable elimination (VE) is a an exact inference algorithm in bayesian networks
infer = pgmpy.inference.VariableElimination(model)
# Calculate the probability of a burglary if John and Mary calls (0: True, 1: False)
```

```
posterior_probability = infer.query(['Burglary'], evidence={'JohnCalls': 0, 'MaryCalls': 0})
# Print posterior probability
print('Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)')
print(posterior_probability)
print()
# Calculate the probability of alarm starting if there is a burglary and an earthquake (0: True,
1: False)
posterior_probability = infer.query(['Alarm'], evidence={'Burglary': 0, 'Earthquake': 0})
# Print posterior probability
print('Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)')
print(posterior_probability)
print()
```

Output:



```
Joint probability distribution, P(Alarm | Burglary, Earthquake)
+------------+---------------+---------------+---------------+---------------+
| Burglary   | Burglary(0)   | Burglary(0)   | Burglary(1)   | Burglary(1)   |
+------------+---------------+---------------+---------------+---------------+
| Earthquake | Earthquake(0) | Earthquake(1) | Earthquake(0) | Earthquake(1) |
+------------+---------------+---------------+---------------+---------------+
| Alarm(0)   | 0.95          | 0.94          | 0.29          | 0.001         |
+------------+---------------+---------------+---------------+---------------+
| Alarm(1)   | 0.05          | 0.06          | 0.71          | 0.999         |
+------------+---------------+---------------+---------------+---------------+

Joint probability distribution, P(JohnCalls | Alarm)
+--------------+----------+----------+
| Alarm        | Alarm(0) | Alarm(1) |
+--------------+----------+----------+
| JohnCalls(0) | 0.9      | 0.05     |
+--------------+----------+----------+
| JohnCalls(1) | 0.1      | 0.95     |
+--------------+----------+----------+

Joint probability distribution, P(MaryCalls | Alarm)
+--------------+----------+----------+
| Alarm        | Alarm(0) | Alarm(1) |
+--------------+----------+----------+
| MaryCalls(0) | 0.7      | 0.01     |
+--------------+----------+----------+
| MaryCalls(1) | 0.3      | 0.99     |
+--------------+----------+----------+
```



```
Finding Elimination Order: : 0%                              0/2 [00:00<?, ?it/s]

Eliminating: Alarm: 100%                                     2/2 [00:00<00:00, 27.84it/s]
Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)
+-------------+-----------------+
| Burglary    |   phi(Burglary) |
+=============+=================+
| Burglary(0) |          0.2842 |
+-------------+-----------------+
| Burglary(1) |          0.7158 |
+-------------+-----------------+

Finding Elimination Order: :    0/0 [00:00<?, ?it/s]

    0/0 [00:00<?, ?it/s]
Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)
+----------+--------------+
| Alarm    |   phi(Alarm) |
+==========+==============+
| Alarm(0) |       0.9500 |
+----------+--------------+
| Alarm(1) |       0.0500 |
+----------+--------------+
```

Result:
Hence the construction of Bayesian network using python is verified successfully.

8.Write a program to infer from the Bayesian network

Scenario:
Monty Hall Problem
This problem is about a contest in which a contestant can select 1 of 3 doors, it is a price behind one of the doors. The host of the show (Monty) opens a empty door after the contestant has selected a door and asks the contestant if he want to switch to the other door. The question is if it is best to stick with the selected door or switch to the other door. It is best to switch to the other door because it is a higher probability that the price is behind that door. Implement the program to construct Bayesian network and exact inference.

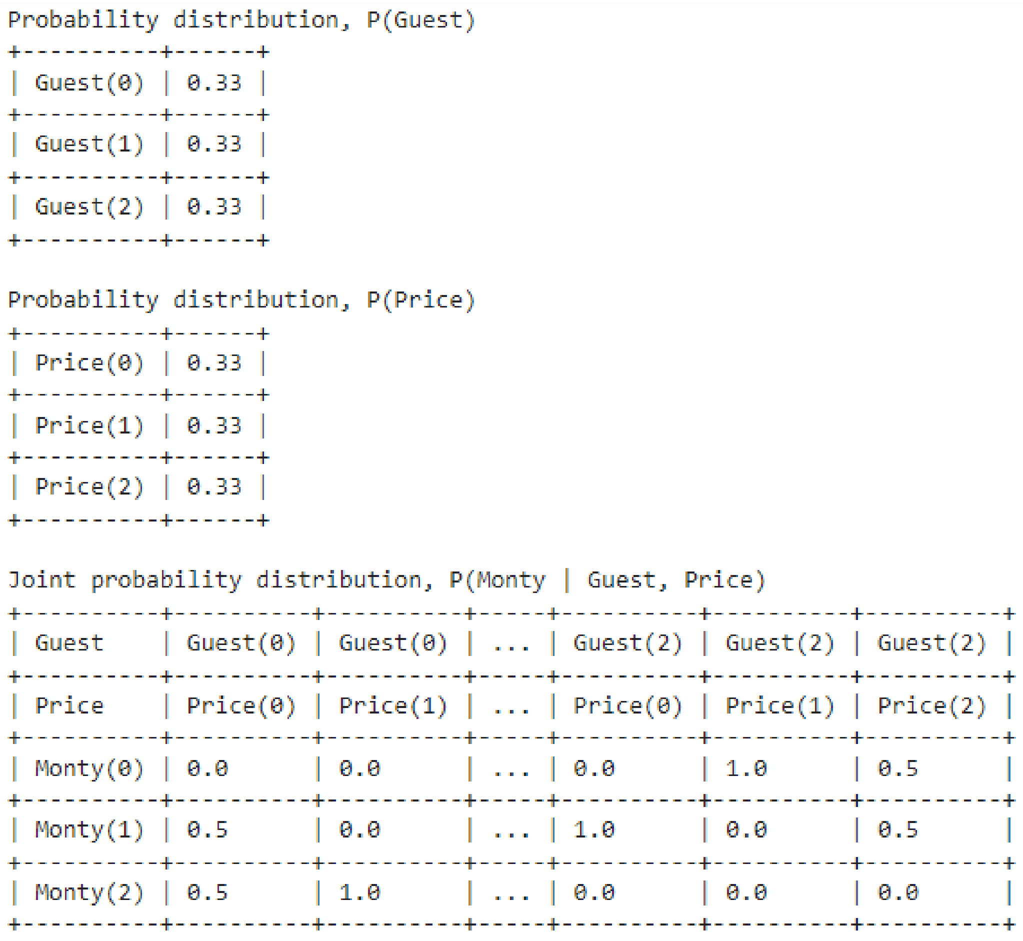
Aim:
To write a program to construct a Bayesian network and exact inference for the given data.

Algorithm:
Step 1: The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off.
Step 2: The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
Step 3: The conditional distributions for each node are given as conditional probabilities table or CPT.
Step 4: Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
Step 5: In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

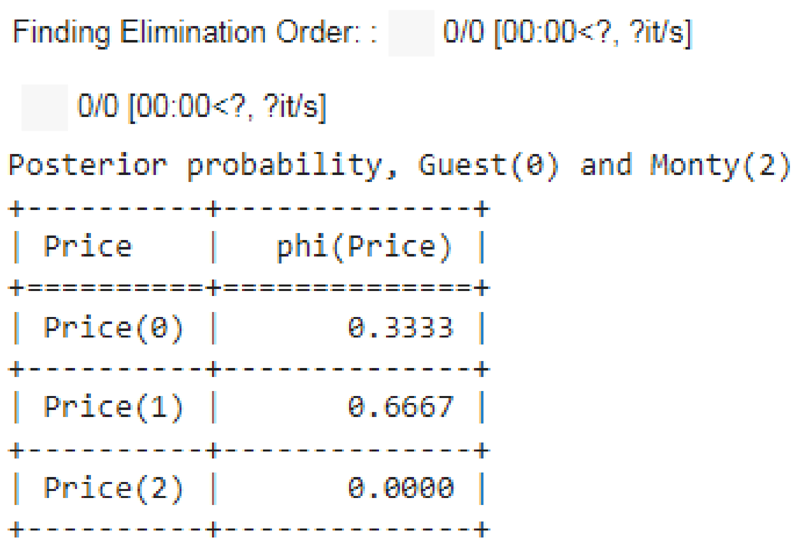
Program:
```
!pip install pgmpy
!pip install pandas
!pip install numpy

# Import libraries
import pgmpy.models
import pgmpy.inference
import networkx as nx
import pylab as plt
# Create a bayesian network
model = pgmpy.models.BayesianModel([('Guest', 'Monty'),
                                    ('Price', 'Monty')])
# Define conditional probability distributions (CPD)
# Probability of guest selecting door 0, 1 and 2
cpd_guest = pgmpy.factors.discrete.TabularCPD('Guest', 3, [[0.33, 0.33, 0.33]])
# Probability that the price is behind door 0, 1 and 2
```

```
cpd_price = pgmpy.factors.discrete.TabularCPD('Price', 3, [[0.33, 0.33, 0.33]])
# Probability that Monty selects a door (0, 1, 2), when we know which door the guest has selected and we know were the price is
cpd_monty = pgmpy.factors.discrete.TabularCPD('Monty', 3, [[0, 0, 0, 0, 0.5, 1, 0, 1, 0.5],
                                                 [0.5, 0, 1, 0, 0, 0, 1, 0, 0.5],
                                                 [0.5, 1, 0, 1, 0.5, 0, 0, 0, 0]],
                                      evidence=['Guest', 'Price'],
                                      evidence_card=[3, 3])
# Add CPDs to the network structure
model.add_cpds(cpd_guest, cpd_price, cpd_monty)
# Check if the model is valid, throw an exception otherwise
model.check_model()
# Print probability distributions
print('Probability distribution, P(Guest)')
print(cpd_guest)
print()
print('Probability distribution, P(Price)')
print(cpd_price)
print()
print('Joint probability distribution, P(Monty | Guest, Price)')
print(cpd_monty)
print()
# Plot the model
nx.draw(model, with_labels=True)
plt.savefig('C:\\DATA\\Python-data\\bayesian-networks\\monty-hall.png')
plt.close()
# Perform variable elimination for inference
# Variable elimination (VE) is a an exact inference algorithm in bayesian networks
infer = pgmpy.inference.VariableElimination(model)
# Calculate probabilites for doors including price, the guest has selected door 0 and Monty has selected door 2
posterior_probability = infer.query(['Price'], evidence={'Guest': 0, 'Monty': 2})
# Print posterior probability
print('Posterior probability, Guest(0) and Monty(2)')
print(posterior_probability)
print()
```

Output:



```
Probability distribution, P(Guest)
+----------+------+
| Guest(0) | 0.33 |
+----------+------+
| Guest(1) | 0.33 |
+----------+------+
| Guest(2) | 0.33 |
+----------+------+

Probability distribution, P(Price)
+----------+------+
| Price(0) | 0.33 |
+----------+------+
| Price(1) | 0.33 |
+----------+------+
| Price(2) | 0.33 |
+----------+------+

Joint probability distribution, P(Monty | Guest, Price)
+----------+----------+----------+-----+----------+----------+----------+
| Guest    | Guest(0) | Guest(0) | ... | Guest(2) | Guest(2) | Guest(2) |
+----------+----------+----------+-----+----------+----------+----------+
| Price    | Price(0) | Price(1) | ... | Price(0) | Price(1) | Price(2) |
+----------+----------+----------+-----+----------+----------+----------+
| Monty(0) | 0.0      | 0.0      | ... | 0.0      | 1.0      | 0.5      |
+----------+----------+----------+-----+----------+----------+----------+
| Monty(1) | 0.5      | 0.0      | ... | 1.0      | 0.0      | 0.5      |
+----------+----------+----------+-----+----------+----------+----------+
| Monty(2) | 0.5      | 1.0      | ... | 0.0      | 0.0      | 0.0      |
+----------+----------+----------+-----+----------+----------+----------+
```



```
Finding Elimination Order: :      0/0 [00:00<?, ?it/s]

      0/0 [00:00<?, ?it/s]
Posterior probability, Guest(0) and Monty(2)
+----------+--------------+
| Price    |   phi(Price) |
+==========+==============+
| Price(0) |       0.3333 |
+----------+--------------+
| Price(1) |       0.6667 |
+----------+--------------+
| Price(2) |       0.0000 |
+----------+--------------+
```

Result:

Hence the construction and inference of Bayesian network using python is verified successfully.