NAME: M. Ananta Naga Rajesh (1BM21CS098)

DIV: ROLL NO.: YOUVA

STD:

SUBJECT:

INDEX

SR. NO.	DATE	TITLE	PAGE NO.	TEACHER'S SIGN
1.	17/11/23	Tic-Tac-Toe Game		
2.	24/11/23	8 puzzle problem (A*)		
3.	8/12/23	8 puzzle problem (BFS)		
4.	8/12/23	8 puzzle problem (IDDFS)		
5.	22/12/23	Vaccuum Cleaner		
6.	28/12/23	Knowledge Base Entailment.		
7.	28/12/23	Knowledge Base Resolution.		
8	2/1/24	Unification.		
9.	12/1/24	FOL to CNF.		
10	23/1/24	Forward Chaining		

10/11/2023.

Artificial Intelligence Lab.

Program 1:

Write a python program to check the age criteria using else-if ladder.

```
age = int (input ("Enter your age:"))
if age<0:
    print ("Enter a valid age")
elif age<12:
    print ("you are a kid")
elif age>=13 and age<=25:
    print ("you are teenager")
elif age>25 and age<70:
    print ("you are adult")
else:
    print ("you are old")
```

Output:

```
Enter your age: 56
you are adult
Enter your age: -9
Enter a valid age
Enter your age: 100
you are old.
```

Program 2:

Write a python program to print multiplication table

```
n = int(input("Enter the number : "))
for i in range(1,11):
    print(n, "*", i, "=", n*i)
```

Output:

Enter the number: 8

8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80

Program 3:

Write a program to print the following patterns:

1
2 2
3 3 3
4 4 4

upto n digits

```
n = int (input (" Enter the number :"))
for i in range (1, n+1):
    for j in range (1, i+1):
        print (i, end =" ")
    print ("\n")
```

Output :

Enter the number : 5

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

b).

```
1
1 2
1 2 3
1 2 3 4
```

upto n digits.

```
n = int (input (" Enter the number :"))
for i in range (1, n+1):
    for j in range (1, i+1):
        print (j, end =" ")
    print ("\n")
```

Output :

Enter the number : 5

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Program 4:

Reverse the digits of the number.

```
n = int(input("Enter the number : "))
rev = 0
while n > 0:
    rem = n % 10
    rev = rev * 10 + rem
    n = n // 10
print(rev)
```

Output:

Enter the number: 5678

8765

Program 5:

Bubble Sort.

```
def bubbleSort(arr):
    n = len(arr)
    for i in range(0, n-1):
        for j in range(0, n-1-i):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

```
n = int(input("Enter the number of elements: "))
arr = []
for i in range(0, n):
    m = int(input())
    arr.append(m)
bubbleSort(arr)
print("Sorted array")
print(arr)
```

Output:

```
Enter the number of elements: 5
78
76
0
45
100
Sorted array
[0, 45, 76, 78, 100]
```

17/11/2023

Tic-Tac-Toe Game

```
board = [' ' for x in range(10)]

def insertLetter(letter, pos):
    board[pos] = letter

def spaceIsFree(pos):
    return board[pos] == ' '

def printBoard(board):
    print('   |   |')
    print(' ' + board[1] + ' | ' + board[2] + ' | ' + board[3])
    print('   |   |')
    print('-----------')
    print(' ' + board[4] + ' | ' + board[5] + ' | ' + board[6])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[7] + ' | ' + board[8] + ' | ' + board[9])
    print('   |   |')

def isWinner(bo, le):
    return (bo[7] == le and bo[8] == le and
            bo[9] == le) or (bo[4] == le and
            bo[5] == le and bo[6] == le) or
```

```
(bo[1] == le and bo[2] == le and bo[3] == le) or
(bo[1] == le and bo[4] == le and bo[7] == le) or
(bo[2] == le and bo[5] == le and bo[8] == le) or
(bo[3] == le and bo[6] == le and bo[9] == le) or
(bo[1] == le and bo[5] == le and bo[9] == le) or
(bo[3] == le and bo[5] == le and bo[7] == le)


def playerMove():
    run = True
    while run:
        move = input("Select a position to place")
        move = int(move)
        if move > 0 and move < 10:
            if spaceIsFree(move):
                run = False
                insertLetter('X', move)
            else: print('Space is occupied')
        else:
            print('Press a no. within range')

def compMove():
    possibleMoves = [x for x, letter in enumerate(board)
                     if letter == ' ' and x != 0]

    move = 0
```

```
for let in ['O','X']:
    for i in possibleMoves:
        boardCopy = board[:]
        boardCopy[i] = let
        if isWinner(boardCopy, let):
            move = i
            return move

cornersOpen = []
for i in possibleMoves:
    if i in [1,3,7,9]:
        cornersOpen.append(i)
if len(cornersOpen) > 0:
    move = selectRandom(cornersOpen)
    return move
if 5 in possibleMoves:
    move = 5
    return move

edgesOpen = []
for i in possibleMoves:
    if i in [2,4,6,8]:
        edgesOpen.append(i)
if len(edgesOpen) > 0
    move = selectRandom(edgesOpen)
return move
```

```
def selectRandom(li):
    import Random
    ln = len(li)
    r = random.randrange(0, ln)
    return li[r]

def isBoardFull(board):
    if board.count(' ') > 1:
        return False
    else:
        return True

def main():
    print('Welcome')
    printBoard(board)
    while not (isBoardFull(board)):
        if not (isWinner(board, 'O')):
            playerMove()
            printBoard(board)
        else:
            print("'O' won the game")
            break
        if not (isWinner(board, 'X')):
            move = compMove()
            if move == 0:
                print('Tie')
```

```
    else:
        insertLetter('O', move)
        print('Computer placed @ in position', move)
        printBoard(board)
    else:
        print('X/s' won this game')
        break
if isBoardFull(board):
    print('Tie Game')
while True:
    answer = input('Do you want to play again?')
    if answer.lower()=='y' or answer.lower=='yes':
        board = [' ' for x in range(10)]
        print('-----')
        main()
    else:
        break
```

Algorithm:

Player Move:

1. Set run to 'True' and enter the loop
2. While run is True:
   - Take move input from user
   - If move is in range of 1 to 10, check if the space is free.

- if the space is free, 'x' is inserted.
- if the space is not free, player is asked to enter another number.

- The loop continues until run='True'

Computer Move:

1. Create a list of possible move, consisting of empty spaces in the board.
2. For each move in computer:
   - For each possible move, create a board copy and check if it is a winning move,
   - if there is no possible moves:
     - Check if empty corners are there and generate a random no. among them.
     - Check if '5' position is free. Place 'O'.
     - Else, Check for remaining empty positions and generate a random no to place 'O'.

24-11-23

24/11/2023.

8-puzzle game.

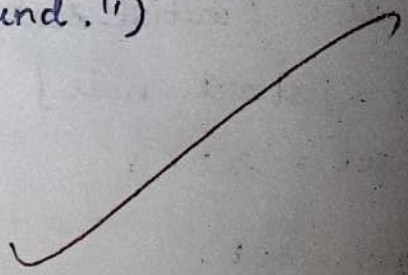
```
import heapq

def get_blank_position(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

def get_neighbours(state, move):
    row, col = get_blank_position(state)
    new_row, new_col = row + move[0], col + move[1]
    if 0 <= new_row < 3 and 0 <= new_col < 3:
        new_state = [list(row) for row in state]
        new_state[row][col], new_state[new_row][new_col] =
        raw_state[new_row][new_col], new_state[row][col]
        return new_state
    return None

def is_goal(state):
    goal_state = [[1,2,3], [4,5,6], [7,8,0]]
    return state == goal_state
```

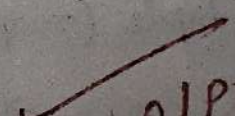
```
def heuristics(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                goal_row, goal_col = divmod(state[i][j]-1, 3)
                distance += abs(i-goal_row) + abs(j-goal_col)
    return distance

def solve_puzzle(initial_state):
    start_node = (initial_state, None, None, 0, heuristic(initial_state))
    open_set = [start_node]
    closed_set = set()

    while open_set:
        current_state, parent, move, cost, _ =
                heapq.heappop(open_set)

        if is_goal(current_state):
            path = []
            while parent:
                path.insert(0, (move, current_state))
                current_state, parent, move, _, _ = parent
            return path

        closed_set.add(tuple(map(tuple, current_state)))

        moves = [(0,1), (1,0), (0,-1), (-1,0)]
        for move in moves:
            neighbor_state = get_neighbors(current_state, move)
            if neighbor_state and tuple(map(tuple,
                neighbor_state)) not in closed_set:
                heapq.heappush(open_set, (neighbor_state,
                (current_state, parent, move, cost+1, heuristic(n
```

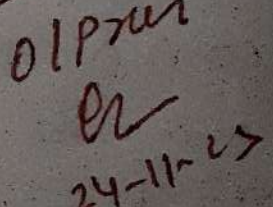
```
initial_state = [[1,2,3], [4,5,6], [0,7,8]]
solution_path = solve_puzzle (initial_state)

if solution_path:
    for move, state in solution_path:
        print ("Move:", move)
        for row in state:
            print (row)
        print ("------")
else:
    print ("No solution found.")
```

Output:

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

1 2 3
0 5 6
4 7 8

1 2 3
4 5 6
7 0 8

0 2 3
1 5 6
4 7 8

1 2 3
5 0 6
4 7 8

1 2 3
4 0 6
7 5 8

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







Algorithm:

1. Create a start node with the initial state, no parent, no move, cost of 0, and the heuristic value.
2. Create an open set and add the start node to it.
3. While set not empty:
   pop the node with the lowest total cost from the open set.
   if the current state is goal state, reconstruct and return the solution path.
4. Get neighboring states by making valid moves from current state.
   for i in neighbours:
      if neighbour in goal state:
         return solution path.
      else:
         add neighbour state to the closed set
   Calculate total cost.
   add the neighbor to open set.
5. If open set becomes empty, no solution is found.

24-11-25

08/12/2023.

8 puzzle problem using BFS.

```
import numpy as np
import pandas as pd
import os

def bfs(src, target):
    queue = []
    queue.append(src)
    exp = []
    while len(queue) > 0:
        source = queue.pop(0)
        exp.append(source)
        print(source)
        if source == target:
            print("success")
            return
        poss_moves_to_do = []
        poss_moves_to_do = possible_moves(source, exp)
        for move in poss_moves_to_do:
            if move not in exp and move not in queue:
                queue.append(move)

def possible_moves(state, visited_states):
    b = state.index(0)
    d = []
```

```
if b not in [0,1,2]:
    d.append('u')
if b not in [6,7,8]:
    d.append('d')
if b not in [0,3,6]:
    d.append('l')
if b not in [2,5,8]:
    d.append('r')
pos_moves_it_can = []
for i in d:
    pos_moves_it_can.append(gen(state, i, b))
return [move_it_can for move_it_can in
pos_moves_it_can if move_it_can not in
visited_states]

def gen(state, m, b):
temp = state.copy()
if m=='d':
  temp[b+3], temp[b] = temp[b], temp[b+3]
if m=='u':
  temp[b-3], temp[b] = temp[b], temp[b-3]
if m=='l':
  temp[b-1], temp[b] = temp[b], temp[b-1]
if m=='r':
  temp[b+1], temp[b] = temp[b], temp[b+1]
return temp
```

08/12/2023.

8 puzzle problem using DFS.

```
def id_dfs(puzzle, goal, get_moves)
    import itertools

def dfs( route, depth):
    if depth == 0:
        return
    if route[-1] == goal :
        return route.

    for moves in get_moves (route[-1]):
        if move not in route:
            next_route = dfs (route + [move], depth - 1)
        if next_route :
            return next_route

    for depth in itertools.count();
        route = dfs ( [puzzle], depth)
        if route:
            return route

def possible_moves (state):
    b = state.index(0)
    d = []
    if b not in [0,1,2]:
        d.append ('u')
    if b not in [6,7,8]:
        d.append ('d')
```

```
if b not in [0,3,6]:
    d.append('l')
if b not in [2,5,8]:
    d.append('r')
pos_moves = []
for ia in d:
    pos_moves.append(generate(state,i,b))
return pos_moves.

def generate(state,m,b):
    temp = state.copy()
    if m=='d':
        temp[b+3], temp[b] = temp[b], temp[b+3]
    if m=='u':
        temp[b-3], temp[b] = temp[b], temp[b-3]
    if m=='l':
        temp[b-1], temp[b] = temp[b], temp[b-1]
    if m=='r'
        temp[b+1], temp[b] = temp[b], temp[b+1]
return temp.
initial = [1,2,3,0,4,6,
```

22/12/2023

Vaccuum Cleaner.

```
def clean_room(floor, room_row, room_col):
    if floor[room_row][room_col] == 1:
        print("Cleaning room at ({room_row+1}, {room_col+1}) (Room was dirty)")
        floor[room_row][room_col] = 0
        print("Room is clean now")
    else:
        print("Room at ({room_row+1}, {room_col+1}) is already clean")

def main():
    rows = 2
    cols = 2
    floor = [[0,0], [0,0]]

    for i in range(rows):
        for j in range(cols):
            status = int(input(f"Enter clean status for room at ({i+1}, {j+1}) (1 for dirty, 0 for clean): "))
            floor[i][j] = status.
```

```
    for i in range(rows):
        for j in range(cols):
            clean_room(floor, i, j)

    print("Returning to Room at (1,1) to check
     if it has become dirty again:")
    clean_room(floor, 0, 0)

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

Output:

```
Enter clean status for room 1 : 1
Enter the clean status for room 2: 0.
[('Room 1', 1), ('Room 2', 0)]
Cleaning Room 1 (Room was dirty)
Room 1 is now clean.
Room 2 is already clean.
Returning to room 1 to check if it has
become dirty again..
Room 1 is already clean.
Returning to room 1 to check if it has
become dirty again.
Room 1 is clean after checking.
```

29/12/23.

Knowledge base entailment.

Entailment refers to the logical relationship between a KB (a set of statement or rules) and a query.

If KB entails a statement, it means that whenever the statements in the KB are true, the given query must also be true.

$KB \models Q$.

Knowledge base resolution.

The resolution rule involves taking two clauses that contain complementary literals and resolving them to produce a new clause.

Knowledge base entailment.

- If its raining (P) then ground is wet.
- If the ground is wet (Q), then the plants will grow (R).
- It's not the case that plants will grow (~R).

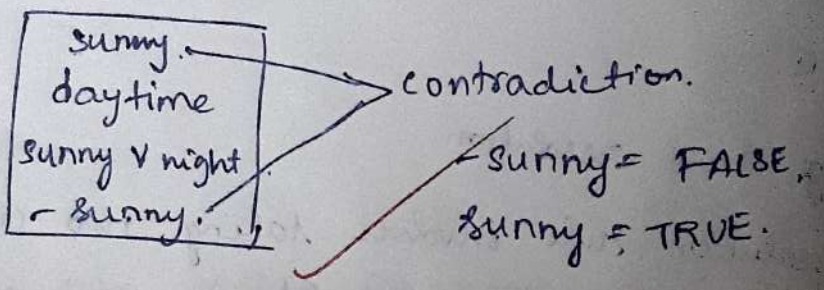
Query: Whether it is raining.

NOT R and R are complement of each other.
The ground is wet.
which means that if is raining, because the ground is wet. 'P' entails query.

Knowledge base Resolution.

Input a KB and an expression, negate the expression add it to KB and find a contradiction, if contradiction is found, the negated statement is false hence the Original statement is true.

Is it SUNNY? Sunny = TRUE ? Prove Sunny



Proved

Code:

```
from sympy import. symbols, And, Not,
Implies, Satisfiable

def create_knowledge_base():
    p = symbols('p')
    q = symbols ('q')
    r = symbols ('r')

    knowledge_base = And (
        Implies (p, q),
        Implies (q, r),
    ) Not(r)
```

```
    return knowledge_base

def query_entails (knowledge_base, query):
    entailment = satisfiable (And (knowledge_base,
                                   NOT(query)))
    return not entailment

if __name__ == "__main__":
    KB = create_knowledge_base ()
    query = symbols ('p')
    result = query_entails (KB, query)

    print ("Knowledge Base", KB)
    print ("Query:", query)
    print ("Query entails knowledge base:", result)
```

Output:

Knowledge Base: ~r & (Implies (p, q)) &
(Implies (q, r))

Query: p

Query entails knowledge Base: False.

Knowledge base Resolution.

Code:

```
def negate_literal(literal):
    if literal[0] == '~':
        return literal[1:]
    else:
        return '~' + literal

def resolve(c1, c2):
    resolved_clause = set(c1) | set(c2)
    for literal in c1:
        if negate_literal(literal) in c2:
            resolved_clause.remove(literal)
            resolved_clause.remove(negate_literal
                                   (literal))
    return tuple(resolved_clause)

def resolution(knowledge_base):
    while True:
        new_clauses = set()
    for i, c1 in enumerate(knowledge_base):
        for j, c2 in enumerate(knowledge_base):
```

```
if i != j:
    new_clause = resolve(c1, c2)
    if len(new_clause) > 0 and
        new clause not in knowledge base
        new_clauses.add(new_clause)

if not new_clauses:
    break.
knowledge_base |= new_clauses

return knowledge_base

if __name__ == "__main__":
    kb = {('p', 'q'), ('~p', 'r'), ('~q', '~r')}
    result = resolution(kb)
    print("Original Knowledge Base :", kb)
    print("Resolved Knowledge Base :", result)
```

12/01/2024

Unification.

e.g; Knows(John,x) Knows(John, Jane).

{x/Jane}.

Step 1: If term 1 or term 2 is variable / constant, then:

a) term 1 or term 2 are identical.
return NIL.

b) else if term1 is available
if term1 occurs is term 2
return FAIL.
else
return {(term2/term1)}.

c) else if term2 is available
if term2 occurs in term1
return FAIL
else
return {(term1/term2)}.

d) else return FAIL.

Step 2: If Predicate (term1) ≠ predicate (term2)
return FAIL.

Step 3: number of arguments not equal.
return FAIL.

Step 4: Set (SUBST) to NIL.

Step 5: For i=1 to the number of elements in term 1

a) call UNIFY (ith term1, ith term2).

put result into S

b) S = FAIL.

return FAIL.

c) If S ≠ NIL.

a) Apply S to the remainder of both L1 & L2

b) SUBST = APPEND (S, SUBST).

Step 6: Return SUBST.

12/01/2024.

FOL to CNF.

```
import re

def fol_to_cnf(fol):
    statement = fol.replace("<=>", "_")
    while '_' in statement:
        i = statement.index('_')
        new_statement = '[' + statement[:i] +
        '=>' + statement[i+1:] + ']' & '[' +
        statement[i+1:] + '=>' + statement[:i] + ']'
        statement = new_statement
    statement = statement.replace("=>", "-")
    expr = '\[([^]]+)\]'
    statements = re.findall(expr, statement)
    for i, s in enumerate(statements):
        if '[' in s and ']' not in s:
            statements[i] += ']'
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    while '-' in statement:
        i = statement.index('-')
        br = statement.index('-')
```

```
new_statement = '~' + statement[br:i] + ')' +
                statement[i+1:]
statement = statement[:br] + new_statement if
  br > 0 else new_statement
while '~∀' in statement:
  i = statement.index('~∀')
  statement = list(statement)
  statement[i], statement[i+1], statement[i+2] = '∃',
  statement[i+2], '~'
  statement = ''.join(statement)
while '-∃' in statement:
  i = statement.index('~∃')
  s = list(statement)
  s[i], s[i+1], s[i+2] = '∀', s[i+2], '~'
  statement = ''.join(s)
Statement = statement.replace('~[∀', '[~∀')
Statement = statement.replace('-[∃', '[~∃')
expr = '(~[∀|∃].)'
Statements = re.findall(expr, statement)
for s in statements:
  statement = statement.replace(s, fol_to_cnf(s))
expr = '~\[[^]]+]'
for s in statements:
  statement = statement.replace(s, DeMorgan(s))
return Statement
```

print (skolemization (fol-to-cnf (" animal (y) <=> loves (x, y)")

print (fol-to-cnf (" [american(x) & weapon(y) & sells (x, y, z)] => criminal (x)"))

Output:

[~animal (y) | loves (x,y)] & [~loves (x, y) | animal (y)]

[~ american (x) | ~ weapon (y) | ~ sells (x, y z) | criminal (x

Animal => lover, A. lover => A

19/1/24

23/01/2024

Forward chaining.

```
import re

def isVariable(x):
    return len(x)==1 and x.islower() and
           x.isalpha()

def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return matches

def getPredicates(string):
    expr = '([a-zA]+)\([^&|]+\)'
    return re.findall(expr, string)

class Fact:
    def __init__(self, expression):
        self.expression = expression
        predicate, params = self.splitExpression
                            (expression)
        self.predicate = predicate
        self.params = params
        self.result = any(self.getConstants())

    def splitExpression(self, expression):
        predicate = getPredicates(expression)[0]
        params = getAttributes(expression)[0].
                 strip('()').split(',')
```

```
        return [predicate, params]

    def getResult(self):
        return self.result

    def getConstants(self):
        return [None if isVariable(c) else c for
                c in self.params]

    def getVariables(self):
        return [v if isVariable(v) else None for
                v in self.params]

    def substitute(self, constants):
        c = constants.copy()
        f = f"{self.predicate}({','.join([constants.
        pop(0) if isVariable(p) else p for p in
        self.params])})"
        return Fact(f)

class Implication:
    def __init__(self, expression):
        self.expression = expression
        l = expression.split('=>')
        self.lhs = [Fact(f) for f in l[0].
                    split('&')]
        self.rhs = Fact(l[1])
```

```
def evaluate(self, facts):
    constants = {}
    new_lhs = []
    for fact in facts:
        for val in self.lhs:
            if val.predicate == fact.predicate:
                for i,v in enumerate(val.getVariables()):
                    if v:
                        constants[v] = fact.getConstants()[i]
                        new_lhs.append(fact)
                        predicate, attributes = getPredicates
                        (self.rhs.expression[0], str(
                            getAttributes(self.rhs.expression)
                            [0])
class KB:
    def __init__(self):
        self.facts = set()
        self.implications = set()
    def tell(self, e):
        if '=>' in e:
            self.implications.add(
```

Output:

```
Querying evil(x)
1) evil(Richard)
2) evil(John)
```

O/P

24/1/24