INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

APPLIED ARTIFICIAL INTELLIGENCE ALGORITHMS LABORATORY								
IV Semester: CSE (AI & ML)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
ACAD04	**Core**	0	0	2	1	40	60	100
Contact Classes: NIL	**Tutorial Classes: NIL**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Python Programming								

I. COURSE OVERVIEW:

Applied Artificial Intelligence (Applied AI) refers to the practical implementation and utilization artificial intelligence (AI) technologies to solve real-world problems and address specific challenges in various domains. The objective of this laboratory course for applied artificial intelligence (AI) typically aims to provide students with hands-on experience in applying AI techniques. Applications provide a foundation for the implementation of applied artificial intelligence across various industries, addressing specific challenges and delivering practical solutions the scope and applications of applied AI will vary based on the specific needs and challenges within different industries and sectors.

II. COURSES OBJECTIVES:

The students will try to learn:

I	The suitability of different search algorithms, unification strategies, and knowledge representation Techniques for given problems.
II	The practical applications of forward chaining in expert systems, decision support, and fundamental probability concepts essential for Bayesian inference.
III	The minimax algorithm as a basic approach for decision-making in two-player zero-sum games, Strategic considerations and algorithmic implementations clearly.
IV	The essential aspects of setting up AI environments, implementing local communication protocols and ensuring secure message verification in the context of AI agents

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Develop programming skills by implementing algorithms related to local search, basic search, and unification through arithmetic design and coding.
CO2	Utilize coding techniques to model and solve problems in AI, emphasizing practical implementation of theoretical concepts.
CO3	Make use of the Bayesian network inference techniques to solve real-world problems in diverse domains, including healthcare, finance, and decision support.
CO4	Demonstrate a comprehensive understanding of fuzzy logic and its application in control systems.
CO5	Identify intelligent agents to address specific challenges in practical domains, demonstrating the versatility of AI techniques.
CO6	Solve techniques for encoding and decoding messages exchanged between AI agents, addressing considerations such as data serialization, compression, and encryption.

IV. COURSE CONTENT:

EXERCISES FOR APPLIED ARTIFICIAL INTELLIGENCE LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Installing Python and other supportive libraries

1. Install Python 3 on your machine by typing the following command on the terminal:

   $ python3 --version

2. Install various useful packages by following the relevant links:

   NumPy: http://docs.scipy.org/doc/numpy-1.10.1/user/install.html
   SciPy: http://www.scipy.org/install.html
   scikit-learn: http://scikit-learn.org/stable/install.html
   matplotlib: http://matplotlib.org/1.4.2/users/installing.html

   Note: If you are on Windows, you should have installed a SciPy-stack compatible version of Python 3

3. Install a couple of packages before starting logic programming in python like logpy and sympy using pip. These packages are useful to work with matching mathematical expressions.

   $ pip3 install logpy
   $ pip3 install sympy

   If you get an error during the installation process for logpy, you can install from source at https://github.com/logpy/logpy.

Try: Practice appropriate python commands to build a model and use the above libraries (relevant packages) to interact with the data. The commands may include importing of packages containing all the datasets, loading of datasets, printing data, printing labels, loading images, extracting a specific image etc.

1.2 Knowledge Representation using FOL - Translate

The goal of this exercise is to translate each of the following sentences into First Order Logic (FOL). Later convert the FOL into a prolog program and asking questions.

Input: Domain Knowledge like:

(a) Not all cars have carburetors.
(b) Some people are either religious or pious.
(c) No dogs are intelligent.
(d) All babies are illogical.
(e) Every number is either negative or has a square root.
(f) Some numbers are not real.

(g) Every connected and circuit-free graph is a tree.

Output: Equivalent FOL statement.

Hints

$\neg\forall x\ [car(x) \rightarrow carburetors(x)]$ or
$\exists x\ [car(x) \wedge \neg carburetors(x)]$

```
# Translate the other statements referring the above
```

Try: Translate each of the following sentences into First Order Logic (FOL):

(a) Not every graph is connected.
(b) All that glitters is not gold.
(c) Not all that glitters is gold.
(d) There is a barber who shaves all men in the town who do not shave themselves.
(e) There is no business-like show business.

1.3 Knowledge Representation using FOL - Translate

The goal of this exercise is to rewrite each proposition symbolically, given that the universe of discourse is a set of real numbers.

Input: Domain Knowledge like:

(a) For each integer x, there exist an integer y such that x + y = 0.
(b) There exist an integer x such that x + y = y for every integer y.
(c) For all integers x and y, x.y = y.x
(d) There are integers x and y such that x+y=5.

Output: Equivalent FOL statement

Hints

$(\forall x \in \mathbb{Z})(\exists y \in \mathbb{Z})(x + y = 0)$.
We could read this as, "For every integer x, there exists an integer y such that $x + y = 0$." This is a true statement.

```
# Translate the other statements referring the above
```

Try: Using FOL, express the following:

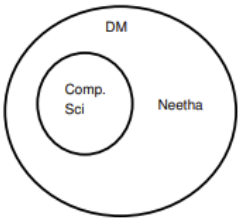
(a) Every student in this class has taken exactly two mathematics courses at this school.
(b) Someone has visited every country in the world except Libya.
(c) No one has climbed every mountain in the Himalayas.

1.4 Check the Availability

Every computer science student takes discrete mathematics. Neetha is taking discrete mathematics. Therefore, Neetha is a computer science student. The given conclusion is false. The following Venn diagram is a counter example for the given conclusion.

If it does not rain or it is not foggy then the sailing race will be held, and lifesaving demonstrations will go on. If the sailing race is held, then the trophy will be awarded. The trophy was not awarded. Therefore, it rained. The goal of this exercise is to translate each of the following sentences into First Order Logic (FOL)

Input: Venn Diagram as a counter example for the given conclusion.



Output: Prove that the above statements are TRUE.

Hints

```
# Consider the below statements and infer the statement by the following arguments
```

premise	$\neg R \vee \neg F \rightarrow S \wedge D$	$\ldots(1)$
premise	$S \rightarrow T$	$\ldots(2)$
premise	$\neg T$	$\ldots(3)$
1	$\neg R \vee \neg F \rightarrow S$	$\ldots(4)$
4, 2	$\neg R \vee \neg F \rightarrow T$	$\ldots(5)$
5	$\neg T \rightarrow \neg(\neg R \vee \neg F)$	$\ldots(6)$
6, 3	$\neg(\neg R \wedge \neg F)$	$\ldots(7)$
7	$R \wedge F$	$\ldots(8)$
8	R	

Try: Prove or Disprove: All doctors are college graduates. Some doctors are not golfers. Hence, some golfers are not college graduates.

1.5 Rewrite the sentences.

The goal of this exercise is to translate each of the following sentences into First Order Logic (FOL). Later convert the FOL into a prolog program and asking questions.

Input:

(a) Some boys are sharp and intelligent.
UOD(x): all persons.
Sharp(x): x is sharp.
Boy(x): x is a boy.
Intelligent(x): x is intelligent.

(b) Not all boys are intelligent.

(c) Some students of DM course have cleared JEE main and the rest cleared SAT.
UOD(x): all persons.
ClearJEE(x): x clears JEE main.
ClearSAT(x): x clears SAT.

(d) Something that is white is not always milk, whereas the milk is always white.
UOD(x): things.
White(x): x is white.
M ilk(x): x is milk.

(e) Breakfast is served in mess on all days between 7am and 9am except Sunday. And, on Sundays it is served till 9.15 am. UOD(x): days. Day(x): x is a day of the week. Breakfast-time-non-sunday(x): Breakfast is served in mess on x between 7am and 9am. Breakfast-time-sunday(x): Breakfast is served in mess on x till 9.15am.

Output: Equivalent FOL statement.

Hints

$\exists x \text{boys}(x) \wedge \text{intelligent}(x)$

```
# Translate the other statements referring the above
```

Try: Translate each of the following sentences into First Order Logic (FOL):

(a) The speed of light is not same in all mediums. The speed of light in fiber is 2×108 m/s. Therefore, there exists at least two mediums having different speed of light. UOD(x): mediums. Medium(x): Light travels in medium x. Speed(x): Speed of light in medium x. P: Speed of light in fiber is 2 × 108 m/s.

(b) Some students have joined IIITDM. There exists a student who has not joined any IIITDM. Not all students have cleared JEE advanced. Therefore, some students have joined deemed universities. UOD(x) : people. UOD(y) : Educational institutes. Stud(x): x is a student. IIIT DM(y) : y is a IIITDM. JoinIIIT DM(x, y) : x joins IIITDM y. ClearJEE(x) : x cleared JEE advanced. JoinDeemed(x) : x joins a deemed university.

1.6 Propositions

Identify propositions from the following. If not a proposition, justify, why it is not.

(a) I shall sleep or study.

(b) x 2 + 5x + 6 = 0 such that x ∈ integers.

Input: Propositions
Output: Justify the statements either to be a proposition or not

Hints

```
# The rule of logic allows to distinguish between valid and invalid arguments.
Example:
If x+1=5, then x=4=4. Therefore, if x≠4, then x+1≠5.
If I watch Monday night football, then I will miss the following Tuesday 8 a.m. class.
Therefore, if I do not miss my Tuesday 8 a.m. class, then I did not watch football the
previous Monday night.
# Use the same format:
If p then q. Therefore, if q is false then p is false.
If we can establish the validity of this type of argument, then we have proved at
once that both arguments are legitimate. In fact, we have also proved that any argument
using the same format is also credible.
# Use the above example and give the justifications
```

Try: Express the following in first order logic (identify the right universe of discourse, predicates before attempting each question. Think twice and do not oversimplify the problem)

(a) The fundamental law of nature is change.

(b) We cannot help everyone, but everyone can help someone.

(c) Power does not corrupt people, people corrupt power.

(d) It is nice of somebody to do something.

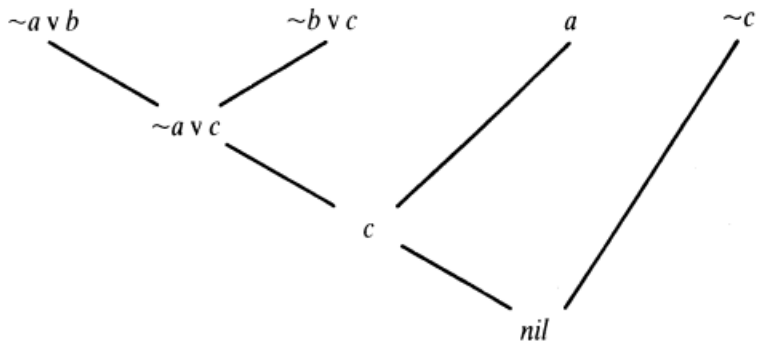
(e) No one who has no complete knowledge of himself will ever have a true understanding of another.

(f) Thought or thinking is what set human beings apart from other living things.

1.7 A Simple Theorem Prover – Resolution Principle

Implement a simple theorem prover as a pattern-directed system by limiting only proving theorem in the simple propositional logic just to illustrate the principle of resolution mechanism. Define the theorem proving as an extendable to handle the first-order predicate calculus.

Input: A formula as a theorem which is always true regardless of the interpretation of the symbols that occur in the formula.



Example:

p v ~p

read as 'p or not p', is always true regardless of the meaning of p. We will be using the following symbols as logic operations:

~ - negation, read as 'not'

& - conjunction, read as 'and'

v – disjunction, read as 'or'

= > - implication, read as 'implies'

Output: Prove that the following propositional formula is a theorem:

(a=>b)&(b=>c)=>(a=>c)

Hints

```
# Write the production rules for resolution theorem proving

# Contradicting the clauses
[ clause(X), clause(~X) ] --->
[ write('Contradiction found'), stop].

# Remove a true clause
```

```
# Simply the clause

# Resolution step, a special case

[clause(P), clause(C), delete(~P, C, C1), not done(P, C, P) ] --->
[assert(clause(C1)), assert(done(~P, C, P))].

# Repeat the above step for other special cases and write the last rule as a resolution
process stuck

# delete(P, E, E1) means: delete a disjunction subexpression P from E giving E1

# in(P,E) means: P is a disjunction subexpression in E

# Write the Translating a propositional formula into(asserted) clauses

# Write the Transformation rules for propositional formulas
```

Try: Implement an interpreter for pattern-directed programs that does not maintain its database as Prolog's own internal database (with assert and retract), but as a procedure argument according to the foregoing remark. Such a new interpreter would allow for automatic backtracking. Try to design a representation of the database that would facilitate efficient pattern matching.

2. Exercises on Logic Programming

2.1 Water Jug Problem

Given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can we get exactly 2 gallons of water into a 4- gallon jug?

Input: 4 3 2

Output: {(0,0),(0,3),(3,0),(3,3),(4,2),(0,2)}

a. Describe the state space as a set of ordered pairs of integers.
b. Generate production rules and perform basic operations to achieve the goal.
c. Initialize the start state and apply the rules iteratively until the goal state is reached.
d. Generate a search tree (Depth-First Search / Breadth-First Search)

Hints

```
/* Define a function to initialize the dictionary elements with a default value.*/

/* Initialize dictionary with default values as false.*/

/* Define a recursive function and print the intermediate steps to reach the final
solution and return the Boolean values.*/

/* Check whether the goal is achieved and return true if achieved. */

/* Check if you have already visited the combination or not. If not, then proceed
further. */
```

```
/* Check whether all the six possibilities and see if a solution is found in any one
of them. */

/* Return false if the combination is already visited to avoid repetition otherwise
recursion will enter an infinite loop*/

/* Call the function and pass the initial amount of water present in both the jugs. */
```

Try: Given two jugs, a 7-gallon one and a 11-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can we get exactly 7 gallons of water into a 7-gallon jug?

2.2 Monkey Banana Problem

Imagine a room containing a monkey, chair and some bananas that have been hung from the center of ceiling. If the monkey is clever enough, he can reach the bananas by placing the box directly below the bananas and climbing on the chair .The problem is to prove whether the monkey can reach the bananas. The monkey wants it but cannot jump high enough from the floor. At the window of the room there is a box that the monkey can use.

Input: The monkey can perform the following actions:-

1) Walk on the floor.
2) Climb the box.
3) Push the box around (if it is beside the box).
4) Grasp the banana if it is standing on the box directly under the banana.

Output:

a. Write down the initial state description and action schemes.
b. Prepare all the required predicates that will make the monkey to perform some action and move from one state to the other until the goal state is reached.
c. Set the initial position of the monkey (initial state) and raise questions to whether the knowledge represented can make the monkey get the banana.
d. Trace the flow of actions from initial state to goal state.

Hints

```
/* Define a class and initialize all the states */

/* Check if the goal state is achieved.*/

/* Return a list of possible actions given the current state */

/* Apply an action and return the resulting state. */

/* Solve the problem using breadth-first search by simultaneously checking the goal
state, generating the successors */

/* Return result if exist otherwise return None */
```

Try: Extensively make use of state space search to represent and solve Tic-Tac-Toe. Represent the problem as a state space and define the rules. In the state space, represent the starting state, set of legal moves, and the goal state.

2.3 Eight Puzzle Problem

The 8-puzzle consists of a 3×3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The task is to reach a specified goal state, such as the one shown on the right of the figure. The objective is to place the numbers on tiles to match the final configuration using the empty space. You can slide four adjacent (left, right, above, and below) tiles into the empty space.

Input:
Initial State

	2	3
1	4	6
7	5	8

Goal State

1	2	3
4	5	6
7	8	

Output: Goal state as a long output on the terminal just like:

```
Initial configuration
1-e-2
6-3-4
7-5-8

After moving 2 into the empty space
1-2-e
6-3-4
7-5-8

After moving 4 into the empty space
1-2-4
6-3-e
7-5-8

After moving 3 into the empty space
1-2-4
6-e-3
7-5-8

After moving 6 into the empty space
1-2-4
e-6-3
7-5-8
```

```
After moving 2 into the empty space
e-2-3
1-4-6
7-5-8

After moving 1 into the empty space
1-2-3
e-4-6
7-5-8

After moving 4 into the empty space
1-2-3
4-e-6
7-5-8

After moving 5 into the empty space
1-2-3
4-5-6
7-e-8

After moving 8 into the empty space. Goal achieved!
1-2-3
4-5-6
7-8-e
```

Hints

```
/* Create a class that contains the methods to solve the 8-puzzle by importing the
following packages. */
from simpleai.search import astar, SearchProblem

/* Define a class that contains the methods to solve the 8-puzzle: */

/* Override the actions method to align it with our problem: */
```

```
/* Check the location of the empty space and create the new action: */

/* Check if the goal has been reached.*/

/* Define the heuristics method and compute the distance.*/

/* Define a function to convert a list of string */

/* Define a function to convert a string to a list. */

/* Define a function to get the location of a given element in the grid. */


/* Define the initial state and the final goal we want to achieve: */

/* Track the goal positions for each piece by creating a variable: */

/* Create the A* solver object using the initial state we defined earlier and extract
the result: */

/* Print the solution */
```

Try: Extensively make use of state space search to represent and solve the 15 Puzzle problem. Represent the problem as a state space and define the rules.

Start state:

3	10	13	7
9	14	6	1
4		15	2
11	8	5	12

Goal state:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

2.4 Blocks Rearrangement Problem

The problem is to find a plan for rearranging a stack of blocks as shown below. We are allowed to move one block at a time. A block can be grasped only when its top is clear. A block can be put on the table or on some other blocks. To find a required plan, we must find a sequence of moves that accomplish the given transformation. Think the problem as a problem of exploring among possible alternatives.

Input:

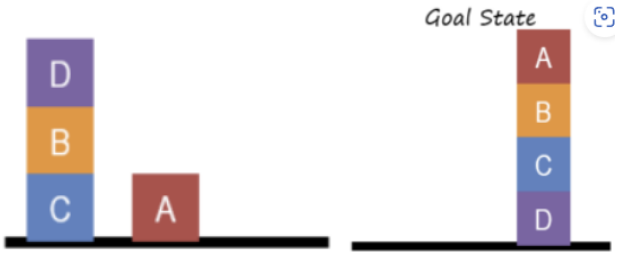
Output:

a. Generate the rules involving accomplishing various tasks involving the blocks world.
b. Formulate the more careful definitions for program and the actions.
c. Present the graphical representation of the problem (state space representations) including initial state and the goal state.

Hints

```
/* Define a class to initialize all the starting states of all the blocks. */

/* Do the same for the goal state also.*/

/* Check whether the current state matches the goal state. */

/* Generate all possible moves from the current state. */

/* Allow any block can be moved to the table or on top of another block. */

/* If a block is not blocked by another, it can be moved. Move onto another block.*/

/* Apply a move to the current state and return the new state.

/* Solve the problem using BFS by checking if the goal state is reached by getting all
the possible moves . */

/* Return the solution if found otherwise return None. */
```

Try: The problem is to find a plan for rearranging a stack of blocks as shown below. We are allowed to move one block at a time. A block can be grasped only when its top is clear. A block can be put on the table or on some other blocks. To find a required plan, we must find a sequence of moves that accomplish the given transformation. Think of the problem as a problem of exploring among possible alternatives.



The goal here is to move Block B from the middle of the pile on the left and onto the top of the pile on the right. Hence this sequence of moves would be an acceptable solution:

[("C", "Table"), ("B", "E"), ("C", "A")]

3. Exercises on Heuristic Search Techniques

3.1 Constructing a String using Greedy Search

Recreate the input string based on the alphabets using a greedy search. Ask the algorithm to search the solution space and construct a path to the solution.

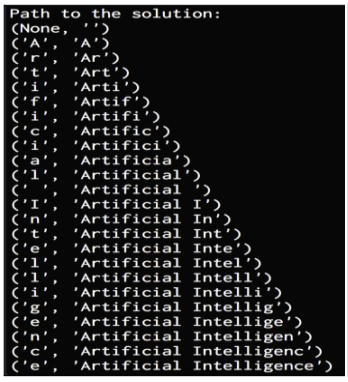
Input:

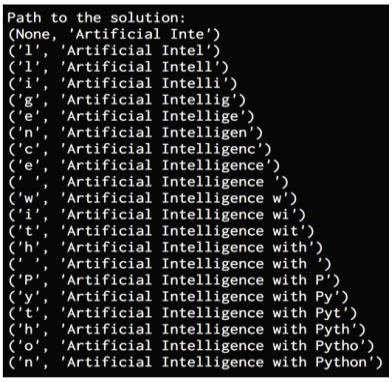
1. A function to parse the input arguments.
2. A class (SearchProblem) that contains the methods needed to solve the problem.
3. Check the current state and take the right action.
4. Concatenate state and action to get the result.
5. Check if the goal has been achieved.

6. The heuristic that will be used.
7. Initialize the CustomerProblem object.
8. Set the starting point and the goal we want to achieve.
9. Run the solver and
10. Print the path to the solution.

Output: Calculate how far we are from the goal and use that as the heuristic to guide it towards the goal.

1. Run the code with an empty initial state.
2. Run the code with a non-empty starting point.





Hints

```
/*Create a new Python file and import the following packages: */
import argparse
import simpleai.search as ss

/* Define a function to parse the input arguments */

class CustomProblem(ss.SearchProblem):
        def set_target(self, target_string):
                self.target_string = target_string

# Check the current state and take the right action
def actions(self, cur_state):
        if len(cur_state) < len(self.target_string):
                alphabets = 'abcdefghijklmnopqrstuvwxyz'
                return list(alphabets + ' ' + alphabets.upper())
        else:
                return []

# Concatenate state and action to get the result
def result(self, cur_state, action):
        return cur_state + action

# Check if goal has been achieved
def is_goal(self, cur_state):
        return cur_state == self.target_string

/* Initialize the CustomProblem object */

/* Set the starting point as well as the goal we want to achieve */

Run the solver:
# Solve the problem
output = ss.greedy(problem)
```

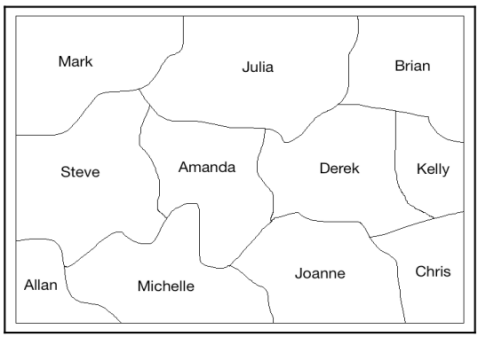
```
/* Print the path to the solution */
```

Try: 1. Solve the same problem with constraints. Specify three constraints as follows: John, Anna, and Tom should have different values. Tom's value should be bigger than Anna's value If John's value is odd, then Patricia's value should be even and vice versa.

2. Implement the above code to your family and start asking the questions in a similar way.

3.2 Solving a Region Coloring Problem

Consider the following screenshot:



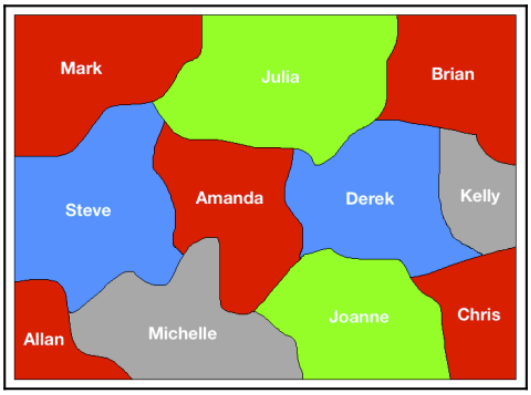
We have a few regions in the preceding figure that are labeled with names. Our goal is to color with four colors so that no adjacent regions have the same color. Make use of Constraint Satisfaction framework to solve the region-coloring problem.

Input:

1. Constraints that specify different values
2. Main function and a list of names.
3. List of possible colors

Output:

```
Color mapping:

Derek ==> blue
Michelle ==> gray
Allan ==> red
Steve ==> blue
Julia ==> green
Amanda ==> red
Joanne ==> green
Mark ==> red
Kelly ==> gray
Brian ==> red
```



Hints

```
/* Create a new Python file and import the following packages:*/
from simpleai.search import CspProblem, backtrack

/* Define the constraint that specifies that the values should be different: */
/* # Define the function that imposes the constraint */
/* # that neighbors should be different */
```

```
def constraint_func(names, values):
        return values[0] != values[1]

/* Define the main function and specify the list of names: */

if __name__=='__main__':
# Specify the variables names = ('Mark', 'Julia', 'Steve', 'Amanda', 'Brian', 'Joanne',
'Derek', 'Allan', 'Michelle', 'Kelly')

/*Define the list of possible colors */

/* convert the map information into something that the algorithm can understand */

/* Use the variables and constraints to initialize the object */

/* Solve the problem and print the solution */
```

Try: Implement the above code to your region and start asking the questions in a similar way.

3.3 Building an 8 Puzzle Solver

8-puzzle is a variant of the 15-puzzle. You can check it out at https://en.wikipedia.org/wiki/15_puzzle. You will be presented with a randomized grid, and your goal is to get it back to the original ordered configuration. You can play the game to get familiar with it at http://mypuzzle.org/sliding. The goal is to use A* algorithm to solve this problem and find the paths to the solution in a graph.

Input:

1. A class that contains the methods.
2. Action method to get the list of the possible numbers.
3. Check the location of the empty space and create a new action.

Output:

1. Return the resulting state after moving a piece to the empty space.
2. Computes the distance between the current state and goal state using Manhattan distance.

```
Initial configuration
1-e-2
6-3-4
7-5-8

After moving 2 into the empty space
1-2-e
6-3-4
7-5-8

After moving 4 into the empty space
1-2-4
6-3-e
7-5-8

After moving 3 into the empty space
1-2-4
6-e-3
7-5-8

After moving 6 into the empty space
1-2-4
e-6-3
7-5-8
```

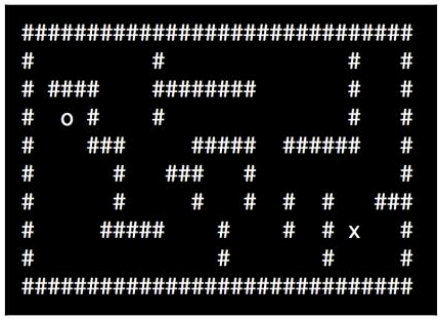
```
After moving 2 into the empty space
e-2-3
1-4-6
7-5-8

After moving 1 into the empty space
1-2-3
e-4-6
7-5-8

After moving 4 into the empty space
1-2-3
4-e-6
7-5-8

After moving 5 into the empty space
1-2-3
4-5-6
7-e-8

After moving 8 into the empty space. Goal achieved!
1-2-3
4-5-6
7-8-e
```

Hints

```
/* Create a new Python file and import the following packages */
from simpleai.search import astar, SearchProblem
```

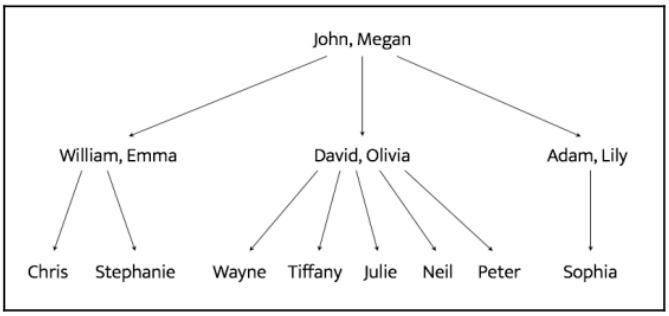
```
/* Define a class that contains the methods to solve the 8-puzzle: */
# Class containing methods to solve the puzzle class PuzzleSolver(SearchProblem):

/* Override the actions method to align it with our problem: */
# Action method to get the list of the possible
# numbers that can be moved in to the empty space
def actions(self, cur_state):
        rows = string_to_list(cur_state)
        row_empty, col_empty = get_location(rows, 'e')

/* Check the location of the empty space and create the new action */

/* Override the result method. Convert the string to a list and extract the location
of the empty space. */

/* # Return the resulting state after moving a piece to the empty space */

/* Check if the goal has been reached */

/* Define the heuristic method. */

/* Compute the distance */

/* Define a function to convert a list to string */

/* Define a function to convert a string to a list */

/* Define a function to get the location of a given element in the grid */

/* Define the initial state and the final goal we want to achieve */

/* Track the goal positions for each piece by creating a variable */

/* Create the A* solver object using the initial state we defined earlier and extract
the result */

/* Print the solution */
```

Try: Use the A* Algorithm to solve a maze. Consider the below figure to build a maze solve.



3.4 Parsing a Family Tree

Use the most familiar Logic programming and solve an interesting problem of Parsing a Family Tree by considering the following diagram. John and Megan have three sons – William, David, and Adam. The wives of William, David, and Adam are Emma, Olivia, and Lily respectively. William and Emma have two children – Chris and Stephanie. David and Olivia have five children – Wayne, Tiffany, Julie, Neil, and Peter. Adam and Lily have one child – Sophia. Based on these facts, create a program that can tell us the name of Wayne's grandfather or Sophia's uncles are.



Input: A .json file specified with the relationship among all the people involved.

Output: Ask the following questions to understand if our solver can come up with the right answer.

1. Who John's children are?
2. Who is William's mother?
3. Who are Adam's parents?
4. Who are Wayne's grandparents?

Hints

```
/*Create a new Python file and import the following packages: */
import json
from logpy import Relation, facts, run, conde, var, eq

/* Define a function to check if x is the parent of y. We will use the logic that if
x is the parent of y, then x is either the father or the mother. We have already
defined “father” and “mother” in our fact base: */

/* Define a function to check if x is the grandparent of y. We will use the logic that
if x is the grandparent of y, then the offspring of x will be the parent of y: */

# Check for sibling relationship between 'a' and 'b'
def sibling(x, y):
        temp = var()
        return conde((parent(temp, x), parent(temp, y)))

/* Define the main function and initialize the relations father and mother */
/* Load the data from the relationships.json file */
/* Read the data and add them to our fact base */
Define the variable x:
x = var()

/* We are now ready to ask some questions and see if our solver can come up with the
right answers. Let's ask who John's children are:

# John's children
name = 'John'
output = run(0, x, father(name, x))
```

```
print("\nList of " + name + "'s children:")
for item in output:
        print(item)

/* Start asking the questions like: Who is William's mother?, Who are Adam's parents?, Who are Wayne's grandparents?, Who are Megan's grandchildren?, Who are David's siblings?, Who are Tiffany's uncles? */
/* List out all the spouses in the family */
```

Try:

1. Who are Megan's grandchildren?
2. Who are David's siblings?
3. Who are Tiffany's uncles?
4. List out all the spouses in the family.

3.5 Analyzing Geography

Use logic programming to build a solver to analyze geography. In this problem, specify information about the location of various states in the US and then query our program to answer various questions based on those facts and rules. The following is a map of the US:

Input:

1. Define the input files to load the data from.
2. Read the files containing the coastal states.
3. Add the adjacency information to the fact base.
4. Initialize the variables x and y.

Output:

1. Print out all the states that are adjacent to Oregon.
2. List all the coastal states that are adjacent to Mississippi.
3. List all the coastal states that are adjacent to Mississippi.

```
Is Nevada adjacent to Louisiana?:
No

List of states adjacent to Oregon:
Washington
California
Nevada
Idaho

List of coastal states adjacent to Mississippi:
Alabama
Louisiana

List of 7 states that border a coastal state:
Georgia
Pennsylvania
Massachusetts
Wisconsin
Maine
Oregon
Ohio
```

Hints

```
/*Create a new Python file and import the following: */

from logpy import run, fact, eq, Relation, var

/*Initialize the relations: */

adjacent = Relation() coastal = Relation()

/*Define the input files to load the data from: */

file_coastal = 'coastal_states.txt'
file_adjacent = 'adjacent_states.txt'

/*Load the data: # Read the file containing the coastal states */
with open(file_coastal, 'r') as f:
        line = f.read()
        coastal_states = line.split(',')

/*Add the information to the fact base:
# Add the info to the fact base
for state in coastal_states:
        fact(coastal, state)

/* Read the adjacency data */

/* Add the adjacency information to the fact base */

/* Initialize the variables x and y */

/* Check if Nevada is adjacent to Louisiana */

/* Print out all the states that are adjacent to Oregon, List all the coastal states
that are adjacent to Mississippi, List seven states that border a coastal state, List
states that are adjacent to both Arkansas and Kentucky */
```

Try: Add more questions like "list states that are adjacent to both Arkansas and Kentucky" to the program to see if it can answer them.

3.6 Building a Puzzle Solver

The interesting application of logic programming is in solving puzzles. The goal of this exercise is to specify the conditions of a puzzle, and the program has to come up with a solution. Also specify various bits and pieces of information about four people and ask for the missing piece of information.

Input: In the logic program, we specify the puzzle as follows:

- Steve has a blue car.
- The person who owns the cat lives in Canada Matthew lives in USA.
- The person with the black car lives in Australia.
- Jack has a cat.
- Alfred lives in Australia.
- The person who has a dog life in France.
- Who has a rabbit?

Output: The goal is to find the person who has a rabbit. Here are the full details about the four people:

Name	Pet	Car color	Country
Steve	dog	blue	France
Jack	cat	green	Canada
Matthew	rabbit	yellow	USA
Alfred	parrot	black	Australia

Hints

```
/*Create a new Python file and import the following packages: */
from logpy import *
from logpy.core
import lall

/* Declare the variable people: */
# Declare the variable
people = var()

/* Define all the rules using lall, The first rule is that there are four people, The
person named Steve has a blue car */

/* The person who has a cat lives in Canada, The person named Matthew lives in USA,
The person who has a black car lives in Australia, The person named Jack has a cat,
The person named Alfred lives in Australia, The person who has a dog lives in France,
The person who has a dog lives in France.*/

/* Run the solver with the preceding constraints */

/* Extract the output from the solution */

/* Print the full matrix obtained from the solver */
```

Try: Demonstrate how to solve a puzzle with incomplete information. You can play around with it and see how you can build puzzle solvers for various scenarios.

4. Exercises on Heuristic Search Techniques

4.1 Best First Search Algorithms

The Best First Search algorithm is a set of rules that work together to perform a search. It considers the various characteristics of a prioritized queue and heuristic search. The goal of this algorithm is to reach the state of final or goal in the shortest possible time.

Input: Best First Search Algorithm

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list.
3. Repeat the next steps until the GOAL node is reached.
   - If the OPEN list is empty, then EXIT the loop returning 'False'.
   - Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node.
   - If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path.
   - If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list.
   - Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

Output:

a. Perform the search process by using additional information to determine the next step towards finding the solution.

b. Perform the search process using an evaluation function to decide which among the various available nodes is the most promising before traversing to that node.

c. Apply priority queues and heuristic search functions to track the traversal.

Hints

```
/* Import heapq. */

/* Initialize the graph and heuristic function, dictionary representing the graph and
heuristic values. */

/* Perform the best-first search by defining the priority queue to store nodes with
their heuristic values. */

/* Get the node with the lowest heuristic value. */

/* If the goal is reached, return the path and total cost. */

/* Mark the current node as visited, explore all the neighbors. */

/* Return the solution if exists otherwise return None. */
```

Try: Make use of Heuristic Search principle and apply the best first search to solve the 8-puzzle problem.

4.2 A* Algorithm

A* Algorithm – A square grid is composed of many obstacles that are scattered randomly. The goal is to find the final cell of the grid in the shortest possible time. Implement A* algorithm to search for the shortest path among the given initial and the final state.

Input: Define the graph as nodes with neighboring nodes and cost. Define the heuristic values of each node.

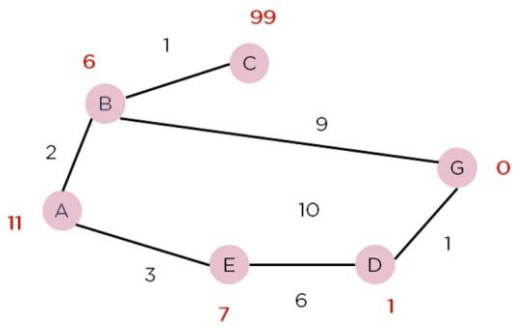
Output:

a. Initially represent the problem statement as a graph traversal problem.
b. Perform the search process to obtain the shorter path first, thus making it optimal.
c. Find the least cost outcome for the problem by finding all the possible outcomes.
d. Make use of weighted graph by using numbers to represent the cost of taking each path and find the best route with the least cost in terms of distance and time.

Hints

```
/* Implements the A* algorithm to find the shortest path in a weighted graph.*/

/* Represent the dictionary for a graph and heuristic values. */

/* Define the starting and goal node keeping in mind to return the tuple containing the optimal path and the total cost. */

/* Pop the node with the lowest f_cost, if the goal is reached, return the path and cost. */

/* Mark the current node as visited */

/* Explore the neighbors to accumulate the path cost by getting the heuristic values, total path cost. */

/* If the path is found then return, otherwise return None */
```

Try: Implement the A* algorithm and calculate the shortest distance between the initial and the goal states by considering the following weighted graph:



4.3 AO* Algorithm

Implement the algorithm to generate AND-OR graph or tree to represent the solution by dividing the problem into sub problems and solve them separately to obtain the result by combining all the sub solutions.

Input: The AO* algorithm works on the formula given below : f(n) = g(n) + h(n) where,

- g(n): The actual cost of traversal from initial state to the current state.
- h(n): The estimated cost of traversal from the current state to the goal state.
- f(n): The actual cost of traversal from the initial state to the goal state.

Output:

a. Follow problem decomposition approach and solve each sub problem separately and later combine all the solutions.

b. Traverse the graph starting at the initial node and following the current best path and accumulate the set of nodes that are on the path and have not yet been expanded.

c. Pick one of these best unexpanded nodes and expand it. Add its successors to the graph and compute cost of the remaining distance for each of them.

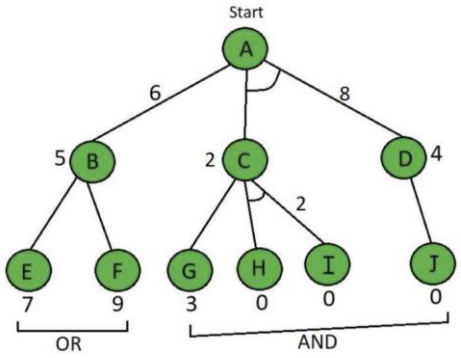
d. Change the cost estimate of the newly expanded node to reflect the new information produced by its successors. Propagate this change backward through the graph. Decide which of the current best path

Optimal Solution Path: A -> B -> D Total Cost: 5

Hints

```
/* Define a class to represent a node in the AND-OR graph. */

/* Define a class to implement AO* algorithm to find the optimal solution path. */

/* Mention the initial state, goal state, the optimal solution path and its total cost.
*/

/* Pick the best node to expand (lowest f_cost) and expand the node. */

/* Propagate changes backward to reflect updated costs, Add successors to the open
list if not already processed. */

/* Generate the optimal path */

/* Expand the given node by calculating f_cost for its successors. */

/* Propagate cost changes backward through the graph.*/

/* Generate the optimal solution path by following the best successors. */

/* Define heuristic values for the nodes, Create nodes for the graph, Define the AND-
OR graph as node connections, Instantiate the AO* algorithm, Perform the AO* search.
*/

/* Generate the result as optimal solution path and total cost.*/
```

Try: Implement the algorithm to generate AND-OR graph or tree to represent the solution by dividing the problem into sub problems and solve them separately to obtain the result by combining all the sub solutions.



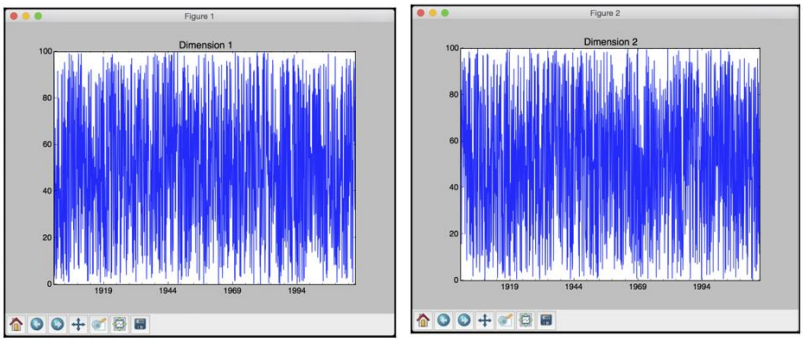
5. Exercises on Probabilistic Reasoning for Sequential Data

5.1. Handling Time-Series Data with Pandas

Time-series data analysis is used extensively in financial analysis, sensor data analysis, speech recognition, economics, weather forecasting, manufacturing, and many more. Explore a variety of scenarios where we encounter time-series data and see how a solution can be built. Create a python file and learn how to handle time-series data in Pandas.

Input: Time-series dataset

Output: Time-Series data with different dimensions like:



Hints

```
/* Create a python file and import the following packages */

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

/* Define a function to read the data from the input file. */

/* Define a lambda function to convert strings to Pandas date format: */

/* Use this lambda function to get the start date from the first line in the input file: */

/* Create a list of indices with dates using the start and end dates with a monthly frequency: */
```

```
/* Create pandas data series using the timestamps: */


/* Define the main function and specify the input file:, Specify the columns that contain the data:*/


/* Iterate through the columns and read the data in each column and Plot the time-series data. */
```

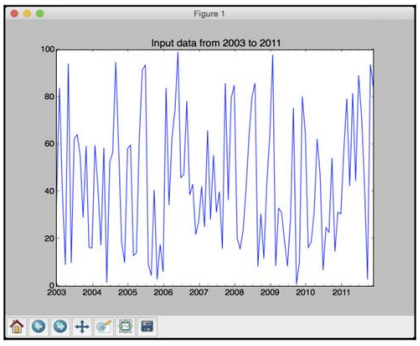
Try: Revise the above code to analyze and visualize the time-series data for three and more dimensions.

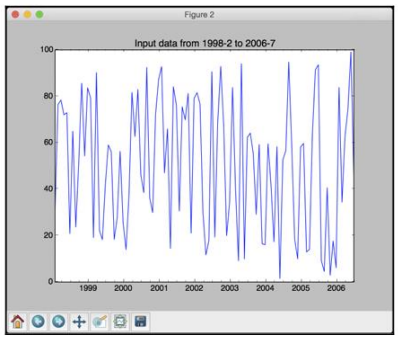
5.2. Slicing Time-Series Data

The process of slicing refers to dividing the data into various sub-intervals and extracting relevant information. This is very useful when you are working with time-series datasets. Instead of using indices, we will use timestamp to slice our data. Develop a python code to analyze the time-series data and visualize the same at different intervals.

Input: Time-series dataset

Output: Visualize the Time-Series data with different levels of granularity.





Hints

```
/* Create a python file and import the following packages */

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

/* Define a function to read the data from the input file. */

/* Define a lambda function to convert strings to Pandas date format: */

/* Use this lambda function to get the start date from the first line in the input file: */

/* Create a list of indices with dates using the start and end dates with a monthly frequency: */
```

```
/* Create pandas data series using the timestamps: */


/* Define the main function and specify the input file:, Specify the columns that contain the data:*/


/* Iterate through the columns and read the data in each column and Plot the time-series data. */
```

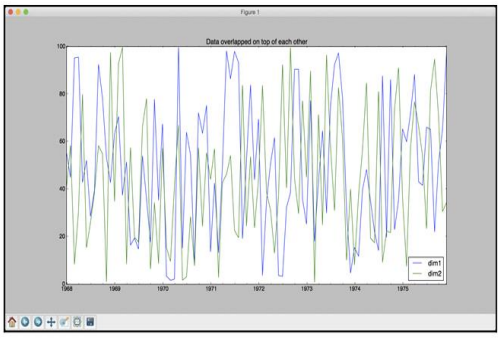
Try: Revise the above code to analyze and visualize the time-series data for different level of granularities.

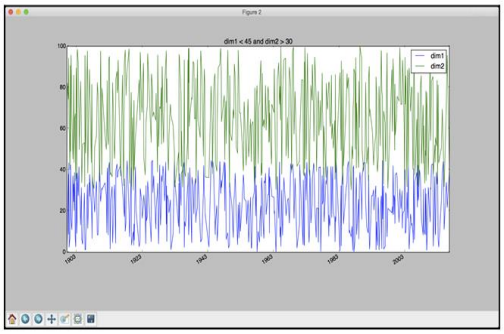
5.3. Operating on Time-Series Data

Pandas allow us to operate on time-series data efficiently and perform various operations like filtering and addition. You can simply set some conditions and Pandas will filter the dataset and return the right subset. Develop a python code that can allow to build various similar applications without having to reinvent the wheel.

Input: Time-series dataset

Output: Visualize the Time-Series data with different data frames.





Hints

```
/* Create a python file and import the following packages */

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from timeseries import read_data

/* Define the input filename and load the third and fourth columns into separate variables. */

/* Create a Pandas dataframe object by naming the two dimensions and plot the data by specifying the start and end years */

/* Filter the data using conditions and then display it. */
```

```
/* Display the summarized results also */
```

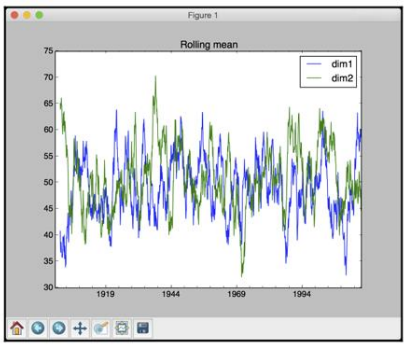
Try: Add two series in Pandas and different dimensions between the given start and end dates.

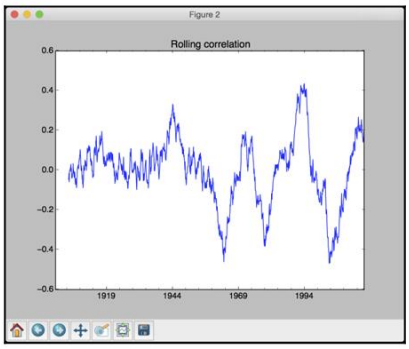
5.4. Extracting Statistics from Time-Series Data

To extract meaningful insights from time-series data, extract statistics from it. These stats can be things like mean, variance, correlation, maximum value, and so on. These stats must be computed on a rolling basis using a window. Use a predetermined window size and keep computing these stats. Develop a python code to extract these statistics from time-series data and visualize them. Produce interesting patterns while visualizing the statistics over time.

Input: Time-series dataset

Output: Visualize the Time-Series data showing the rolling mean, rolling correlation and also include the terminal output.






```
Maximum values for each dimension:
dim1    99.98
dim2    99.97
dtype: float64

Minimum values for each dimension:
dim1    0.18
dim2    0.16
dtype: float64

Overall mean:
dim1    49.030541
dim2    50.983291
dtype: float64
```

```
Row-wise mean:
1900-01-31    85.595
1900-02-28    75.310
1900-03-31    27.700
1900-04-30    44.675
1900-05-31    31.295
1900-06-30    44.160
1900-07-31    67.415
1900-08-31    56.160
1900-09-30    51.495
1900-10-31    61.260
1900-11-30    30.925
1900-12-31    30.785
Freq: M, dtype: float64

Correlation coefficients:
          dim1     dim2
dim1  1.00000  0.00627
dim2  0.00627  1.00000
```

Hints

```
/* Create a python file and import the following packages */

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from timeseries import read_data

/* Define the input filename and load the third and fourth columns into separate
variables. */

/* Create a Pandas dataframe object by naming the two dimensions and extract maximum
and minimum values along each dimension */

/* Extract the overall mean and the row-wise mean for the first 12 rows and plot the
rolling mean using a window size of 24. */
```

```
/* Plot the rolling mean using a window size of 24 */

/* Print the correlation coefficients and plot the rolling correlation using a window size of 60. */
```

Try: Add two series in Pandas and visualize the correlation coefficients not mentioned in the preceding figures.

5.5. Generating Data using Hidden Markov Models

A Hidden Markov Model (HMM) is a powerful analysis technique for analyzing sequential data. It assumes that the system being modeled is a Markov process with hidden states. This means that the underlying system can be one among a set of possible states. It goes through a sequence of state transitions, thereby producing a sequence of outputs. We can only observe the outputs but not the states. Hence these states are hidden from us. Develop the python code to model the data so that we can infer the state transitions of unknown data.

Input: Traveling Salesman data representing the information with a transition matrix.

P(London -> London) = 0.10

P(London -> Barcelona) = 0.70

P(London -> NY) = 0.20

P(Barcelona -> Barcelona) = 0.15

P(Barcelona -> London) = 0.75

P(Barcelona -> NY) = 0.10

P(NY -> NY) = 0.05

P(NY -> London) = 0.60

P(NY -> Barcelona) = 0.35

Let's represent this information with a transition matrix:

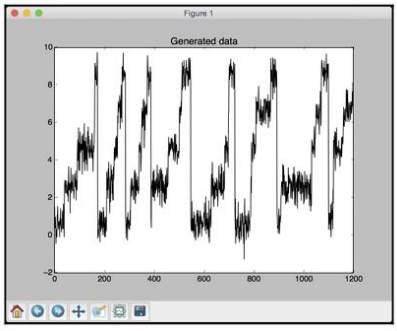
London Barcelona NY

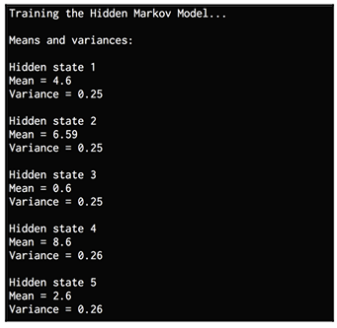
London 0.10 0.70 0.20

Barcelona 0.75 0.15 0.10

NY 0.60 0.35 0.05

Output: Generate and Visualize samples using the trained HMM model like:





Hints

```
/* Create a python file and import the following packages */

import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd
from timeseries import read_data

/* Define the input filename and extract the third column for training. */

/* Create a Gaussian HMM with 5 components and diagonal covariance and Train the HMM.
*/

/* Print the mean and variance values for each component of the HMM. */
```

Try: 1. Identifying Alphabet Sequence with Conditional Random Fields

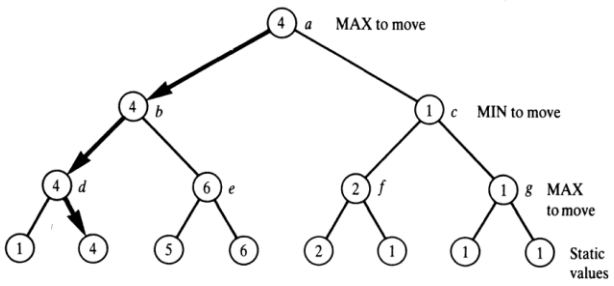
2. Analyze the stock market data using hidden Markov model.

6. Exercises on Building Games with AI

6.1. Minimax Algorithm

As searching game trees exhaustively is not feasible for interesting games, other methods that rely on searching only part of the game tree have been developed. Among these, a standard technique used in computer game playing (chess) is based on the minimax principle. The goal of this exercise is to implement the minimax principle and identify the changes that a player has to with a game.

Input: Game Tree and a Search Tree like:



Output: Make use of static and backup values and estimate the best position from a list of candidate positions.

Hints

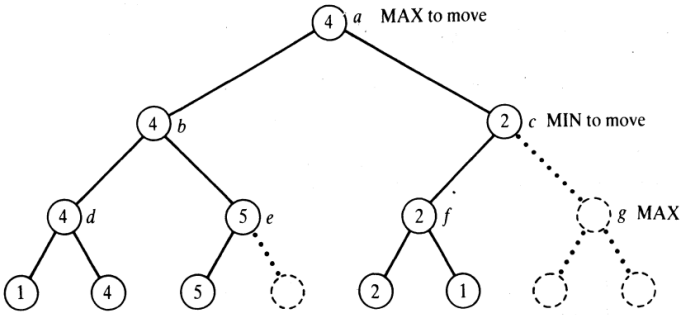
```
/* Calculate the heuristic estimator using the estimation function and estimate the
changes that a player must win. */

/* Implement the procedure as: Minimax procedure: minimax( Pos, BestSucc, Val)  Pos is
a position, Val is its minimax value; best move Vo from Pos leads to position BestSuc
*/
```

Try: Create a knowledge base including the clauses that help in implementing the straightforward method of minimax principle.

6.2. Alpha Beta Pruning

Create a knowledge base representing the straightforward procedure to implement alpha-beta algorithm. Develop a prolog program that systematically visits all the positions in the search tree, up to its terminal positions in a depth-first fashion, and statically evaluates all the terminal positions of this tree.

Input: Search Tree, starting point, legal moves, maximum successors of each move, and apply backtracking wherever applicable.



Output: Compute the exact value of a root position P by setting the bounds as follows:

$$V(P, -infinity, +infinity) = V(P)$$

Hints

```
alphabeta(/* Pass the 4 arguments like position, alpha beta values, good position and
the current value */) :-
        moves(Pos, PostList), !,
        boundedbest(Postlist, Alpha, Beta, GoodPos, Val);
        staticval(Pos, Val).

boundedbest( [Pos | Poslist], Alpha, Beta, GoodPos, GoodVal) :-
        alphabeta( Pos, Alpha, Beta, _, Val),
        goodenough( Poslist, Alpha, Beta, Pos, Val, GoodPos, GoodVal).

goodenough( /* Mention an empty list */, -, -, Pos, Val, Pos, Val) '- !. % No other
candidate

goodenough( -, Alpha, Beta, Pos, Val, Pos, Val) :-
        min-to-rnove( Pos), Val ) Beta, !; % Maximizer attained upper bound
        marto-mov{ Pos), Val ( Alpha, !. % Minimizer attained lower bound

goodenough( Poslist, Alpha, Beta, Pos, Val, GoodPos, GoodVal) :-
        newbounds( /* Pass the 6 arguments like alpha beta values, good position,
current value, new alpha and beta values */), % Refine bounds boundedbes( Poslist,
NewAlpha, NewBeta, Posl, Vall),
        betterof( Pos, Val, Posl, Vall, GoodPos, GoodVal).

newbounds( Alpha, Beta, Pos, Val, Val, Beta) :-
        min-to-rnove( Pos), Vd > Alpha, !. % Maximizer increased lower bound

newbounds( Alpha, Beta, Pos, Val, Alpha, Vat) :-
        marto-rnove( Pos), Val < Beta, !. % Minimizer decreased upper bound

newbounds( Alpha, Beta, -, -, Alpha, Beta)

betterof( Pos, Val, Posl, Val1, Pos, Val) :-
```

```
        min-to-rnove( Pos), Vd > Vall, !;
        marto:nove( Pos), Val < Vall, !.

betterof( -, -, Pos1, Val1, Posl, Vall).
```

Try: Consider a two-person game (for example, some non-trivial version of tic-tac-toe). Write game-definition relations (legal moves and terminal game positions) and propose a static evaluation function to be used for playing the game with the alpha-beta procedure. Use the principle of alpha beta to reduce the search in the tree mentioned above.

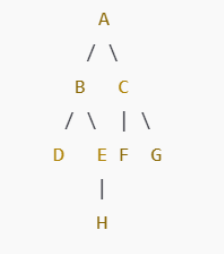
6.3. Iterative Deepening Techniques

The goal is to prove that search is ubiquitous in artificial intelligence. The performance of most AI systems is dominated by the complexity of a search algorithm in their inner loops. Prove with an example that this algorithm gives optimal solution for exponential tree searches.

Input: Starting node and goal node.

Output:

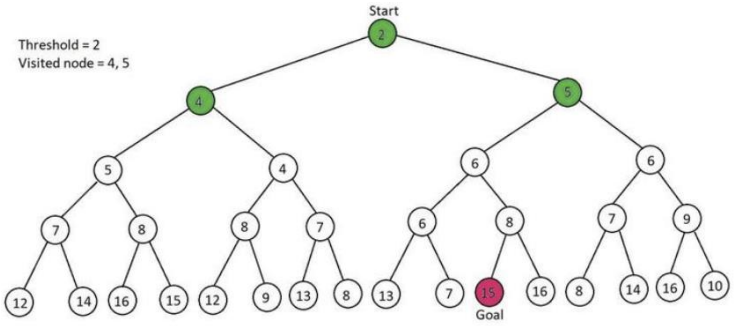
a. Complete the search process if the branching factor is finite and there is a solution at some finite depth and obtain optimal in finding the shortest solution first.
b. Avoid exploring each non-solution branch of the tree, omit cycle detection and retain completeness.
c. Use additional logical features of Prolog to terminate the search process whenever if there are no solutions identified even after backtracking.
d. Document the steps if the search process does not obtain optimal solution even after backtracking.



Hints

```
/* Define a class to initialize the graph, start node, and goal node. */

/* Perform the DLS to a specified depth with current node, current depth limit, set of
visited nodes and return the tuple with a path cost and the path. */

/* Perform the iterative deepening DFS with maximum depth limit to explore.*/

/* Generate the optimal path if found otherwise generate None. */

/* Apply backtracking if the solution is not found. */
```

Try: The 15-puzzle problem is a classic example of a **sliding puzzle game**. It consists of a 4×4 grid of numbered tiles with one tile missing. The aim is to rearrange the tiles to form a specific goal configuration. The state space of the puzzle can be represented as a tree where each node represents a configuration of the puzzle, and each edge represents a legal move. IDA* can be used to find the **shortest sequence of moves** to reach the goal state from the initial state.



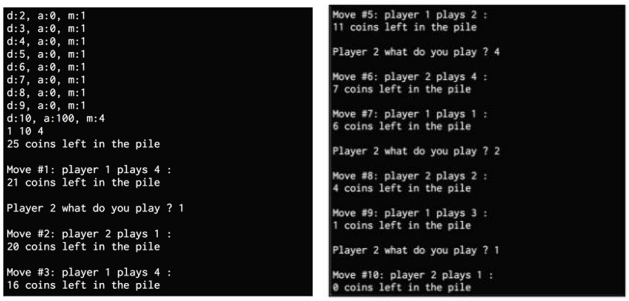
7. Exercises on Building Games with AI

7.1 Building a Bot to Play Last Coin Standing

This is a game where we have a pile of coins, and each player takes turns to take a number of coins from the pile. There is a lower and an upper bound on the number of coins that can be taken from the pile. The goal of the game is to avoid taking the last coin in the pile. This recipe is a variant of the Game of Bones recipe given in the easyAI library. Develop the python code to build a game where the computer can play against the user.

Input: Libraries like TwoPlayerGame, id_solve, Human_Player, AI_Player, and TT

Output: Force the computer to take the last coin, so that you win the game.



Hints

```
% Create a new Python file and import the following packages:
from easyAI import TwoPlayersGame, id_solve, Human_Player, AI_Player from easyAI.AI import TT
% Create a class to handle all the operations of the game.

/* Define who is going to start the game */
```

```
/* Define the maximum number of coins that can be taken out in any move. */

/* Define all the possible moves, define a method to remove the coins and keep track
of the number of coins remaining in the pile.  */

/* Check if somebody won the game by checking the number of coins remaining. */

/* Stop the game after somebody wins it and compute the score based on the win method.
*/

/* Define a method to show the current status of the pile. */

/* Define the main function and start by defining the transposition table. */

/* Define the method ttenttry to get the number of coins */
```

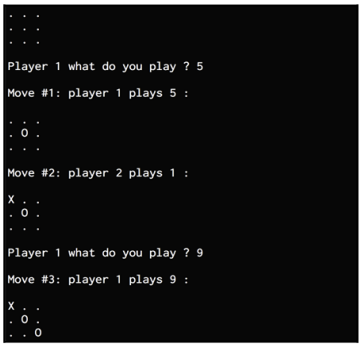
Try: Implement the same code so that you can win the game instead of a computer.

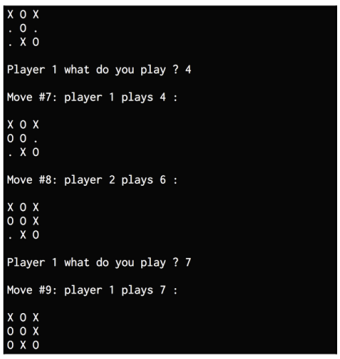
7.2 Building two Bots to Play Tic-Tac-Toe

Tic-Tac-Toe (Noughts and Crosses) is probably one of the most famous games. Let's see how to build a game where the computer can play against the user. This is a minor variant of the Tic-Tac-Toe recipe given in the easyAI library.

Input: Libraries like TwoPlayerGame, id_solve, Human_Player, and AI_Player.

Output: Force the computer to take the last coin, so that you win the game.





Hints

```
% Create a new Python file and import the following packages:
from easyAI import TwoPlayersGame, id_solve, Human_Player, AI_Player from easyAI.AI
import TT
% Create a class to handle all the methods to play the game.

/* Define a method to compute all the possible moves by defining a 3 x 3 board. */

/* Define the method to update the board after making a move. */
```

```
/* Define a method to see if somebody has lost the game and check if the game is over
using the loss_condition method.  */

/* Define a method to show the current progress and compute the score using the
loss_condition method. */

/* Define the main function and start by defining the algorithm and start the game. */
```

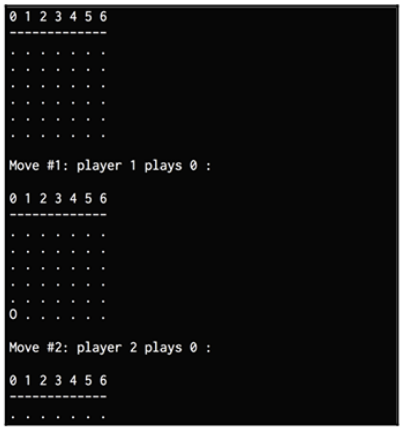
Try: Implement the same code so that you can win the game instead of a computer.

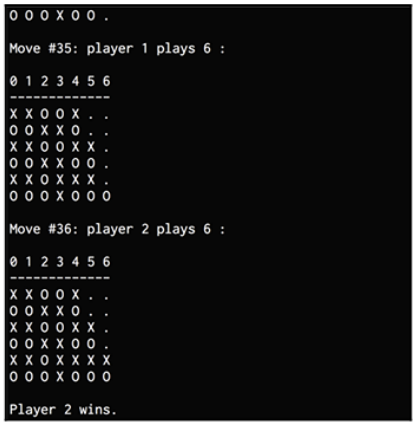
7.3 Building two Bots to Play Connect Four against each other

Connect Four™ is a popular two-player game sold under the Milton Bradley trademark. It is also known by other names such as Four in a Row or Four Up. In this game, the players take turns dropping discs into a vertical grid consisting of six rows and seven columns. The goal is to get four discs in a line. This is a variant of the Connect Four recipe given in the easyAI library. Develop the python code to build it instead of playing against the computer, we will create two bots that will play against each other.

Input: Libraries like TwoPlayerGame, id_solve, Human_Player, and AI_Player.

Output: Algorithm for each bot and see which one wins.





Hints

```
% Create a new Python file and import the following packages:
from easyAI import TwoPlayersGame, id_solve, Human_Player, AI_Player, Negamax, and SSS
from easyAI.AI import TT
% Create a class that contains all the methods needed to play the game.

/* Define the board with six rows and seven columns. */

/* Define who's going to start the game while defining the board positions. */

/* Define a method to get all the possible moves.  */

/* Define a method to control how to make a move and to show the current status. */
```

```
/* Define a method to compute what a loss looks like and check whether the game is
over or not. */


/* Compute the score to decide the winning bot */
```

Try: Implement the same code so that SSS algorithm can win the game instead of a Negamax algorithm.

8. Exercises on Fuzzy Logic

8.1 Fuzzy Inference Systems

Fuzzy logic involves understanding the principles of fuzzy logic and implementing them using several python libraries that deal with a form of logic and appropriate reasoning by allowing the representation of uncertainty and imprecision. The product has a price and a benefit, so we can create some rules. For example, if the cost of the product is low and the benefit is high, then cost benefit is high. Another rule if the cost of the product is high and the benefit is also high, then the cost benefit is median and not the rule.

If the cost of the product is low and the benefit is also low, then the cost benefit is also made in the next row. If the cost is high and the benefit is low, then the cost benefit is low. And the last one, if cost is high and benefit is low, then the cost benefit is low.

There are three results a high-cost benefits, medium cost benefits and low-cost benefits. While the variables responsible for the forecasts are costs and benefits and they have two values low and high. When there are only two values we can consider, it is a binary problem. The low category could be represented by number zero, while the high category could be represented by number one. The idea of fuzzy logic is to represent the reasoning process in a way more like the way humans think, for example. For some people, the cost may be low, but for others it may be a little lower, as if there was a level before this category here. For other people, the costs might be a little higher, not that low. Develop python code to understand how fuzzy logic can be used to various problem domains effectively.

Input: Represent the reasoning process in a way more likely the way humans think.

Output: Evaluate the efficiency of the fuzzy logic system with sample inputs.

Hints

```
% Create a new Python file and import the following packages:
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl


/* Define the fuzzy sets for the input variables (cost and benefit) */


/* Define the fuzzy sets for the output variables (cost and benefit)  */


/* Define the fuzzy sets based on the provided conditions.  */


/* Implement the fuzzy inference system using the chosen python libraries like scikit-
fuzzy. */
```

```
/* Test the fuzzy logic system with sample inputs. */
```

Try: Implement the code to understand the linguistic variables and membership functions for effectively modeling and solving problems using fuzzy logic.

8.2 Fuzzy Inference Systems

Fuzzy inference involves applying fuzzy logic rules to input data to generate meaningful output. Several steps for fuzzy inference include converting crisp (exact) input data into fuzzy sets using membership functions and determining the degree of membership of each input value in the appropriate fuzzy sets. Develop the python code to apply the fuzzy rules to implement aggregation, and defuzzification.

Input: Linguistic variables like 'cost' and 'benefit' from the previous example.

Output: Crisp output: 8.775.

Hints

```
% Create a new Python file and import the following packages:
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl


/* Define universe of discourse (range) for cost and benefit */

/* Define linguistic variables: cost, benefit, and cost benefit.  */

/* Define membership functions for cost and membership functions for benefits. */

/* Define membership functions for cost_benefit (output). */

/* Define fuzzy rules and create fuzzy control system. */

/* Provide input values, apply fuzzy rules, and obtain crisp output. */
```

Try: Develop the code to implement defuzzification to convert a fuzzy set into a crisp value.

8.3 Calculating the probability of Fan Speed

implemented by taking a basic example of calculating the probability of Fan-speed being low, medium or high based on current temperature and humidity as provided by user. Fuzzy logic allows you to model complex relationships, make decisions in ambiguous environments, and create intelligent systems that mimic human-like reasoning. Develop the python code while understanding the foundational logic behind fuzzy sets, where uncertainty is embraced, and imprecision becomes a strength.

Input: Understanding the key components like membership functions, inference system, defuzzification, and rule-based systems.

Output: Temperature, Humidity, and Speed.

Hints

```
% Create a new Python file and import the following packages:
```

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl


/* Define a method with temperature and humidity parameters and conduct inferencing.
*/

/* Define a method to understand the fan speed as low, medium, and high. Apply
defuzzification.  */
```

Try: Implement the code to understand the linguistic variables and membership functions for effectively modeling and solving problems using fuzzy logic.

9. Exercises on Expert Systems

9.1 Identify Animals

The goal is to create an expert system that can identify animals. We can use the rules of inference that we have learned about animals to perform this task. These rules serve as a starting point for developing an expert system, and they show the importance of having input from the users. The goal of an expert system is to provide useful information based on its users' inputs.

Input:

- If it has a tawny color and has dark spots, then the animal is a cheetah.
- If it has a tawny color and has black stripes, then the animal is a tiger.
- If it has a long neck and has long legs, then the animal is a giraffe.
- If it has black stripes, then the animal is a zebra.
- If it does not fly and has long neck, then the animal is an ostrich.
- If it does not fly, swims, black and white in color, then the animal is penguin.
- If it appears in story ancient mariner and flys well, then the animal is albatross.

Output:

a. Create an expert system that can identify the animal class using the inference rules.
b. Utilize the user inputs and predict the animal class based on the behaviors already learned by the expert system.

```
Welcome to the Animal Classifier Expert System!
Please answer the following questions with 'yes' or 'no':

Is the animal a vertebrate (has a backbone)? yes
Is the animal cold-blooded? yes
Does the animal have scales? yes

The animal class is: Fish
```

Hint:

```
/* Define a class to initialize an expert system with predefined inference rules. */

/* Define a class to predict the animal class based on user inputs. */

/* Define the features to ask the user and collect the user inputs. */
```

```
/* Create and use the expert systems. */

/* Display the results by predicting the animal class, other generate the results as
“unable to determine.” */
```

Try: Consider the if-then rules of figures and translate them into our rule notation. Propose extensions to the notation to handle certainty measures when needed.

if

1 the infection is primary bacteremia, and
2 the site of the culture is one of the sterilesites, and
3 the suspected portal of entry of the organism is the gastrointestinal tract

then

there is suggestive evidence (0.7) that the identity of the organism is bacteroides.

if

1 there is a hypothesis, H, that a plan P succeeds, and
2 there are two hypotheses,
H_1, that a plan R_1 refutes plan P, and
H_2, that a plan R_2 refutes plan P, and
3 there are facts: H_1 *is false*, and H_2 *is false*

then

1 generate the hypothesis, H_3, that the combined plan 'R_1 *or* R_2' refutes plan P, and
2 generate the fact: H_3 *implies not(H)*

9.2 Locating Failures in a Simple Electric Network

Create a knowledge base which can help locating failures in a simple electric network that consists of some electric devices and fuses. Such a network is shown in figure.

if

light1 is on *and*
light1 is not working *and*
fuse1 is proved intact

then

light1 is proved broken.

Another rule can be:

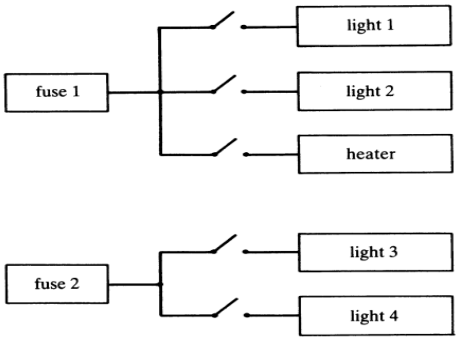
if

heater is working

then

fuse1 is proved intact.

Input:



Hint:

```
% A small knowledge base for locating faults in an electric network
% If a device is on and not working and its fuse is intact then the device is broken

broken_rule:
        if
                on(Device) and
                device(Device) and
                not working(Device) and
                connected(Device, Fuse) and
                proved(intact(Fuse))
        then
                proved(broken(Device))
% If a unit is working then its fuse is OK
fuse_ok_rule:
        if
                connected(Device,Fuse) and
                working(Device)
        then
                proved(intact(Fuse)).

% If two different devices are connected to a fuse and are both on and not working
% then the fuse has failed.
% NOTE: This assumes that at most one device is broken!

fused_rule:
        if
                connected(Device1, Fuse) and
                on(Device1) and
                not working(Device1) and
                samefuse(Device2, Device1) and
                on(Device2) and
```

```
                not working(Device2)
        then
                proved(failed(Fuse)).
same_fuse_rule:
        if
                connected(Device1, Fuse) and
                connected(device2, Fuse) and
                different(Device1, Device2)
        then
                samefuse(Device1, Device2).
fact: different(X, Y) :- not(X=Y).

fact: device(heater).
fact(device(light1).
fact(device(light2).
fact(device(light3).
fact(device(light4).
fact: connected(light1, fuse1).
fact: connected(light2, fuse1).
fact: connected(heater, fuse1).
fact: connected(light3, fuse2).
fact: connected(light4, fuse2).

askable(on(D), on('Device')).
askable(working(D), working('Device')).
```

Try: Think of some decision problem and try to formulate the corresponding knowledge in the form of if-then rules. You may consider choice of holiday, weather prediction, simple medical diagnosis, and treatment, etc.

10. Exercises on Expert Systems

10.1 Mental Health Disorder

Create an expert system in prolog to improve the understanding of declarative programming paradigm based on the logic rules. Develop the program with an idea of identifying the health disorder based on the database provide with mental health conditions.

Input: Set of logical rules with mental health conditions.

Output: Model the disorder of the patient health.

Hints

```
diagnose :-
 write('This is an expert system for dignosis of mental disorders.'), nl,
 write('There are several questions you need to answer for dignosis of mental
disorders.'), nl, nl,
 disorder(X),
```

```
 write('Condition was diagnosed as '),
 write(X),
 write('.').
diagnose :-
    write('The diagnose was not found.').


%The question predicate will have to determine from the user
%whether or not a given attribute-value pair is true
question(Attribute, Value):-
 retract(yes, Attribute, Value), !.
question(Attribute, Value):-
 retract(_, Attribute, Value),  !, fail.
question(Attribute, Value):-
 write('Is the '),
 write(Attribute),
 write(' - '),
 write(Value),
 write('?'),
 read(Y),
 asserta(retract(Y, Attribute, Value)),
 Y == yes.

%question with additional argument which contains
%a list of possible values for the attribute.
questionWithPossibilities(Attribute, Value, Possibilities) :-
 write('What is the patient`s '), write(Attribute), write('?'), nl,
 write(Possibilities), nl,
 read(X),
 check_val(X, Attribute, Value, Possibilities),
 asserta( retract(yes, Attribute, X) ),
 X == Value.

check_val(X, _, _, Possibilities) :-  member(X, Possibilities),
 !.
check_val(X, Attribute, Value, Possibilities) :-
 write(X), write(' is not a legal value, try again.'), nl,
 questionWithPossibilities(Attribute, Value, Possibilities).

%retract equips this system with a memory that remembers the facts that are already
%known because they were already entered by the user at some point during the
interaction.
:- dynamic(retract/3).

%. The program needs to be modified to specify which attributes are askable
food_amount(X) :- question(food_amount,X).
symptom(X) :- question(symptom,X).
mentality(X) :- question(mentality,X).
cause(X) :- question(cause, X).
indication(X) :- question(indication,X).
social_skill(X) :- question(social_skill,X).
condition(X) :- question(condition, X).
consequence(X) :- question(consquence,X).
specialty(X) :- question(specialty,X).
face_features(X) :- question(face_features,X).
ears_features(X) :- question(ears_features,X).
brain_function(X) :- question(brain_function,X).
perceptions(X) :- question(perceptions, X).
```

```
behavior(X) :- questionWithPossibilities(behavior, X, [repetitive_and_restricted,
narcissistic, aggresive]).


disorder(anorexia_nervosa) :- type(eating_disorder),
                              consequence(low_weight),
                              food_amount(food_restriction).

disorder(bulimia_nervosa) :- type(eating_disorder),
                             consequence(purging),
                     food_amount(binge_eating).

disorder(asperger_syndrome) :- type(neurodevelopmental_disorder),
                               specialty(psychiatry),
                               social_skill(low),
                               behavior(repetitive_and_restricted).

disorder(dyslexia) :- type(neurodevelopmental_disorder),
                      social_skill(normal),
                      perceptions(low),
                      symptom(trouble_reading).

disorder(autism) :- type(neurodevelopmental_disorder),
                    social_skill(low),
                    symptom(impaired_communication).


disorder(tourettes_syndrome) :- type(neurodevelopmental_disorder),
                                social_skill(normal),
                                specialty(neurology),
                                symptom(motor_tics).

disorder(bipolar_disorder) :- type(psychotic_disorder),
                              indication(elevated_moods).

disorder(schizophrenia) :- type(psychotic_disorder),
                           indication(hallucinations).

disorder(down_syndrome) :- type(genetic_disorder),
                           symptom(delayed_physical_growth),
                           face_features(long_and_narrow),
                           ears_features(large),
                           brain_function(intellectual_disability).

disorder(fragile_X_syndrome) :- type(genetic_disorder),
                                face_features(small_chin_and_slanted_eyes),
                                brain_function(intellectual_disability).


type(eating_disorder) :- symptom(abnormal_eating_habits),
                         mentality(strong_desire_to_be_thin).

type(neurodevelopmental_disorder) :-  condition(affected_nervous_system),
                                      brain_function(abnormal),
                                      cause(genetic_and_enviromental).

type(psychotic_disorder) :- symptom(false_beliefs),
                            mentality(manic_depressive),
```

```
                                cause(genetic_and_enviromental).

type(genetic_disorder) :- cause(abnormalities_in_genome).
```

Try: Create an Expert System suggesting the medical support for the diagnosis of kidney diseases. Include the features like:

1. Forward chaining reasoning
2. History and questions management
3. Backtrack and facts **revocation.**
4. Management of uncertainty through the CF approach proposed for the first time in the MyCIN expert **system.**
5. Explanation and translation form of the technical glossary

10.2 War Crimes Explorer

Create an expert system in prolog to improve the understanding of declarative programming paradigm based on the logic rules. Develop the program with an idea of:

- In-browser learning about genocide, war crimes, crimes against humanity, and aggression.
- Interactively enter facts and discover the laws that might have been broken.
- Possibly submit the information to the ICC (International Criminal Court) as a witness statement.

Input: Set of logical rules with laws and crimes within the jurisdiction of the International Criminal Court.

Output: Model the legal statutes.

Hints

```
/*
 * Crimes within the jurisdiction of the International Criminal Court.
 *
 *
https://world.public.law/rome_statute/article_5_crimes_within_the_jurisdiction_of_the
_court
 */
crime(genocide).
crime(war_crime).
crime(crime_against_humanity).
crime(crime_of_aggression).


/*
 * A first, simple attempt at Protected Persons under
 * the Geneva Conventions of 1949.
 */
protected_by_geneva_convention(P) :- civilian(P).
protected_by_geneva_convention(P) :- prisoner_of_war(P).
protected_by_geneva_convention(P) :- medical_personnel(P).
protected_by_geneva_convention(P) :- religious_personnel(P).
```

```
/*
 * D = Defendant
 * V = Victim
 */

/*
 * Genocide
 * https://world.public.law/rome_statute/article_6_genocide
 */
criminal_liability(genocide, Statute, D, V) :-
        elements(Statute, D, V).

/*
 * War crimes
 * https://world.public.law/rome_statute/article_8_war_crimes
 */
criminal_liability(war_crime, Statute, D, V) :-
        protected_by_geneva_convention(V),
        international_conflict(D, V),
        elements(Statute, D, V).


elements(article_8_2_a_i, D, V) :-
        act(D, killed, V).

elements(article_8_2_a_ii, D, V) :-
        act(D, tortured, V).
```

Try: Create an Expert System suggesting the medical support for the diagnosis of kidney diseases. Include the features like:

1. Forward chaining reasoning
2. History and questions management
3. Backtrack and facts **revocation.**
4. Management of uncertainty through the CF approach proposed for the first time in the MyCIN expert **system.**
5. Explanation and translation form of the technical glossary

---

10.3 DP Film Expert System

Create an expert system in prolog to improve the understanding of declarative programming paradigm based on the logic rules. Develop the program in such a way that it allows you to get film recommendations based on your answer and logic rules with a little film database.

Input: Set of logical rules with some film database.

Output: Recommend a file based on the persons name, mood, sex, time they have, and type of films interested.

Hints

```
# Sample database
/** DRAMA */
film('Zielona mila','Frank Darabont',1999,'drama', 'others', 188,'USA','Tom Hanks',8.7,719).
film('Pif Paf! Jestes trup', 'Guy Ferland', 2002, 'drama', 'others', 87, 'Kanada', 'Ben Foster', 7.7, 16).
film('Dogville', 'Lars von Trier', 2003, 'drama', 'others', 178, 'Dania', 'Nicole Kidman', 7.7, 45).
film('Z dystansu', 'Ton Kayne', 2011, 'drama', 'others', 100, 'USA', 'Adrien Brody', 7.9, 30).
film('Lista Schindlera', 'Steven Spielberg', 1993, 'drama', 'others', 195, 'USA', 'Adrien Brody', 8.4, 259).
film('Requiem dla snu', 'Darren Aronofsky', 2000, 'drama', 'others', 102, 'USA', 'Jared Leto', 7.9, 520).
film('Biutiful', 'Alejandro Gonzalez Inarritu', 2010, 'drama', 'others', 148, 'Hiszpania', 'Javier Bardem', 7.6, 12).
film('Czarny labedz', 'Darren Aronofsky', 2010, 'drama', 'others', 108, 'USA', 'Natalie Portman', 7.7, 248).
film('Gladiator', 'Ridley Scott', 2000, 'drama', 'others', 155, 'USA', 'Russell Crowe', 8.1, 552).
film('Dzien swira', 'Marek Koterski', 2002, 'drama', 'others', 123, 'Polska', 'Marek Kondrat', 7.8, 438).
film('Pianista', 'Roman Polanski', 2002, 'drama', 'others', 150, 'Polska', 'Adrien Brody', 8.3, 410).

/** COMEDY */
film('Seksmisja', 'Juliusz Machulski', 1984, 'comedy', 'others', 118, 'Polska', 'Jerzy Stuhr', 7.9, 420).
film('Forrest Gump', 'Robert Zemeckis', 1994, 'comedy', 'others', 144, 'USA', 'Tom Hanks', 8.6, 697).
film('Kac Vegas', 'Todd Phillips', 2009, 'comedy', 'others', 100, 'USA', 'Bradley Cooper', 7.3, 537).
film('Notykalni', 'Olivier Nakache', 2011, 'comedy', 'others', 112, 'Francja', 'Francois Cluzet', 8.7, 393).
film('Truman Show', 'Peter Weir', 1998, 'comedy', 'others', 103, 'USA', 'Jim Carrey', 7.4, 383).
film('Kiler', 'Juliusz Machulski', 1997, 'comedy', 'others', 104, 'Polska', 'Cezary Pazura', 7.7, 315).
film('Kevin sam w domu', 'Chris Columbus', 1990, 'comedy', 'others', 103, 'USA', 'Macaulay Culkin', 7.1, 297).
film('Mis', 'Stanislaw Bareja', 1980, 'comedy', 'others', 111, 'Polska', 'Stanislaw Tym', 7.8, 261).
film('Diabel ubiera sie u Prady', 'David Frankel', 2006, 'comedy', 'others', 109, 'USA', 'Meryl Streep', 6.9, 227).
film('Jak rozpetalem druga wojne swiatowa', 'Tadeusz Chmielewski', 1969, 'comedy', 'others', 236, 'Polska', 'Marian Kociniak', 7.9, 195).

# Write the predicates related to actors and create some helper functions

# Write the code to design an expert system asking some questions related to the mode, time available, gendre etc.
```

Try: Create an Expert System suggesting the medical support for the diagnosis of kidney diseases. Include the features like:

1. Forward chaining reasoning
2. History and questions management
3. Backtrack and facts **revocation.**
4. Management of uncertainty through the CF approach proposed for the first time in the MyCIN expert **system.**
5. Explanation and translation form of the technical glossary

11. Exercises on Genetic Algorithms

11.1 Generating a Bit Pattern using Predefined Parameters

Generate a bit string that contains a predefined number of ones. Perform the selection process during each iteration by applying the genetic algorithm. Choose the strongest individuals and terminate the weakest one where the survival of the fittest concept comes into play. Carry out the selection process by using a fitness function and compute the strength of each individual.

Input:

1. Deap Python Library
2. Consider the problem as solving the variant of the One Max problem.

Output:

1. Generate a bit string that contains a predefined number of ones.
    a. Generate a bit pattern of length 75.
2. Evaluate all the individuals in the population using the fitness function.
3. Evaluate all the individuals with invalid fitness values.
4. Print the stats for the current generation to see how its progressing.

```
Starting the evolution process

Evaluated 500 individuals

===== Generation 0
Evaluated 297 individuals
Min = 58.0 , Max = 75.0
Average = 70.43 , Standard deviation = 2.91

===== Generation 1
Evaluated 303 individuals
Min = 63.0 , Max = 75.0
Average = 72.44 , Standard deviation = 2.16

===== Generation 2
Evaluated 310 individuals
Min = 65.0 , Max = 75.0
Average = 73.31 , Standard deviation = 1.6

===== Generation 3
Evaluated 273 individuals
Min = 67.0 , Max = 75.0
Average = 73.76 , Standard deviation = 1.41
```

```
===== Generation 57
Evaluated 306 individuals
Min = 68.0 , Max = 75.0
Average = 74.02 , Standard deviation = 1.27

===== Generation 58
Evaluated 276 individuals
Min = 69.0 , Max = 75.0
Average = 74.15 , Standard deviation = 1.18

===== Generation 59
Evaluated 288 individuals
Min = 69.0 , Max = 75.0
Average = 74.12 , Standard deviation = 1.24

==== End of evolution

Best individual:
 [1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0
, 1, 0, 0, 0, 1]

Number of ones: 45
```

Hints

```
/* Install the Python package like DEAP */
$ pip3 install deap

/* Create a new Python file and import the following: */
import random from deap
import base, creator, tools

# Evaluation function
def eval_func(individual):
        target_sum = 45
        return len(individual) - abs(sum(individual) - target_sum),

# Create the toolbox with the right parameters
# Initialize the toolbox
# Generate attributes
# Initialize structures
# Define the population to be a list of individuals
# Register the evaluation operator
# Register the crossover operator
# Register a mutation operator
# Operator for selecting individuals for breeding

/* Write the appropriate code here by printing the stats for the current generation
individuals and print the final output */
```

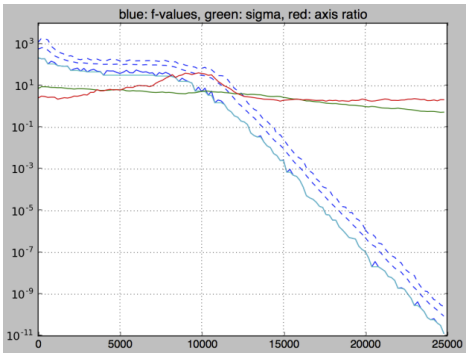
Try: Understand the working of genetic algorithms and use how to use it to solve similar kinds of problems.

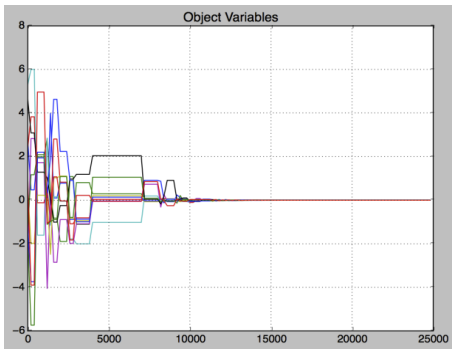
11.2 Visualizing the Evolution

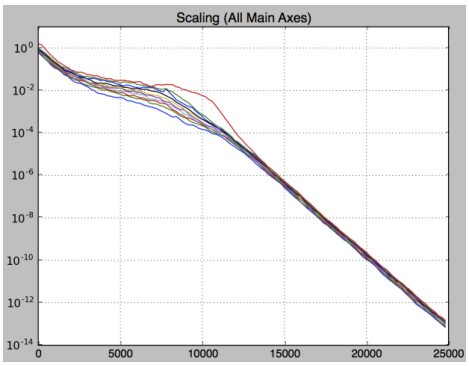
Understand the process of visualizing the evolution process by using a CMA-ES to solve non-linear problems in the continuous domain. The goal of this exercise is to work by delving into the code provided in their source code by using DEAP library.

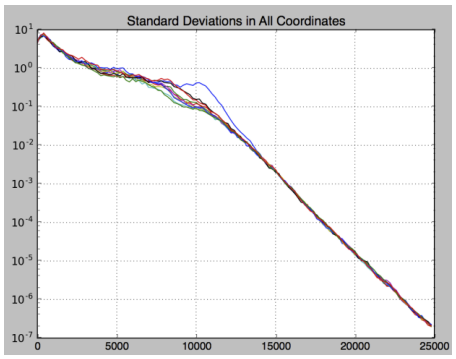
Input: Code mentioned in the DEAP library to implement the evolutionary algorithm for solving non-linear problems in the continuous domain.

Output: Plot the progress as:









Hints

```
/*Create a new Python file and import the following: */
import numpy as np
import matplotlib.pyplot as plt from deap
import algorithms, base, benchmarks, \ cma, creator, tools

/* Define a function to create the toolbox. We will define a FitnessMin function using
negative weights: */

/* Create the toolbox and register the evaluation function */

/* Register the generate and update methods, Define the main function, define a strategy
before we start the process */

/* Create the toolbox based on the strategy */

/* Register the stats using the Statistics method */

/* Define objects to compile all the data, define objects to compile all the data, and
Evaluate individuals using the fitness function */

/* Update the strategy based on the population and save the data for plotting*/

/* Define the x axis and plot the stats and plot the progress */
```

Try: Extend the same program to see the progress printed on the Terminal. Observe the values keep decreasing as the process progress and indicate that it's converging.

11.3 Solving the Symbol Regression Problem

Use genetic programming to solve the symbol regression problem and understand that genetic programming is not the same as genetic algorithms. The goal of this exercise is to understand how the programs are modified, at each iteration.

Input: Create a division operator, define the evaluation function, and compute the mean squared error (MSE).

Output: Run the evolutionary algorithm using the above parameters:

```
population, log = algorithms.eaSimple(population, toolbox,
        probab_crossover, probab_mutate, num_generations,
        stats=mstats, halloffame=hall_of_fame, verbose=True)
```

Hints

```
/* Create a new Python file and import the following: */
import operator import math
import random
import numpy as np
from deap import algorithms, base, creator, tools, gp

/* Create a division operator that can handle divide-by-zero error gracefully: */

/* Create a division operator that can handle divide-by-zero error gracefully: */

/* Compute the mean squared error (MSE) between the function defined earlier and the
original expression: */

/* Define a function to create the toolbox and register the stats using the objects
defined previously. */

/* Define the crossover probability, mutation probability, and the number of
generations */

/* Define the crossover probability, mutation probability, and the number of
generations */
```

		fitness				size			
gen	nevals	avg	max	min	std	avg	max	min	std
0	450	18.6918	47.1923	7.39087	6.27543	3.73556	7	2	1.62449
1	251	15.4572	41.3823	4.46965	4.54993	3.80222	12	1	1.81316
2	236	13.2545	37.7223	4.46965	4.06145	3.96889	12	1	1.98861
3	251	12.2299	60.828	4.46965	4.70055	4.19556	12	1	1.9971
4	235	11.001	47.1923	4.46965	4.48841	4.84222	13	1	2.17245
5	229	9.44483	31.478	4.46965	3.8796	5.56	19	1	2.43168
6	225	8.35975	22.0546	3.02133	3.40547	6.38889	15	1	2.40875
7	237	7.99309	31.1356	1.81133	4.08463	7.14667	16	1	2.57782
8	224	7.42611	359.418	1.17558	17.0167	8.33333	19	1	3.11127
9	237	5.70308	24.1921	1.17558	3.71991	9.64444	23	1	3.31365
10	254	5.27991	30.4315	1.13301	4.13556	10.5089	25	1	3.51898

Try: Building two bots to play Hexapawn against each other.

12. Exercises on Genetic Algorithms

12.1 Building a bot to play Last Coin Standing

This is a game where we have a pile of coins, and each player takes turns to take a number of coins from the pile. There is a lower and an upper bound on the number of coins that can be taken from the pile. The goal of the game is to avoid taking the last coin in the pile.

Input: A class handling all the operations of the game, define who starts the game, overall number of coins in the pile, maximum number of coins per move, and possible moves.

Output:

```
d:2, a:0, m:1
d:3, a:0, m:1
d:4, a:0, m:1
d:5, a:0, m:1
d:6, a:0, m:1
d:7, a:0, m:1
d:8, a:0, m:1
d:9, a:0, m:1
d:10, a:100, m:4
1 10 4
25 coins left in the pile

Move #1: player 1 plays 4 :
21 coins left in the pile

Player 2 what do you play ? 1

Move #2: player 2 plays 1 :
20 coins left in the pile

Move #3: player 1 plays 4 :
16 coins left in the pile
```

```
Move #5: player 1 plays 2 :
11 coins left in the pile

Player 2 what do you play ? 4

Move #6: player 2 plays 4 :
7 coins left in the pile

Move #7: player 1 plays 1 :
6 coins left in the pile

Player 2 what do you play ? 2

Move #8: player 2 plays 2 :
4 coins left in the pile

Move #9: player 1 plays 3 :
1 coins left in the pile

Player 2 what do you play ? 1

Move #10: player 2 plays 1 :
0 coins left in the pile
```

Hints

```
/* Create a new Python file and import the following: */
from easyAI import TwoPlayersGame, id_solve, Human_Player, AI_Player from easyAI.AI
import TT

/* Create a class to handle all the operations of the game */

/* Define who is going to start the game */

/* Define the number of coins in the pile, define the maximum number of coins that can
be taken out in any move */

/* Define all the possible moves, define a method to remove the coins and keep track
of the number of coins remaining in the pile */

/* Check if anyone won the game by checking the number of coins remaining and stop the
game after someone wins its. */

/* Compute the score and define a method to show the current status of the pile */
```

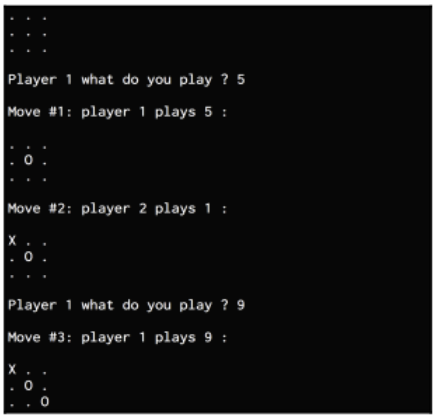
Try: Solve the same game using iterative deepening algorithm by determining who can win a game using all the paths.

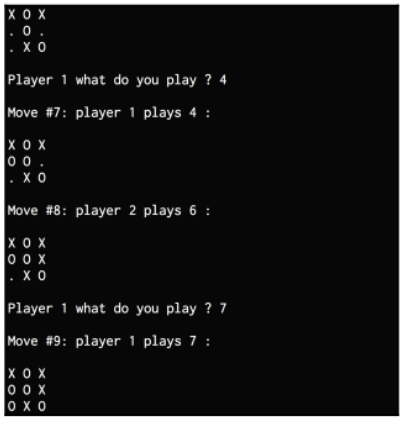
12.2 Building a bot to play Tic-Tac-Toe.

Tic-Tac-Toe (Noughts and Crosses) is probably one of the most famous games. Here the goal is to build a game where the computer can play against the user.

Input: A 3×3 board numbered from one to nine row-wise, all possible moves, and updates the board moves until some player has lost the game.

Output:





Hints

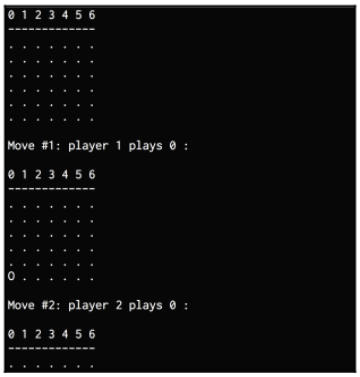
```
# Create a new Python file and import the following packages:
from easyAI import TwoPlayersGame, AI_Player, Negamax
from easyAI.Player import Human_Player

# Define a class that contains all the methods to play the game. Start by defining the
players and who starts the game

# Define a method to compute all the possible moves

# Define a method to update the board after making a move

# Define a method to see if somebody has lost the game

# Define a method to show the current progress and compute the score using the
loss_condition method
```

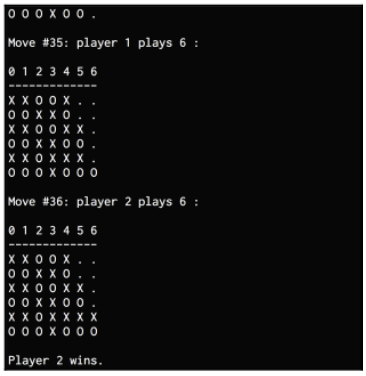
12.3 Building Two Bots to Play Connect Four against each other.

Connect Four™ is a popular two-player game sold under the Milton Bradley trademark. It is also known by other names such as Four in a Row or Four Up. In this game, the players take turns dropping discs into a vertical grid consisting of six rows and seven columns. The goal is to get four discs in a line. This is a variant of the Connect Four recipe given in the easyAI library. In this recipe, instead of playing against the computer, create two bots that will play against each other. We will use a different algorithm for each to see which one wins.

Input: A board with six rows and seven columns, define the who is going to start the games, define the positions, and define all the possible legal moves.

Output: Compute the score and print the results. Get the following output on your Terminal at the beginning and towards the end.





Hints

```
# Create a new Python file and import the following packages:
import numpy as np from easyAI
import TwoPlayersGame, Human_Player, AI_Player, \ Negamax, SSS

# Define a class that contains all the methods needed to play the game

# Define the board with six rows and seven columns

# Define who's going to start the game. In this case, let's have player one starts the
game and define the position as:

# Define the positions
self.pos_dir = np.array([[[i, 0], [0, 1]]
        for i in range(6)] + [[[0, i], [1, 0]]
        for i in range(7)] + [[[i, 0], [1, 1]]
        for i in range(1, 3)] + [[[0, i], [1, 1]]
        for i in range(4)] + [[[i, 6], [1, -1]]
        for i in range(1, 3)] + [[[0, i], [1, -1]] for i in range(3, 7)])

# Define a method to get all the possible moves, define a method to control how to
make a move, and define a method to show the current status.
```

Try: Building two bots to play Hex pawn against each other.

13. Exercises on Robotics

13.1 Two-Persons, Perfect Information Games

Consider games with just two outcomes: win and Loss. Games where a draw is a possible outcome can be reduced to two outcomes: win, not win. The two players will be called 'us' and 'them'. 'Us' can win in a non-terminal 'us-to-move' position if there is a legal move that leads to a won position. On the other hand, a non-terminal 'them-to-move' position is won for 'us' if all the legal moves from this position lead to won positions. These rules correspond to AND/OR tree representation of problems. The goal of this exercise is to find whether an us-to-move position is won.

Input: Rules corresponding to AND/OR trees that can be adopted for searching game trees.

Output: Find whether an us-to-move position

Hints

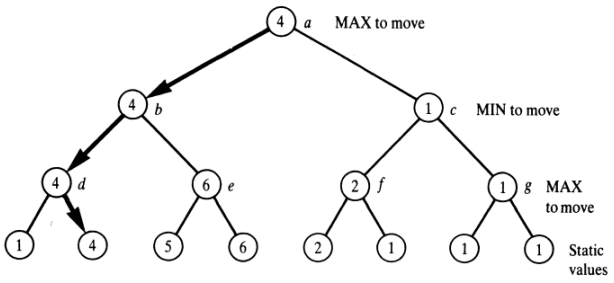
```
# Create a new Python file and import the following packages:
import numpy as np from easyAI
import TwoPlayersGame, Human_Player, AI_Player, \ Negamax, SSS

# Define a class that contains all the methods needed to play the game

# Define the board with six rows and seven columns

# Define who's going to start the game. In this case, let's have player one starts the
game and define the position as:

# Define the positions
self.pos_dir = np.array([[[i, 0], [0, 1]]
        for i in range(6)] + [[[0, i], [1, 0]]
        for i in range(7)] + [[[i, 0], [1, 1]]
        for i in range(1, 3)] + [[[0, i], [1, 1]]
        for i in range(4)] + [[[i, 6], [1, -1]]
        for i in range(1, 3)] + [[[0, i], [1, -1]] for i in range(3, 7)])

# Define a method to get all the possible moves, define a method to control how to
make a move, and Define a method to show the current status.
```

Try: Write a program to play some simple game (like nim) using the straightforward AND/OR search approach.

13.2 The minimax principle

The goal of this exercise is the straightforward implementation of the minimax principle.

Input:



Output: Compute the minimax backed-up value for a given position.

Hints

```
# minimax_simplenim.py

def minimax(state, max_turn):
    if state == 0:
        return 1 if max_turn else -1

    # ...
```

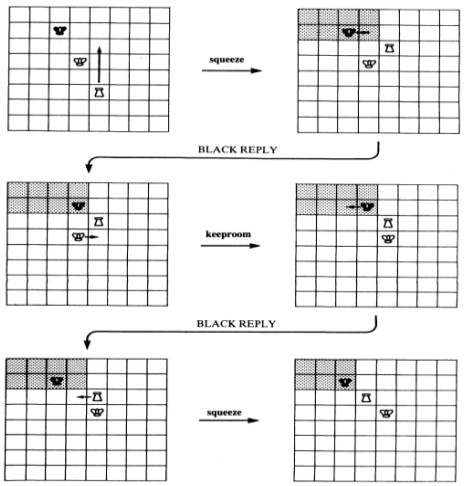
```
def minimax(state, max_turn):
    if state == 0:
        return 1 if max_turn else -1

    possible_new_states = [
        state - take for take in (1, 2, 3) if take <= state
    ]
    if max_turn:
        scores = [
            minimax(new_state, max_turn=False)
            for new_state in possible_new_states
        ]
        return max(scores)

    # ...
```

Try: You should play a few games of Nim to get a feel for how the new rules change the strategy. Try it with a different number of piles, say three, four, or five. You don't need a lot of counters in each pile. Between three and nine is a good starting point.

13.3 Advice Language 0

A broad strategy for winning with the king and rook against the sole opponent's king is to force the king to the edge, or into a corner if necessary, and deliver mate in a few moves. The goal is to develop a program that a play a game from a given starting position using knowledge representation in Advice Language 0.

Input: Goal predicates and Move constraint predicates.



Output: Predicate library for king and rock Vs king.

Hints

```
# minimax_simplenim.py

def minimax(state, max_turn):
    if state == 0:
        return 1 if max_turn else -1

    # ...
```

```
def minimax(state, max_turn):
    if state == 0:
        return 1 if max_turn else -1

    possible_new_states = [
        state - take for take in (1, 2, 3) if take <= state
    ]
    if max_turn:
        scores = [
            minimax(new_state, max_turn=False)
            for new_state in possible_new_states
        ]
        return max(scores)

    # ...
```

Try: Consider some other simple chess endgames, such as king and pawn vs. king, and write an ALO program (together with the corresponding predicate definitions) to play this endgame.

14. Exercises on Neural Networks

14.1 Building a Perceptron Based Classifier

A Perceptron is the building block of an artificial neural network. It is a single neuron that takes inputs, performs computation on them, and then produces an output. It uses a simple linear function to make the decision. Let's say we are dealing with an N-dimension input data point. A Perceptron computes the weighted summation of those N numbers, and it then adds a constant to produce the output. The constant is called the bias of the neuron. It is remarkable to note that these simple Perceptron's are used to design very complex deep neural networks. Develop the python code to build a perceptron-based classifier using NeuroLab.

Input: Text file data_perceptron.txt

Output: Plot showing the training progress using the error metric.

Hints

```
# Create a new python file and import the following packages: #

import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Load the input data and separate each data into datapoints and labels.

# Plot the input data.

# Extract the minimum and maximum values for each dimension.

# Define a perceptron with 2 input neurons.
```

```
# Train the perceptron using the training data and plot the training progress using the error metric.
```

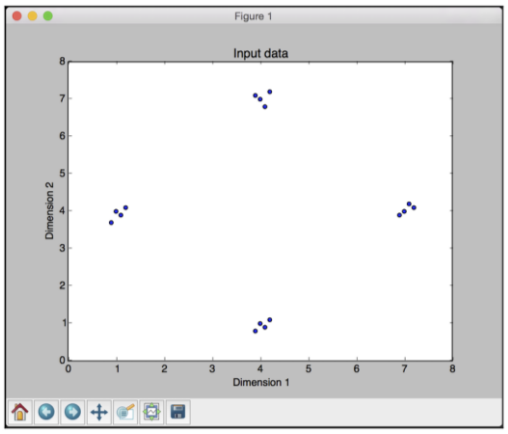
Try: Implement the same code using different datasets.

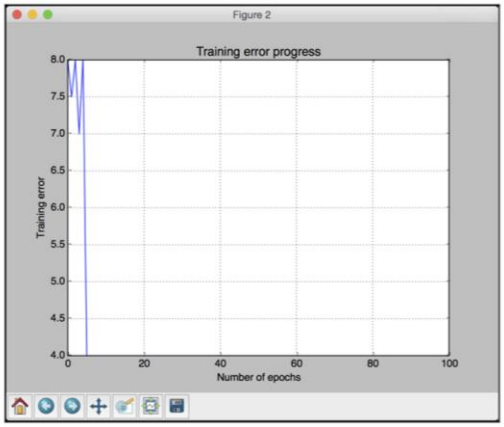
14.2 Constructing a Single Layer Neural Network

To enable higher accuracy, we need to give more freedom to the neural network. This means that a neural network needs more than one layer to extract the underlying patterns in the training data. Develop the python code to create a single layer neural network to extract the training data patterns.

Input: Generate the training data

Output: Error rates for each epoch and test results.






```
Epoch: 20; Error: 4.0;
Epoch: 40; Error: 4.0;
Epoch: 60; Error: 4.0;
Epoch: 80; Error: 4.0;
Epoch: 100; Error: 4.0;
The maximum number of train epochs is reached

Test results:
[0.4, 4.3] --> [ 0.  0.]
[4.4, 0.6] --> [ 1.  0.]
[4.7, 8.1] --> [ 1.  1.]
```

Hint

```
# Create a new python file and import the following packages: #

import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Load the input data and separate each data into datapoints and labels.

# Plot the input data.

# Extract the minimum and maximum values for each dimension.
```

```
# Define the number of neurons in the output layer.

# Define a single layer neural network using above parameters.

# Train the neural network using the training data that was generated.

# Run the neural network on the training datapoints and plot the training progress.

# Plot the training progress.

# Define some sample test datapoints and run the network on those points.
```

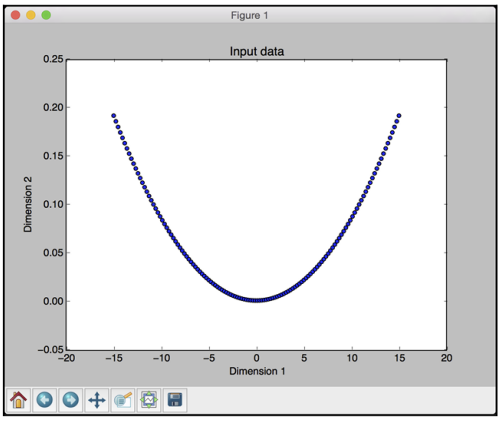
Try: Continue training the network and reduce the errors. Verify that the predicted outputs are corrected by locating the data points on a 2D graph.

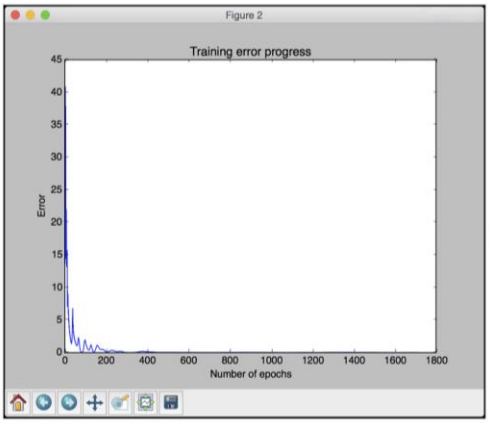
14.3 Constructing a Multi-Layer Neural Network

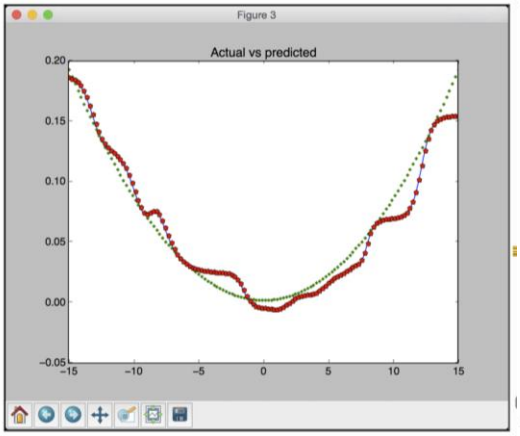
To enable higher accuracy, we need to give more freedom to the neural network. This means that a neural network needs more than one layer to extract the underlying patterns in the training data. Develop the python code to create a multilayer neural network to extract the training data patterns.

Input: Generate the training data

Output: Error rates for each epoch.








```
Epoch: 100; Error: 1.9247718251621995;
Epoch: 200; Error: 0.15723294798079526;
Epoch: 300; Error: 0.021680213116912858;
Epoch: 400; Error: 0.1381761995539017;
Epoch: 500; Error: 0.04392553381948737;
Epoch: 600; Error: 0.02975401597014979;
Epoch: 700; Error: 0.014228560930227126;
Epoch: 800; Error: 0.03460207842970052;
Epoch: 900; Error: 0.035934053149433196;
Epoch: 1000; Error: 0.025833284445815966;
Epoch: 1100; Error: 0.013672412879982398;
Epoch: 1200; Error: 0.01776586425692384;
Epoch: 1300; Error: 0.04310242610384976;
Epoch: 1400; Error: 0.03799681296096611;
Epoch: 1500; Error: 0.02467030041520845;
Epoch: 1600; Error: 0.010094873168855236;
Epoch: 1700; Error: 0.01210866043021068;
The goal of learning is reached
```

Hint

```
# Create a new python file and import the following packages: #

import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Generate some sample data points based on the equation y = 3x^2 + 5 and then normalize
the points

# Reshape the above variables to create a training dataset:

# Plot the input data.

# Define a multilayer neural network with two hidden layers.

# Set the training algorithm to gradient descent.

# Train the neural network using the training data that was generated.

# Run the neural network on the training datapoints and plot the training progress.

# Plot the predicted output.
```

Try: Continue training the network and reduce the errors. Try to see that the predicted output will match the input curve even more accurately.

15. Exercises on Natural Language Processing

15.1 Converting words to their base forms using stemming

Working with text has a lot of variations included in it. We have to deal with different forms of the same word and enable the computer to understand that these different words have the same base form. For example, the word sing can appear in many forms such as sang, singer, singing, singer, and so on. We just saw a set of words with similar meanings. Humans can easily identify these base forms and derive context. Develop the python code to analyze the text and identify the base forms and derive the context.

Input: Bag of some input words like writing, calves, branded etc.

Output: Extract useful statistics to analyze the input text.

INPUT WORD	PORTER	LANCASTER	SNOWBALL
writing	write	writ	write
calves	calv	calv	calv
be	be	be	be
branded	brand	brand	brand
horse	hors	hors	hors
randomize	random	random	random
possibly	possibl	poss	possibl
provision	provis	provid	provis
hospital	hospit	hospit	hospit
kept	kept	kept	kept
scratchy	scratchi	scratchy	scratchi
code	code	cod	code

Hint

```
# Create a new python file and import the following packages.
```

```
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.snowball import SnowballStemmer

# Define some input words

# Create objects for Porter, Lancaster, and Snowball stemmers.

# Create a list of names for table display and format the output text accordingly.

# Iterate through the words and stem them using the three stemmers
```

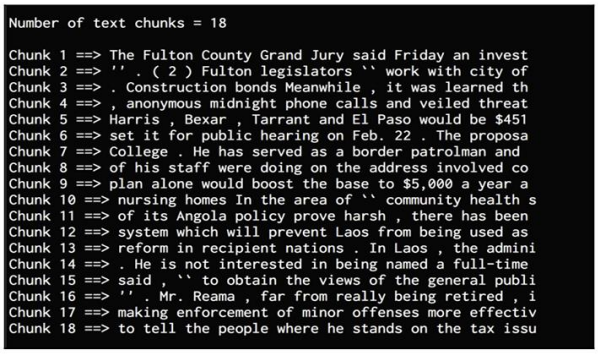
Try: Implement the code to understand the three stemming algorithms to achieve the same goal. Converting words to their base forms using lemmatization.

15.2 Dividing text data into chunks

Text data usually needs to be divided into pieces for further analysis. This process is known as chunking. This is used frequently in text analysis. The conditions that are used to divide the text into chunks can vary based on the problem at hand. This is not the same as tokenization where we also divide text into pieces. During chunking, we do not adhere to any constraints and the output chunks need to be meaningful. Develop the python code to divide the text into chunks to extract meaningful information.

Input: A large text document named brown.

Output: Divide the input text into chunks and display the output.



Hint

```
# Create a new python file and import the following packages.

import numpy as np
from nltk.corpus import brown

# Define a function to divide the input text into chunks.

# Iterate through the words and divide them into chunks using the input parameter.

# Define the main function and read the input data using the Brown corpus.

# Define the number of words in each chunk:
```

```
# Divide the input text into chunks and display the output:
```

Try: Implement the code to understand the three stemming algorithms to achieve the same goal. Converting words to their base forms using lemmatization.

15.3 Extracting the frequency of terms using a Bag of Words Model

One of the main goals of text analysis is to convert text into numeric form so that we can use machine learning on it. Let's consider text documents that contain many millions of words. In order to analyze these documents, develop the python code to extract the text and convert it into a form of numeric representation.

Input: Consider the following sentences.

Sentence 1: The children are playing in the hall
Sentence 2: The hall has a lot of space
Sentence 3: Lots of children like playing in an open space

If you consider all the three sentences, we have the following nine unique words: the, children, are, playing, in, hall, has, a, lot, of, space, like, an, open.

Output: Extract useful statistics to analyze the input text.

INPUT WORD	PORTER	LANCASTER	SNOWBALL
writing	write	writ	write
calves	calv	calv	calv
be	be	be	be
branded	brand	brand	brand
horse	hors	hors	hors
randomize	random	random	random
possibly	possibl	poss	possibl
provision	provis	provid	provis
hospital	hospit	hospit	hospit
kept	kept	kept	kept
scratchy	scratchi	scratchy	scratchi
code	code	cod	code

Hint

```
# Create a new python file and import the following packages.

import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import brown
from text_chunker import chunker

# Build a bag of words model in NLTK.

# Define the number of words in each chunk

# Divide the input text into chunks:

# Convert the chunks into dictionary items:

# Extract the document term matrix where we get the count of each word.

# Extract the vocabulary and display it and generate the names for display.

# Print the document term matrix.
```

Try: 1. Build a category predictor.
2. Construct a gender identifier.
3. Building a sentiment analyzer

16. Final Notes

The only way to learn programming is program, program, and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests. Check out these sites:

1. Introduction to Artificial Intelligence with Python, Associated with Harvard University. CS50's Introduction to Artificial Intelligence with Python | Harvard University

2. NPTEL: An Introduction to Artificial Intelligence, https://nptel.ac.in/courses/106105077/

3. NPTEL: Artificial Intelligence Search Methods for Problem Solving, Artificial Intelligence Search Methods for Problem Solving - Course (nptel.ac.in).

4. IFACET (iitk.ac.in)

5. Introduction to Artificial Intelligence (AI) | Coursera in association with IBM.

6. http://www.ai.eecs.umich.edu

Student must have any one of the following certifications:

- Competitive Coding with AlphaCode Team - Competitive programming with AlphaCode (deepmind.com)
- IIIT Hyderabad Certification - Competitive programming with AlphaCode (deepmind.com)