**Task 1**

**Data Structure**

The code defines the coordinates of each point (A through I) on a 3x3 grid. Each key corresponds to a point, and its value is an object containing `x` and `y` coordinates.

**canConnect() Function**

* used to determines if two points, `a` and `b`, can be directly connected without violating the rules of pattern formation.
* Coordinate Calculation: Retrieves the coordinates for points `a` and `b`.
* Midpoint Calculation: Computes the midpoint's coordinates. This midpoint is used to check if a straight line between two points is valid only when the midpoint has already been visited (used in the path).

Connection Rules:

* Straight Line: Checks if the points are in a direct horizontal, vertical, or diagonal line.
* Midpoint Visit: Ensures that if a midpoint exists, it must either not be between the points or already visited.

**searchPatterns() Function**

* It implements a recursive DFS to explore all paths starting from a given point.
* Base Case: Stops if the path reaches the specified depth and if the current point is the target point, adding the path to the list of valid patterns.
* Recursive Exploration: Iteratively attempts to move to every other point not yet visited that can be connected directly from the current point. This recursion expands the search tree depth-first, exploring all possible paths.

**listPatterns() Function**

* Initializes the process by starting from a given first point and trying to reach a second point with paths of a specific depth.
* Path Expansion: Once initial paths to the second point are found, it continues from the second point to the third, extending the paths found in the first phase.
* Pattern Validation: Ensures that only paths that correctly go from the first to the second and then to the third point are considered valid.

**Methodology: Depth-First Search (DFS)**

* DFS: The algorithm uses depth-first search, a common technique for exploring all possible paths through a graph or grid. DFS is particularly useful here as it allows exploring all potential combinations of moves (paths) from a starting point to an endpoint.
* Backtracking: The algorithm inherently uses backtracking, where it goes as deep as possible along a branch and then backtracks to explore other branches once it reaches a dead end or a solution is found for a branch.

Task 2

**Objective 1**

For this part I only change the values of the variables below

POP\_SIZE = 500

SELECTION\_SIZE = 100

ELITE\_SIZE = 25

NUM\_GENS = 500

MUTATION\_RATE = 0.1

CROSSOVER\_RATE = 0.9

These changes results in a total cost of RM 118.06 which is lesser than RM 120   
A screenshot of a computer program

Description automatically generated

Adjusting these parameters directly impacts the practical outcomes by enhancing its ability to efficiently search the solution space for the lowest-cost routes while respecting all constraints. The careful tuning of these parameters is essential to help the algorithm towards the specific needs of this vehicle routing problem.