Certificate

Name: Aditi Indoori

Class: IT-1

Roll No: 160118737001

Exam No:

Institution Chaitanya Bharathi Institute of technology

This is certified to be the bonafide work of the student in the Artificial Intelligence *Laboratory during the academic year 20 / 20 .*

No. of practicals certified _____ *out of* _____ *in the subject of* _____

.....

Teacher In-charge

.....

Examiner's Signature

.....

Principal

Date: .....

Institution Rubber Stamp

(N.B: The candidate is expected to retain his/her journal till he/she passes in the subject.)

Index

Index

Date _____

Expt. No. 1 Page No. 1

1) Write a program to conduct uninformed search.

a) Breadth First Search.

```
from collections import defaultdict.
class Graph:
    def __init__ (self):
        self.graph = defaultdict(list)

    def addEdge (self, u, v):
        self.graph[u].append(v)

    def BFS (self, v):
        print("The BFS traversal of the given
                graph is as follows:")
        visited = [false] * (max(self.graph) + 1)
        queue = []
        queue.append(v)
        visited[v] = True.

        while queue:
            v = queue.pop(0)
            print(v, end = " ")
            for neighbour in self.graph[v]:
                if visited[neighbour] == false:
                    queue.append(neighbour)
                    visited[neighbour] = True
```

Teacher's Signature _____

Date _____

Expt. No. 1 Page No. 2

```
MyGraph = Graph()
MyGraph.addEdge(0, 1)
MyGraph.addEdge(0, 2)
MyGraph.addEdge(1, 2)
MyGraph.addEdge(2, 0)
MyGraph.addEdge(2, 3)
MyGraph.addEdge(3, 3)

MyGraph.BFS(2)
```

Teacher's Signature _____

Output

The BFS traversal of the given graph is as follows;

2 0 3 1

(b) Depth first Search

```
from collections import dfault dict.

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self, u, v):
        self.graph[u].append(v)

    def DfsUtil(self, v, visited):
        visited.add(v)
        print(v, end=" ")

        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DfsUtil(neighbour, visited)

    def DFS(self, v):
        visited=set()
        print("The DFS traversal is as follows:")
        self.DFSUtil(v, visited)

MyGraph = Graph()
Mygraph.addEdge(0,1)
Mygraph.addEdge(0,2)
Mygraph.addEdge(1,2)
Mygraph.addEdge(2,0)
Mygraph.addEdge(2,3)
Mygraph.addEdge(3,3)
Mygraph.DFS(2)
```

Teacher's Signature _____

Output

The DFS traversal is as follows :

2 0 1 3

(2) Write a program to conduct informed Search.

(a) Best first Search

```
from queue import PriorityQueue
V=14
graph = [ [] for i in range(V) ]

# gives o/p path having lowest search
def best_first_search (source, target, n) :
    visited = [0] * n.
    visited[0] - 1

    pq = PriorityQueue()
    pq.put ((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print (u, end = " ")
        if u == target :
            break.
        for v,c in graph[u]:
            if visited[v] == 0:
                visited[v] = 1
                pq.put((c, v))
    print()

# adding edges to graph
def addedge (x, y, cost):
    graph[x]. append ((y, cost))
    graph[y] . append ((x, cost))
```

addedge (0, 1, 3)
addedge (0, 2, 6)
add edge (0, 3, 5)
add edge (1, 4, 9)
add edge (1, 5, 8)
addedge (2, 6, 12)
add edge (2, 7, 14)
add edge (3, 8, 7)
addedge (8, 9, 5)
add edge (8, 10, 6)
add edge (9, 11, 1)
add edge (9, 12, 10)
add edge (9, 13, 2)

source = 0
target = 9
best_first_search(source, target, V)

output

0 1 3 2 8 9

(b) A* Algorithm

```
class Node():
    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position
        self.g = 0
        self.h = 0
        self.f = 0
    def __eq__(self, other):
        return self.position == other.position

def astar(maze, start, end):
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0
    open_list = []
    closed_list = []
    open_list.append(start_node)
    while len(open_list) > 0:
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index
        open_list.pop(current_index)
        closed_list.append(current_node)
```

Teacher's Signature _____

```
if current_node == end_node :
    path = []
    current = current_node.
    while current is not None :
        path.append (current.position)
        current = current.parent -
    return path [:: -1 ]
children = []
for new_position in [(0,-1), (0,1), (-1,0), (1,0),
                     (-1,-1), (1,-1), (-1,1), (1,1)]:
    node_position = (current_node.position[0] +
                     new_position[0],
                     current_node.position[1] +
                     new_position[1])
    if node_position[0] > (len(maze) - 1) or
    node_position[0] < 0 or node_position[1] >
    (len(maze[len(maze) -1]) -1) or node_position[1] <0:
        continue.
    if maze[node_position[0]][node_position[1]] != 0:
        continue.
    new_node = Node (current_node, node_position)
    children.append (new_node)

for child in children:
    for closed_child in closed_list:
        if child == closed_child :
            continue
    child.g = current_node.g +1
```

```
child.h = (child.position[0] - end_node.position[0]) ** 2)
        + (child.position[1] - end_node.position[1]) ** 2)
child.f = child.g + child.h
for open_node in open_list:
    if child == open_node and child.g > open_node.g:
        continue
open_list.append(child)

def main():
    maze = [[0, 0, 0, 0, 1, 0],
            [0, 0, 0, 0, 1, 0],
            [0, 0, 0, 0, 1, 0],
            [0, 0, 0, 0, 1, 0],
            [0, 0, 0, 0, 1, 0],
            [0, 0, 0, 0, 0, 0]]

    graph = [[0, 1, 0, 0, 0, 0],
             [1, 0, 1, 0, 1, 0],
             [0, 1, 0, 0, 0, 1],
             [0, 0, 0, 0, 1, 0],
             [0, 1, 0, 1, 0, 0],
             [0, 0, 1, 0, 0, 0]]

    start = (0,0)
    end = (5,5)
    end1 = (5,5)
    path = astar(maze, start, end)
    print(path)
    path1 = astar(graph, start, end1)
    print(path1)
if __name__ == '__main__':
    main()
```

Teacher's Signature _____

output:

[(0,0) , (1,1) , (2,2) , (3,3) , (4,3) , (5,4) , (5,5)]

[(0,0) , (1,1) , (2,2) , (3,3) , (4,4) , (5,5)]

(c) Hill climbing Algorithm

```
import random

def randomSolution(tsp):
    cities = list(range(len(tsp)))
    solution = []

    for i in range(len(tsp)):
        randomCity = cities[random.randint(0, len(cities)-1)]
        solution.append(randomCity)
        cities.remove(randomCity)

    return solution.

def routeLength(tsp, solution):
    routeLength = 0
    for i in range(len(solution)):
        routeLength += tsp[solution[i-1]][solution[i]]
    return routeLength.

def getNeighbours(solution):
    neighbours = []
    for i in range(len(solution)):
        for j in range(i+1, len(solution)):
            neighbour = solution.copy()
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
```

```
                neighbours.append(neighbour)
        return neighbours.

def getBestNeighbour(tsp, neighbours):
    bestRouteLength = routeLength(tsp, neighbours[0])
    bestNeighbour = neighbours[0]
    for neighbour in neighbours:
        currentRouteLength = routeLength(tsp, neighbour)
        if currentRouteLength < bestRouteLength:
            bestRouteLength = currentRouteLength.
            bestNeighbour = neighbour.
    return bestNeighbour, bestRouteLength.

def hillClimbing(tsp):
    currentSolution = randomSolution(tsp)
    currentRouteLength = routeLength(tsp, currentSolution)
    neighbours = getNeighbours(currentSolution)
    bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp,
                                                neighbours)
    while bestNeighbourRouteLength < currentRouteLength:
        currentSolution = bestNeighbour.
        currentRouteLength = bestNeighbourRouteLength.
        neighbours = getNeighbours(currentSolution)
        bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp,
                                                neighbours)
    return currentSolution, currentRouteLength.
```

```
def main():
    tsp = [
            [0, 400, 500, 300],
            [400, 0, 300, 500],
            [500, 300, 0, 400],
            [300, 500, 400, 0]
        ]
    print(hillClimbing(tsp))

if __name__ == "__main__":
    main()
```

Output

([1, 2, 3, 0], 1400)

Date _____

Expt. No. 3 Page No. 12

(3) Implementation of Game Search.

(a) MiniMax Algorithm

```
import math

def minimax (curDepth, nodeIndex, maxTurn, scores,
             targetDepth):
    if (curDepth == targetDepth):
        return scores[nodeIndex].

    if (maxTurn):
        return max(minimax(curDepth+1, nodeIndex*2,
                           False, scores, targetDepth),
                   minimax(curDepth+1, nodeIndex*2+1,
                           False, scores, targetDepth))

    else:
        return min(minimax(curDepth+1, nodeIndex*2,
                           True, scores, targetDepth),
                   minimax(curDepth+1, nodeIndex*2+1,
                           True, scores, targetDepth))

scores = [3, 5, 2, 9, 12, 5, 23, 23]
treeDepth = math.log(len(scores), 2)
print("The optimal value is :", end="")
print(minimax(0, 0, True, scores, treeDepth))
```

Teacher's Signature _____

output

The optimal value is: 12.

(b) Alpha Beta Pruning:

```
MAX, MIN = 1000, -1000

def minimax (depth, nodeIndex, maximizingPlayer, values,
             alpha, beta):
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:
        best = MIN

        for i in range (0, 2):
            val = minimax (depth+1, nodeIndex * 2 + i,
                           False, values, alpha, beta)
            best = max (best, val)
            alpha = max (alpha, best)

            if beta <= alpha:
                break.

        return best

    else:
        best = MAX
        for i in range (0, 2):
            val = minimax (depth + 1, nodeIndex * 2 + i,
                           True, values, alpha, beta)
            best = min (best, val)
            beta = min (beta, best)
```

Teacher's Signature _____

```
        if beta <= alpha:
            break.

    return best.

if __name__ == "__main__":
    values = [3, 5, 6, 9, 1, 2, 0, -1]
    print("The optimal value is:", minimax(0,0, True, values, MIN, MAX))
```

The optimal value is : 5

(4) Prolog

(a) prime.pl

```
is_prime(2). is_prime(3). is_prime(P) :- integer(P),
P > 3,
P mod 2 =\= 0,
\+ has_factor(P,3).

% has_factor(N,L) :- N has an odd factor F >= L.
% (integer, integer) (+,+)

has_factor(N,L) :- N mod L =:= 0.
```

(b) factorial.pl

```
factorial(0,1).
factorial(N,F) :-
N > 0,
N1 is N-1,
factorial(N1, F1), F is N * F1.
```

Teacher's Signature

(c) adjacent.pl

```
adjacent(1,2).   adjacent(2,1).
adjacent(1,3).   adjacent(3,1).
adjacent(1,4).   adjacent(4,1).
adjacent(1,5).   adjacent(5,1).
adjacent(2,3).   adjacent(3,2).
adjacent(2,4).   adjacent(4,2).
adjacent(3,4).   adjacent(4,3).
adjacent(4,5).   adjacent(5,4).
```

(d) family.pl.

```
% This is the prolog version of the family example

child(john, sue).    child(john, sam).
child(jane, sue).    child(jane, sam).
child(sue, george).  child(sue, gina).

male(john).   male(sam).   male(george).
female(sue).  female(jane). female(jane).

parent(Y,X) :- child(X,Y).
father(Y,X) :- child(X,Y), male(Y).
opp_sex(X,Y) :- male(X), female(Y).
opp-sex(Y,X) :- male(X), female(Y).
grand-father(X,Z) :- father(X,Y), parent(Y,Z).
```

Teacher's Signature _____

(5) Bayesian Network.

```
→ from pgmpy.models import BayesianModel.
→ from pgmpy.inference import VariableElimination

alarm_model = BayesianModel([('Burglary', 'Alarm'),
                             ('Earthquake', 'Alarm'),
                             ('Alarm', 'JohnCalls'),
                             ('Alarm', 'MaryCalls')])

#Defining the parameters using CPT
→ from pgmpy.factors.discrete import TabularCPD

cpd_burglary = TabularCPD(variable='Burglary', variable_card=2,
                          values=[[.999], [0.001]])

cpd_earthquake = TabularCPD(variable='Earthquake', variable_card=2,
                            values=[[.998], [0.002]])

cpd_alarm = TabularCPD(variable='Alarm', variable_card=2,
                       values=[[0.999, 0.71, 0.06, 0.05],
                               [0.001, 0.29, 0.94, 0.95]],
                       evidence=['Burglary', 'Earthquake'],
                       evidence_card=[2, 2])

cpd_johncalls = TabularCPD(variable='JohnCalls', variable_card=2,
                           values=[[0.95, 0.1], [0.05, 0.9]],
                           evidence=['Alarm'], evidence_card=[2])
```

```
cpd-marycalls = TabularCPD (variable = 'Mary Calls', variable_card=2,
                            values = [[0.1, 0.7], [0.9, 0.3]],
                            evidence = ['Alarm'], evidence_card = [2])

# Associating the parameters with the model structure.
alarm_model.add_cpds(cpd-burglary, cpd-earthquake,
                     cpd-alarm, cpd-john calls,
                     cpd-marycalls)
```

→ alarm_model.check_model()

(*) True.

→ alarm_model.nodes()

(*) NodeView (('Burglary', 'Alarm', 'Earthquake', 'JohnCalls', 'MaryCalls'))

→ alarm_model.edges()

(*) outEdgeView ([('Burglary', 'Alarm'), ('Alarm', 'John Calls'), ('Alarm', 'MaryCalls'), ('Earthquake', 'Alarm')])

→ alarm_model.local_independencies('Burglary')

(*) (Burglary ⊥ Earthquake).

Date

Expt. No. 6 Page No. 19

(6) Bayesian Model

→
```
from pybbn.graph.dag import Bbn
from pybbn.graph.edge import Edge, EdgeType
from pybbn.graph.jointree import EvidenceBuilder.
from pybbn.graph.node import BbnNode.
from pybbn.graph.variable import Variable
from pybbn.pptc.inferencecontroller import InferenceController.
```

→
```
Season = BbnNode(Variable(0, 'season', ['winter', 'summer']),
                 [0.5, 0.5])
atmos_pres = BbnNode(Variable(1, 'atmos-press', ['high', 'low']),
                     [0.5, 0.5])
allergies = BbnNode(Variable(2, 'allergies', ['allergic',
                    'non-allergic']),
                    [0.7, 0.3, 0.2, 0.8])
rain = BbnNode(Variable(3, 'rain', ['rainy', 'sunny']),
               [0.9, 0.1, 0.7, 0.3, 0.3, 0.7, 0.1, 0.9])
grass = BbnNode(Variable(4, 'grass', ['grass', 'no-grass']),
                [0.8, 0.2, 0.3, 0.7])
umbrellas = BbnNode(Variable(5, 'umbrellas', ['on', 'off']),
                    [0.99, 0.01, 0.80, 0.20, 0.20, 0.80, 0.01,
                     0.99])
dog-bark = BbnNode(Variable(6, 'dog-bark', ['bark', 'not-bark']),
                   [0.8, 0.2, 0.1, 0.9])
cat-mood = BbnNode(Variable(7, 'cat_mood', ['good', 'bad']),
                   [0.05, 0.95, 0.95, 0.05])
cat_hide = BbnNode(Variable(8, 'cat_hide', ['hide', 'show']),
                   [0.20, 0.80, 0.95, 0.05, 0.95, 0.05, 0.70, 0.30])
```

Teacher's Signature

```
bbn = Bbn() \
    • add_node (Season) \
    • add_node (atmos_pres) \
    • add_node (allergies) \
    • add_node (rain) \
    • add_node (grass) \
    • add_node (umbrellas) \
    • add_node (dog_bark) \
    • add_node (cat_mood) \
    • add_node (cat_hide) \
    • add_edge (Edge (season, allergies, EdgeType.DIRECTED)) \
    • add_edge (Edge (season, umbrellas, EdgeType.DIRECTED)) \
    • add_edge (Edge (season, rain, EdgeType.DIRECTED)) \
    • add_edge (Edge (atmos_pres, rain, EdgeType.DIRECTED)) \
    • add_edge (Edge (rain, grass, EdgeType.DIRECTED)) \
    • add_edge (Edge (rain, umbrellas, EdgeType.DIRECTED)) \
    • add_edge (Edge (rain, dog_bark, EdgeType.DIRECTED)) \
    • add_edge (Edge (rain, cat_mood, EdgeType.DIRECTED)) \
    • add_edge (Edge (dog_bark, cat_hide, EdgeType.DIRECTED)) \
    • add_edge (Edge (cat_mood, cat_hide, EdgeType.DIRECTED)) \

→ join_tree = InferenceController.apply (bbn)
#insert an observation reference.
ev = EvidenceBuilder() \
    • with_node(join_tree.get_bbn_node_by_name('season')) \
    • with_evidence('winter', 1.0) \
    • build ()

join_tree.set_observation (ev)
```

Teacher's Signature _____

```
#print the marginal probabilities.
for node in join_tree.get_bbn_nodes():
        potential = join_tree.get_bbn_potential(node)
        print(node)
        print(potential)
        print('------>')
```

(*)

```
0|season|winter, summer.
0 = winter|1.00000
0 = summer|0.00000.
---------->
2|allergies|allergic, non-allergic
2 = allergic|0.70000
2 = non-allergic|0.30000.
---------->
3|rain|rainy, sunny.
3 = rainy|0.80000.
3 = sunny|0.20000
---------->
4|grass|grass, no_grass
4 = grass|0.70000
4 = no_grass|0.30000
---------->
1|atmos_pres|high, low
1 = high|0.50000
1 = low|0.50000.
---------->
5|umbrellas|on, off.
5 = on|0.95200
5 = off|0.04800
```

Teacher's Signature _____

6 | dog_bark | bark, not_bark.
6 = bark | 0.66000
6 = not_bark | 0.34000.
-------------->

7 | cat_mood | good, bad.
7 = good | 0.23000.
7 = bad | 0.77000
-------------->

8 | cat_hide | hide, show.
8 = hide | 0.87150.
8 = show | 0.12850.

```
→ join_tree = InferenceController.apply(bbn)
  # insert an observation evidence.
  ev = EvidenceBuilder() \
       .with_node(join_tree.get_bbn_node_by_name('season')) \
       .with_evidence('winter', 1.0) \
       .build()
  ev2 = EvidenceBuilder() \
       .with_node(join_tree.get_bbn_node_by_name('dog_bark')) \
       .with_evidence('not_bark', 1.0) \
       .build()

join_tree.set_observation(ev)
join_tree.set_observation(ev2)
```

```
# print the marginal probabilities
for node in join_tree.get_bbn_nodes():
    potential = join_tree.get_bbn_potential(node)
    print(node)
    print(potential)
    print('------------>')
```

(*)
0|season|winter, summer.
0 = winter |1.00000
0 = summer |0.00000
---------->

2|allergies|allergic, non-allergic.
2 = allergic |0.70000.
2 = non-allergic | 0.30000
---------->

3|rain|rainy, sunny.
3 = rainy | 0.47059
3 = sunny |0.52941
---------->

4|grass|grass, no_grass
4 = grass |0.53529
4 = no-grass |0.46471
---------->

1|atmos-pres|high, low.
1 = high |0.39706
1 = low |0.60294
---------->

5|umbrellas|on, off.
5 = on |0.88941
5 = off |0.11059
---------->

Teacher's Signature _____

Expt. No. 6 Page No. 24

6 | dog_bark | bark, not_bark.
6 = bark | 0.00000
6 = not_bark | 1.00000
------------>
7 | cat_mood | good, bad
7 = good | 0.52647.
7 = bad | 0.47353
------------>
8 | cat_hide | hide, show
8 = hide | 0.83162
8 = show | 0.16838.
------------>

Teacher's Signature

Date

Expt. No. 7

Page No. 25

(7) Supervised Learning.

→
```
print (--doc--)
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split
```

→
```
digits = datasets.load_digits()
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10,3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Training: %i' % label)
```

(*)

Training: 0	Training: 1	Training: 2	Training: 3
0	1	2	3

→
```
#flatten the images
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

#create a classifier: a support vector classifier.
clf = svm.SVC(gamma=0.001)

# split data into 50% train and 50% test subsets.
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False)
```

Teacher's Signature

Date

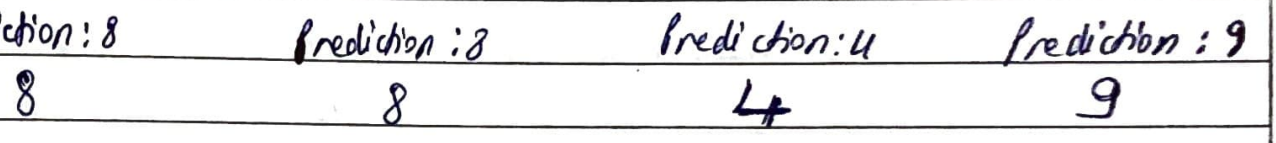
Expt. No. 7

Page No. 26

```
# Learn the digits on the train subset.
clf.fit(X_train, y_train)
# predict the value of the digit on the test subset
Predicted = clf.predict(X_test)
```

→
```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10,3))
for ax, image, prediction in zip(axes, X_test, predicted):
    ax.set_axis_off()
    image = image.reshape(8,8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title(f'Prediction: {prediction}')
```

(*)



→
```
print(f"classification report for classifier {clf}:\n"
      f"{metrics.classification_report(y_test, predicted)}\n")
```

(*) Classification report for classifier SVC(gamma=0.001):

	precision	recall	f1-score	support
0	1.00	0.99	0.99	88
1	0.99	0.97	0.98	91
2	0.99	0.99	0.99	86
3	0.98	0.87	0.92	91
4	0.99	0.96	0.97	92
5	0.95	0.97	0.96	91
6	0.99	0.99	0.99	91
7	0.96	0.99	0.97	89
8	0.94	1.00	0.97	88
9	0.93	0.98	0.95	92

Teacher's Signature

Date

Expt. No. 7 Page No. 27

Accuracy			0.97	899
macro avg	0.97	0.97	0.97	899
weighted avg	0.97	0.97	0.97	899.

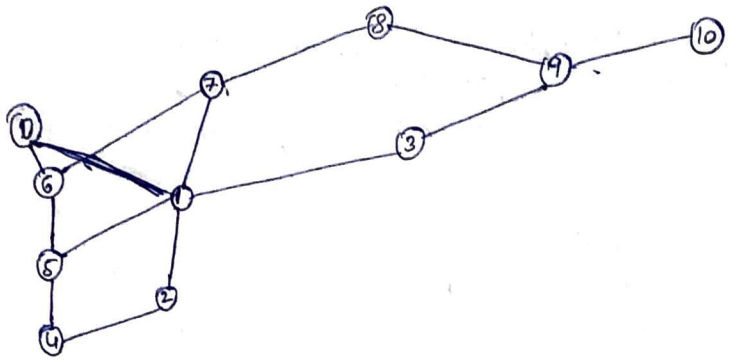
→ disp = metrics.plot_confusion_matrix(clf, x_test, y_test)
disp.figure_.suptitle("confusion matrix")
print(f"confusion matrix: \n {disp.confusion_matrix}")
plt.show()

(*) Confusion matrix:

```
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```

Teacher's Signature

2 (*)



3 (*)

(0,1)
(1,5)
(5,6)
(5,4)
(1,2)
(1,3)
(9,10)
(2,4)
(0,6)
(6,7)
(8,9)
(7,8)
(1,7)
(3,9)

Expt. No. 8 Page No. 28 Date

(8) Reinforcement Learning.

1 →
```
import numpy as np
import pylab as pl
import networkx as nx
```

2 →
```
edges = [(0,1), (1,5), (5,6), (5,4), (1,2), (1,3),
         (9,10), (2,4), (0,6), (6,7), (8,9),
         (7,8), (1,7), (3,9)]

goal = 10
G = nx.Graph()
G.add_edges_from(edges)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edges(G,pos)
nx.draw_networkx_labels(G, pos)
pl.show()
```

3 →
```
MATRIX_SIZE = 11
M = np.matrix(np.ones(shape = (MATRIX_SIZE, MATRIX_SIZE)))
M *= -1

for point in edges:
    print(point)
    if point[1] == goal:
        M[point] = 100
    else:
        M[point] = 0
```

Teacher's Signature

```
[[ -1.   0.  -1.  -1.  -1.  -1.   0.  -1.  -1.  -1.  -1.]
 [  0.  -1.   0.   0.  -1.   0.  -1.   0.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.   0.  -1.  -1.  -1.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.  -1.  -1.  -1.  -1.  -1.   0.  -1.]
 [ -1.  -1.   0.  -1.  -1.   0.  -1.  -1.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.   0.  -1.   0.  -1.  -1.  -1.  -1.]
 [  0.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.  -1.]
 [ -1.  -1.  -1.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.]
 [ -1.  -1.  -1.   0.  -1.  -1.  -1.  -1.   0.  -1. 100.]
 [ -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.   0. 100.]]
```

4 (*) 0

Expt. No. 9 Date

Page No. 28

```
        if point[o] == goal :
            M[point[::-1]] = 100
        else :
            M[point[::-1]] = 0

    M[goal, goal] = 100
    print(M)

4→  Q = np.matrix(np.zeros([MATRIX_SIZE, MATRIX_SIZE]))

    gamma = 0.75
    initial_state = 1

    def available_actions(state):
        current_state_row = M[state, ]
        available_action = np.where(current_state_row>=0)[1]
        return available_action

    available_action = available_actions(initial_state)

    def sample_next_action(available_actions_range):
        next_action = int(np.random.choice(available_action,1))
        return next_action

    action = sample_next_action(available_action)
```

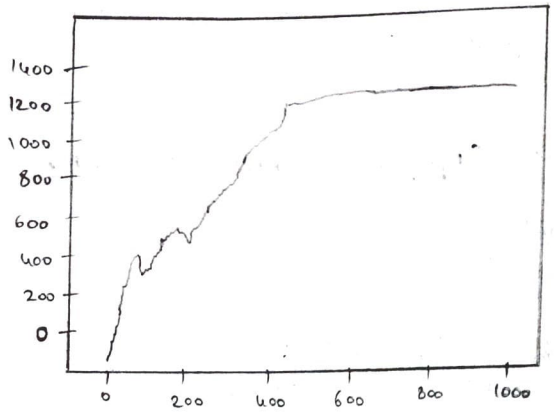
Teacher's Signature

Date

Expt. No. 8 Page No. 30

(*) Most efficient path :

[0, 1, 3, 9, 10]




```
def update ( current_state, action, gamma):
    max_index = np. where (Q[action, ] = np. max(Q[action, ])
                                                    [1]
    if max_index. shape [0] > 1:
        max_index = int (np. random. choice (max_index,
                                            size = 1))
    else :
        max_index = int (max_index)
    max_value = Q [action, max_index]
    Q [current_state, action ] = M[current_state, action]
                                + gamma * max_value
    if (np. max(Q) > 0) :
        return (np. sum (Q/ np. max(Q)* 100))
    else :
        return (0)

update (initial_state, action, gamma)
```

5 →

```
Scores = []
for i in range (1000):
    current_state = np.random. randint (0, int (Q.shape[0]))
    available_action = available_actions (current_state)
    action = sample_next_action (available_action)
    score = update (current_state, action, gamma)
    scores. append (score)

Current_state = 0
steps = [current_state]
```

Teacher's Signature

```
while current_state != 10 :
    next_step_index = np.where(Q[current_state,]
                      == np.max(Q[current_state,])[1]
    if next_step_index.shape[0] > 1 :
        next_step_index = int(np.random.choice(next_step_
                                               index,
                                               size = 1))
    else :
        next_step_index = int(next_step_index)
    steps.append(next_step_index)
    current_state = next_step_index

print("Most efficient path :")
print(steps)

pl.plot(scores)
pl.xlabel('No. of iterations')
pl.ylabel('Reward gained')
pl.show()
```

Teacher's Signature

Date _____

Expt. No. 9 Page No. 32

(9) Artificial neural network with Back Propagation.

→

```
import numpy as np.

def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))

def sigmoid_prime(x):
    return sigmoid(x) * (1.0 - sigmoid(x))

def tanh(x):
    return np.tanh(x)

def tanh_prime(x):
    return 1.0 - x**2.

class NeuralNetwork:
    def __init__(self, layers, activation = 'tanh'):
        if activation == 'sigmoid':
            self.activation = sigmoid.
            self.activation_prime = sigmoid_prime.
        elif activation == 'tanh':
            self.activation = tanh.
            self.activation_prime = tanh_prime.

        self.weights = []
        for i in range(1, len(layers) - 1):
            r = 2*np.random.random((layers[i-1]+1,
                                    layers[i]+1)) -1
            self.weights.append(r)
```

Teacher's Signature _____

Date

Expt. No. 9 Page No. 33

```
r = 2*np.random.random((layers[i]+1,
                        layers[i+1])) -1
self.weights.append(r)

def fit(self, X, y, learning_rate=0.5, epochs=100000):
    ones = np.atleast_2d(np.ones(X.shape[0]))
    X = np.concatenate((ones.T, X), axis=1)
    for k in range(epochs):
        if k % 10000 == 0:
            print('epochs:', k)
        i = np.random.randint(X.shape[0])
        a = [X[i]]

        for l in range(len(self.weights)):
            dot_value = np.dot(a[l], self.weights[l])
            activation = self.activation(dot_value)
            a.append(activation)

        error = y[i] - a[-1]
        deltas = [error * self.activation_prime(a[-1])]

        for l in range(len(a)-2, 0, -1):
            deltas.append(deltas[-1].dot(self.weights[l].T)*
                self.activation_prime(a[l]))

        deltas.reverse()
        for i in range(len(self.weights)):
            layer = np.atleast_2d(a[i])
            delta = np.atleast_2d(deltas[i])
            self.weights[i] += learning_rate * layer.T.dot(delta)
```

Teacher's Signature

```
def predict(self, x):
    a = np.concatenate((np.ones(1).T, np.array(x)))
    for l in range(0, len(self.weights)):
        a = self.activation(np.dot(a, self.weights[l]))
    return a

if __name__ == '__main__':
    nn = NeuralNetwork([2,2,1])
    x = np.array([[0,0],
                  [0,1],
                  [1,0],
                  [1,1]])
    y = np.array([0,1,1,0])
    nn.fit(x,y)
    for e in x:
        print(e, nn.predict(e))
```

(*)

epochs : 0
epochs : 10000
epochs : 20000
epochs : 30000
epochs : 40000
epochs : 50000
epochs : 60000
epochs : 70000
epochs : 80000
epochs : 90000

[0, 0] [-0.00533844]
[0 1] [0.9977984]
[1 0] [0.99770688]
[1 1] [0.00568913]

Date _____

Expt. No. 10 Page No. 35

(10) CHATBOT

→ pip install chatterbot
→ pip install --upgrade chatterbot-corpus
→ pip install --upgrade chatterbot.

→ from chatterbot import chatBot
from chatterbot.trainers import ListTrainer.
my_bot = chatBot(name = 'PyBot', read_only = True,
logic_adapters = ['chatterbot.logic.
MathematicalEvaluation',
'chatterbot.logic.BestMatch'])

→ small_talk = ['hi there!',
'hi!',
'how do you do?',
'how are you?',
'i'm cool.',
'fine, you?',
'always cool.',
'i'm ok',
'glad to hear that.',
'i'm fine',
'glad to hear that',
'i feel awesome',
'excellent, glad to hear that',
'not so good',
'sorry to hear that',
'what's your name?',
'i'm pybot. ask me a math question, please.']

Teacher's Signature _____

```
math_talk_1 = ['pythagorean theorem',
               'a squared plus b squared equals c squared']
math_talk_2 = ['law of cosines',
               'c**2 = a**2 + b**2 - 2*a*b*cos(gamma)']

→ list_trainer = ListTrainer(my_bot)
for item in (small_talk, math_talk_1, math_talk_2):
    list_trainer.train(item)
```

(*) List Trainer: [##########] 100%
List Trainer: [##########] 100%
List Trainer: [##########] 100%

```
→ print(my_bot.get_response("glad to hear that"))
```
(*) i'm fine

```
→ print(my_bot.get_response("i feel awesome today"))
```
(*) excellent, glad to hear that.

```
→ print(my_bot.get_response("what's your name?"))
```
(*) i'm pybot. ask me a math question, please.

```
→ print(my_bot.get_response("pythagorean theorem"))
```
(*) a squared plus b squared equals c squared.

Expt. No. 10 Date

Page No. 37

→ from chatterbot.trainers import ChatterBotCorpusTrainer
corpus_trainer = ChatterBotCorpusTrainer(my_bot)
corpus_trainer.train('chatterbot.corpus.english')

(+) Training ai.yml: [########] 100%
Training bot_profile.yml: [
Training computers.yml:
Training conversations.yml:
Training emotion.yml:
Training food.yml:
Training gossip.yml:
Training greetings.yml:
Training health.yml:
Training history.yml:
Training humor.yml:
Training literature.yml:
Training money.yml:
Training movies.yml:
Training politics.yml:
Training psychology.yml:
Training science.yml:
Training sports.yml
Training trivia.yml.

→ print(my_bot.get_response("movies"))
(*) what kind of movies do you like?

Teacher's Signature

Date

Expt. No. 10 Page No. 38

→
```
from chatterbot import chatBot
from chatterbot.trainers import ListTrainer

chatbot = chatBot('CBIT')
trainer = ListTrainer(chatbot)
trainer.trainer([ 'hi, can I help you find a course',
                  'sure I'd love to find you a course',
                  'your course has been selected' ])

response = chatbot.get_response("I want a course")
print(response)
```

(*) List trainer : [# # # # # # # # # #] 100%
Sure I'd love to find you a course

Teacher's Signature

Date

Expt. No. 11

Page No. 39

(11) Operations on text data.

(a) Removal of punctuations in the given string

→
```
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
my_str = input("Enter a string: ")
no_punct = ""
for char in my_str:
    if char not in punctuations:
        no_punct = no_punct + char.
print(no_punct)
```

(*) Enter a string: Hello!!!, They said -- and went.
Hello They said and went.

(b) Generation of string Tokens

→ import secrets
→ secrets.randbelow(2)
secrets.randbelow(10)
secrets.randbelow(1)
serets.randbelow(9)

(*) 1

→ Secrets.randbits(1)
secrets.randbits(2)
Secrets.randbits(4)
secrets.randbits(8)

(*) 188

Teacher's Signature

Date _____

Expt. No. 11

Page No. 40

→ colour = ['red', 'blue', 'green', 'purple', 'yellow']
secrets.choice(colour)

(*) 'blue'

→ secrets.token_bytes(8)

(*) b'\x80\x92\x05f3\x9c\x15'

→ b'\x16g\x8e\x83\x08\x62g\x17'

(*) b'\x16g\x8e\x83\x08\x62g\x17'

→ secrets.token_hex(16)

(*) '8a45d655331f550fc3d80d60e2dded82b'

→ secrets.token_urlsafe(16)

(*) 'uVyVSG1AGhQvP6muW46uw'

→ import string # alphanumeric 10 digit password.
import secrets
alphabet = string.ascii_letters + string.digits
password = ''.join(secrets.choice(alphabet) for i in range(10))
print(password)

(*) uAqidwPvwD

→ import string
import secrets
alphabet = string.hexdigits + string.punctuation
password = ''.join(secrets.choice(alphabet) for i in range(10))
print(password)

(*) ->{@?JL1"`

Teacher's Signature _____

Date _____

Expt. No. 4 Page No. 41

→
```
import string
import secrets
alphabet = string.ascii_letters + string.digits
while True:
    password = ''.join(secrets.choice(alphabet) for i in range(10))
    if (any(c.islower() for c in password) and any(c.isupper()
            for c in password) and any(c.digit() for c in password)):
        break
print(password)
```
(*) OHmNr 7hYzP

→
```
import string
import secrets
alphabet = string.ascii_letters + string.digits
while True:
    password = ''.join(secrets.choice(alphabet) for i in range(10))
    if (sum(c.isupper() for c in password) >= 2 and sum(c.isdigit()
            for c in password) >= 2):
        break
print(password)
```
(*) U5126dfBlp.

→
```
import secrets
animal = ['horse', 'elephant', 'monkey', 'donkey', 'goat', 'chicken',
          'duck', 'mouse']
fruit = ['apple', 'banana', 'peach', 'orange', 'papaya', 'watermelon',
         'durian']
electronic = ['computer', 'laptop', 'smartphone', 'battery', 'charger',
              'cable']
```

Teacher's Signature _____

```
vegetable = ['electronic 'lettuce', 'spinach', 'celery', 'cabbage',
            'turnip', 'cucumber', 'eggplant']
password = set()
while True:
    password.add(secrets.choice(word_list))
    if (len(password) >= 4):
        break.
print(" ".join(password))
```

(*) battery peach donkey smartphone

→ import secrets
```
url = 'https://mywebsite/reset?key=' + secrets.token_urlsafe()
print(url)
```

(*) https://mywebsite/reset?key=ia_icSmLAhrrs3osikoHQRE Jsunpbtkvy

Date _____

Expt. No. 12 Page No. 43

(12) Operations using nltk.

(a) Removal of stop words for a given passage from a text file.

→
```
import nltk
nltk.download('stopwords')
```

(*)
```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] unzipping corpora/stopwords.zip.
True
```

→
```
import nltk
nltk.download('punkt')
```

(*)
```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] unzipping tokenizers/punkt.zip.
True
```

→
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
set(stopwords.words('english'))

text = """He determined to drop his litigation with the monastry
& relinquish his claims to the wood-cutting &
fishery rights at once. He was the more ready
to do this because the rights had become
much less valuable, and he had indeed the
vaguest idea where the wood and river in question
were."""
```

Teacher's Signature _____

```
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(text)
filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print("\n\n original Sentence \n\n")
print(" ".join(word_tokens))
print("\n\n Filtered Sentence \n\n")
print(" ".join(filtered_sentence))
```

(*) original sentence

He determined to drop his litigation with the monastry, and relinquish his claims to the wood-cutting and fishery rights at once. He was the more ready to do this because the rights had become much less valuable, and he had indeed the vaguest idea where the wood and river in question were.

Filtered Sentence

He determined drop litigation monastry, relinquish claims wood-cutting fishery rights. He ready because rights become much less valuable, indeed vaguest idea. wood river question.

Teacher's Signature

Date

Expt. No. 12

Page No. 45

(b) Stemming for a given sentence

→
```
from nltk.stem import porterstemmer
st = porterStemmer()
text = ['where did he learn to dance like that?',
        'His eyes were dancing with humor.',
        'She shook her head and danced away',
        'Alex was an excellent dancer']

output = []
for sentence in text:
    output.append(" ".join([st.stem(i) for i in
                            sentence.split()]))

for item in output:
    print(item)

print("-" * 50)
print(st.stem('jumping'), st.stem('jumps'), st.stem('jumped'))
```

(*)
```
where did he learn to danc like that?
hi eye were danc with humor.
She shook her head and danc away
alex a wa an exel dancer.
```

```
jump jump jump
```

Teacher's Signature

Date

Expt. No. 12

Page No. 46

(C) POS tagging for a given sentence to classify text data.

→ import nltk

→ nltk.download('brown')
nltk.download('averaged_perceptron_tagger')

→ text = word_tokenize("And now for something completely different")
nltk.pos_tag(text)

(*) [('And', 'CC'),
('now', 'RB'),
('for', 'IN'),
('something', 'NN'),
('completely', 'RB'),
('different', 'JJ')]

→ text = word_tokenize("They refuse to permit us to obtain the refuse permit")
nltk.pos_tag(text)

(*) [('They', 'PRP'),
('refuse', 'VBP'),
('to', 'TO'),
('permit', 'VB'),
('us', 'PRP'),
('to', 'TO'),
('obtain', 'VB'),
('the', 'DT'),
('refuse', 'NN'),
('permit', 'NN')]

Teacher's Signature

Expt. No. 12 Date

Page No. 47

→ text = nltk.Text(word.lower() for word in nltk.corpus.brown.words())

text.similar('woman')

(*) man time day year car moment world house family child country boy state job place way war girl work word.

→ text.similar('bought')

(*) made said done put had seen found given left heard was been brought set got that took in told felt.

→ text.similar('over')

is on to of and for with from at by that into as up out down through is all about.

Teacher's Signature