**1. Recursive DFS (Graph from CSV)**

Start

Read graph from CSV file into adjacency list

Initialize empty visited set

Define recursive function DFS(node):

Mark node as visited

Print node

For each neighbor of node:

If neighbor not in visited:

Call DFS(neighbor)

Call DFS(start node)

End

**🔹 2. Non-Recursive DFS (Graph from User)**

Start

Read number of edges

For each edge:

Add to adjacency list (undirected)

Initialize empty visited set

Initialize stack with start node

While stack not empty:

Pop top node

If node not in visited:

Mark as visited

Print node

Push all neighbors onto stack

End

**🔹 3. BFS (Graph from User)**

Start

Read number of edges

For each edge:

Add to adjacency list (undirected)

Initialize empty visited set

Initialize queue with start node

While queue not empty:

Dequeue node

If node not in visited:

Mark as visited

Print node

Enqueue all neighbors

End

**🔹 4–7. Best First Search (Directed/Undirected, Weighted/Unweighted)**

Start

Read graph edges (directed/undirected, weighted/unweighted)

Read heuristic values

Initialize priority queue with (heuristic, start node)

While queue not empty:

Pop node with lowest heuristic

Print node

If node is goal, stop

For each neighbor:

If not visited:

Push (heuristic, neighbor) into queue

End

✅ (Only input type changes.)

**🔹 8–11. A\* Algorithm (Directed/Undirected, CSV/User)**

Start

Read graph edges (weighted) and heuristic values

Initialize priority queue with (cost + heuristic, start node)

Initialize g\_score[start] = 0

While queue not empty:

Pop node with lowest (g + h)

Print node

If node is goal, reconstruct and print path

For each neighbor:

Calculate tentative g = g(current) + edge weight

If better path found:

Update g\_score

Calculate f = g + h

Push (f, neighbor) into queue

End

✅ (Only graph reading method changes.)

**🔹 12. Fuzzy Set Operations (3 sets)**

Start

Define 3 fuzzy sets

For each element:

Union = max(set1, set2, set3)

Intersection = min(set1, set2, set3)

Complement of each set = 1 - membership value

Display union, intersection, and complements

End

**🔹 13. De Morgan's Law (Complement of Union)**

Start

Define 2 fuzzy sets

For each element:

Complement of Union = 1 - max(set1, set2)

Intersection of Complements = min(1 - set1, 1 - set2)

Compare both results

Display if equal

End

**🔹 14. De Morgan's Law (Complement of Intersection)**

Start

Define 2 fuzzy sets

For each element:

Complement of Intersection = 1 - min(set1, set2)

Union of Complements = max(1 - set1, 1 - set2)

Compare both results

Display if equal

End

**🔹 15. Two-Player Game (Computer Wins or Draw)**

Start

Initialize empty board

While game not over:

Player makes move

If player wins, end

Computer uses Min-Max (maximize score) to choose best move

Update board

Display winner or draw

End

**🔹 16. Two-Player Game (Computer Loses or Draw)**

Start

Initialize empty board

While game not over:

Player makes move

If player wins, end

Computer uses Min-Max (minimize its score) to choose worst move

Update board

Display winner or draw

End

**🔹 17. MLP (N inputs, 2 hidden, 1 output — Random weights)**

Start

Initialize random weights and biases

For defined steps:

Perform forward pass (input → hidden1 → hidden2 → output)

No backpropagation

Display final weights, biases, and steps

End

**🔹 18. MLP (4 inputs, 1 hidden, 2 outputs — Random weights)**

Start

Initialize random weights and biases

For defined steps:

Perform forward pass (input → hidden → output)

No backpropagation

Display final weights, biases, and steps

End

**🔹 19. MLP (Sigmoid activation, backpropagation)**

Start

Initialize random weights and biases

For each epoch:

Forward pass using Sigmoid activation

Compute output error

Backpropagate error

Update weights and biases

Display final model

End

**🔹 20. MLP (ReLU activation, backpropagation)**

Start

Initialize random weights and biases

For each epoch:

Forward pass using ReLU activation

Compute output error

Backpropagate error

Update weights and biases

Display final model

End

**🔹 21. MLP (Tanh activation, backpropagation)**

Start

Initialize random weights and biases

For each epoch:

Forward pass using Tanh activation

Compute output error

Backpropagate error

Update weights and biases

Display final model

End

**🔹 22. Text Preprocessing (Cleaning, Tokenizing, Spell Check)**

Start

Read text from file

Remove punctuation, special characters, numbers (regex)

Convert text to lowercase

Tokenize text

Remove stopwords

Correct misspelled words

Display cleaned tokens

End

**🔹 23. Text Preprocessing (Cleaning, Stemming, Lemmatization, Trigrams)**

Start

Read text from file

Remove punctuation, special characters, numbers (regex)

Convert text to lowercase

Tokenize text

Apply stemming

Apply lemmatization

Create 3-word sequences (trigrams)

Display trigrams

End

**🔹 24. One-Hot Encoding (3 Technical Files)**

Start

Read 3 technical text files

Use CountVectorizer(binary=True) to one-hot encode

Fit and transform text data

Display one-hot encoded matrix

End

**🔹 25. Bag of Words (3 Movie Reviews)**

Start

Read 3 movie review files

Use CountVectorizer to create word frequency matrix

Fit and transform text data

Display Bag of Words matrix

End

**🔹 26. TF-IDF (3 Tourist Places)**

Start

Read 3 tourist place files

Use TfidfVectorizer

Fit and transform text data

Display TF-IDF matrix

End