Date : 17-11-2023 KRA 17-11-23

Program 1 : Implement Tic Tac Toe

```
import random
board = [' ' for i in range(10)]

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

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

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

Date :

```
def is_winner(board, len):
return(board[1] == len and board[2] == len and board[3] ==
                    or
 (board[4] == len and board[5] = len and board[6] == len)
                    or
 (board[7] == len and board[8] == len and board[9] = len)
                    or
 (board[1] == len and board[4] == len and board[7] = len)
                    or
 (board[2] == len and board[5] == len and board[8] == len)
                    or
 (board[3] == len and board[6] == len and board[9] == len)


def player1():
    run = True
    while run:
        move = input("Enter a position to place an 'X' (1-9)")
        try:
            move = int(move)
            if move > 0 and move < 10:
                if space_free(move):
                    run = False
                    insert('X', move)
                else:
                    print("Occupied")
            else:
```

Date: 17-11-2023

```
def CompMove():
    possible = [x for x, letter in enumerate(
        board) if letter == " " and x != 0]
    move
```

```
def compmove():
    run = True
    while run:
        move = random.randint(1,10)
        if (move > 0 and move <10):
            if spaceIsFree(move):
                run = False
                insertLetter('O', move)
            else:
                continue
        else:
            continue
```

```
def main():
    print("Tic Tac Toe Game")
    printBoard(board)
    if board.count(' ') > 1:
        if not (isWinner(board, 'O')):
            playerMove()
            printBoard(board)
        else:
            print("Sorry O won\n")
            break
```

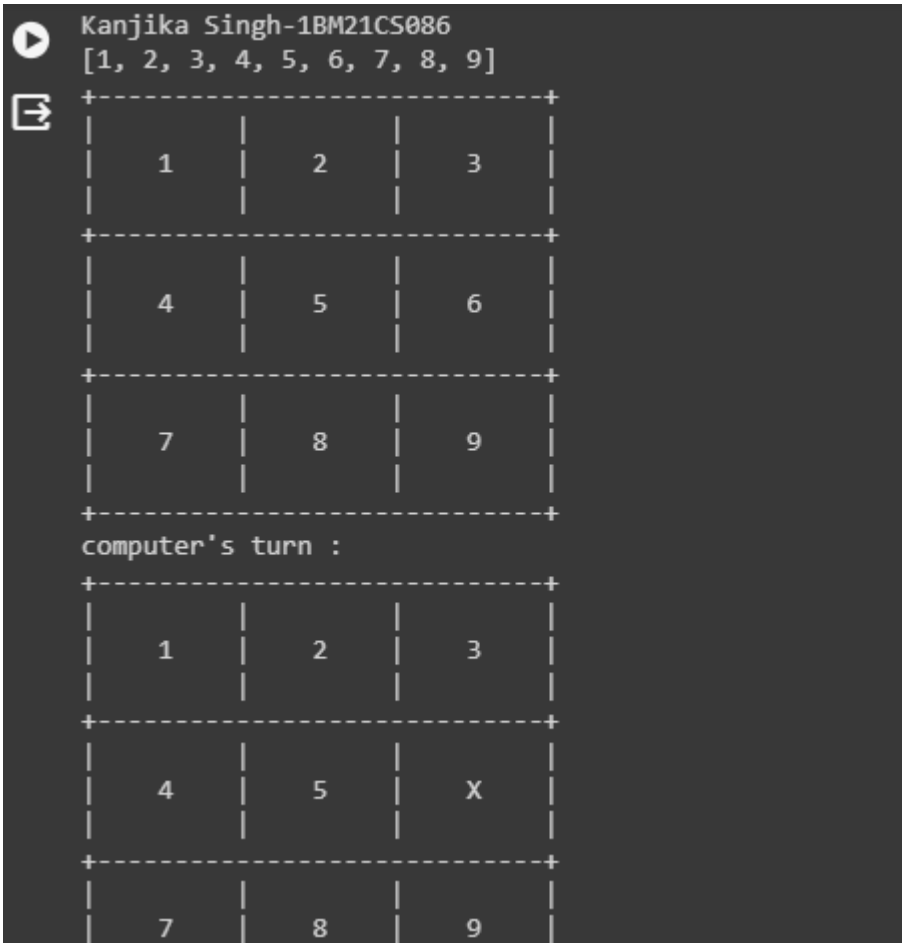
Date : 17-11-2023

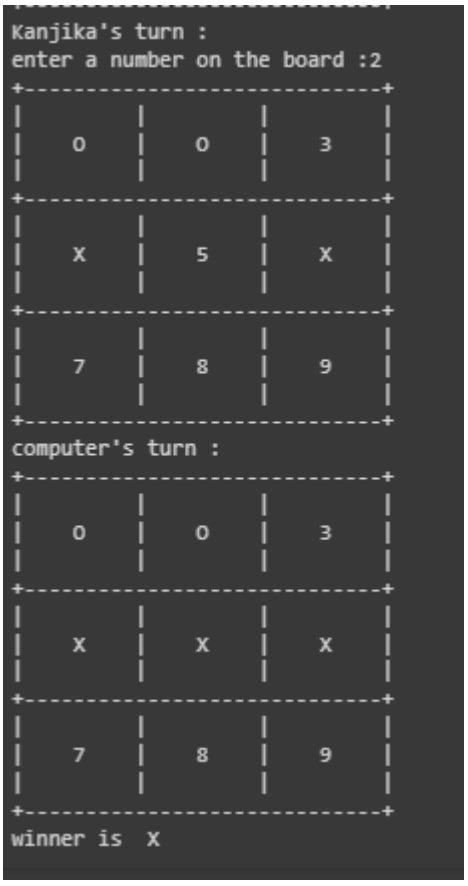
```
if not (isWinner(board, 'x')):
    move = compMove()
    if move == 0
    print("Tie game)"
    else:
        insert('0', move)
        print

if not (board.count(' ') <10):
    playerMove()
    printBoard(board)
    if (is winner(board, 'x')):
        print('you won')
        break
    else else:
        compMove()
        print Board(board)
        if (isWinner(board, '0')):
            print("Computer won")
            break
    else:
        print("Tie this is")
```

Date : 17-11-2023

Algorithm : Tic Tac Toe

- Create a 3×3 board consisting of empty space
- Create function insert() to insert a letter to the board and space_free() to check if position is free
- First allow player to play
  - If the board is free, insert X
  - Then check if move leads to the player to win or not
  - If the player does not wins, give computer the chance to play
- Continue till the board is empty.





Date : 24/11/23

24-11-23

8 Puzzle Problem Using BFS

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

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

In the problem, initial state will be given and and we have do reach goal state.

Suppose

Initial state

1	2	3
	4	6
7	5	8

Goal state

1	2	3
4	5	6
7	8	

Rules

- Empty space can move in 4 directions
  1) Up 2) Down 3) Right 4) Left
- It cannot move diagonally
- It can only take one step at a time

o	x	o
x	#	x
o	x	o

Tiles at o — no. of possible moves=2
Tiles at x — no of possible moves=3
Tile at # — no. of possible moves=4

Using Breadth first search, which is uninformed search. This problem is solved by non heuristic approach.

Complexity O(b^d) where
b — branching factor
d — depth

Date : 24/11/23

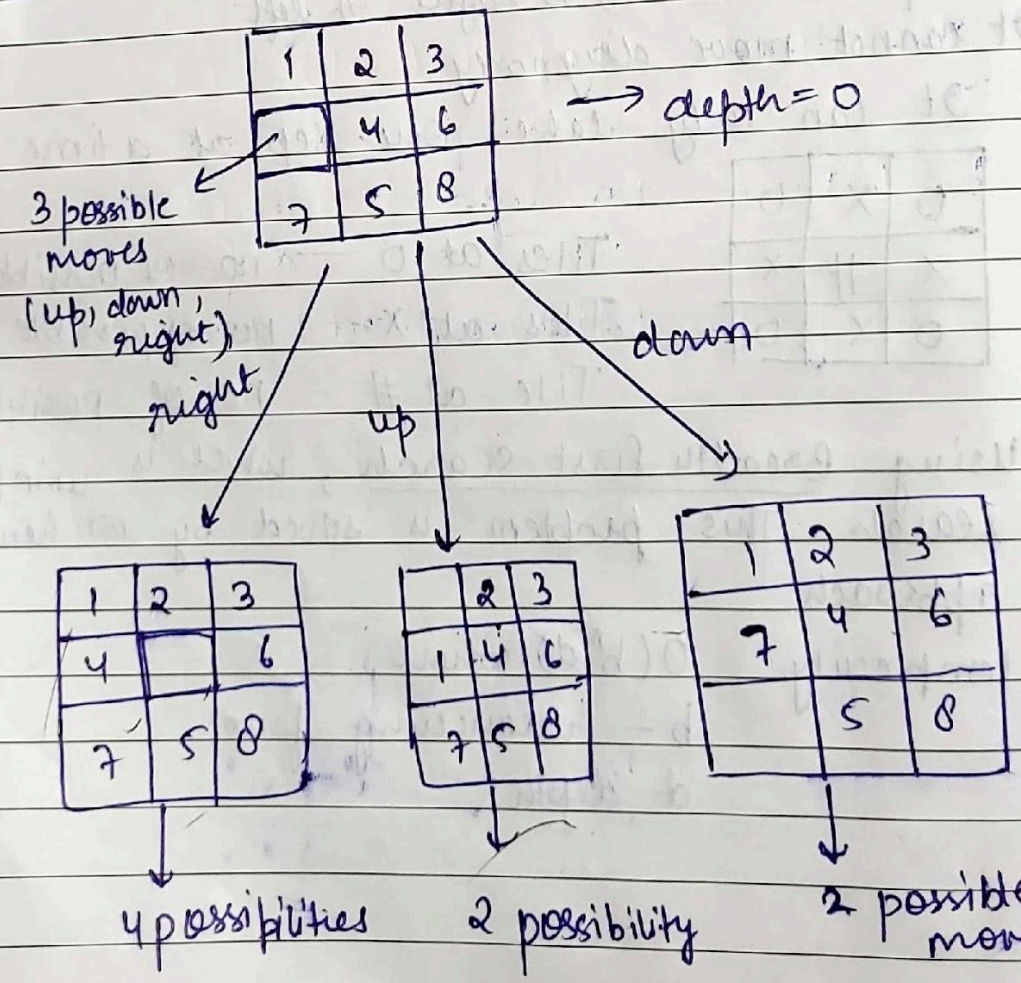
For 8 puzzle

Branching factor b = $\frac{\text{all possible moves of empty tile at each position}}{\text{no. of tiles}}$

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

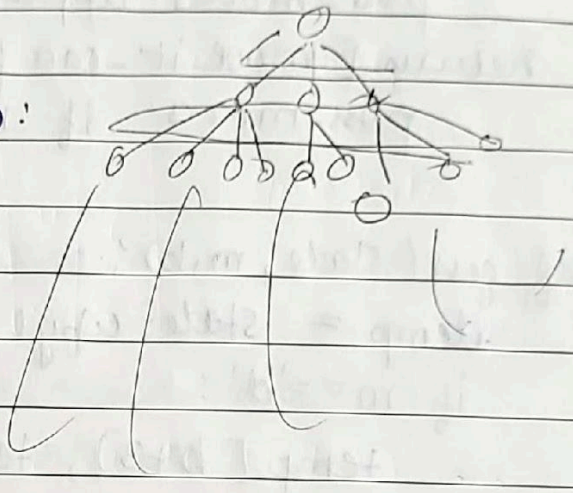
depth factor = initially 0
As we explore the nodes, depthfactor increases
Every time, we check all possible moves of tiles branch it and consider every case and finally to goal state

For example:



Date : 24/11/23

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

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

def possible (state, visited ):
    b = state. index(0)  // 0 indicates index of white space
                          so b has index of it
    d = [ ]
    if b not in [0,1,2]:
        d. append ('u')
    if b not in [6,7,8]:
      d. append ('d')
```



Date : 24/11/23

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


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.
```

```
Kanjika Singh-1BM21CS086
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
```

6/12/23 A8(12)23

8 Puzzle problem using ID-DFS

Code:

```
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+[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]:
```

Date :

```
        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_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 = [1,2,3,4,5,6,7,8,0]
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(fail)
```

e :

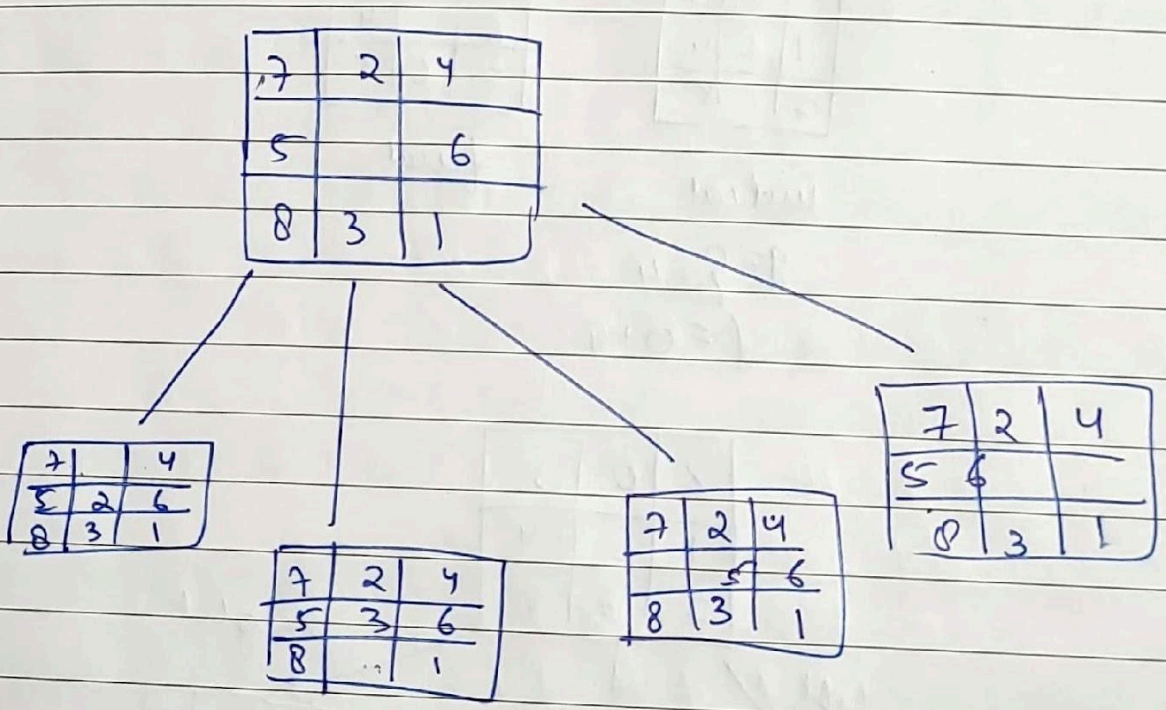
ID_DFS Combination of DFS and BFS
- DFS in BFS manner.

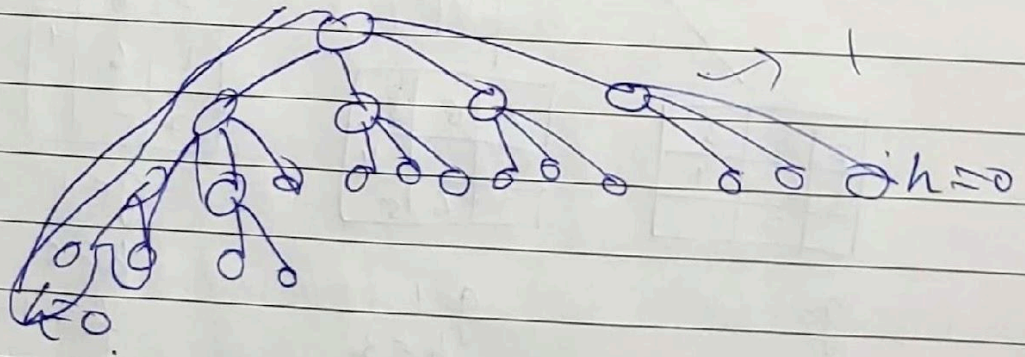
7	2	4
5		6
8	3	1

initial

	1	2
3	4	5
6	7	8

goal





```
Kanjika singh-1BM21CS086
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]]
```

Date : 8/12/2023

8 Puzzle Problem using A* Algorithm

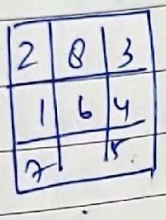
Finds most cost effective path to reach the final state from initial state

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

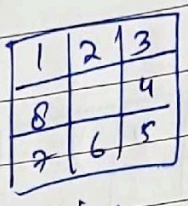
$g(n) \rightarrow$ depth of node

$h(n) \rightarrow$ no. of misplaced tiles

Suppose



initial

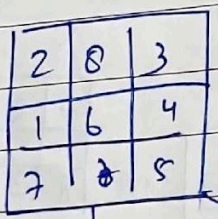


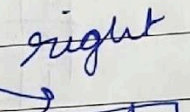
final

$g = 0$

$h = 4$

$f = 0 + 4$

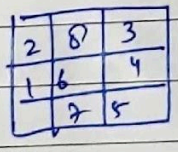


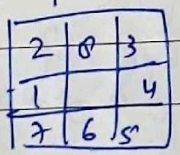


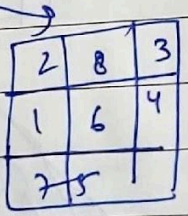
right

left

up







$g = 1$

$h = 5$

$f = 1 + 5 = 6$

$g = 1$

$h = 3$

$f = 1 + 3 = 4$

$g = 1$

$h = 5$

$f = 1 + 5 = 6$

Date :

```
Class Node:
    def __init__(self, data, level, fval):
        """ Initialise the node with datato"""
        self.data = data
        self.level = level
        self.fval = fval.

    def generate_child(self):
        x, y = self.find(self.data, '_')
        val_list = [[x, y-1], [x, y+1], [x-1, y], [x+1, y]]
        children = []
        for i in val_list:
            cuis = self.shuffle(self.data, x, y, i[0], i[1])
            if child is not None:
                child_node = Node(child, self.level+1, 0)
                children.append(child node)
        return children.

    def shuffle(self, puz, x1, y1, x2, y2):
        if x2 >=0 and x2 < len(self.data) and y2 >=
            and y2 < len(self.data):
                temp_puz = []
                temp_puz = self.copy(puz)
                temp = temp_puz[x2][y2]
                temp_puz[x2][y2] = temp_puz[x1][y1]
                temp_puz[x1][y1] = temp
                return temp_puz
        else:
            return None.
```

Date :

```
def find (self, puz, x):

    for i in range (0, len(self.data)):
        for j in range (0, len(self.data)):
            if puzz[i][j] == x:
                return i,j


def copy (self, root):
    temp = []
    for i in root:
        t = []
        for j in i:
            t.append(j)
        temp.append(t)
    return temp

class Puzzle:
    def __init__(self,size):
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puzz
```

```
def f(self, start, goal):
    return self.h(start.data, goal) + start.level

def h(self, start, goal):
    temp = 0
    for i in range(0, self.n):
        for j in range(0, self.n):
            if start[i][j] != goal[i][j] and
                    start[i][j] != '_':
                temp += 1
    return temp

def process(self):
    print("Enter the start state matrix \n")
    start = self.accept()
    print("Enter goal state \n")
    goal = self.accept()
    start = Node(start, 0, 0)
    start.fval = self.f(start, goal)
    self.open.append(start)
    print("\n")
    while True:
        cur = self.open[0]
        print("")
        print("|")
        print(" | ")
        print("\\|/ \n")
        for i in cur.data:
            for j in i:
```

Date :

```
            print(j, end=" ")
        print("")
    if(self.h(cur.data, goal) ==0):
        break
for i in cur.generate_child():
    i.fval = self.f(i, goal)
    self.open.append(child)
    self.closed.append(cur)
del self.open[0]

self.open.sort(key = lamda x:x.fval,
                reverse = False)


puz = Puzzle(3)
puz.process()
```

Output

```
Enter start matrix
1 2 3
- 4 6
7 5 8
Enter the goal state
1 2 3
4 5 6
7 8 -
  1
  |
  v
1 2 3
  4 6
- 7 5 8
```

Date :

1 2 3

4 - 6

7 5 8

1 2 3

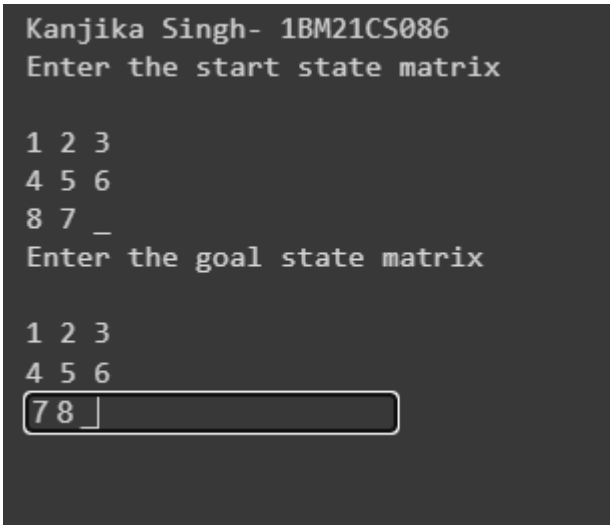
4 5 6

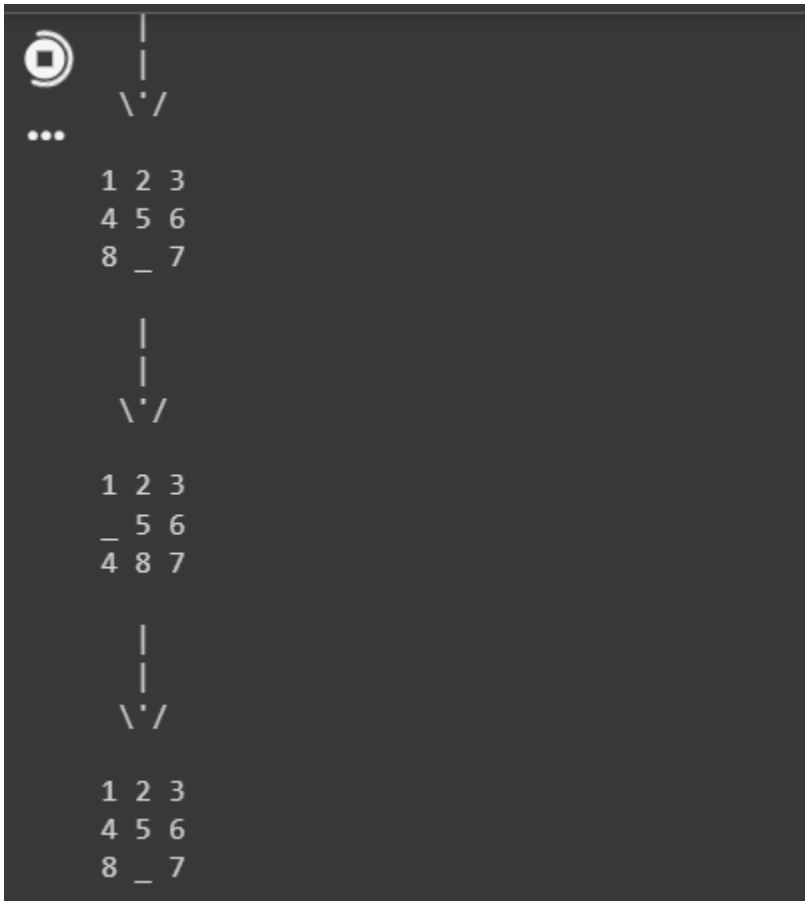
7 - 8

1 2 3

4 5 6

78 -





Date : 22-12-2023 22-12-23

Vacuum Cleaner Problem

Algorithm and code

We assume that vacuum cleaner cannot jump directly to any position. So for all even rows, vacuum cleaner moves from right to left

logic

```
def clean(floor, room):
    if room = 0 goto room2
    for i in range(len(floor)):
        if i%2 == 0:
            for j in range(len(floor[i])):
                if floor[i][j] == 1:
                    print_f(floor, i, j)
                    floor[i][j] = 0
            print_f(floor, i, j)
        else:
            for j in range(len(floor[i])-1, -1, -1):
                if floor[i][j] == 1
                    print_f(floor, i, j)
                    floor[i][j] = 0
            print_f(floor, i, j)
    if all(floor[i][j] == 0 in room
        then room = 0.
```

Date :

```
def print_f(floor, row, col):
    print("The floor matrix is below")
        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
```

```
def main():
    floor = [ ]
    r = int(input("Enter number of room"),
    m = int(input("Enter rows"))
    p = int(input(" Enter no. of rows"))
    print("Enter clean status for each ~~rr~~ ceel")
    for i in range(m):
        f = list(map(int, input().split(" ")))
        floor.append(f)
    print()
    clean(floor, room)
```

Two rooms

logic

- Input for rooms 2 we take
- Initially, start from room 1 and inspect every grid
- If room is clean, [room1] = 0, goto room 2.

```
Kanjika Singh-1BM21CS086
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): 1
Cleaning Room 1 (Room was dirty)
Room 1 is now clean.
Cleaning Room 2 (Room was dirty)
Room 2 is now clean.
Returning to Room 1 to check if it has become dirty again:
Room 1 is already clean.
Room 1 is clean after checking.
```

Date :

Room 1

0	1	1
0	0	1
1	0	0

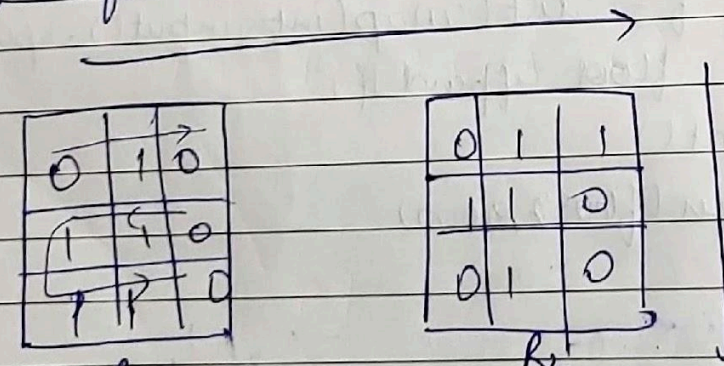
Room 2

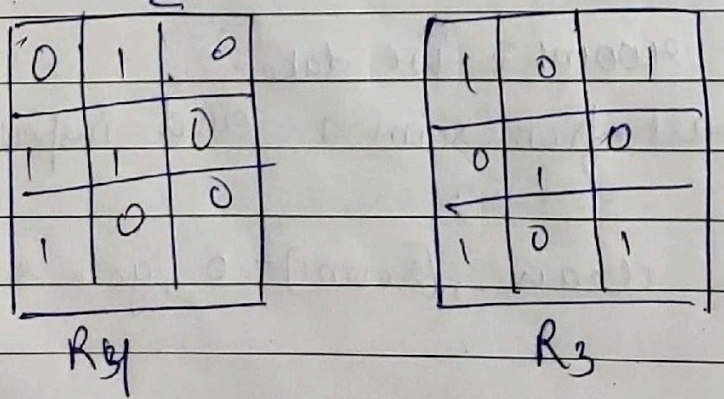
0	1	0
0	0	0
0	0	1

Room 1 → dirty → clean
check if room2 is completely clean (all grids 0)
if clean,
move to room 2
If room 2 → dirty
clean it by calling 'clean' function
If completely clean return and exit

Room1: status clean → move to Room 2

for four rooms





```
Kanjika Singh-1BM21CS086
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): 0
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.
Room at (2, 2) is already clean.
Returning to Room at (1, 1) to check if it has become dirty again:
Room at (1, 1) is already clean.
```

Date : 29/12/23

URA 29/12/23

Knowledge Based Entailement

```
// from Sympy import symbols, And, Not, Implies, satisfiable
// not used

def create knowledge_base():
    p = symbols('p')
    q = symbols('q')
    r = symbols('r')
    knowledgebase = And(Implies(p, q), Implies(q, r),
                        Not(r))
                    # (p→q) ∧ (q→r) ∧(¬r)
    return knowledgebase

def query_entaild knowledge_base, query):
    entailment = satisfiable(And(knowledge_Base, Not
    // return not entailment(Query)))
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 knowledgebase,' result)
```

def implies(q, r): if pq then r

return all(expr for expr in kb) and not query

// $\alpha \models \beta$ i.e. alpha (α) is said to entail β, if in every model where α is true, β is true

A give logic/argument is said to be satisfiable if it satisfies for some logic

$\alpha \models \beta$ if $\alpha \rightarrow (\alpha \wedge \beta)$

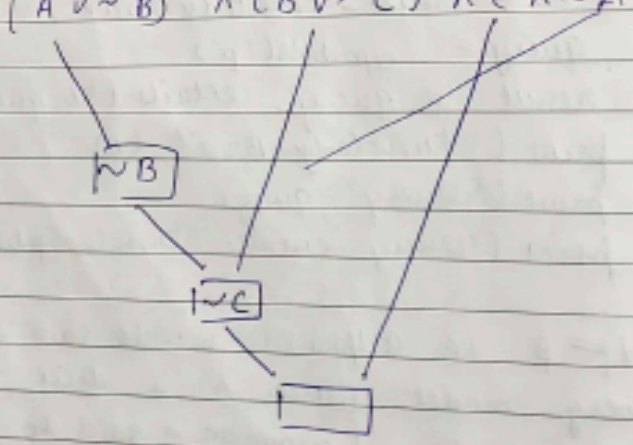
```
Kanjika Singh-1BM21CS086
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

Date : 29/12/23

Knowledge Based 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 c1
        if negate_literal(literal) in c2:
            resolved_clause.remove(literal)
            resolved_clause.remove(negate_literal(literal))
    return tuple(resolved_clause)
```

KB : $(A \vee \sim B) \wedge (B \vee \sim C) \wedge C \wedge \sim A$



```
Kanjika Singh-1BM21CS086

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.
```

```
Kanjika Singh-1BM21CS086

Step    |Clause |Derivation
-----------------------------
 1.     | PvQ   | Given.
 2.     | ~PvR  | Given.
 3.     | ~QvR  | Given.
 4.     | ~R    | Negated conclusion.
 5.     | QvR   | Resolved from PvQ and ~PvR.
 6.     | PvR   | Resolved from PvQ and ~QvR.
 7.     | ~P    | Resolved from ~PvR and ~R.
 8.     | ~Q    | Resolved from ~QvR and ~R.
 9.     | Q     | Resolved from ~R and QvR.
 10.    | P     | Resolved from ~R and PvR.
 11.    | R     | Resolved from QvR and ~Q.
 12.    |       | Resolved R and ~R to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

```
Kanjika Singh-1BM21CS086

Step    |Clause |Derivation
-----------------------------
 1.     | PvQ   | Given.
 2.     | PvR   | Given.
 3.     | ~PvR  | Given.
 4.     | RvS   | Given.
 5.     | Rv~Q  | Given.
 6.     | ~Sv~Q | Given.
 7.     | ~R    | Negated conclusion.
 8.     | QvR   | Resolved from PvQ and ~PvR.
 9.     | Pv~S  | Resolved from PvQ and ~Sv~Q.
 10.    | P     | Resolved from PvR and ~R.
 11.    | ~P    | Resolved from ~PvR and ~R.
 12.    | Rv~S  | Resolved from ~PvR and Pv~S.
 13.    | R     | Resolved from ~PvR and P.
 14.    | S     | Resolved from RvS and ~R.
 15.    | ~Q    | Resolved from Rv~Q and ~R.
 16.    | Q     | Resolved from ~R and QvR.
 17.    | ~S    | Resolved from ~R and Rv~S.
 18.    |       | Resolved ~R and R to ~RvR, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

Date: 19/1/2024

19-1-24

Unification

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

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(exp2)
        if checkOccurs(exp2, exp1)
            return False
        else
            return [(exp1, exp2)]
    if getInitialPredicate(exp1) != getInitialPredicate
        print("Predicates don't match!") (exp2)
        return False
```

Date :

```
attribute count 1 = len(getAttributes(exp1))
if attribute count 1 = attribute count 2:
    return false
head1, head2 = getFirstPart(exp1), getFirstPart(exp2)
initial_substitution = unify(head1, head2)
    if not initialSubstitution:
        return false.
    if attribute count 1 == 1:
        return initialSubstitution
tail 1 = getRemaining(exp1)
tail 2 = getRemaining(exp2)

remaining substitution = unify(tail1, tail 2)
    if not remainingSubstitution
        return false.
initialSubstitution.extend(remaingSolution)
return initialSubstitution
exp1 = 'knows(x)'
exp2 = 'knows(Richard")
Substitutions = unify(exp1, exp2)
print(Substitutions:)
```

Output

```
[('x', 'Richard')]
```

```
Kanjika Singh-1BM21CS086
Substitutions:
[('X', 'Richard')]
```

```
[6] exp1 = "knows(A,x)"
    exp2 = "knows(y,mother(y))"
    substitutions = unify(exp1, exp2)
    print("Substitutions:")
    print(substitutions)
```

```
Substitutions:
[('A', 'y'), ('mother(y)', 'x')]
```

e: 19/1/2024

FOL to CNF

```
import re:
def fol_to_cnf (fol):
    statement = fol.replace("<=>", '_')
    while '_' in statement:
        i = statement.index('_')
        newstatement = '[' + statement[:i] +
                       '=>' + statement[i+1:]
                    + ']' + '[' + statement[:i]
                + '=>' + statement [:i]
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 statements:
    i = statement.index('-')
    br = statement.index('[') if '[' in statement
    new_statement = '~' + statement[br:i] +
                    '|' + statement[i+1:]
    statement = statement[:br] + new_state
        ment if br > 0 else
    new_statement
```

Date :

while '~∀' in statement:
    i = statementindex ('~∀')
    statement = list(statement)
    statement[i], statement[i+1], statement[i+2] = '[', statement[i+2], ']'
    statement = ''.join(s)

statement = list(statement)
statement = statement.replace('~[∃', '[~∀')
expr = '[[∀|∃].]'

for s in statements:
    statement = statement.replace(s, Demorgan(s))

return statement

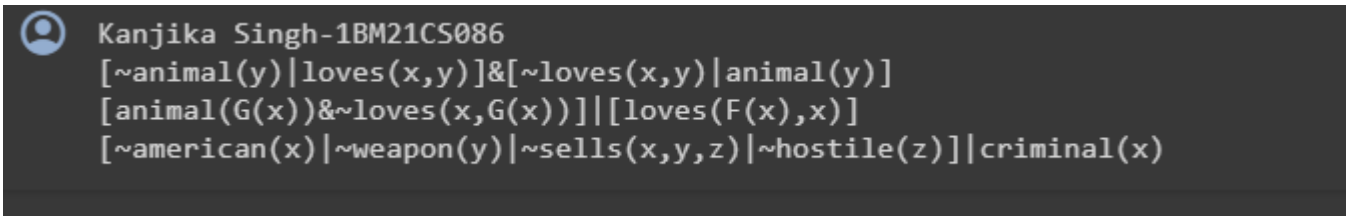
print (CNFConversion(fol_to_cnf('animal(y) ⇒ loves(x,y)')))

_____ " _____ ('∀x [∀y [animal(y) ⇒ loves(x,y)]]) ⇒ (∃z [loves(z,x)])

Output:

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

[animal(G(x)) & ~loves(x, G(x))] | [[loves(z,x) | ~animal(x) | ~weapon(y) | ~sell(x,y,z) | ~hostile(z)] | criminal(x)]



Date: 19/1/24

Forward Reasoning

```
def getPredictions(string):
    exps = '([a-z~]+)\([^&|]+\)'
    return re.findall(exps, string)

class Implication:
    def __init__(self, expression):
        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:
                            constant[v] = fact.getConstants()[i]

                    new lhs.append(fact)

class Kb:
    def tell(self, e):
        if '=>' in 'e'
            self.implication.add(Implication(e))
```

Date : _____

```
def display(self):
    print("All facts:")
    for i, f in enumerate(set([f.expression
        for f in self.facts])):
        print(f'{i+1}. {f}')

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)')
```

```
Kanjika Singh-1BM21CS086
Querying criminal(x):
        1. criminal(West)
All facts:
        1. hostile(Nono)
        2. weapon(M1)
        3. enemy(Nono,America)
        4. sells(West,M1,Nono)
        5. criminal(West)
        6. owns(Nono,M1)
        7. missile(M1)
        8. american(West)
```

```
Kanjika Singh-1BM21CS086
Querying evil(x):
        1. evil(John)
```