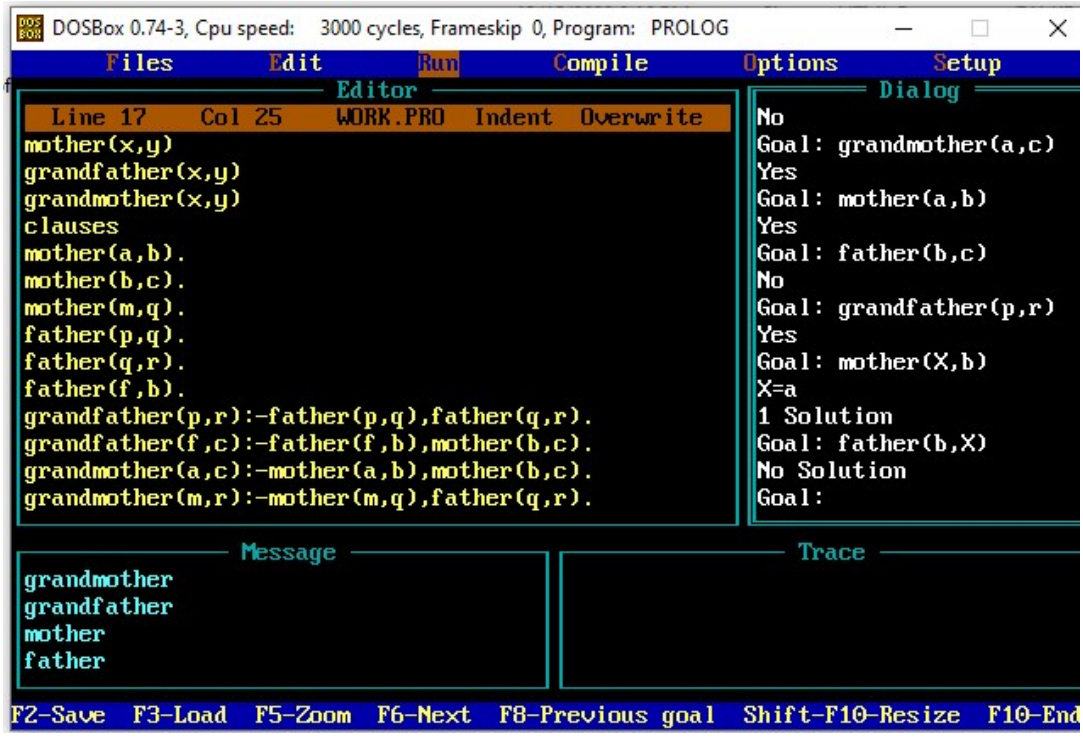
PRACTICAL-1

AIM: Write a program in prolog to implement simple facts and Queries.

Code:

```
domains
x,y = symbol
predicates
father(x,y)
mother(x,y)
grandfather(x,y)
grandmother(x,y)
clauses
mother(a,b).
mother(b,c).
mother(m,q).
father(p,q).
father(q,r).
father(f,b).
grandfather(p,r):-father(p,q),father(q,r).
grandfather(f,c):-father(f,b),mother(b,c).
grandmother(a,c):-mother(a,b),mother(b,c).
grandmother(m,r):-mother(m,q), father(q,r).
```

Output:



PRACTICAL-7

AIM: Write a python program to implement Breadth First Search Traversal?

Code:

```
from collections import defaultdict
class Graph_bfs:
  def __init__(self):
    self.graph_dict = defaultdict(list)
  def edge(self, From, To):
   self.graph_dict[From].append(To)
  def bfs(self, start):
    visited_node = [False] * (len(self.graph_dict))
    queue1 = []
    queue1.append(start)
    visited_node[start] = True
    while queue1:
      start = queue1.pop(0)
      print(start, end=" / ")
      for i in self.graph_dict[start]:
        if visited_node[i] == False:
          queue1.append(i)
          visited_node[i] = True
b1 = Graph_bfs()
print('Enter path from vertex 1 and vertex 2 :- ')
while(1):
  new = int(input('u want to add path? 1(yes)/0(no) :- '))
  k = bool(new)
  if(k == False):
    break
  key = int(input('Enter the vertex 1 :- '))
  value = int(input('Enter the vertex 2 :- '))
  b1.edge(key, value)
n = int(input('Enter number of start vertex :- '))
b1.bfs(n)
```

Output:

```
Enter path from vertex 1 and vertex 2 :-

u want to add path? 1(yes)/0(no) :- 1

Enter the vertex 1 :- 2

Enter the vertex 2 :- 0

u want to add path? 1(yes)/0(no) :- 1

Enter the vertex 1 :- 0

Enter the vertex 2 :- 1

u want to add path? 1(yes)/0(no) :- 1

Enter the vertex 1 :- 1

Enter the vertex 2 :- 2

u want to add path? 1(yes)/0(no) :- 0

Enter number of start vertex :- 1
1 / 2 / 0 /
```

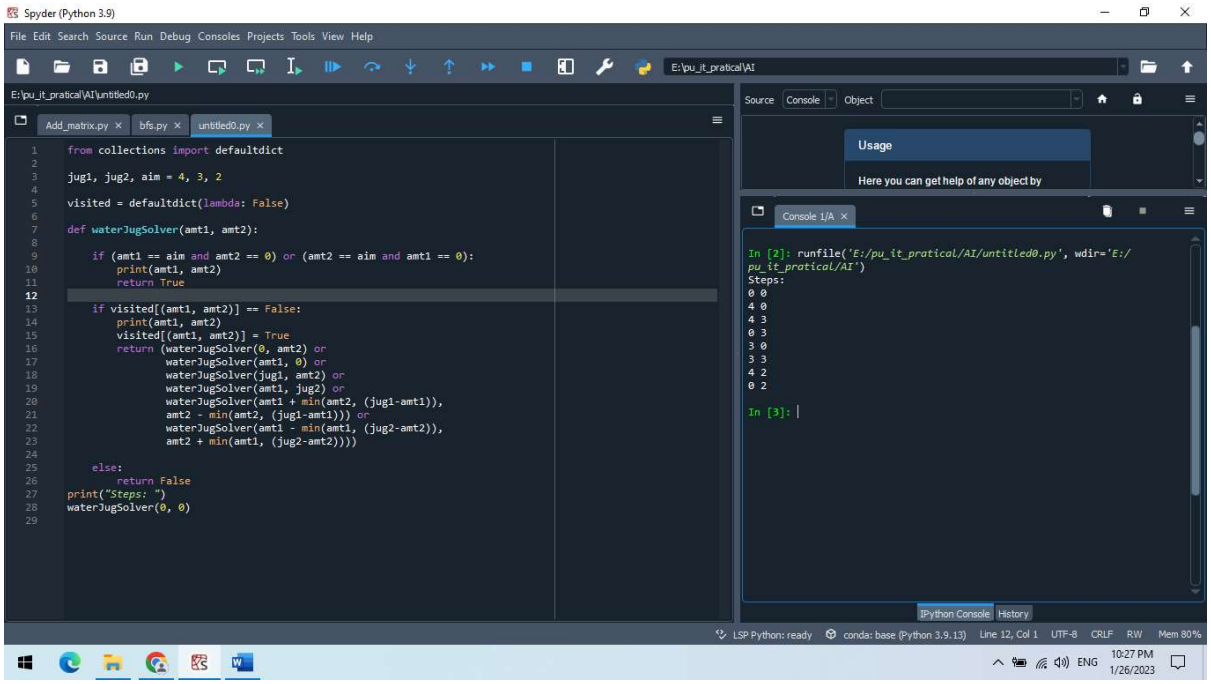
PRACTICAL-8

AIM: Write a python program to implement Water Jug Problem?

Code:

```
from collections import defaultdict
jug1, jug2, aim = 4, 3, 2
visited = defaultdict(lambda: False)
def waterJugSolver(amt1, amt2):
        if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
                print(amt1, amt2)
                return True
        if visited[(amt1, amt2)] == False:
                print(amt1, amt2)
                visited[(amt1, amt2)] = True
                return (waterJugSolver(0, amt2) or
                                waterJugSolver(amt1, 0) or
                                waterJugSolver(jug1, amt2) or
                                waterJugSolver(amt1, jug2) or
                                waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
                                amt2 - min(amt2, (jug1-amt1))) or
                                waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
                                amt2 + min(amt1, (jug2-amt2))))
        else:
                return False
print("Steps: ")
waterJugSolver(0, 0)
```

Output:



PRACTICAL-9

AIM: Write a program to implement Tic-Tac-Toe game using python.

Code:

```
import random
class TicTacToe:
    def __init__(self):
        self.board = []
    def create_board(self):
        for i in range(3):
            row = []
            for j in range(3):
                row.append('-')
            self.board.append(row)
    def get_random_first_player(self):
        return random.randint(0, 1)
    def fix_spot(self, row, col, player):
        self.board[row][col] = player
    def is_player_win(self, player):
        win = None
        n = len(self.board)
        for i in range(n):
            win = True
            for j in range(n):
                if self.board[i][j] != player:
                    win = False
                    break
            if win:
                return win
        for i in range(n):
            win = True
            for j in range(n):
                if self.board[j][i] != player:
                    win = False
                    break
            if win:
                return win
        win = True
        for i in range(n):
            if self.board[i][i] != player:
                win = False
                break
        if win:
            return win
```

```
        win = True
        for i in range(n):
            if self.board[i][n - 1 - i] != player:
                win = False
                break
        if win:
            return win
        return False
        for row in self.board:
            for item in row:
                if item == '-':
                    return False
        return True
    def is_board_filled(self):
        for row in self.board:
            for item in row:
                if item == '-':
                    return False
        return True
    def swap_player_turn(self, player):
        return 'X' if player == 'O' else 'O'
    def show_board(self):
        for row in self.board:
            for item in row:
                print(item, end=" ")
            print()
    def start(self):
        self.create_board()
        player = 'X' if self.get_random_first_player() == 1 else 'O'
        while True:
            print(f"Player {player} turn")
            self.show_board()
            row, col = list(
                map(int, input("Enter row and column numbers to fix spot: ").split()))
            print()
            self.fix_spot(row - 1, col - 1, player)
            if self.is_player_win(player):
                print(f"Player {player} wins the game!")
                break
            if self.is_board_filled():
                print("Match Draw!")
                break
            player = self.swap_player_turn(player)
        print()
```

```
        self.show_board()
tic_tac_toe = TicTacToe()
tic_tac_toe.start()
```

Output: