3/11/2023

Date
Page

LAB 1 Kaggle Python Course

The topics 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 defining our own.
   Using builtin documentation.
3. Booleans and conditionals
   Using bools for branching logics
4. Lists.
   Included slicing, indexing and mutating lists.
5. Loops and List comprehension.
   For & while loops.
6. Strings and Dictionaries.
   Working with strings & dictionaries
7. Working with external libraries.
   Import, operator overloading and external libraries.

A 3-11-23

10/11/23

LAB 2 Python Code Practice

1. Age criteria using else-if ladder.
2. Table of a number.
3. Pattern to print → 1, 22, 333, 4444 ....
4. Bubble sort.
5. Print reverse of integer.

Codes and Outputs:

① // Age criteria

```
age = int(input("Enter your age"))
x = "you are"
if (age < 10):
    print(x+" too young").
elif (age < 20):
    print(x+" teenager").
elif (age < 30):
    print(x+"midlife")
elif (age < 50):
    print(x+" experienced").
else:
    print(x+" too old")
```

//output

Enter your age 7

you are too young.

Date
Page

② // Multiplication table.

```
number = int(input("Enter number to get the table"))
i=1
while (i<=10):
    print(number, "*", i, "=", (number*i))
    i=i+1.
```

// Output.

Enter the number to get the table. 8

8 * 1 = 8

8 * 2 = 16

;

;

8 * 10 = 80.

③ // Pattern printing.

```
x = int(input("Enter number of times the pattern
                should be printed"))
i=1
while (i<=x):
    j=0
    while (j<i):
        print(i)
        j=j+1.
    i=i+1.
    print("\n")
```

// Output.

Enter number of times pattern should be printed.

4

1

2 2

④ // Bubble sort.

```
n=input ("Enter size of array")
n= int (n)
arr =[]
for - in range (n):
    arr.append (int (input()))
print ("unsorted 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 the array 6.
2 5 8 4 9 3
unsorted array [2, 5, 8, 4, 9, 3]
sorted array [2, 3, 4, 5, 8, 9]

Date
Page

(5) #Integer reversal.

```
num = int(input("Enter the integer to be reversed"))
numrev = 0
while(num>0):
    numrev = numrev*10 + num%10
    num = num//10
print(numrev)
```

// Output:

Enter the integer to be reversed: 258

852

10-11-23

17/11/23

RA 17/11/23.

LAB 3 TIC-TAC-TOE

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

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

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

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

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
        #add all eight cases of winning

def playermove():
    run=True
    while run:
        move=input('Please select a position(1-9):')
        try:
            move=int(move)
            if move>0 and move<10:
                if spaceisfree(move):
                    run=False
                    insertletter('X',move)
                else:
                    print('Sorry, place occupied!')
            else:
                print('Enter numbers in range!')
        except:
            print('Please type a number!')
```

```
def computermove():
    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
        # checking if winning move is possible
          for computer itself.
    cornersopen = []
    if 5 in possiblemoves:
        move = 5
        return move
    if i in possiblemoves:
        if i in [1,3,7,9]:
            cornersopen.append(i)
    if len(cornersopen) > 0:
        move = selectrandom(cornersopen)
        return move
    # fill up centre first & then fill corners
      then selecting at random.
    edgesopen = []
    for i in possiblemoves:
        if i in [2,4,6,8]:
            edgesopen.append(i)
    if len(edgesopen) > 0:
        move = selectrandom(edgesopen)
    return move
    # fill up edges then.
```

```
def selectrandom(li):
    import random
    ln = len(li)
    r = random.randrange(0, ln)
    return li[r]
// random function.
def isboardfull(board):
    if board.count(' ')>1:
        return False.
    else:
        return True.
// check if board is full.
def main():
    print('welcome to the game')
    printboard(board)

    while not(isBoardfull(board)):
        if not(iswinner(board,'o')):
            playermove().
            printBoard(board).
        else.
            print('sorry, o won this time')
            break.
        if not(iswinner(board,'x')):
            move = compMove().
            if move == 0.
                print('Tie!')
            else:
                insertletter('o', move).
                printboard(board).
    else:
        print('x won this time').
        break.
```

```
if isboardfull(board):
    print("The Game").
//main function to take input.
```

Output

Welcom to tic tac toe.

```
   |   |
   |   |
   |   |
```

Please select position to place an 'X' (1-9): 7

```
   |   |
   | O |
 X |   | O
```

Please select position to place an 'X' (1-9): 9

```
   |   |
   | O |
 X | O | X
```

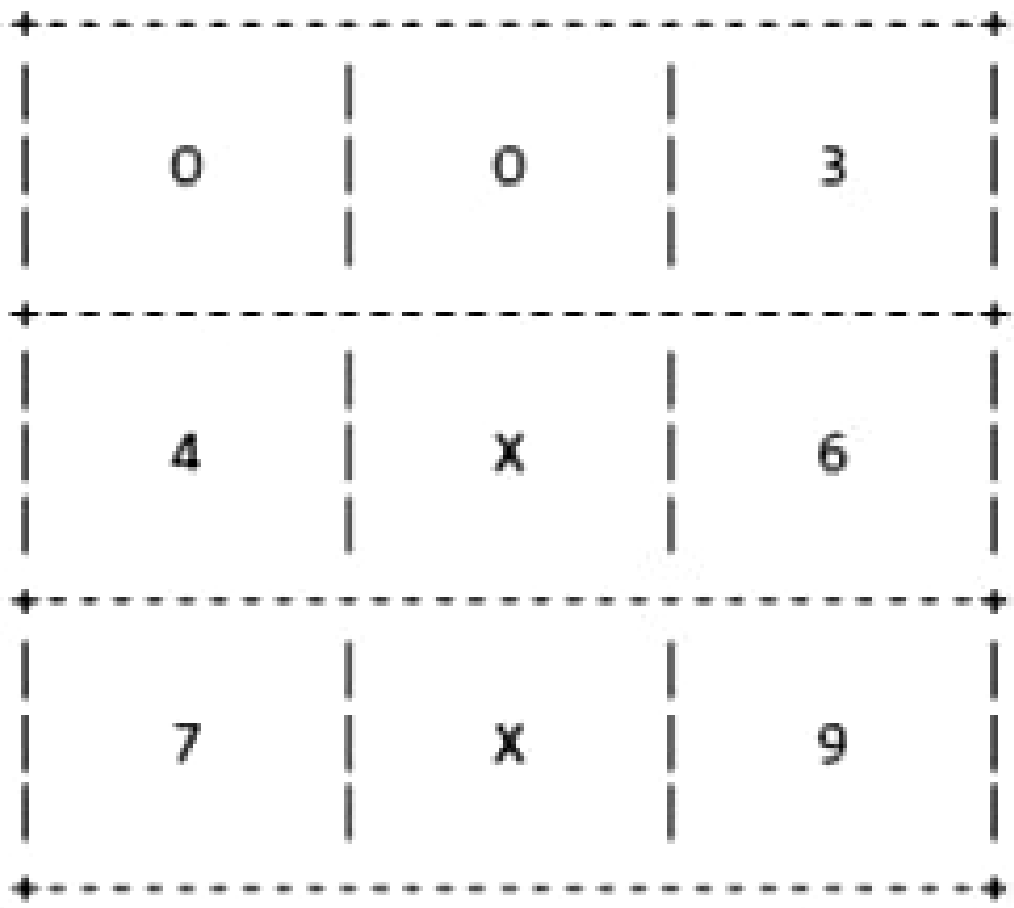
Please select position to place an 'X' (1-9): 1

```
 X | O |
   | O |
 X | O | X
```

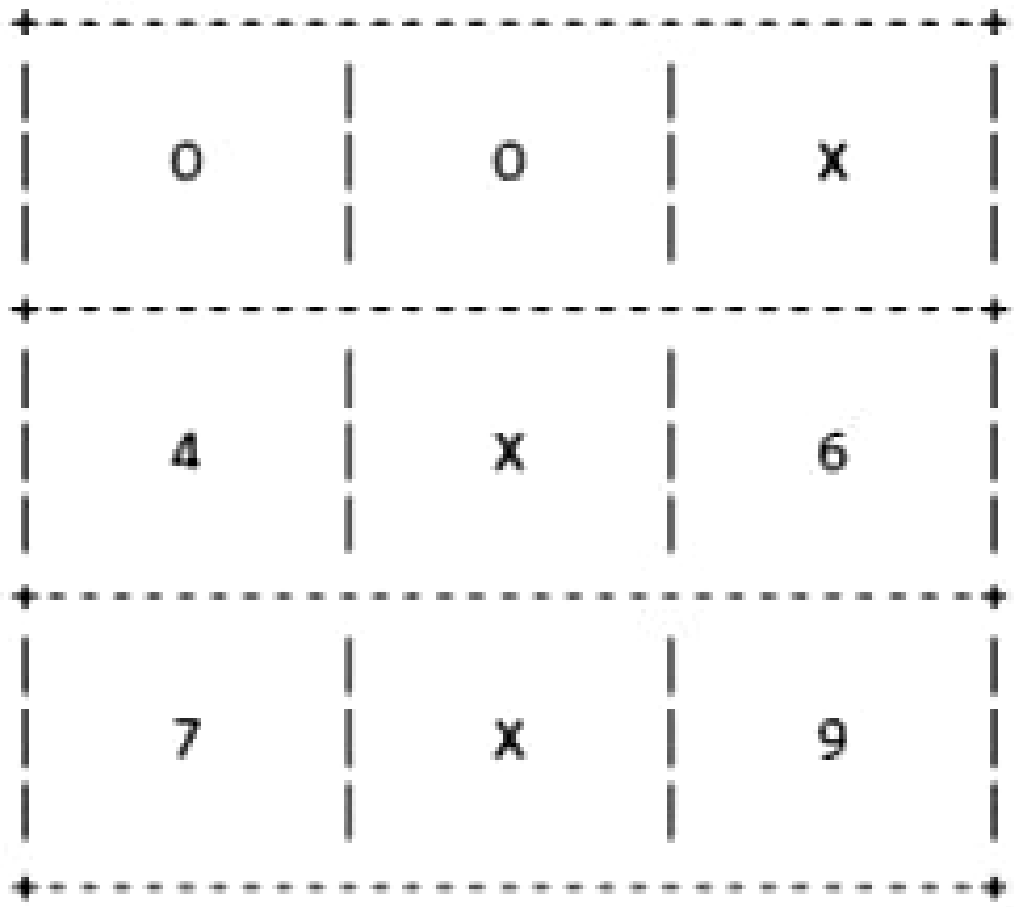
sorry O won this time!

```
Krishn Maloo - 1BM21CS092
[1, 2, 3, 4, 5, 6, 7, 8, 9]
+-------------------------------+
|       |       |       |
|   1   |   2   |   3   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   4   |   5   |   6   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------------------------------+
computer's turn :
+-------------------------------+
|       |       |       |
|   1   |   2   |   3   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   4   |   X   |   6   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------------------------------+
Your turn :
enter a number on the board :1
```

```
+---+---+---+
|   |   |   |
| 0 | 2 | 3 |
|   |   |   |
+---+---+---+
|   |   |   |
| 4 | X | 6 |
|   |   |   |
+---+---+---+
|   |   |   |
| 7 | 8 | 9 |
|   |   |   |
+---+---+---+
computer's turn :
+---+---+---+
|   |   |   |
| 0 | 2 | 3 |
|   |   |   |
+---+---+---+
|   |   |   |
| 4 | X | 6 |
|   |   |   |
+---+---+---+
|   |   |   |
| 7 | X | 9 |
|   |   |   |
+---+---+---+
Your turn :
enter a number on the board :2
```

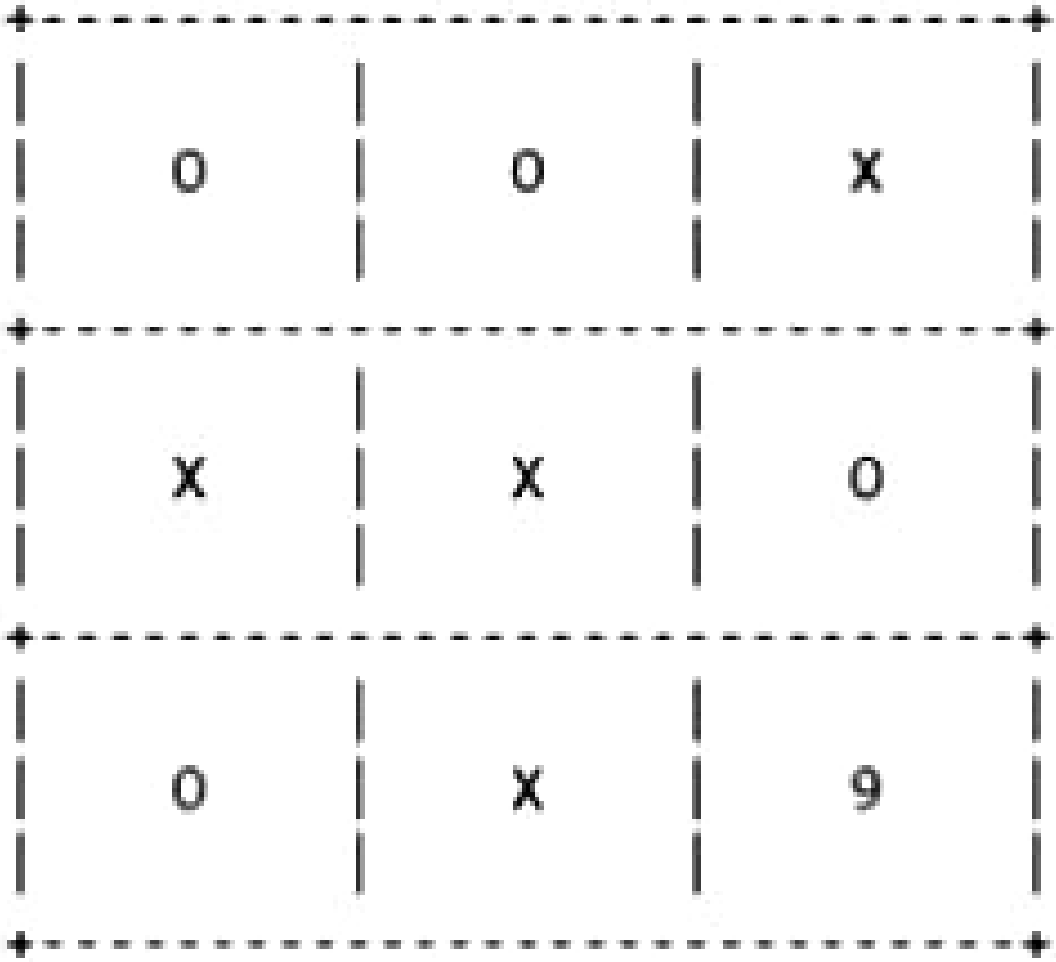


computer's turn :

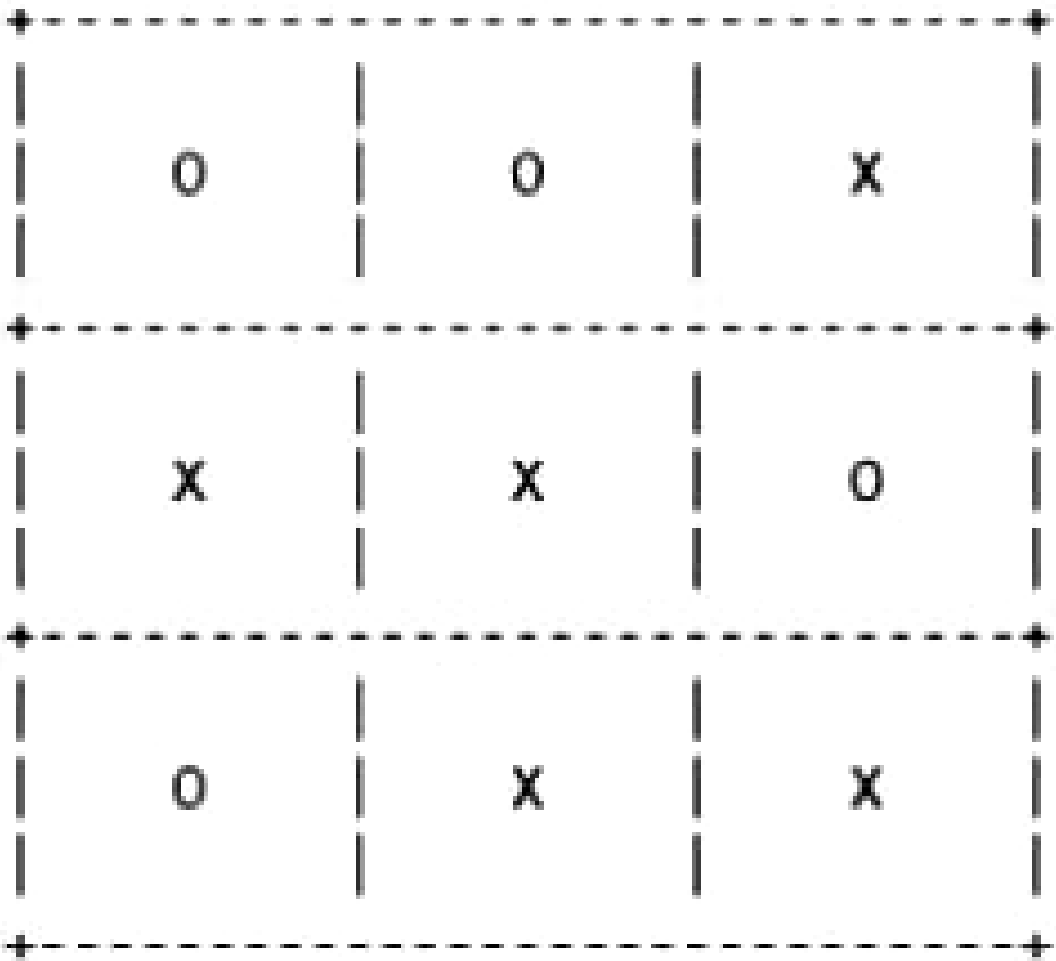


Your turn :
enter a number on the board :7

```
+---+---+---+
|   |   |   |
| O | O | X |
|   |   |   |
+---+---+---+
|   |   |   |
| 4 | X | 6 |
|   |   |   |
+---+---+---+
|   |   |   |
| O | X | 9 |
|   |   |   |
+---+---+---+
computer's turn :
+---+---+---+
|   |   |   |
| O | O | X |
|   |   |   |
+---+---+---+
|   |   |   |
| X | X | 6 |
|   |   |   |
+---+---+---+
|   |   |   |
| O | X | 9 |
|   |   |   |
+---+---+---+
Your turn :
enter a number on the board :6
```



computer's turn :



24/11/2023

LRA 2a-11-23

LAB 4 Game of 8 (BFS approach)

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

def bfs(src, target):
    queue = []
    queue.append(src)
    exp = [] //visited array
    while len(queue)>0:
        source = queue.pop(0)
        exp.append(source)
        print(source)
        if source == target:
            print("success")
            return.
    poss_moves_to_do = [] //possible array
    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) //queue mai
                                put possible (src
```

initial element

Breadth first search algorithm searches → way

```
def possiblemoves(state, visited_states):
    b = state.index(0) //blank space
    d = [] //array for all possible moves
    if b not in [0,1,2]: //when not at top then
        d.append('u')          only can move up
    if b not in [6,7,8]: //when not at bottom
        d.append('d')
    if b not in [0,3,1]: //when not in up
        d.append('l')
    if b not in [2,5,6]: //when not in right
        d.append('r')
```

```
possmovesit can = []
if ( ind :
    possmovesit can ( gen ( state, i, b)) // queuing states
return [ move_it_can for move_it_can in
        possmovesit 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.
```

updation based on move.

```
src = [1, 2, 3, 4, 5, 6, 0, 7, 8].
target = [1, 2, 3, 4, 5, 6, 7, 8, 0]
bfs (src, target).
```

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]
[1, 2, 3, 4, 5, 0, 7, 6, 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]
```

success.

Krishn Maloo - 1BM21CS092

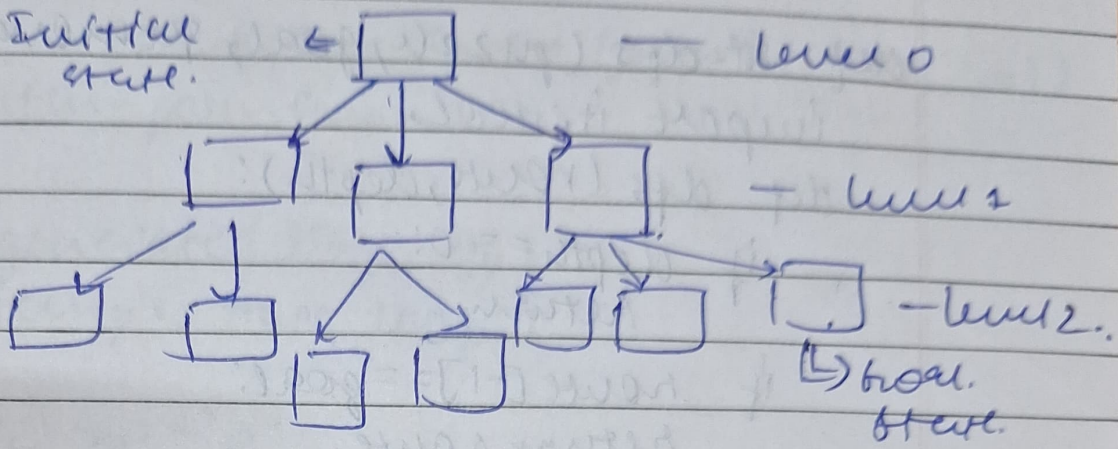
```
1 | 2 | 3
4 | 5 | 6
0 | 7 | 8

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

success
```

8/12/2023

8/12/23

Date
Page

* 8 Puzzle problem using IDDFS. (LAB-5)



1. Goal state and initial state provided.
2. Start with depth = 0 and increase depth by 1 on every iteration.
   & perform depth first search till that particular depth level.
3. Here generate function gives us possible moves by swapping the '0' tile.
4. We stop when we reach the goal state.
   & print the path followed.

Code:

```
def id-dfs (puzzle, goal, getmoves):
    import itertools
    def dfs (route, depth):
        if depth == 0:
            return
        if route[-1] == goal:
            return route
        for move in getmoves(route[-1]):
            if move not in route:
                nextmove = dfs (route + [move], depth-1)
                if nextmove:
                    return nextroute.

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

def possiblemoves (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').
```

```
    posmoves = []
    for i in d:
        pos_moves.append(generate(state, i, b))
    return posmoves.

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, 7, 5, 8]
goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]

route = id_dfs(initial, goal, possible moves)

if route:
    print("Success!!")
    print("Path:", route)
else:
    print("Failed to find a solution")
```

Output:

Success!!

Path: [[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]].

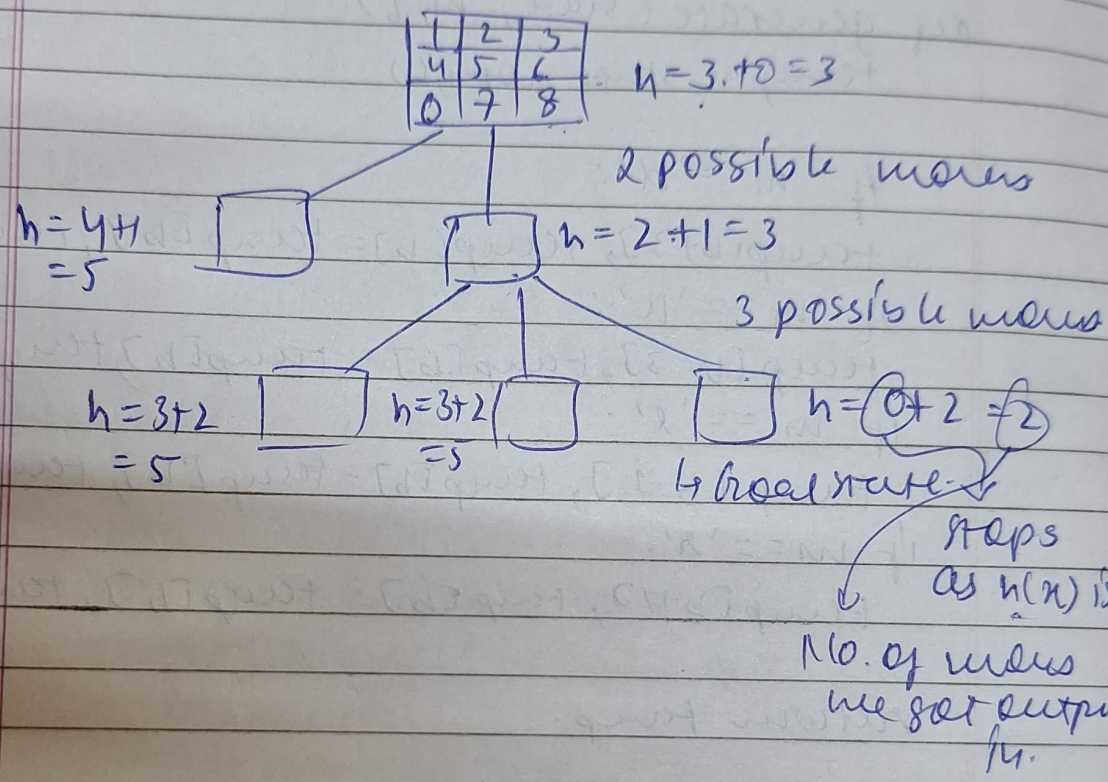
```
Krishn Maloo - 1BM21CS092
Success!! It is possible to solve 8 Puzzle problem
Path: [[1, 2, 3, 0, 4, 6, 7, 5, 8], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
```

* 8 Puzzle problem using A* Algorithm (LAB-5)

Goal state

1	2	3
4	5	6
7	8	0

h=0 (heuristics, i.e. no. of misplaced tiles)



1. Create initial and goal state for problem.
2. We calculate value at each step.
   fvalue = hvalue + pathcost
   ↳ no. of steps.
3. Goto the direction when fvalue is least.
4. All nodes stored in open list (keeps track of all the nodes) & sorted.
   Explored nodes stored in closed list.
5. Goal is when hvalue is 0, implies that we have reached final state.

```
Krishn Maloo - 1BM21CS092
Enter the start state matrix

1 2 3
4 5 6
_ 7 8
Enter the goal state matrix

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

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

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

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

A 22-12-23

22/12/23

* Vaccum Cleaner Agent (Lab-6)

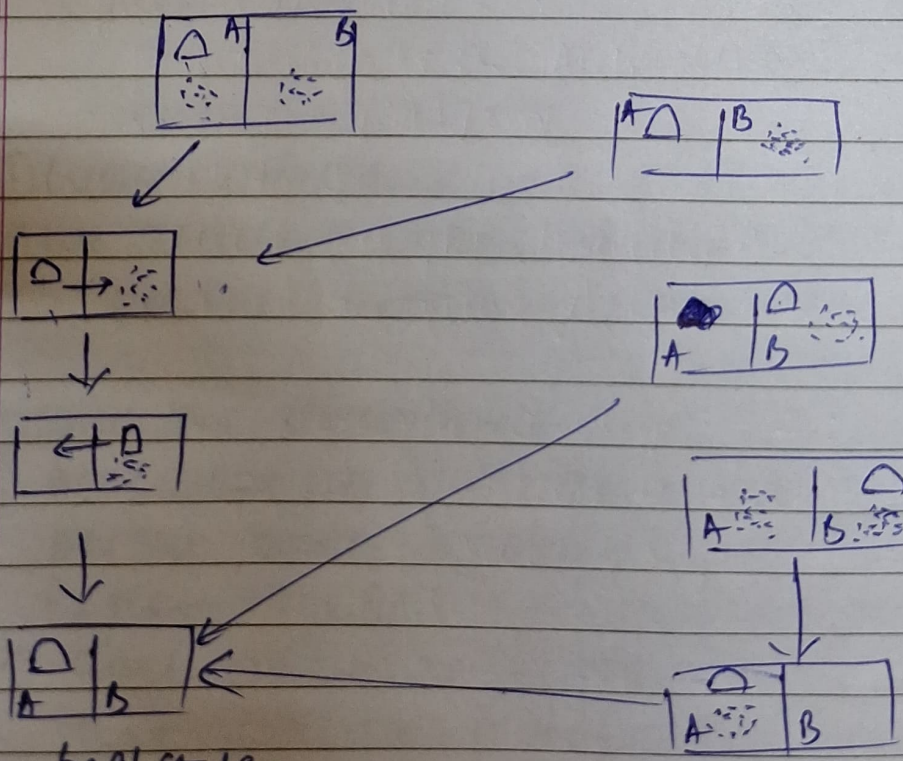
2 ROOMS

① Initialize the starting and goal state, goal is to clean both the rooms A & B.

② If status = dirty then clean.
else if location = A and status = clean
move right
else if location = B and status = clean
move left.
else exit.

③ If vaccum cleaner finds both the rooms to be clean then the task is completed.

Start state.



Goal state.

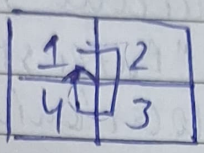
4 ROOMS

Date
Page

4 Rooms

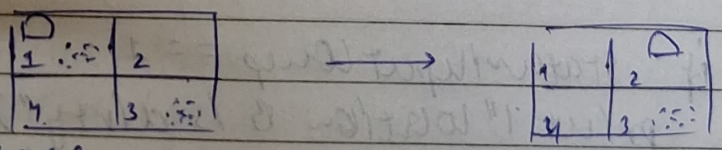
① Initialize the starting state and goal state, goal is to clean all the rooms A, B, C and D.

② Take input of each room from the user and location of vaccum cleaner.

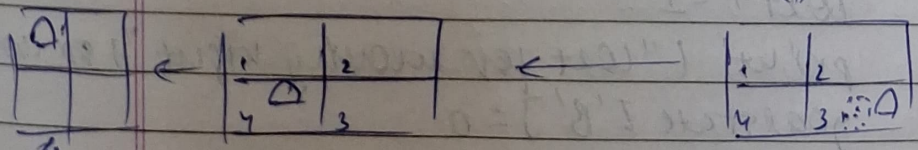


Room numbers and order in which the vaccum cleaner moves.

If status = dirty then clean.
else if location = 1 and status = clean.
move right
else if location = 2 and status = clean.
move down.
else if location = 3 and status = clean.
move left.
else if location = 4 and status = clean.
move up.
else if (all clean)
exit.



Start state.



Goal state

Move all the four rooms are cleaned.

③ If all rooms clean then task is completed

Code:

```
def vaccumworld():
    goalstate = {'A': 0, 'B': 0}  // final state
    cost = 0  // global variable to be summed

    locationInput = input("Enter location of
                           vaccum")
    statusInput = int(input("Enter status of"+
                            locationInput))
    statusInputComp = int(input("Enter status
                          of another room"))
    print("final location condition" + str(goalstate))

    if locationInput == 'A':
    print("Vaccum cleaner placed in location A")
    if statusInput == 1:
      print("location A is dirty").
      goalstate['A'] = 0
      cost += 1
      print("cost for cleaning A " + str(cost))
      print("location A has been cleaned.")

        if statusInputComp == 1:
          print("location B is dirty").
          print("moving right to location B")
          cost += 1
          print("cost for moving right " + str(cost))
          goalstate['B'] = 0
          cost += 1
          print("cost for suck " + str(cost))
          print("location B has been cleaned").
```

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

    if status_input == 0:
        print("location A is already clean")
        if status_input_comp == 1
            print("location B is dirty")
            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 has been cleared")
        else:
            print("No action" + str(cost).)
            print(cost)
            print("location B is already clean")

else:
    //same code.

print("final state:")
print(goal_state)
print(performance Measurement" + str(cost))

vacuum_world()
```

```
Krishn Maloo - 1BM21CS092
Enter clean status for Room 1 (1 for dirty, 0 for clean): 1
Enter clean status for Room 2 (1 for dirty, 0 for clean): 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.
Room 1 is clean after checking.
```

```
Krishn Maloo - 1BM21CS092
Enter clean status for Room at (1, 1) (1 for dirty, 0 for clean): 1
Enter clean status for Room at (1, 2) (1 for dirty, 0 for clean): 0
Enter clean status for Room at (2, 1) (1 for dirty, 0 for clean): 1
Enter clean status for Room at (2, 2) (1 for dirty, 0 for clean): 1
Cleaning Room at (1, 1) (Room was dirty)
Room is now clean.
Room at (1, 2) is already clean.
Cleaning Room at (2, 1) (Room was dirty)
Room is now clean.
Cleaning Room at (2, 2) (Room was dirty)
Room is now clean.
Returning to Room at (1, 1) to check if it has become dirty again:
Room at (1, 1) is already clean.
```

29/12/23

VRA 29/12/23

* Knowledge based Entailment (Lab-7)

Algorithm:

① Defining the knowledge base
We define the knowledge base provided.

② Checking entailment.
satisfiableAnd (knowledge base, not (query))
return not entailment.
↳ even if one element is true then it is satisfyable.
↳ mix of both.

③ Entailment is KB |= K

Inputs →
Knowledge base
Query statement.

Steps →
① Negate the query.
② Combine with knowledge base using conjunction.
③ Check satisfiability:
If conjunction is not satisfiable, it means there's no assignment of truth values that satisfies both knowledge base and negation of query.
④ Determine entailment.

* Knowledge base Resolution.

Algorithm ① create an instance of knowledge base class with an empty list of clauses
self.clauses = [].

② Adding a clause:
Append a new clause to the list of clauses in the knowledge base.

③ Resolving clause:-
Combine 2 classes by common literals (eliminating complementary literals)
Negate the query and add it to the knowledge base.

④ Repeatedly resolved the pairs of clauses in the knowledge base until. a contradiction found. or no new resolution are possible.

def resolve (self, clause_a, clause_b)
return [literal for literal in clause_a + clause_b)
if (not literal) not in clause a
↓
literal not in clause b.

```
Krishn Maloo - 1BM21CS092
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

Krishn Maloo - 1BM21CS092

Step	Clause	Derivation
1.	Rv~P	Given.
2.	Rv~Q	Given.
3.	~RvP	Given.
4.	~RvQ	Given.
5.	~R	Negated conclusion.
6.		Resolved Rv~P and ~RvP to Rv~R, which is in turn null.

A contradiction is found when ~R is assumed as true. Hence, R is true.

19/1/24

LAB-9-1-24

LAB-8

* Unification.

Step 1: If term1 or term2 is a variable or constant then:

a) term1 or term2 are identical
return NIL.

b) Else if term1 is a variable.
if term1 occurs in term2
return FAIL

else
return a (term2/term1)}

c) else if term2 is a variable.
if term2 occurs in term1.
return FAIL.

else.
return a (term1/term2)}

d) else return FAIL.

Step2: If predicate (term1) ≠ predicate (term2)
Return FAIL.

Step 3: number of args ≠ return FAIL.

Step 4: set (SUBST) to NIL.

Step 5: for i=1 to the number of elements in term1

a) call unify (term1, term2)
put result into S.

b) S = FAIL
return FAIL.

c) If S ≠ NIL.

a. Apply S to the remainder of both L1 and L2

b. SUBST = APPEND(S, SUBST)

Return SUBST.

knows(John, x) | knows(John, Jane)

S = [ ]

S = {[x/Jane]}

knows(x, Ram) knows(Laxman, x)

x y

x f(x)

* Unification - code.

```
import re
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:
            attributes[index]=new
    predicate = getInitialPredicate(exp)
    return predicate + "(" + ",".join(attributes)
                + ")"

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 [(exp2, exp1)]
    if isVariable(exp2):
        if checkOccurs(exp2, exp1):
            return False:
        else:
            return [(exp1, exp2)]
    if getInitialPredicate(exp1) != getInitialPredicate
                                    (exp2):
        print("Predicates do not match. Cannot be
                unified")
        return False.

    attributeCount1 = len(getAttributes(exp1))
    attributeCount2 = len(getAttributes(exp2))
    if attributeCount != attributeCount2:
        return False.
    head1 = getFirstPart(exp1)
    head2 = getFirstPart(exp2)
```

initial substitution = unify (head1, head2)
if not initial substitution:
    return false
if attributeCount != 1:
    return initialSubstitution
tail1 = getRemainingPart (exp1)
tail2 = getRemainingPart (exp2)
if initialSubstitution != []:
    tail1 = apply (tail1, initialSubstitution)
    tail2 = apply (tail2, initialSubstitution)

remainingSubstitution = unify (tail1, tail2)
if not remainingSubstitution:
    return false

initialSubstitution.extend (remainingSubstitution)
return initialSubstitution

```
Krishn Maloo - 1BM21CS092
Substitutions:
[('X', 'Richard')]
```

* FOL to CNF conversion.

Step 1: Create a list of skolem constants.

Step 2: Find ∀, ∃
If attributes → lower case.
Replace them with skolem constant and remove used skolem constant or function from the list.
If attributes are both lower case and uppercase replace the uppercase attribute with a skolem function.

Step 3: Replace (⇒) with '→'
transform → as $Q \Rightarrow (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

Step 4: Replace ⇒ with '→'.

Step 5: Apply demorgan's law
replace ~[
as ~P & ~Q if (was present)
replace ~[
as ~P | ~Q if (was present)
replace ~~ with ''.

$\forall x \quad P_{xy} \cdot King(x) \Rightarrow Person(x)$

* FOL TO CNF - Code.

```
def getattributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return [m for m in str(matches) if m.isalpha()]
def getpredicates(string):
    expr = '[a-z~]+\([A-Za-z,]+\)'
    return findall(expr, string)
def Demorgan(sentence):
    string = ''.join(list(sentence).copy())
    string = string.replace("~~", "")
    flag = '[' in string
    string = string.replace('~[', "")
    string = string.strip(']')
    for predicate in getpredicates(string):
        string = string.replace(predicate, f'~{predicate}')
    s = list(string)
    for i, c in enumerate(string):
        if c == '|':
            s[i] = '&'
        elif c == '&':
            s[i] = '|'
    string = ''.join(s)
    string = string.replace('~~', '')
    return f'[{string}]' if flag else string

def skolemization(sentence):
    skolem_constants = [f'{chr(c)}' for c
            in range(ord('A'), ord('Z')+1)]
    statement = ''.join(list(sentence).copy())
    matches = re.findall('[∀∃].', statement)
```

```
        for match in matches[::-1]:
            statement = statement.replace(match,'')
            statement = re.findall('\[\[^\]]+\]',statement)
        for s in statement:
            statement = statement.replace(s,s[1:-1])

    return statement

import re
def foltocnf(fol):
    statement = fol.replace("<=>","-")
    while '-' in statement:
        i = statement.index('-')
        newstatement = '[' + statement[:i] + '=>' +
        statement[i+1:] + '&[' + statement[i+1:] +
        '=>' + statement[:i] + ']'
        statement = newstatement
    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,foltocnf(s))
    while '-' in statement:
        i = statement.index('-')
        br = statement.index('[') if '[' in
        statement else 0
        newstatement = '~' + statement[br:i] + '|' +
                statement[i+1:]
        statement = statement[:br] + newstatement
                if br>0 else newstatement
```

```
while '~∃' in statement:
    i = statement.index('~∃')
    S = list(statement)
    statement[i], statement[i+1], statement[i+2]
    = '∃', statement[i+2], '~'
    statement = ''.join(statement)
statement = statement.replace('~[A', '[~∀')
statement = statement.replace('~[', '[~∃')
expr = '(~[A|∃].)'
statements = re.findall(expr, statement)
for s in statements:
    statement = statement.replace(s, toltoy(s))
expr = '~\([^]]+\]'
statements = re.findall(expr, statement)
for s in statements:
    statement = statement.replace(s, toltoy(s))
expr = '~\([[^]]+\]'
statements = re.findall(expr, statement)
for s in statements:
    statement = statement.replace(s, toltoy(s))
for s in statements:
    statement = statement.replace(s, Denoyau(s))
return statement.
```

Krishn Maloo - 1BM21CS092

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

[animal(G(x))&~loves(x,G(x))]|[loves(F(x),x)]

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

23/1/24 IRA 24-1-24

LAB-8

* forward reasoning.

```
import re
def Variable(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-z^]+)\([^&|]+\)'
    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 splitExpressions(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
        ([c.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 = {}
        newlhs = []
        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]
                    newlhs.append(fact)
        predicate, attributes = getPredicate(self.rhs.expr)
                str(getAttributes(self.rhs.expression[0]))
        for key in constants:
            if constants[key]:
                attributes = attributes.replace(key,
                                constants[key])
```

```
        expr = f'{predicate}({",".join(attributes)})'
        return Fact(expr) if len(matches) and
        all([f.getresult() for f in matches]) else None

class KB:
    def init(self):
        self.facts = set()
        self.implications = set()
    def tell(self, e):
        if '=>' in e:
            self.implications.add(Implication(e))
        else:
            self.facts.add(Fact(e))
        for i in self.implications:
            res = i.evaluate(self.facts)
            if res:
                self.facts.add(res)
    def query(self, e):
        facts = set([f.expression for f in self.facts])
        i = 1
        print(f'Querying {e}:')
        for f in facts:
            if Fact(f).predicate == Fact(e).predicate:
                print(f'\t{i}. {f}')
                i += 1
    def display(self):
        print("All facts:")
        for i, f in enumerate(set([f.expression
                for f in self.facts])):
            print(f'\t{i+1}. {f}')
```

Output:

```
kb = KB()
kb.tell('king(x) & greedy(x) => evil(x)')
kb.tell('king(John)')
kb.tell('greedy(John)')
kb.tell('king(Richard)')
kb.query('evil(x)')
```

Query evil(x):

1. evil(John)

```
Krishn Maloo - 1BM21CS092
Querying criminal(x):
        1. criminal(West)
All facts:
        1. criminal(West)
        2. hostile(Nono)
        3. weapon(M1)
        4. missile(M1)
        5. sells(West,M1,Nono)
        6. enemy(Nono,America)
        7. owns(Nono,M1)
        8. american(West)
```