K Venkata Ratnam
RA1911032020001
CSE-IOT

EXP.NO:1 IMPLEMENTATION OF TOY PROGRAMS

DATE:07/01/22

Aim: To implement tic-tac-toe toy problem.

Algorithm:

1. The program is started.
2. We created the boardnumbers.
3. We printed the specific board pattern.
4. Nowwewritefunctionsforbothplayerstogivepositioninput.
5. Wecheckedifplayer XorOwon,foreverymoveafter5moves.
6. We printed if X or O wins or they have a tie.
7. Atlast,apromptisaskediftheplayerwantstorestartthegame and recursion occurs here.
8. The program is ended.

Source Code:

```
#Implementation of Two Player Tic-Tac-Toe game in Python. theBoard = {'7': ' ' ,
'8': ' ' , '9': ' ' ,
            '4': '' , '5': '' , '6': '' ,
            '1': '', '2': '', '3': ''}
board_keys = []
for key in theBoard:
    board_keys.append(key)
def printBoard(board):
    print(board['7']+'|'+board['8']+'|'+board['9']) print('-+-+-
')
print(board['4'] + '|' + board['5'] + '|' + board['6'])
```

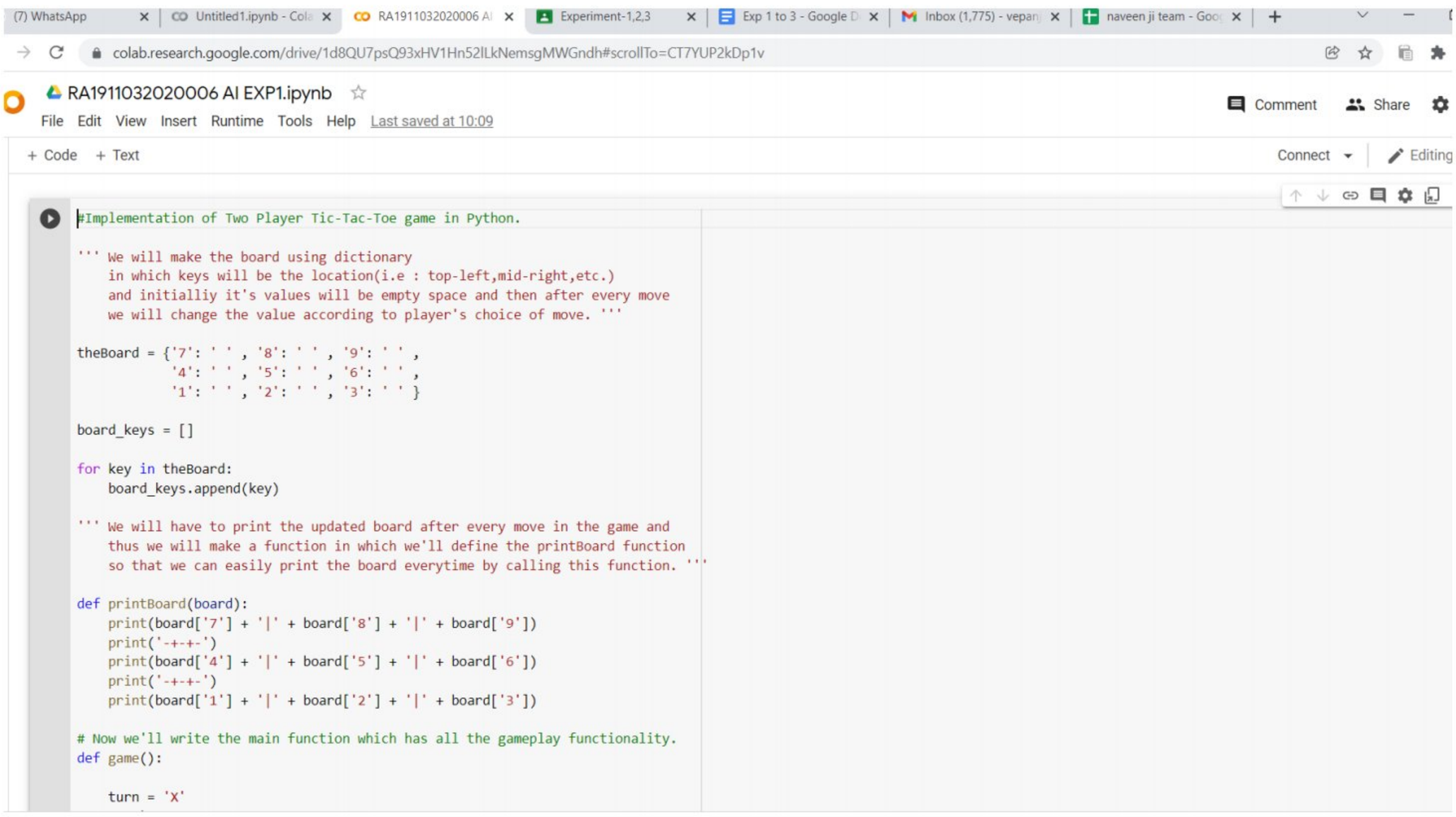
```
    print('-+-+-')
    print(board['1'] + '|' + board['2'] + '|' + board['3'])
#Nowwe'llwritethemainfunctionwhichhasallthegameplay functionality.
def      game():
    turn  =  'X'
    count=0
for      i      in      range(10):
    printBoard(theBoard)
    print("It'syourturn,"+turn+".Movetowhichplace?") move =input()
    iftheBoard[move]=='':
        theBoard[move]=turn count
        += 1
    else:
        print("That place is already filled.\nMove to which place?") continue
        #Nowwe willcheckifplayerXorOhaswon,foreverymoveafter5 moves.
        if count >= 5:
            if theBoard['7'] == theBoard['8'] == theBoard['9'] != ' ': # across
thetop
                printBoard(theBoard)
                print("\nGame Over.\n")
                print("****"+turn+"won.****") break
```

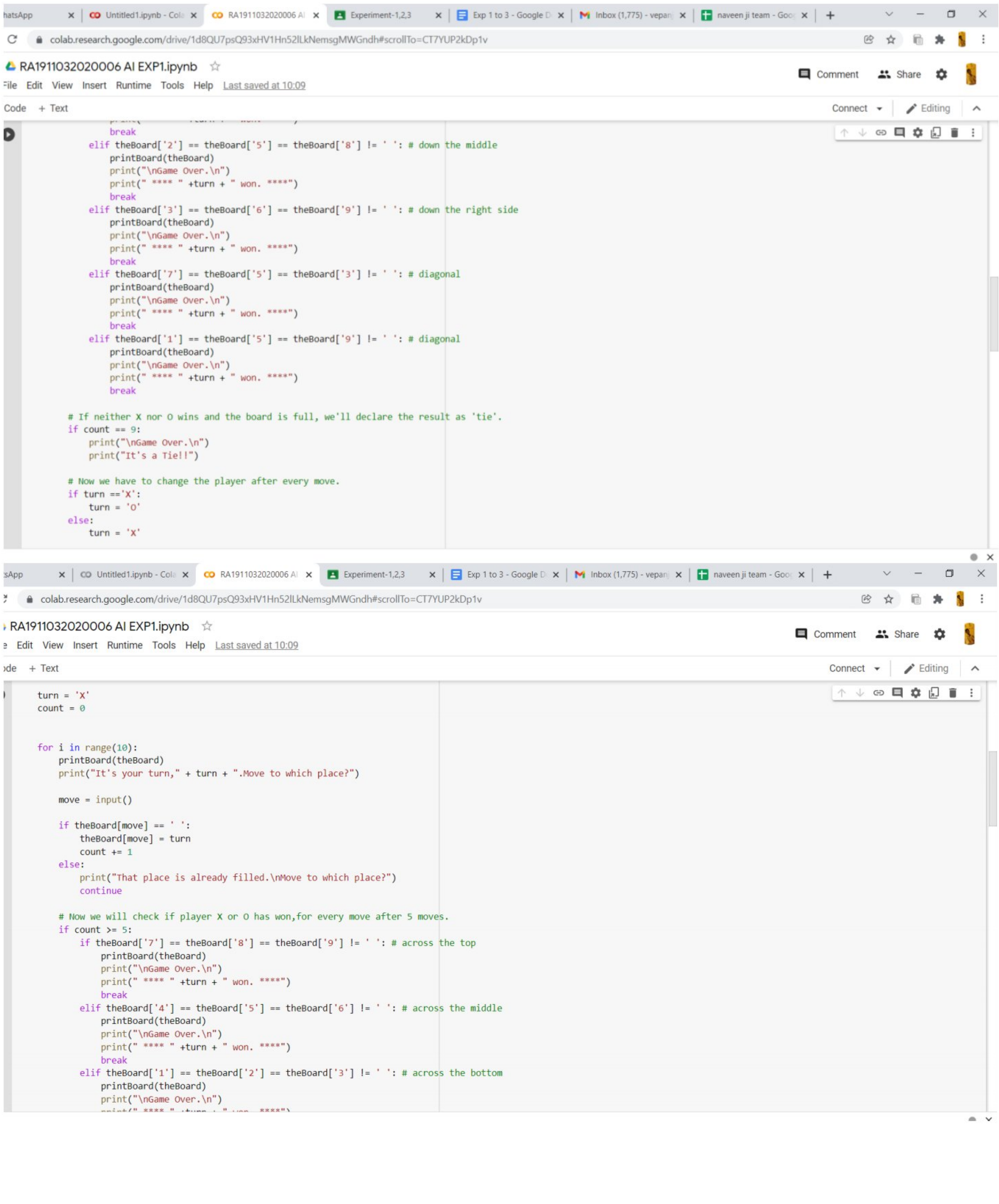
```
elif theBoard['4'] == theBoard['5'] == theBoard['6'] != '': #across the middle
    printBoard(theBoard)
    print("\nGame Over.\n")
    print("****"+turn+" won.****") break
elif theBoard['1'] == theBoard['2'] == theBoard['3'] != '': #across the bottom
    printBoard(theBoard)
    print("\nGame Over.\n")
    print("****"+turn+" won.****") break
elif theBoard['1'] == theBoard['4'] == theBoard['7'] != '': # down the left side
    printBoard(theBoard)
    print("\nGame Over.\n")
    print("****"+turn+" won.****") break
elif theBoard['2'] == theBoard['5'] == theBoard['8'] != '': # down the middle
    printBoard(theBoard)
    print("\nGame Over.\n")
    print("****"+turn+" won.****") break
elif theBoard['3'] == theBoard['6'] == theBoard['9'] != '': # down the right side
```

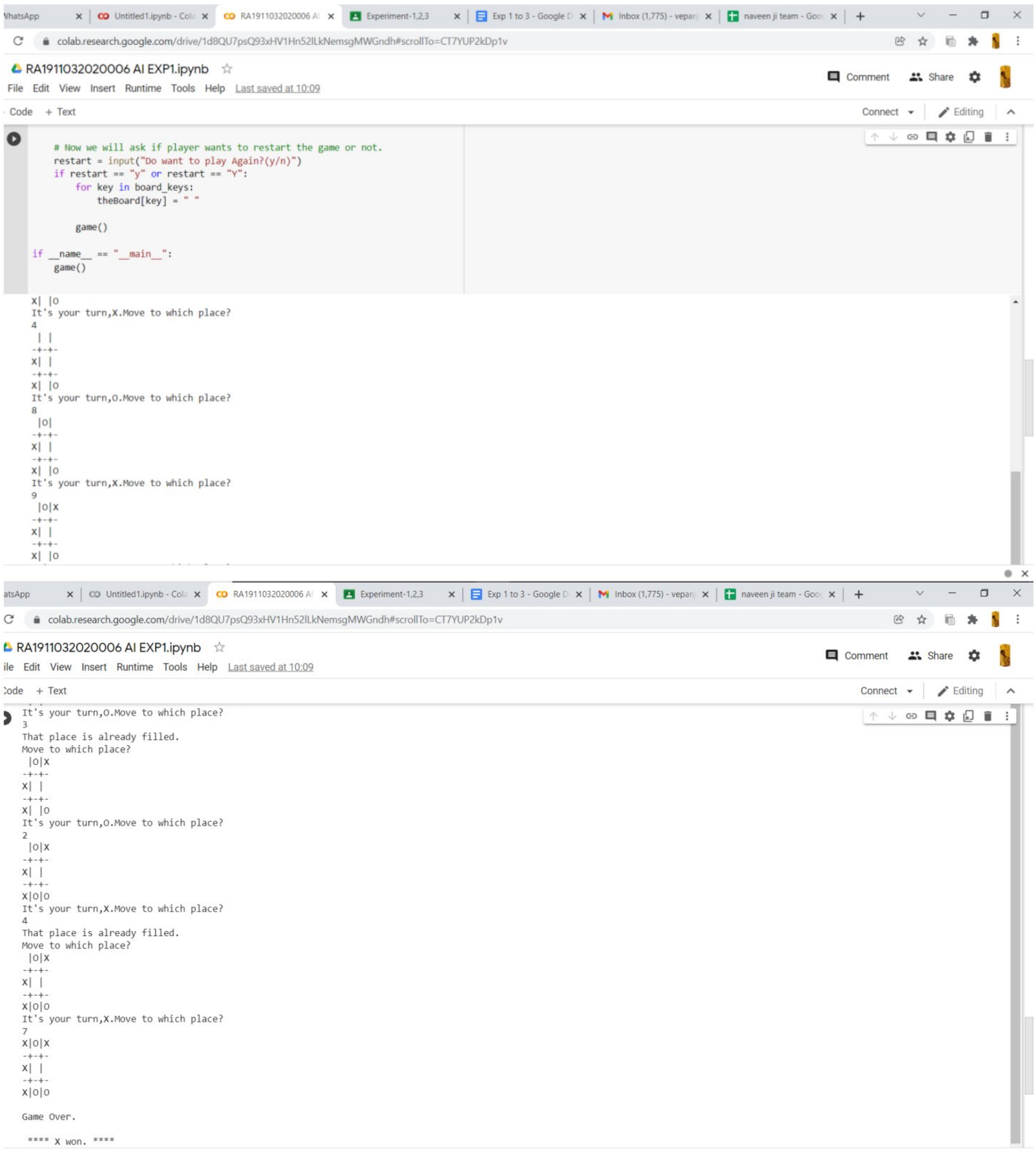
```
        printBoard(theBoard)
        print("\nGame Over.\n")
        print("****"+turn+"won.****") break
    elif theBoard['7'] == theBoard['5'] == theBoard['3'] != ' ': # diagonal
        printBoard(theBoard)
        print("\nGame Over.\n")
        print("****"+turn+"won.****") break
#If neither X nor O wins and the board is full, we'll declare the result as 'tie'.
if count == 9: print("\nGame
    Over.\n") print("It's a Tie!!")

#Now we have to change the player after every move. if turn =='X':
    turn='O'
    else:
```

```
        turn = 'X'
#Now we will ask if player wants to restart the game or not. restart = input("Do
want to play Again?(y/n)")
if restart == "y" or restart == "Y": for key in
    board_keys:
        theBoard[key] = ""
        game()
if __name__ == "__main__": game()
```

Output:







Result: Hence, Tic-Tac-Toe problem is implemented

Developing agent programs for real world problems (8-puzzle)

K Venkata Ratnam
RA1911032020001
CSE-IOT

EXP:2

DATE:21/01/22

Aim:

To develop agent programs for real world problems like 8-puzzle.

Algorithm:

1. The program is started.
2. We imported random and math modules.
3. Final state is created.
4. Eight puzzle matrix is created.
5. Solve function is used to solve the matrix we want to find.
6. Here function is used to provide current and target position for each number.
7. Required matrix is printed in each step.
8. The program is closed.

Source Code:

```
# Solves a randomized 8-puzzle using A* algorithm with plug-in
heuristics
import random
import math
_goal_state = [[1,2,3],
[4,5,6],
[7,8,0]]
def index(item, seq):
"""Helper function that returns -1 for non-found index value of a
seq"""
if item in seq:
return seq.index(item)
```

```
else:
return -1
class EightPuzzle:
def init (self):
# heuristic value
self._hval =0
# search depth of current instance
self._depth = 0
# parent node in search path
self._parent = None
self.adj_matrix =[]
for i in range(3):
self.adj_matrix.append(_goal_state[i][:])
def eq__(self,other):
ifself. class != other. class :
return False
else:
return self.adj_matrix == other.adj_matrix
def str__(self):
res = ''
for row in range(3):
res += ' '.join(map(str, self.adj_matrix[row]))
res += '\r\n'
return res
def _clone(self):


p = EightPuzzle()
for iinrange(3):
p.adj_matrix[i] = self.adj_matrix[i][:]
return p
```

```
def _get_legal_moves(self):
"""Returns list of tuples with which the free space may be swapped"""
# get row and column of the empty piece
row, col = self.find(0)
free = []
#findwhichpiecescanmovethere
if row > 0:
free.append((row- 1,col))
if col >0:
free.append((row,col-1))
if row <2:
free.append((row + 1, col))
if col <2:
free.append((row, col + 1))
return free
def _generate_moves(self):
free = self._get_legal_moves()
zero =self.find(0)
def swap_and_clone(a, b):
p= self._clone()


p.swap(a,b)
p._depth = self._depth + 1
p._parent =self
return p
return map(lambda pair: swap_and_clone(zero, pair), free)
def _generate_solution_path(self, path):
if self._parent == None:
return path
else:
path.append(self)
```

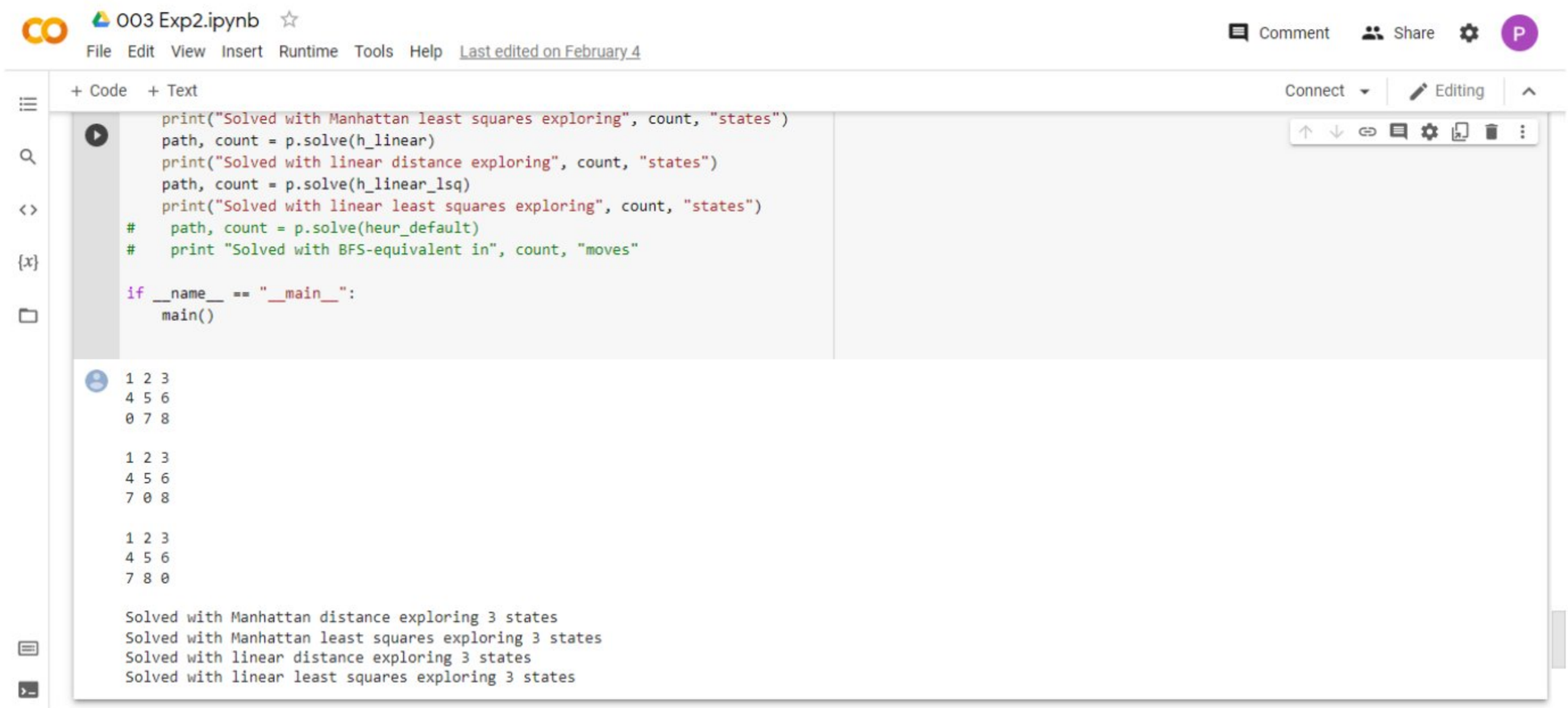
```
return self._parent._generate_solution_path(path)

def solve(self, h):

def is_solved(puzzle):

return puzzle.adj_matrix == _goal_state

openl = [self]

closedl = []

move_count = 0

whilelen(openl) > 0:

x = openl.pop(0)

move_count += 1

if (is_solved(x)):

if len(closedl) >0:

return x._generate_solution_path([]), move_count

else:

return [x]



succ = x._generate_moves()

idx_open = idx_closed = -1

for move in succ:

#havewealreadyseenthisnode?

idx_open = index(move, openl)

idx_closed = index(move, closedl)

hval = h(move)

fval = hval + move._depth

ifidx_closed==-1andidx_open==-1:

move._hval = hval

openl.append(move)

elif idx_open >-1:

copy =openl[idx_open]

if fval < copy._hval + copy._depth:

# copy move's values over existing
```

```
copy._hval = hval

copy._parent = move._parent

copy._depth = move._depth

elif idx_closed >-1:

copy = closedl[idx_closed]

iffval < copy._hval+ copy._depth:

move._hval = hval

closedl.remove(copy)

openl.append(move)

closedl.append(x)


openl = sorted(openl, key=lambda p: p._hval + p._depth)

# if finished state not found, return failure

return [], 0

def shuffle(self, step_count):

for i in range(step_count):

row, col =self.find(0)

free = self._get_legal_moves()

target = random.choice(free)

self.swap((row, col), target)

row, col = target

def find(self,value):

"""returns the row, col coordinates of the specified value

in the graph"""

if value < 0 or value > 8:

raise Exception("value out of range")

for row in range(3):

for col in range(3):

if self.adj_matrix[row][col] == value:

return row, col

def peek(self, row, col):
```

```
"""returns the value at the specified row and column"""
return self.adj_matrix[row][col]
def poke(self, row, col,value):
"""sets the value at the specified row and column"""
self.adj_matrix[row][col] = value


def swap(self, pos_a, pos_b):
"""swaps values at the specified coordinates"""
temp = self.peek(*pos_a)
self.poke(pos_a[0], pos_a[1], self.peek(*pos_b))
self.poke(pos_b[0], pos_b[1],temp)
defheur(puzzle,item_total_calc, total_calc):
t = 0
for rowinrange(3):
forcolinrange(3):
val = puzzle.peek(row, col)- 1
target_col = val % 3
target_row = val / 3
#account for0 as blank
if target_row < 0:
target_row = 2
t += item_total_calc(row, target_row, col, target_col)
return total_calc(t)
def h_manhattan(puzzle):
return heur(puzzle,
lambda r, tr, c, tc: abs(tr - r) + abs(tc - c),
lambda t : t)
def h_manhattan_lsq(puzzle):
return heur(puzzle,
lambda r, tr, c, tc:(abs(tr - r) + abs(tc -c))**2,
lambda t:math.sqrt(t))
```

```
def h_linear(puzzle):
return heur(puzzle,
lambdar, tr,c, tc:math.sqrt(math.sqrt((tr -r)**2+(tc-c)**2)),
lambda t: t)
def h_linear_lsq(puzzle):
return heur(puzzle,
lambda r, tr, c, tc:(tr - r)**2 + (tc - c)**2,
lambda t:math.sqrt(t))


def h_default(puzzle):
return 0
def main():
p = EightPuzzle()
p.shuffle(20)
print(p)
path, count = p.solve(h_manhattan)
path.reverse()
for iinpath:
print(i)
print("Solved with Manhattan distance exploring", count, "states")
path, count =p.solve(h_manhattan_lsq)
print("Solved with Manhattan least squares exploring", count, "states")
path, count = p.solve(h_linear)
print("Solved with linear distance exploring", count, "states")
path, count =p.solve(h_linear_lsq)
print("Solved with linear least squares exploring", count, "states")


if name__ == " main ":
main()
```

OUTPUT:



RESULT:

Hence, agent programs for real world problems like 8-puzzle implemented.

Implementation of constraint satisfaction problems (Cryptarithmetic Problem)

K Venkata Ratnam

RA1911032020001

CSE-IOT

EXP:3

DATE:04/02/22

AIM:

To Implement a constraint satisfaction problem (Cryptarithmetic Problem).

ALGORITHM:

1.Start program

2.Get input from the user

3.input is run through the functions

4.each input is calculated with a corresponding for loop

5.if the sum of the first two numbers is equal to third, return true. Else return False.

6.Then, it is put through a recursive function to check solution for all permutations

7.The result should satisfy the predefined arithmetic rules

8.Result is printed

9.End program

PROGRAM:

```
//CPP Program for Solving Cryptographic Puzzles

#include <bits/stdc++.h>

using namespace std;

vector<int> use(10);

struct node

{

char c;

int v;

};
```

```
int check(node* nodeArr, const int count, string s1,string s2, string s3)
{
int val1 = 0, val2 = 0, val3 = 0, m = 1, j, i;
for (i = s1.length() - 1; i >= 0; i--)
{
char ch = s1[i];
for (j = 0; j < count; j++)
if (nodeArr[j].c == ch)
break;
val1 += m * nodeArr[j].v;
m *= 10;
}
m = 1;
for (i = s2.length() - 1; i >= 0; i--)
{
char ch = s2[i];
for (j = 0; j < count; j++)
if (nodeArr[j].c == ch)
break;
val2 += m * nodeArr[j].v;
m *= 10;
}
m = 1;
for (i = s3.length() - 1; i >= 0; i--)
{
char ch = s3[i];
for (j = 0; j < count; j++)
if (nodeArr[j].c == ch)
break;
val3 += m * nodeArr[j].v;
```

```
m *= 10;
}
if (val3 == (val1 + val2))
return 1;

// else return false
return 0;
}
bool permutation(const int count, node* nodeArr, int n,
string s1, string s2, string s3)

{
if (n == count - 1)
{
for (int i = 0; i < 10; i++)
{
if (use[i] == 0)
{
nodeArr[n].v = i;
if (check(nodeArr, count, s1, s2, s3) == 1)
{
cout << "\nSolution found: ";
for (int j = 0; j < count; j++)
cout << " " << nodeArr[j].c << " = "
<< nodeArr[j].v;

return true;
}
}
}
return false;
```

```
}
for (int i = 0; i < 10; i++)
{
if (use[i] == 0)
{
nodeArr[n].v = i;
use[i] = 1;
if (permutation(count, nodeArr, n + 1, s1, s2, s3))
return true;
use[i] = 0;


}
}
return false;
}
bool solveCryptographic(string s1, string s2,


string s3)
{
int count = 0;
int l1 = s1.length();
int l2 = s2.length();
int l3 = s3.length();
vector<int> freq(26);
for (int i = 0; i < l1; i++)
++freq[s1[i] - 'A'];
for (int i = 0; i < l2; i++)
++freq[s2[i] - 'A'];
for (int i = 0; i < l3; i++)
++freq[s3[i] - 'A'];
for (int i = 0; i < 26; i++)
```

```
if (freq[i] > 0)
count++;
if (count > 10)
{
cout << "Invalid strings";
return 0;
}
node nodeArr[count];
for (int i = 0, j = 0; i < 26; i++)
{
if (freq[i] > 0)
{
nodeArr[j].c = char(i + 'A');
j++;
}
}
return permutation(count, nodeArr, 0, s1, s2, s3);
}
int main()
{
string s1 = "SEND";
string s2 = "MORE";
string s3 = "MONEY";
if (solveCryptographic(s1, s2, s3) == false)
cout << "No solution";
return 0;
}
```

OUTPUT:

003_Exp3.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on February 4

Comment Share P

+ Code + Text Connect Editing

```
  }
  return permutation(count, nodeArr, 0, s1, s2, s3);
}

// Driver function
int main()
{
  string s1 = "SEND";
  string s2 = "MORE";
  string s3 = "MONEY";

  if (solveCryptographic(s1, s2, s3) == false)
    cout << "No solution";
  return 0;
}
```

```
Overwriting AI3.cpp
```

```
[ ] %%shell
    g++ AI3.cpp -o output2
    ./output2
```

```
Solution found:  D = 1 E = 5 M = 0 N = 3 O = 8 R = 2 S = 7 Y = 6
```

RESULT:

Thus, constraint satisfaction Problem (Cryptarithmetic Problem) is implemented.

K Venkata Ratnam

RA1911032020001

CSE-IOT

Implementation and Analysis of DFS and BFS for an application

EXP.NO:4

DATE:11/02/22

AIM:

To Implement and Analyse the DFS and BFS Algorithm for an Application.

ALGORITHM:

DFS ALGORITHM

1. Start by putting any one of the graph's vertices on top of the stack.
2. After that take the top item of the stack and add it to the visited list of the vertex.
3. Next, create a list of that adjacent node of the vertex. Add the ones which aren't in the visited list of vertexes to the top of the stack.
4. Lastly, keep repeating steps 2 and 3 until the stack is empty.
5. Stop

BFS ALGORITHM

1. Start by putting any one of the graph's vertices at the back of the queue.
2. Now take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
4. Keep continuing steps two and three till the queue is empty.
5. Stop

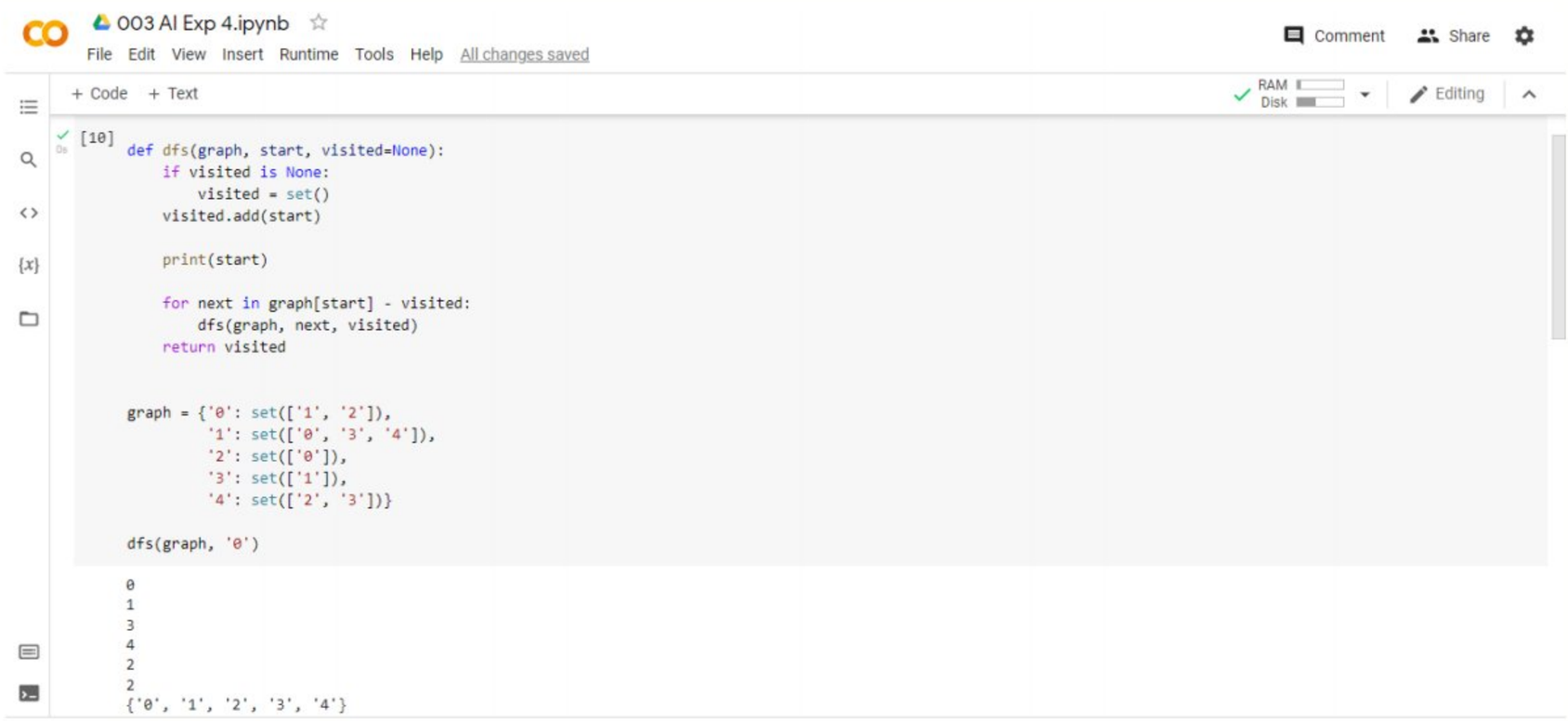
SOURCE CODE:

```
#  DFS ALGORITHM CODE

def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited
graph = {'0': set(['1', '2']),
        '1': set(['0', '3', '4']),
        '2': set(['0']),
        '3': set(['1']),
        '4': set(['2', '3'])}

dfs(graph, '0')
```

```
# BFS ALGORITHM CODE

import collections
def bfs(graph, root):

    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:

        vertex = queue.popleft()
        print(str(vertex) + " ", end="")

        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)


if __name__ == '__main__':
    graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
    print("Following is Breadth First Traversal: ")
    bfs(graph, 0)
```

OUTPUT:

DFS:



BFS:

003 AI Exp 4.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
# BFS ALGORITHM CODE

import collections
def bfs(graph, root):

    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:

        vertex = queue.popleft()
        print(str(vertex) + " ", end="")

        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)


if __name__ == '__main__':
    graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
    print("Following is Breadth First Traversal: ")
    bfs(graph, 0)
```

```
Following is Breadth First Traversal:
0 1 2 3
```

RESULT:

Thus, we have implemented and Analysed the DFS and BFS Algorithm for an Application Successfully.

Experiment – 5

Date:18/02/2022

Artificial Intelligence Laboratory

K VENKATA RATNAM

RA1911032020001

CSE-IOT

AIM:

To Develop Best first search and A* Algorithm for real world problems

ALGORITHM:

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

PROGRAM:

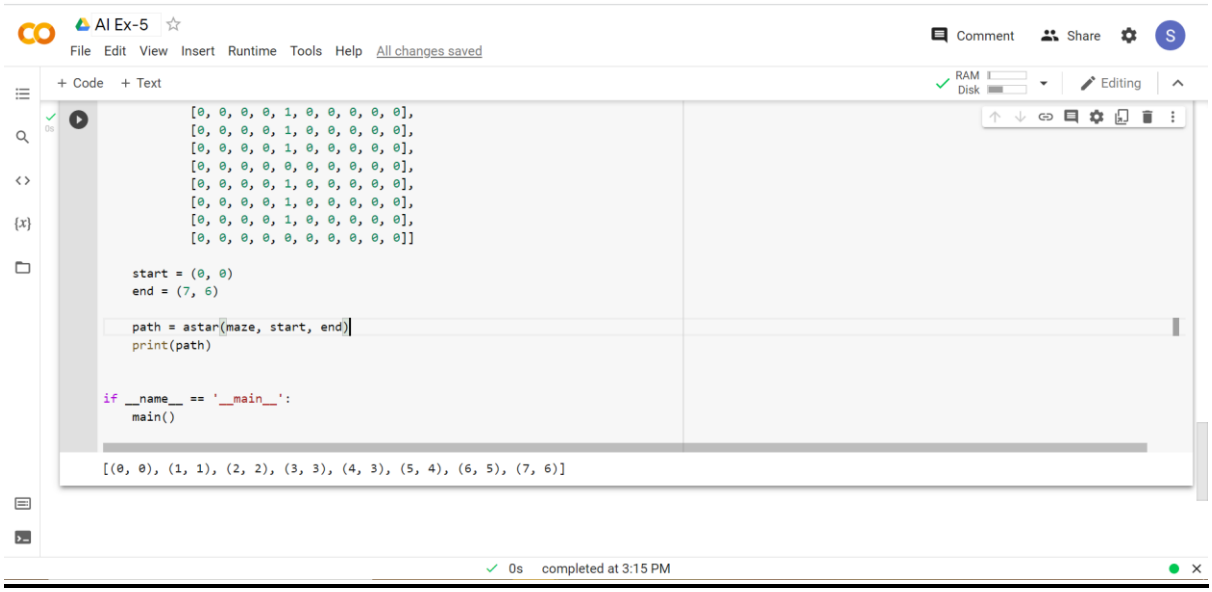
```
class Node():
    """A node class for A* Pathfinding"""

    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):
    """Returns a list of tuples as a path from the given start to the given end in
the given maze"""
```

```
    # Create start and end node
    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

    # Initialize both open and closed list
    open_list = []
    closed_list = []

    # Add the start node
    open_list.append(start_node)

    # Loop until you find the end
    while len(open_list) > 0:

        # Get the current node
        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

        # Pop current off open list, add to closed list
        open_list.pop(current_index)
        closed_list.append(current_node)

        # Found the goal
        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
            return path[::-1] # Return reversed path

        # Generate children
        children = []
        for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1,
1)]: # Adjacent squares

            # Get node position
```

```
            node_position = (current_node.position[0] + new_position[0],
current_node.position[1] + new_position[1])

            # Make sure within range
            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

            # Make sure walkable terrain
            if maze[node_position[0]][node_position[1]] != 0:
                continue

            # Create new node
            new_node = Node(current_node, node_position)

            # Append
            children.append(new_node)

        # Loop through children
        for child in children:

            # Child is on the closed list
            for closed_child in closed_list:
                if child == closed_child:
                    continue

            # Create the f, g, and h values
            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

            # Child is already in the open list
            for open_node in open_list:
                if child == open_node and child.g > open_node.g:
                    continue

            # Add the child to the open list
            open_list.append(child)
 def main():

    maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
```

```
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

    start = (0, 0)
    end = (7, 6)

    path = astar(maze, start, end)
    print(path)


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

OUTPUT:



RESULT:

Thus, The Best first search and A* Algorithm is developed in Python.

Experiment – 6

Date:25/02/2022

Artificial Intelligence

Laboratory

K Venkata Ratnam
RA1911032020001
CSE-IOT

AIM:- Implementation of Minmax algorithm of an application

ALGORITHM:

1. Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
2. Mini-Max algorithm uses recursion to search through the game-tree.
3. Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
4. In this algorithm two players play the game, one is called MAX and other is called MIN.
5. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
6. Both Players of the game are opponents of each other, where MAX will select the maximized value and MIN will select the minimized value.
7. The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
8. The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

PROGRAM:

```
import sys
import random
# This class represent a tic tac to game
class TicTacToeGame:
   # Create a new game
   def __init__(self, rows:int, columns:int, goal:int, max_depth:int=4):

      # Create the game state
      self.state = []
      self.tiles = {}
      self.inverted_tiles = {}
```

```
        tile = 0
        for y in range(rows):
            row = []
            for x in range(columns):
                row += '.'
                tile += 1
                self.tiles[tile] = (y, x)
                self.inverted_tiles[(y, x)] = tile
            self.state.append(row)
        # Set the number of noughts and crosses in a row that is needed to win the
game
        self.goal = goal
        # Create vectors
        self.vectors = [(1,0), (0,1), (1,1), (-1,1)]
        # Set lengths
        self.rows = rows
        self.columns = columns
        self.max_row_index = rows - 1
        self.max_columns_index = columns - 1
        self.max_depth = max_depth
        # Heuristics for cutoff
        self.winning_positions = []
        self.get_winning_positions()
        # Set the starting player at random
        #self.player = 'O'
        self.player = random.choice(['X', 'O'])
    # Get winning positions
    def get_winning_positions(self):
        # Loop the board
        for y in range(self.rows):
            for x in range(self.columns):

                # Loop vectors
                for vector in self.vectors:

                    # Get the start position
                    sy, sx = (y, x)
                    # Get vector deltas
                    dy, dx = vector
                    # Create a counter
                    counter = 0
                    # Loop until we are outside the board
                    positions = []
```

```
                while True:
                    # Add the position
                    positions.append(self.inverted_tiles.get((sy, sx)))
                    # Check if we have a winning position
                    if (len(positions) == self.goal):
                        # Add winning positions
                        self.winning_positions.append(positions)
                        # Break out from the loop
                        break
                    # Update the position
                    sy += dy
                    sx += dx

                    # Check if the loop should terminate
                    if(sy < 0 or abs(sy) > self.max_row_index or sx < 0 or abs(sx) >
self.max_columns_index):
                        break
    # Play the game
    def play(self):
        # Variables
        result = None
        # Create an infinite loop
        print('Starting board')
        while True:
            # Draw the state
            self.print_state()
            # Get a move from a player
            if (self.player == 'X'): # AI
                # Print AI move
                print('Player X moving (AI) ...')
                # Get the best move
                max, py, px, depth = self.max(-sys.maxsize, sys.maxsize)
                # Get a heuristic move at cutoff
                print('Depth: {0}'.format(depth))
                if(depth > self.max_depth):
                    py, px = self.get_best_move()
                # Make a move
                self.state[py][px] = 'X'
                # Check if the game has ended, break out from the loop in that case
                result = self.game_ended()
                if(result != None):
                    break
                # Change turn
```

```
            self.player = 'O'
        elif (self.player == 'O'): # Human player

            # Print turn
            print('Player O moving (Human) ...')
            # Get a recommended move
            min, py, px, depth = self.min(-sys.maxsize, sys.maxsize)
            # Get a heuristic move at cutoff
            print('Depth: {0}'.format(depth))
            if(depth > self.max_depth):
                py, px = self.get_best_move()
            # Print a recommendation
            print('Recommendation: {0}'.format(self.inverted_tiles.get((py, px))))
            # Get input
            number = int(input('Make a move (tile number): '))
            tile = self.tiles.get(number)
            # Check if the move is legal
            if(tile != None):

                # Make a move
                py, px = tile
                self.state[py][px] = 'O'
                # Check if the game has ended, break out from the loop in that case
                result = self.game_ended()
                if(result != None):
                    break
                # Change turn
                self.player = 'X'
            else:
                print('Move is not legal, try again.')
    # Print result
    self.print_state()
    print('Winner is player: {0}'.format(result))
# An evaluation function to get the best move based on heuristics
def get_best_move(self):
    # Create an heuristic dictionary
    heuristics = {}
    # Get all empty cells
    empty_cells = []
    for y in range(self.rows):
        for x in range(self.columns):
            if (self.state[y][x] == '.'):
                empty_cells.append((y, x))
```

```
        # Loop empty positions
        for empty in empty_cells:
            # Get numbered position
            number = self.inverted_tiles.get(empty)
            # Loop winning positions
            for win in self.winning_positions:
                # Check if number is in a winning position
                if(number in win):
                    # Calculate the number of X:s and O:s in the winning position
                    player_x = 0
                    player_o = 0
                    start_score = 1
                    for box in win:
                        # Get the position
                        y, x = self.tiles[box]
                        # Count X:s and O:s
                        if(self.state[y][x] == 'X'):
                            player_x += start_score if self.player == 'X' else start_score *
2
                            start_score *= 10
                        elif (self.state[y][x] == 'O'):
                            player_o += start_score if self.player == 'O' else start_score *
2
                            start_score *= 10
                    # Save heuristic
                    if(player_x == 0 or player_o == 0):
                        # Calculate a score
                        score = max(player_x, player_o) + start_score
                        # Update the score
                        if(heuristics.get(number) != None):
                            heuristics[number] += score
                        else:
                            heuristics[number] = score
        # Get the best move from the heuristic dictionary
        best_move = random.choice(empty_cells)
        best_count = -sys.maxsize
        for key, value in heuristics.items():
            if(value > best_count):
                best_move = self.tiles.get(key)
                best_count = value
        # Return the best move
        return best_move
    # Check if the game has ended
```

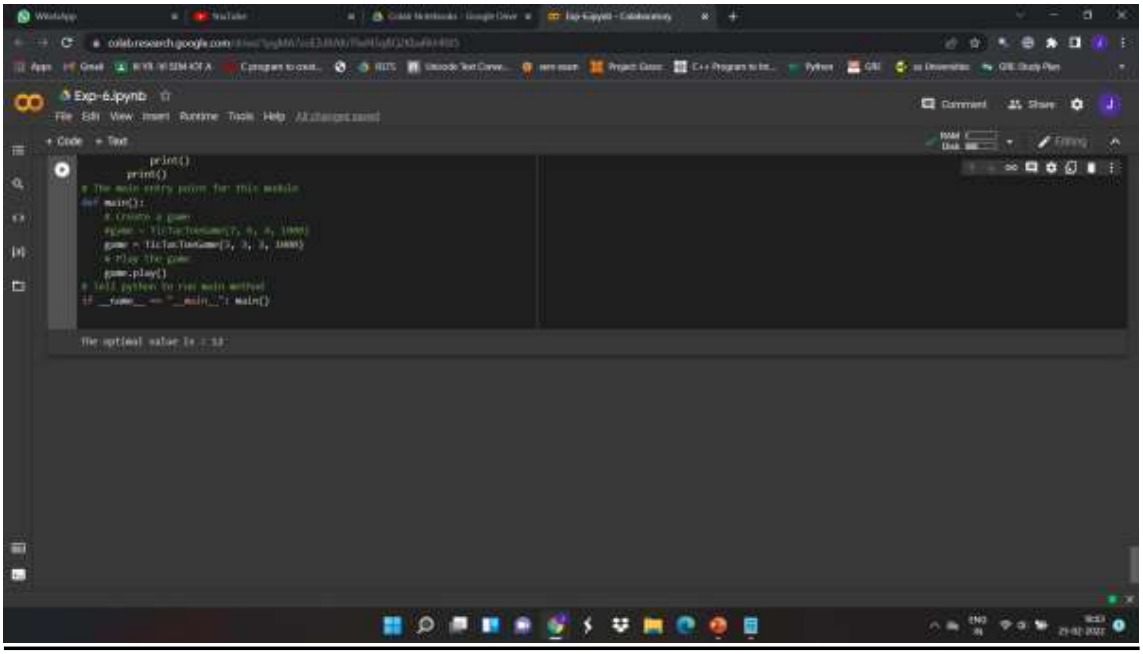
```
    def game_ended(self) -> str:
        # Check if a player has won
        result = self.player_has_won()
        if(result != None):
            return result
        # Check if the board is full
        for y in range(self.rows):
            for x in range(self.columns):
                if (self.state[y][x] == '.'):
                    return None
        # Return a tie
        return 'It is a tie!'

    # Check if a player has won
    def player_has_won(self) -> str:

        # Loop the board
        for y in range(self.rows):
            for x in range(self.columns):

                # Loop vectors
                for vector in self.vectors:

                    # Get the start position
                    sy, sx = (y, x)
                    # Get vector deltas
                    dy, dx = vector
                    # Create counters
                    steps = 0
                    player_x = 0
                    player_o = 0
                    # Loop until we are outside the board or have moved the number of
steps in the goal
                    while steps < self.goal:
                        # Add steps
                        steps += 1
                        # Check if a player has a piece in the tile
                        if(self.state[sy][sx] == 'X'):
                            player_x += 1
                        elif(self.state[sy][sx] == 'O'):
                            player_o += 1
                        # Update the position
                        sy += dy
```

```
                    sx += dx

                    # Check if the loop should terminate
                    if(sy < 0 or abs(sy) > self.max_row_index or sx < 0 or abs(sx) >
self.max_columns_index):
                        break
                # Check if we have a winner
                if(player_x >= self.goal):
                    return 'X'
                elif(player_o >= self.goal):
                    return 'O'
        # Return None if no winner is found
        return None
    # Get a min value (O)
    def min(self, alpha:int=-sys.maxsize, beta:int=sys.maxsize, depth:int=0):

        # Variables
        min_value = sys.maxsize
        by = None
        bx = None

        # Check if the game has ended
        result = self.game_ended()
        if(result != None):
            if result == 'X':
                return 1, 0, 0, depth
            elif result == 'O':
                return -1, 0, 0, depth
            elif result == 'It is a tie!':
                return 0, 0, 0, depth
        elif(depth > self.max_depth):
            return 0, 0, 0, depth
        # Loop the board
        for y in range(self.rows):
            for x in range(self.columns):
                # Check if the tile is empty
                if (self.state[y][x] == '.'):
                    # Make a move
                    self.state[y][x] = 'O'
                    # Get max value
                    max, max_y, max_x, depth = self.max(alpha, beta, depth + 1)

                    # Set min value to max value if it is lower than curren min value
```

```
                if (max < min_value):
                    min_value = max
                    by = y
                    bx = x

                # Reset the tile
                self.state[y][x] = '.'
                # Do an alpha test
                if (min_value <= alpha):
                    return min_value, bx, by, depth
                # Do a beta test
                if (min_value < beta):
                    beta = min_value
        # Return min value
        return min_value, by, bx, depth
    # Get max value (X)
    def max(self, alpha:int=-sys.maxsize, beta:int=sys.maxsize, depth:int=0):
        # Variables
        max_value = -sys.maxsize
        by = None
        bx = None
        # Check if the game has ended
        result = self.game_ended()
        if(result != None):
            if result == 'X':
                return 1, 0, 0, depth
            elif result == 'O':
                return -1, 0, 0, depth
            elif result == 'It is a tie!':
                return 0, 0, 0, depth
        elif(depth > self.max_depth):
            return 0, 0, 0, depth
        # Loop the board
        for y in range(self.rows):
            for x in range(self.columns):
                # Check if the current tile is empty
                if (self.state[y][x] == '.'):

                    # Add a piece to the board
                    self.state[y][x] = 'X'
                    # Set max value to min value if min value is greater than current
max value
                    min, min_y, min_x, depth = self.min(alpha, beta, depth + 1)
```

```
                # Adjust the max value
                if (min > max_value):
                    max_value = min
                    by = y
                    bx = x
                # Reset the tile
                self.state[y][x] = '.'
                # Do a beta test
                if (max_value >= beta):
                    return max_value, bx, by, depth
                # Do an alpha test
                if (max_value > alpha):
                    alpha = max_value
        # Return max value
        return max_value, by, bx, depth
    # Print the current game state
    def print_state(self):
        for y in range(self.rows):
            print('| ', end='')
            for x in range(self.columns):
                if (self.state[y][x] != '.'):
                    print(' {0}  | '.format(self.state[y][x]), end='')
                else:
                    digit = str(self.inverted_tiles.get((y,x))) if
len(str(self.inverted_tiles.get((y,x)))) > 1 else ' ' +
str(self.inverted_tiles.get((y,x)))
                    print('{0}  | '.format(digit), end='')
            print()
        print()
# The main entry point for this module
def main():
    # Create a game
    #game = TicTacToeGame(7, 6, 4, 1000)
    game = TicTacToeGame(3, 3, 3, 1000)
    # Play the game
    game.play()
# Tell python to run main method
if __name__ == "__main__": main()
```

OUTPUT:



RESULT:

Thus, Minmax algorithm is developed in Python.

Artificial Intelligence Laboratory

K Venkata Ratnam

RA1911032020001

CSE-IOT

Experiment – 7

Date:04/03/2022

AIM:

To Implement Unification and resolution in Python.

Algorithm:

1. If Ψ1 or Ψ2 is a variable or constant, then:

    i. If Ψ1 or Ψ2 are identical, then return NULL.

    ii. Else if Ψ1 is a variable:

1. Then if Ψ1 occurs in Ψ2, then return False
2. Else return (Ψ2 / Ψ1)

    iii. Else if Ψ2 is a variable:

1. Then if Ψ2 occurs in Ψ1, then return False
2. Else return (Ψ1 / Ψ2)

    iv. Else return False

2. If the initial Predicate symbol in Ψ1 and Ψ2 are not same, then returnFalse.
3. IF Ψ1 and Ψ2 have a different number of arguments, then return False.
4. Create Substitution list.
5. For i=1 to the number of elements in Ψ1.

    i. Call Unify function with the ith element of Ψ1 and ith

    element of Ψ2, and put the result into S.

    ii. If S = False then returns False

    iii. If S ≠ Null then append to Substitution list

6. Return Substitution list.

Program:

```
def get_index_comma(string):
"""
Return index of commas in string
"""
index_list = list()
# Count open parentheses
par_count = 0
for i in range(len(string)):

if string[i] == ',' and par_count == 0:
index_list.append(i)
elif string[i] == '(':
par_count += 1
elif string[i] == ')':
par_count -= 1
return index_list

def is_variable(expr):
"""
Check if expression is variable
"""
for i in expr:
if i == '(':
return False
return True
```

```
def process_expression(expr):
"""
input: - expression:
'Q(a, g(x, b), f(y))'
return: - predicate symbol:
Q
- list of arguments
['a', 'g(x, b)', 'f(y)']
"""
# Remove space in expression
expr = expr.replace(' ', '')
# Find the first index == '('
index = None
for i in range(len(expr)):
if expr[i] == '(':
index = i
break
# Return predicate symbol and remove predicate symbol in expression
predicate_symbol = expr[:index]
expr = expr.replace(predicate_symbol, '')
# Remove '(' in the first index and ')' in the last index
expr = expr[1:len(expr) - 1]
# List of arguments
arg_list = list()
# Split string with commas, return list of arguments
indices = get_index_comma(expr)
if len(indices) == 0:
arg_list.append(expr)
```

```
else:
arg_list.append(expr[:indices[0]])
for i, j in zip(indices, indices[1:]):
arg_list.append(expr[i + 1:j])
arg_list.append(expr[indices[len(indices) - 1] + 1:])
return predicate_symbol, arg_list
def get_arg_list(expr):
"""
input: expression:
'Q(a, g(x, b), f(y))'
return: full list of arguments:
['a', 'x', 'b', 'y']
"""
_, arg_list = process_expression(expr)
flag = True
while flag:
flag = False
for i in arg_list:
if not is_variable(i):
flag = True
_, tmp = process_expression(i)
for j in tmp:
if j not in arg_list:
arg_list.append(j)
arg_list.remove(i)
return arg_list
def check_occurs(var, expr):
"""
```

```
Check if var occurs in expr
"""
arg_list = get_arg_list(expr)
if var in arg_list:
return True
return False
def unify(expr1, expr2):
# Step 1:
if is_variable(expr1) and is_variable(expr2):
if expr1 == expr2:
return 'Null'
else:
return False
elif is_variable(expr1) and not is_variable(expr2):
if check_occurs(expr1, expr2):
return False
else:
tmp = str(expr2) + '/' + str(expr1)
return tmp
elif not is_variable(expr1) and is_variable(expr2):
if check_occurs(expr2, expr1):
return False
else:
tmp = str(expr1) + '/' + str(expr2)
return tmp
else:
predicate_symbol_1, arg_list_1 = process_expression(expr1)
predicate_symbol_2, arg_list_2 = process_expression(expr2)
```

```
# Step 2
if predicate_symbol_1 != predicate_symbol_2:
return False

# Step 3
elif len(arg_list_1) != len(arg_list_2):
return False
else:
# Step 4: Create substitution list
sub_list = list()
# Step 5:
for i in range(len(arg_list_1)):
tmp = unify(arg_list_1[i], arg_list_2[i])
if not tmp:
return False
elif tmp == 'Null':
pass
else:
if type(tmp) == list:
for j in tmp:
sub_list.append(j)
else:
sub_list.append(tmp)
# Step 6
return sub_list
if __name__ == '__main__':
# Data 1
f1 = 'p(b(A), X, f(g(Z)))'
```

```
f2 = 'p(Z, f(Y), f(Y))'
# Data 2
# f1 = 'Q(a, g(x, a), f(y))'
# f2 = 'Q(a, g(f(b), a), x)'
# Data 3
# f1 = 'Q(a, g(x, a, d), f(y))'
# f2 = 'Q(a, g(f(b), a), x)'
result = unify(f1, f2)
if not result:
print('Unification failed!')
else:
print('Unification successful!')
print(result)
```

Output:

Result:-

Thus, Unification and resolution have been implemented in Python.

Date: 11-03-2022 **RA1911032020001**

ARTIFICIAL INTELLIGENCE LABORATORY **K. V. Ratnam**

CSE – IOT

IMPLEMENTATION OF KNOWLEDGE REPRESENTATION SCHEMES

Exp no 8: Implementation of Knowledge Representation schemes

ALGORITHM:

1) Start

2) Get the input.

3) Check function to check if all cells are assigned or not. If there is any unassigned cell, then this function will change the values of row and col accordingly.

4) Execute function to check if we can put a value in a particular cell or not.

5) If all cells are assigned then the sudoku is already solved. Else, pass by reference because number_unassigned will change the values of row and col

6) Assign a solution for the sudoku. if we can't proceed with this solution then, reassign the cell.

7) If a Solution is found, Print the Solved Sudoku. Else, print “No Solution”.

8) Stop.

Source Code:

```
#include <stdio.h>
#define SIZE 9
//sudoku problem
int matrix[9][9] = {
   {5,3,0,0,7,0,0,0,0},
   {6,0,0,1,9,5,0,0,0},
   {0,9,8,0,0,0,0,6,0},
   {8,0,0,0,6,0,0,0,3},
   {4,0,0,8,0,3,0,0,1},
   {7,0,0,0,2,0,0,0,6},
   {0,6,0,0,0,0,2,8,0},
```

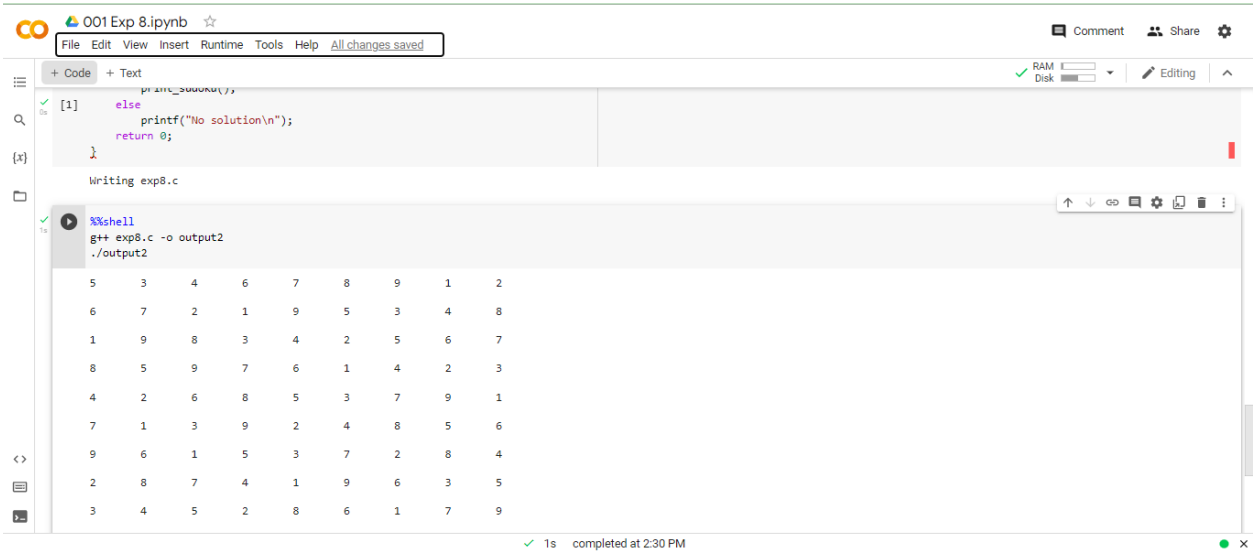
```
    {0,0,0,4,1,9,0,0,5},
    {0,0,0,0,8,0,0,7,9}
};
//function to print sudoku
void print_sudoku()
{
    int i,j;
    for(i=0;i<SIZE;i++)
    {
        for(j=0;j<SIZE;j++)
        {
            printf("%d\t",matrix[i][j]);
        }
        printf("\n\n");
    }
}
//function to check if all cells are assigned or not
//if there is any unassigned cell
//then this function will change the values of
//row and col accordingly
int number_unassigned(int *row, int *col)
{
    int num_unassign = 0;
    int i,j;
    for(i=0;i<SIZE;i++)
    {
        for(j=0;j<SIZE;j++)
        {
            //cell is unassigned
            if(matrix[i][j] == 0)
            {
                //changing the values of row and col
                *row = i;
```

```
            *col = j;
            //there is one or more unassigned cells
            num_unassign = 1;
            return num_unassign;
          }
        }
    }
    return num_unassign;
}
//function to check if we can put a
//value in a paticular cell or not
int is_safe(int n, int r, int c)
{
    int i,j;
    //checking in row
    for(i=0;i<SIZE;i++)
    {
        //there is a cell with same value
        if(matrix[r][i] == n)
            return 0;
    }
    //checking column
    for(i=0;i<SIZE;i++)
    {
        //there is a cell with the value equal to i
        if(matrix[i][c] == n)
            return 0;
    }
    //checking sub matrix
    int row_start = (r/3)*3;
    int col_start = (c/3)*3;
    for(i=row_start;i<row_start+3;i++)
    {
```

```
        for(j=col_start;j<col_start+3;j++)
        {
            if(matrix[i][j]==n)
                return 0;
        }
    }
    return 1;
}
//function to solve sudoku
//using backtracking
int solve_sudoku()
{
    int row;
    int col;
    //if all cells are assigned then the sudoku is already solved
    //pass by reference because number_unassigned will change the values of row and col
    if(number_unassigned(&row, &col) == 0)
        return 1;
    int n,i;
    //number between 1 to 9
    for(i=1;i<=SIZE;i++)
    {
        //if we can assign i to the cell or not
        //the cell is matrix[row][col]
        if(is_safe(i, row, col))
        {
            matrix[row][col] = i;
            //backtracking
            if(solve_sudoku())
                return 1;
            //if we can't proceed with this solution
            //reassign the cell
            matrix[row][col]=0;
```

```
        }
    }
    return 0;
}
int main()
{
    if (solve_sudoku())
        print_sudoku();
    else
        printf("No solution\n");
    return 0;
}
```

Output:



Result: Thus, we have Implemented Knowledge Representation Schemes Successfully.

Experiment – 9

Date:18/03/2022

Artificial Intelligence

Laboratory

Name :P.R.Samraysh
Class :B.Tech CSE IOT 'A'
Reg No. : RA1911032020009

AIM:

To Implement Uncertain Methods for an application in Python.

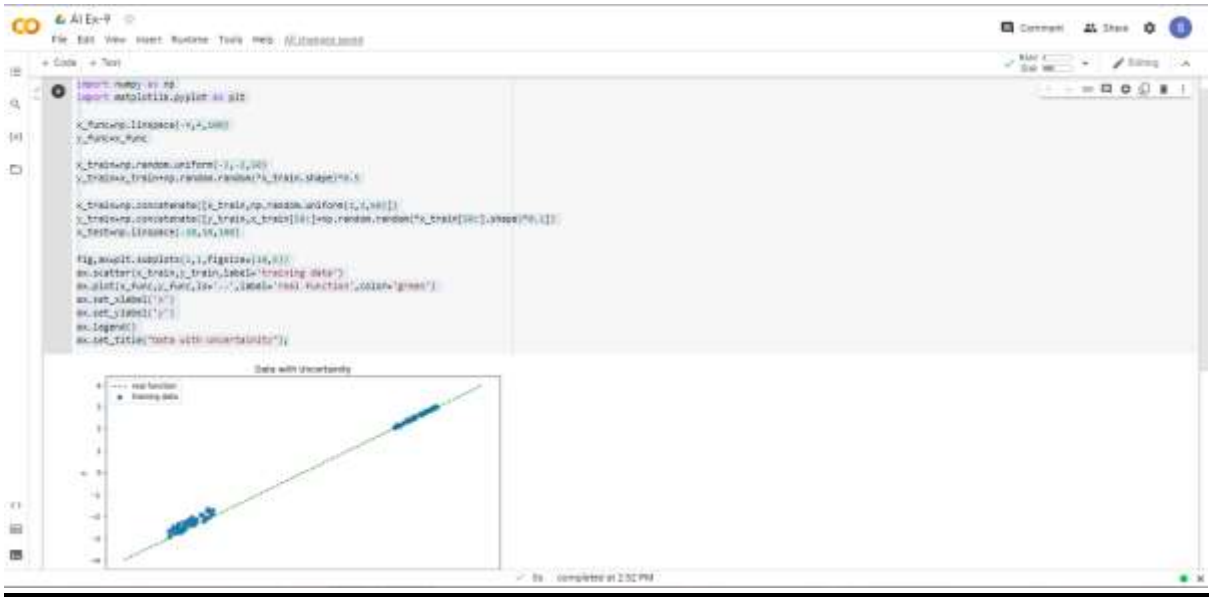
ALGORITHM:

1. Start Program.
2. Import the required libraries.
3. Get the input for functions.
4. Train the program using the data.
5. Set X and Y axes on a graph.
6. Plot the points.
7. Print the Graph.
8. End Program.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt

x_func=np.linspace(-4,4,100)
y_func=x_func

x_train=np.random.uniform(-3,-2,50)
y_train=x_train+np.random.random(*x_train.shape)*0.5

x_train=np.concatenate([x_train,np.random.uniform(2,3,50)])
y_train=np.concatenate([y_train,x_train[50:]+np.random.random(*x_train[50:].
shape)*0.1])
x_test=np.linspace(-10,10,100)

fig,ax=plt.subplots(1,1,figsize=(10,5))
ax.scatter(x_train,y_train,label='training data')
ax.plot(x_func,y_func,ls='--',label='real function',color='green')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend()
ax.set_title("Data with Uncertainity");
```

OUTPUT:



RESULT:

Thus, Uncertain Methods for an application have been implemented in Python.

EXP:10 **REGNO:RA1911032020001**

DATE:25/03/22 **NAME: K. Ratnam**

CLASS: CSE-IOT

Artificial Intelligence Laboratory

Implementation of block world problem

AIM:

To Implement of block world problems.

DESCRIPTION:

- Uniformed Search
- breadth (Breadth-First Search
- depth (Depth-First Search)
- Informed Search
- best (Best-First Search))
- astar (A-StarSearch)

ALGORITHM:

Breadth First Search (BFS) There are many ways to traverse graphs. BFS is the most commonly used approach.

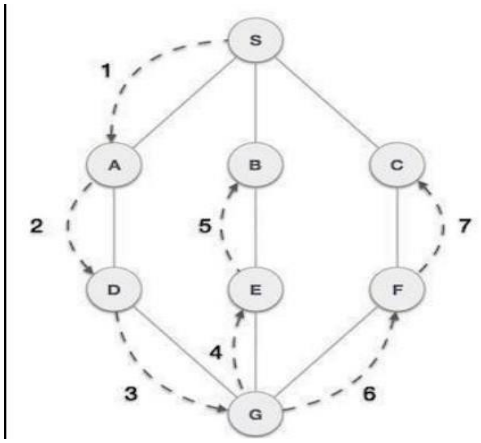
BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer

2. Move to the next layer

Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

- Rule 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- Rule 2 − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- Rule 3 − Repeat Rule 1 and Rule 2 until the stack is empty

Best First Search Algorithm

1. Create 2 empty lists: OPEN and CLOSED

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

3. Repeat the next steps until GOAL node is reached

    1. If OPEN list is empty, then EXIT the loop returning 'False'

    2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also capture the information of the parent node

    3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path

    4. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list

    5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

This algorithm will traverse the shortest path first in the queue. The time complexity of the algorithm is given by O(n*logn) .

Algorithm

We create two lists – Open List and Closed List (just like Dijkstra Algorithm)

// A* Search Algorithm

1. Initialize the open list

2. Initialize the closed list put the starting node on the open list (you can leave its f at zero)

3. while the open list is not empty

a) find the node with the least f on the open list, call it "q"

b) pop q off the open list

c)generate q's 8 successors and set their parents to q

d) for each successor

i) if successor is the goal, stop search

successor.g = q.g + distance between successor and q

successor.h = distance from goal to successor (This can be done using many ways, we will discuss three heuristics Manhattan, Diagonal and Euclidean Heuristics)

successor.f = successor.g + successor.

ii) if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor

iii) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list end (for loop)

e) push q on the closed list end (while loop)

PROGRAM:

```
class  PREDICATE:
  def__str__(self):
    pass
  def_repr_(self):
    pass
  def_eq_(self, other) :
    pass
  def_hash_(self):
    pass
  def get_action(self, world_state):
    pass

class  Operation:
  def__str__(self):
    pass
  def_repr_(self):
    pass
  def_eq_(self, other) :
    pass
  def precondition(self):
    pass
  def delete(self):
    pass
  def add(self):
    pass
```

```
class ON(PREDICATE):
  def__init__(self, X, Y):
    self.X = X
    self.Y = Y
  def__str__(self):
    return "ON({X},{Y})".format(X=self.X,Y=self.Y)
  def__repr__(self):
    return self.__str__()
  def__eq__(self, other) :
    return self.__dict__== other.__dict__and self.__class__== other.
__class__
  def__hash__(self):
      return hash(str(self))
  def get_action(self, world_state):
    return StackOp(self.X,self.Y)

class ONTABLE(PREDICATE):
  def__init__(self, X):
    self.X = X
  def__str__(self):
    return "ONTABLE({X})".format(X=self.X)
  def__repr__(self):
    return self.__str__()
  def__eq__(self, other) :
    return self.__dict__== other.__dict__and self.__class__== other.
__class__
  def__hash__(self):
      return hash(str(self))
  def get_action(self, world_state):
    return PutdownOp(self.X)

class CLEAR(PREDICATE):
  def__init__(self, X):
    self.X = X
  def__str__(self):
    return "CLEAR({X})".format(X=self.X)
    self.X = X
  def__repr__(self):
    return self.__str__()
  def__eq__(self, other) :
    return self.__dict__== other.__dict__and self.__class__== other.
__class__
  def__hash__(self):
    return hash(str(self))
  def get_action(self, world_state):
    for predicate in world_state:
      if isinstance(predicate,ON) and predicate.Y==self.X:
```

```
        return UnstackOp(predicate.X, predicate.Y)
    return None

class HOLDING(PREDICATE):
  def __init__(self, X):
    self.X = X
  def __str__(self):
    return "HOLDING({X})".format(X=self.X)
  def __repr__(self):
    return self.__str__()
  def __eq__(self, other) :
    return self.__dict__ == other.__dict__ and self.__class__ == other.
__class__
  def __hash__(self):
    return hash(str(self))
  def get_action(self, world_state):
    X = self.X
    if ONTABLE(X) in world_state:
      return PickupOp(X)
    else:
      for predicate in world_state:
        if isinstance(predicate,ON) and predicate.X==X:
          return UnstackOp(X,predicate.Y)

class ARMEMPTY(PREDICATE):
  def __init__(self):
    pass
  def __str__(self):
    return "ARMEMPTY"
  def __repr__(self):
    return self.__str__()
  def __eq__(self, other) :
    return self.__dict__ == other.__dict__ and self.__class__ == other.
__class__
  def __hash__(self):
    return hash(str(self))
  def get_action(self, world_state=[]):
    for predicate in world_state:
      if isinstance(predicate,HOLDING):
        return PutdownOp(predicate.X)
    return None

class StackOp(Operation):
  def __init__(self, X, Y):
    self.X = X
    self.Y = Y
  def __str__(self):
    return "STACK({X},{Y})".format(X=self.X,Y=self.Y)
```

```
  def __repr__(self):
    return self.__str__()
  def __eq__(self, other) :
    return self.__dict__ == other.__dict__ and self.__class__ == other.
__class__
  def precondition(self):
    return [ CLEAR(self.Y) , HOLDING(self.X) ]
  def delete(self):
    return [ CLEAR(self.Y) , HOLDING(self.X) ]
  def add(self):
    return [ ARMEMPTY() , ON(self.X,self.Y) ]

class UnstackOp(Operation):
  def __init__(self, X, Y):
    self.X = X
    self.Y = Y
  def __str__(self):
    return "UNSTACK({X},{Y})".format(X=self.X,Y=self.Y)
  def __repr__(self):
    return self.__str__()
  def __eq__(self, other) :
    return self.__dict__ == other.__dict__ and self.__class__ == other.
__class__
  def precondition(self):
    return [ ARMEMPTY() , ON(self.X,self.Y) , CLEAR(self.X) ]
  def delete(self):
    return [ ARMEMPTY() , ON(self.X,self.Y) ]
  def add(self):
    return [ CLEAR(self.Y) , HOLDING(self.X) ]

class PickupOp(Operation):
  def __init__(self, X):
    self.X = X
  def __str__(self):
    return "PICKUP({X})".format(X=self.X)
  def __repr__(self):
    return self.__str__()
  def __eq__(self, other) :
    return self.__dict__ == other.__dict__ and self.__class__ == other.
__class__
  def precondition(self):
    return [ CLEAR(self.X) , ONTABLE(self.X) , ARMEMPTY() ]
  def delete(self):
    return [ ARMEMPTY() , ONTABLE(self.X) ]
  def add(self):
    return [ HOLDING(self.X) ]


class PutdownOp(Operation):
```

```
  def __init__(self, X):
    self.X = X
  def __str__(self):
    return "PUTDOWN({X})".format(X=self.X)
  def __repr__(self):
    return self.__str__()
  def __eq__(self, other) :
    return self.__dict__ == other.__dict__ and self.__class__ == other.
__class__
  def precondition(self):
    return [ HOLDING(self.X) ]
  def delete(self):
    return [ HOLDING(self.X) ]
  def add(self):
    return [ ARMEMPTY() , ONTABLE(self.X) ]

def isPredicate(obj):
  predicates = [ON, ONTABLE, CLEAR, HOLDING, ARMEMPTY]
  for predicate in  predicates:
    if isinstance(obj,predicate):
      return True
  return False

def isOperation(obj):
  operations = [StackOp, UnstackOp, PickupOp, PutdownOp]
  for operation in operations:
    if isinstance(obj,operation):
      return True
  return False

def arm_status(world_state):
  for predicate in world_state:
    if isinstance(predicate, HOLDING):
      return predicate
  return ARMEMPTY()


class GoalStackPlanner:

  def __init__(self, initial_state, goal_state):
    self.initial_state = initial_state
    self.goal_state = goal_state

  def get_steps(self):
    steps = []
    stack = []

    #World State/Knowledge Base
    world_state = self.initial_state.copy()
```

```
    #Initially push the goal_state as compound goal onto the stack
    stack.append(self.goal_state.copy())

    #Repeat until the stack is empty
    while len(stack)!=0:

      #Get the top of the stack
      stack_top = stack[-1]

      #If Stack Top is Compound Goal, push its unsatisfied goals onto s
tack
      if type(stack_top) is list:
        compound_goal = stack.pop()
        for goal in compound_goal:
          if goal not in world_state:
            stack.append(goal)

      elif  isOperation(stack_top):
        operation = stack[-1]
        all_preconditions_satisfied = True
        for predicate in operation.delete():
          if predicate not in world_state:
            all_preconditions_satisfied = False
            stack.append(predicate)

        if all_preconditions_satisfied:

          stack.pop()
          steps.append(operation)

          for predicate in operation.delete():
            world_state.remove(predicate)
          for predicate in operation.add():
            world_state.append(predicate)

      elif stack_top in world_state:
        stack.pop()
      else:
        unsatisfied_goal = stack.pop()
        action = unsatisfied_goal.get_action(world_state)

        stack.append(action)
        for predicate in action.precondition():
          if predicate not in world_state:
            stack.append(predicate)

    return steps
```

```
if__name___== '_main_':
  initial_state = [
    ON('B','A'),ON('E', 'B'),
    ONTABLE('A'),ONTABLE('C'),ONTABLE('D'),
    CLEAR('B'),CLEAR('C'),CLEAR('D'),CLEAR('E'),
    ARMEMPTY()
  ]

  goal_state = [
    ON('B','D'),ON('D','C'), ON('C', 'A'),ON('A', 'E'),
    ONTABLE('A'),
    CLEAR('B'),CLEAR('C'), CLEAR('D'),CLEAR('E'),
    ARMEMPTY()
  ]

  goal_stack = GoalStackPlanner(initial_state=initial_state, goal_state
=goal_state)
  steps = goal_stack.get_steps()
  print("UNSTACK(E,B)")
  print("PUTDOWN(E)")
  for i in steps:
    print(i)
```

OUTPUT:

RESULT:

Implementation of Block World problem using Python is carried out successfully

Experiment – 11

Date:28/03/2022

Artificial Intelligence

Laboratory

Name :P.R.Samraysh
Class :B.Tech CSE IOT 'A'
Reg No. : RA1911032020009

AIM:

To Implement learning algorithms for an application in Python.

DESCRIPTION:

Here, we implement a Decision Tree to determine if a product has enough weight and is pushed to the right (R) or is of insufficient weight and is pushed to the left (L). The Dataset used is the Balance Scale Weight & Distance Database.

ALGORITHM:

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy the more the information content.

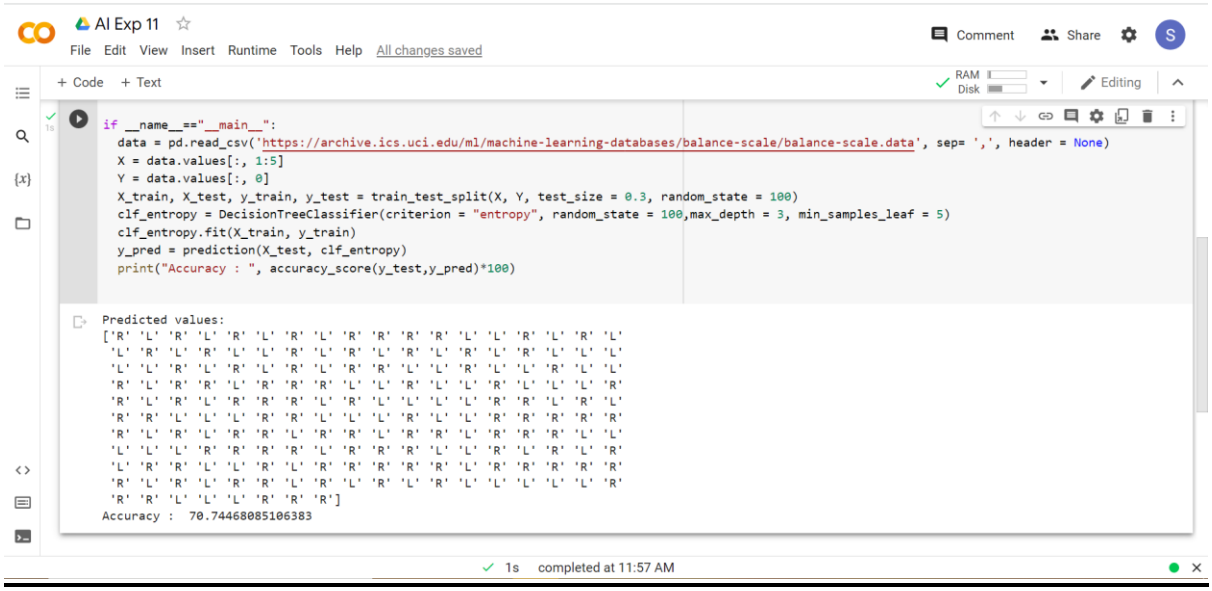
1. Start
2. Consider the whole training set as the root
3. Attributes are assumed to be categorical and are distributed recursively according to values using statistical methods
4. Find the best attribute and place it on the root node of the tree.
5. Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of the training dataset has the same value for an attribute. The entropy increases for each partition.
6. Find leaf nodes in all branches by repeating steps 1 and 2 on each subset
7. End

PROGRAM:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

def splitdataset(balance_data):
        X = balance_data.values[:, 1:5]
        Y = balance_data.values[:, 0]
        X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size = 0.3, random_state = 100)
        return X, Y, X_train, X_test, y_train, y_test
```

```
def prediction(X_test, clf_object):
        y_pred = clf_object.predict(X_test)
        print("Predicted values:")
        print(y_pred)
        return y_pred

if __name__=="__main__":
  data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data', sep= ',', header = None)
  X = data.values[:, 1:5]
  Y = data.values[:, 0]
  X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 100)
  clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,max_depth = 3, min_samples_leaf = 5)
  clf_entropy.fit(X_train, y_train)
  y_pred = prediction(X_test, clf_entropy)
  print("Accuracy : ", accuracy_score(y_test,y_pred)*100)
```

OUTPUT:



RESULT:

Thus, Implementation of learning algorithms using Python has been carried out successfully

Experiment – 12

Date:31/03/2022

Artificial Intelligence

Laboratory

Name :P.R.Samraysh
Class :B.Tech CSE IOT 'A'
Reg No. : RA1911032020009

AIM:

To Implement Development of ensemble model for an application in Python.

DESCRIPTION:

Ensembles can give you a boost in accuracy on your dataset. In this post you will discover how you can create some of the most powerful types of ensembles in Python using scikit-learn. This case study will step you through Boosting, Bagging and Majority Voting and show you how you can continue to ratchet up the accuracy of the models on your own datasets.

ALGORITHM:

Initialization:

1. Given training data from the instance space $S = \{(x_1, y_1), ..., (x_m, y_m)\}$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y} = \{-1, +1\}$.
2. Initialize the distribution $D_1(i) = \frac{1}{m}$.

Algorithm:

for $t = 1, ..., T$: **do**

Train a weak learner $h_t : \mathcal{X} \rightarrow \mathrm{R}$ using distribution D_t.

Determine weight α_t of h_t.

Update the distribution over the training set:

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

where Z_t is a normalization factor chosen so that D_{t+1} will be a distribution.

end for

Final score:

$f(x) = \sum_{t=0}^{T} \alpha_t h_t(x)$ and $H(x) = sign(f(x))$

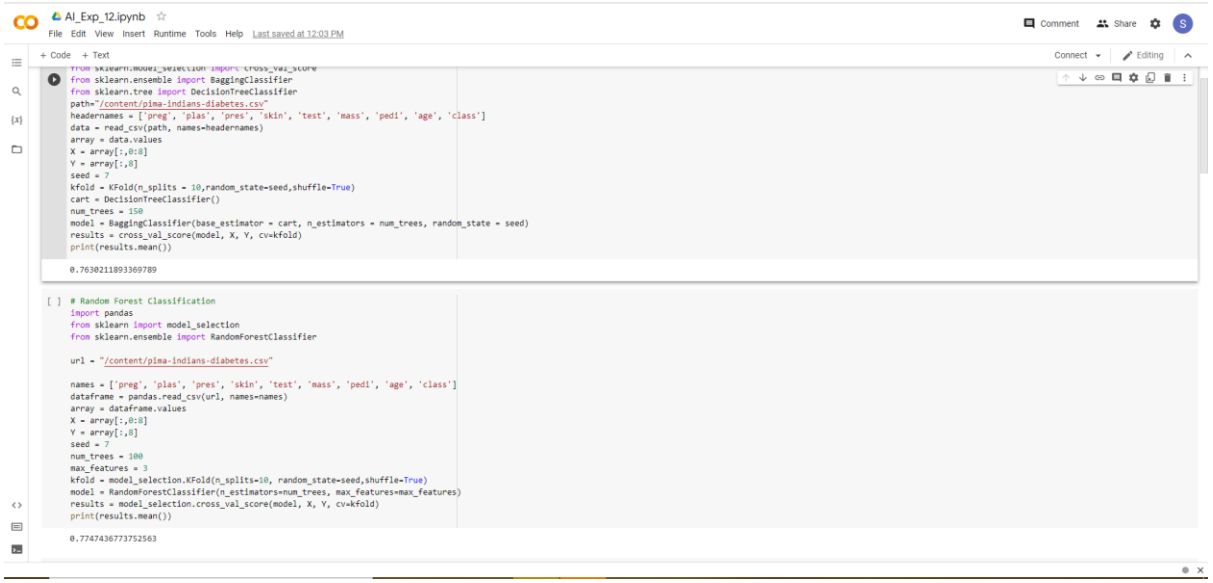
PROGRAM:

```
# Bagged Decision Trees for Classificationfrom inspect import Traceback
from inspect import Traceback
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
path="/content/pima-indians-diabetes.csv"
headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(path, names=headernames)
array = data.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = KFold(n_splits = 10,random_state=seed,shuffle=True)
cart = DecisionTreeClassifier()
num_trees = 150
model = BaggingClassifier(base_estimator = cart, n_estimators = num_trees,
random_state =
seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
# Random Forest Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
url = "/content/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
max_features = 3
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = RandomForestClassifier(n_estimators=num_trees,
max_features=max_features)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
# Extra Trees Classification
```

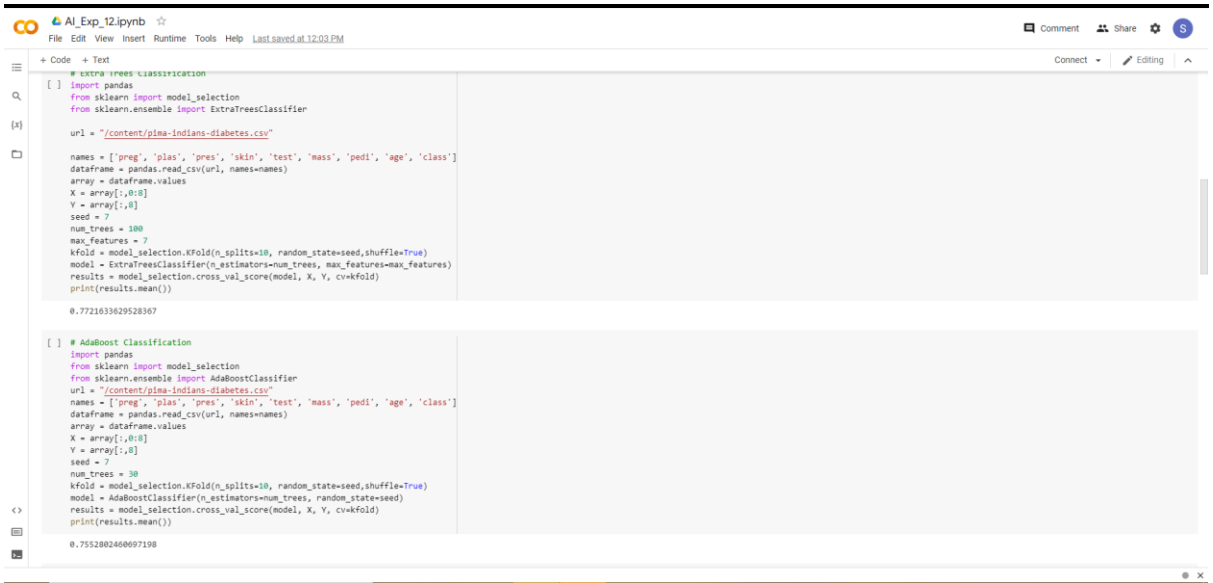
```
import pandas
from sklearn import model_selection
from sklearn.ensemble import ExtraTreesClassifier
url = "/content/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)

array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
max_features = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = ExtraTreesClassifier(n_estimators=num_trees,
max_features=max_features)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
# AdaBoost Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "/content/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
# Stochastic Gradient Boosting Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingClassifier
url = "/content/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
```

```
Y = array[:,8]
seed = 7
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = GradientBoostingClassifier(n_estimators=num_trees,
random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
# Voting Ensemble for Classification
import pandas
from sklearn import model_selection

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
url = "/content/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

OUTPUT:



```
AI_Exp_12.ipynb
File Edit View Insert Runtime Tools Help Last saved at 12:03 PM
Comment Share S
+ Code + Text
Connect Editing

from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
path="/content/pima-indians-diabetes.csv"
headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(path, names=headernames)
array = data.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = KFold(n_splits = 10,random_state=seed,shuffle=True)
cart = DecisionTreeClassifier()
num_trees = 150
model = BaggingClassifier(base_estimator = cart, n_estimators = num_trees, random_state = seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

0.7630211893369789

[ ] # Random Forest Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier

url = "/content/pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
max_features = 3
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

0.7747436773752563
```



```
AI_Exp_12.ipynb
File Edit View Insert Runtime Tools Help Last saved at 12:03 PM
Comment Share S
+ Code + Text
Connect Editing

[ ] # Extra Trees Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import ExtraTreesClassifier

url = "/content/pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
max_features = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_features)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

0.7721633629528367

[ ] # AdaBoost Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "/content/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

0.7552802460697198
```

```
[ ] # Stochastic Gradient Boosting Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingClassifier

url = "/content/pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

0.760492139439507 9
```

```python
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)

# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

RESULT:

Thus, Implementation of Implementation of Development of ensemble model for an application Python is carried out successfully

Experiment – 14

Date:08/04/2022

Artificial Intelligence

Laboratory

Name :P.R.Samraysh
Class :B.Tech CSE IOT 'A'
Reg No. : RA1911032020009

AIM:

To implement NLP programs using python.

ALGORITHM:

Text Preprocessing
Since, text is the most unstructured form of all the available data, various types of noise are present
in it and the data is not readily analyzable without any pre-processing. The entire process of cleaning
and standardization of text, making it noise-free and ready for analysis is known as text preprocessing.

It is predominantly comprised of three steps:

1. Noise Removal
2. Lexicon Normalization
3. Object Standardization

Noise Removal
Any piece of text which is not relevant to the context of the data and the end-output can be specified as the noise.

For example – language stop words (commonly used words of a language – is, am, the, of, in etc.), URLs or links, social media entities (mentions, hashtags), punctuations and industry specific words. This step deals with removal of all types of noisy entities present in the text.

A general approach for noise removal is to prepare a dictionary of noisy entities, and iterate the text object by tokens (or by words), eliminating those tokens which are present in the noise dictionary.

PROGRAM:

```
import pandas as pd
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('all')
```

```
# reading and wragling data
df_avatar = pd.read_csv('/avatar.csv',encoding = 'unicode_escape',
engine='python')
df_avatar_lines = df_avatar.groupby('character').count()
df_avatar_lines = df_avatar_lines.sort_values(by=['character_words'],
ascending=False)[:10]
top_character_names = df_avatar_lines.index.values

# filtering out non-top characters
df_character_sentiment =
df_avatar[df_avatar['character'].isin(top_character_names)]
df_character_sentiment = df_character_sentiment[['character',
'character_words']]

# calculating sentiment score
sid = SentimentIntensityAnalyzer()
df_character_sentiment.reset_index(inplace=True, drop=True)
df_character_sentiment[['neg', 'neu', 'pos', 'compound']] =
df_character_sentiment['character_words'].apply(sid.polarity_scores).apply(pd.
Series)
df_character_sentiment

# sentiment score for each character (mean)
df_character_sentiment =
df_character_sentiment.groupby('character').mean().round(3).sort_values('pos',
ascending=True)
df_character_sentiment.reset_index(inplace=True)
df_character_sentiment
#plotting Sentiment
import matplotlib.pyplot as plt
import numpy as np

# preparing data
X = np.arange(len(df_character_sentiment['pos']))
#bar plot
fig = plt.figure(figsize = (17, 12))
plt.barh(X, df_character_sentiment['pos'], facecolor='#9999ff',
edgecolor='white')
plt.barh(X, -df_character_sentiment['neg'], facecolor='#ff9999',
edgecolor='white')
# plt.rcParams.update({'font.size':13})
plt.xlim([-.16,.22])
plt.yticks(ticks=X, labels=df_character_sentiment['character'], rotation='0')
```

```
plt.show()

#Name entity Relation
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Biden invites Ukrainian president to White House this summer")
print([(X.text, X.label_) for X in doc.ents])

#Stemming
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
# PorterStemmer
porter = PorterStemmer()
# LancasterStemmer
lancaster = LancasterStemmer()
print(porter.stem("friendship"))
print(lancaster.stem("friendship"))

#Lemmatization
from nltk import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
words = ['articles', 'friendship', 'studies', 'phones']
for word in words:
   print(lemmatizer.lemmatize(word))

from nltk import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
words = ['be', 'is', 'are', 'were', 'was']
for word in words:
   print(lemmatizer.lemmatize(word, pos='v'))

#Bag of Words(BoW)
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

text = ["I love writing code in Python. I love Python code",
       "I hate writing code in Java. I hate Java code"]

df = pd.DataFrame({'review': ['review1', 'review2'], 'text':text})
cv = CountVectorizer(stop_words='english')
cv_matrix = cv.fit_transform(df['text'])
df_dtm = pd.DataFrame(cv_matrix.toarray(),
                      index=df['review'].values,
```

```
                columns=cv.get_feature_names())
df_dtm
```

#TD-IDF

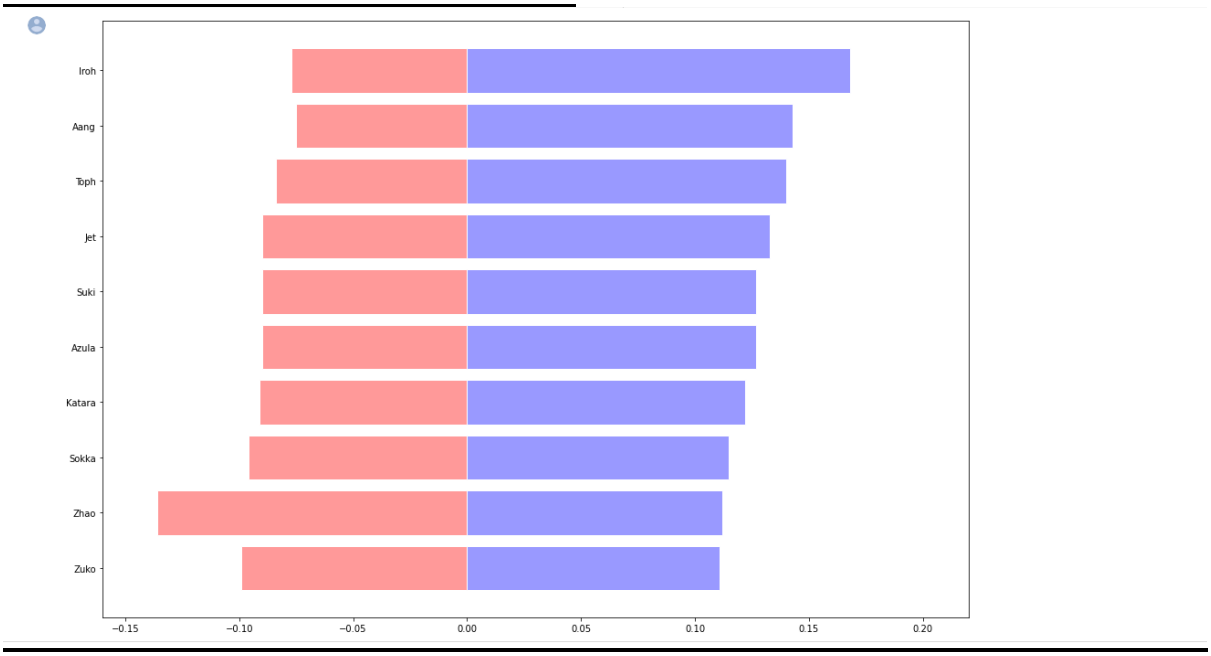
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]

df = pd.DataFrame({'review': ['review1', 'review2'], 'text':text})
tfidf = TfidfVectorizer(stop_words='english', norm=None)
tfidf_matrix = tfidf.fit_transform(df['text'])
df_dtm = pd.DataFrame(tfidf_matrix.toarray(),
                index=df['review'].values,
                columns=tfidf.get_feature_names())
df_dtm
```

OUTPUT:

	character	character_words	neg	neu	pos	compound
0	Katara	Water. Earth. Fire. Air. My grandmother used t...	0.130	0.804	0.066	-0.6874
1	Sokka	It's not getting away from me this time. Watc...	0.000	1.000	0.000	0.0000
2	Katara	Sokka, look!	0.000	1.000	0.000	0.0000
3	Sokka	Sshh! Katara, you're going to scare it away. ...	0.200	0.800	0.000	-0.5411
4	Katara	But, Sokka! I caught one!	0.000	1.000	0.000	0.0000
...	...	...	...	...	...	...
7053	Zuko	At least you don't look like a boar-q-pine! My...	0.183	0.817	0.000	-0.4007
7054	Suki	And why did you paint me firebending?	0.000	1.000	0.000	0.0000
7055	Sokka	I thought it looked more exciting that way. O...	0.000	0.687	0.313	0.7501
7056	Iroh	Hey, my belly's not that big anymore. I've rea...	0.000	1.000	0.000	0.0000
7057	Toph	Well I think you all look perfect!	0.000	0.396	0.604	0.7263

7058 rows × 6 columns

	character	neg	neu	pos	compound
0	Zuko	0.099	0.789	0.111	0.026
1	Zhao	0.136	0.752	0.112	-0.051
2	Sokka	0.096	0.789	0.115	0.024
3	Katara	0.091	0.787	0.122	0.046
4	Azula	0.090	0.783	0.127	0.080
5	Suki	0.090	0.783	0.127	0.089
6	Jet	0.090	0.777	0.133	0.063
7	Toph	0.084	0.776	0.140	0.078
8	Aang	0.075	0.782	0.143	0.084
9	Iroh	0.077	0.755	0.168	0.163



```
#Name entity Relation
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Biden invites Ukrainian president to White House this summer")
print([(X.text, X.label_) for X in doc.ents])
```

```
[('Ukrainian', 'NORP'), ('White House', 'ORG'), ('this summer', 'DATE')]
```

```
#Stemming
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
# PorterStemmer
porter = PorterStemmer()
# LancasterStemmer
lancaster = LancasterStemmer()
print(porter.stem("friendship"))
print(lancaster.stem("friendship"))
```

```
friendship
friend
```

```
#Lemmatization
from nltk import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
words = ['articles', 'friendship', 'studies', 'phones']
for word in words:
    print(lemmatizer.lemmatize(word))
```

```
article
friendship
study
phone
```

```
from nltk import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
words = ['be', 'is', 'are', 'were', 'was']
for word in words:
    print(lemmatizer.lemmatize(word, pos='v'))
```

```
be
be
be
be
be
```

```
#Bag of Words(BOW)
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]

df = pd.DataFrame({'review': ['review1', 'review2'], 'text':text})
cv = CountVectorizer(stop_words='english')
cv_matrix = cv.fit_transform(df['text'])
df_dtm = pd.DataFrame(cv_matrix.toarray(),
                      index=df['review'].values,
                      columns=cv.get_feature_names())
df_dtm
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	code	hate	java	love	python	writing
review1	2	0	0	2	2	1
review2	2	2	2	0	0	1

```
#TD-IDF
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]

df = pd.DataFrame({'review': ['review1', 'review2'], 'text':text})
tfidf = TfidfVectorizer(stop_words='english', norm=None)
tfidf_matrix = tfidf.fit_transform(df['text'])
df_dtm = pd.DataFrame(tfidf_matrix.toarray(),
                      index=df['review'].values,
                      columns=tfidf.get_feature_names())
df_dtm
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	code	hate	java	love	python	writing
review1	2.0	0.00000	0.00000	2.81093	2.81093	1.0
review2	2.0	2.81093	2.81093	0.00000	0.00000	1.0

RESULT:

Thus, Implementation of NLP using Python is carried out successfully.

Experiment – 15

Date:08/04/2022

Artificial Intelligence

Laboratory

Name :P.R.Samraysh
Class :B.Tech CSE IOT 'A'
Reg No. : RA1911032020009

AIM:

To Create a Program to apply Deep Learning method for Automatic Handwriting Recognition in Python

ALGORITHM:

1. Start Program.
2. Load the data.
3. Train the program.
4. Select random data from the csv.
5. Predict which letter the selected data is.
6. Check the accuracy.
7. Print the result.
8. End Program.

PROGRAM:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import matplotlib.pyplot as plt
import cv2
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.utils import to_categorical
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

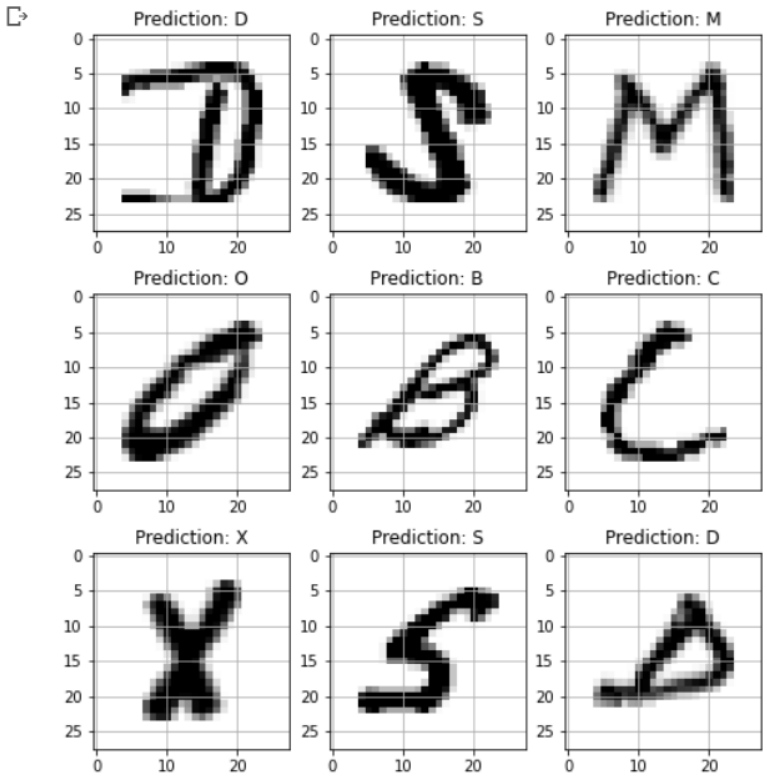
```
from sklearn.utils import shuffle

from google.colab import drive
drive.mount('/drive')
data = pd.read_csv('/drive/My Drive/A_Z Handwritten
Data.csv').astype('float32')

X = data.drop('0',axis = 1) # axis=1 for dropping column
y = data['0']

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.2)

X_train = np.reshape(X_train.values, (X_train.shape[0], 28,28))
X_test = np.reshape(X_test.values, (X_test.shape[0], 28,28)) #0=> B&W,
28x28 matrix

print("Train data shape: ", X_test.shape)
print("Test data shape: ", X_test.shape)

word_dict =
{0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14
:'O',15:'P',16:'Q',17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X', 24:'Y',25:'Z'}

y_int = np.int0(y)
count = np.zeros(26, dtype='int') #a vector of size 26 with all 0 values
for i in y_int:
    count[i] +=1 #total count of each alphabet

alphabets = []
for i in word_dict.values():
    alphabets.append(i) #all alphabets

fig, ax = plt.subplots(1,1, figsize=(10,10))
ax.barh(alphabets, count)

plt.xlabel("Number of elements ")
plt.ylabel("Alphabets")
plt.grid()
plt.show()

shuff = shuffle(X_train[:100])

fig, ax = plt.subplots(3,3, figsize = (10,10))
```

```
axes = ax.flatten()

for i in range(9):
    _, shu = cv2.threshold(shuff[i], 30, 200, cv2.THRESH_BINARY)
    axes[i].imshow(np.reshape(shuff[i], (28,28)), cmap="Greys")
plt.show()

X_train =
X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1) #RGB
=>Channel of 1
print("New shape of train data: ", X_train.shape)

X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2],1)
#RGB =>Channel of 1
print("New shape of train data: ", X_test.shape)

train_yOHE = to_categorical(Y_train, num_classes = 26, dtype='int')
print("New shape of train labels: ", train_yOHE.shape)

test_yOHE = to_categorical(Y_test, num_classes = 26, dtype='int')
print("New shape of test labels: ", test_yOHE.shape)

model = Sequential()
#CNN
# input -> conv -> maxpool -> conv -> maxpool .....->flattened vector->
#.                    hidden layer -> hidden layer -> softmax layer
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding =
'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding =
'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))
```

```
model.add(Dense(26,activation ="softmax"))

print("The validation accuracy is :", history.history['val_accuracy'])
print("The training accuracy is :", history.history['accuracy'])
print("The validation loss is :", history.history['val_loss'])
print("The training loss is :", history.history['loss'])

fig, axes = plt.subplots(3,3, figsize=(8,9))
axes = axes.flatten()

for i,ax in enumerate(axes):
    img = np.reshape(X_test[i], (28,28))
    ax.imshow(img, cmap="Greys")

    pred = word_dict[np.argmax(test_yOHE[i])]
    ax.set_title("Prediction: "+pred)
    ax.grid()

fig, axes = plt.subplots(3,3, figsize=(8,9))
axes = axes.flatten()

for i,ax in enumerate(axes):
    img = np.reshape(X_test[i], (28,28)) # reshaping it for displaying
    ax.imshow(img, cmap="Greys")
    img_final =np.reshape(img, (1,28,28,1)) # reshapng it for passing into model
for prediction
    pred = word_dict[np.argmax(model.predict(img_final))]
    ax.set_title("Prediction: "+pred)
    ax.grid()
```

OUTPUT:

```
[ ] print("The validation accuracy is :", history.history['val_accuracy'])
    print("The training accuracy is :", history.history['accuracy'])
    print("The validation loss is :", history.history['val_loss'])
    print("The training loss is :", history.history['loss'])

    The validation accuracy is : [0.9748019576072693]
    The training accuracy is : [0.958329975605011]
    The validation loss is : [0.09028583765029907]
    The training loss is : [0.15422528982162476]
```



RESULT:

Thus, A Program to apply Deep Learning method for Automatic Handwriting Recognition using Python is created and executed successfully.