## Challenge Problem 7: Relation property finder functions

MATH2603: Discrete Mathematics

---

**Warning:** This Challenge Problem is designated as a **Programming Challenge Problem**. I have compiled resources for you to learn the minimum required amount of Python (and other related material) in order to complete this (and other) challenge problems. These can be found in `Blackboard` → `Content` → `Technology Skills`. If this is your first Programming Challenge Problem, you should start there.

**The Problem:** In this Challenge Problem, you'll have a chance to dive a little deeper into mathematical relations. You will write four different functions in SageMath: `isReflexive`, `isSymmetric`, `isAntisymmetric`, and `isTransitive`. The functions should behave like this:

- The input to the function is a relation on a set, entered as a *dictionary*.
- The output of the function `isReflexive` is a Boolean, returning `True` if the relation that was input has the reflexive property, or `False` if it does not have the reflexive property.
- The remaining three functions `isSymmetric`, `isAntisymmetric`, and `isTransitive` should behave similarly: `isSymmetric` should return `True` if the relation is symmetric and `False` otherwise; `isAntisymmetric` should return `True` if the relation is antisymmetric and `False` otherwise; `isTransitive` should return `True` if the relation is transitive and `False` otherwise.

At the end of this document, you can find a screenshot with some examples of the input and outputs that are expected.

**Tools you can use:** You can use any function or method found in the standard Python tool set or in networkX to help you, as long as the function or method does not actually perform the operations your code is being asked to do. For example, if you discover a method that will determine directly, in a single command, whether a relation is reflexive, then you may not use it. But if you find helper functions and methods that can be chained together or used creatively to do this, then go for it.

**Restrictions to follow:** You are not allowed to use any other libraries besides networkX. Also to reiterate, do not use any Python or networkX functions or methods that would do this work directly.

**Other notes:** Please note that the functions you are writing should return, not print to give the result.

**Submitting your work:** You will submit your work in a Jupyter notebook with each of the functions above appearing in the same code block. So, there should only be one large block of code in your submission. Please **do not include any examples or test cases that you might use.** (But please *do use* test cases to check the correctness of your code.) Each function should also be well-documented by including a clear, thorough description that explains in English how the code for each function works. You can put those explanations in a separate cell in your Jupyter notebook as text, or you can include them as comments in your Python code. Also, please make sure you have given your function the correct names as indicated above.

**How to submit your work:** Go to Blackboard and submit your work in the area for Challenge Problem 7 by uploading your Python notebook (.ipynb) to Blackboard.

**Evaluation:** Like all Advanced Explorations, your work will be evaluated using the EMRN rubric. Please see the statement of this rubric in the syllabus for an explanation of how it is used. When applied to this Advanced Exploration, the following criteria help to assign the grade:

- **E:** The functions produce correct output on all (100%) of the test cases. Also, each function has an English description that provides a clear explanation of how each function works.
- **M:** The functions produce correct output on at least 3/4 of the test cases. Also, each function has an English description that explains how the code works.
- **R:** There are no syntax errors in the code but correct output is produced on less than 75% of the test cases. Or, the explanations are provided but are not clear or do not explain how all parts of the code work.
- **N:** There is a syntax error produced when the code is executed; or there is at least one explanation missing; or the code uses external libraries; or the code has systemic flaws.

**Please note:** You are expected to test your code thoroughly before submitting it. Make sure you do the following:

1. Before you submit, put your code in a notebook and run it one last time to make sure it does not produce errors when the code is executed. If the submission throws an error when I execute it, the grade on the work will automatically be N, because debugging your code is your responsibility.
2. Before you submit, test your code with several diverse test cases to make sure it is producing the correct output each time. Use a wide variety of test cases for maximum certainty that you've solved the problem.

In [17]:
```
relation1 = {1:[1, 3, 4], 2:[2,3], 3:[3], 4:[3,4]}
isReflexive(relation1)
```

Out[17]: True

In [19]:
```
isSymmetric(relation1) # Should be False because 2 -> 3 but not 3 -> 2
```

Out[19]: False

In [23]:
```
isAntisymmetric(relation1) # Should be true because all connections are one-way
```

Out[23]: True

In [24]:
```
isTransitive(relation1)
```

Out[24]: True

In [25]:
```
relation2 = {'Grand Rapids': ['Ann Arbor'], 'Ann Arbor':['Lansing', 'Detroit'],
             'Lansing': [], 'Detroit':['Grand Rapids']}
```

In [26]:
```
isReflexive(relation2)
```

Out[26]: False

In [27]:
```
isSymmetric(relation2)
```

Out[27]: False

In [28]:
```
isAntisymmetric(relation2)
```

Out[28]: True

In [29]:
```
isTransitive(relation2)
```

Out[29]: False