1. Depth First Search (DFS) and Breadth First Search (BFS)

Depth First Search (DFS)

- **Definition**: DFS is a graph traversal algorithm that explores as far as possible along a branch before backtracking.
- **How it works**:
    - Uses recursion or a stack.
    - Starts at the root node and explores each branch completely before moving to the next.
- **Use Cases**:
    - Solving puzzles (e.g., maze problems).
    - Detecting cycles in graphs.
    - Topological sorting.
- **Time Complexity**: O(V + E)
- **Space Complexity**: O(V) due to the stack.

Breadth First Search (BFS)

- **Definition**: BFS explores all neighbors at the current depth before moving deeper.
- **How it works**:
    - Uses a queue.
    - Starts at the root and visits level-by-level.
- **Use Cases**:
    - Shortest path in unweighted graphs.
    - Social network friend suggestions.
- **Time Complexity**: O(V + E)
- **Space Complexity**: O(V)

---

2. A (A-Star) Algorithm*

Overview:

The *A Algorithm** helps find the shortest path from a starting point to a destination. It combines the cost of the current path and a heuristic estimate of the remaining distance.

Key Concepts:

- *A is an informed search algorithm**, which means it uses information (a heuristic) to guide its search for the best path.

- It combines:
    1. **Actual cost so far** (the distance already traveled, denoted g(n)).
    2. **Heuristic estimate** (a guess of how much further you need to go, denoted h(n)).

How it works:

1. **Starting Point**: You start at the beginning, with a list of possible paths.
2. **Cost Function**: A* calculates the total cost to reach each path using the formula:

$f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)$

    - g(n) is the cost you've spent getting to point n.
    - h(n) is an estimate of the cost to get from n to the goal.
    - f(n) is the total estimated cost of the path through n.
3. **Exploration**: The algorithm explores paths with the lowest f(n) first (the one it believes is the cheapest or fastest).
4. **Goal**: It continues exploring the lowest-cost paths until it finds the destination.

Advantages:

- **Optimal**: If the heuristic is good, A* will always find the shortest path.
- **Efficient**: It's faster than blind search methods because it avoids exploring irrelevant paths.

Use Cases:

- **Games**: Pathfinding for characters in a game.
- **GPS Systems**: Finding the shortest driving route.
- **Robotics**: Navigating robots in obstacle-filled environments.

---

3. Dijkstra's Algorithm

Overview:

Dijkstra's Algorithm finds the shortest path from a starting node to all other nodes in a graph. Unlike A*, it only focuses on the actual cost from the start and doesn't use a heuristic (no estimation).

Key Concepts:

- Dijkstra's Algorithm works by progressively finding the node with the shortest distance and exploring its neighbors.
- It is **greedy**: at each step, it picks the node that appears to have the smallest cost to the destination.
- It **only works with non-negative edge weights** (i.e., there can be no negative costs between nodes).

How it works:

1. **Initialization**: Set the distance to the start node as 0, and to all others as infinity.
2. **Exploration**: Visit the closest node and update the distance for each of its neighbors.
3. **Updating Distances**: If traveling through a node offers a shorter path to a neighbor, update its distance.
4. **Repeat**: Continue visiting the closest node until all nodes are processed.

Steps:

1. Set the distance of the start node as 0 and all others to infinity.
2. Visit the node with the smallest tentative distance.
3. For each neighbor, check if going through the current node offers a shorter path and update the distance.
4. Repeat until all nodes are visited.

Advantages:

- **Optimal Solution**: It guarantees the shortest path to every node if all edges have non-negative weights.
- **Simplicity**: Dijkstra's is easier to understand and implement than A* since it doesn't involve heuristics.

Use Cases:

- **Network Routing**: Determining the best path for data packets in a network.
- **GPS**: Finding the shortest route between locations (for example, avoiding traffic).
- **Robotics**: Mapping out the shortest travel path for robots without obstacles.

Key Differences between A* and Dijkstra's:

1. **Heuristic**:
    - **A*** uses a heuristic to prioritize paths, making it faster and more efficient for large problems.
    - **Dijkstra's** doesn't use any guesses, so it explores paths more blindly, which can take longer.
2. **Optimality**:
    - **A*** can find the optimal solution only if the heuristic is admissible (never overestimates).
    - **Dijkstra's** always finds the optimal solution.
3. **Use Cases**:
    - **A*** is better when you have a heuristic available (e.g., in games, or when you know the destination location).

- **Dijkstra's** is simpler and better suited for uniform edge weights or when you don't need a heuristic.

---

4. Constraint Satisfaction Problem (CSP) – N-Queens Problem

Overview:

In a **Constraint Satisfaction Problem (CSP)**, the goal is to assign values to variables under certain constraints (like a puzzle). A popular CSP example is the **N-Queens Problem**.

N-Queens Problem:

- Place **N queens** on an N×N chessboard such that no two queens threaten each other (i.e., no two queens can be on the same row, column, or diagonal).

Techniques:

- **Backtracking**: Try placing queens row by row and backtrack when you encounter a conflict.
- **Branch and Bound**: Skip entire branches of the search if they can't lead to a solution.

Applications:

- Puzzle solving.
- Exam timetable creation.
- Resource scheduling.

---

5. Elementary Chatbot

Overview:

A **chatbot** is a software that simulates conversations with users. It can be used for customer service or to interact with users on websites.

Components:

- **Input Processing**: How the system interprets user input.
- **Pattern Matching**: Identifying keywords and determining the best response.
- **Response Generation**: Formulating a response based on the input.

Types of Chatbots:

- **Rule-based**: Simple, predefined patterns.
- **AI-based**: Uses machine learning and natural language processing (NLP).

Applications:

- Customer support.
- Automated FAQ responders.

- Virtual assistants like Siri, Alexa.

---

6. Expert System – Medical Diagnosis

Overview:

An **expert system** mimics human decision-making. It uses a **knowledge base** (facts and rules) and an **inference engine** (which applies those rules) to make decisions.

Components:

- **Knowledge Base**: A set of rules about a domain.
- **Inference Engine**: The component that applies the rules.
- **User Interface**: How the system interacts with the user.

Example:

In a **Medical Expert System**, the system asks questions about symptoms (e.g., fever, cough) and uses rules to provide a possible diagnosis.

Applications:

- Medical diagnosis.
- Helpdesk support.
- Fault detection in machines.

---

7. Salesforce with Apex Programming

Overview:

Salesforce is a cloud-based CRM platform, and **Apex** is a programming language used to implement backend logic for Salesforce applications.

Key Features:

- **Trigger-driven actions**: Automatically execute logic based on events in Salesforce.
- **SOQL** (Salesforce Object Query Language): Used to query data in Salesforce.
- **Apex Classes and Triggers**: Define business logic and automate processes.

Use Cases:

- Automating tasks (e.g., sending emails).
- Handling business logic for custom applications.

---

8. Mini Project using Salesforce Cloud

Steps to Develop:

1. **Requirement Gathering**: Define the problem (e.g., leave tracker).
2. **Data Modeling**: Create custom objects and fields.
3. **Logic Implementation**: Use Apex triggers, classes, workflows.
4. **UI Creation**: Create Visualforce or Lightning components.
5. **Testing & Deployment**: Test in sandbox and deploy to production.

Advantages:

- Scalable and secure cloud infrastructure.
- Fast development and deployment.
- Customizable according to business needs.