INDEX

Name Nachiketha

Standard CSE Section B Roll No. 1BM21CS109

Subject AI Lab.

Sl.No.	Date	Title	Page No	Teacher's Sign
1.				
2.	10-11-23	Python Programming	1-4	
3	17-11-23	Tic-tac-toe	5-	
4	24-11-23	Puzzle-8 using BFS		
5	~~15-12-23~~	Puzzle-8 using IDDFS		
6	8-12-23	Puzzle-8 using A*		
7	22-12-23	Vaccum Cleaner simulation		
8.		Knowledge based entailment		
9.		Resolution, Knowledge based		
10		Unification		
11		FOL to CNF		
12		Moving Query castle Forward Chaining		

Date :
Page No : 1

LAB-2

1. Check the age criteria :

```
def Criteria(age1):
    age = int(age1)
    if (age < 13):
        return print("kid")
    elif(age <20):
        return "teen"
    elif(age <60):
        return "adult"
    else:
        return "old"

age = input("Enter the age")
print("Person is", Criteria(age))
```

Output :

```
Enter the age: 5
Person is kid
```

2. Multiplication Table for given num:

```
def m_table(k):
    n2 int(k)
    for i in range(1,11):
        print(n, "x", i, "=", n*i)

k= input("Enter the Number: ")
m_table(k)
```

Output: Enter the number : 5

```
5x1 =5      5x6 = 30
5x2 =10     5x7 =35
5x3 =15     5x8 =40
5x4 =20     5x9 =45
5x5 =25     5x10 =50
```

3. Program to print pattern:

```
def Pattern(m):
    k = int(m)+1
    for i in range(1,k)
        for j in range(k-i):
            print(" ", end="")
        for n in range(i):
            print(i= end=" ")
        print("\n")

k = input("Enter the number between 1 to 10:\n")
Pattern(k)
```

output:

Enter the number between 1 to 10. : 6

```
     1
    2 2
   3 3 3
  4 4 4 4
 5 5 5 5 5
6 6 6 6 6 6
```


Write a program to reverse a number.

```
num = int(input("Enter the number\n"))
reversed num = 0

while num!=0:
    digit = num % 10
    reversed_num *= 10 +digit
    num //=10.

print("Reversed Number:" + str(reversed_num
```

Output:

Enter the number
256
Reversed Number: 652

Write a program sort list using bubble sort.

```
def bubble_sort(list_item):
    for i in range(len(list_item)):
        for j in range(i+1, len(list_item
            if (list_item[i] > list_item[j]):
                temp = list_item[i]
                list_item[j] = list_item[j]
                list_item[j] = temp

    return list_item
```

```
k = (input ("Enter the size:")
list_item[]
print ("Enter the elements:")
for j in range (int(k)):
        list_item.append (int (input()))
list_item = bubble_sort (list_item)
print (list_item)
```

Output:

```
Enter the size : 3.
Enter the elements:
5
3
8
[3, 5, 8]
```

LAB-3

Write a tic-tac-toe game to simulate the

```
list = [][]
for i in range(3):
    for j in range(3):
        list[i][j] = [illegible] ; ++j+1

def create_board (list):
    for i in range(3):
        for j in range (3):
            print(list[i][j])
            print(" | ")
        print(' ---------- \n")


def check_winner(list):
    if(list[0]==list[1] && list[1]
    for i in range (2):
        if ( list[i][0] == list[i][1] && list[i][1]
            == list[i][2]):
            return True
    elif (list[0][i] == list[1][i] && list[1][i]
            == list[2][i]):
            return True
    else:
        for j in range (2):
            X = list[0][0]
            count = 0, count1 = 0
            if (i == j && list[i][j] == X ):
                count ++
    if (count == [illegible]
        return True
```

```
if (i+j==2 && list[i][j]==Y)
    count1++
if(count==3 || count1==3)
    return True

def play(k,i)
    n= k/3 -1
    if( list[n][k%3] =='x' || list[n][k%3] =='0'):
        print ("player",i," has already played this position")
    else:
        if (i==1):
            list[n][k%3] = 'x'
        else
            list[n][k%3] = '0'

count=0
while (1)
    create_board (list).
    print ("Player", count%2 +1," to play")
    x= int (input())
    play (x, count%2+1)
    if count>=9):
        print("its a draw");
    if (check_winner):
        print ("player", count%2+1, "won"), break
    count++
    if (count==9);
        print (" Its a draw")
        break
```

Output:

1	2	3
4	5	6
7	8	9

Player 1 to play: 1

X		

player 2 to play: 2

X	O	

player 1 to play: 5

X	O	
	X	

player 2 to play: 3

X	O	O
	X	

player 1 to play: 9

X	O	O
	X	
		X

player 1 won the game

Date: 24-11-2021
Page No:

Write a program to simulate 8 puzzle problem

n=3

row = [0, 1, 0, -1]

col = [-1, 0, 1, 0]

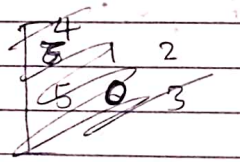
n/s :

def

1. Define board's initial position with random values from 0-8 in a matrix 3×3.
2. Set final goal state :

1	2	3
4	5	6
7	8	0

3. Use BFS algorithm. Check with goal state after each move left, right, up, down.



4	1	2
0	5	3
7	8	6

up / down / right / left.

invalid

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

left /

1 0 2 4 5 3 7 8 6

left

1 2 0 4 5 3 7 8 6	up, up	1 2 3 4 5 6 7 8 0

Date:
Page No:

In the each state check state with final state, if it matches print solution.

24-11-21

code :

```
import copy
from heapq import heappush, heappop

n = 3
row = [1, 0, -1, 0]
col = [0, -1, 0, 1]

class PriorityQueue:
    def __init__(self):
        self.heap = []

    def push(self, k):
        heappush(self.heap, k)

    def pop(self):
        return heappop(self.heap)

    def empty(self):
        return not bool(self.heap)

class Node:
    def __init__(self, parent, mat, empty_tile_pos,
                 cost, level):
        self.parent = parent
        self.mat = mat
        self.empty_tile_pos = empty_tile_pos
        self.cost = cost
        self.level = level
```

```
def __lt__(self, nxt):
    return self.cost < nxt.cost

def calculateCost(mat, final) -> int:
    count = 0
    for i in range(n):
        for j in range(n):
            if mat[i][j] and (mat[i][j] !=
                              final[i][j]):
                count += 1
    return count

def newNode(mat, empty_tile_pos, new_empty_tile_pos,
            level, parent, final) -> Node:
    new_mat = copy.deepcopy(mat)
    x1, y1 = empty_tile_pos
    x2, y2 = new_empty_tile_pos
    new_mat[x1][y1], new_mat[x2][y2] =
    new_mat[x2][y2], new_mat[x1][y1]
    cost = calculateCost(new_mat, final)
    return new_node

solve()
```

Output:

initial

1 2 3
8 0 4
7 6 5

final

1 2 3
4 8 6
7 0 5

1 2 3
0 4 6
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

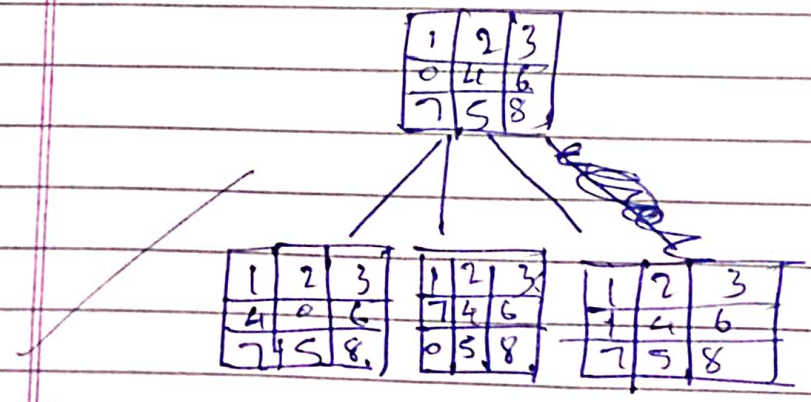
1 2 3
4 5 6
7 8 0

Write a program to simulate puzzle-8 p
using IDDFS.

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

    def dfs(route, depth):
        if depth == 0:
            return
        if route[-1] == goal:
            return route
        for move in get_moves(route[-1]):
            if move not in route:
                next_route = dfs(route+[mo
                                 depth-1)
                if next_route:
                    return next_route

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



Write a program to simulate 8 Puzzle problem using A* algorithm

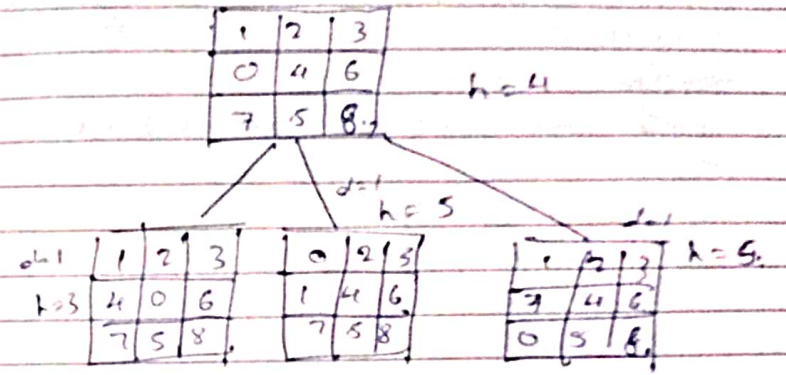
Algorithm:

- create initial & goal state.
- Heuristics is considered, lower heuristics is considered in each step.
  f value = h value + path cost.
- Initially expand the node, find the location of empty tile & generate the node calculate the heuristic function value in each step

$$f(x) = h(x) + g(x)$$

- Maintain two list namely 'open' & 'close' The nodes generated are stored in the list, sort using f(x) values
- Explore the node i.e, stored in the open list. using f(x) values.

The goal is reached when h(x)=0. Implies all tile are in correct position



LAB-6

Write program to simulate vaccum cleaner for 2 rooms.

steps:

step 1: Create rooms of ~~room~~ array ~~room~~ ~~room~~.

step 2: In each cell of ~~room~~ array '1' is represents dirty, '0' as clean.

step 3: Create clean function through which vaccum cleaner moves right, left, up, down & checks if its dirty.

step 4: After cleaning room travel to next room and starts ~~the~~ cleaning.

eg: room = [0, 1]

room1 room2

0	1
1	0

Proved

if room1 is dirty cleans the room else move to room 2.

Code:

```
def clean (floor):
    i, j, col, row = 0, 0, len(floor[0]), len(floor)
    for i in range(row):
        if (i%2==0)
            for j in range(col):
                if (floor[i][j]==1):
                    printf(floor,i,j)
                    floor[i][j]=0
                    printf(floor,i,j)
```

```
else
    for j in range(col-1, -1, -1):
        if (floor[i][j]==1):
            printF(floor,i,j)
            floor[i][j]=0
            printF(floor i, j)

def printF(floor, row, col):
    print("Floor Matrix:")
    for r in range(len(floor)):
        for c in range(len(floor[r])):
            if r==row and c==col
                print(f" >{floor[r][c]}", end
                ="")
            else:
                print(f" {floor[r][c]}", end = "")
        print(end = '\n')
    print(end = '\n')
```

Output:

Enter the clean status for each cell (1-dirty, 0-clean)

1 1

1 1

The Floor Matrix:

> 1 1

1 1

Date : 22-12-24
Page No :

LAB-6

Write program to simulate vaccum cleaner in 2 rooms.

step 1: Create rooms of array

step 2: In each cell of array '1' is represented dirty '0' as clean.

step 3: Create a function through which vaccum cleaner moves right, left, up, down & checks if its dirty.

step 4: After cleaning room move to next room and starts cleaning.

eg: room = [1 , 1]

room1 room2

0	1
1	0

Logic:

if room1 is dirty cleans the room else move to room 2.

Code:

```
def clean(floor):
    i, j, col, row = 0, 0, len(floor[0]), len(floor)
    for i in range(row):
        if (i%2==0)
            for j in range(col):
                if floor[i][j]==1):
                    printF(floor,i,j)
                    floor[i][j]=0
                    printF(floor,i,j)
```

```
else
    for j in range(col-1, -1, -1):
        if (floor[i][j]==1):
            printF(floor,i,j)
            floor[i][j]=0
            printF(floor,i,j)

def print_F(floor, row, col):
    print('Floor Matrix:')
    for r in range(len(floor)):
        for c in range(len(floor[r])):
            if r==row and c==col:
                print(f'>{floor[r][c]}<', end=' ')
            else:
                print(f' {floor[r][c]} ', end=' ')
        print(end='\n')
    print(end='\n')
```

Output:

Enter the clean status for each cell (1-dirty 0-clean)

1 1

1 1

The Floor Matrix:

>1< 1

1 1

Page No :

Floor Matrix:

```
>0<  1
 1   1
```

Floor Matrix:

```
 0   >1<
 1    1
```

Floor Matrix:

```
 0   >0 <
<0>  1
```

Floor Matrix:

```
 0    0
 1   >1<
```

Floor Matrix:

```
 0    0
 1  >0<
```

Floor Matrix

```
 0    0
>1<   0
```

Floor Matrix

```
 0    0
>0<   0
```

22/12/23

Date :
Page No :

Implement Knowledge based ~~entailment~~ resolution for
KB = P, ~PVQ, PV~Q, PV~QVR, ~QVP

```
class KnowledgeBase:
    def __init__(self):
        self.facts = set()

    def add_fact(self, fact):
        self.facts.add(fact)

    def check_entailment(self, statement):
        return all(rule(statement) for rule in
                   self.facts)

def rule_0(statement):
    return 'p' in statement.lower()

def rule_1(statement):
    return "np" in statement.lower() or "nq"
           in statement.lower()

def rule_2(statement):
    return "p" in statement.lower() or "nq" in
           statement.lower() or 'r' in statement.
           lower()

def rule_3(statement):
    rule_3(statement.lower() or 'r' in statement.lower()
           return nq in statement.lower() or
                  r in statement.lower()
```

Date :
Page No :

Floor Matrix:

```
>0<  1
 1   1
```

Floor Matrix:

```
 0   >1<
 1    1
```

Floor Matrix:

```
 0   >0<
 1    1
```

Floor Matrix:

```
 0    0
 1   >1<
```

Floor Matrix:

```
 0    0
 1   >0<
```

Floor Matrix

```
 0    0
>1<   0
```

Floor Matrix

```
 0    0
>0<   0
```

Sneha
22/12/23

Date :
Page No :

Implement Knowledge based resolution for entailment for
KB = P, ~P V Q, P V ~Q, P V ~Q V R, ~Q V R.

```
class KnowledgeBase:
    def __init__(self):
        self.facts = set()

    def add_fact(self, fact):
        self.facts.add(fact)

    def check_entailment(self, statement):
        return all(rule(statement) for rule in
                   self.facts)

def rule_0(statement):
    return 'p' in statement.lower()

def rule_1(statement):
    return "~p" in statement.lower() or "~q"
           in statement.lower()

def rule_2(statement):
    return "p" in statement.lower() or "~q" in
           statement.lower() or "r" in statement.
           lower()

def rule_3(statement):
    rule_3(statement.lower() or r in statement.lower()
    return ~q in statement.lower() or
           r in statement.lower()
```

```
kb = KnowledgeBase()
kb.add_fact(rule_0)
kb.add_fact(rule_1)
kb.add_fact(rule_2)
kb.add_fact(rule_3)

user_statement = input("Enter a statement:")
entailed_result = kb.check_entailment(user_sta

if entailment_result:
    print("The statent is entailed")
else:
    print("That statment is not
                                    entail
```

Output:

Enter the statement: ~pVr

This statement is entailed

Enter the statement: ~nVp.

The statement not entailed.

Implement

Knowledge Based Entailment.

```
from sympy import symbols, And, Not,
Implies, satisfiable

def create_knowledge_base():

    p = symbol('p')
    q = symbol('q')
    r = symbol('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)
```

Date :
Page No :

Output :

Knowledge Base : ∧∨ ¬

(Implies (p, q)) ∧

¬(Implies (q, r))

Query : p

Query entails Knowledge Base : False

Sneha
29/12/23

Implement conversion of first order logic to Conjunctive Normal form & Unification of FOL

```
import re

def getAttributes(expression):
    exp = exp.split("(")[1:]
    exp = "(".join(exp)
    exp = exp[:-1]
    exp = re.split("(?<!\(.),(?!.\))", exp)
    return exp

def getInitialPredicate(exp):
    return exp.split("(")[0]

def isConstant(char):
    return char.isupper() and len(char)==1

def isVariable(char)
    return char.islower() and len(char)==1

def replaceAttributes(exp, old, new):
    attributes = getAttributes(exp)
    for index, val in enumerate(attributes):
        if val==old:
            attributes[index] = new

def apply(exp, substitutions):
    for substitution in substitutions:
        new, old = substitution
        exp = replaceAttributes(exp, old, new)
    return exp
```

```
def unify(exp1,exp2):
    if exp1==exp2:
        return []

    if isConstant(exp1 and isConstant(exp2):
        if exp1 != exp2:
            return False

    if isConstant(exp1):
        return [(exp1, exp2)]

    if isConstant(exp2):
        return [(exp2, exp1)]

    if isVariable(exp1):
        if checkOccurs(exp1,exp2)
            return False
        else:
            return [(exp1,exp2)]

    if isVariable(exp2):
        if checkOccurs(exp1,exp2):
            return False
        else:
            return [(exp1, exp2)]

    if getInitialPredicate(exp1) != getInitial
    Predicate(exp2):
        print("Predicate do not match cannot
        be Unified")
        return False
```

initialSubstitution . extend (remainingSubstitution)
return initialSubstitution

```
exp1 = "Knows(x)"
exp2 = "Knows("Richard")"
substitutions = unify(exp1, exp2)
print( substitutions)
```

Output:
Substitution:
[('x', 'Richard')]

Fol to CNF

```
def fol_to_cnf (fol):
while '-' in state:
    i = state.index('-')
    new_state = '[' + state[illegible] + '=>' +
    statement[i+1:] + '-2[' + state[illegible] +
    '=>' + state = new_state
    state = state.replace("=>", [illegible])

while '-' in state:
    i = state.index([illegible])
    br = state.index('[') if '[' in
in statement else 0
    state = state[:br] + new_state if [illegible]
else new_state
```

```
while '∀' in state:
    i = state.index('∀')
    state = list(state)
    state[i], state[i+1], state[i+2] =
    '∃', state[i+2], '.'
    state = ''.join(state)

while '¬E' in state:
    i = state.index('¬E')
    s = list(state)
    s[i], s[i+1], s[i+2] = '∀', s[i+2]

state = state.replace('→', '
state = state.replace('¬[E', '∃E
```

Output:

~~[∀x Animal(x)~~ animal(y)] => loves(x,y)]

[∃animal(y) ∧ loves(x,y)] ∧ [∃y loves(x,y)

animal(y)]

Date :
Page No :

Prove the given Query using Forward Chaining

```
class Fact:
    def __init__(self, expression):
        self.expression = expression
        pred, params = self.splitExp(expression)
        self.pred = pred
        self.params = params
        self.result = any(self.getConstants())

    def splitExp(self, exp):
        pred = getPredicate(exp)[0]
        param = getAttribute(exp)[0].strip('()')
                .split(',')
        return [pred, param]

    def getRes(self):
        return self.result

    def getConstants(self):
        return [None if isVariable(v) else
                v 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.exp = exp
        l = exp.split('=>')
        self.lhs = [Fact(f) for f in l[0].
        split('&')]
        self.rhs = Fact(l[1])
    def evaluate(self, facts):
        constant = {}
        new_lhs = []
        for fact in facts:
            for val in self.lhs:
                if val.pred == fact.pred:
                    for i,v in enumerate(val.
                    getVariables()):
                        if v:
                            constant[v] = fact.getConstants()[i]
                    new_lhs.append(fact)

        return Fact(expr) if len(new_lhs)
        and all([f.getResult() for f in new
        else None.

kb = KB().
kb.tell('missile(x) => weapon(x)')
kb.tell('missile(M1)')
kb.tell('enemy(x, America) => hostile(x)')
kb.tell('American(West)')
kb.tell('enemy(Nono, America)')
kb.tell('Owns(Nono, M1)')
kb.tell('Missile(x) & Owns(Nono, x) =>
        sells(West, x, Nono)')
```

Kb.tell (American(x) & Weapon(y) & sells(x, y, z) & Hostile(z) ⇒ Criminal(x))

Kb.query ('Criminal(x)')

Kb.display()

Output:

Query: Criminal(x):

1. Criminal(West)

All facts:

1. American(West)
2. sell(West, M1, Nono)
3. missile(M1)
4. Enemy(Nono, America)
5. Criminal(West)
6. Weapon(M1)
7. Own(Nono, M1)
8. Hostile(Nono)

Date: 25/11/24