INDEX

Name : Mohammad Faiz Mahmud. Class : AI R6

Section : VI Roll No. : 106 Subject :

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
1	3/10/23	Google Python Course		
2.	10/10/23	Practice Questions		
3	17/11/23	Tic tac toe game		
4		8 puzzle game using BFS		
5		" " IDDS		
6		" " A^*		
7		Vaccum Cleaner		
8	29/12/23	K.B Entailment		
9	29/12/23	K.B Resolution		
10	19/1/24	unification		
11	19/1/24	convert FOL to CNF		
12	19/1/24	Knowledge Base containing first order logic statement.		

25/1/24

10/11/23

Date
Page

Practice Questions :-

(Q1) Write a python program to check age criteria using else-if ladder.

(Q2) Write a python program to print the multiplication table.

(Q3) Write a python program to print the pattern : 1, 2, 2, 3, 3, 3, 4, 4, 4, 4 . . . .

(A1)
```
age = int(input('Please enter a persons age:'))
    if( age <=12):
        print("Child")
elif( age >12 and age <=17):
        print("Teenager")
elif( age >=18 and age <65):
        print("Adult")
else :
    print("Older Adults")
```

Output:-

Please Enter a persons age : 15
teenager.

Scanned with CamScanner

(A2)

```
num= int(input("Enter the Number:"))
for x in range (1,11):
    print(num, "x", x, "=", x * num)
```

(A3) Output:-

Enter the number: 8.

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

(A3)

```
num= int(input("Enter the Number:"))
for x in range (num):
    n= x+1
    for y in range (x+1):
        print(n, end="")
    print("\n")
```

Output:

1
22
333
4444

Scanned with CamScanner

Date
Page

(Q4) Bubble sort.

```
n= input("Enter size of array")
n=int(n)
arr = []
for _ in range(n):
    arr.append(int(input()))
print("Sorted array", arr)
i= 0
while (i<n):
    j=i
    while (j<n):
        if(arr[i] > arr[j]:
            x= arr[i]
            arr[i]= arr[j]
            arr[j]=x.
        j=j+1
    i= i+1.
print('Sorted array,arr)
```

Output:-
Enter size of array 6.
2 5 8 4 9 3
unsorted array [2,5,8,4,9,3].
Sorted array [2,3,4,5,8,9].

(Q5) Integer Reversal.

```
num = int(input('Enter the integer to be reversed').
reverse =0.
while (num>0):
    runner = runner 10+ num % 10.
    num= num //10
    print (runner).
```

Output:
Enter the integer to be reversed 258
852

10/4/22

Scanned with CamScanner

Lab-1: Kaggle Python Course

The topic covered the above mentioned python course are :-

1. Hello python
Quick introduction to python, syntax, variable assignment and numbers.
2. functions and getting help.
Calling functions and ~~distinguishing~~ defining sum.
using built in documentation.
3. Boolean and conditions.
using tools for branching logics.
4. Units
including slicing, indexing and working lists.
5. loops and list comperasions.
for & while loops.
6. Strings and Dictionaries.
working with strings & dictionaries.
7. Working with external libraries.
import operation over loading over external libraries.

Abhay
10/11/25

Scanned with CamScanner

Date
Page

tictac toe:-

1)

for creating a blank board import numpy as np. import random.

```
def create():
    return (np.array([[0,0,0],
                      [0,0,0],
                      [0,0,0]]))

def posi(board):
    l=[].

    for i in range(len[board]):
        for j in range(len(board)):
            if board[i][j] == 0:
                l.append((i,j))
    return(l)

def randomplace (board, player):

    selection = possibilities (board)
    cur = random.choice(selection)
    board [cur] = player
    return board

def row.win ( board, player)

    for x in range (in(board)):
```

Scanned with CamScanner

```
    win = true
    for y in range(len(board)):

        if board[x,y] != player:
            win = false
            continue
    if win == true:
        return(win)
    return(win)

def colwin(board, player)
    for x in range(len(board)):
        win = true
        for y in range(len(board)):
            if board[y][x] != player
                win = false
                continue
        if win == false
            return(win)
    return(win),
```

Date
Page

8 puzzle problem using bfs-

In the problem the square will have N+1 tiles where N = 8, 15, 24 and so on

N=8 means the square will have 9 tiles (3 rows and 3 columns)

In this problem, initial state / initial configuration (start state) will be given and we have to reach the goal state

suppose

Initial state:-

1	2	3
4	6	
7	5	8

good state:-

1	2	3
4	5	6
7	8	

The puzzle can be solved by moving the tiles one by one in the style empty space and this achieve the goal state

rules:-

the empty can only move in 4 directions

Scanned with CamScanner

1) up 2) down 3) right 4) left

It cannot be moved diagonaley
can take only one step at a time.

Complexity:- $O(b^d)$ where

b branching factor, d-depth factor
worst case - 3^{20}
for 8 puzzle problem.

Branching factor b = $\frac{\text{all possible moves}}{\text{no. of tiles}}$

$= \frac{24}{9} = 2.67 \approx 3$

Example.

d=0.

1	2	3
4		6
7	5	8

d-1

1	2	3
4		6
7	5	8

1	2	
4	6	3
7	5	8

1	2	3
4	6	8
7	5	

Scanned with CamScanner

Output :- solution found :

Number of moves : 2
(0,1)
(0,1)

program:

```
from collection import queue
import random

def find blank (board):
    for i in range (3):
        for j in range (3):
            if board (i) [j] == 0:
                return i, j:

def generate move [board]:
    moves = []
    blank row, blank col = find_blank (board)
    possible_moves = [
        (1,0), (-1,0), (0, 1), (0,-1)
    ]
    for dr, dc in possible moves:
        new_row, new_col = blank_row +
        dr, blank row + dc

if 0 <= new_row <3 and 0 <= new_col<3
    new_board = [row [:] for row in board]
    new_board [blank_row] [blank_col]
```

```
new_board [new_row] [new_col]
  = new_board [new_row] [new_col]
new board [blank row] [blank col]
  moves. append (new_board

def solve_puzzle (initial state, goal, state):
  verified_set()
  queue_deque ([(initial_state)])

while queue:
    current_state path = queue.pop left()
  visited. add (tuple (map(tuple, array_state))
        if current_state == goal_state:
            return path

    possible_moves = generate_moves (current_state)
    for move in possible_moves:
        if tuple (map(tuple, move)) not invisible
            queue append((move_path+[move]))

    return none

def print_steps(solution_path):
    if solution_path:
        print(" Steps for reach the goal")
        for step in solution_path:
            print(" ... ")
        for row in step
            print("", end=" ")
```

Scanned with CamScanner

```
for val in row:
    if val == 0:
        print("  ", end="|")
    else
        print(val, end="|")
        print()
        print()
else
    print("No solution exists for this puzzle
```

initial $\begin{bmatrix} 1 & 2 & 3 \\ & 4 & 6 \\ 7 & 5 & 8 \end{bmatrix}$

```
goal = [
        [0, 1, 2]
        [3, 4, 5]
        [6, 7, 8].
]
solution_path = solve_puzzle(initial, goal)
print_steps(solve_path).
```

Output:

1	2	3
	4	6
7	5	8

1st move:

1	2	3
4		6
7	5	8

2nd move.

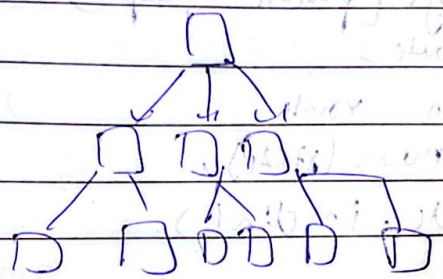
1	2	3
4	5	6
7		8

3rd move.

1	2	3
4	5	6
7	8	

Scanned with CamScanner

8 puzzle problem by IDDS



① Goal state and initial state predicted

② start with depth = 0 and increase depth by 1 on every iteration and perform depth first search the that particular depth

③ New get function goes in

④ we stop when we have the goal state print the path followed

Codes

```
def id_dfs (puzzle, goal, get_move)
    import itertoolz
    def dfs (route, depth):
        if depth == 0:
            return;
        if route[-1] == goal
            return route;
        for move in get_move (with ( if move not in
            route, ~~yield move~~ dfs (not + depth)
            if hentler
            return next route
```

```
for depth in iterrtools.count()
        route = dfs(puzzle, depth)
            if route:
                return route
def possible moves(state):
        b = state.index(0)
        d = []
        if b int 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 i in d:
            pos_move.append(general state)
        return pos moves.

def generate(state, m, b):
        temp = state copy()
        if m == 'd':
        temp[b+3], temp[b] = temp[b], temp
        temp

        return temp.
unit = [1,2,3,0,4,6,7,8,9].
goal = [1,2,3,4,5,6,7,8,9].

result = idfs(unit, goal, possible moves)
```

```
if route:
    print ("successor", i)
    print ("path:", rank)
else
    print ("Failed to find set").
```

Output:

successor

path = ([1, 7], 0, 4, 6, 7, 5, 8), (1, 7).

8 puzzle problem using A* algo

Find the most cost effective path to reach the final state from the initial state.

```
import heapq

def get_blank_pos(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j;
def get_neighbour(state, moves)
    row, col = get_blank_pos(state
    new_row, new_col = row + move[q
                                + col
                                + move[i]
    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]
        return new_state
        return None

def in_good(state)
    goal_state = [[1,2,3], [4,5,6], [7,8,9]
    return state == good_state

def heuristics(state)
    distance = 0
```

Scanned with CamScanner

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

return distance;

def solve_puzzle (initial_state):
    start_node = (initial state, none, none, 4,
                  manhattan (initial_state))
    open_set = [start_node]
    closed_set = set()

while openset:
    current state, parent, move, mh
        heapq, heappop (open set)

if (is goal (current_state).
        path[]
    while parent:
        path in sett (0, [move, cur, state]

    return path

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

    mov ver[(0,1) (1,0), (0,-1), (-1,0)]

initial state = [[1,2,3], [4,5,6], [0,7,8]].
```

Scanned with CamScanner

```
set_path = solve_puzzle (initial_state)

if set_path
    print ("Move", move)
    for row in state
        print (row)
    print ("....")
else
    Print ("No sol")
```

Output

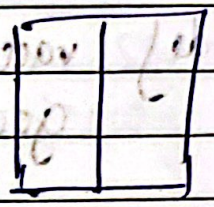
```
1 2 3
4 5 6
0 7 8

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
4 0 6
4 7 8
```

Scanned with CamScanner

Date
Page

Vaccum cleaner.

Algo:



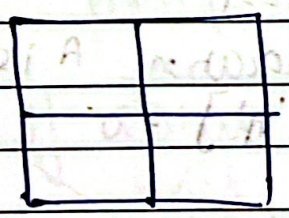
for 2 rooms:

→ We let ones divide which room they start from

→ if it is dirty clean it and if it is clean move (to ch left or right accordingly to starting position)

→ return a message cleaned rooms if no room is dirty.

for 4 rooms



→ Same procedure but the robot can move left - right and up down but not diagonal

code:

```
def vaccum_Moded():
    goal_state = {'A' = 0, 'B' = 0}
    cost = 0;
    location_input = input("Enter location of vaccum)
    Statline_input = int (input ("Enter state" + location_input))
    Statin_input_complement = int (input ("Enter status of the nos"))
    print ("Initial location condition" + str (goal_state))
    if location_input == 'A':
        print ("Vaccum is placed in location A")
        if status_input == 1
            print ("location A is dirt")
            goal_state ['A'] = 0
            cost += 1
            print ("cost for cleaning A" + str (cost))
            print ("location A has been cleaned")

        if status_input_complement == 1
            print ("location B is Dirtane")
            print ("Moving right to location B")

            cost += 1
            print ("cost for moving right
                + str (cost))
```

```
goal_state['B'] = 0
    cost += 1
    print("cost for suck" + str(cost))
    print("location B is already clean")

else:
    print("No action" + str(cost))
    print(cost)
    print("location B is already clean")

else:
    print("Vacuum is placed in location B")
    if status_input == 1
        print("location B is dirty")
        goal_state['B'] = 0
        cost += 1
        print("Cost for cleaning" + str(cost))
        print("loc B has been cleaning")

    if status_input_complement == 1:
        print("location A is dirty")
        print("moving left to location A

        cost += 1
        print("Cost for moving left" + str(cost))
        goal_state['A'] = 0
        cost += 1
        print("cost for suck" + str(cost))
        print("loc A has been cleaned")
else:
        print(cost)
        print("location B is already clean")
```

Scanned with CamScanner

```
i) | status_count_compl == 1:

    print ("loc A is dirty")
    print ("Moving left to watch A

    cost += 1
    print ("cost for moving left " + str(cost))
    goal_state ['A'] = 0
    cost += 1
    print ("cost for suck " + str(cost))
    print ("loc A has been cleaned")

else
    print ("No action" + str(cost))
    print ("loc A is alive")

print ("Goal state:")
print (goal_state)
print ("Performance measurement" + str(cost))

vacuum_world()
```

Output:

```
Enter loc of vac : A
Enter status of A :
Enter status of other room
Initial location condition = {'A': 0, 'B': 0}
Vacuum is placed in location A.
loc A is dirty
cost for cleaning A is
loc A has been cleaned
loc B is dirty
Moving right to B.
cost for movement 2
cost for suck 3
loc B has been clean
goal state:
{'A': 0, 'B': 0}
perform measurement :7
```

29/12/23

Q) Create a knowledge based using propositional logic and that the given query entails the knowledge base or not.

Algorithm:

Knowledge Base Class:

initialization:

def __init__(self):
self.clauses = []

(Create an instance of knowledge class with an empty list).

Adding clauses.

Appends a new clause to the list of clause in the knowledge base.

Resolving Clauses:

Combine the clauses by resolving.

Negate the query:
Obtain the negate of query
→ Combine with knowledge base.

Check satisfiability
def satisfiability()
(Check the conjunction)

Determine entailment
→ if (conjunction == true,
return True.
else
return false.

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)))

if __name__ == "__main__":
    kb = create_knowledge_base()
    query = symbols('not(p)')
    result = query_entails(kb, query)
    print("Knowledge Base:", kb)
    print("Query", query)
    print("Query entails Knowledge Base:", result)
```

Output:

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

Query = (not, p)

Query entails Knowledge Base: False

29/12/23

Lab-8

Knowledge Base Resolution

```
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 c2:
        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, negate(knowledge base)
            if i != j.
```

```
new_clause = resolve(c1,c2)
if len(new_clause) > 0 and new_c
                not in KB:
    new_clause.add(new_clause)

if not new_clause:
    break;
knowledge_base 1 = new_clause
return knowledge_base.

if __name__ == "__main__":
    kb = {('p', 'q'), ('~p', 'r'),
    ('~q', '~r')}

    result = resolution(kb)
    print("Original KB", kb)
    print("Resolve KB", result)
```

19/10/24

lab-9:

unification:

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

def getInitialPredicate(expression):
    return expression.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:
            attrib[index] = new.
```

Scanned with CamScanner

```
    predicate = get Initial Predicate(exp)
    return predicate + "(" + ",".
            join (attributes + ")"

def apply (exp, substitution):
    for substitution in substitution.
        new, old = substitution
        exp = replace Attributes(exp
                , old, new)
    return exp

def check Occurs (var, exp):
    if (exp.find (var) = -1)
        return false.
    return true

def get First Part (exp expression),
        predicat = get attributes.(expression)
        return attributes [0].

def get Remaining Part (expressions):
    predicate = get Initial Predicate
    (exp
    attributes = get Attribute (exp)
    new Expression = predicate + "(" + ",".
        (attributes [1:] + ")"
    return new Expression
```

```
def unify (exp1, exp2):
    if (exp1 = exp2)
        return []
    if is constant (exp1) and
            is constant (exp2).
        if exp1 != exp2.
            return false
    if is constant (exp1)
        return [(exp1, exp2)].
    if is constant (exp2)
        return [(exp2, exp1)].
    if is variable (exp1):
        if check occurs (exp1, exp2)
            return False
        else
            return [(exp2, exp1)]
    if is variable (exp2).
        if check occurs (exp2, exp1)
            return False.
        else:
            return [(exp1, exp2)]
    if get Initial Predicate (exp1),
            get Initial
                Predicate(exp)
        print ("Predicate do not match,
                cannot to unified
```

Scanned with CamScanner

```
return False.

attibute count1 = len(get Attibute (exp1)
attibute count2 = len(get Attibute(exp2)
if attibute count1 != attibute count2:
    return False

head1 = get first1 part (exp1)
head2 = get first part (exp2)
initial substitution = unify
                (head1, head2)
if not initial substitution:
    return False.
if attibute count == 1:
    return initial substitution

tail1 = get Remaining Part (exp1)
tail2 = get Remaining Part (exp2).

if initial substitution != []:
    tail1 = apply (tail1, initial
                substitution)

tail2 = apply (tail2, initial subst,

remaining substitution = unify(tail, tail2)

if not remain substitution:
    return False
initial sub. extent (remaining
                subs) return
```

Convert FOL into CNF:

```
def get Attibute (string):
    expr = '([^)]+)'
    matches = re.findall(expr, strsin)
    return [m for m in str(matches)
        if m.isalpha()]

def get Predicates (string):
    expr [a-z~]+([A-Za-z]+)
    return re.findall (expr, string)

def skolemization (statement):
    SKOLEM_CONSTANTS = [] {chr(c)}
    for c in range (ord('A'),
                ord('Z')+1)]
    matches = re.findall('[?]',
                statement)
    for match in matches[::-1]
        statement = statement.replace(match,'')

    for predicate in get Predicates
                (statement):
        attributes = get Attibutes
                (predicate)
        if ''.join(attributes) is lower():
            statement = statement.replace
                (match[0], SKOLEM_CONSTANTS
                .pop(0))
```

Scanned with CamScanner

return statement

import re

```
def for_to_cnf(fov):
    statement = fol.replace("->", "-")
    expr = '([^]+)'
    statement = re.findall(expr, statement)
    for i, s in enumerate(statement):
        if '[' in s and ']' not in s:
            statement[i] += ']'
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    while '-' in statement:
        i = statement.index('-')
        br = statement.index('[')
        if '[' in statement else 0
            new_statement = '~' + statement[br:i]
            + '|' + statement[i+1:]
            statement = statement[:br]
            + new_statement if br>0
            else new_statement

    return skolmization(statement)

print(for_to_cnf("bird(x) -> ~fly(x)"))

print(fol_to_cnf("∃x (bird(x) ->
                ~fly(x)]"))
```

Date
Page

Q) Create a knowledge base containing of first order logic statement and prove the given query using forward chaining.

```
from sympy import symbols, Eq,
                And, or, Implies, ask,
                satisfied.

John, Mary, Alice, Bob = symbols(John, Mary
                , Alice, Bob)

Parent = symbols('Parents')
Grand Parent = symbols('grandparent')

Kb = [Eq(Parent(John, Alice), True)
    , Eq(Parent(Mary, Alice), True)
    , Eq(Parent(Alice, Bob), True)
    , Implies(Parent(x, y)
                Grandparent(x, y)]

query = Grandparent(John, Bob)

def forward_searching(kb, query):
    new_facts = set()
    while true:
        for fact in kb:
            if ask(fact):
                continue
```

Scanned with CamScanner

```
        if satisfiable(fact):
            new_facts.add(fact)
        if not new_facts:
            break
    kb.extend(new_facts)
    return ask(query)

result = forward_reasoning(kb, query)
print("Query", query)
print("Result", result)
```

Output:

Query: Grandparent(John, Bob)
Result: True

25/11/24