Dynamic Programming

1. Implement Fibonacci series using dynamic programming.
2. Write a program to find the factorial of a number using dynamic programming.
3. Solve the coin change problem with dynamic programming.
4. Implement a dynamic programming solution for the rod cutting problem.
5. Solve the 0-1 knapsack problem using dynamic programming.
6. Write a dynamic programming solution for the longest common subsequence problem.
7. Implement a program for the longest increasing subsequence.
8. Solve the matrix chain multiplication problem using dynamic programming.
9. Implement the minimum cost path problem in a 2D matrix.
10. Write a program for the word break problem using dynamic programming.

Graph Algorithms

1. Implement graph representation using an adjacency matrix.
2. Implement graph representation using an adjacency list.
3. Write a program to detect a cycle in an undirected graph.
4. Implement a method to detect a cycle in a directed graph.
5. Write a function to check if a graph is bipartite.
6. Implement a graph-based solution for the river crossing problem.
7. Write a program to find all bridges in a given graph.
8. Implement an algorithm to find articulation points in a graph.
9. Write a function to perform topological sorting in a directed graph.
10. Implement a method to find strongly connected components in a directed graph.

Graph Search Algorithms

1. Write a BFS algorithm for a graph.
2. Implement a DFS algorithm for a graph.
3. Write a program to find the number of islands using DFS in a matrix.
4. Implement a solution to solve the maze problem using BFS.
5. Write a program to check for a path between two nodes in a graph using BFS.
6. Implement an algorithm to find the level of each node in a graph using BFS.
7. Write a function to print all paths between two vertices in a graph using DFS.
8. Implement a method to find leaf nodes in a graph using DFS.
9. Write a program to detect a cycle in a graph using DFS.
10. Implement a DFS-based solution to find the connected components of a graph.

Breadth-First Search (BFS)

1. Implement a BFS traversal in a binary tree.
2. Write a program to find the shortest path in an unweighted graph using BFS.
3. Implement a BFS solution to find the minimum number of edges between two nodes.
4. Write a program for level order traversal in a binary tree using BFS.
5. Implement a method to find the maximum width of a binary tree using BFS.

Depth-First Search (DFS)

1. Implement a DFS traversal in a binary tree.
2. Write a program to find all root-to-leaf paths in a binary tree using DFS.
3. Implement a DFS solution to check if a tree is a valid binary search tree.
4. Write a program to find the diameter of a tree using DFS.
5. Implement a method to find if a graph is connected using DFS.

Minimum Path Search

1. Implement an algorithm to find the shortest path in a grid.
2. Write a program to find the minimum path sum in a 2D matrix.
3. Implement a solution to find the shortest path in a maze.

4. Write a program to find the minimum number of steps from the start to the end of a grid.
5. Implement a method to find the shortest path in a weighted graph.

Dijkstra Algorithm

1. Implement Dijkstra's algorithm to find the shortest path in a graph.
2. Write a program to find the shortest path from a single source to all vertices.
3. Implement a solution to find the shortest path in a grid using Dijkstra's algorithm.
4. Write a program to enhance Dijkstra's algorithm to handle negative weights.
5. Implement a method to find the shortest path and its distance using Dijkstra's algorithm.

Search for the Minimum Leaf Tree

1. Implement an algorithm to find the minimum leaf from the root in a binary tree.
2. Write a program to find the minimum leaf sum in a binary tree.
3. Implement a solution to find the minimum leaf path in a binary tree.
4. Write a program to find the closest leaf to a given node in a binary tree.
5. Implement a method to find the number of leaves in a minimum height tree.

Kruskal Algorithm

1. Implement Kruskal's algorithm to find the minimum spanning tree of a graph.
2. Write a program to find the total weight of a minimum spanning tree using Kruskal's algorithm.
3. Implement a solution to optimize Kruskal's algorithm for a dense graph.
4. Write a program to detect cycles during the execution of Kruskal's algorithm.
5. Implement a method to choose the best edge in each iteration of Kruskal's algorithm.

Advanced Dynamic Programming

1. Solve the palindrome partitioning problem using dynamic programming.
2. Implement a program for the edit distance problem using dynamic programming.
3. Write a dynamic programming solution for the subset sum problem.
4. Solve the partition problem using dynamic programming.
5. Implement a dynamic programming solution for the box stacking problem.

Advanced Graph Algorithms

1. Write an algorithm for graph coloring using backtracking.
2. Implement the Ford-Fulkerson algorithm for the maximum flow problem.
3. Write a program for Hamiltonian cycle detection in a graph.
4. Implement the Tarjan’s algorithm for strongly connected components.
5. Write a function to find the shortest path in a directed acyclic graph (DAG).

Complex Graph Search Problems

1. Implement a program to find the shortest path in a 3D maze.
2. Write an algorithm to solve the sliding puzzle game using BFS.
3. Implement a solution for the water jug problem using graph search.
4. Write a program to implement the Knight's tour problem using DFS.
5. Implement a solution to solve the N-Queens problem using graph search.

BFS and DFS Applications

1. Write a BFS-based solution for the snake and ladder game.
2. Implement a DFS-based solution to generate mazes.
3. Write a program to find the number of connected components in an undirected graph using BFS.
4. Implement a method to print all possible paths in a grid using DFS.
5. Write a program to clone a graph using BFS.

Dijkstra and Kruskal Applications

1. Implement Dijkstra's algorithm in a network routing simulation.
2. Write a program to find the most reliable path in a network using Dijkstra's algorithm.

3. Implement Kruskal's algorithm to plan the layout of a network.
4. Write a solution to optimize network bandwidth using Kruskal's algorithm.
5. Implement a program to find the shortest path in a time-dependent graph using Dijkstra's algorithm.

Mixed and Miscellaneous Problems

1. Write a program to merge two binary trees.
2. Implement a solution to find the number of islands in a matrix using DFS and BFS.
3. Solve the Sudoku puzzle using graph algorithms.
4. Implement a method to find the meeting point in a maze using BFS and DFS.
5. Write a program to simulate packet routing in a network using graph algorithms.

Practical Applications

1. Develop a graph-based recommendation system for e-commerce.
2. Implement a program to optimize delivery routes using Dijkstra's algorithm.
3. Write a network congestion simulation using graph algorithms.
4. Create a program for airport connectivity analysis using Kruskal's algorithm.
5. Implement a graph-based solution for social network analysis.

Challenges and Puzzles

1. Solve the Rubik's cube problem using graph algorithms.
2. Implement a program to solve the bridge and torch problem using graph search.
3. Write a solution for the escape room puzzle using graph search algorithms.
4. Develop a method to solve complex logic puzzles using graph algorithms.
5. Create a program to simulate traffic flow in a city using Dijkstra's algorithm and BFS/DFS.