# Part2. Srinivas Bhatt. Registration: 100455111

2.1. Changes for Wraparound and Zero-Cost Shifts

Modify Successor Generation:

Wraparound Columns: Adjust the generate\_successors method to account for moves that wrap around the grid. When calculating possible moves, if the blank tile is at the top row and a move up is considered, the blank tile should move to the bottom row of the same column, and vice versa.

Row Shifts: Implement logic to generate states resulting from shifting all rows circularly up or down. Since these moves have zero cost, they should be considered differently in the heuristic and path cost calculation.

Adjust Heuristic Calculation:

The Manhattan distance heuristic must be modified to account for the wraparound nature of the grid. This means considering the minimum distance a tile can move to its goal position, either directly or by wrapping around.

Since row shifts have zero cost, the heuristic might also need to consider the optimal row alignment achieved through these shifts, ensuring it still provides an admissible estimate.

Path Cost Calculation:

When calculating the cost of moves (g in the IDA\* terminology), ensure that shifts (which have zero cost) do not contribute to the path cost. This might involve tracking the type of move made to reach a new state and only incrementing the cost for non-shift moves.

2.2. Efficient Solution Modifications

Enhanced Heuristic: Develop a more sophisticated heuristic that better captures the reduced cost of achieving the goal state due to the new move types. This could involve precalculating the optimal row alignment for each tile and using this information to refine the Manhattan distance calculation.

State Representation: Consider a more complex state representation that can efficiently capture and update the grid state for both tile moves and row shifts. This might involve data structures that can quickly perform circular shifts.

Cycle Detection: Given the zero-cost shifts, there's a risk of generating and revisiting states without advancing towards the goal, especially in a depth-first search like IDA\*. Implement cycle detection mechanisms that prevent the algorithm from getting stuck in loops of zero-cost moves.

Pruning Strategies: Incorporate pruning strategies to reduce the search space. This could involve identifying when row shifts do not meaningfully contribute to reaching the goal faster and avoiding these moves.

2.3. Physical Design of an n-tile Puzzle

A physical design that corresponds to this modified n-tile problem could resemble a traditional sliding puzzle but with a few key differences:

Flexible Rows: The rows of the puzzle must be capable of shifting left and right in a circular manner, suggesting a mechanism that allows for the continuous movement of tiles around the grid's perimeter.

Wraparound Columns: Similarly, columns would need the ability to wrap around, meaning moving a tile off the top edge would bring it back at the bottom of the same column, and vice versa. This could be achieved with a cylindrical or toroidal puzzle design.

Shift Mechanism: There would need to be a way to shift all rows or columns simultaneously without increasing the move count, possibly via a separate control mechanism that engages the entire row or column in a shift movement.

This design would offer a unique twist on the classic puzzle, introducing a new layer of complexity and requiring solvers to think in terms of both two-dimensional and circular space.