# Part 2: Warnsdorff Knight's Tour

Now let's strategize a little, our goal is to get as many moves as possible. For this section, we'll use Warnsdorff's heuristic/algorithm. To summarize what you'll be doing in this section: Always choose the move which has the fewest possible subsequent moves. Why the fewest? We want to make sure the moves we leave have options, that way we can make more moves.

Each "next" move should:

1. Be a valid move according to the current position
2. Be unvisited previously
3. Must have the least number of subsequent moves of the resulting options (from part a & b)

Please start your program off with the following adjacency matrix (all possible moves):

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |
| 3 | 4 | 6 | 6 | 6 | 6 | 4 | 3 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 3 | 4 | 6 | 6 | 6 | 6 | 4 | 3 |
| 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |

Lets start a tour from the top left (0,0) place:
(Right off the bat, we know the top left corner is inaccessible, so we subtract one from its possible moves)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |
| 3 | 4 | **5** | 6 | 6 | 6 | 4 | 3 |
| 4 | **5** | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 3 | 4 | 6 | 6 | 6 | 6 | 4 | 3 |
| 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |

Let's say I choose row 2, column 1 (you choose how to break ties):

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1** | 3 | **3** | 4 | 4 | 4 | 3 | 2 |
| 3 | 4 | 5 | **5** | 6 | 6 | 4 | 3 |
| 4 | **5** | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | **7** | 8 | 8 | 6 | 4 |
| **3** | 6 | **7** | 8 | 8 | 8 | 6 | 4 |
| 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 3 | 4 | 6 | 6 | 6 | 6 | 4 | 3 |
| 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |

Etc. etc.

Keep going with the process, always choosing the lowest value in the accessibility matrix. There are a couple of things you MUST keep in mine:

- Is an accessibility matrix value of 1 a good choice to make? Why or why not?
- What if the matrix dips to negative values? Is this good or bad?

## Last two logistical items:

- Please make sure to implement the Knights Tour interface in WarnsdorffKnightsTour.
- Also, write a method called developAccessibilityMatrix(int num) that creates a two 2D matrix of accessibility values **dynamically** (aka don't hard code values!).