## **🔹 Algorithm: Backtracking 15-Puzzle Solver (with DFS)**

1. **Input the board**
   * Read a 4×4 grid of numbers 1–15 and a blank "-" into a 2D array.
2. **Define the target state**
   * The goal board is [1, 2, 3, …, 15, "-"].
3. **Find the blank tile**
   * Scan the board to locate the row and column of "-".
4. **Prepare helper structures**
   * visited ← empty set (to store already seen states).
   * solution\_moves ← empty list (to record successful moves).
5. **Recursive search (DFS with backtracking)** Define puzzle(board, blank\_row, blank\_col):  
   * If the board matches the goal state → return **True**.
   * Convert board to a **unique representation** (e.g., string).
   * If this state is visited → return **False**.
   * Otherwise, add this state to visited.
   * For each possible move of the blank (Up, Down, Left, Right):  
     + Check if the move keeps the blank inside the board.
     + Swap the blank with the adjacent tile.
     + Recursively call puzzle() with the new blank position.
     + If the recursive call returns **True**:  
       - Append the move to solution\_moves.
       - Return **True**.
     + Otherwise, swap back (backtrack).
   * If no move leads to a solution, return **False**.
6. **Call the solver**
   * Run puzzle(board, blank\_row, blank\_col).
7. **Output result**
   * If True → print "Solved!" and the moves in reverse order.
   * If False → print "No solution found".

**FUNCTION is\_solved(board):**

target ← "123456789101112131415-"

flat\_board ← concatenate all elements of board row by row into a string

RETURN (flat\_board == target)

**FUNCTION board\_to\_string(board):**

result ← empty string

FOR each row in board:

FOR each cell in row:

result ← result + cell

RETURN result

**FUNCTION find\_blank(board):**

FOR r from 0 to 3:

FOR c from 0 to 3:

IF board[r][c] == "-":

RETURN (r, c)

RETURN (−1, −1) // not found

**FUNCTION puzzle(board, blank\_row, blank\_col, visited, solution\_moves):**

IF is\_solved(board): #Base Case

RETURN TRUE

state ← board\_to\_string(board)

IF visited contains state:

RETURN FALSE #Pruning

ADD state to visited

directions ← [(−1, 0, "U"), (1, 0, "D"), (0, −1, "L"), (0, 1, "R")]

FOR each (dr, dc, move) in directions:

new\_row ← blank\_row + dr

new\_col ← blank\_col + dc

IF new\_row and new\_col are inside board boundaries:

SWAP board[blank\_row][blank\_col] with board[new\_row][new\_col]

IF puzzle(board, new\_row, new\_col, visited, solution\_moves):

ADD move to solution\_moves

RETURN TRUE

SWAP board[blank\_row][blank\_col] with board[new\_row][new\_col] // backtrack

RETURN FALSE

**MAIN:**

board ← 4x4 array read from input

visited ← empty set of strings

solution\_moves ← empty list

(r, c) ← find\_blank(board)

IF puzzle(board, r, c, visited, solution\_moves):

PRINT "Solved!"

PRINT solution\_moves in reverse order

ELSE:

PRINT "No solution found"

15 Puzzle just solution -> Backtracking with DFS

15 Puzzle solution with least moves/least time -> Backtracking with DFS + BFS