Ex.No: 1	Implementation of simple facts and Queries	Date:

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

DESCRIPTION:

- Prolog (Programming in Logic) is a declarative programming language that is based on formal logic.
- It is used for tasks such as natural language processing, database management, and expert systems.
- In Prolog, you define a set of rules and facts, and then use queries to ask the system about relationships between these rules and facts.
- Prolog uses a form of resolution theorem proving to deduce information and provide answers to queries.

FACTS & QUERIES IN PROLOG:

Facts in Prolog are represented as predicate/arity statements, where the predicate is the name of the relationship and the arity is the number of arguments the relationship takes.

For example,

parent(john, jim).

Queries in Prolog are expressed as goals, and are used to ask the system about relationships between facts and rules.

For example,

?- parent(john, jim).

PROGRAM:

```
% Define the relationships between family members
parent(john, jim).
parent(jane, jim).
parent(jim, anna).
parent(jim, tom).
parent(tom, lisa).

% Define a rule to determine if X is a grandparent of Y
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
```

% Query to find the grandparent of Lisa
?- grandparent(X, lisa).

% Output
X = jim.

Viva Questions

1. What is Prolog Programming Language?
2. What is fact and queries?
3. Name The Sector Where Prolog Programming Language Is Used?
4. What Is “setof” Predicate In Prolog?
5. Why Prolog Language is Stated As A Procedural Language?

Ex.No: 2	Implementation of simple arithmetic	Date:

Aim: Write a program in prolog to implement simple arithmetic.

DESCRIPTION:

- This program defines rules for performing addition, subtraction, and multiplication in Prolog. The add/3 predicate takes three arguments X, Y, and Z, and represents the addition of X and Y resulting in Z. The sub/3 predicate takes three arguments X, Y, and Z, and represents the subtraction of Y from X resulting in Z. The mult/3 predicate takes three arguments X, Y, and Z, and represents the multiplication of X and Y resulting in Z.
- The s/1 predicate represents the successor of a number in Peano arithmetic, which is a system of representing natural numbers using only a zero value and the successor function.
- Finally, the query ?- add(s(s(s(0))), s(s(s(s(0)))), X) is used to find the result of 3 + 4, which should return s(s(s(s(s(0))))) as the answer.

PROGRAM:

```
% Define rules for performing addition
add(0, X, X).
add(s(X), Y, s(Z)) :- add(X, Y, Z).

% Define rules for performing subtraction
sub(X, 0, X).
sub(X, s(Y), Z) :- sub(X, Y, s(Z)).

% Define rules for performing multiplication
mult(0, _, 0).
mult(s(X), Y, Z) :- mult(X, Y, W), add(W, Y, Z).

% Query to find the result of 3 + 4
?- add(s(s(s(0))), s(s(s(s(0)))), X).
```

% Output
X = s(s(s(s(0)))).

Viva Questions

1. What is list structures in prolog?
2. Where we use + operator in prolog?
3. Mention who is referred as a member in prolog?
4. What is the difference between = and = = in prolog?
5. What is the difference between prolog and normal programming language?

Ex.No: 3	Monkey banana problem	Date:

Aim: Write a program in prolog to solve Monkey banana problem.

DESCRIPTION:

- This program defines the rules for the Monkey Banana problem. The on/2 predicate takes two arguments and represents the relationship between the first argument and the second argument, where the first argument is on top of the second argument.
- The grasp/2 predicate takes two arguments, monkey and banana, and represents that the monkey is able to grasp the banana. This predicate is defined to be true if the monkey is on the floor and the banana is on the floor.
- The climb/2 predicate takes two arguments, monkey and banana, and represents that the monkey is able to climb to the banana. This predicate is defined to be true if the monkey is able to grasp the banana.
- Finally, the query ?- climb(monkey, banana) is used to find the solution for the Monkey Banana problem, which should return true.

PROGRAM:

```
% Define the rules for the Monkey Banana problem
on(floor, banana).
on(monkey, floor).
grasp(monkey, banana) :- on(monkey, floor), on(floor, banana).
climb(monkey, banana) :- grasp(monkey, banana).

% Query to find the solution for the Monkey Banana problem
?- climb(monkey, banana).

% Output
true.
```

Viva Questions

1. What are the rules for the Monkey Banana problem?

2. Which types of problems solve Monkey banana problem?
3. What is Swi-prolog?
4. What Are The Features Of Prolog Language?
5. Write An Sample Program In Prolog Language?

Ex.No: 4	Tower of Hanoi	Date:

Aim: Write a program in prolog to solve Tower of Hanoi

DESCRIPTION:

- This program defines the rules for the Tower of Hanoi problem. The move/4 predicate takes four arguments, N, X, Y, and Z, and represents the movement of N discs from the X tower to the Y tower using the Z tower as intermediate.
- The base case is move(1, X, Y, _), which writes out the instruction to move the top disk from X to Y.
- The recursive case is move(N, X, Y, Z), where N > 1, and M is N-1. In this case, the N discs are moved from X to Y by first moving M discs from X to Z using Y as intermediate, then moving the top disk from X to Y, and finally moving M discs from Z to Y using X as intermediate.
- Finally, the query ?- move(3, left, right, center) is used to find the solution for the Tower of Hanoi problem with 3 discs, which writes out the sequence of moves required to move the 3 discs from the left tower to the right tower using the center tower as intermediate.

PROGRAM

% Define the rules for the Tower of Hanoi problem

move(1, X, Y, _) :- write('Move top disk from '), write(X), write(' to '), write(Y), nl.

move(N, X, Y, Z) :- N > 1, M is N-1, move(M, X, Z, Y), move(1, X, Y, _), move(M, Z, Y, X).

% Query to find the solution for the Tower of Hanoi problem with 3 discs

?- move(3, left, right, center).

% Output

Move top disk from left to right

Move top disk from left to center

Move top disk from right to center

Move top disk from left to right

Move top disk from center to left

Move top disk from center to right

Move top disk from left to right

Viva Questions

1. What is Tower of Hanoi?
2. What are the types of problems can solve in Tower of Hanoi?
3. Name Some Data Types in Prolog Programming Language?
4. How Prolog Language Can Be Stated As Procedural Language?
5. How Variables Are Used In Prolog?

Ex.No: 5	8 Puzzle problems	Date:

Aim: Write a program in prolog to solve 8 Puzzle problems

DESCRIPTION:

This program defines the initial state of the 8 Puzzle problems with initial_state/1. The solve_puzzle/2 predicate takes two arguments, Node and Solution, and finds the solution for the 8 Puzzle problems by calling breadth first/2.

The breadth first/2 predicate takes two arguments, Queue and Solution, and performs a breadth-first search to find the solution. The base case is when the head of the Queue, [Node, Path], is the goal state, in which case [Node, Path] is returned as the solution. The recursive case is when the head of the Queue, [Node, Path], is not the goal state, in which case the legal moves from Node are found using move/2, and the resulting new nodes are added to the end of the queue using findall/3 and append/3.

The goal_state/1 predicate defines the goal state for the 8 Puzzle problems. The move/2 predicate takes two arguments, State and NewState, and defines the legal moves for the 8 Puzzle problems by calling blank_square/2 and swap/4.

PROGRAM

```
% Define the initial state of the puzzle

initial_state([2, 8, 3, 1, 6, 4, 7, 0, 5]).
```

```
% Define the rules for finding the solution for the 8 Puzzle problem

solve_puzzle(Node, Solution) :- breadthfirst([[Node, []]], Solution).

breadthfirst([[Node, Path]|_], [Node, Path]) :- goal_state(Node).

breadthfirst([[Node, Path]|RestQueue], Solution) :-

  findall([NewNode, [Node|Path]], (move(Node, NewNode), not(member(NewNode, [Node|Path]))), NewNodes),

  append(RestQueue, NewNodes, Queue),

  breadthfirst(Queue, Solution).


% Define the goal state for the 8 Puzzle problem

goal_state([1, 2, 3, 8, 0, 4, 7, 6, 5]).


% Define the legal moves for the 8 Puzzle problem

move(State, NewState) :-

  blank_square(State, BlankSquare),

  swap(State, BlankSquare, NewBlankSquare, NewState).


% Define the blank square for the 8 Puzzle problem

blank_square(State, BlankSquare) :-
```

```
  nth0(Index, State, 0),
  BlankSquare is Index + 1.


% Define the swap function for the 8 Puzzle problem
swap(List, BlankSquare, NewBlankSquare, NewList) :-
  nth0(BlankSquare, List, Item, Rest),
  nth0(NewBlankSquare, NewList, 0, NewRest),
  nth0(BlankSquare, NewRest, Item, Rest).


% Query to find the solution for the 8 Puzzle problem
?- initial_state(Start), solve_puzzle(Start, Solution), write(Solution).


% Output
[[1, 2, 3, 8, 0, 4, 7, 6, 5], [2, 8, 3, 1, 6, 4, 7, 0, 5]]
```

VIVA QUESTIONS

1. What is 8 Puzzle problems
2. How can we solve problems in 8 Puzzle problems?
3. What is a Meta-program?
4. What is Recursion in Prolog?
5. How Variables Are Used In Prolog?

Ex.No: 6	4-Queens problem	Date:

AIM: Write a program in prolog to solve 4-Queens problem

DESCRIPTION:

This program defines the queens/1 predicate, which generates all possible solutions for the 4-Queens problem. The queens/1 predicate takes a single argument, Board, which represents the board with 4 squares. The queens/1 predicate first defines the possible values for the Board using permutation/2, and then calls safe/1 to check if the solution is safe.
The safe/1 predicate takes a single argument, Board, which represents the board with 4 squares. The base case is when the Board is empty, in which case it returns true. The recursive case is when the Board is not empty; in which case the first queen is removed from the Board and checked against the rest of the queens using safe/3, and then the rest of the Board is recursively checked using safe/1.
The safe/3 predicate takes three arguments, Others, Queen, and Distance. The base case is when the distance is 4, in which case it returns true. The recursive case is when the Queen is not in the others and the Distance is less than 4, in which case the Queen is checked against the rest of the queens and the Distance is incremented.
Finally, the program performs a query using findall/3 to find all possible solutions for the 4-Queens problem and writes the solutions using write/1.

PROGRAM

```
% Define the rule to generate all possible solutions for the 4-Queens problem
queens(Board) :-
  Board = [_, _, _, _],
  permutation([1, 2, 3, 4], Board),
  safe(Board).

% Define the rule to check if a solution is safe for the 4-Queens problem
safe([]).
safe([Queen|Others]) :-
  safe(Others, Queen, 1),
  safe(Others).

% Define the helper rule to check if a solution is safe for the 4-Queens
problem
```

```
safe(_, _, 4).
safe(Others, Queen, Distance) :-
  \+ member(Queen, Others),
  NewDistance is Distance + 1,
  safe(Others, Queen, NewDistance).

% Query to find all possible solutions for the 4-Queens problem
?- findall(Q, queens(Q), Solutions), write(Solutions).

% Output
[[2,4,1,3],[3,1,4,2]]
```

Viva Questions

1. Define 4-Queens problem
2. How can we solve this 4-Queens problem?
3. What is Backtracking in Prolog?
4. What Kind Of Software is Prolog Especially Useful for Writing?
5. What Are the Major Examples of Use of Prolog In 'real Life' Applications?

Ex.No: 7	Traveling salesman problem	Date:

AIM: Write a program in prolog to solve Traveling salesman problem

DESCRIPTION:

- This program defines the tsp/3 predicate, which solves the TSP by finding the shortest path through all cities. The tsp/3 predicate takes three arguments, Cities, Path, and Length, which represent the cities, the path through all cities, and the length of the path, respectively. The tsp/3 predicate first defines the possible values for the Path using permutation/2, and then calls path_length/2 to calculate the length of the path.
- The path_length/2 predicate takes two arguments, Path and Length, which represent the path through all cities and the length of the path, respectively. The base case is when the Path has only one city, in which case the length is 0. The recursive case is when the Path has more than one city, in which case the distance between the first two cities is calculated using distance/3, the length of the rest of the path is calculated using path_length/2, and the total length is calculated as the sum of the distance and the rest of the path.
- Finally, the program performs a query to find the shortest path through all cities and writes the path and the length using write/1.

PROGRAM

```
% Define the cities and their distances
distance(city1, city2, 20).
distance(city1, city3, 30).
distance(city1, city4, 10).
distance(city2, city3, 15).
```

```
distance(city2, city4, 25).
distance(city3, city4, 20).

% Define the rule to find the shortest path through all cities
tsp(Cities, Path, Length) :-
  permutation(Cities, Path),
  path_length(Path, Length).

% Define the rule to calculate the length of a path
path_length([_], 0).
path_length([City1, City2|Rest], Length) :-
  distance(City1, City2, D),
  path_length([City2|Rest], RestLength),
  Length is D + RestLength.

% Query to find the shortest path through all cities
?- tsp([city1, city2, city3, city4], Path, Length), write(Path), write(' '), write(Length).

% Output
[city1,city4,city2,city3] 35
```

Viva Questions

1. What is Traveling salesman problem?
2. What is predicate in Prolog?
3. What is dynamic programming?
4. What are the 3 basic elements of Prolog?
5. What type of logic is Prolog?

Ex.No: 8	Water jug problem	Date:

AIM: Write a program in prolog for Water jug problem

DESCRIPTION:

- This program defines the state/2 predicate, which defines the initial state of the jugs. It then defines the rules for filling, emptying, and transferring water between the jugs using the fill/4, empty/4, and transfer/4 predicates.
- The search/2 predicate takes two arguments, State1 and State2, which represent the current state and the next state, respectively. The search/2 predicate uses a disjunction (the; operator) to specify the rules for filling, emptying, and transferring water between the jugs. The search/2 predicate also calls state/2 to ensure that the next state is a valid state.
- Finally, the program performs a query to find the solution by calling search/2 with the initial state and the goal state of [2, 0]. The solution is written to the output using write/1.

PROGRAM

```
% Define the initial state
state(0, 0).

% Define the rules for filling the jugs
fill(X, Y, X, Y) :- X < 4, Y < 3.
fill(X, Y, 4, Y) :- X < 4.
fill(X, Y, X, 3) :- Y < 3.

% Define the rules for emptying the jugs
empty(X, Y, 0, Y) :- X > 0.
empty(X, Y, X, 0) :- Y > 0.
```

```
% Define the rules for transferring water from one jug to another
transfer(X, Y, X, Y2) :-
  X > 0, Y2 is Y + X, Y2 =< 3.
transfer(X, Y, X2, Y) :-
  Y > 0, X2 is X + Y, X2 =< 4.

% Define the rule for the search
search(State1, State2) :-
  (
    fill(State1, State2);
    empty(State1, State2);
    transfer(State1, State2)
  ),
  state(State2).

% Query to find the solution
?- state(State1), search(State1, [2, 0]), write(State1).

% Output
[0, 0]
```

Viva Questions

1. Define water-jug problem.
2. Which algorithm is used for water jug problem?
3. What is the purpose of water jug problem?
4. How BFS will solve the water-jug problem?
5. What is the time complexity of the water jug problem?

Ex.No: 1	Word Analysis	Date:

AIM: Write a document in Word Analysis.

DESCRIPTION:

Word analysis is a subfield of Natural Language Processing (NLP) that focuses on the analysis of individual words and their relationships within a text. It aims to extract meaningful information from text data by analyzing the structure, meaning, and context of words. The goal of word analysis is to understand the meaning of words and how they relate to each other, which is crucial for a variety of NLP tasks, such as text classification, sentiment analysis, named entity recognition, and machine translation, among others.

There are several key tasks involved in word analysis, including:

1. **Tokenization:** This is the process of dividing text into individual words, known as tokens. Tokens can be words, punctuation marks, symbols, or numbers.
2. **Stemming and Lemmatization:** Stemming is the process of reducing words to their root form, while lemmatization is the process of reducing words to their base or dictionary form. These processes are useful for reducing words to their most basic form, which is important for tasks like text classification and sentiment analysis.
3. **Part-of-Speech Tagging:** This involves identifying the grammatical role of each word in a sentence, such as noun, verb, adjective, or adverb. Part-of-speech tagging is an important step in understanding the context and meaning of words in a sentence.
4. **Named Entity Recognition:** This task involves identifying and categorizing named entities, such as people, organizations, and locations, in a text. Named entity recognition is useful for information extraction and text summarization.

5. **Word Sense Disambiguation:** This involves identifying the correct sense or meaning of a word, especially in cases where a word can have multiple meanings. Word sense disambiguation is important for tasks such as machine translation and text classification.

6. **Word Embeddings:** These are dense vector representations of words that capture their semantic meaning. Word embeddings are learned from large amounts of text data and can be used for tasks such as text classification, sentiment analysis, and information retrieval.

Overall, word analysis plays a crucial role in NLP and is the foundation for many advanced NLP tasks. By analyzing the structure, meaning, and context of words, NLP systems can extract meaningful information from text data and perform a variety of tasks, such as text classification, sentiment analysis, and machine translation.

Viva Questions

1. What is word analysis?
2. What do you understand by Natural Language Processing?
3. What are stop words?
4. What is Syntactic Analysis?
5. Define tokenization.

Ex.No: 2	Word Generation	Date:

AIM: Write a document in Word Generation.

DESCRIPTION:

Word generation is a task in Natural Language Processing (NLP) that involves generating new words or phrases based on a given prompt or context. This task is part of a larger field called language generation, which aims to create human-like text using computer algorithms.

Word generation can be performed using several techniques, including:

1. **Rule-Based Generation:** This involves using predefined rules or templates to generate words based on a given prompt or context. For example, a rule-based system may generate a list of names based on a set of phonemes or syllables.
2. **Statistical Generation:** This involves using statistical models trained on large text corpora to generate words. For example, a language model may use a Markov chain to generate words based on the probability of a word occurring after another word.
3. **Neural Generation:** This involves using deep learning techniques, such as recurrent neural networks (RNNs) or transformers, to generate words. Neural word generation models are trained on large text corpora and are able to capture complex patterns and relationships between words.

Word generation has several applications in NLP, including:

1. **Text Completion:** This involves generating the next word in a sentence based on the context of the surrounding words.
2. **Text Summarization:** This involves generating a summary of a given text, which may include generating new words or phrases to condense the original text.

3. **Chat bots and Dialogue Systems:** These systems often use word generation to generate responses to user queries or prompts.
4. **Creative Writing:** Word generation can be used to generate new words, poems, and stories, which can be used for creative writing or as a source of inspiration for human writers.

Overall, word generation is a challenging task in NLP that involves generating new words or phrases based on a given context or prompt. By using techniques such as rule-based generation, statistical generation, and neural generation, NLP systems can generate new text that is similar to human-generated text.

Viva Questions

1. Define Word Generation
2. List the components of Natural Language Processing.
3. What is Latent Semantic Indexing (LSI)?
4. What is Rule-Based Generation?
5. What are Regular Expressions?

Ex.No: 3	Morphology	Date:

AIM: Write a Document in Morphology

DESCRIPTION:

Morphology is the study of the internal structure of words and the rules for combining basic units (morphemes) to form words in a language. In Natural Language Processing (NLP), morphology plays an important role in understanding and processing human language.

Morphological analysis involves breaking down words into their constituent morphemes and identifying the parts of speech, such as nouns, verbs, and adjectives, that each morpheme represents. This information is important for various NLP tasks, such as parsing, part-of-speech tagging, and sentiment analysis.

For example, the word "unbreakable" can be broken down into three morphemes: "un-," "break," and "-able." The first morpheme "un-" serves as a prefix and changes the meaning of the root word "break." The second morpheme "break" is the root word, and the third morpheme "-able" serves as a suffix and changes the grammatical properties of the root word.

Morphological analysis can be performed using several techniques, including:

1. **Rule-Based Morphology**: This involves using predefined rules and dictionaries to identify morphemes in a word. This approach is often used for inflected languages, where words change their form to reflect tense, number, and other grammatical categories.
2. **Statistical Morphology:** This involves using statistical models trained on large text corpora to identify morphemes in a word. These models can capture complex patterns and relationships between words, and can be used for languages with less predictable morphological structure.

3. **Neural Morphology:** This involves using deep learning techniques, such as recurrent neural networks (RNNs) or transformers, to identify morphemes in a word. Neural morphological analysis models are trained on large text corpora and can capture complex patterns and relationships between words.

Overall, morphological analysis is an important task in NLP that involves breaking down words into their constituent morphemes and identifying the parts of speech that each morpheme represents. By using techniques such as rule-based morphology, statistical morphology, and neural morphology, NLP systems can gain a deeper understanding of human language and perform various NLP tasks more accurately.

Viva Questions

1. Define Morphology
2. What is the NLP step morphology states?
3. What is morphological and lexical analysis in NLP?
4. Define statistical morphology.
5. Why morphology should be performed in NLP?

Ex.No: 4	N-Grams	Date:

AIM: Write a document in N-Grams

DESCRIPTION:

N-grams are sequences of words or characters in a text. In Natural Language Processing (NLP), N-grams are used as a basic feature representation for text data. An N-gram is a contiguous sequence of N items from a given sample of text or speech.

N-grams are used in NLP for a variety of tasks, including:

1. **Text classification:** N-grams can be used as features to represent the content of a text, which can then be used to train machine learning algorithms for text classification tasks, such as sentiment analysis or spam filtering.
2. **Language modeling:** N-grams can be used to model the probability distribution of words in a language, which can be used to predict the next word in a sentence or to generate text.
3. **Spelling correction:** N-grams can be used to identify commonly occurring word sequences in a language, which can be used to correct spelling errors in a text.
4. **Named entity recognition:** N-grams can be used to identify named entities, such as people, places, and organizations, in a text.

N-grams can be represented as either character-level N-grams or word-level N-grams. Character-level N-grams model the probability distribution of individual characters in a language, while word-level N-grams model the probability distribution of sequences of words.

The choice of the value of N, or the size of the N-gram, can impact the performance of N-gram-based NLP models. Larger values of N can capture more context and meaning in a text, but also require more data to train and may lead to over fitting. Smaller values of N capture fewer contexts, but are less prone to over fitting and can be trained on smaller amounts of data.

Overall, N-grams are a simple and effective representation of text data in NLP, and are used in a variety of NLP tasks, including text classification, language modeling, spelling correction, and named entity recognition.

Viva Questions

1. What is N-grams?
2. What is Ngram used for?
3. How Are N-Grams Classified?
4. Define datasets.
5. What is test data?

Ex.No: 5	N-gram smoothing	Date:

AIM: Write a Document in N-Gram Smoothing

DESCRIPTION:

N-gram smoothing is a technique used in Natural Language Processing (NLP) to address the issue of sparse data in language modeling. The goal of n-gram smoothing is to estimate the probability distribution of words in a language based on a limited sample of text data.

In NLP, language models are used to predict the next word in a sentence, given the previous N-1 words. The probability of each word is estimated based on the frequency of its occurrences in the sample text data. However, when the sample data is limited, many word sequences may not be present in the data, leading to the issue of sparse data. In such cases, the estimated probabilities of these sequences can be 0, making the language model unreliable.

N-gram smoothing is a technique used to address this issue by adjusting the estimated probabilities to account for the possibility of unseen sequences in the sample data. There are several n-gram smoothing techniques, including:

1. **Add-k Smoothing**: The simplest form of n-gram smoothing, add-k smoothing, involves adding a small constant k to each word count in the n-gram model. This ensures that all word sequences have non-zero probability, making the model more robust to unseen sequences.
2. **Good-Turing Smoothing:** Good-Turing smoothing adjusts the estimated probabilities based on the frequency of word occurrences in the sample data. The idea behind Good-Turing smoothing is to estimate the probability of unseen sequences based on the frequency of seen sequences.
3. **Kneser-Ney Smoothing:** Kneser-Ney smoothing is a more sophisticated n-gram smoothing technique that adjusts the estimated

probabilities based on the frequency of word sequences and the frequency of the previous words in the sequence.

N-gram smoothing is an important technique in NLP for improving the reliability and robustness of language models, especially when dealing with limited sample data. The choice of the n-gram smoothing technique depends on the size and quality of the sample data, and the specific requirements of the NLP task.

Viva questions

1. What is N-gram smoothing?
2. Define NLTK.
3. Why do we need smoothing in N-Gram language models?
4. Define Good Turing Smoothing.
5. Define Additive Smoothing