Practical 1 – SQL Queries (DDL, DML, Constraints, Functions, Set Operations)

🎯 Objective

To understand and implement SQL commands for defining, manipulating, and controlling data using DDL, DML, and constraints in relational databases.

🧠 Concept

SQL (Structured Query Language) is a standard language used to store, retrieve, and manage data in relational databases. It consists of several sublanguages:

- **DDL (Data Definition Language)** defines and modifies the structure of database objects such as tables and views.
- **DML (Data Manipulation Language)** handles data operations like inserting, updating, and deleting records.
  Constraints like **PRIMARY KEY**, **FOREIGN KEY**, **CHECK**, and **UNIQUE** maintain data integrity.
  Functions such as SUM(), AVG(), and COUNT() help perform calculations, while **set operations** like UNION and INTERSECT combine results from multiple queries.

🧠 Algorithm / Steps

1. Create a new database and required tables using CREATE TABLE.
2. Define constraints for each column to ensure data accuracy.
3. Insert sample data using INSERT INTO.
4. Retrieve data using SELECT statements with conditions.
5. Apply aggregate functions and grouping.
6. Use set operations (UNION, MINUS, INTERSECT) to combine queries.
7. Commit changes using COMMIT.

❓ Viva Questions and Answers

1. **What is SQL?**
   → SQL is a standard language for creating, retrieving, updating, and managing data in relational databases like MySQL and Oracle.
2. **What is the difference between DDL and DML?**
   → DDL defines and modifies database structure, while DML is used for inserting, updating, or deleting data inside tables.
3. **What are constraints in SQL?**
   → Constraints are rules applied on columns to maintain integrity, e.g., PRIMARY KEY ensures uniqueness, FOREIGN KEY maintains relationships.
4. **What is a primary key?**
   → It uniquely identifies each record in a table and does not allow NULL or duplicate values.

5. **What is a foreign key?**
   → It links two tables together by referencing the primary key of another table.
6. **What are aggregate functions?**
   → They perform calculations on data sets and return a single value, e.g., SUM(), AVG(), and COUNT().
7. **What is a view?**
   → A view is a virtual table based on the result of an SQL query that simplifies data access.
8. **What are set operations?**
   → Set operations like UNION, INTERSECT, and MINUS combine or compare results of multiple queries.
9. **What is normalization?**
   → Normalization organizes data to minimize redundancy and dependency for better efficiency.
10. **What is the use of a transaction in SQL?**
    → A transaction groups multiple operations into a single unit of work to ensure data consistency.

Practical 2 – SQL Queries: Joins, Subqueries, and Views

🎯 Objective

To implement and understand SQL queries using different types of joins, subqueries, and views for relational data retrieval.

❓ Concept

Joins combine rows from multiple tables based on a related column. Common types include:

- **INNER JOIN** (common rows), **LEFT JOIN** (all from left), **RIGHT JOIN** (all from right), and **FULL JOIN** (all records).
  A **subquery** is a query nested inside another query for filtering or computation.
  Views are virtual tables created from one or more tables using SQL queries, allowing simplified data access and security.
  These operations are essential for multi-table relational data handling and efficient query writing.

❓ Algorithm / Steps

1. Create at least two related tables with primary and foreign keys.
2. Insert sufficient sample data.
3. Perform INNER, LEFT, and RIGHT joins to combine records.
4. Write nested queries to extract specific information.
5. Create views using CREATE VIEW.
6. Execute and verify outputs.

❓ Viva Questions and Answers

1. **What is a join in SQL?**
   → A join combines rows from two or more tables based on related columns.
2. **What are different types of joins?**
   → INNER, LEFT, RIGHT, and FULL joins are used to merge data in various ways.
3. **What is an INNER JOIN?**
   → It returns only matching records from both tables based on a common column.
4. **What is a LEFT JOIN?**
   → It retrieves all records from the left table and matching ones from the right, filling unmatched with NULLs.
5. **What is a subquery?**
   → A query inside another query used to filter, calculate, or retrieve data dynamically.
6. **What is a correlated subquery?**
   → A subquery that references the outer query, executing once per row of the outer query.
7. **What is a view?**
   → A virtual table derived from an SQL query that stores no data itself but simplifies data access.
8. **Can a view be updated?**
   → Yes, if it is based on a single table without group functions or joins.

9. **Difference between JOIN and UNION?**
   → JOIN merges columns horizontally, while UNION merges rows vertically.
10. **Why are joins important?**
   → They allow retrieval of meaningful information from related tables efficiently.

Practical 3 – MongoDB Queries Using CRUD Operations

🎯 Objective

To design and execute MongoDB queries using CRUD operations and understand the basic structure of NoSQL databases.

📘 Concept

MongoDB is a NoSQL, document-oriented database that stores data in JSON-like BSON documents. Unlike relational databases, MongoDB does not require predefined schemas.
CRUD stands for **Create, Read, Update, and Delete**, which are the four basic operations performed on data.
MongoDB supports commands like insertOne(), find(), updateOne(), and deleteOne().
This practical helps understand flexible schema design and how MongoDB differs from traditional RDBMS systems in handling unstructured or semi-structured data.

📝 Algorithm / Steps

1. Start the MongoDB server using mongod.
2. Connect to the shell using mongo.
3. Create or switch to a database using use <dbname>.
4. Insert documents using insertOne() or insertMany().
5. Retrieve documents using find().
6. Modify documents using updateOne() or updateMany().
7. Delete using deleteOne() or deleteMany().

❓ Viva Questions and Answers

1. **What is MongoDB?**
   → MongoDB is a NoSQL database that stores data in JSON-like documents, offering flexibility and scalability.
2. **What are CRUD operations?**
   → CRUD stands for Create, Read, Update, and Delete — the four core operations for managing data.
3. **How do we insert data in MongoDB?**
   → By using insertOne() for single documents or insertMany() for multiple entries.
4. **What is the command to read data?**
   → The find() command retrieves documents matching the query conditions.
5. **How is data updated in MongoDB?**
   → The updateOne() or updateMany() methods modify existing documents based on criteria.
6. **How is data deleted?**
   → Using deleteOne() or deleteMany() methods to remove specific documents.
7. **What is a collection?**
   → A collection is a group of MongoDB documents, similar to a table in SQL.

8. **What is a document?**
   → A document is a single record stored in BSON (Binary JSON) format.
9. **Difference between MongoDB and SQL?**
   → MongoDB is schema-less and document-based, while SQL is schema-defined and table-based.
10. **What is indexing in MongoDB?**
   → Indexing improves query performance by allowing faster document retrieval.

Practical 4 – PL/SQL Code Block: Control Structure and Exception Handling

🎯 Objective

To implement a PL/SQL code block that uses control structures and exception handling to manage library fine calculation based on the number of days a book is issued.

🧠 Concept

PL/SQL (Procedural Language SQL) extends SQL by adding programming constructs like variables, loops, and conditions. It allows embedding business logic inside the database itself for efficiency.
Control structures such as IF–THEN–ELSE and loops (FOR, WHILE) are used to process data conditionally or repeatedly.
Exception handling ensures that runtime errors like division by zero or missing records are caught and handled gracefully.
In this practical, logic is implemented to calculate fines for delayed book returns and store them in a table using conditions and exceptions.

🧩 Algorithm / Steps

1. Create tables Borrower (Roll_no, Name, Date_of_Issue, Name_of_Book, Status) and Fine (Roll_no, Date, Amt).
2. Accept Roll number and Book name from the user.
3. Calculate the number of days delayed using system date and Date_of_Issue.
4. Apply conditions:
    - If 15 ≤ days ≤ 30 → Fine = ₹5/day
    - If days > 30 → Fine = ₹5/day for days above 30
    - If days < 15 → No fine
5. Insert fine record in the Fine table and update the Borrower table.
6. Use exception handling for missing data or invalid input.

❓ Viva Questions and Answers

1. **What is PL/SQL and how is it different from SQL?**
   → PL/SQL adds procedural features like loops, variables, and exception handling to SQL, making it more powerful for automation.
2. **What are control structures in PL/SQL?**
   → Control structures manage the flow of execution and include IF–THEN–ELSE, FOR, WHILE, and CASE statements.
3. **Why is exception handling important?**
   → It prevents abrupt termination by catching and managing runtime errors using the EXCEPTION block.

4. **What are predefined exceptions?**
   → Oracle provides exceptions like NO_DATA_FOUND, ZERO_DIVIDE, and TOO_MANY_ROWS for common runtime issues.
5. **Can you define custom exceptions?**
   → Yes, user-defined exceptions can be declared in the DECLARE section and raised using the RAISE statement.
6. **What is the structure of a PL/SQL block?**
   → It consists of four sections: DECLARE (optional), BEGIN, EXCEPTION (optional), and END.
7. **What is the purpose of the BEGIN-END block?**
   → It marks the executable part of the PL/SQL program where SQL and logic statements are written.
8. **What happens if no exception is handled?**
   → The program terminates and Oracle displays an unhandled exception error message.
9. **Can multiple exceptions be handled in one block?**
   → Yes, multiple WHEN clauses can be used in the EXCEPTION block to handle different error types.
10. **What are the benefits of using PL/SQL blocks in databases?**
    → They reduce network traffic, improve performance, ensure data integrity, and provide modular programming capabilities.

Practical 6 – Cursors (Implicit, Explicit, FOR Loop, Parameterized Cursor)

🎯 Objective

To develop a PL/SQL program demonstrating all types of cursors including implicit, explicit, cursor FOR loop, and parameterized cursors.

🧠 Concept

A **cursor** is a database pointer that allows row-by-row processing of query results in PL/SQL.

- **Implicit cursors** are automatically created by Oracle for single SQL statements.
- **Explicit cursors** are declared by the programmer for multi-row queries.
- **Cursor FOR loops** simplify fetching by automatically handling open, fetch, and close operations.
- **Parameterized cursors** accept arguments, allowing dynamic queries during execution.
  This practical builds understanding of fetching and processing multiple records efficiently using cursors.

🧠 Algorithm / Steps

1. Declare an explicit or parameterized cursor using the CURSOR keyword.
2. Open the cursor using OPEN cursor_name;.
3. Fetch rows into variables using FETCH cursor_name INTO variable;.
4. Process records inside a loop.
5. Close the cursor with CLOSE cursor_name;.
6. Use a cursor FOR loop for simpler iteration.

❓ Viva Questions and Answers

1. **What is a cursor in PL/SQL?**
   → A cursor is a pointer to a result set that allows processing query results row by row.
2. **What are the types of cursors?**
   → Implicit, Explicit, Cursor FOR Loop, and Parameterized cursors.
3. **What are cursor attributes?**
   → Attributes like %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN give information about cursor status.
4. **What is an implicit cursor?**
   → It is automatically created for single SQL statements like INSERT or UPDATE.
5. **What is an explicit cursor?**
   → It is defined manually for multi-row queries requiring custom row-by-row processing.
6. **What is a parameterized cursor?**
   → A cursor that accepts parameters to filter query results dynamically at runtime.
7. **How does a cursor FOR loop work?**
   → It automatically opens, fetches, and closes the cursor, simplifying code.

8. **Why must cursors be closed?**
   → To release memory and system resources once the result set is processed.
9. **Can cursors be reused?**
   → Yes, cursors can be reopened and executed multiple times with different parameters.
10. **Where are cursors used?**
   → They are used in reporting, payroll, and applications requiring record-by-record data processing.

Practical 7 – Database Connectivity (MySQL/Oracle with Python/Java)

🎯 Objective

To write a program that establishes database connectivity with MySQL or Oracle using a front-end language like Python or Java and perform database navigation operations (add, delete, edit).

❓ Concept

Database connectivity allows applications to communicate with databases using APIs such as **JDBC (Java Database Connectivity)** or **ODBC (Open Database Connectivity)**.
In Python, libraries like mysql.connector or PyODBC provide similar functionality. These APIs enable CRUD operations: **Create**, **Read**, **Update**, and **Delete**.
This practical demonstrates how to connect an application with a database, execute SQL commands, and handle navigation operations efficiently.
Proper connection management and exception handling ensure stability and security in real-world systems.

❓ Algorithm / Steps

1. Install required libraries or drivers (e.g., MySQL Connector, JDBC).
2. Establish a database connection using connection strings.
3. Create a cursor or statement object to execute queries.
4. Implement operations like INSERT, UPDATE, DELETE, and SELECT.
5. Display the data and navigate between records.
6. Handle exceptions for invalid queries or connection issues.
7. Close the cursor and database connection.

❓ Viva Questions and Answers

1. **What is database connectivity?**
   → It is the process of linking an application to a database so data can be stored, retrieved, and updated through code.
2. **What is JDBC?**
   → JDBC is a Java API that allows Java programs to interact with databases by executing SQL statements.
3. **What are the main steps in JDBC connectivity?**
   → Load driver, establish connection, create statement, execute queries, process results, and close connection.
4. **What is ODBC?**
   → ODBC (Open Database Connectivity) is a standard API that enables communication between databases and various programming languages.
5. **What are CRUD operations?**
   → CRUD stands for Create, Read, Update, and Delete — the four primary database operations.

6. **How do you connect to a MySQL database in Python?**
   → Using the mysql.connector.connect() method with parameters like host, user, password, and database name.
7. **What is a connection string?**
   → It's a string containing details like database type, server name, user credentials, and port number used to establish a connection.
8. **Why is exception handling important in database programming?**
   → It helps manage errors such as connection failures or invalid queries without crashing the program.
9. **Why should database connections be closed?**
   → To release system resources and prevent data corruption or connection leaks.
10. **What is a ResultSet in JDBC?**
    → A ResultSet is an object that stores data retrieved from a database and allows navigation through records.

Practical 8 – Breadth First Search (BFS) and Depth First Search (DFS)

🎯 Objective

To implement and understand the working of Breadth First Search (BFS) and Depth First Search (DFS) algorithms for traversing graphs efficiently.

🧠 Concept

Breadth First Search (BFS) and Depth First Search (DFS) are two fundamental graph traversal techniques used to visit every node in a graph systematically.
BFS explores nodes level by level, ensuring that all nodes at a given depth are visited before moving deeper. It uses a **queue**, making it ideal for finding the shortest path in unweighted graphs.
DFS, on the other hand, explores one branch completely before backtracking. It uses a **stack** (or recursion) and is particularly useful for exploring all possible paths, detecting cycles, or checking connectivity.
Both algorithms are widely used in AI, computer networks, and pathfinding applications like maze solving and social network analysis.

🧠 Algorithm

BFS Algorithm:

1. Start from the initial node and mark it as visited.
2. Enqueue the starting node.
3. While the queue is not empty, dequeue a node and explore its unvisited neighbors.
4. Mark each neighbor as visited and enqueue them.
5. Repeat until all reachable nodes are visited.

DFS Algorithm:

1. Start from the initial node and mark it as visited.
2. Visit an unvisited neighbor recursively or push it onto a stack.
3. Continue the process until all nodes are visited.
4. Backtrack when there are no more unvisited neighbors.

❓ Viva Questions and Answers

1. **What is BFS and where is it used?**
   → BFS is a graph traversal technique that visits nodes level by level using a queue. It is widely used in shortest path finding, social networking, and AI search problems.
2. **What data structure is used in BFS?**
   → BFS uses a **queue**, which helps in exploring all neighbors before moving to the next level. The first inserted node is processed first (FIFO order).

3. **What is DFS and how does it differ from BFS?**
   → DFS explores as deep as possible before backtracking, using recursion or a stack. BFS is level-based, while DFS follows a single path to depth.
4. **Which algorithm guarantees the shortest path?**
   → BFS guarantees the shortest path in an **unweighted graph**, as it explores nodes in increasing order of depth.
5. **What data structure is used in DFS?**
   → DFS typically uses a **stack** or recursion stack to keep track of the nodes being visited and backtracked.
6. **What are the time and space complexities of BFS and DFS?**
   → Both algorithms have **O(V + E)** complexity, where V = vertices and E = edges. BFS may consume more memory due to queue storage.
7. **Can DFS be implemented iteratively?**
   → Yes, DFS can be implemented iteratively using an explicit stack instead of recursion. This is useful when recursion depth is limited.
8. **Where is DFS mainly used?**
   → DFS is used in tasks like cycle detection, topological sorting, and connectivity testing in graphs.
9. **Where is BFS mainly used?**
   → BFS is mainly used in shortest path algorithms, web crawling, and peer-to-peer network traversal.
10. **What is backtracking in DFS?**
    → Backtracking occurs when the algorithm returns to a previous node after exploring all paths from the current node. This helps in exhaustive exploration.

Practical 9 – A (A-Star) Search Algorithm*

🎯 Objective

To implement and understand the working of the A* search algorithm for pathfinding using heuristic-based informed search.

❓ Concept

The A* algorithm is an **informed search strategy** that uses both actual cost and heuristic estimation to find the optimal path between two nodes.
It combines the advantages of **Dijkstra's algorithm** (which guarantees optimality) and **Greedy Best-First Search** (which uses heuristics for speed).
The evaluation function is defined as **f(n) = g(n) + h(n)**, where:

- g(n) is the cost from the start node to the current node, and
- h(n) is the estimated cost from the current node to the goal (heuristic).
  A* is widely used in robotics, GPS navigation, video games, and route optimization systems due to its accuracy and efficiency.

❓ Algorithm

1. Initialize two lists: **Open List** (nodes to be explored) and **Closed List** (visited nodes).
2. Add the starting node to the Open List.
3. Select the node with the smallest f(n) = g(n) + h(n).
4. Move it to the Closed List and expand its neighbors.
5. For each neighbor, calculate g(n), h(n), and f(n).
6. If a better path is found, update parent and cost values.
7. Stop when the goal node is selected and reconstruct the path.

❓ Viva Questions and Answers

1. *What is the A search algorithm?**
   → A* is an informed search algorithm that uses both path cost and heuristic values to find the optimal path efficiently between two nodes.
2. *What are g(n), h(n), and f(n) in A?**
   → g(n) is the actual path cost, h(n) is the estimated cost (heuristic), and f(n) = g(n) + h(n) is the total estimated cost from start to goal.
3. **What is an admissible heuristic?**
   → An admissible heuristic is one that never overestimates the actual cost to reach the goal, ensuring the A* algorithm remains optimal.
4. **Give examples of heuristic functions.**
   → Common heuristics include **Manhattan distance** (for grids) and **Euclidean distance** (for continuous spaces).
5. *When does A become Dijkstra's algorithm?**
   → When h(n) = 0 for all nodes, A* behaves exactly like Dijkstra's algorithm.

6. *When does A become Greedy Best-First Search?**
   → When g(n) = 0, A* becomes Greedy Best-First Search, focusing only on heuristic values.
7. *Is A always optimal?**
   → Yes, A* is optimal when the heuristic used is admissible and consistent, meaning it respects triangle inequality.
8. *What are the advantages of A?**
   → A* finds the shortest path efficiently by balancing exploration (g) and exploitation (h), making it fast and accurate.
9. *Where is A used?**
   → It is used in pathfinding for maps, GPS navigation, robotics, and game AI for real-time movement planning.
10. *What is the time complexity of A?**
    → The worst-case time complexity is exponential, but with good heuristics, it performs much faster in practice.

Practical 10 – Alpha-Beta Pruning

🎯 Objective

To implement Alpha-Beta pruning and understand how it optimizes the Minimax algorithm for decision-making in two-player games.

🧠 Concept

Alpha-Beta Pruning is an optimization of the **Minimax algorithm**, which is used in adversarial search for decision-making in two-player games.
In Minimax, all possible game states are evaluated, which becomes computationally expensive for large trees. Alpha-Beta pruning eliminates branches that cannot influence the final outcome, thus reducing computations.

- **Alpha (α)** represents the maximum lower bound (best value for the maximizer).
- **Beta (β)** represents the minimum upper bound (best value for the minimizer).
  If α ≥ β, that branch is pruned since further exploration cannot improve the result. This makes Alpha-Beta much faster while producing the same output as Minimax.

🧠 Algorithm

1. Start with the root node representing the current game state.
2. Apply the Minimax algorithm recursively to explore possible moves.
3. Maintain α and β values during traversal.
4. If α ≥ β, prune the branch (stop exploring it).
5. Continue until all relevant nodes are explored or pruned.
6. Return the best move for the current player.

❓ Viva Questions and Answers

1. **What is the Minimax algorithm?**
   → It is a decision-making algorithm used in two-player games to minimize the possible loss in a worst-case scenario by evaluating all possible moves.
2. **What is Alpha-Beta pruning?**
   → Alpha-Beta pruning is a technique that reduces the number of nodes evaluated in the Minimax algorithm by pruning unnecessary branches.
3. **What are α and β values?**
   → α is the best value that the maximizer can guarantee, and β is the best value the minimizer can guarantee during traversal.
4. **When does pruning occur?**
   → Pruning occurs when α becomes greater than or equal to β, as further exploration cannot affect the final decision.
5. **Does Alpha-Beta pruning affect the result?**
   → No, pruning only reduces computation; the final Minimax value remains the same as without pruning.

6. **What is the advantage of Alpha-Beta pruning?**
   → It significantly improves efficiency, allowing deeper search in the same time compared to plain Minimax.
7. **What are typical applications of this algorithm?**
   → It is used in AI game agents like chess, tic-tac-toe, and checkers to select optimal moves.
8. **What is the time complexity of Minimax and Alpha-Beta?**
   → Minimax is O(b^d), while Alpha-Beta reduces it to nearly O(b^(d/2)) in the best case.
9. **What is a terminal node?**
   → A terminal node represents a final state in the game where the outcome (win, loss, draw) is known.
10. **What is a heuristic evaluation function?**
    → It is a scoring function that estimates the desirability of a game state when full search to terminal nodes isn't feasible.

Practical 11 – Constraint Satisfaction Problem (CSP)

🎯 Objective

To implement and understand the working of Constraint Satisfaction Problems (CSP) using backtracking or constraint propagation techniques.

❓ Concept

A **Constraint Satisfaction Problem (CSP)** is a type of problem defined by a set of variables, each with a domain of possible values, and a set of constraints that restrict which value combinations are allowed. The goal is to assign values to all variables while satisfying every constraint. For example, the **N-Queens**, **Sudoku**, and **Map Coloring** problems are classical CSPs.
CSPs are commonly solved using **backtracking search**, where solutions are built incrementally, abandoning paths that violate constraints. Advanced methods like **forward checking** and **arc consistency** improve performance by eliminating inconsistent choices early.
These methods are widely used in AI for scheduling, planning, and resource allocation problems.

❓ Algorithm

1. Define variables, domains, and constraints for the problem.
2. Select an unassigned variable and assign it a value from its domain.
3. Check whether this assignment satisfies all constraints.
4. If consistent, continue with the next variable; if not, backtrack.
5. Apply forward checking or propagation to eliminate invalid options.
6. Continue until all variables are assigned valid values or no solution exists.

❓ Viva Questions and Answers

1. **What is a Constraint Satisfaction Problem?**
   → A CSP is a problem consisting of variables, their domains, and constraints that restrict the values these variables can take. The objective is to find an assignment satisfying all constraints.
2. **What are examples of CSPs?**
   → Common examples include **N-Queens**, **Sudoku**, **Map Coloring**, and **Scheduling** problems, where specific conditions must be satisfied simultaneously.
3. **What is backtracking in CSP?**
   → Backtracking is a systematic search technique where partial solutions violating constraints are abandoned, and the algorithm returns to previous steps to explore alternatives.
4. **What is forward checking?**
   → Forward checking eliminates inconsistent values from the domains of unassigned variables after each assignment, reducing future conflicts.
5. **What is constraint propagation?**
   → Constraint propagation spreads the effect of constraints to reduce domain sizes of other variables, helping to detect inconsistencies early.

6. **What is arc consistency?**
   → A variable is arc consistent if every value in its domain satisfies the binary constraint with some value in the domain of a connected variable.
7. **What is the domain of a variable?**
   → It is the set of all possible values that a variable can take while solving the CSP.
8. **What are the applications of CSP?**
   → CSPs are used in scheduling tasks, map coloring, exam timetabling, and resource allocation problems.
9. **What is the difference between backtracking and forward checking?**
   → Backtracking checks constraint violations after assignment, while forward checking predicts future conflicts before assigning.
10. **Why are CSPs important in AI?**
    → They provide a structured way to model and solve complex problems involving interdependent variables efficiently.

Practical 12 – Greedy Algorithm Krushkal's and Prim's

🎯 Objective

To understand and implement Greedy algorithms for solving optimization problems where local choices lead to global solutions.

❓ Concept

A **Greedy Algorithm** builds a solution step-by-step by selecting the best available option at each step, aiming for a globally optimal solution.
It is applicable when the problem exhibits the **greedy choice property** and **optimal substructure**, meaning locally optimal choices lead to global optimality.
Examples include **Prim's Algorithm**, **Kruskal's Algorithm**, and **Dijkstra's Shortest Path Algorithm**.
Greedy algorithms are simple and efficient but do not always guarantee optimal solutions, except for specific problems like Minimum Spanning Tree or Huffman Coding.

❓ Algorithm (General Structure)

1. Identify the optimal substructure and greedy choice property.
2. Initialize an empty solution set.
3. Choose the best local option according to the selection criterion.
4. Add it to the solution if feasible.
5. Repeat until the problem is completely solved.
6. Return the constructed solution.

❓ Viva Questions and Answers

1. **What is a Greedy algorithm?**
   → A Greedy algorithm makes the best possible choice at each step without reconsidering previous decisions, aiming for an overall optimal solution.
2. **What are examples of Greedy algorithms?**
   → Common examples include **Prim's** and **Kruskal's** algorithms for MST, **Dijkstra's algorithm** for shortest paths, and **Huffman Coding** for data compression.
3. **What is the greedy choice property?**
   → It means a globally optimal solution can be achieved by making a series of locally optimal decisions.
4. **What is optimal substructure?**
   → It means that an optimal solution to a problem contains optimal solutions to its subproblems.
5. **When does the Greedy approach fail?**
   → It fails when the locally optimal decisions do not lead to a globally optimal solution, as in the 0/1 Knapsack problem.
6. **Difference between Greedy and Dynamic Programming?**
   → Greedy makes one-pass local decisions, while DP explores all possibilities and uses past results to build global optimal solutions.

7. **What is the complexity of Prim's and Kruskal's algorithms?**
   → Both have a time complexity of **O(E log V)**, where E is edges and V is vertices.
8. **What is Dijkstra's Algorithm used for?**
   → It finds the shortest path from a single source to all other vertices in a weighted graph with non-negative edges.
9. **What is Huffman coding?**
   → Huffman Coding is a Greedy algorithm used for optimal data compression based on symbol frequencies.
10. **Advantages and disadvantages of Greedy algorithms?**
    → They are simple, fast, and require less memory, but may produce suboptimal results when the problem lacks the greedy property.

Practical 13 – Dijkstra's Algorithm (Shortest Path Algorithm)

🎯 Objective

To understand and implement **Dijkstra's algorithm** for finding the shortest path between nodes in a weighted graph.

❓ Concept

Dijkstra's algorithm is one of the most widely used **greedy algorithms** in graph theory. It finds the **shortest path** from a given source node to all other nodes in a graph with **non-negative edge weights**.
The algorithm works by progressively exploring the closest vertex that has not yet been visited, updating the distance of all its adjacent vertices if a shorter path is found. This process continues until all vertices are visited.
The main principle behind Dijkstra's algorithm is **greediness**, meaning it always selects the next node with the **minimum tentative distance** from the source.
It is commonly used in **network routing, GPS navigation, and transport optimization systems**, where finding the most efficient route is essential.

❓ Algorithm / Steps

1. Start with a weighted graph and select the source vertex.
2. Assign distance = 0 to the source and ∞ (infinity) to all other vertices.
3. Mark all vertices as unvisited.
4. Select the unvisited vertex with the smallest distance value — mark it as the current vertex.
5. Update the distance of all adjacent vertices by comparing:

   `new_distance = distance[current] + edge_weight(current, neighbor)`

   If `new_distance` is smaller, update it.

6. Mark the current vertex as visited (it will not be checked again).
7. Repeat steps 4–6 until all vertices are visited.
8. The final distance array now contains the shortest distance from the source to every other vertex.

❓ Example

Consider a graph with vertices A, B, C, D, E and the following edges with weights:

A–B: 4, A–C: 2, B–C: 5, B–D: 10, C–E: 3, E–D: 4

- Start from A → distance[A] = 0
- Initially: B = 4, C = 2, others = ∞
- Next pick C (minimum 2): update E = 5
- Next pick B (4): D = 14 (not shortest yet)
- Next pick E (5): D = 9 → updated
- Shortest distances:

- A:0, B:4, C:2, D:9, E:5

⏱ Time Complexity

- Using simple arrays → $O(V^2)$
- Using priority queue (min-heap) → $O((V + E) \log V)$

❓ Viva Questions and Answers

1. **What is Dijkstra's algorithm used for?**
   → It is used to find the shortest path from a single source to all other nodes in a weighted graph with non-negative edges.
2. **Why is it called a greedy algorithm?**
   → Because at each step, it selects the vertex with the smallest known distance — a local optimum that leads to the global solution.
3. **Can Dijkstra's algorithm work with negative edge weights?**
   → No, it fails with negative weights because the algorithm assumes once a node's shortest distance is found, it won't change.
4. **What data structures are used in Dijkstra's algorithm?**
   → Typically, an adjacency matrix or adjacency list for the graph and a priority queue (min-heap) for selecting the next vertex.
5. **What is the time complexity of Dijkstra's algorithm?**
   → $O(V^2)$ using arrays and $O((V + E) \log V)$ using a priority queue (heap).
6. **What is the space complexity?**
   → $O(V)$ for storing distances and visited nodes.
7. **What is the main difference between Dijkstra's and Bellman-Ford algorithms?**
   → Dijkstra's cannot handle negative edges, while Bellman-Ford can handle them but is slower.
8. **In which real-world applications is Dijkstra's algorithm used?**
   → It's used in GPS navigation, internet routing protocols, and shortest path problems in logistics and robotics.
9. **What is the role of a priority queue in Dijkstra's algorithm?**
   → It helps efficiently select the next vertex with the minimum tentative distance in $O(\log V)$ time.
10. **When does Dijkstra's algorithm stop?**
    → When all vertices have been visited and the shortest distance to every node from the source has been determined.

Practical 14 – Expert System (Mini Project)

🎯 Objective

To design and implement an Expert System that emulates the decision-making ability of a human expert using rule-based reasoning.

❓ Concept

An **Expert System** is an AI application that uses a **knowledge base** and an **inference engine** to solve complex problems requiring human expertise.
The knowledge base contains domain-specific facts and rules (IF–THEN form), while the inference engine applies logical reasoning to draw conclusions.
Two main reasoning techniques are used: **Forward Chaining** (data-driven) and **Backward Chaining** (goal-driven).
Expert systems are widely applied in fields like medical diagnosis, finance, and troubleshooting due to their ability to store and reuse expert knowledge efficiently.

❓ Algorithm

1. Collect expert knowledge and represent it as IF–THEN rules.
2. Initialize the knowledge base and inference engine.
3. Accept facts or symptoms from the user.
4. Apply inference rules to derive conclusions.
5. Continue reasoning until no new conclusions are possible.
6. Display the result or decision to the user.

❓ Viva Questions and Answers

1. **What is an Expert System?**
   → It is an AI system that mimics human experts by using knowledge and reasoning to solve problems in a specific domain.
2. **What are the components of an Expert System?**
   → The main components are the **Knowledge Base**, **Inference Engine**, and **User Interface**, which together process and present knowledge.
3. **What is the role of the Knowledge Base?**
   → It stores domain-specific facts, data, and decision rules provided by human experts.
4. **What is the function of the Inference Engine?**
   → It applies logical reasoning on the knowledge base to infer new facts or make conclusions from existing information.
5. **What is forward chaining?**
   → It is a data-driven reasoning method that starts from known facts and applies rules to reach conclusions.

6. **What is backward chaining?**
   → It is a goal-driven approach that starts from a hypothesis and works backward to determine supporting facts.
7. **What are examples of Expert Systems?**
   → MYCIN for medical diagnosis, DENDRAL for chemical analysis, and CLIPS for rule-based reasoning are common examples.
8. **What are the advantages of Expert Systems?**
   → They provide consistent, fast, and reliable decisions, reduce human error, and preserve expert knowledge.
9. **What are limitations of Expert Systems?**
   → They cannot learn automatically, lack creativity, and are expensive to develop and maintain.
10. **What are the applications of Expert Systems?**
    → They are used in medical diagnosis, financial analysis, equipment troubleshooting, and intelligent tutoring systems.

📄 **Prepared by: Karan Salunkhe ☺**