1.	**Write a program to implement Tic-Tac-Toe game problem.**

```
def tic_tac_toe():
    board = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    end = False
    win = [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8], [2, 4, 6]]

    def draw():
     #  print(board[0])
        print(board[0], board[1], board[2])
        print(board[3], board[4], board[5])
        print(board[6], board[7], board[8])
        print("****************************************")
    def p1():
        n = choose_number()
        if board[n] == "X" or board[n] == "O":  #X and O are already placed
            print("\nPlease enter another digit for board position")
            p1()
        else:
            board[n] = "X"  #simply print X on nth position

    def p2():
        n = choose_number()
        if board[n] == "X" or board[n] == "O":  #X and O already palced
            print("\nPlease enter another digit for board position")
            p2()
        else:
            board[n] = "O"  #simply print O on nth position

    def choose_number():
        while True:
            while True:
                a = input()
                try:
                    a  = int(a)
                    a -= 1
                    if a in range(0, 9):
                        return a  #return board value between 0-9
                    else:
                        print("\nThat's not on the board. Try again")
                        continue
                except ValueError:
                    print("\nThat's not a number. Try again")
                    continue

    def check_board():
        count = 0
        #    win = [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8], [2, 4, 6]]
        for a in win:
```

```
        # print(a[1])
         if board[a[0]] == board[a[1]] == board[a[2]] == "X":
            print("Player 1 Wins!\n")
            print("Congratulations!\n")
            return True

         if board[a[0]] == board[a[1]] == board[a[2]] == "O":
            print("Player 2 Wins!\n")
            print("Congratulations!\n")
            return True
      for a in range(9):
         if board[a] == "X" or board[a] == "O":
            count += 1
         if count == 9:
            print("The game ends in a Tie\n")
            return True

#start main.....................
   while not end:
      #player 1
      draw()
      end = check_board() #check board for winner posibilities
      if end == True:
         break
      print("Player 1 choose where to place a X")
      p1()
      print(".....................................")
      #player 2
      draw()
      end = check_board()
      if end == True:
         break
      print("Player 2 choose where to place a O")
      p2()
      print(".....................................")


   if input("Play again (y/n)\n") == "y":
      print()
      tic_tac_toe()

tic_tac_toe()
```

OUTPUT:-

```
1 2 3
4 5 6
7 8 9
**************************************
Player 1 choose where to place a X
```

```
1
.....
X 2 3
4 5 6
7 8 9
**************************************
Player 2 choose where to place a O
5
.....
X 2 3
4 O 6
7 8 9
**************************************
Player 1 choose where to place a X
9
.....
X 2 3
4 O 6
7 8 X
**************************************
Player 2 choose where to place a O
3
.....
X 2 O
4 O 6
7 8 X
**************************************
Player 1 choose where to place a X
7
.....
X 2 O
4 O 6
X 8 X
**************************************
Player 2 choose where to place a O
4
.....
X 2 O
O O 6
X 8 X
**************************************
Player 1 choose where to place a X
2
.....
X X O
O O 6
X 8 X
**************************************
```

	Player 2 choose where to place a O 6 ..... X X O O O O X 8 X ********** Player 2 Wins! Congratulations!

2.	**Write a program to implement BFS (for 8 puzzle problem or Water Jug problem or any AI search problem.**

State.java:

```
package waterjug;
class State {
 int x;
 int y;
 State pre;

 public State(int x, int y) {
  super();
  this.x = x;
  this.y = y;
 }
 public State(int x, int y, State pre) {
  super();
  this.x = x;
  this.y = y;
  this.pre = pre;
 }

 @Override
 public int hashCode() {
  final int prime = 31;
  int result = 1;
  result = prime * result + x;
  result = prime * result + y;
  return result;
 }

 @Override
 public boolean equals(Object obj) {
  if (this == obj) {
   return true;
  }
  if (obj == null) {
   return false;
  }
  if (getClass() != obj.getClass()) {
   return false;
  }
  State other = (State) obj;
  if (x != other.x) {
```

```
      return false;
    }
    if (y != other.y) {
      return false;
    }
    return true;
  }

  @Override
  public String toString() {
    StringBuilder builder = new StringBuilder();
    if (pre != null) {
      builder.append(pre);
    }
    builder.append(x);
    builder.append("    ").append(y).append("\n");
    return builder.toString();
  }
}
```

WaterJug.java:

```
package waterjug;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Scanner;

class WaterJug {

  static int jug1 = 0;
  static int jug2 = 0;
  static int amtNeeded = 0;
  HashSet<State> uniqueStates;

  void letsRoll() {

    State initialState = new State(0, 0);
    State finalState = new State(amtNeeded, 0);
    State finalPath = null;

    uniqueStates = new HashSet<>();
    LinkedList<State> queue = new LinkedList<>();
    queue.add(initialState);

    while (!queue.isEmpty()) {
```

```
    State currState = queue.poll();
    if (currState.equals(finalState)) {
     finalPath = currState;
     break;
    }

    if (currState.x == 0) {
     State nextState = new State(jug1, currState.y, currState);
     checkUniqueStates(uniqueStates, nextState, queue);
    }

    if (currState.y == 0) {
     State nextState = new State(currState.x, jug2, currState);
     checkUniqueStates(uniqueStates, nextState, queue);
    }

    if (currState.x > 0) {
     State nextState = new State(0, currState.y, currState);
     checkUniqueStates(uniqueStates, nextState, queue);
    }

    if (currState.y > 0) {
     State nextState = new State(currState.x, 0, currState);
     checkUniqueStates(uniqueStates, nextState, queue);
    }

    if (currState.x > 0 && currState.y < jug2) {
     int amtToTransfer = Math.min(currState.x, jug2 - currState.y);
     State nextState = new State(currState.x - amtToTransfer, currState.y
           + amtToTransfer,
           currState);
     checkUniqueStates(uniqueStates, nextState, queue);
    }

    if (currState.y > 0 && currState.x < jug1) {
     int amtToTransfer = Math.min(currState.y, jug1 - currState.x);
     State nextState = new State(currState.x + amtToTransfer, currState.y
           - amtToTransfer,
           currState);
     checkUniqueStates(uniqueStates, nextState, queue);
    }
   }
   if (finalPath != null) {
    System.out.println("J1  J2");
```

```
      System.out.println(finalPath);
    } else {
      System.out.println("Not Possible");

    }
  }

  void checkUniqueStates(HashSet<State> uniqueStates, State toCheck,
        LinkedList<State> queue) {
    if (!uniqueStates.contains(toCheck)) {
      uniqueStates.add(toCheck);
      queue.add(toCheck);
    }
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the maximum amount of Jugs capacity:");
    jug1 = sc.nextInt();
    jug2 = sc.nextInt();
    System.out.println("Enter the Amount needed:");
    amtNeeded = sc.nextInt();
    new WaterJug().letsRoll();
  }
}
```

OUTPUT:-

```
Enter the maximum amount of Jugs capacity:
4 3
Enter the Amount needed:
2
J1  J2
0   0
0   3
3   0
3   3
4   2
0   2
2   0
```

3.	**Write a program to implement DFS (for 8 puzzle problem or Water Jug problem or any AI search problem).**

```
package dfs8puzzle;

import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Scanner;

public class DFS8Puzzle {

  public static LinkedHashSet<String> OPEN = new LinkedHashSet<String>();
  public static HashSet<String> CLOSED = new HashSet<String>();
  public static boolean STATE = false;

  public static void main(String args[]) {

    int statesVisited = 0;

    String goal = "123804765";
    String start = "281043765";
    String X = "";
    String temp = "";

    Scanner sc = new Scanner(System.in);
    System.out.println("Enter Your input as a single line from 0 - 9:");
    start = sc.nextLine();

    OPEN.add(start);

    while (OPEN.isEmpty() == false && STATE == false) {

      X = OPEN.iterator().next();
      OPEN.remove(X);

      int pos = X.indexOf('0');
      if (X.equals(goal)) {
        System.out.println("initial position is:");
        print(start);
        System.out.println(" you reach to goal position:");
        print(goal);
        System.out.println("SUCCESS");

        STATE = true;
      } else {
```

```
      CLOSED.add(X);

      temp = up(X, pos);
      if (!(temp.equals("-1"))) {
        OPEN.add(temp);
      }
      temp = left(X, pos);
      if (!(temp.equals("-1"))) {
        OPEN.add(temp);
      }
      temp = down(X, pos);
      if (!(temp.equals("-1"))) {
        OPEN.add(temp);
      }
      temp = right(X, pos);
      if (!(temp.equals("-1"))) {
        OPEN.add(temp);
      }
    }
  }

}

public static String up(String s, int p) {
  String str = s;
  if (!(p < 3)) {
    char a = str.charAt(p - 3);
    String newS = str.substring(0, p) + a + str.substring(p + 1);
    str = newS.substring(0, (p - 3)) + '0' + newS.substring(p - 2);

  }
  if (!OPEN.contains(str) && CLOSED.contains(str) == false) {
    return str;
  } else {
    System.out.println("Up");
    return "-1";
  }
}

public static String down(String s, int p) {
  String str = s;
  if (!(p > 5)) {
    char a = str.charAt(p + 3);
    String newS = str.substring(0, p) + a + str.substring(p + 1);
```

```
    str = newS.substring(0, (p + 3)) + '0' + newS.substring(p + 4);

   }
   if (!OPEN.contains(str) && CLOSED.contains(str) == false) {
    return str;
   } else {
    System.out.println("Down");
    return "-1";
   }
  }
  public static String left(String s, int p) {
   String str = s;
   if (p != 0 && p != 3 && p != 7) {
    char a = str.charAt(p - 1);
    String newS = str.substring(0, p) + a + str.substring(p + 1);
    str = newS.substring(0, (p - 1)) + '0' + newS.substring(p);
   }
   if (!OPEN.contains(str) && CLOSED.contains(str) == false) {
    return str;
   } else {

    System.out.println("Left");
    return "-1";
   }
  }
  public static String right(String s, int p) {
   String str = s;
   if (p != 2 && p != 5 && p != 8) {
    char a = str.charAt(p + 1);
    String newS = str.substring(0, p) + a + str.substring(p + 1);
    str = newS.substring(0, (p + 1)) + '0' + newS.substring(p + 2);
   }
   if (!OPEN.contains(str) && CLOSED.contains(str) == false) {
    return str;
   } else {
    System.out.println("Right");
    return "-1";
   }
  }
  public static void print(String s) {
   System.out.println(s.substring(0, 3));
   System.out.println(s.substring(3, 6));
   System.out.println(s.substring(6, 9));
   System.out.println();
```

```
  }
}
```

OUTPUT:-

```
Enter Your input as a single line from 0 - 9:
281043765
Left->Up->Left->Down->Up->Down->Left
initial position is:
281
043
765
 you reach to goal position:
123
804
765
SUCCESS
```

4. Write a program to implement Single Player Game (Using Heuristic Function).

```
package singleplayer;
import java.util.*;
class Point {
  int x, y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  @Override
  public String toString() {
    return "[" + x + ", " + y + "]";
  }
}

class PointsAndScores {

  int score;
  Point point;

  PointsAndScores(int score, Point point) {
    this.score = score;
    this.point = point;
  }
}

class Board {

  List<Point> availablePoints;
  Scanner scan = new Scanner(System.in);
  int[][] board = new int[3][3];

  public Board() {
  }

  public boolean isGameOver() {
    //Game is over is someone has won, or board is full (draw)
    return (hasXWon() || hasOWon() || getAvailableStates().isEmpty());
  }

  public boolean hasXWon() {
    if ((board[0][0] == board[1][1] && board[0][0] == board[2][2] && board[0][0] == 1) ||
(board[0][2] == board[1][1] && board[0][2] == board[2][0] && board[0][2] == 1)) {
      //System.out.println("X Diagonal Win");
      return true;
    }
    for (int i = 0; i < 3; ++i) {
```

```
    if (((board[i][0] == board[i][1] && board[i][0] == board[i][2] && board[i][0] == 1)
        || (board[0][i] == board[1][i] && board[0][i] == board[2][i] && board[0][i] == 1))) {
      // System.out.println("X Row or Column win");
      return true;
    }
   }
   return false;
  }

  public boolean hasOWon() {
   if ((board[0][0] == board[1][1] && board[0][0] == board[2][2] && board[0][0] == 2) ||
(board[0][2] == board[1][1] && board[0][2] == board[2][0] && board[0][2] == 2)) {
    // System.out.println("O Diagonal Win");
    return true;
   }
   for (int i = 0; i < 3; ++i) {
    if ((board[i][0] == board[i][1] && board[i][0] == board[i][2] && board[i][0] == 2)
        || (board[0][i] == board[1][i] && board[0][i] == board[2][i] && board[0][i] == 2)) {
      //  System.out.println("O Row or Column win");
      return true;
    }
   }

   return false;
  }

  public List<Point> getAvailableStates() {
   availablePoints = new ArrayList<>();
   for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
     if (board[i][j] == 0) {
      availablePoints.add(new Point(i, j));
     }
    }
   }
   return availablePoints;
  }

  public void placeAMove(Point point, int player) {
   board[point.x][point.y] = player;   //player = 1 for X, 2 for O
  }

  public Point returnBestMove() {
   int MAX = -100000;
   int best = -1;

   for (int i = 0; i < rootsChildrenScores.size(); ++i) {
    if (MAX < rootsChildrenScores.get(i).score) {
```

```
      MAX = rootsChildrenScores.get(i).score;
      best = i;
    }
   }

   return rootsChildrenScores.get(best).point;
  }

  void takeHumanInput() {
   System.out.println("Your move: ");
   int x = scan.nextInt();
   int y = scan.nextInt();
   Point point = new Point(x, y);
   placeAMove(point, 2);
  }

  public void displayBoard() {
   System.out.println();

   for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
     System.out.print(board[i][j] + " ");
    }
    System.out.println();

   }
  }

  public int returnMin(List<Integer> list) {
   int min = Integer.MAX_VALUE;
   int index = -1;
   for (int i = 0; i < list.size(); ++i) {
    if (list.get(i) < min) {
     min = list.get(i);
     index = i;
    }
   }
   return list.get(index);
  }

  public int returnMax(List<Integer> list) {
   int max = Integer.MIN_VALUE;
   int index = -1;
   for (int i = 0; i < list.size(); ++i) {
    if (list.get(i) > max) {
     max = list.get(i);
     index = i;
    }
```

```
    }
    return list.get(index);
  }

  List<PointsAndScores> rootsChildrenScores;

  public void callMinimax(int depth, int turn) {
    rootsChildrenScores = new ArrayList<>();
    minimax(depth, turn);
  }

  public int minimax(int depth, int turn) {

    if (hasXWon()) {
      return +1;
    }
    if (hasOWon()) {
      return -1;
    }

    List<Point> pointsAvailable = getAvailableStates();
    if (pointsAvailable.isEmpty()) {
      return 0;
    }

    List<Integer> scores = new ArrayList<>();

    for (int i = 0; i < pointsAvailable.size(); ++i) {
      Point point = pointsAvailable.get(i);

      if (turn == 1) { //X's turn select the highest from below minimax() call
        placeAMove(point, 1);
        int currentScore = minimax(depth + 1, 2);
        scores.add(currentScore);

        if (depth == 0) {
          rootsChildrenScores.add(new PointsAndScores(currentScore, point));
        }

      } else if (turn == 2) {//O's turn select the lowest from below minimax() call
        placeAMove(point, 2);
        scores.add(minimax(depth + 1, 1));
      }
      board[point.x][point.y] = 0; //Reset this point
    }
    return turn == 1 ? returnMax(scores) : returnMin(scores);
  }
}
```

```
public class SinglePlayer {

  public static void main(String[] args) {
    Board b = new Board();
    Random rand = new Random();

    b.displayBoard();

    System.out.println("Who's gonna move first? (1)Computer (2)User: ");
    int choice = b.scan.nextInt();
    if (choice == 1) {
      Point p = new Point(rand.nextInt(3), rand.nextInt(3));
      b.placeAMove(p, 1);
      b.displayBoard();
    }

    while (!b.isGameOver()) {
      System.out.println("Your move: ");
      Point userMove = new Point(b.scan.nextInt(), b.scan.nextInt());

      b.placeAMove(userMove, 2); //2 for O and O is the user
      b.displayBoard();
      if (b.isGameOver()) {
        break;
      }
      b.callMinimax(0, 1);
      for (PointsAndScores pas : b.rootsChildrenScores) {
        System.out.println("Point: " + pas.point + " Score: " + pas.score);
      }
      b.placeAMove(b.returnBestMove(), 1);
      b.displayBoard();
    }
    if (b.hasXWon()) {
      System.out.println("Unfortunately, you lost!");
    } else if (b.hasOWon()) {
      System.out.println("You win! This is not going to get printed.");
    } else {
      System.out.println("It's a draw!");
    }
  }
}
```

OUTPUT:-

```
0 0 0
0 0 0
0 0 0
Who's gonna move first? (1)Computer (2)User:
2
```

```
Your move:
2 2

0 0 0
0 0 0
0 0 2
Point: [0, 0] Score: -1
Point: [0, 1] Score: -1
Point: [0, 2] Score: -1
Point: [1, 0] Score: -1
Point: [1, 1] Score: 0
Point: [1, 2] Score: -1
Point: [2, 0] Score: -1
Point: [2, 1] Score: -1

0 0 0
0 1 0
0 0 2
Your move:
2 0 …….
.
.
.
Your move:
1 0

1 2 2
2 1 1
2 1 2
It's a draw!
```

5. Write a program to Implement A* Algorithm.

```
package astar;
import java.util.*;
public class AStar {

 public static final int DIAGONAL_COST = 14;
 public static final int V_H_COST = 10;

 static class Cell {

  int heuristicCost = 0; //Heuristic cost
  int finalCost = 0; //G+H
  int i, j;
  Cell parent;

  Cell(int i, int j) {
   this.i = i;
   this.j = j;
  }

  @Override
  public String toString() {
   return "[" + this.i + ", " + this.j + "]";
  }
 }

 static Cell[][] grid = new Cell[5][5];

 static PriorityQueue<Cell> open;

 static boolean closed[][];
 static int startI, startJ;
 static int endI, endJ;

 public static void setBlocked(int i, int j) {
  grid[i][j] = null;
 }

 public static void setStartCell(int i, int j) {
  startI = i;
  startJ = j;
 }

 public static void setEndCell(int i, int j) {
  endI = i;
  endJ = j;
 }
```

```
static void checkAndUpdateCost(Cell current, Cell t, int cost) {
  if (t == null || closed[t.i][t.j]) {
    return;
  }
  int t_final_cost = t.heuristicCost + cost;

  boolean inOpen = open.contains(t);
  if (!inOpen || t_final_cost < t.finalCost) {
    t.finalCost = t_final_cost;
    t.parent = current;
    if (!inOpen) {
      open.add(t);
    }
  }
}

public static void AStar() {

  open.add(grid[startI][startJ]);

  Cell current;

  while (true) {
    current = open.poll();
    if (current == null) {
      break;
    }
    closed[current.i][current.j] = true;

    if (current.equals(grid[endI][endJ])) {
      return;
    }

    Cell t;
    if (current.i - 1 >= 0) {
      t = grid[current.i - 1][current.j];
      checkAndUpdateCost(current, t, current.finalCost + V_H_COST);

      if (current.j - 1 >= 0) {
        t = grid[current.i - 1][current.j - 1];
        checkAndUpdateCost(current, t, current.finalCost + DIAGONAL_COST);
      }

      if (current.j + 1 < grid[0].length) {
        t = grid[current.i - 1][current.j + 1];
        checkAndUpdateCost(current, t, current.finalCost + DIAGONAL_COST);
      }
    }
```

```
    if (current.j - 1 >= 0) {
      t = grid[current.i][current.j - 1];
      checkAndUpdateCost(current, t, current.finalCost + V_H_COST);
    }

    if (current.j + 1 < grid[0].length) {
      t = grid[current.i][current.j + 1];
      checkAndUpdateCost(current, t, current.finalCost + V_H_COST);
    }

    if (current.i + 1 < grid.length) {
      t = grid[current.i + 1][current.j];
      checkAndUpdateCost(current, t, current.finalCost + V_H_COST);

      if (current.j - 1 >= 0) {
        t = grid[current.i + 1][current.j - 1];
        checkAndUpdateCost(current, t, current.finalCost + DIAGONAL_COST);
      }

      if (current.j + 1 < grid[0].length) {
        t = grid[current.i + 1][current.j + 1];
        checkAndUpdateCost(current, t, current.finalCost + DIAGONAL_COST);
      }
    }
  }
}
public static void test(int tCase, int x, int y, int si, int sj, int ei, int ej, int[][] blocked) {
  System.out.println("\n\nTest Case #" + tCase);
  grid = new Cell[x][y];
  closed = new boolean[x][y];
  open = new PriorityQueue<>((Object o1, Object o2) -> {
    Cell c1 = (Cell) o1;
    Cell c2 = (Cell) o2;

    return c1.finalCost < c2.finalCost ? -1
          : c1.finalCost > c2.finalCost ? 1 : 0;
  });
  setStartCell(si, sj);  //Setting to 0,0 by default. Will be useful for the UI part

  setEndCell(ei, ej);

  for (int i = 0; i < x; ++i) {
    for (int j = 0; j < y; ++j) {
      grid[i][j] = new Cell(i, j);
      grid[i][j].heuristicCost = Math.abs(i - endI) + Math.abs(j - endJ);
    }
  }
```

```
grid[si][sj].finalCost = 0;

for (int i = 0; i < blocked.length; ++i) {
 setBlocked(blocked[i][0], blocked[i][1]);
}
System.out.println("Grid: ");
for (int i = 0; i < x; ++i) {
 for (int j = 0; j < y; ++j) {
  if (i == si && j == sj) {
   System.out.print("SO  "); //Source
  } else if (i == ei && j == ej) {
   System.out.print("DE  ");  //Destination
  } else if (grid[i][j] != null) {
   System.out.printf("%-3d ", 0);
  } else {
   System.out.print("BL  ");
  }
 }
 System.out.println();
}
System.out.println();

AStar();
System.out.println("\nScores for cells: ");
for (int i = 0; i < x; ++i) {
 for (int j = 0; j < x; ++j) {
  if (grid[i][j] != null) {
   System.out.printf("%-3d ", grid[i][j].finalCost);
  } else {
   System.out.print("BL  ");
  }
 }
 System.out.println();
}
System.out.println();

if (closed[endI][endJ]) {
 System.out.println("Path: ");
 Cell current = grid[endI][endJ];
 System.out.print(current);
 while (current.parent != null) {
  System.out.print(" -> " + current.parent);
  current = current.parent;
 }
 System.out.println();
} else {
 System.out.println("No possible path");
}
```

```
  }

  public static void main(String[] args) throws Exception {
    test(1, 5, 5, 0, 0, 3, 2, new int[][]{{0, 4}, {2, 2}, {3, 1}, {3, 3}});
    test(2, 5, 5, 0, 0, 4, 4, new int[][]{{0, 4}, {2, 2}, {3, 1}, {3, 3}});
    test(3, 7, 7, 2, 1, 5, 4, new int[][]{{4, 1}, {4, 3}, {5, 3}, {2, 3}});

    test(1, 5, 5, 0, 0, 4, 4, new int[][]{{3, 4}, {3, 3}, {4, 3}});
  }
}
```

OUTPUT:-

```
Test Case #1
Grid:
SO  0   0   0   BL
0   0   0   0   0
0   0   BL  0   0
0   BL  DE  BL  0
0   0   0   0   0


Scores for cells:
0   14  27  41  BL
14  17  29  42  56
27  29  BL  45  59
39  BL  43  BL  0
52  55  0   0   0

Path:
[3, 2] -> [2, 1] -> [1, 1] -> [0, 0]


Test Case #2
Grid:
SO  0   0   0   BL
0   0   0   0   0
0   0   BL  0   0
0   BL  0   BL  0
0   0   0   0   DE


Scores for cells:
0   17  33  48  BL
17  20  35  49  62
33  35  BL  52  64
48  BL  52  BL  67
62  65  64  67  77

Path:
```

[4, 4] -> [3, 4] -> [2, 3] -> [1, 2] -> [1, 1] -> [0, 0]

Test Case #3
Grid:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 SO 0 BL 0 0 0
0 0 0 0 0 0 0
0 BL 0 BL 0 0 0
0 0 0 BL DE 0 0
0 0 0 0 0 0 0

Scores for cells:
40 35 37 40 53 68 0
22 17 20 34 48 63 0
17 0 15 BL 48 61 75
20 15 18 31 43 56 70
34 BL 31 BL 46 58 73
48 48 43 BL 56 61 0
63 61 56 59 0 0 0

Path:
[5, 4] -> [4, 4] -> [3, 3] -> [3, 2] -> [2, 1]

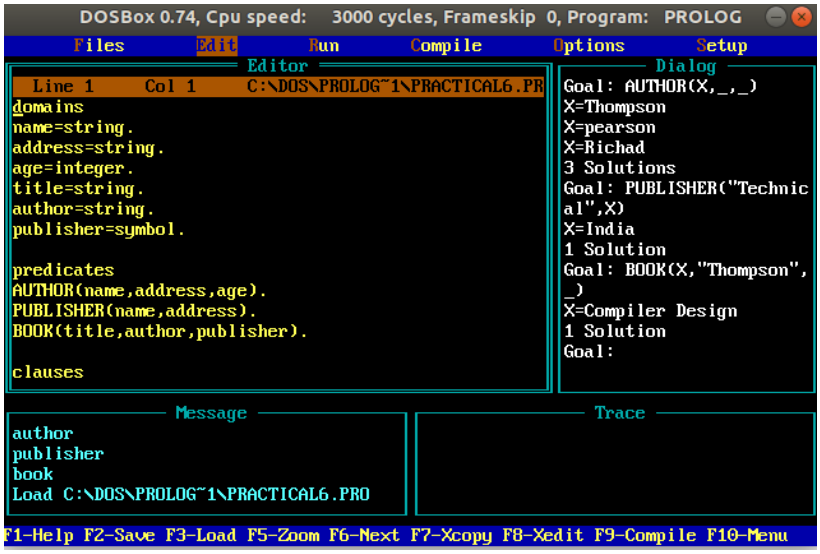
Test Case #1
Grid:
SO 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 BL BL
0 0 0 BL DE

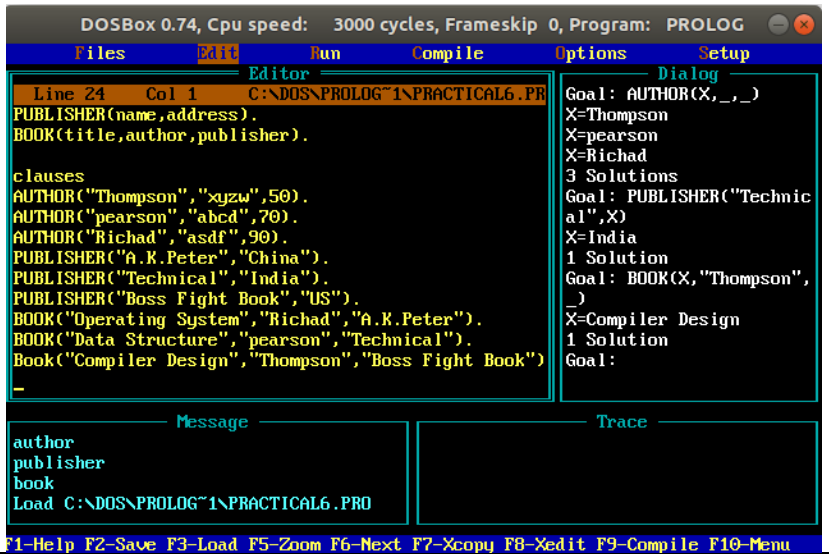
Scores for cells:
0 17 33 48 62
17 20 35 49 62
33 35 38 51 63
48 49 51 BL BL
62 62 63 BL 0

No possible path

6	**Write a program in Prolog that will answer the question for the following facts.** **Author (name,address,age)** **Publisher (name,address)** **Book (title,author,publisher)** **a. What are the names of all authors?** **b. What is the address of publisher abc?** **c. What are the titles published by abc?**
	Domains name=string. address=string. age=integer. title=string. author=string. publisher=symbol. Predicates AUTHOR(name,address,age). PUBLISHER(name,address). BOOK(title,author,publisher). Clauses AUTHOR("Thompson","xyzw",50). AUTHOR("pearson","abcd",70). AUTHOR("Richad","asdf",90). PUBLISHER("A.K.Peter","China"). PUBLISHER("Technical","India"). PUBLISHER("Boss Fight Book","US"). BOOK("Operating System","Richad","A.K.Peter"). BOOK("Data Structure","pearson","Technical"). Book("Compiler Design","Thompson","Boss Fight Book").

OUTPUT :-


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: PROLOG
Files Edit Run Compile Options Setup
Editor
Line 1 Col 1 C:\DOS\PROLOG~1\PRACTICAL6.PR
domains
name=string.
address=string.
age=integer.
title=string.
author=string.
publisher=symbol.
predicates
AUTHOR(name,address,age).
PUBLISHER(name,address).
BOOK(title,author,publisher).
clauses
Dialog
Goal: AUTHOR(X,_,_)
X=Thompson
X=pearson
X=Richad
3 Solutions
Goal: PUBLISHER("Technical",X)
X=India
1 Solution
Goal: BOOK(X,"Thompson",_)
X=Compiler Design
1 Solution
Goal:
Message
author
publisher
book
Load C:\DOS\PROLOG~1\PRACTICAL6.PRO
Trace
F1-Help F2-Save F3-Load F5-Zoom F6-Next F7-Xcopy F8-Xedit F9-Compile F10-Menu

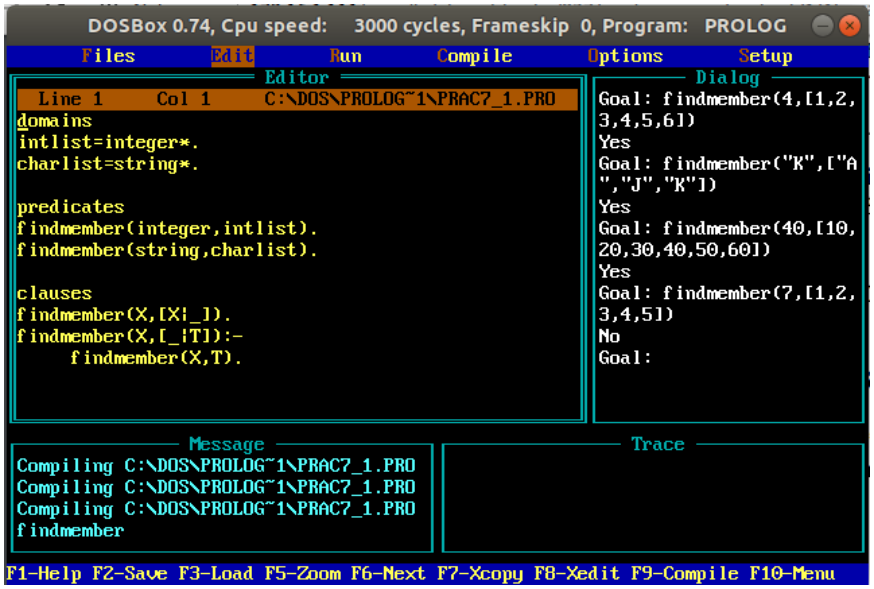

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: PROLOG
Files Edit Run Compile Options Setup
Editor
Line 24 Col 1 C:\DOS\PROLOG~1\PRACTICAL6.PR
PUBLISHER(name,address).
BOOK(title,author,publisher).
clauses
AUTHOR("Thompson","xyzw",50).
AUTHOR("pearson","abcd",70).
AUTHOR("Richad","asdf",90).
PUBLISHER("A.K.Peter","China").
PUBLISHER("Technical","India").
PUBLISHER("Boss Fight Book","US").
BOOK("Operating System","Richad","A.K.Peter").
BOOK("Data Structure","pearson","Technical").
Book("Compiler Design","Thompson","Boss Fight Book")
Dialog
Goal: AUTHOR(X,_,_)
X=Thompson
X=pearson
X=Richad
3 Solutions
Goal: PUBLISHER("Technical",X)
X=India
1 Solution
Goal: BOOK(X,"Thompson",_)
X=Compiler Design
1 Solution
Goal:
Message
author
publisher
book
Load C:\DOS\PROLOG~1\PRACTICAL6.PRO
Trace
F1-Help F2-Save F3-Load F5-Zoom F6-Next F7-Xcopy F8-Xedit F9-Compile F10-Menu

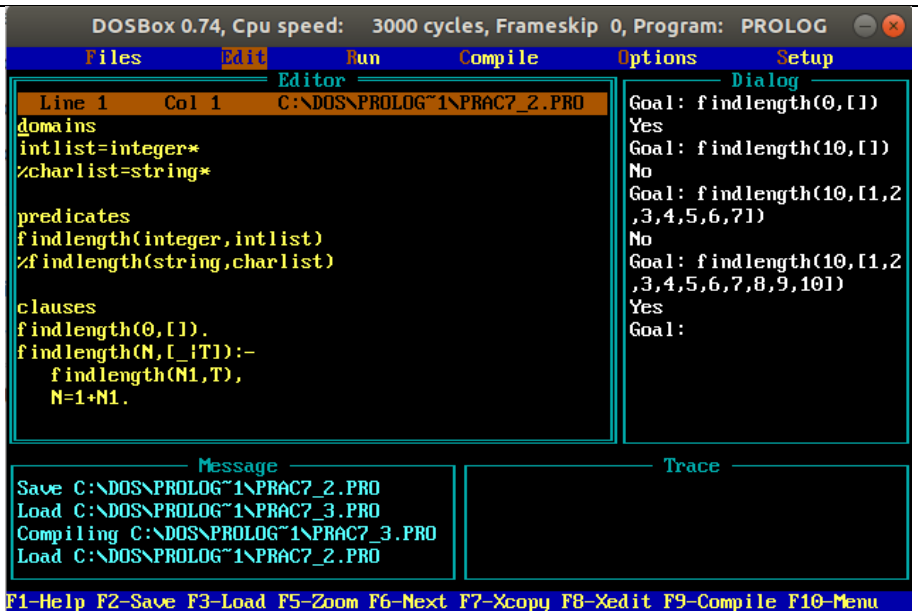
7	**Write a program in Prolog to find,** **A)Member of a list** **B)The length of an input list.** **C)Concatenation of two list** **D)Reverse of a list.** **E)Delete an item from list**

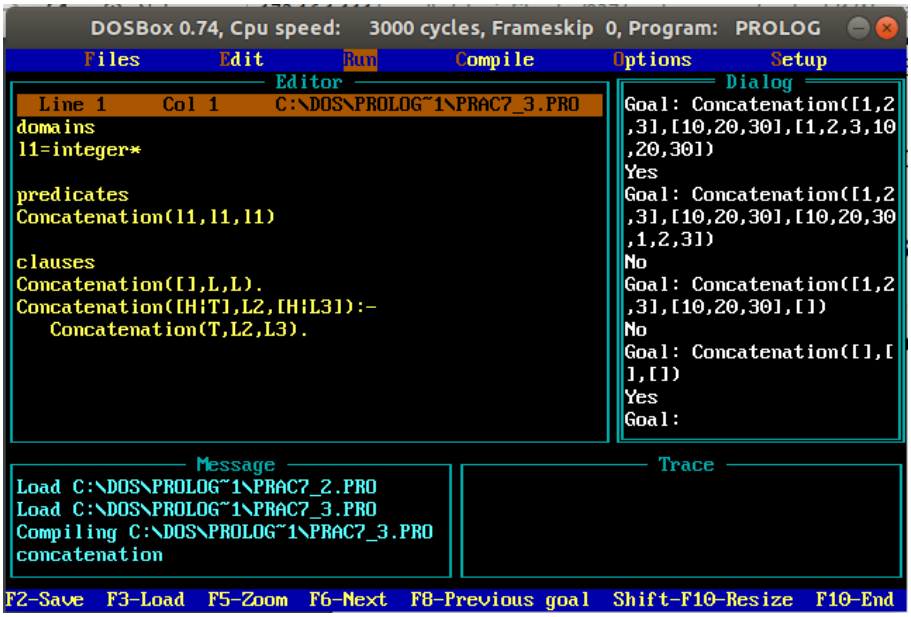
```
//member of the list

domains
intlist=integer*
charlist=string*

predicates
findmember(integer,inntlist)
findmember(string,charlist)

clauses
findmember(X,[X|_]).
findmember(X,[_|T]):-
   findmember(X,T).


//length of an input list

domains
intlist=integer*
charlist=string*

predicates
findlength(integer,intlist)
findlength(string,charlist)

clauses
findlength(0,[]).
findlength(N,[_|T]):-
  findlength(N1,T),
  N=1+N1.


//Concatenation of the list

domains
l1=integer*

predicates
Concatenation(l1,l1,l1)

clauses
```

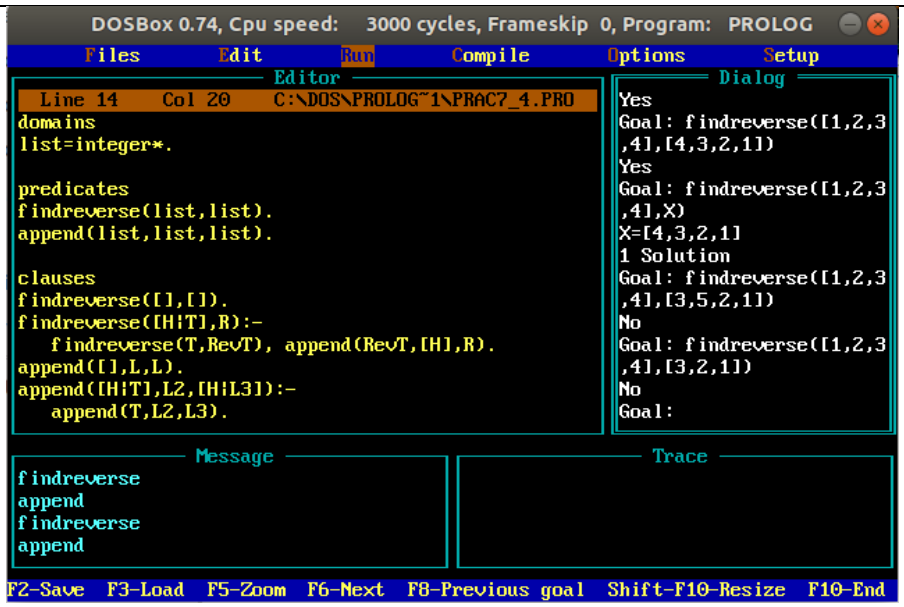
```
Concatenation([],L,L).
Concatenation([H|T],L2,[H|L3]):-
  Concatenation(T,L2,L3).


//Delete the element from the list

domains
list=symbol*

predicates
delete(integer,list,list)

clauses
delete(X,[X|T1],T1).
delete(X,[Y|T1],[Y|T2]):-
  delete(X,T1,T2).


//Reverse of the list.

domains
list=integer*

predicates
findreverse(list,list)

clauses
findreverse()
```

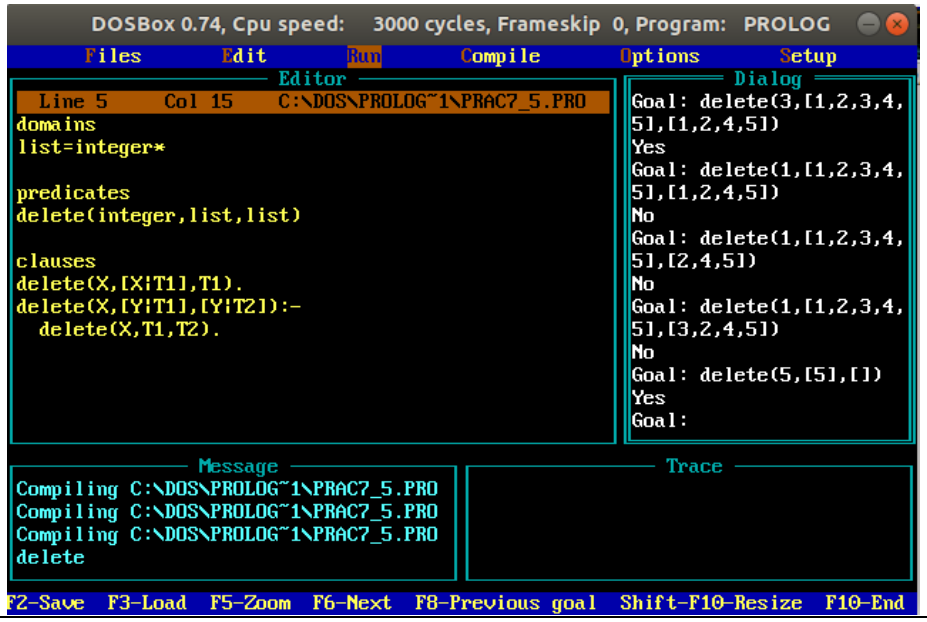
OUTPUT :-




DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: PROLOG
Files Edit Run Compile Options Setup
Editor
Line 1 Col 1 C:\DOS\PROLOG~1\PRAC7_2.PRO
domains
intlist=integer*
%charlist=string*
predicates
findlength(integer,intlist)
%findlength(string,charlist)
clauses
findlength(0,[]).
findlength(N,[_|T]):-
findlength(N1,T),
N=1+N1.
Dialog
Goal: findlength(0,[])
Yes
Goal: findlength(10,[])
No
Goal: findlength(10,[1,2,3,4,5,6,7])
No
Goal: findlength(10,[1,2,3,4,5,6,7,8,9,10])
Yes
Goal:
Message
Save C:\DOS\PROLOG~1\PRAC7_2.PRO
Load C:\DOS\PROLOG~1\PRAC7_3.PRO
Compiling C:\DOS\PROLOG~1\PRAC7_3.PRO
Load C:\DOS\PROLOG~1\PRAC7_2.PRO
Trace
F1-Help F2-Save F3-Load F5-Zoom F6-Next F7-Xcopy F8-Xedit F9-Compile F10-Menu


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: PROLOG
Files Edit Run Compile Options Setup
Editor
Line 1 Col 1 C:\DOS\PROLOG~1\PRAC7_3.PRO
domains
l1=integer*
predicates
Concatenation(l1,l1,l1)
clauses
Concatenation([],L,L).
Concatenation([H|T],L2,[H|L3]):-
Concatenation(T,L2,L3).
Dialog
Goal: Concatenation([1,2,3],[10,20,30],[1,2,3,10,20,30])
Yes
Goal: Concatenation([1,2,3],[10,20,30],[10,20,30,1,2,3])
No
Goal: Concatenation([1,2,3],[10,20,30],[])
No
Goal: Concatenation([],[],[])
Yes
Goal:
Message
Load C:\DOS\PROLOG~1\PRAC7_2.PRO
Load C:\DOS\PROLOG~1\PRAC7_3.PRO
Compiling C:\DOS\PROLOG~1\PRAC7_3.PRO
concatenation
Trace
F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End

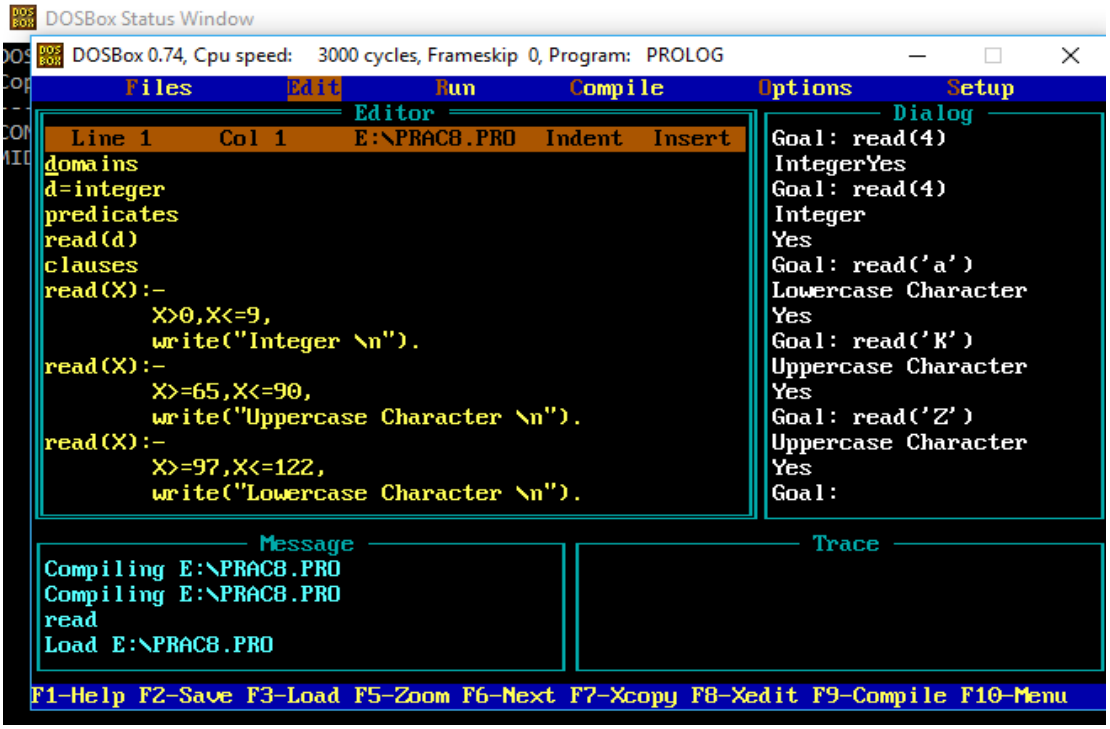

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: PROLOG
Files Edit Run Compile Options Setup
Editor
Line 14 Col 20 C:\DOS\PROLOG~1\PRAC7_4.PRO
domains
list=integer*.
predicates
findreverse(list,list).
append(list,list,list).
clauses
findreverse([],[]).
findreverse([H|T],R):-
findreverse(T,RevT), append(RevT,[H],R).
append([],L,L).
append([H|T],L2,[H|L3]):-
append(T,L2,L3).
Dialog
Yes
Goal: findreverse([1,2,3,4],[4,3,2,1])
Yes
Goal: findreverse([1,2,3,4],X)
X=[4,3,2,1]
1 Solution
Goal: findreverse([1,2,3,4],[3,5,2,1])
No
Goal: findreverse([1,2,3,4],[3,2,1])
No
Goal:
Message
findreverse
append
findreverse
append
Trace
F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: PROLOG
Files Edit Run Compile Options Setup
Editor
Line 5 Col 15 C:\DOS\PROLOG~1\PRAC7_5.PRO
domains
list=integer*
predicates
delete(integer,list,list)
clauses
delete(X,[X|T1],T1).
delete(X,[Y|T1],[Y|T2]):-
delete(X,T1,T2).
Dialog
Goal: delete(3,[1,2,3,4,5],[1,2,4,5])
Yes
Goal: delete(1,[1,2,3,4,5],[1,2,4,5])
No
Goal: delete(1,[1,2,3,4,5],[2,4,5])
No
Goal: delete(1,[1,2,3,4,5],[3,2,4,5])
No
Goal: delete(5,[5],[])
Yes
Goal:
Message
Compiling C:\DOS\PROLOG~1\PRAC7_5.PRO
Compiling C:\DOS\PROLOG~1\PRAC7_5.PRO
Compiling C:\DOS\PROLOG~1\PRAC7_5.PRO
delete
Trace
F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End

8	**Write a Program in a Prolog for reading in a character and decide whether it is a digit or an alpha numeric character.**

```
Domains
d=integer
predicates
read(d)
clauses
read(X):-
        X>0,X<=9,
        write("Integer \n").
read(X):-
        X>=65,X<=90,
        write("Uppercase Character \n").
read(X):-
        X>=97,X<=122,
        write("Lowercase Character \n").
```

OUTPUT:-

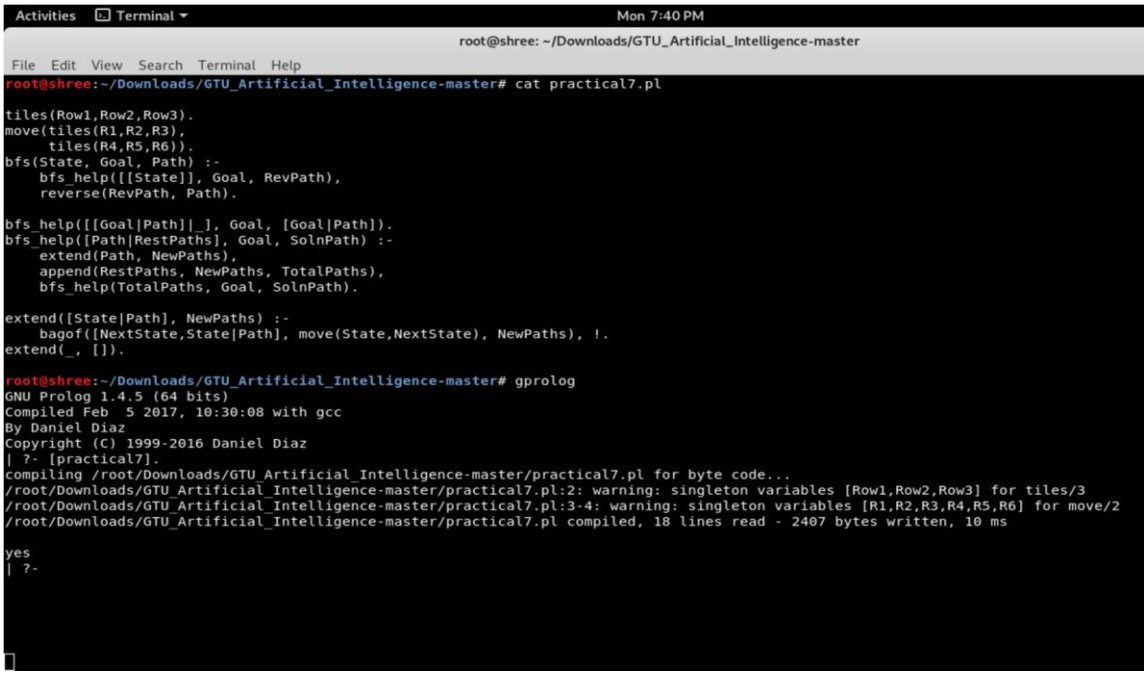


9 **Write a program to solve N-Queens problem using Prolog.**

```
queens(N, Queens) :-
   length(Queens, N),
        board(Queens, Board, 0, N, _, _),
        queens(Board, 0, Queens).

board([], [], N, N, _, _).
board([_|Queens], [Col-Vars|Board], Col0, N, [_|VR], VC) :-
        Col is Col0+1,
        functor(Vars, f, N),
        constraints(N, Vars, VR, VC),
        board(Queens, Board, Col, N, VR, [_|VC]).

constraints(0, _, _, _) :- !.
constraints(N, Row, [R|Rs], [C|Cs]) :-
        arg(N, Row, R-C),
        M is N-1,
        constraints(M, Row, Rs, Cs).

queens([], _, []).
queens([C|Cs], Row0, [Col|Solution]) :-
        Row is Row0+1,
        select(Col-Vars, [C|Cs], Board),
        arg(Row, Vars, Row-Row),
        queens(Board, Row, Solution).
```

OUTPUT:-

```
?- queens(8, Queens).
Queens = [1, 5, 8, 6, 3, 7, 2, 4] .

?- queens(4, Queens).
Queens = [2, 4, 1, 3]
```

10	**Write a program to solve 8 puzzle problem using Prolog.**
	tiles(Row1,Row2,Row 3). move(tiles(R1,R2,R3), tiles(R4,R5,R6)). bfs(State, Goal, Path) : bfs_help([[State]], Goal, RevPath), reverse(RevPath, Path). bfs_help([[Goal|Path]|_], Goal, [Goal|Path]). bfs_help([Path|RestPaths], Goal, SolnPath) : extend(Path, NewPaths), append(RestPaths, NewPaths, TotalPaths), bfs_help(TotalPaths, Goal, SolnPath). extend([State|Path], NewPaths) : bagof([NextState,State|Path], move(State,NextState), NewPaths), !. extend(_, []).
	OUTPUT:- 

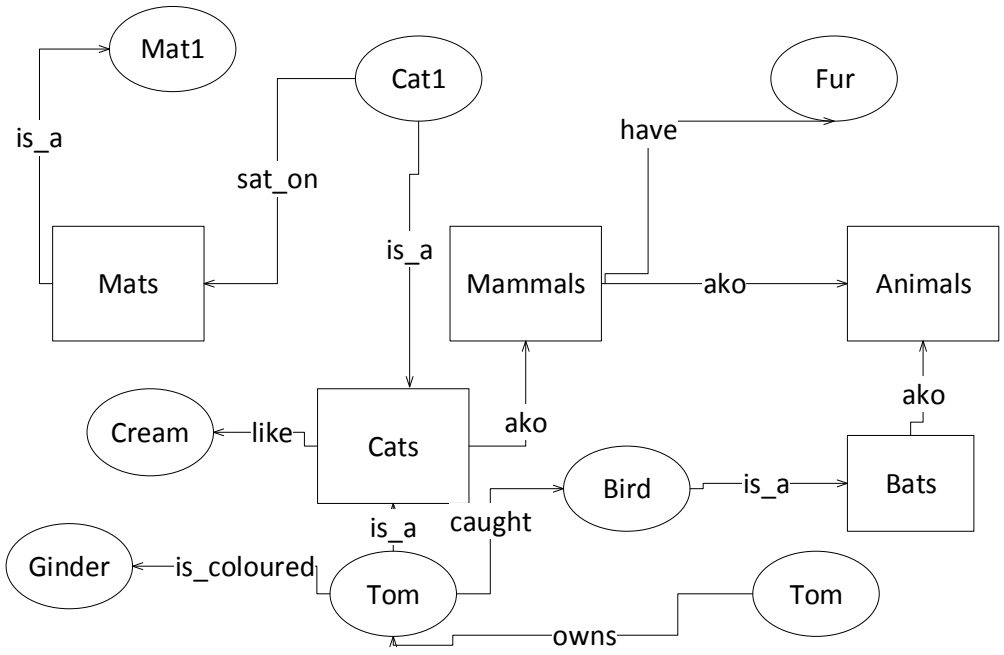
11 **Write a program to solve travelling salesman problem using Prolog.**

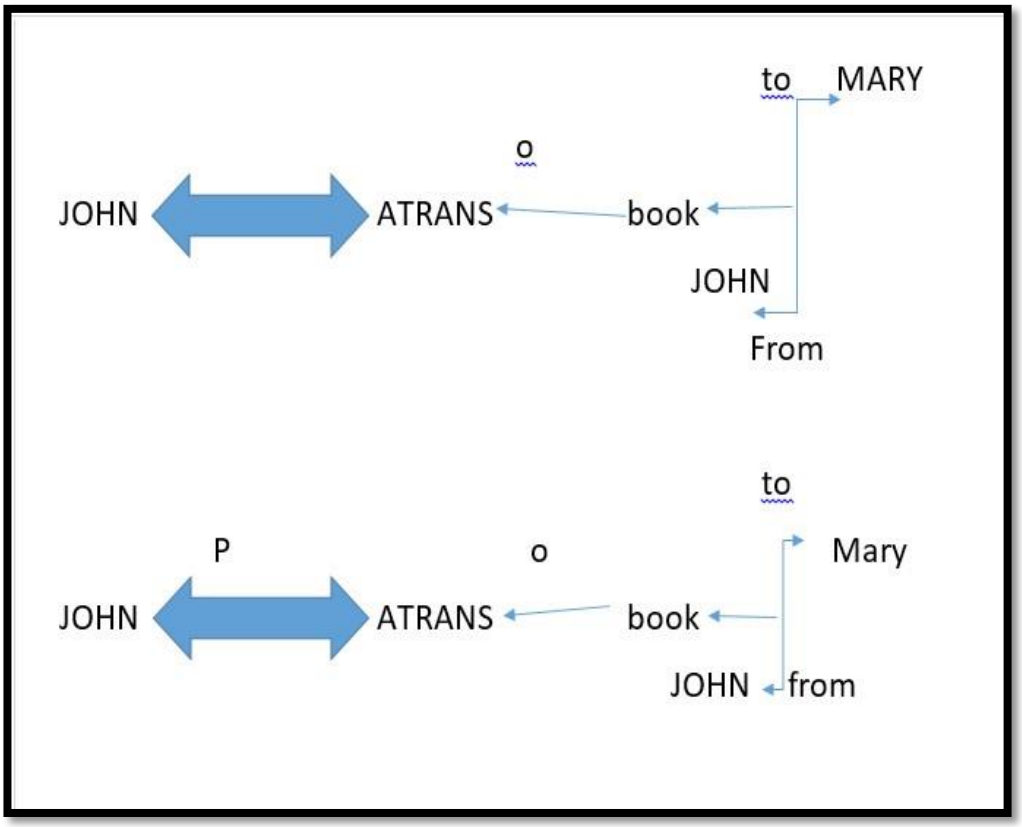
```
Domains

  town = symbol
  distance = unsigned
  rib = r(town,town,distance)
  tlist = town*
  rlist = rib*
predicates
            nondeterm way(town,town,rlist,distance)
  nondeterm route(town,town,rlist,tlist,distance)
  nondeterm route1(town,tlist,rlist,tlist,distance)
  nondeterm ribsmember(rib,rlist)
  nondeterm townsmember(town,tlist)
  nondeterm tsp(town,town,tlist,rlist,tlist,distance)
  nondeterm ham(town,town,tlist,rlist,tlist,distance)
  nondeterm shorterRouteExists(town,town,tlist,rlist,distance)
  nondeterm alltown(tlist,tlist)
  nondeterm write_list(tlist)
clauses
  write_list([]).
  write_list([H|T]):-
    write(H,' '),
    write_list(T).

  townsmember(X,[X|_]).
  townsmember(X,[_|L]):-
    townsmember(X,L).

  ribsmember(r(X,Y,D),[r(X,Y,D)|_]).
  ribsmember(X,[_|L]):-
    ribsmember(X,L).

  alltown(_,[]).
  alltown(Route,[H|T]):-
    townsmember(H,Route),
    alltown(Route,T).

  way(Town1,Town2,Ways,OutWayDistance):-
    ribsmember(r(Town1,Town2,D),Ways),
    OutWayDistance = D.

  way(Town1,Town2,Ways,OutWayDistance):-
    ribsmember(r(Town2,Town1,D),Ways),    OutWayDistance = D.

  route(Town1,Town2,Ways,OutRoute,OutDistance):-
```

```
  route1(Town1,[Town2],Ways,OutRoute,T1T2Distance),

 route1(Town1,[Town1|Route1],_,[Town1|Route1],OutDistance):-
  OutDistance = 0.

 route1(Town1,[Town2|PassedRoute],Ways,OutRoute,OutDistance):-
  way(TownX,Town2,Ways,WayDistance),
  not(townsmember(TownX,PassedRoute)),
  route1(Town1,[TownX,Town2|PassedRoute],Ways,OutRoute,CompletingRoadDistance),
  OutDistance = CompletingRoadDistance + WayDistance.

 shorterRouteExists(Town1,Town2,Towns,Ways,Distance):-
   ham(Town1,Town2,Towns,Ways,_,Other),
     Other < Distance.

 tsp(Town1,Town1,Towns,Ways,BestRoute,MinDistance):-
  way(OtherTown,Town1,Ways,_),
    tsp(Town1,OtherTown,Towns,Ways,BestRoute,MinDistance).

 tsp(Town1,Town2,Towns,Ways,BestRoute,MinDistance):-
    ham(Town1,Town2,Towns,Ways,Route,Distance),
    not(shorterRouteExists(Town1,Town2,Towns,Ways,Distance)),
  BestRoute = Route,
  MinDistance = Distance.

 ham(Town1,Town2,Towns,Ways,Route,Distance):-
  route(Town1,Town2,Ways,Route,Distance),

  alltown(Route,Towns), % if you want simple road without including all towns you could
uncomment this line
  write_list(Route),
  write(" tD = ",Distance,"n").
%  fail.
```

OUTPUT :-

```
 AllTowns = [a,b,c,d],
 AllWays = [r(a,b,1),r(a,c,10),r(c,b,2),r(b,c,2),r(b,d,5),r(c,d,3),r(d,a,4)],

Output:
 a e d b c     D = 15
 a e d b c     D = 15
 a d e b c     D = 24
 a e b d c     D = 25
 a b e d c     D = 27
 a d b e c     D = 31
 a b d e c     D = 24
 Finally:
a e d b c     MIN_D = 15
```

12	**Convert following Prolog predicates into Semantic Net:** **cat(tom).** **cat(cat1).** **mat(mat1).** **sat_on(cat1,mat1).** **bird(bird1).** **caught(tom,bird1).** **like(X,cream) :– cat(X).** **mammal(X) :– cat(X).** **has(X,fur) :– mammal(X).** **animal(X) :– mammal(X).** **animal(X) :– bird(X).** **owns(john,tom).** **is_coloured(tom,ginger).**
	

13:	Write the Conceptual Dependency for following statements. (a) John gives Mary a book. (b) John gave Mary the book yesterday
Solution:	 Arrows indicate the direction of dependency. Double arrows (⇔) indicate *two-way* links between the actor (PP) and action (ACT). Letters above indicate certain relationships: O : object R : recipient-donor. P : past