SAHIL BISWAS

16010422013

% Define predicate to solve the puzzle

solvepuzzle(State, Goal, Cost) :-

solvepuzzle(State, Goal, [State], 0, Cost).

% Base case: puzzle is already solved

solvepuzzle(State, Goal, \_, Cost, Cost) :-

State == Goal.

% Recursive case: try moving empty space and continue searching

solvepuzzle(State, Goal, Visited, CurrentCost, Cost) :-

move(State, NextState), % Move the empty space

not(member(NextState, Visited)), % Check if the new state has been visited

NewCost is CurrentCost + 1, % Increment the cost

total\_manhattan\_distance(NextState, Goal, Distance), % Calculate the total Manhattan distance

NewCost + Distance =< Cost, % Check if the new cost is less than or equal to the current cost

solvepuzzle(NextState, Goal, [NextState|Visited], NewCost, Cost). % Recursively call solvepuzzle with the new state

% Add the base case for the empty state

solvepuzzle([], \_, \_, \_, \_) :-

write('No solution'). % If the puzzle is not solvable, print "No solution"