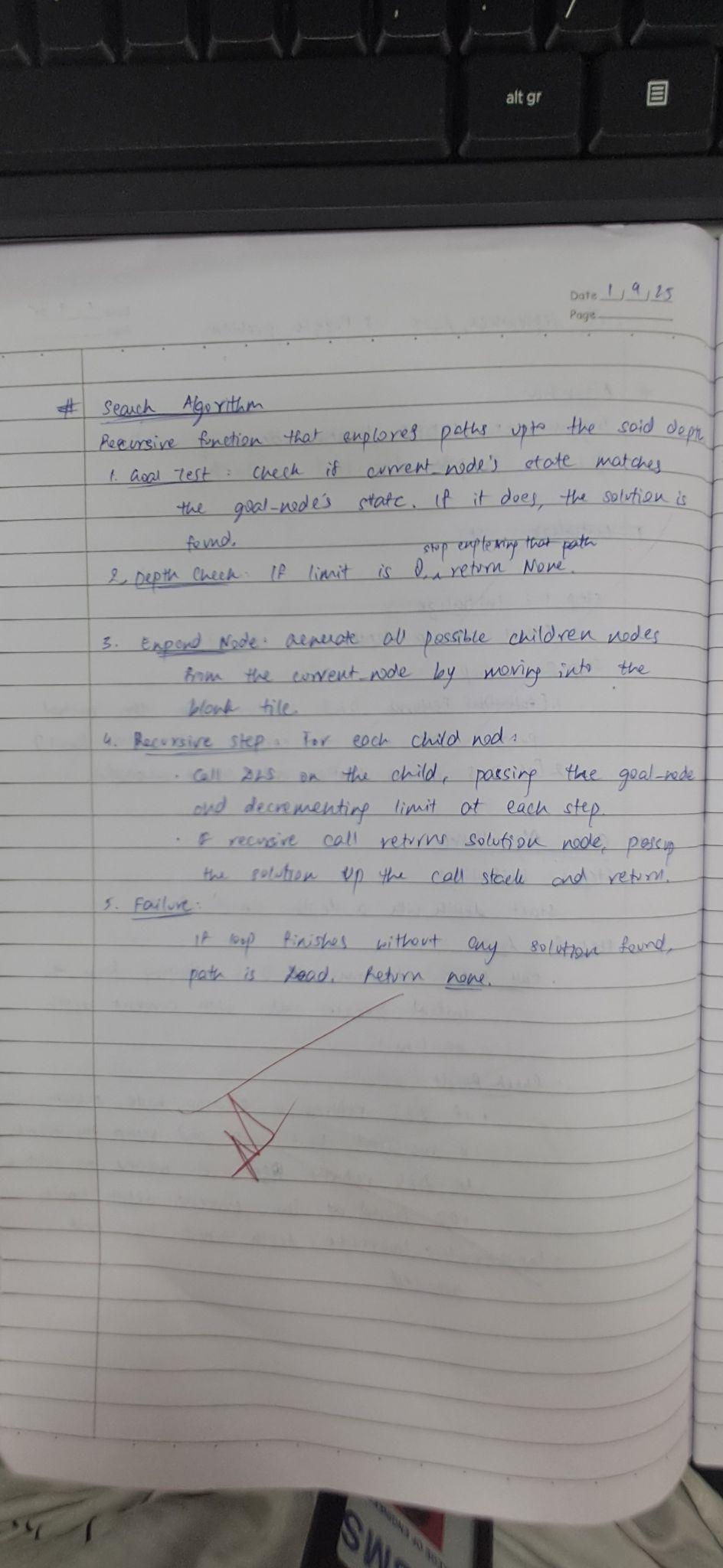
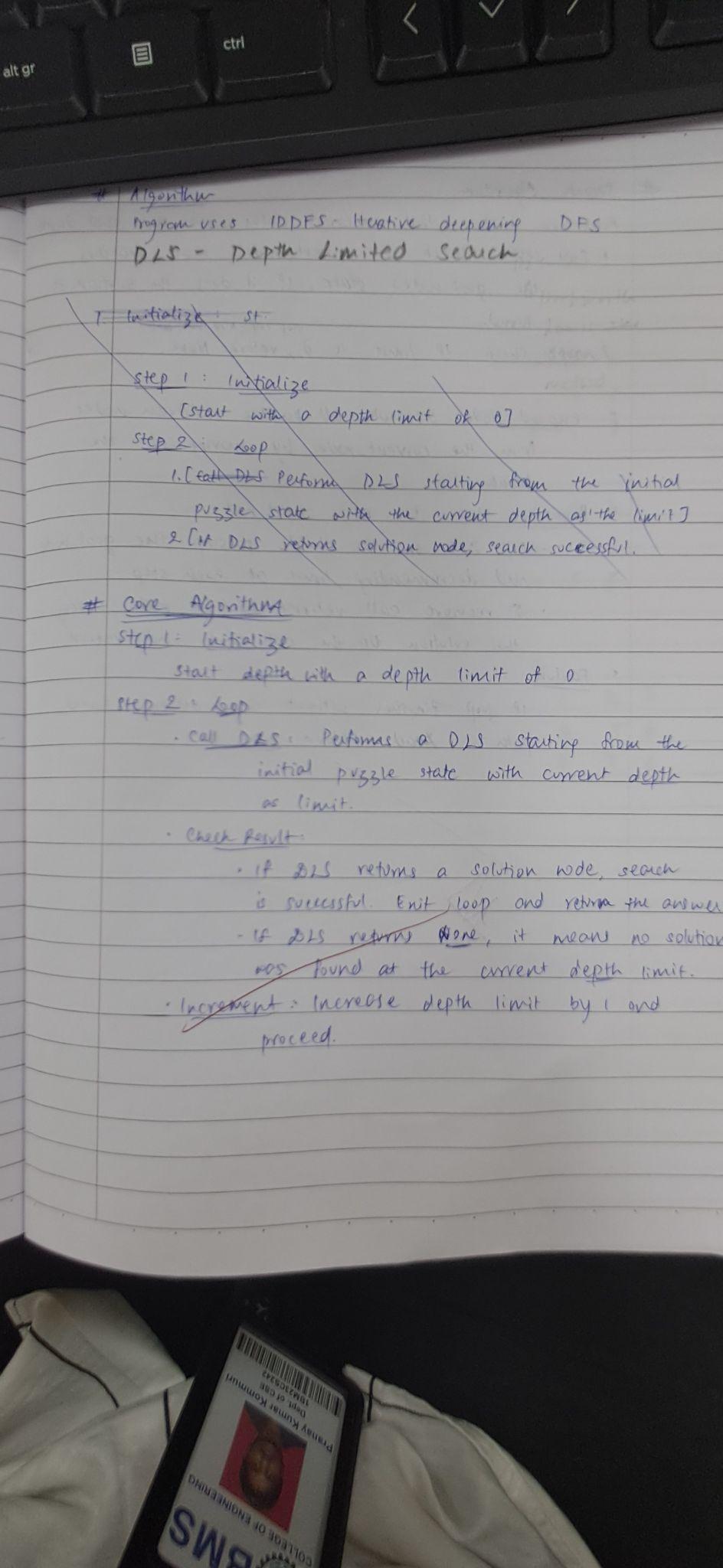
**Date:** 1st September, 2025

**Program Title:** 8-Puzzle Program

**Observation:**

****

****

**Syntax:**

**import collections**

**# A class to represent a node in the search tree**

**class Node:**

**"""**

**Represents a state in the 8-puzzle problem.**

**- state: A tuple of tuples representing the puzzle grid.**

**- parent: The node that generated this node.**

**- action: The move that was taken to reach this state ('UP', 'DOWN', etc.).**

**- depth: The depth of the node in the search tree.**

**"""**

**def \_\_init\_\_(self, state, parent=None, action=None, depth=0):**

**self.state = state**

**self.parent = parent**

**self.action = action**

**self.depth = depth**

**def \_\_hash\_\_(self):**

**return hash(self.state)**

**def \_\_eq\_\_(self, other):**

**return self.state == other.state**

**def find\_blank(state):**

**"""Finds the coordinates (row, col) of the blank tile (0)."""**

**for r, row in enumerate(state):**

**for c, val in enumerate(row):**

**if val == 0:**

**return r, c**

**return None**

**def get\_successors(node):**

**"""Generates all valid successor nodes from the current node."""**

**successors = []**

**state = node.state**

**r, c = find\_blank(state)**

**# Possible moves: (row\_change, col\_change, action\_name)**

**moves = [(-1, 0, 'UP'), (1, 0, 'DOWN'), (0, -1, 'LEFT'), (0, 1, 'RIGHT')]**

**for dr, dc, action in moves:**

**nr, nc = r + dr, c + dc**

**# Check if the new coordinates are within the grid bounds**

**if 0 <= nr < 3 and 0 <= nc < 3:**

**# Create a new state by swapping the blank tile**

**new\_state\_list = [list(row) for row in state]**

**new\_state\_list[r][c], new\_state\_list[nr][nc] = new\_state\_list[nr][nc], new\_state\_list[r][c]**

**new\_state\_tuple = tuple(tuple(row) for row in new\_state\_list)**

**successors.append(Node(new\_state\_tuple, parent=node, action=action, depth=node.depth + 1))**

**return successors**

**def recursive\_dls(node, goal\_state, limit):**

**"""**

**Recursive Depth-Limited Search (DLS).**

**This is the core recursive DFS function.**

**"""**

**# If the current node is the goal, we've found a solution**

**if node.state == goal\_state:**

**return node**

**# If the depth limit is reached, stop exploring this path**

**if limit <= 0:**

**return None**

**# Recursively explore successors**

**for successor in get\_successors(node):**

**result = recursive\_dls(successor, goal\_state, limit - 1)**

**if result is not None:**

**return result**

**return None**

**def iddfs(initial\_state, goal\_state):**

**"""**

**Iterative Deepening Depth-First Search (IDDFS).**

**Calls the recursive DLS with an increasing depth limit.**

**"""**

**depth\_limit = 0**

**while True:**

**print(f"Searching with depth limit: {depth\_limit}...")**

**root = Node(state=initial\_state)**

**result = recursive\_dls(root, goal\_state, depth\_limit)**

**if result is not None:**

**print("\n✅ Solution Found!")**

**return result # Return the goal node if found**

**depth\_limit += 1**

**def print\_path(node):**

**"""Prints the solution path from the goal node back to the root."""**

**if node is None:**

**print("No solution found.")**

**return**

**path = collections.deque()**

**current = node**

**while current is not None:**

**path.appendleft((current.action, current.state))**

**current = current.parent**

**print(f"Path length: {len(path) - 1} moves\n")**

**for i, (action, state) in enumerate(path):**

**if i == 0:**

**print("Initial State:")**

**else:**

**print(f"Move {i}: {action}")**

**for row in state:**

**print(row)**

**print("-" \* 15)**

**# --- Main Execution ---**

**if \_\_name\_\_ == "\_\_main\_\_":**

**# Define the initial and goal states as tuples of tuples for immutability**

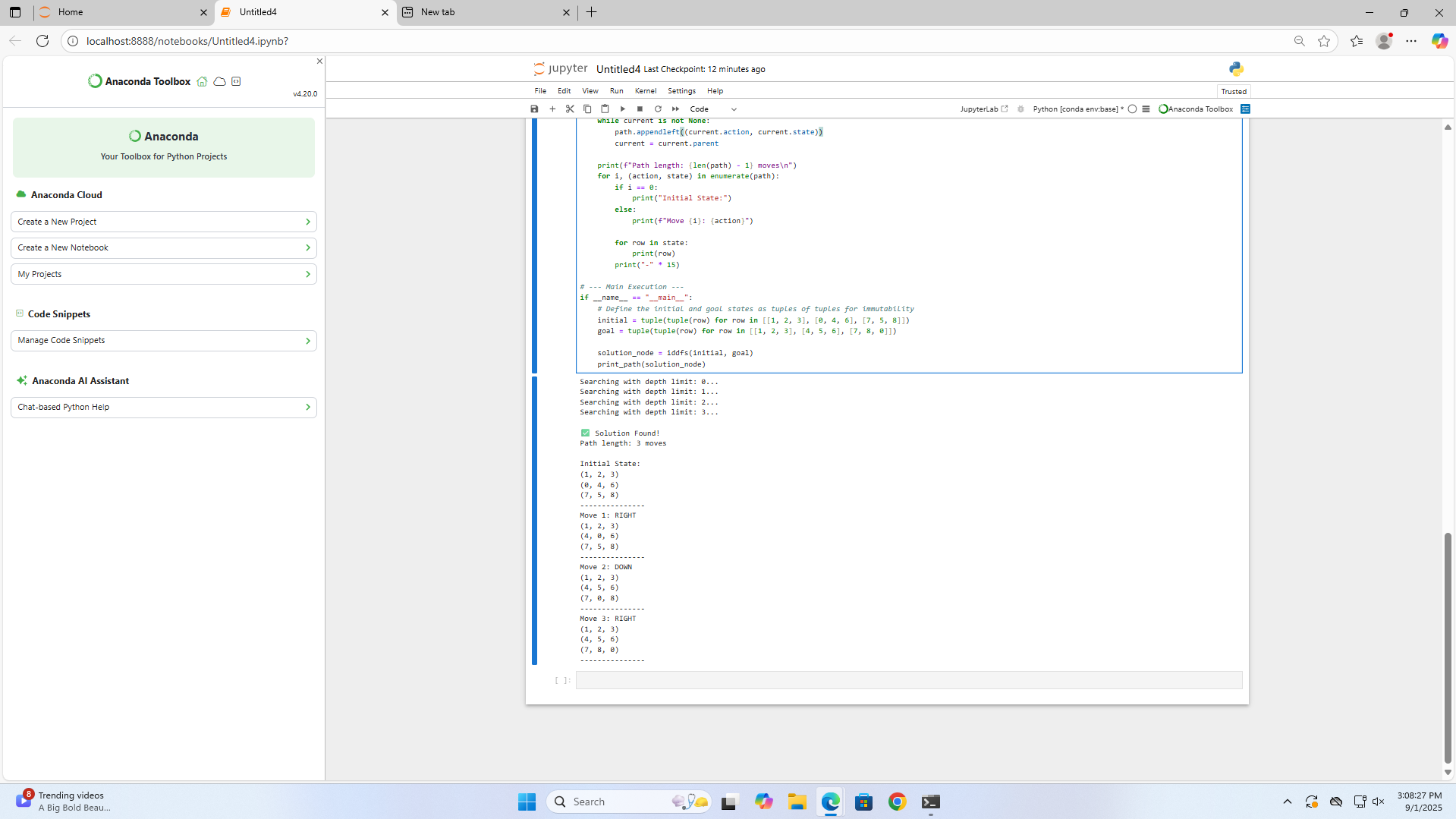
**initial = tuple(tuple(row) for row in [[1, 2, 3], [0, 4, 6], [7, 5, 8]])**

**goal = tuple(tuple(row) for row in [[1, 2, 3], [4, 5, 6], [7, 8, 0]])**

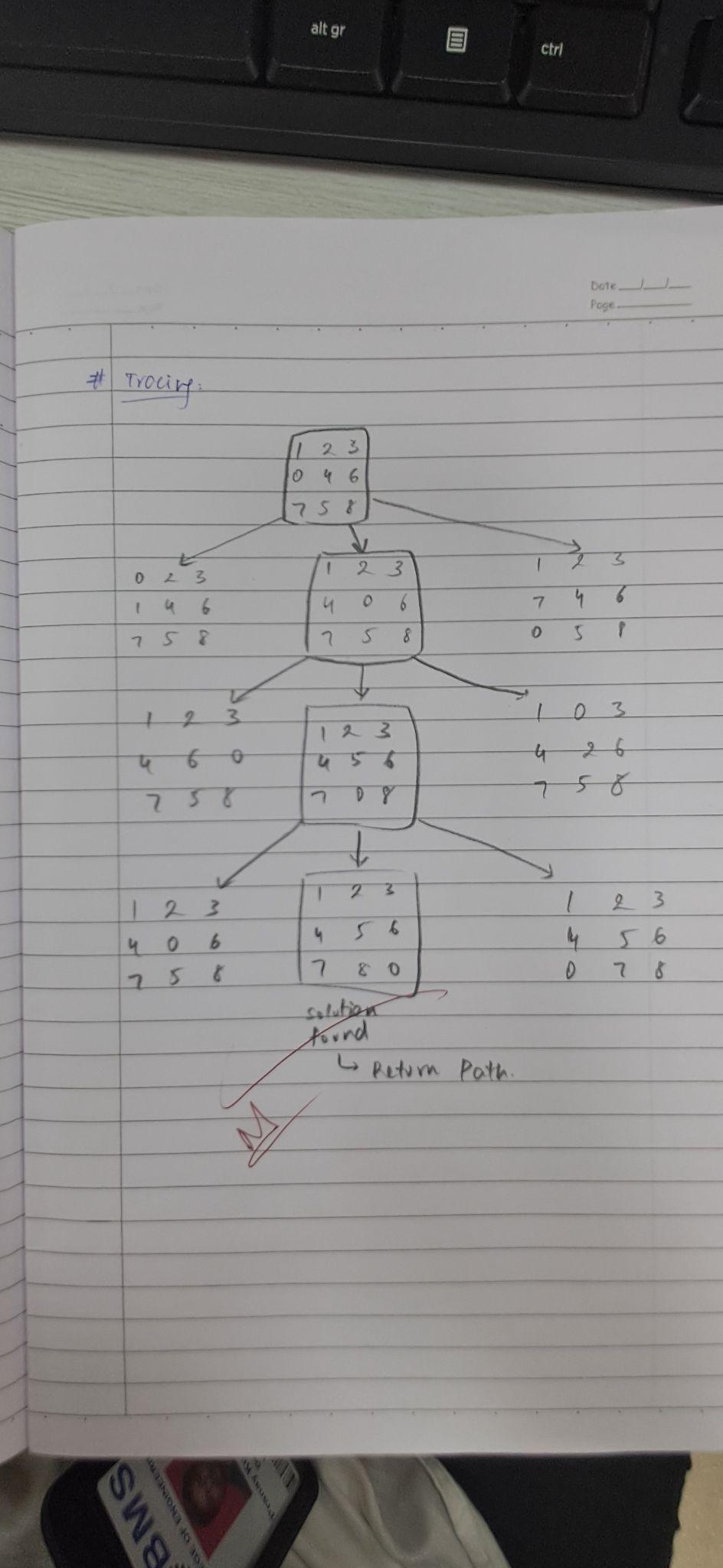
**solution\_node = iddfs(initial, goal)**

**print\_path(solution\_node)**

**Output:**



**Tracing:**

****