EXPERIMENT NO- 01

Aim: : Design/construct the workflow of a general AI project using draw.io

Description:

Draw.io: Draw.io is a versatile online diagramming tool offering a user-friendly interface for crafting flowcharts, mind maps, wireframes, and more. With its intuitive drag-and-drop functionality and extensive customization options, it's ideal for brainstorming, planning, and visualizing ideas.

Automated Taxi Driver:Automated taxi drivers leverage cutting-edge technology like AI and autonomous vehicles to provide efficient and safe transportation. With advanced sensors and algorithms, they navigate routes, handle traffic, and ensure passenger comfort, promising a seamless and futuristic commuting experience.

1. Agent: Automated Taxi Driver
2. Performance Measure: Safe, fast, legal, comfortable trip, maximize profits
3. Environment: Roads, other traffic, pedestrians, customers
4. Actuators: Steering wheel, accelerator, brake,signal, horn
5. Sensors: Cameras, sonar, speedometer, GPS, engine sensors, keyboard

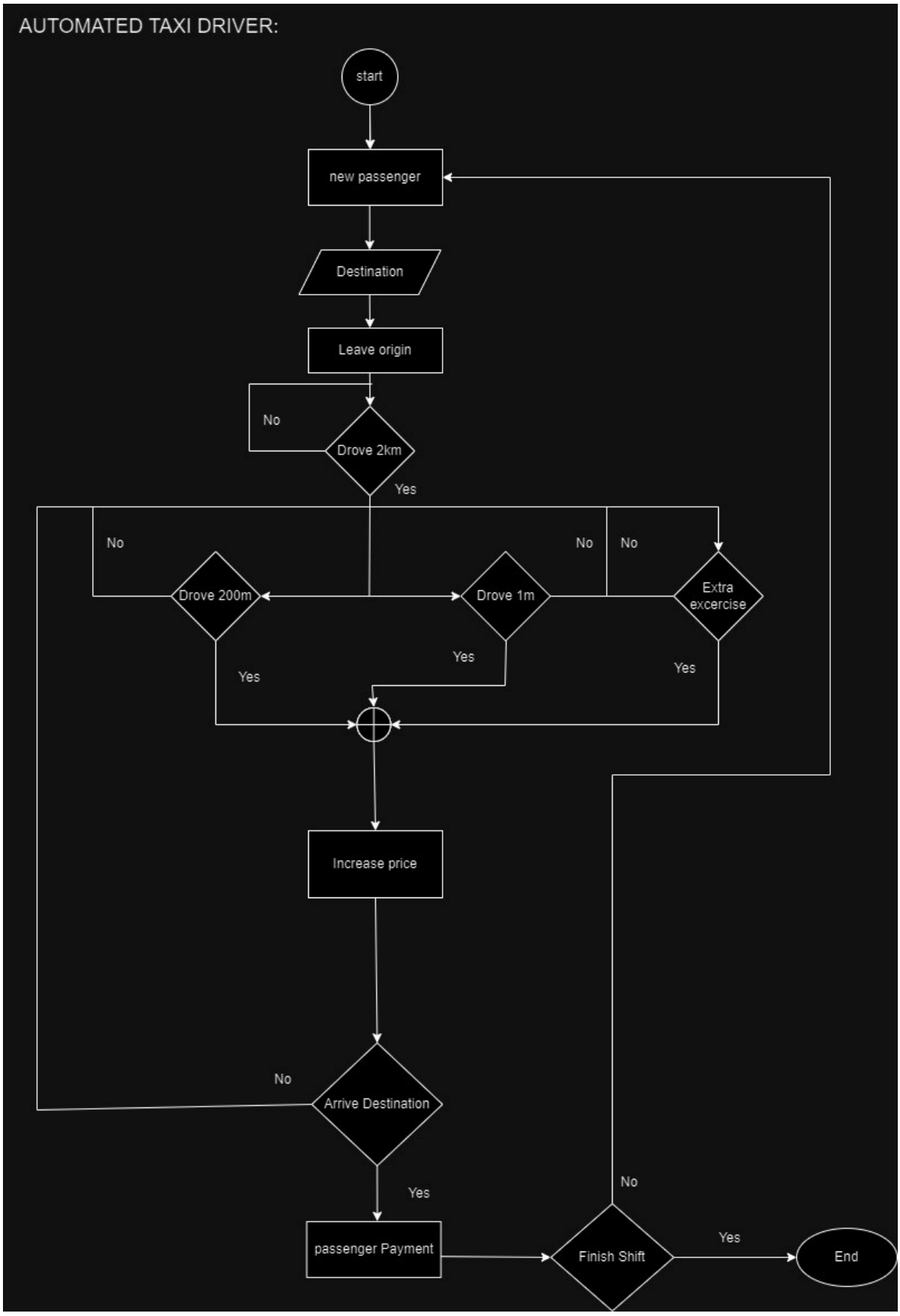
Automated English Tutor: Automated English tutors employ AI to offer personalized language learning with interactive lessons and real-time feedback, revolutionizing education for global learners. Through advanced algorithms, they enhance grammar, pronunciation, and comprehension, optimizing the learning process.

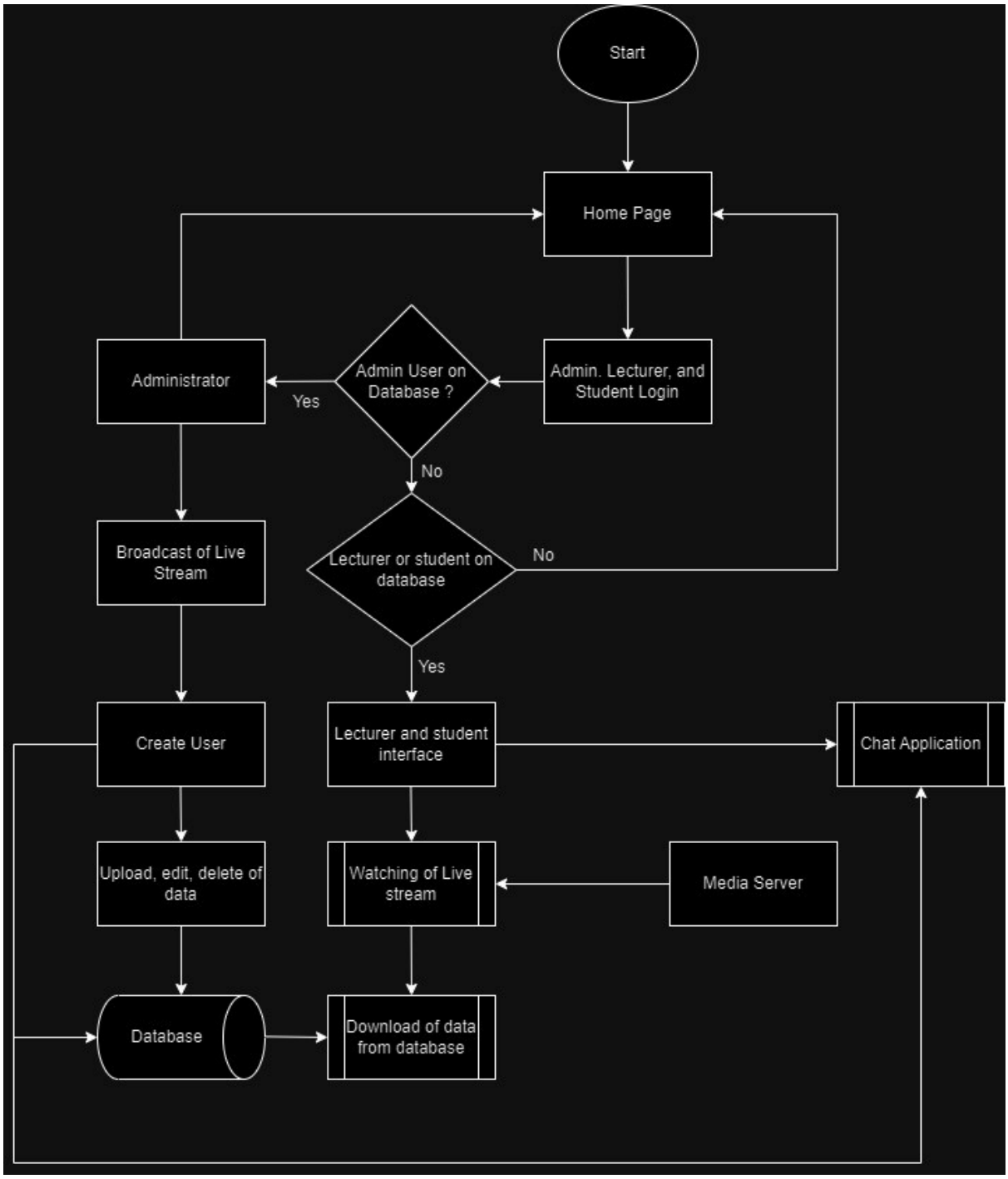
1. Agent: Automated English Tutor
2. Performance measure: Maximize student'sscore on test
3. Environment: Set of students
4. Actuators: Screen display (Exercises, suggestions, corrections)
5. Sensors: Keyboard

Automated Diagnosis System: Automated diagnosis systems employ AI algorithms to analyze symptoms and medical data for accurate disease identification. Through machine learning, they continuously refine their diagnostic accuracy, aiding healthcare professionals in timely and precise treatment decision

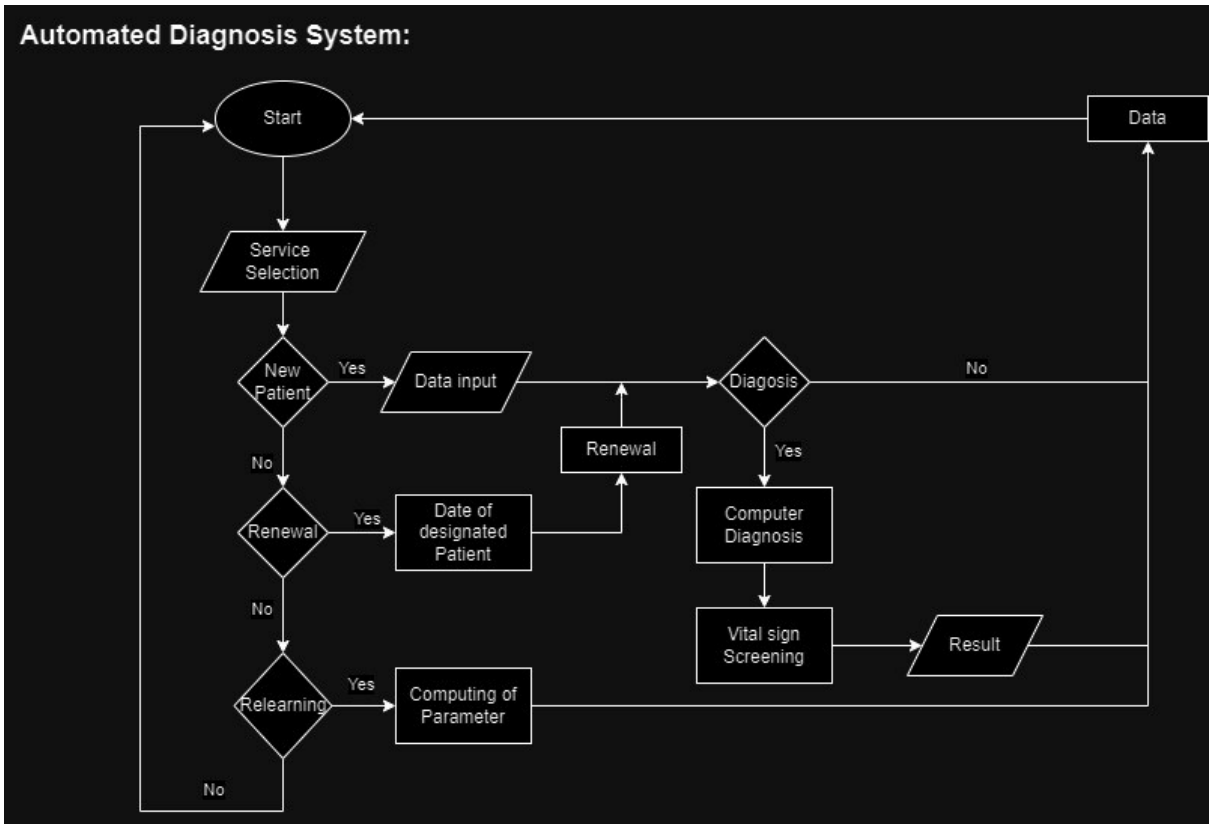
1. Agent: Medical diagnosissystem
2. Performance: Healthy patient, minimize costs, lawsuits
3. Environment: Patient, hospital, staff
4. Actuators: Screen display (questions, tests, diagnosis, treatments, referrals)
5. Sensors: Keyboard (entry of symptoms, finding, patient's answers)

Output:


AUTOMATED TAXI DRIVER:
start
new passenger
Destination
Leave origin
No
Drove 2km
Yes
No
No
No
Drove 200m
Drove 1m
Extra excercise
Yes
Yes
Yes
Increase price
No
Arrive Destination
Yes
No
passenger Payment
Finish Shift
Yes
End


Start
Home Page
Administrator
Admin User on Database ?
Admin. Lecturer, and Student Login
Yes
No
Broadcast of Live Stream
Lecturer or student on database
No
Yes
Create User
Lecturer and student interface
Chat Application
Upload, edit, delete of data
Watching of Live stream
Media Server
Database
Download of data from database

Automated Diagnosis System:



EXPERIMENT NO-02

Aim: Implement Water Jug Problem Using A star.

Description: Water jug problem, also known as the water pouring problem, is a classic puzzle in mathematics and computer science. It involves using two or more jugs of different sizes to measure out a specific volume of water, typically using a series of pouring actions (filling, emptying, or transferring water between jugs) to achieve the desired volume. The challenge often lies in figuring out the optimal sequence of actions to reach the target volume while minimizing the number of steps or water waste. Water Jug Problem can be defined as You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring mark on it. There is a pump that can be used to fill the jugs with water. How can you get exactly gallons of waterinto the 4-gallon jug. Solving the water jug problem often involves logical thinking and strategic planning. Players must consider the capacities of the jugs, the actions they can perform with them (filling, emptying, pouring), and how these actions can be combined to reach the target volume. It's not just about pouring water randomly; it's about finding a systematic approach to achieve the desired result while minimizing waste and using the fewest possible steps

ALGORITHM:

Step-1: Start

Step-2: Fill the 3-litre jug completely.

Step-3: Transfer 3 litres from a 3-liter jug to a 4-liter jug.

Step-4: Now, fill the 3-liter jug fully.

Step-5: Pour 1 litre from a 3-liter jug into a 4-liter jug.

Step-6: Now, in 3-litre jug 2-litre water will be left

Step-7: Empty the 4-litre Jug

Step-8: Pour 2-litre from a 3-litre jug into a 4-litre jug

Step-9: Therefore, in a 4-litre jug 2-litre water will be present. Hence, Goal

State reached

Step-10: End

Code: Tensorflow and Keras

```
from collections import deque
```

```
# Function to pour water from one jug to another
def pour(state, jug1, jug2):
    amt = min(state[jug1], (jug_caps[jug2] - state[jug2]))
    new_state = list(state)
    new_state[jug1] -= amt
    new_state[jug2] += amt
    return tuple(new_state)
# Function to generate successor states
def get_successors(state):
    successors = []
    # Pouring operations between jugs
    for jug1, jug2 in [(0, 1), (1, 0)]:
        new_state = pour(state, jug1, jug2)
        if new_state != state:
            successors.append(new_state)
    # Fill operations for each jug
    for jug in [0, 1]:
        new_state = list(state)
        new_state[jug] = jug_caps[jug]
        successors.append(tuple(new_state))
    # Empty operations for each jug
    for jug in [0, 1]:
        new_state = list(state)
        new_state[jug] = 0
```

```
            successors.append(tuple(new_state))
        return successors
    # Heuristic function for A* algorithm
    def heuristic(state, goal):
        return sum(abs(state[i] - goal[i]) for i in
       range(len(state)))
    # A* search algorithm
    def a_star(start, goal):
        open_list = [(heuristic(start, goal), start)]
        closed_list = set()
        parent = {start: None}
        while open_list:
            _, curr_state = open_list.pop(0)
            if curr_state == goal:
                # Reconstructing the path
                path = deque()
                state = curr_state
                while state is not None:
                    path.appendleft(state)
                    state = parent[state]
                return list(path)
            closed_list.add(curr_state)
            for succ_state in get_successors(curr_state):
                if succ_state not in closed_list:
                    succ_cost = heuristic(succ_state, goal)
```

```
                open_list.append((succ_cost, succ_state))
        open_list.sort()  # Sorting based on heuristic cost
        parent[succ_state] = curr_state
    return None
# Jug capacities and initial/final states
jug_caps = (4, 3)
start_state = (0, 0)
goal_state = (2, 0)
# Running A* algorithm and printing solution
solution = a_star(start_state, goal_state)
if solution:
    print("Solution:")
    for state in solution:
        print(state)
else:
    print("No solution exists.")
```

Output:

```
Finished in 42 ms
Solution:
(0, 0)
(4, 0)
(1, 3)
(1, 0)
(0, 1)
(4, 1)
(2, 3)
(2, 0)
```

EXPERIMENT NO-03

Aim: Implement an 8-puzzle problem solver using Heuristic search technique.

Description: The A* algorithm is a search algorithm that is widely used in artificial intelligence and game development. It combines the use of a heuristic function and a cost function to estimate the cost of reaching the goal state and measure the actual cost of the path taken to reach the current state. The algorithm is useful for finding the shortest path between two points in a graph, and it updates the cost of each node in the graph based on the cost function and the heuristic function. The algorithm continues to expand nodes until it reaches the goal state or determines that there is no path to the goal state. The A* algorithm is known for its efficiency and effectiveness in finding the shortest path and has been used in various real-world applications.

Algorithm:

```
import heapq

class Node:

    def __init__(self, g_value, h_value, i_cell, j_cell,

shuffled_matrix):

        self.g_value = g_value

        self.h_value = h_value

        self.i_cell = i_cell

        self.j_cell = j_cell

        self.shuffled_matrix = shuffled_matrix

    # comparators used for arranging nodes based on (g+h) values

where g is depth and h is no of misplaced

    def __lt__(self, other):

        return (self.g_value + self.h_value) < (other.g_value +

other.h_value)

    def __eq__(self, other):

        return (self.g_value + self.h_value) == (other.g_value +

other.h_value)

# function used for checking curr state of matrix is already

visited or not
```

```
def is_visited(visited, new_matrix):
    for visited_matrix in visited:
        if visited_matrix == new_matrix:
            return True
    return False
# finding shuffled matrix from by swapping curr space cell with
(new_i,new_j)
def find_shuffled_matrix(original, i, j, new_i, new_j):
    shuffled = [row[:] for row in original]
    shuffled[i][j],shuffled[new_i][new_j]=
 shuffled[new_i][new_j], shuffled[i][j]
    return shuffled
# function used for counting no of misplaced cells
def compare(original, shuffled):
    count = 0
    for i in range(len(original)):
        for j in range(len(original[0])):
            if original[i][j] != shuffled[i][j]:
                count += 1
    return count
# function used for printing
def print_function(Node, k):
    if k == 0:
        print("shifting up")
    elif k == 1:
        print("shifting right")
```

```
    elif k == 2:
        print("shifting down")
    else:
        print("shifting left")
    print("shuffled matrix is", Node.shuffled_matrix)
    print("h_value", Node.h_value)
    print("g_value", Node.g_value)
    print(end='\n')
def Heuristic_search(source, original, goal_state):
    row = len(original)
    col = len(original[0])
    # dx dy used for coordinate shift
    dx = [-1, 0, 1, 0]
    dy = [0, 1, 0, -1]
    # list used for storing visited matrices
    visited = [original]
    paths = []
    # min heap used for arranging nodes based on g+h value
    pq = [(0, source)]
    while pq:
        _, curr_node = heapq.heappop(pq)
        paths.append(curr_node)
        if curr_node.shuffled_matrix == goal_state:
            print("Goal state reached")
            return curr_node, paths
```

```
        print("curr matrix is", curr_node.shuffled_matrix)
        print(end='\n')
        for k in range(4):
            new_i = dx[k] + curr_node.i_cell
            new_j = dy[k] + curr_node.j_cell
            # condition used for checking whether new_i and
new_j are valid indices or not
            if 0 <= new_i < row and 0 <= new_j < col:
                neighbour = Node(0, 0, 0, 0, [])
                neighbour.shuffled_matrix =

find_shuffled_matrix(curr_node.shuffled_matrix,
curr_node.i_cell,curr_node.j_cell, new_i, new_j)

                neighbour.g_value = curr_node.g_value + 1
                neighbour.h_value=
compare(neighbour.shuffled_matrix, goal_state)
                neighbour.i_cell = new_i
                neighbour.j_cell = new_j
                if not is_visited(visited,
neighbour.shuffled_matrix):
                    heapq.heappush(pq, (neighbour.g_value +
 neighbour.h_value, neighbour))
                    print_function(neighbour, k)
                    visited.append(neighbour.shuffled_matrix)
```

```
row = 3
col = 3
original = [[1, 2, 3], [-1, 4, 6], [7, 5, 8]]
goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, -1]]
source = Node(0, compare(original, goal_state), 0, 0, original)
for i in range(row):
    for j in range(col):
        if original[i][j] == -1:
            source.i_cell = i
            source.j_cell = j
            break
target, paths = Heuristic_search (source, original, goal_state)
print("h_value:", target.h_value)
print("g_value:", target.g_value)
print("\nPath_of_Matrices:\n")
for node in paths:
    curr_matrix = node.shuffled_matrix
    for row in curr_matrix:
        print(*row)
    print()
```

Output:

```
Finished in 34 ms
curr matrix is [[1, 2, 3], [-1, 4, 6], [7, 5, 8]]

shifting up
shuffled matrix is [[-1, 2, 3], [1, 4, 6], [7, 5, 8]]
h_value 5
g_value 1

shifting right
shuffled matrix is [[1, 2, 3], [4, -1, 6], [7, 5, 8]]
h_value 3
g_value 1

shifting down
shuffled matrix is [[1, 2, 3], [7, 4, 6], [-1, 5, 8]]
h_value 5
g_value 1
```

```
curr matrix is [[1, 2, 3], [4, -1, 6], [7, 5, 8]]

shifting up
shuffled matrix is [[1, -1, 3], [4, 2, 6], [7, 5, 8]]
h_value 4
g_value 2

shifting right
shuffled matrix is [[1, 2, 3], [4, 6, -1], [7, 5, 8]]
h_value 4
g_value 2

shifting down
shuffled matrix is [[1, 2, 3], [4, 5, 6], [7, -1, 8]]
h_value 2
g_value 2
```

```
curr matrix is [[1, 2, 3], [4, 5, 6], [7, -1, 8]]

shifting right
shuffled matrix is [[1, 2, 3], [4, 5, 6], [7, 8, -1]]
h_value 0
g_value 3

shifting left
shuffled matrix is [[1, 2, 3], [4, 5, 6], [-1, 7, 8]]
h_value 3
g_value 3
```

```
Goal state reached
h_value: 0
g_value: 3

Path_of_Matrices:

1 2 3
-1 4 6
7 5 8

1 2 3
4 -1 6
7 5 8

1 2 3
4 5 6
7 -1 8

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

EXPERIMENT NO-04

Aim: Implement Constraint Satisfaction Problem

Description:

CSPs are basically mathematical problems that are defined as a set of variables that must satisfy a number of constraints. When we arrive at the final solution, the states of the variables must obey all the constraints. This technique represents the entities involved in a given problem asa collection of a fixed number of constraints over variables. These variables need to be solved by constraint satisfaction methods. Let's use the Constraint Satisfaction framework to solve the region-colouring problem.

Variables: A CSP involves a set of variables, each of which can take on values from a specified domain. These variables represent the elements of the problem that we are trying to find a solution for. For example, in a scheduling problem, variables might represent tasks to be scheduled.

Domains: Each variable has a domain, which is the set of possible values it can take. The domain might be finite or infinite, depending on the problem. For example, in a scheduling problem, the domain of each task variable might be the set of possible time slots when the task could be scheduled.

Constraints: Constraints specify the relationships or restrictions between variables. These constraints limit the combinations of values that are valid for the variables. For example, in a scheduling problem, constraints might specify that certain tasks cannot be scheduled at the same time, or that certain tasks must be scheduled before others.

Goal: The goal of a CSP is to find an assignment of values to the variables that satisfies all of the constraints. This assignment is called a solution to the problem. Depending on the problem, there may be multiple solutions, or there may be none.

Solving Methods: Various algorithms can be used to solve CSPs. Backtracking search is a common approach, where the search systematically explores different assignments of values to variables, backtracking when it reaches a dead end. Constraint propagation techniques can also be used to simplify the problem by deducing additional constraints from the existing ones. Additionally, local search algorithms such as constraint satisfaction algorithms iteratively improve solutions by making small modifications.

Complexity: The complexity of solving CSPs can vary depending on factors such as the size of the problem, the structure of the constraints, and the properties of the domains. Some CSPs can be solved efficiently in polynomial time, while others may be NP-complete or even undecidable.

ALGORITHM:

Step-1: Start

Step-2: Define a function constraint_func(names, values):

Step-2.1: return values[0] != values[1]

Step-3: Declare names

Step-4: Declare colors

Step-5: Declare constraints

Step-6: problem = CspProblem(names, colors, constraints)

Step-7: output = backtrack(problem)

Step-8: print('\nColor mapping:\n')

Step-9: for k, v in output.items():

Step-9.1: print(k, '==>', v)

Step-10: End

Code:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for _ in range(vertices)] for _ in
range(vertices)]
    # Function to check if the current color assignment is
 safe for vertex v
    def is_safe(self, v, colour, c):
        for i in range(self.V):
            if self.graph[v][i] == 1 and colour[i] == c:
                return False
        return True
    # Recursive function to solve graph coloring problem
    def graph_colour_util(self, m, colour, v):
        if v == self.V:
```

```
            return True
        for c in range(1, m + 1):
            if self.is_safe(v, colour, c):
                colour[v] = c
                if self.graph_colour_util(m, colour, v + 1):
                    return True
                colour[v] = 0
    # Main function to solve graph coloring problem
    def graph_colouring(self, m):
        colour = [0] * self.V
        if not self.graph_colour_util(m, colour, 0):
            print("Solution does not exist")
            return False

        print("Solution exists and following are the assigned
colors:")
        for c in colour:
            print(c, end=" ")
        return True
# Example usage:
g = Graph(4)
g.graph = [[0, 1, 1, 1],
           [1, 0, 1, 0],
           [1, 1, 0, 1],
           [1, 0, 1, 0]]

m = 3  # Number of colors
g.graph_colouring(m)
```

Output:

```
Finished in 32 ms
Solution exists and following are the assigned colors:
1 2 3 2
```

EXPERIMENT NO – 05

Aim: To implement a program for game search.

Description:

Minimax: The Minimax algorithm is a decision-making algorithm used in game theory and artificial intelligence. It is used to determine the best move for a player in a two-player zero-sum game where the outcome depends on the choices made by both players. The algorithm works by exploring the game tree to a certain depth, assigning a score to each possible outcome, and selecting the move that minimizes the maximum loss. The algorithm alternates between maximizing and minimizing phases, and while it is effective for determining the best move, it can be computationally expensive for large game trees. Improvements such as alpha-beta pruning can be used to improve its efficiency.

Alpha Beta Pruning: Alpha-beta pruning is an optimization technique used in the Minimax algorithmto reduce the number of nodes that need to be explored in a game tree. It maintains two values, alpha and beta, to represent the best score achievable bythe maximizer and the minimizer, respectively. If the algorithm finds a move that leads to a worse outcome than the current alpha or beta value, it can safely prune that part of the tree because it will not be chosen. This allows the algorithm to skip large portions of the game tree and significantly reduces the number of nodes that need to be explored, making the algorithm more efficient.

Algorithm:

Minimax

Step-1: Start

Step-2: Construct the game tree representing all possible moves and outcomesof the game.

Step-3: Assign a score to each terminal node of the tree (i.e., the leaves) basedon the outcome of the game.

Step-4: Work back up the tree from the leaves to the root, assigning scores toeach non-terminal node based on the scores of its children

Step-5: For each maximizing node, choose the child with the highest score asthe optimal move.

Step-6: For each minimizing node, choose the child with the lowest score asthe optimal move.

Step-7: Continue recursively until the root node is reached, which representsthe optimal move for the current player.

Step-8 : END

Alpha-Beta Pruning:

Step-1: START

Step-2: Construct the game tree representing all possible moves and outcomesof the game.

Step-3: Assign a score to each terminal node of the tree (i.e., the leaves) basedon the outcome of the game.

Step-4: Work back up the tree from the leaves to the root, assigning scores toeach non-terminal node based on the scores of its children.

Step-5: For each maximizing node, choose the child with the highest score asthe optimal move and update the alpha value accordingly.

Step-6: For each minimizing node, choose the child with the lowest score asthe optimal move and update the beta value accordingly.

Step-7: If the alpha value becomes greater than or equal to the beta value atany point during the search, prune the remaining children of that node and return the current score.

Step-8: Continue recursively until the root node is reached, which representsthe optimal move for the current player.

Code:

Minimax

```
import math

def minimax(curDepth, nodeIndex, maxTurn, scores, targetDepth):

    # If the current depth equals the target depth, return the score

of the current node

    if curDepth == targetDepth:

        return scores[nodeIndex]
```

```
    # If it's the maximizing player's turn
    if maxTurn:
        # Return the maximum value of the children nodes
        return max(minimax(curDepth + 1, nodeIndex * 2, False,
scores, targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1, False,
scores, targetDepth))
    else:
        # If it's the minimizing player's turn, return the minimum
value of the children nodes
        return min(minimax(curDepth + 1, nodeIndex * 2, True, scores,
 targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1, True,
 scores, targetDepth))
# Sample scores for the nodes in the tree
scores = [3, 5, 2, 17, 2, 5, 23, 23]
# Calculate the depth of the tree
treeDepth = math.log(len(scores), 2)
# Perform the minimax algorithm starting from the root node
print("The optimal value is:", end=" ")
print(minimax(0, 0, True, scores, treeDepth))
```

Output:

```
The optimal value is : 12
>>>
```

Alpha-Beta Pruning:

MAX, MIN = 1000, -1000

```
def minimax(depth, nodeIndex, maximizingPlayer, values, alpha,
beta):
    # If we have reached the maximum depth, return the value of
the current node
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:
        # If it's the maximizing player's turn
        best = MIN
        for i in range(0, 2):
            # Recursively call minimax for the child nodes
            val = minimax(depth + 1, nodeIndex * 2 + i, False,
values, alpha, beta)
            # Update the best value
            best = max(best, val)
            # Update alpha with the maximum value seen so far
            alpha = max(alpha, best)
            # Perform alpha-beta pruning
            if beta <= alpha:
                break
        return best
    else:
```

```
        # If it's the minimizing player's turn
        best = MAX
        for i in range(0, 2):
            # Recursively call minimax for the child nodes
            val = minimax(depth + 1, nodeIndex * 2 + i, True,
values, alpha, beta)
            # Update the best value
            best = min(best, val)
            # Update beta with the minimum value seen so far
            beta = min(beta, best)
            # Perform alpha-beta pruning
            if beta <= alpha:
                break
        return best

# Sample values for the nodes in the tree
values = [3, 18, 6, 9, 1, 2, 0, -1]

# Perform minimax algorithm starting from the root node
print("The optimal value is:", minimax(0, 0, True, values, MIN,
MAX))
```

Output:

```
The optimal value is : 5
>>>
```

EXPERIMENT NO – 06(a)

Aim: Implement Bayesian network

Description:

A Bayesian Network falls under the category of Probabilistic Graphical Modelling (PGM) technique that is used to compute uncertainties by using the concept of probability. Popularly known as Belief Networks, Bayesian Networks are used to model uncertainties by using Directed Acyclic Graphs (DAG).

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles

Algorithm:

Step-1: Start

Step-2: Import necessary packages and modules like numpy, pandas, pgmpy, urllib, etc.

Step-3: Define the column names of the dataset in a list.

Step-4: Read the heart disease dataset using pandas read_csv function and replace missing values denoted by '?' with NaN.

Step-5: Define a Bayesian network model using pgmpy's BayesianModel class and specify the directed edges between the nodes.

Step-6: Fit the model to the dataset using MaximumLikelihoodEstimator estimator.

Step-7: Create an instance of VariableElimination class from pgmpy.inference module.

Step-8: Query the model to find the probability distribution of the target variable given the evidence that age is 37, using the query method of the HeartDisease_infer instance.

Step-9: Print the query result.

Step-10: End

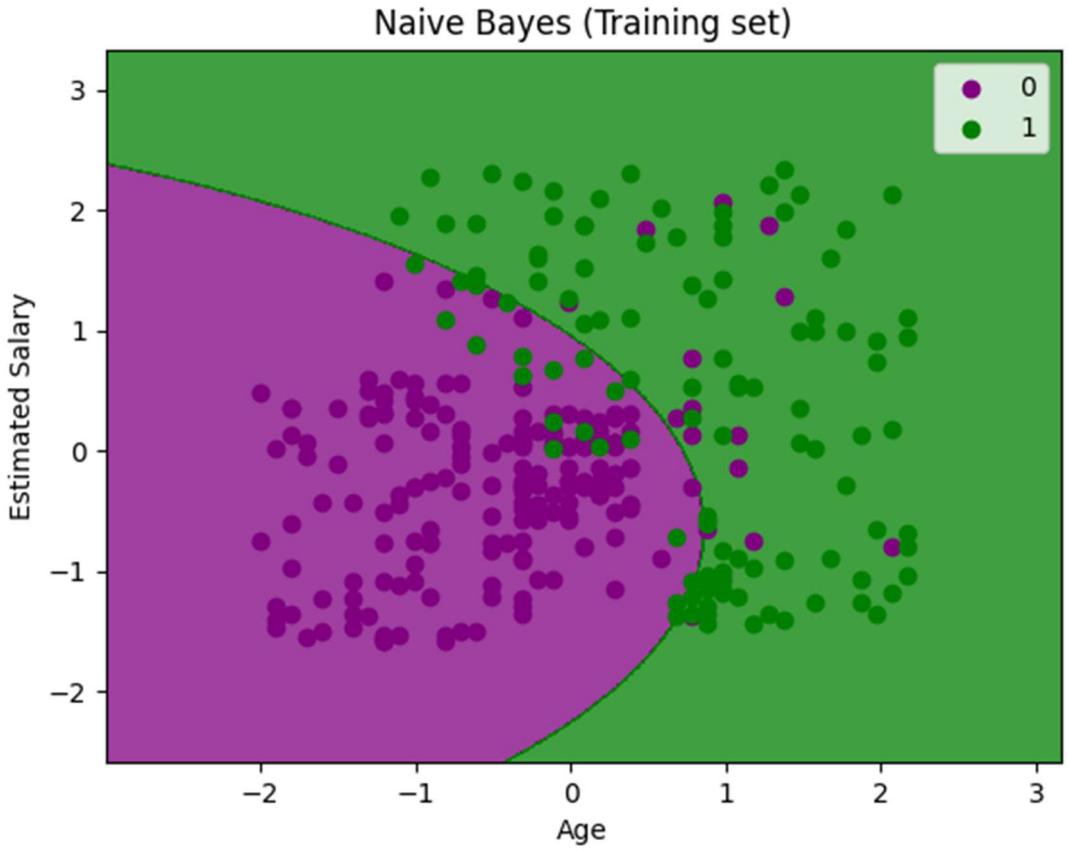
Program:

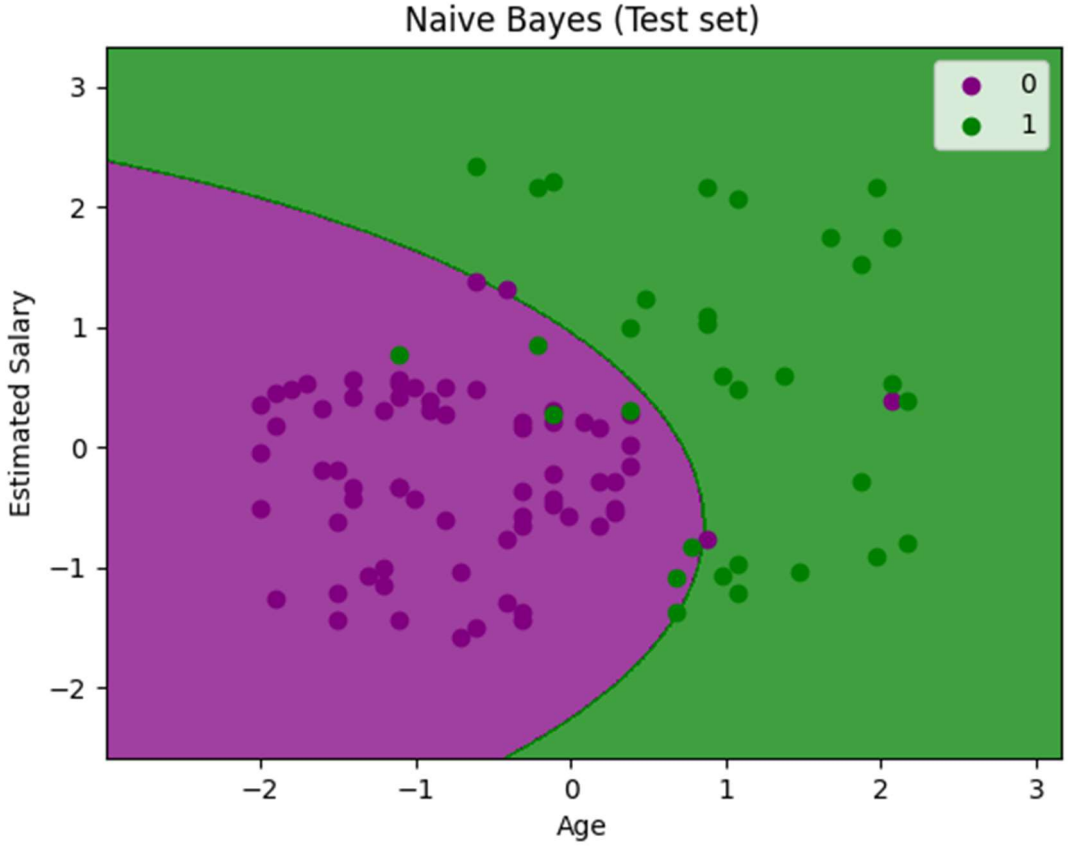
```
import numpy as nm
import matplotib.pyplot as mtp
import pandas as pd
# Importing the dataset
dataset=pd.read_csv('User_Data.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4] values
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size
=0.25, random_state = 0)
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
x_train = sc.fit_transform(x_train)
x_test =sc.transform(x_test)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train,y_train)
#Predicting the Test set results

y_pred = classifier.predict(x_test)
```

```
#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
#Visualising the Training set results
from matplotlib.colors import ListedColormap
x_set, y_set =x_train, y_train
Xl,X2 = nm.meshgrid(mn.arange{start = x_set[:,O].min() - 1,stop
=x_set[:.O].max()+ 1,step
=0.01),
nm.arange(start = x_set[:,l).min() -1,stop= x_set[:,l].max() +
1,step=0.01))
mtp.contourf(Xl, X2,
classifier.predict(nm.array([Xl.ravel(),X2.ravel()]).T).reshape(Xl.
shape),
alpha =0.75, cmap =ListedColormap(('purple','green')))
mtp.xlim(Xl.min(). Xl.max())
mtp.ylim(X2.min(), X2.max())
for i,j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set =J,0], x_set[y_set==j, 1],
c= listedColormap(('purple'. 'green'))(i), label =j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age ')
mtp.ylabefl'Estimated Salary' )
mtp.legend()
mtp.show()
```

```
# Vlsualisln,e the Test set results
from matplotlib.colors import ListedColormap
x_set, y_set =x_test, y_test
X1,X2 = nm.meshgrid(mn.arange{start = x_set[:,O].min() - 1,stop
=x_set[:.O].max()+ 1,step
=0.01),
nm.arange(start = x_set[:,l).min() -1,stop= x_set[:,l].max() +
1,step=0.01))

mtp.contourf(Xl, X2,
classifier.predict(nm.array([Xl.ravel(),X2.ravel()]).T).reshape(Xl.
shape),
alpha =0.75, cmap =ListedColormap(('purple','green')))
mtp.xlim(Xl.min(). Xl.max())
mtp.ylim(X2.min(), X2.max())
for i,j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set =J,0], x_set[y_set==j, 1],
c= listedColormap(('purple'. 'green'))(i), label =j)
mtp.title('Naive Bayes (Test set)')
mtp.xlabel('Age ')
mtp.ylabefl'Estimated Salary' )
mtp.legend()
mtp.show()
```

Output:


Naive Bayes (Training set)
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3


Naive Bayes (Test set)
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

Conclusion:

We have successfully implemented a Bayesian network from a given data and infer the data from the Bayesian network

EXPERIMENT NO – 06(b)

Aim: Infer the data from the Bayesian network.

Description:

A Bayesian Network is a Directed Acyclic Graph in which each edge is corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian Network consist of 2 major parts:

1.A Directed Cyclic Graph

2. Set of Conditional Probability Distributions

A Directed Acyclic Graph::-A Directed Acyclic Graph is used to represent a Bayesian Network and like any other statistical graph, a DAG contains a set of nodes and links, where the links denote the relationship between the nodes.

Directed Acyclic Graphs - Bayesian Networks

The nodes here represent random variables and the edges define the relationship between these variables. But what do these graphs model? What output can you get from a DAG?

A DAG models the uncertainty of an event occurring based on the Conditional Probability Distribution (CDP) of each random variable. A Conditional Probability Table (CPT) is used to represent the CPD of each variable in the network.

Program:

```
from pgmpy.models import BayesianNetwork

from pgmpy.factors.discrete import TabularCPD

import networkx as nx

import pylab as plt

# Defining Bayesian Structure

model = BayesianNetwork([('Guest', 'Host'), ('Price', 'Host')])

# Defining the CPDs:
```

```
cpd_guest = TabularCPD('Guest', 3, [[0.33], [0.33], [0.33]])

cpd_price = TabularCPD('Price', 3, [[0.33], [0.33], [0.33]])

cpd_host = TabularCPD('Host', 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])

# Associating the CPDs with the network structure.

model.add_cpds(cpd_guest, cpd_price, cpd_host)

#Now we will check the model structure and associated conditional
probability

model.check_model()

# Infering the posterior probability

from pgmpy.inference import VariableElimination

infer = VariableElimination(model)

posterior_p = infer.query(['Host'], evidence={'Guest': 2, 'Price':
2})

print(posterior_p)
```

Output:

```
+---------+-------------+
| Host    |   phi(Host) |
+=========+=============+
| Host(0) |      0.5000 |
+---------+-------------+
| Host(1) |      0.5000 |
+---------+-------------+
| Host(2) |      0.0000 |
+---------+-------------
```

EXPERIMENT NO – 07(a)

Aim: Implement a MDP to run value iteration

Description:

MDP: A Markov Decision Process (MDP) is a mathematical framework used to model decision-making problems in situations where outcomes are partly random andpartly under the control of a decision-maker. The goal of MDP is to find the optimal policy for a given environment, which is a mapping from states to actions that maximizes the expected total reward received by the agent.

Value iteration is an algorithm used to solve MDPs by iteratively computing theoptimal values of each state. The algorithm starts with an initial guess for the value of each state and repeatedly updates them until convergence. At each iteration, the algorithm applies the Bellman update equation, which expresses the value of a state as the sum of the immediate reward and the discounted value of the next state. The discount factor is a parameter that determines the relative importance of immediate and future rewards.

Algorithm:

Step-1: Start

Step-2: Initialize the parameters including reward, discount, maximum error, number of actions, actions, number of rows, number of columns, and the utilitymatrix U.

Step-3: Define a function named 'printEnvironment' that takes the array and policy as parameters, then iterates over the rows and columns of the array, andprints the corresponding values according to the given conditions.

Step-4: Define a function named 'getU' that takes U, row number r, column number c, and action as parameters. It then calculates the new row and columnnumber after applying the given action and returns the value of the utility at that position. If the position is outside the matrix or corresponds to the wall, then itreturns the utility value of the current position.

Step-5: Define a function named 'calculateU' that takes U, row number r, column number c, and action as parameters. It calculates the utility for the given actionat the given position using the formula provided.

Step-6: Define a function named 'valueIteration' that takes U as a parameter. Ititeratively updates the utility matrix by calculating the maximum utility over all actions at each position until the maximum error is less than the specified

threshold. It prints the intermediate utilities using the 'printEnvironment' function.

Step-7: Define a function named 'getOptimalPolicy' that takes U as a parameter.It calculates the optimal policy by finding the action that gives the maximum utility at each position.

Step-8: Print the initial utility matrix using the 'printEnvironment' function. Step-9: Call the 'valueIteration' function with the initial utility matrix U andobtain the final utility matrix.

Step-10: Call the 'getOptimalPolicy' function with the final utility matrix to getthe optimal policy.

Step-11: Print the optimal policy using the 'printEnvironment' function.

Step-12: End

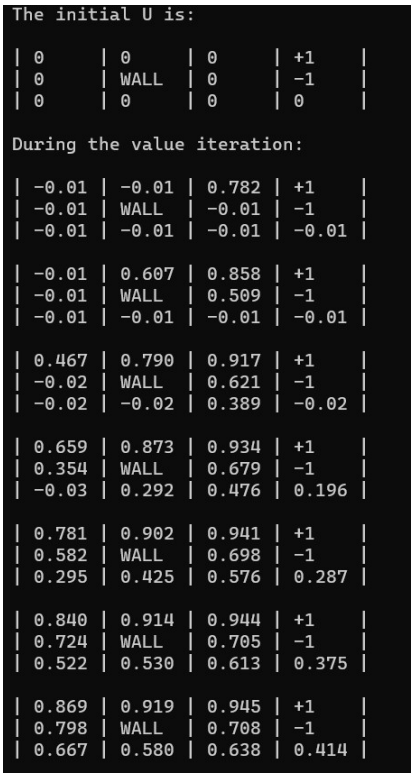
Program:

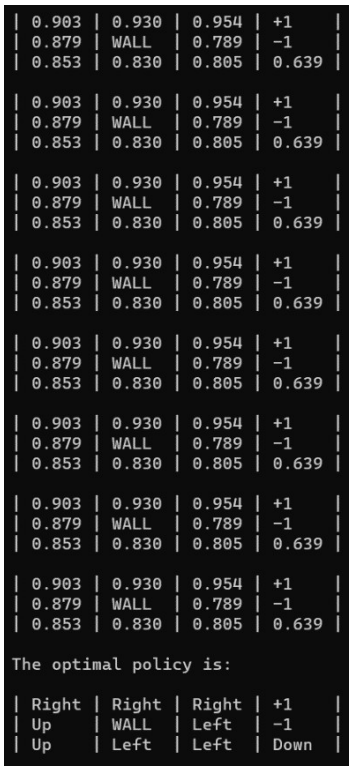
```
REWARD = -0.01
DISCOUNT = 0.99
MAX_ERROR = 10**(-
3) NUM_ACTIONS = 4
ACTIONS = [(1, 0), (0, -1), (-1, 0), (0, 1)]
NUM_ROW = 3
NUM_COL = 4
U = [[0, 0, 0, 1], [0, 0, 0, -1], [0, 0, 0, 0], [0, 0, 0, 0]]
 def printEnvironment(arr,
   policy=False): res = ""
   for r in
     range(NUM_ROW): res
     += "|"
     for c in
       range(NUM_COL): if
       r == c == 1:
         val = "WALL"
       elif r <= 1 and c == 3:
         val = "+1" if r == 0
       else "-1" else:
         if policy:
           val = ["Down", "Left", "Up",
         "Right"][arr[r][c]] else:
           val = str(arr[r][c])
       res += " " +
     val[:5].ljust(5) + " |" res
     += "\n"
   print(res)
```

```
 def getU(U, r, c,
   action): dr, dc =
   ACTIONS[action]
   newR, newC = r+dr,
   c+dc
  if newR < 0 or newC < 0 or newR >= NUM_ROW or newC >=
NUM_COL or (newR == newC == 1):
    return
   U[r][c]
   else:
    return U[newR][newC]
def calculateU(U, r, c,
action):
   u = REWARD
   u += 0.1 * DISCOUNT * getU(U, r, c,
   (action-1)%4) u += 0.8 * DISCOUNT *
   getU(U, r, c, action)
   u += 0.1 * DISCOUNT * getU(U, r, c,
   (action+1)%4) return u
def valueIteration(U):
   print("During the value
   iteration:\n") while True:
    nextU = [[0, 0, 0, 1], [0, 0, 0, -1], [0, 0, 0, 0], [0, 0, 0,
    0]]
    error = 0
    for r in
      range(NUM_ROW): for
      c in range(NUM_COL):
        if (r <= 1 and c == 3) or (r
          == c == 1): continue
       nextU[r][c]=  max([calculateU(U,  r, c,     action)
                  for                actionin
range(NUM_ACTIONS)])
        error = max(error, abs(nextU[r][c]-U[r][c]))
    U = nextU
    printEnvironment
    (U)
    if error < MAX_ERROR * (1-DISCOUNT) / DISCOUNT:
      brea
   k return
   U
def getOptimalPolicy(U):
   policy = [[-1, -1, -1, -1] for i in
   range(NUM_ROW)] for r in
   range(NUM_ROW):
    for c in range(NUM_COL):
      if (r <= 1 and c == 3) or (r
        == c == 1): continue
```

```
      maxAction, maxU = None, -
      float("inf") for action in
      range(NUM_ACTIONS):
        u = calculateU(U, r, c, action
if u > maxU:
          maxAction, maxU =
      action, u policy[r][c] =
      maxAction
  return policy
print("The initial U
is:\n")
printEnvironment(U)
U = valueIteration(U)
policy =
getOptimalPolicy(U)
print("The optimal policy
is:\n")
printEnvironment(policy,
True)
```

Output:





Conclusion:

We have successfully implemented a MDP to run value iteration in any environment.

EXPERIMENT NO – 07(b)

Aim: Implement a MDP to run policy iteration

Description:

A Markov Decision Process (MDP) is a mathematical framework used to model decision-making situations in which an agent interacts with an environment thatis affected by random events. Policy iteration is a technique used to solve an MDP and find the optimal policy for an agent.

Algorithm:

Step-1: Start

Step-2: Define the constants and variables used in the MDP, including the reward, discount factor, maximum error, number of actions, action space,number of rows and columns, and initial utility and policy matrices.

Step-3: Define the printEnvironment function to print the current state of the environment, including the utility values or actions at each grid cell.

Step-4: Define the getU function to get the utility value at the given grid cell andaction.

Step-5: Define the calculateU function to calculate the utility value at the givengrid cell and action.

Step-6: Define the policyEvaluation function to evaluate the current policy and calculate the utility values for each grid cell.

Step-7: Define the policyIteration function to perform the policy iteration algorithm, which alternates between policy evaluation and policy improvementuntil the policy converges to an optimal policy.

Step-8: Print the initial random policy.

Step-9: Call the policyIteration function with the initial policy and utility matricesas inputs.

Step-10: Print the optimal policy.

Step-11: End

Program:

```
import random
REWARD = -0.01
DISCOUNT = 0.99
MAX_ERROR = 10**(-
3) NUM_ACTIONS = 4


ACTIONS = [(1, 0), (0, -1), (-1, 0), (0, 1)]
NUM_ROW = 3
NUM_COL = 4
U = [[0, 0, 0, 1], [0, 0, 0, -1], [0, 0, 0, 0], [0, 0, 0, 0]]
policy =  [[random.randint(0,  3)  for  j                in
       range(NUM_COL)]     fori   in range(NUM_ROW)]
 def printEnvironment(arr,
   policy=False): res = ""
   for r in
     range(NUM_ROW): res
     += "|"
     for c in
       range(NUM_COL): if
       r == c == 1:
         val = "WALL"
       elif r <= 1 and c == 3:
         val = "+1" if r == 0
       else "-1" else:
         val = ["Down", "Left", "Up",
       "Right"][arr[r][c]] res += " " +
       val[:5].ljust(5) + " |"
     res +=
   "\n"
   print(res)
 def getU(U, r, c,
   action): dr, dc =
   ACTIONS[action]
   newR, newC = r+dr,
   c+dc
  if newR < 0 or newC < 0 or newR >= NUM_ROW or newC >=
NUM_COL or (newR == newC == 1):
     return
   U[r][c]
   else:
    return U[newR][newC]
def calculateU(U, r, c,
action):
   u = REWARD
```

```
  u += 0.1 * DISCOUNT * getU(U, r, c,
  (action-1)%4) u += 0.8 * DISCOUNT *
  getU(U, r, c, action)
  u += 0.1 * DISCOUNT * getU(U, r, c,
  (action+1)%4) return u
def
  policyEvaluation(policy
  , U): while True:
    nextU = [[0, 0, 0, 1], [0, 0, 0, -1], [0, 0, 0, 0], [0, 0, 0,
    0]]
    error = 0
    for r in
      range(NUM_ROW): for
      c in range(NUM_COL):


        if (r <= 1 and c == 3) or (r
          == c == 1): continue
        nextU[r][c] = calculateU(U, r, c, policy[r][c])
        error = max(error, abs(nextU[r][c]-U[r][c]))
    U = nextU
    if error < MAX_ERROR * (1-DISCOUNT) / DISCOUNT:
      brea
  k return
  U
def policyIteration(policy, U):
  print("During the policy
  iteration:\n") while True:
    U =
    policyEvaluation(policy,
    U) unchanged = True
    for r in
      range(NUM_ROW): for
      c in range(NUM_COL):
        if (r <= 1 and c == 3) or (r
          == c == 1): continue
        maxAction, maxU = None, -
        float("inf") for action in
        range(NUM_ACTIONS):
          u = calculateU(U, r,
          c, action) if u > maxU:
            maxAction, maxU = action, u
        if maxU > calculateU(U, r, c,
          policy[r][c]): policy[r][c] =
          maxAction
          unchanged =
    False if unchanged:
      break
    printEnvironment(poli
```

```
    cy)
  return policy
print("The initial random
policy is:\n")
printEnvironment(policy)
policy =
policyIteration(policy, U)
print("The optimal policy
is:\n")
```

Output:

```
| Up    | Up    | Left  | +1    |
| Up    | WALL  | Down  | -1    |
| Right | Down  | Right | Left  |

During the policy iteration:

| Left  | Left  | Right | +1    |
| Up    | WALL  | Up    | -1    |
| Up    | Left  | Left  | Down  |

| Left  | Right | Right | +1    |
| Up    | WALL  | Up    | -1    |
| Up    | Right | Up    | Down  |

| Right | Right | Right | +1    |
| Down  | WALL  | Left  | -1    |
| Right | Right | Up    | Down  |

| Right | Right | Right | +1    |
| Up    | WALL  | Left  | -1    |
| Right | Right | Up    | Down  |

| Right | Right | Right | +1    |
| Up    | WALL  | Left  | -1    |
| Up    | Right | Up    | Down  |

| Right | Right | Right | +1    |
| Up    | WALL  | Left  | -1    |
| Up    | Left  | Up    | Down  |

| Right | Right | Right | +1    |
| Up    | WALL  | Left  | -1    |
| Up    | Left  | Left  | Down  |

The optimal policy is:

| Right | Right | Right | +1    |
| Up    | WALL  | Left  | -1    |
| Up    | Left  | Left  | Down  |
```

Conclusion:

We have successfully implemented a MDP to run policy iteration in anyenvironment.

EXPERIMENT NO – 08

Aim: Understanding of GitHub and Conda environments

Description:

Version Control Systems:

A version control system tracks any kind of changes made to the project file, why these changes were made, and references to the problems fixed or enhancements introduced. It allows developer teams to manage and track changes in code over time. It allows a person to switch to the previous states of the file, compare the versions and helps to identify issues in a file in a more efficient way.

Git:

- Git is one of the ways of implementing the idea of version control. It is a Distributed Version Control System (DVCS).
- Unlike a Centralized Version Control System that uses a central server to store all files and enables team collaboration, DVCS just can be implemented just with the help of a desktop, single software available at a command line. So, the failure of the central server does not create any problems in DVCS. So, a lot of operations can be performed when you are offline.

Installation of Git:

Installing on Linux

If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the package management tool that comes with your distribution. If you're on Fedora (or any closely-related RPM-based distribution, such as RHEL or CentOS), you can use dnf:

$ sudo dnf install git-all

If you're on a Debian-based distribution, such as Ubuntu, try apt:

$ sudo apt install git-all

For more options, there are instructions for installing on several different Unix distributions on the Git website, at https://git-scm.com/download/linux.

Installing on macOS

There are several ways to install Git on macOS. The easiest is probably to install the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run git from the Terminal the very first time.

$ git --version

If you don't have it installed already, it will prompt you to install it.

If you want a more up to date version, you can also install it via a binary installer. A macOS Git installer is maintained and available for download at the Git website, at https://git-scm.com/download/mac.

Installing on Windows

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to https://git-scm.com/download/win and the download will start automatically. Note that this is a project called Git for Windows, which is separate from Git itself; for more information on it, go to https://gitforwindows.org.

To get an automated installation you can use the Git Chocolatey package. Note that the Chocolatey package is community-maintained.

Creating an account on GitHub

To get started with GitHub, you'll need to create a free personal account on GitHub.com and verify your email address.

Every person who uses GitHub.com signs in to a personal account. Your personal account is your identity on GitHub.com and has a username and profile.

CODE:

Step 1: git config

→ git config --global user.name "My Name"
→ git config --global user.email "myemail@example.com"

In Git, the **git config** command is used to configure settings specific to Git operations. These settings are stored in three distinct scopes: **system, global, and local**.

System Configuration: This scope applies system-wide, affecting all users and repositories on the system.
Configuration settings at this level are stored in the /etc/gitconfig file.

Global Configuration: Global configurations are specific to a single user and apply to all repositories associated with that user.

Local Configuration: Local configurations are specific to a single repository and override settings from higher levels.

Step 2: Create a folder for your project and add the required files

- Create a new folder and open in on your code editor.
- Add the necessary files for your project into the folder

Step 3: git init

→ git init

It is used to create an empty Git repository or reinitialize an existing one

Step 4: git add

The git add command is used to stage files in your project directory for the next commit. It takes one or more paths as arguments and adds the current content of those paths to the staging area. The staging area is a file that Git uses to keep track of which files are going to be included in the next commit.

- **git add <file>** - Stages all changes in <file> for the next commit.
- **git add <directory>** - Stages all changes in <directory> for the next commit.
- **git add .** - Stages all changes in the current working directory for the next commit.
- **git add -A** - Stages all changes in the current working directory and all of its subdirectories for the next commit.

Step 5: git status

The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git.

```
saiki@SaiKiranHP MINGW64 /d/GitWorkshop (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   waterjug.py
```

Step 6: git commit

A Git commit is a snapshot of your repository at a specific time. Commits are the building blocks of "save points" in Git's version control. The git commit command takes several options, but the most important one is the -m option, which allows you to specify a commit message.

Conclusion:

We have successfully understood Git environment.

EXPERIMENT NO – 9

Aim: Use Github Packages and Libraries to frame a standard project and commit back to Github.

Description:

GitHub is a cornerstone of modern software development, providing a robust platform for collaboration, version control, and project management. Developers leverage GitHub to host their code repositories, enabling seamless collaboration among team members regardless of their geographical location. Through features like pull requests, issues tracking, and code reviews, GitHub facilitates efficient communication and code improvement workflows. Its version control system, based on Git, ensures that changes to code are tracked systematically, allowing developers to revert to previous versions if needed and maintaining a clear history of modifications. Moreover, GitHub's integration with various continuous integration and deployment tools streamlines the software development lifecycle, enabling automated testing and deployment processes.

CODE and Execution:

Step 1: Create a remote repository on Github

A remote repository on GitHub serves as a central hub for collaboration, version control, and project management.

- Go to your GitHub account, click on your profile in the top right corner, and click on Your repositories.
- Click on the new icon to create a new repository
- Create a new repository with the required specifications.
- Once the repository is created, click on the Code icon and copy the HTTPS code, as we will use it to push your code.

Step 2: Create a folder for your project and add the required files

- Create a new folder and open in on your code editor.
- Add the necessary files for your project into the folder

Step 3: git init

git init

It is used to create an empty Git repository or reinitialize an existing one

Step 4: git add

The git add command is used to stage files in your project directory for the next commit. It takes one or more paths as arguments and adds the current content of those paths to the staging area. The staging area is a file that Git uses to keep track of which files are going to be included in the next commit.

- **git add <file>** - Stages all changes in <file> for the next commit.
- **git add <directory>** - Stages all changes in <directory> for the next commit.
- **git add .** - Stages all changes in the current working directory for the next commit.
- **git add -A** - Stages all changes in the current working directory and all of its subdirectories for the next commit.

git add .

Step 6: git commit

A Git commit is a snapshot of your repository at a specific time. Commits are the building blocks of "save points" in Git's version control. The git commit command takes several options, but the most important one is the -m option, which allows you to specify a commit message.

git commit -m "Add codes"

Step 7: Adding your new remote repository to your local git repository

The git remote command is essentially an interface for managing a list of remote entries that are stored in the repository's ./. git/config file.

git remote add <name> <url>

Create a new connection to a remote repository. After adding a remote, you'll be able to use ＜name＞ as a convenient shortcut for ＜url＞ in other Git commands.

Step 8: Renaming default branch

This command performs two tasks simultaneously: it renames the current branch to "main" and sets "main" as the new default branch. This command is often used to follow the trend of naming the default branch "main" instead of "master" for better inclusivity and clarity.

git branch -M main

Step 9: git push

The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo. Pushing has the potential to overwrite changes, caution should be taken when pushing.

git push -u origin main

Conclusion:

Git commands were executed and a project was pushed onto a remote repository on GitHub