VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

ARTIFICIAL INTELLIGENCE

Submitted by

KEERTHI P REDDY
(1BM21CS090)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Nov-2023 to Feb-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

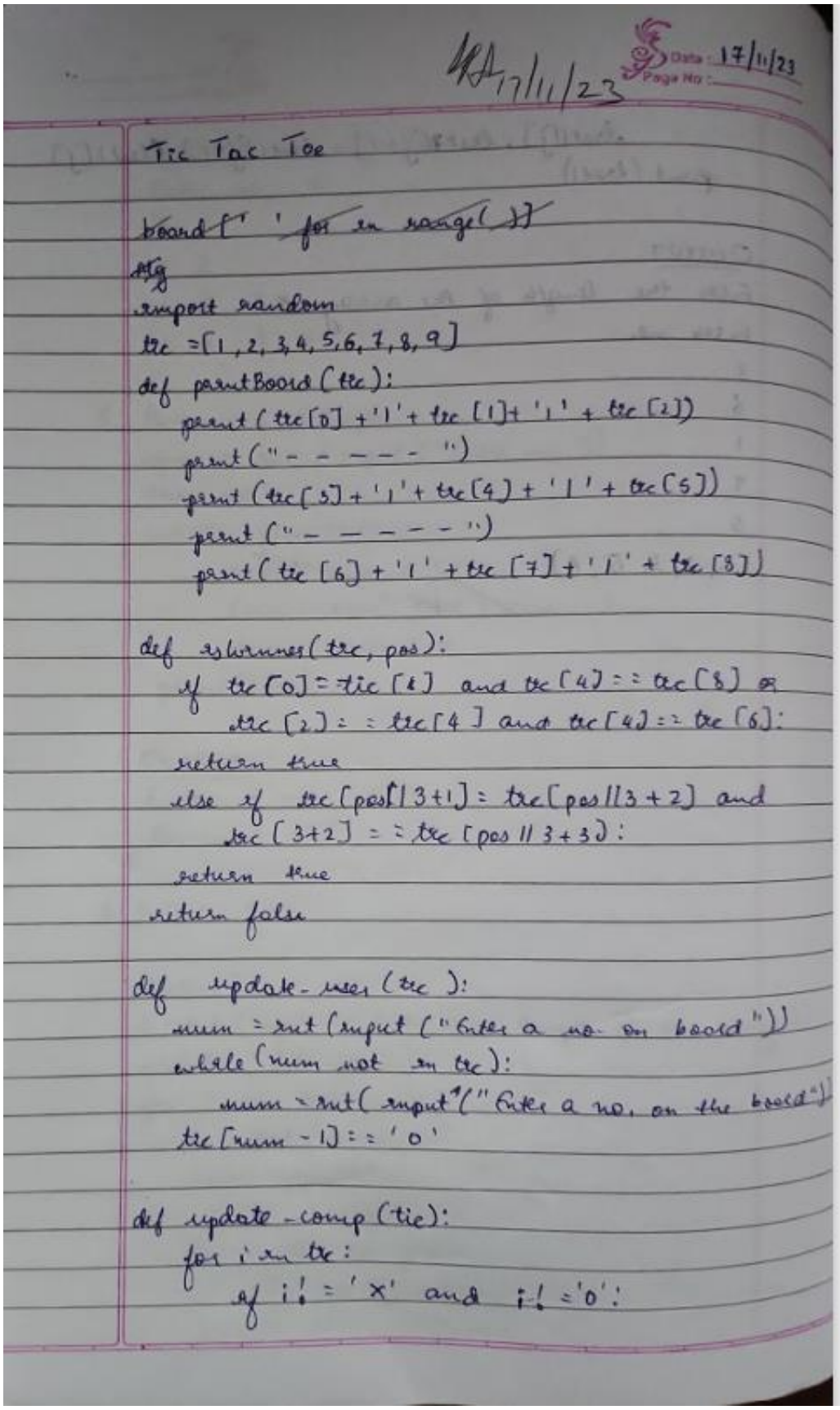
CERTIFICATE

This is to certify that the Lab work entitled "**ARTIFICIAL INTELLIGENCE**" carried out by **KEERTHI P REDDY (1BM21CS090),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to Sep-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **ARTIFICIAL INTELLIGENCE LAB (22CS5PCAIN)** work prescribed for the said degree.

Dr. Asha G R
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Program 1: Tic Tac Toe



17/11/23

Tic Tac Toe

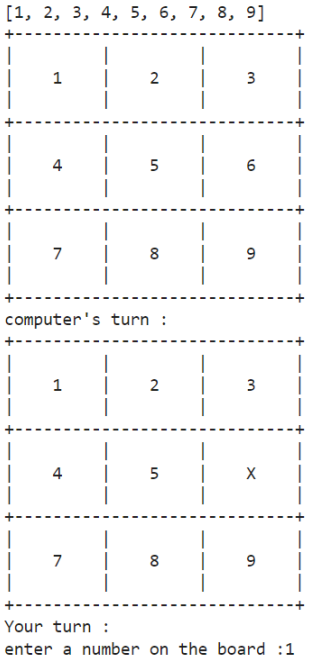
~~board[' ' for in range(9)]~~

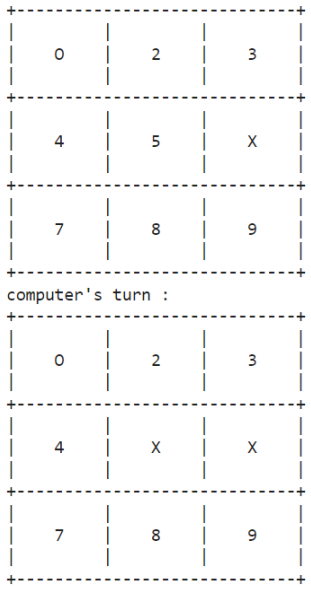
```
#g
import random
trc =[1,2,3,4,5,6,7,8,9]
def printBoard(trc):
    print(trc[0]+'|'+trc[1]+'|'+trc[2])
    print("- - - - -")
    print(trc[3]+'|'+trc[4]+'|'+trc[5])
    print("- - - - -")
    print(trc[6]+'|'+trc[7]+'|'+trc[8])

def isWinner(trc,pos):
    if trc[0]=tic[1] and trc[4]==trc[8] &
       trc[2]==trc[4] and trc[4]==trc[6]:
    return true
    else if trc[pos//3+1]=trc[pos//3+2] and
       trc[3+2]==trc[pos//3+3]:
    return true
return false

def update_user(trc):
    num=int(input("Enter a no. on board"))
    while(num not in trc):
        num=int(input("Enter a no. on the board"))
    trc[num-1]=='0'

def update_comp(tic):
    for i in trc:
        if i!='x' and i!='0':
```

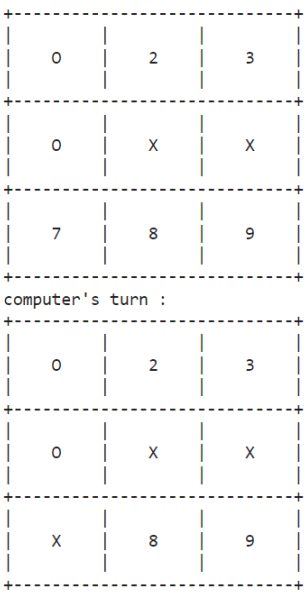
```
        trc[i-1] = 'x'
        if (winner(trc, i-1) == True):
            return
        else:
            trc[i-1] = i
    for i in trc:
        if i != 'x' and i != 'o':
            trc[i-1] = 'o'
            if (winner(trc, i-1) == False):
                return
            else
                trc[i-1] = i
    num = random.round(a)
    while (num not in trc):
        num = random.round(a)
    trc[num-1] = 'x'

print Board(trc)
count = 0
if count % 2 == 0:
    print ("Computer's turn")
    update - comp (trc)
    count += 1
else if count % 2 != 0
    print ("user turn")
    update_user(trc)
    count +1
if count >= 5
    if (winner (trc, pos) == true):
        print ("winner", trc[pos-1])
```

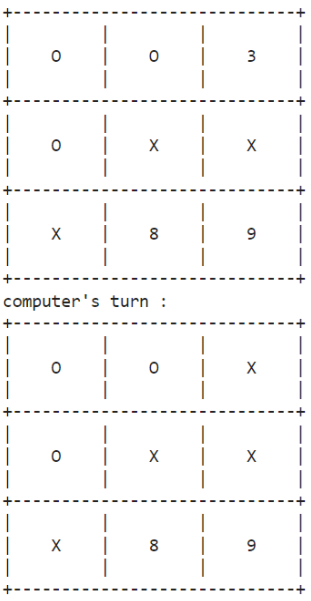
Output:

KEERTHI P REDDY 1BM21CS090

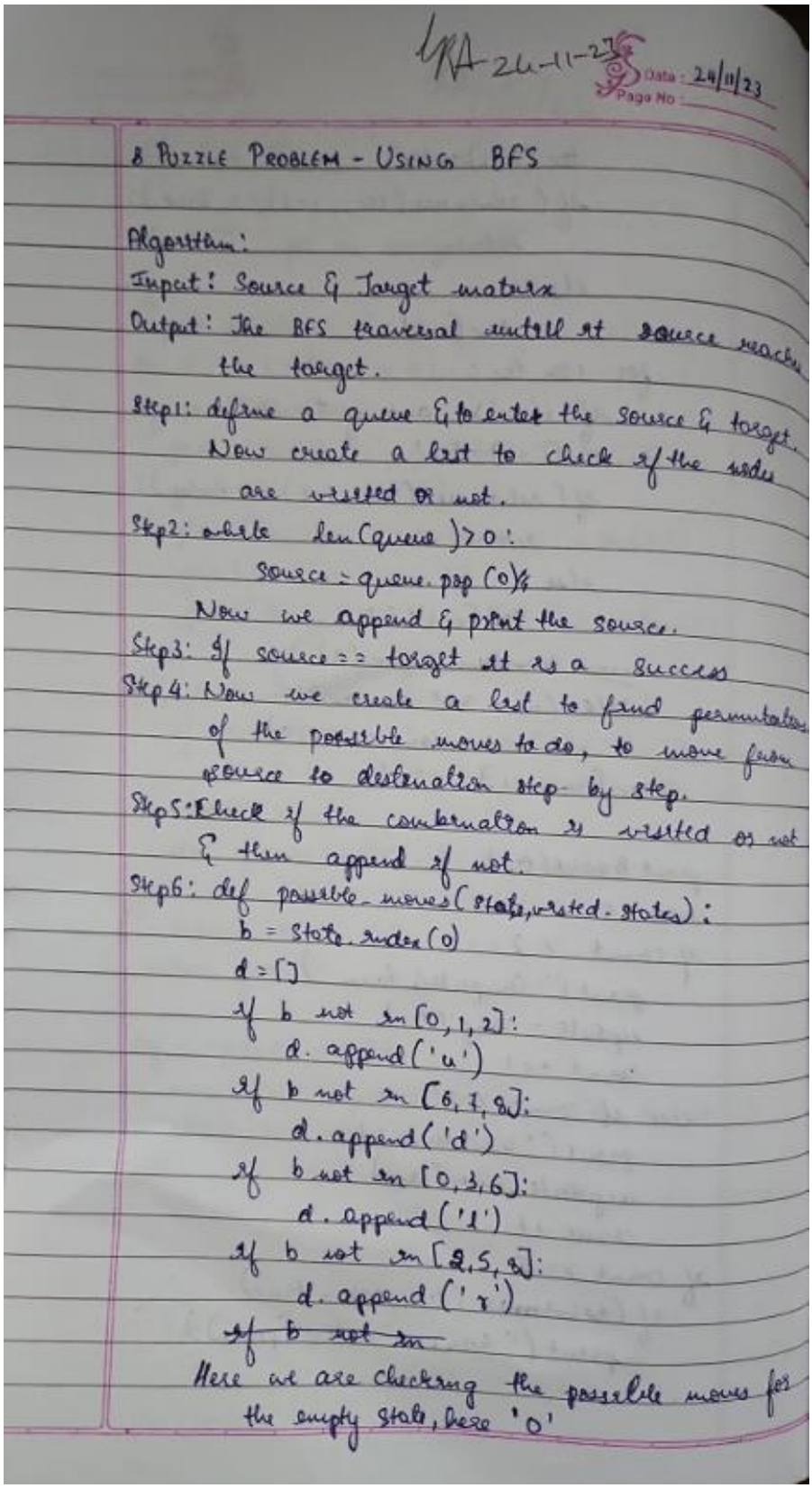








Program 2: 8 Puzzle Problem Using BFS



LKA 24-11-23 Date: 24/11/23 Page No:

8 PUZZLE PROBLEM - USING BFS

Algorithm:

Input: Source & Target matrix

Output: The BFS traversal until it source reaches the target.

Step1: define a queue & to enter the source & target. Now create a list to check if the nodes are visited or not.

Step2: while len(queue)>0:

source = queue.pop(0)

Now we append & print the source.

Step3: If source == target it is a success

Step4: Now we create a list to find permutations of the possible moves to do, to move from source to destination step by step.

Step5: Check if the combination is visited or not & then append if not.

Step6: def possible_moves(state,visited_states):

b = state.index(0)

d = []

if b not in [0,1,2]:

d.append('u')

if b not in [6,7,8]:

d.append('d')

if b not in [0,3,6]:

d.append('l')

if b not in [2,5,8]:

d.append('r')

~~if b not in~~

Here we are checking the possible moves for the empty state, here '0'

Step 7: for i in d:

possible_moves.append(gen(state, i, b))

gen(state, m, b):

temp = state.copy()

if m == 'd':

swap(b+3, b)

if m == 'u':

swap(b-3, b)

if m == 'l':

swap(b-1, b)

if m == 'r':

swap(b+1, b)

Step 8: Now return temp.

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

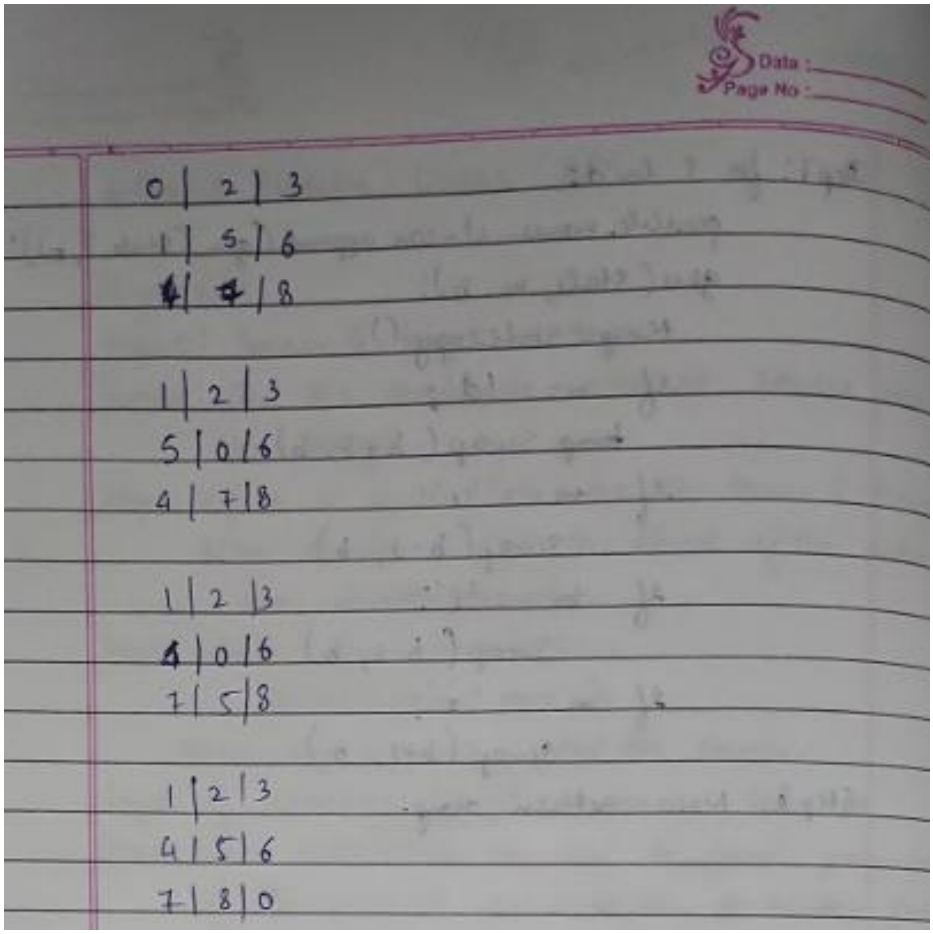
Src tgt

OUTPUT:

1	2	3
A	5	6
0	7	8

1	2	3
0	5	6
4	7	8

1	2	3
4	5	6
7	0	8

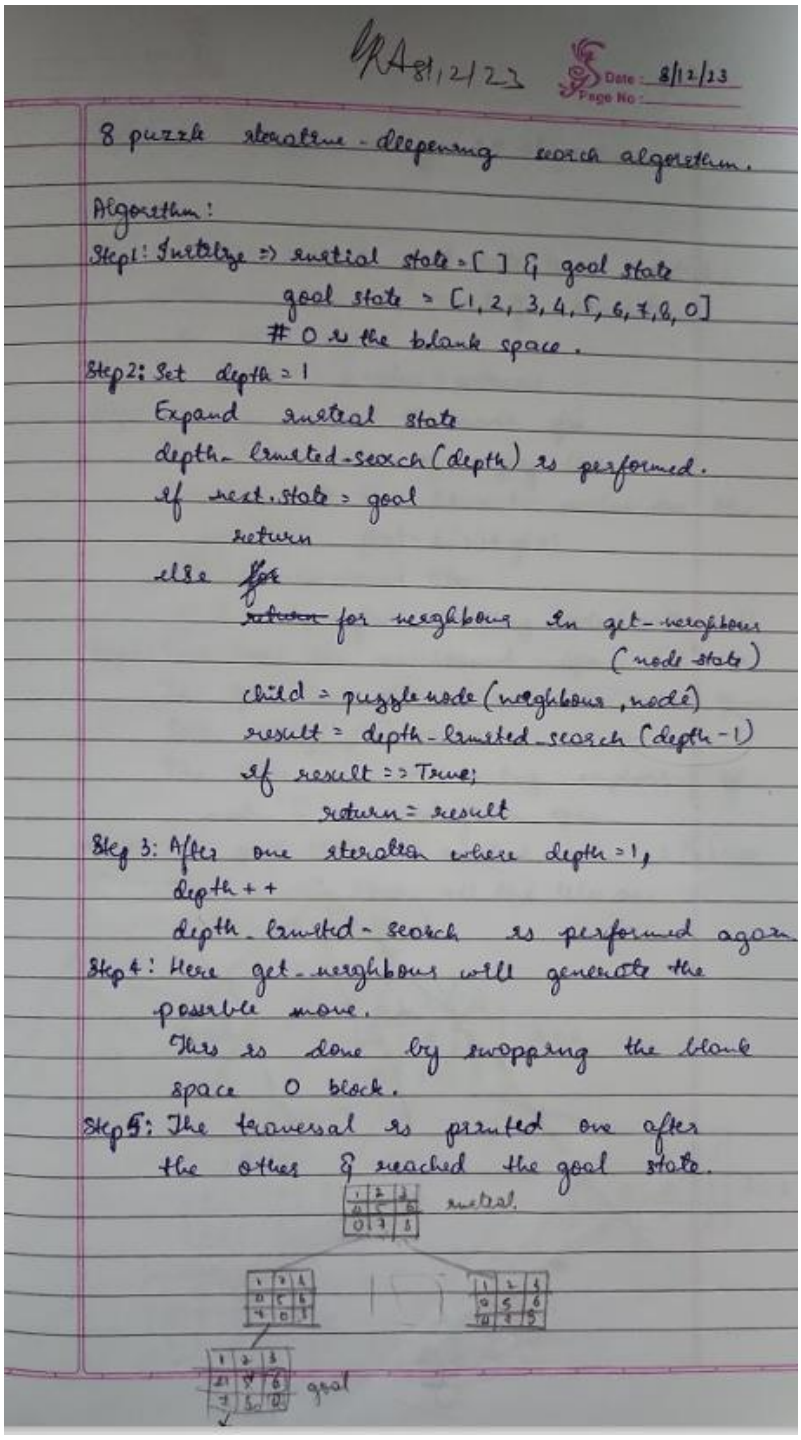


Output:

KEERTHI P REDDY 1BM21CS090

```
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
```

Program 3: 8 Puzzle Problem using Iterative Deepening Search algorithm

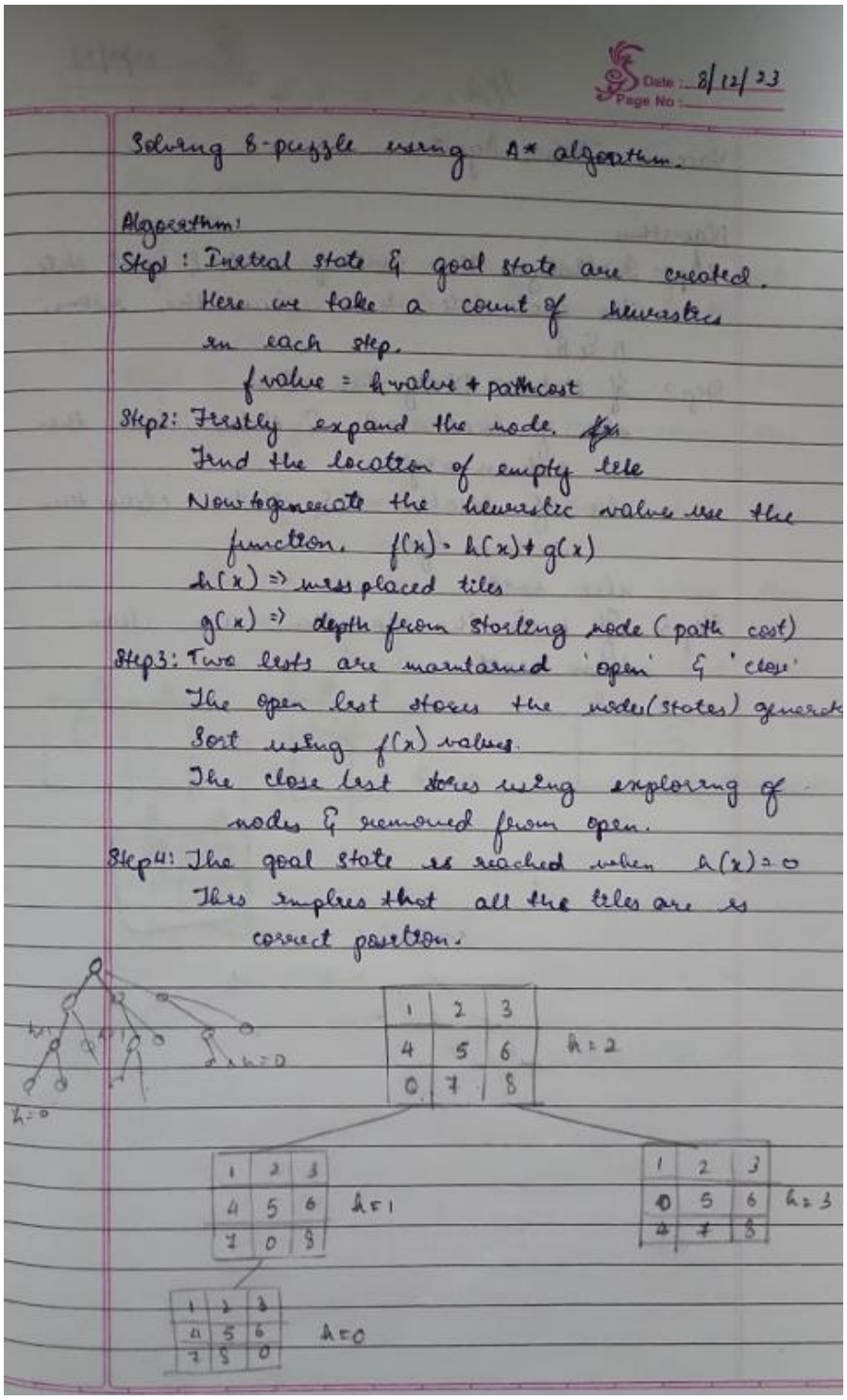


Output:

KEERTHI P REDDY 1BM21CS090

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

Program 4: 8 Puzzle Problem using A*



Date : 8/12/23
Page No :

Solving 8-puzzle using A* algorithm.

Algorithm:

Step1 : Initial state & goal state are created.
Here we take a count of heuristics in each step.
f value = h value + path cost

Step2: Firstly expand the node.
Find the location of empty tile
Now to generate the heuristic value use the function. f(x) = h(x) + g(x)
h(x) => misplaced tiles
g(x) => depth from starting node (path cost)

Step3: Two lists are maintained 'open' & 'close'
The open list stores the node(states) generated
Sort using f(x) values.
The close list stores using exploring of nodes & removed from open.

Step4: The goal state is reached when h(x) = 0
This implies that all the tiles are in correct position.

h=0

1	2	3
4	5	6
0	7	8

h=2

1	2	3
4	5	6
7	0	8

h=1

1	2	3
0	5	6
4	7	8

h=3

1	2	3
4	5	6
7	8	0

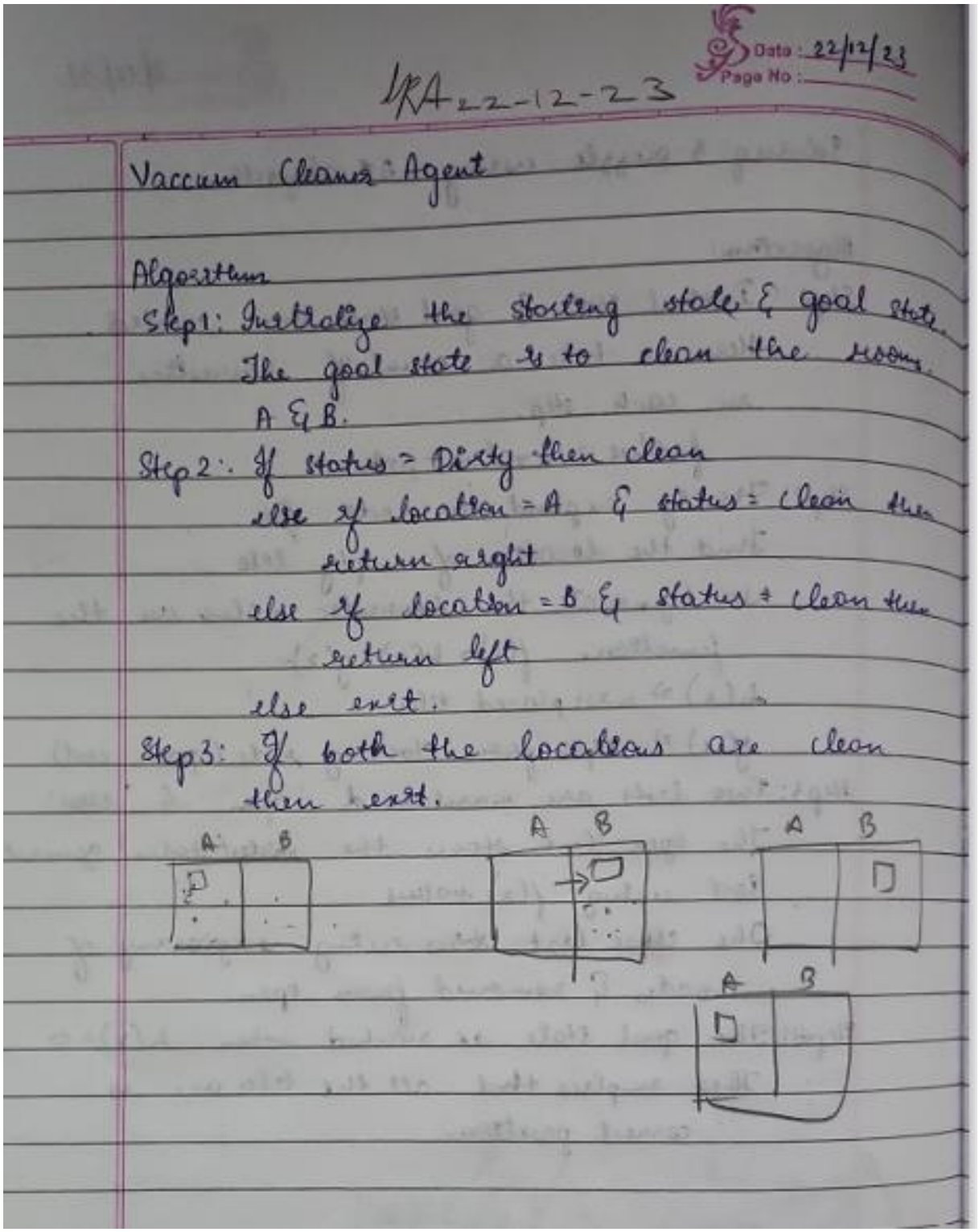
h=0

Output:

KEERTHI P REDDY 1BM21CS090

```
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 _
```

Program 5: Vacuum Cleaner Agent

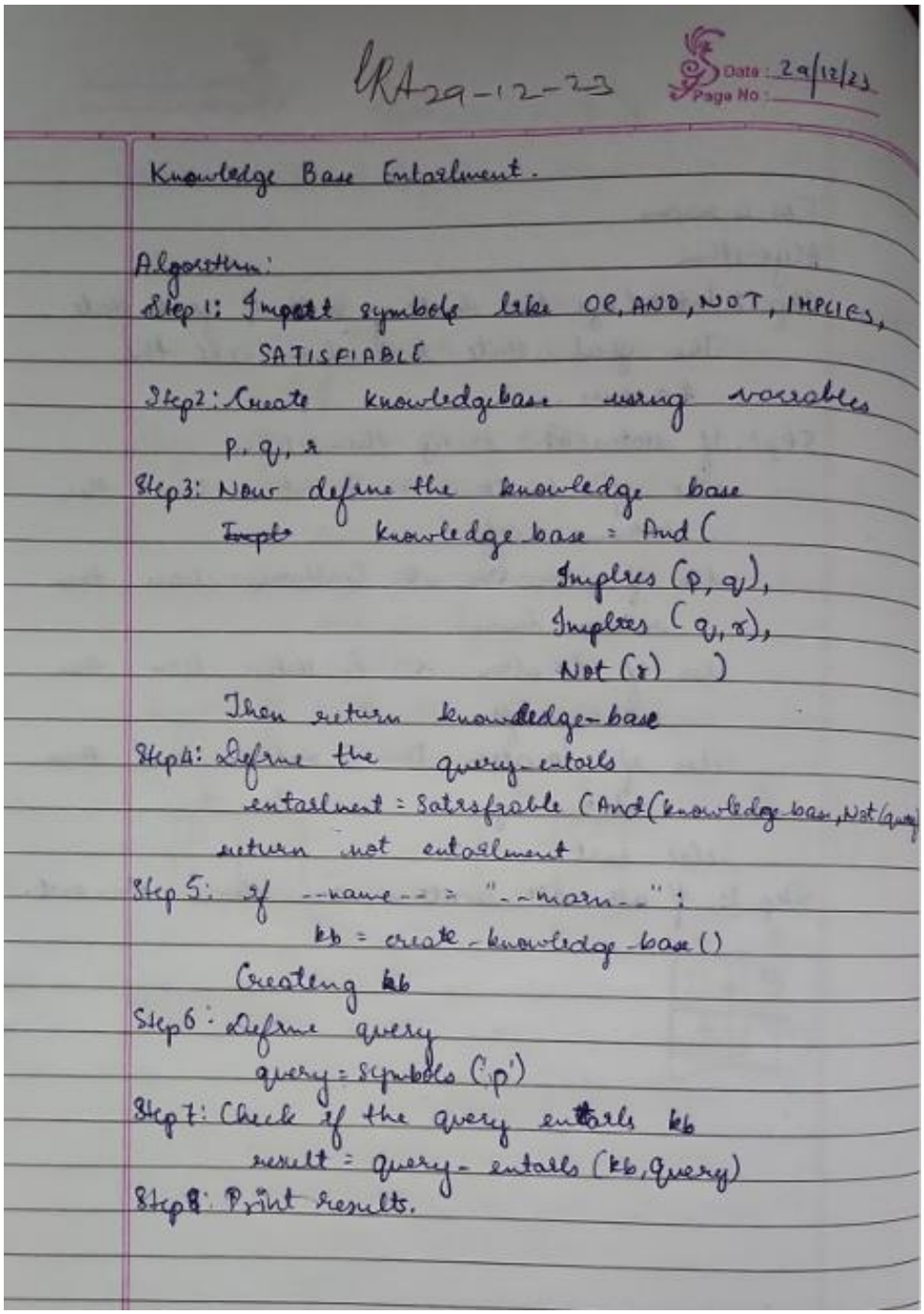


Output:

KEERTHI P REDDY 1BM21CS090

```
0 indicates clean and 1 indicates dirty
Enter Location of Vacuum        a
Enter status of a1
Enter status of other room1
Vacuum is placed in location B
Location B is Dirty.
COST for CLEANING 1
Location B has been Cleaned.
Location A is Dirty.
Moving LEFT to the Location A.
COST for moving LEFT2
COST for SUCK 3
Location A has been Cleaned.
GOAL STATE:
{'A': '0', 'B': '0'}
Performance Measurement: 3
```

Program 6: Knowledge Base Entailment



LRA 29-12-23

Date: 29/12/23
Page No:

Knowledge Base Entailment.

Algorithm:

Step 1: Import symbols like OR, AND, NOT, IMPLIES, SATISFIABLE

Step 2: Create knowledgebase using variables p, q, r

Step 3: Now define the knowledge base

Knowledge base = And (
Implies (p, q),
Implies (q, r),
Not (r))

Then return knowledge-base

Step 4: Define the query_entails

entailment = Satisfiable (And(knowledge_base, Not(query

return not entailment

Step 5: if --name-- = "--main--":

kb = create_knowledge_base()

Creating kb

Step 6: Define query

query = Symbols ('p')

Step 7: Check if the query entails kb

result = query_entails (kb, query)

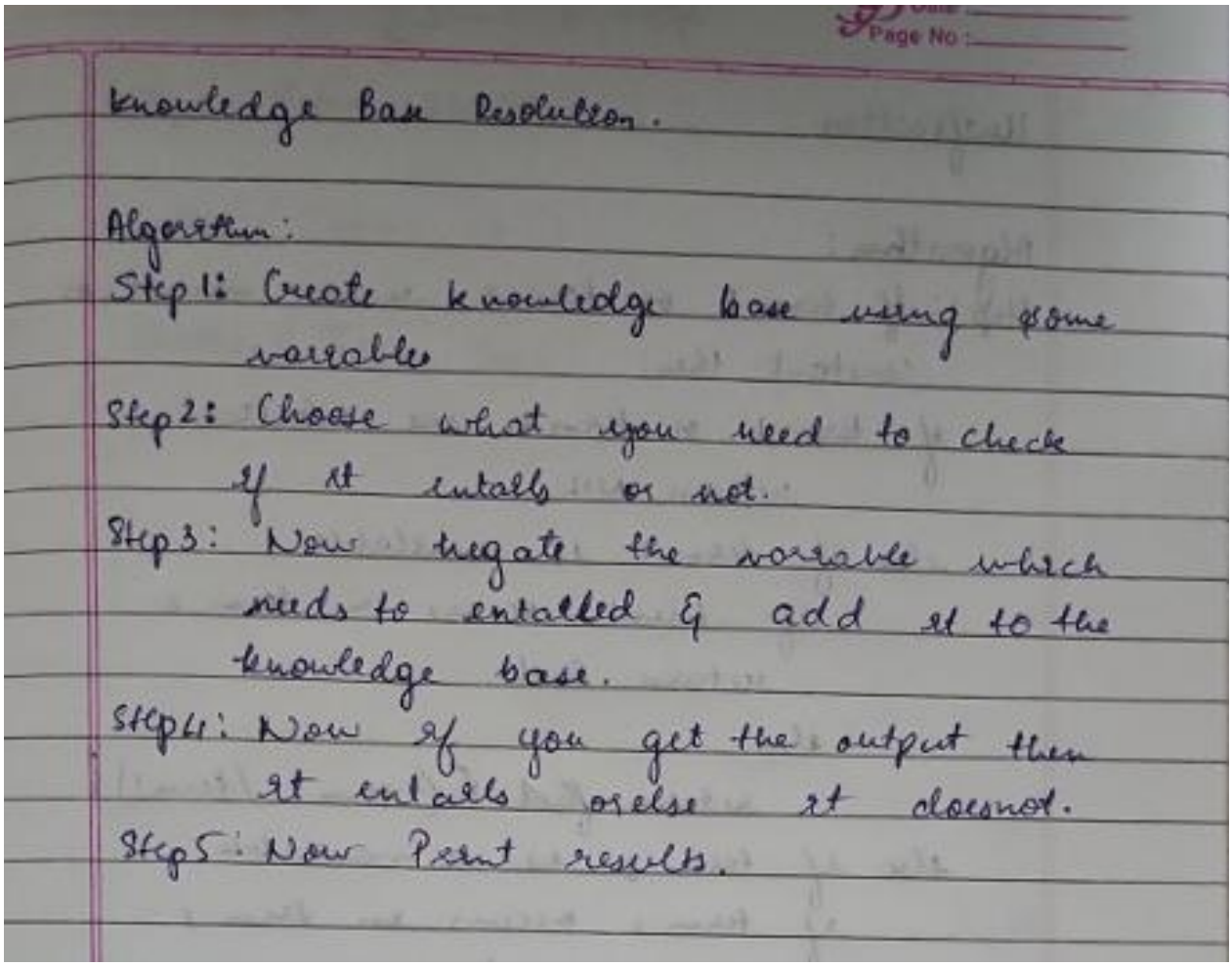
Step 8: Print results.

Output:

KEERTHI P REDDY 1BM21CS090

```
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

Program 7: Knowledge Base Resolution



Knowledge Base Resolution.

Algorithm:

Step 1: Create knowledge base using some variables

Step 2: Choose what you need to check if it entails or not.

Step 3: Now negate the variable which needs to entailed & add it to the knowledge base.

Step 4: Now if you get the output then it entails orelse it doesnot.

Step 5: Now Print results.

Output:

```
KEERTHI P REDDY 1BM21CS090

 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.
```

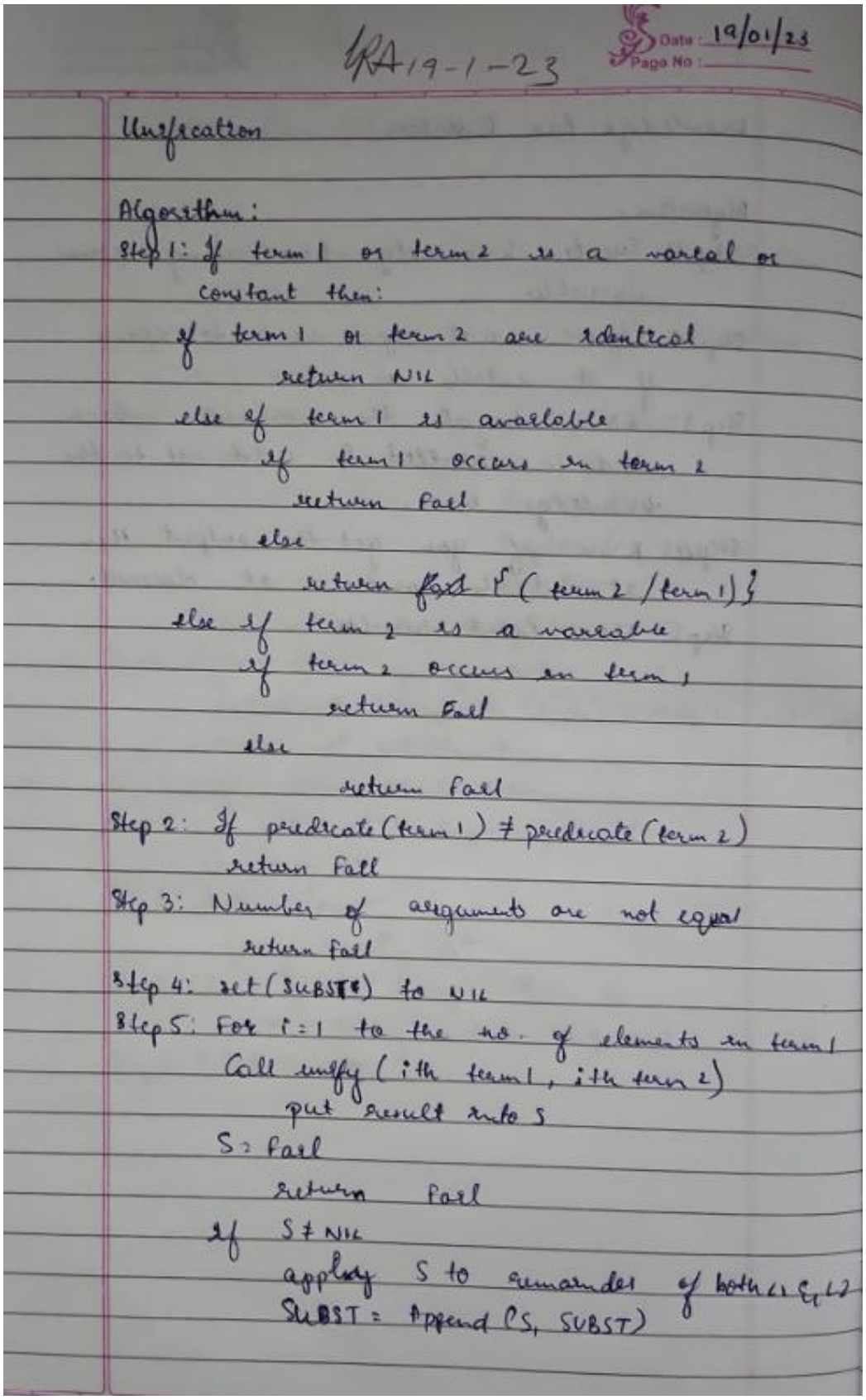
Step	Clause	Derivation
1.	PvQ	Given.
2.	~PvR	Given.
3.	~QvR	Given.
4.	~R	Negated conclusion.
5.	QvR	Resolved from PvQ and ~PvR.
6.	PvR	Resolved from PvQ and ~QvR.
7.	~P	Resolved from ~PvR and ~R.
8.	~Q	Resolved from ~QvR and ~R.
9.	Q	Resolved from ~R and QvR.
10.	P	Resolved from ~R and PvR.
11.	R	Resolved 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.

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.

Program 8: Unification



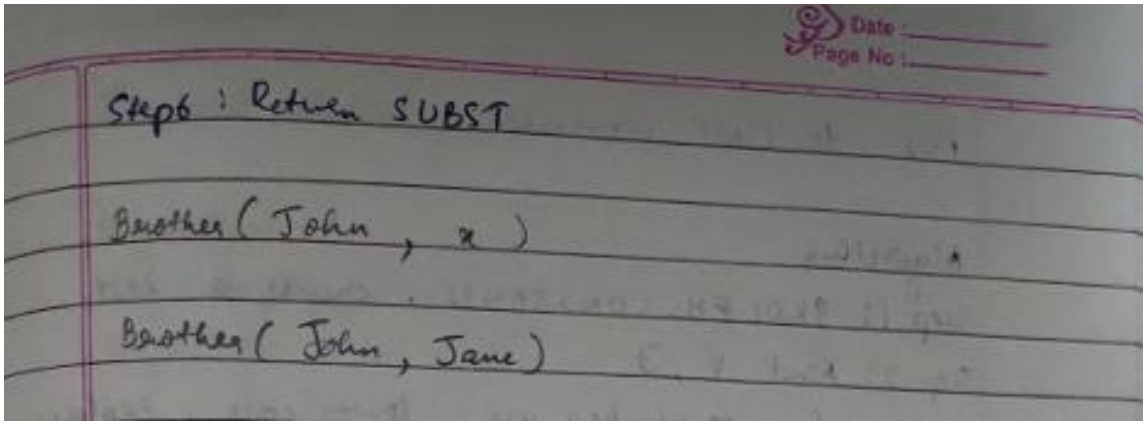
19-1-23

Date: 19/01/23
Page No:

Unification

Algorithm:

Step 1: If term1 or term2 is a variable or constant then:
  If term1 or term2 are identical
    return NIL
  else if term1 is available
    if term1 occurs in term2
      return Fail
    else
      return {(term2/term1)};
  else if term2 is a variable
    if term2 occurs in term1
      return Fail
    else
      return Fail
Step 2: If predicate(term1) ≠ predicate(term2)
  return Fail
Step 3: Number of arguments are not equal
  return Fail
Step 4: set(SUBST) to NIL
Step 5: For i=1 to the no. of elements in term1
  Call unify(ith term1, ith term2)
    put result into S
  S = Fail
    return Fail
  If S ≠ NIL
    apply S to remainder of both L1 & L2
    SUBST = Append(S, SUBST)

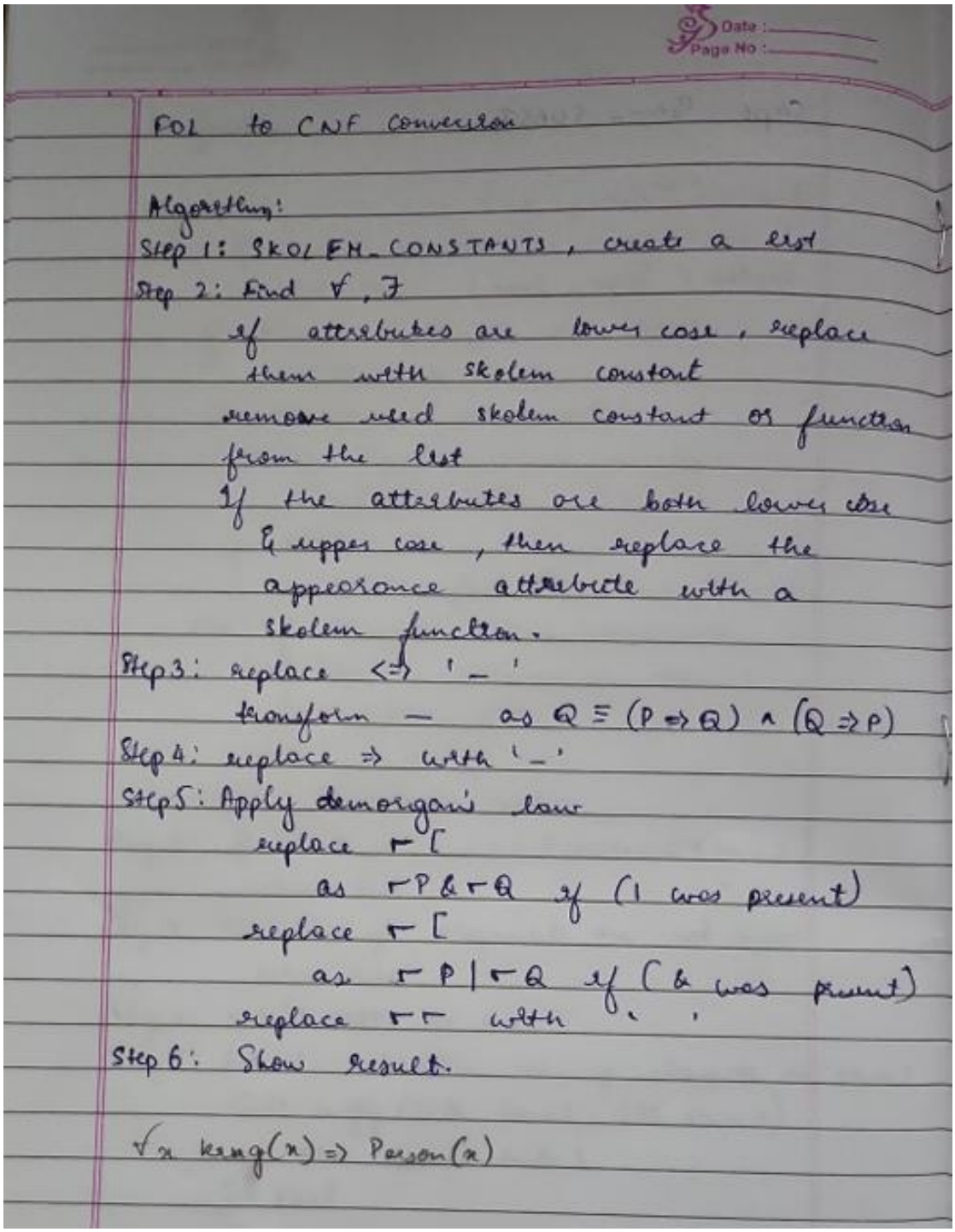


Output:

KEERTHI P REDDY 1BM21CS090

```
Substitutions:
[('X', 'Richard')]

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

Program 9: FOL to CNF conversion



FOL to CNF conversion

Algorithm:

Step 1: SKOLEM_CONSTANTS, create a list

Step 2: Find ∀, ∃

if attributes are lower case, replace them with skolem constant

remove used skolem constant or function from the list

If the attributes are both lower case & upper case, then replace the appearance attribute with a skolem function.

Step 3: replace <=> '_'

transform — as $Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

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 ' '

Step 6: Show result.

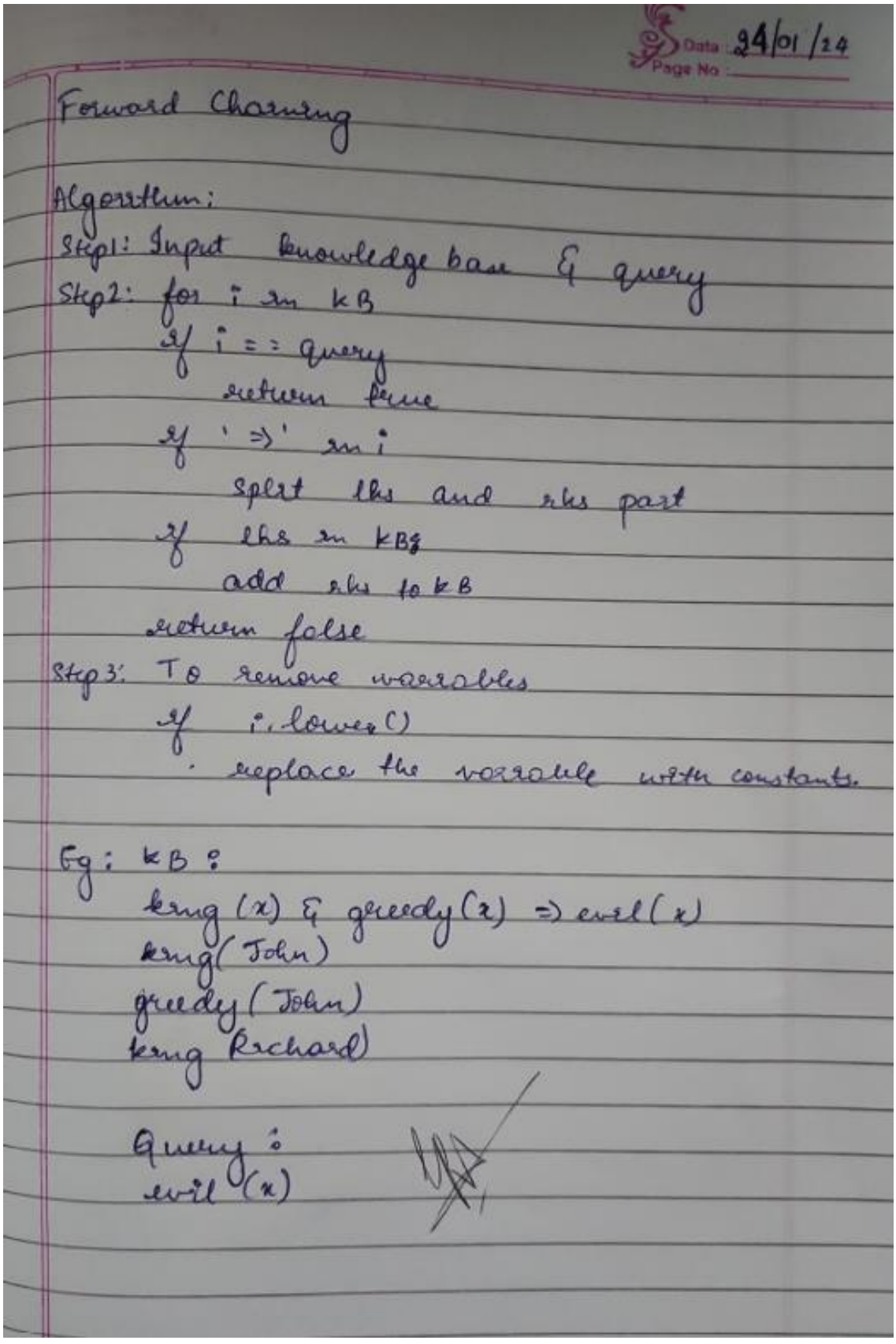
$\forall x$ king(x) => Person(x)

Output:

KEERTHI P REDDY 1BM21CS090

```
[~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)
```

Program 10: Forward Chaining



Date 24/01/24
Page No :

Forward Chaining

Algorithm:

Step1: Input knowledge base & query

Step2: for i in KB
    if i == query
        return true
    if '=>' in i
        split lhs and rhs part
    if lhs in KB:
        add rhs to KB
    return false

Step3: To remove variables
    if i.lower()
    . replace the variable with constants.

Eg: KB:
    king(x) & greedy(x) => evil(x)
    king(John)
    greedy(John)
    king Richard)

    Query:
    evil(x)

Output:

KEERTHI P REDDY 1BM21CS090

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

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