Artificial Intelligence Lab Report

**Submitted by**

**Aayush Basnet**

BSc. CSIT | Batch of 2078

Symbol: 29131

**Submitted To**

Nawaraj Joshi

Department of Computer Science and Information Technology

Madan Bhandari Memorial College

Baneshor, Kathmandu

Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| S.N. | Tasks | Date of Submission | Signature |
| 1. | Write programs simple chat bot (concepts knowledge representation systems) |  |  |
| 2. | Write programs for DFS, BFS. |  |  |
| 3. | Write programs for Greedy Best First, A\* |  |  |
| 4. | Write programs for to demonstrate water jug, n-queen problem (constraint satisfaction  problem) |  |  |
| 5. | Write program for implementing Naive Bayes. |  |  |
| 6. | Write program for implementing Neural Networks for realization of AND, OR gates. |  |  |
| 7. | Write program for implementing Backpropagation Learning. |  |  |
| 8. | Write program for implementing expert systems for Covid prediction. |  |  |

**Lab 2: Write programs for DFS, BFS.**

**Introduction**

DFS – Depth First Search (DFS) is an algorithm for traversing or searching tree or graph data structures. In DFS, we start at the root and explore as far as possible along each branch before backtracking. The time complexity of the DFS algorithm is represented in the form of O (V + E), where V is the number of nodes and E is the number of edges. The space complexity of the algorithm is O (V).

BFS - Breadth First Search (BFS) is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal. The time complexity of the BFS algorithm is represented in the form of O (V + E), where V is the number of nodes and E is the number of edges. The space complexity of the algorithm is O (V).

**Objective**

1. To implement DFS algorithm.

2. To implement BFS algorithm

**Program Code**

1. DFS Algorithm

inputGraph = {

"A" : ["D","C"],

"B" : ["B", "F"],

"C" : ["F"],

"D" : [“A”],

"E" : [],

"F" : []

}

visitedNodes=[]

def depthFirstSearch(graph, node):

if node not in visitedNodes:

visitedNodes.append(node)

neighbours = graph[node]

for neighbour in neighbours:

depthFirstSearch(graph, neighbour)

return visitedNodes

visitedNodes = depthFirstSearch(inputGraph, 'A')

print(visitedNodes)

**Output**



2. BFS Algorithm

inputGraph = {

"A" : ["D","B"],

"B" : ["F", "A"],

"C" : ["F"],

"D" : [],

"E" : ["E"],

"F" : []

}

visitedNodes=[]

def breadthFirstSearch(inputGraph, startNode):

queue = [startNode]

while queue:

node = queue.pop(0)

if node not in visitedNodes:

visitedNodes.append(node)

neighbours = inputGraph[node]

for neighbour in neighbours:

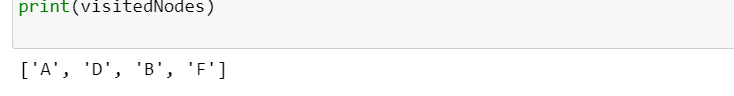
queue.append(neighbour)

return visitedNodes

visitedNotes = breadthFirstSearch(inputGraph, "A")

print(visitedNodes)

**Output**

Fig: BFS

**Conclusion**

Hence, implemented BFS and DFS algorithm using the python programming language.

**Lab 3: Write programs for Greedy Best First, A\***

**Introduction**

Greedy best-first search algorithm – This algorithm always selects the path which appears best at the moment. It is the combination of DFS and BFS algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function

A\* - A\* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A\* search algorithm finds the

shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster.

**Objective**

1. To implement Greedy best-first search algorithm.

2. To implement A\* search algorithm

**Program Code**

1. Greedy best-first

inputGraph = {

'A': [('B', 3), ('D', 8)],

'B': [('A', 6), ('C', 5)],

'C': [('D', 1), ('B', 2)],

'D': [('A', 6), ('C', 8)]

}

goal="C"

def gbfs(graph, start):

queue = [start]

visitedNode = []

while queue:

queue = sorted(queue, key=lambda x: x[1])

node = queue.pop(0)

if node not in visitedNode:

visitedNode.append(node)

if node[0] == goal:

break

neighbours = graph[node[0]]

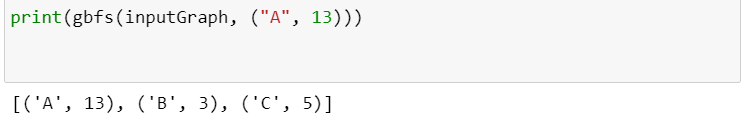
for neighbour in neighbours:

queue.append(neighbour)

return visitedNode

print(gbfs(inputGraph, ("A", 13)))

**Output**



2. A\*

inputGraph = {

'A': [('B', 4, 7), ('D', 7, 4)],

'B': [('A', 4 7), ('C', 3, 8)],

'C': [('D', 7, 3), ('B', 7, 1)],

'D': [('A', 3, 9), ('C', 6, 0)],

}

goal="D"

def gbfs(graph, start):

rootToParentCost = 0

queue = [start + (rootToParentCost, )]

visitedNodes = []

while queue:

queue = sorted(queue, key=lambda x: x[1] + x[2] + x[3])

node = queue.pop(0)

rootToParentCost = node[1] + node[3]

if node not in visitedNodes:

visitedNodes.append(node)

if node[0] == goal:

break

neighbours = graph[node[0]]

for neighbour in neighbours:

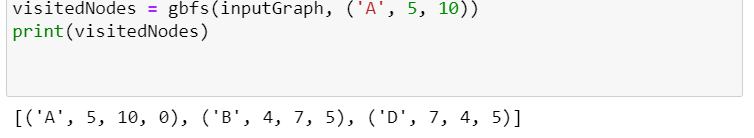
queue.append(neighbour + (rootToParentCost, ))

return visitedNodes

visitedNodes = gbfs(inputGraph, ('A', 5, 10))

print(visitedNodes)

**Output**

****

**Conclusion**

Hence, we implemented greedy best-first and A\* algorithms with the help of python programming language.

**Lab 4: Write programs for to demonstrate water jug, n-queen problem (constraint satisfaction problem)**

**Introduction**

The water jug problem

You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug? To solve this we have to make some assumptions not mentioned in the problem. They are:

∙ We can fill a jug from the pump.

∙ We can pour water out of a jug to the ground.

∙ We can pour water from one jug to another.

∙ There is no measuring device available.

n-Queen problem

The N-Queen problem is to place n Queens in such a manner on an n\*n chessboard that no queens attack each other by being in the same row, column or diagonal. It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - Queen Problem.

**Objective**

1. To implement the water jug problem.

2. To implement the n-Queen problem

**Program Code**

1. Water jug problem

def waterJugSolution(xCapacity, yCapacity, target, reverse=False):

path = [[0, 0]]

y = yCapacity

x = 0

path.append([x, y])

steps = 1

while ((y != target) and (x != target)):

temp = min(y, xCapacity - x)

x = x + temp

y = y - temp

path.append([x, y])

steps = steps + 1

if ((y == target) or (x == target)):

break

if y == 0:

y = yCapacity

path.append([x, y])

steps = steps + 1

if x == xCapacity:

x = 0

path.append([x, y])

steps = steps + 1

if reverse:

path = [[y, x] for x, y in path]

return [path, steps]

2

xCapacity = int(input("Jug 1= "))

yCapacity = int(input("Jug 2= "))

target = int(input("Target vol = "))

def solveGCD(a, b):

if b == 0:

return a

return solveGCD(b, a % b)

if target % solveGCD(xCapacity, yCapacity) == 0:

path1, step1 = waterJugSolution(xCapacity, yCapacity, target, reverse=False) path2, step2 = waterJugSolution(yCapacity, xCapacity, target, reverse=True) if step1 <= step2:

print("The path is:", path1)

print("The steps are:", step1)

else:

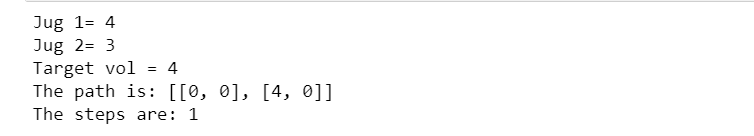
print("The path is:", path2)

print("The steps are:", step2)

else:

print("Solution failed")

**Output**

****

2. n-Queen problem

def nQueenSolution(board, col, N):

if col >= N:

return True

for i in range(N):

val1=0

row=i

for i in range(col):

if board[row][i] == 1:

val1=False

return

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

val1=False

return

for i, j in zip(range(row, N, 1), range(col, -1, -1)):

if board[i][j] == 1:

val1=False

return

val1=True

if val1:

board[i][col] = 1

if nQueenSolution(board, col + 1, N) == True:

return True

board[i][col] = 0

return False

N = int(input("Chesboard size = "))

printoutBoard = [[0] \* N for i in range(N)]

if nQueenSolution(printoutBoard, 0, N) == False:

print("Solution failed")

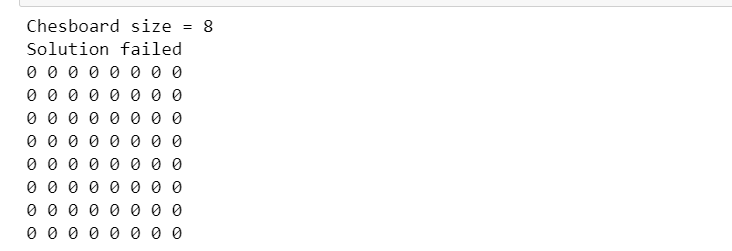
for i in range(N):

for j in range(N):

print(printoutBoard[i][j], end=" ")

print("")

**Output**



**Conclusion**

Hence, we implemented the water jug problem and n Queen Problem using the python programming language as shown in the lab above.

**Lab 1: Write programs simple chat bot (concepts of Knowledge Representation systems)**

**Introduction**

Chatbot can be defined as an application that can be used to conduct a live chat conversation via text. Chatbots are designed to simulate the way human would talk or chat with other person. The term ‘Chatterbot” was originally coined by Michael Maildin in 1994 to describe these conversational programs.

**Objective**

1. To create a simple chatbot.

**Program Code**

print("##############################")

print(" CHATBOT")

print("##############################")

while True:

question = input("Enter your question= ")

question = question.lower()

if question in ["hi","hello"]:

print("Hello")

elif question in ["where do you stay?"]:

print("I stay inside the computer")

elif question in ["who are you?","what are you?"]:

print("I am a bot")

elif question in ["how are you?"]:

print("I am doing fine today")

elif question in ['what are you doing?']:

print("I am watching a movie")

elif question in ["are you busy?"]:

print("Yes, I am working right now")

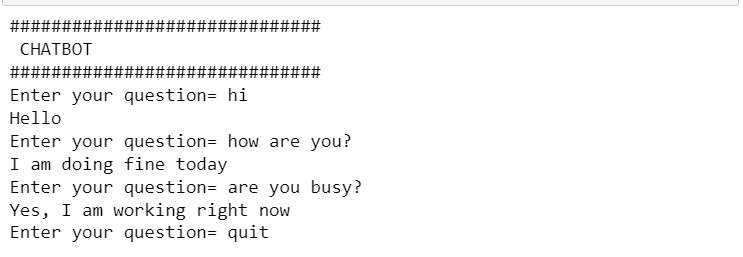
elif question in ['quit']:

break

else:

print("I cannot answer that")

**Output**

****

**Conclusion**

Hence, we created a simple Chabot application which can react to the users input as questions and answer those questions.

**Lab 5: Write program for implementing Naive Bayes.**

**Introduction**

Naïve Bayes algorithm is a probabilistic machine learning algorithm based on the Bayes Theorem which is used in a wide variety of classification tasks. Typical applications of this algorithm include filtering spam, classifying documents, sentiment prediction etc. It is based on the works of Rev. Thomas Bayes and that’s where the name came from. The name naive is used because it assumes the features that go into the model is independent of each other. That is changing the value of one feature, does not directly influence or change the value of any of the other features used in the algorithm.

**Objective**

1. To implement Naïve Bayes algorithm.

**Program Code**

from sklearn import metrics

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

1

trainX, textX, trainY, testY = train\_test\_split(X, y, test\_size=0.6, random\_state=3) gnb = GaussianNB()

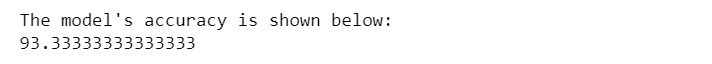
gnb.fit(trainX, trainY)

predictY = gnb.predict(textX)

print("The Models accuracy is shown below: ")

print(metrics.accuracy\_score(testY, predictY)\*100)

**Output**

****

**Conclusion**

Hence, we implemented Naïve Bayes algorithm in the above lab using the python programming language

**Lab 6 : Write program for implementing Neural Networks for realization of AND, OR gates.**

**Introduction**

A neural network is a system of hardware or software patterned after the operation of neurons in the human brain. They are a variety of deep learning technology which also falls under the umbrella of AI. In commercial sector, this technology generally focuses on solving complex signal processing or pattern recognition problems. Examples of significant commercial applications since 2000 include handwriting recognition for check processing, speech-to-text transcription, oil-exploration data analysis and facial recognition.

**Objective**

1. To implement Neural Networks for realization of AND, OR gates.

**Program Code**

from math import exp

def ANDGateCalculation(x1, x2):

stringLiteral = f"{x1} AND {x2} = "

weights = [-2, 1, 1]

Z = weights[0] + x1 \* weights[1] + x2 \* weights[2]

sigmod\_val = 1/(1 + exp(-Z))

if sigmod\_val >= 0.5: print(stringLiteral + "1")

else: print(stringLiteral + "0”

def ORGateCalculation(x1, x2):

stringLiteral = f"{x1} OR {x2} = "

weights = [-1, 2, 2]

Z = weights[0] + x1 \* weights[1] + x2 \* weights[2]

sigmod\_val = 1/(1 + exp(-Z))

if sigmod\_val >= 0.5: print(stringLiteral + "1")

else: print(stringLiteral + "0")

for x1 in range(2):

for x2 in range(2):

ORGateCalculation(x1,x2)

ANDGateCalculation(x1,x2)

**Output**

****

**Conclusion**

Hence, we simulated AND and OR gates using the knowledge of Neural Networks.

**Lab 7: Write program for implementing Backpropagation Learning.**

**Introduction**

Backpropagation algorithm is the most fundamental building block in a neural network. This algorithm is used to effectively train a neural network through a method called chain rule. After each forward pass through a network, backpropagation performs a backward pass while adjusting the model’s parameters. Backpropagation is the essence of neural net training. It is the practice of fine tuning weights of a neural net based on the error rate obtained un the previous epoch.

**Objective**

1. To implement Backpropagation learning by creating a Sudoku solver.

**Program Code**

N = 9

def checkSafe(grid, row, col, num):

for x in range(9):

if grid[row][x] == num:

return False

for x in range(9):

if grid[x][col] == num:

return False

startRow = row - row % 3

startCol = col - col % 3

for i in range(3):

for j in range(3):

1

Artificial Intelligence Lab 8

if grid[i + startRow][j + startCol] == num:

return False

return True

def solveSuduko(grid, row, col):

if (row == N - 1 and col == N):

return True

if col == N:

row += 1

col = 0

if grid[row][col] > 0:

return solveSuduko(grid, row, col + 1)

for num in range(1, N + 1, 1):

if checkSafe(grid, row, col, num):

grid[row][col] = num

if solveSuduko(grid, row, col + 1):

return True

grid[row][col] = 0

return False

sudokuProblem = [

[0, 9, 0, 5, 6, 1, 0, 0, 7],

[0, 3, 0, 0, 0, 8, 0, 0, 1],

[0, 0, 8, 0, 4, 0, 0, 5, 9],

[7, 0, 0, 2, 8, 0, 1, 0, 0],

[9, 0, 6, 0, 0, 0, 8, 7, 0],

[0, 0, 0, 0, 0, 6, 5, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 3, 0, 0, 2, 0, 0, 0],

[0, 4, 9, 7, 1, 0, 0, 0, 5]

]

print("The solution of above sudoku is:\n")

if (solveSuduko(sudokuProblem, 0, 0)):

for i in range(N):

for j in range(N):

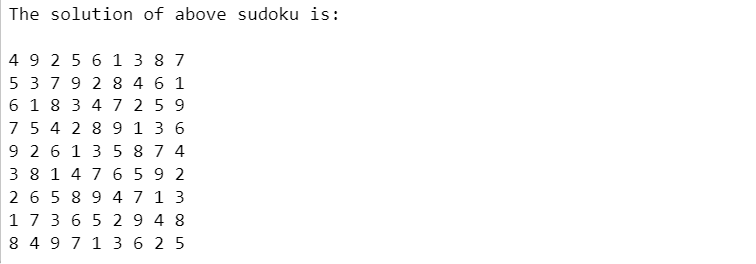
print(sudokuProblem[i][j], end = " ")

print()

else:

print("Solution failed")

**Output**

****

**Conclusion**

Hence, we implemented Backpropagation learning by creating a Sudoku solver using the python programming language.

**Lab 8: Write program for implementing expert systems for Covid prediction.**

**Introduction**

Expert system is a computer program that uses AI methods to solve problems within a specialized domain that ordinarily requires human expertise. In order to accomplish feats of apparent intelligence, an expert system relies on two components: a knowledge base and an inference engine.

**Objective**

1. To implement expert systems for Covid prediction.

**Program Code**

print("====================================")

print(" COVID CHECKER ")

print("====================================")

print("Answer the following questions:\n")

cough = (input("Do you have a Cough? ")).lower() == "y"

shortnessOfBreath = (input("Do you have shorness of breath? ")).lower() == "y" fever = (input("Do you have a fever? ")).lower() == "y"

lossOfTaste = (input("Have you lost your taste or smell? ")).lower() == "y" symtomFactorWeight = {

"fever": [3, -2],

"cough": [2, -1],

"breath": [3, -2],

"taste": [2, 0],

}

valuePredictor = 0

if fever: valuePredictor += symtomFactorWeight["fever"][0]

else: valuePredictor += symtomFactorWeight["fever"][1]

if cough: valuePredictor += symtomFactorWeight["cough"][0]

else: valuePredictor += symtomFactorWeight["cough"][1]

if shortnessOfBreath: valuePredictor += symtomFactorWeight["breath"][0] else: valuePredictor += symtomFactorWeight["breath"][1]

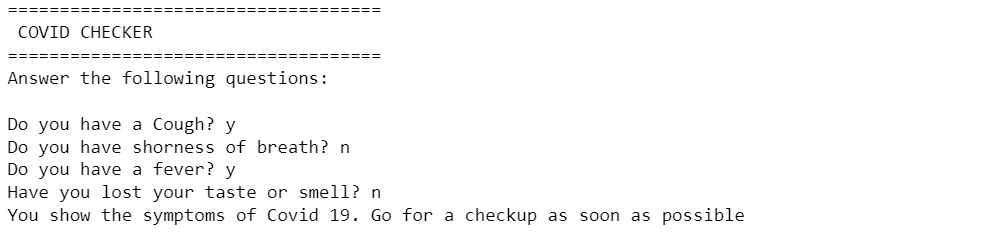
if lossOfTaste: valuePredictor += symtomFactorWeight["taste"][0]

else: valuePredictor += symtomFactorWeight["taste"][1]

if valuePredictor > 0:

print("You show the symptoms of Covid 19. Go for a checkup as soon as possible") else: print("You do not show the symptoms of Covid 19")

**Output**

****

**Conclusion**

Hence, we implemented expert systems for Covid prediction using the python programming language.