N - Puzzle

Observations & Findings :

Puzzle	BFS	DFS	A*	SMA*
1 8 2 0 4 3 7 6 5	Iterations : 262 Max Nodes : 175 Moves : 9	Iterations : 39007 Max Nodes : 26984 Moves :	Iterations : 262 Max Nodes : 175 Moves : 9	Iterations : 172 Max Nodes : 52 Moves : 9
1 6 2 5 7 3 0 4 8	Iterations : 354 Max Nodes : 227 Moves : 10	Iterations : Max Nodes : Moves :	Iterations : 354 Max Nodes : 227 Moves : 10	Iterations : 46716 Max Nodes : 58 Moves : 890
0 1 3 4 2 5 7 8 6	Iterations : 14 Max Nodes : 13 Moves : 4	Iterations : Max Nodes : Moves :	Iterations : 14 Max Nodes : 13 Moves : 4	Iterations : 9 Max Nodes : 6 Moves : 4
8 1 3 4 0 2 7 6 5	Iterations : 3177 Max Nodes : 1948 Moves : 14	Iterations : Max Nodes : Moves :	Iterations : 3177 Max Nodes : 1948 Moves : 14	Iterations : 2349 Max Nodes : 723 Moves : 14
1 2 3 4 5 6 0 8 9 10 7 11 13 14 15 12	Iterations : 8 Max Nodes : 12 Moves : 3	Iterations : Max Nodes : Moves :	Iterations : 8 Max Nodes : 12 Moves : 3	Iterations : 6 Max Nodes : 10 Moves : 3

SMA* uses less memory when compared with A* and BFS (refer to “Max Nodes” for more info)

Approach :

BFS & DFS -

1. Given an n x n grid with (n^2) - 1 elements spread randomly across n^2 cells (1 empty cell)
2. Assume you are located at the empty cell; 4 moves are possible -
   a. swap empty cell and Left cell
   b. swap empty cell and Right cell
   c. swap empty cell and top cell
   d. swap empty cell and bottom cell
3. Consider only those configurations of grid that are not formed previously (don't consider duplicate grids)
4. Repeat this process until -
   a. you get the grid you wanted (or)
   b. no other unique grid exists
5. In BFS, Queue is used to store unique configurations of grid. Each configuration is dequeued in FIFO order to explore even more unique grids

6. In DFS, instead of Queue, stack is used i.e LIFO

A* -

1. Instead of using LIFO or FIFO order for retrieving the unique configurations of grid, the configuration with minimum total_cost is chosen
2. total_cost (of a cell) = previous_cost + future_cost (or)
3. total_cost (of a cell) = #_of_cells_it_has_travelled_from_it's_original_given_random_configuration + "how_far_from_the_position_in_desired_grid_configuration (Manhattan Distance) i.e in terms of # of cells" .
    a. Implies, total_cost_of_the_grid = total_cost_of_each_cell_in_it
4. In step 1, make sure you remove the grid with minimum total_cost which you've retrieved

SMA* -

It is same as A* except - before retrieving the grid with minimum total_cost, make sure the the total # of grids doesn't exceed the threshold, say, 180; if exceeded, remove the grid with maximum total_cost

N x N Coloring

Observations & Findings:

Input	BFS	DFS

1 2 3 2
2 2 4 1
2 2 5 1
1 1 6 2
	Iterations : 335047
	Max Nodes : 77623
		Iterations : 52
		Max Nodes : 17

1 2 3
2 2 4
2 2 5
	Iterations : 937
	Max Nodes : 260
		Iterations : 489
		Max Nodes : 12

1 2 3 2	Iterations : 11 Max Nodes : 8	Iterations : 4 Max Nodes : 5

1. DFS uses less memory than BFS (see “Max Nodes”)
2. DFS has found the solutions in lesser iterations than that of BFS

Approach :

BFS -

```
1. Given a ramdom_grid, find the frequency of all the colors (1 to k) --- freq_lst
2. Make an empty grid, all 0s, n rows, n cols --- grid
3. Put grid[0][0] in the queue
4. while(queue is not empty)
        ___node = dequeue
        ___given a cell, put all "available colors (taken from freq_lst)" in the cell --- modified_grid
        ___update the freq_list according the color you've chosen in the prev step --- modified_freq_lst
        ___if(modified_grid is valid)
                ___put modified_grid, next_cell, modified_freq_lst - wrapped in an object, in the queue
```

DFS -

Same as DFS, except - use Stack instead of Queue
