You may use class notes, class text, Python books or the internet, but please reference your use with appropriate detail. Work on your own or (if you wish) with one partner.

Due Friday, 2010-04-16. Electronic submissions only. All programs must be submitted either as Sage/Python worksheets. Document all your code. If you work with a partner, both of you must test all the documentation included in your submission.

1. (Option A) Write a Python module containing two classes, A MatrixSpace class (the “ring” of matrices kxn matrixes (mod m), where m is an integer), and a MatrixElement class (representing the individual matrices). You may not use pre-existing Sage classes for this. Here the integer m¿1 is not necessarily a prime.

   • MatrixSpace should have the following methods: __repr__, __str__, __call__, row_dimension, column_dimension.
   
   • MatrixElement should have the following methods: __repr__, __str__, row_vectors (returning the list of rows), column_vectors (returning the list of columns), matrix_entry (which has input i, j and outputs the i,j-th entry of the matrix), __add__, __sub__

   For extra credit, implement cofactor (which has input i, j and outputs the i,j-th cofactor of the matrix), __mul__ (when the matrices are square) and det (when the matrices are square).

   When finished,

   MatrixSpaceModm(10, 2, 3)

   will represent the set of 2 × 3 matrices mod 10, with addition mod 10, and

   A = MatricesModm(10, [[1,2,3],[4,5,6]])
   B = MatricesModm(10, [[1,0,7],[2,8,3]])
will represent elements of that set. You module should enable you to compute \( A+B \) and \( A-B \) correctly.

Fully document your methods with INPUT, OUTPUT, and EXAMPLES for each docstring.

2. (Option B) A graph is a pair \( G = (V,E) \), where \( V \) is a set of vertices (often labeled by non-negative integers) and \( E \subset V \times V \) is a set of edges. Implement a graph class with methods \_\_repr\_, \_\_str\_, vertex_list (which lists all the vertices), edge_list (which lists all the edges), add_vertex, add_edge, neighbors (which has input a vertex \( v \) in the graph and returns all those vertices connected to \( v \) by a single edge).

Extra credit: Also, allow your vertices to have integer weights. Implement the chip firing game: In this care, the vertices are the players and the weights are their “chips” (imagine each chip is worth 1 dollar). A vertex is active if it has more chips than neighboring vertices. Only active vertices can “fire”. When you “fire” a vertex, the player must pay one chip to each neighbor (so if there were 5 neighbors then the weight of that vertex would be 5 lower than before the firing).

3. Using your class in Option A (or using Sage’s matrices over \( GF(2) \)), implement the 3 \( \times \) 3 determinant game. Here are the rules: there are two players - Player 0 and Player 1. You start with an “empty” matrix and players alternately enter either 0 (for Player 0) or 1 (for Player 1). Player 0 wins if the determinant is 0 (mod 2) and loses otherwise. A “flip of a coin” decides whose turn is first.

Extra credit if you can determine a winning strategy (assuming both players play “optimally”).

4. The integers 1 to 500 are written on the blackboard of a classroom. Students Alice and Bob play the following game: the students alternate erasing a number on the board. The game ends when there are exactly two numbers remaining. If the numbers are additive inverses in \( \mathbb{Z}/3\mathbb{Z} \) then Bob wins; otherwise Alice wins. If Alice starts, does Bob have a winning strategy (assuming both players play “optimally")?

(Okay, this can be solved with no programming, but if you can program this, you will get extra credit.)
5. Write Python code to solve the following problem.

A reporter asks a military officer how many soldiers are at a certain military base. The officer, not wanting to reveal such sensitive information, but also not wanting to seem overly secretive, gives an indirect answer:

When my soldiers form 2 columns there is 1 soldier left.
When my soldiers form 3 columns there are 2 soldier left.
When my soldiers form 4 columns there are 3 soldier left.
When my soldiers form 5 columns there are 4 soldier left.
When my soldiers form 6 columns there are 5 soldier left.
When my soldiers form 7 columns there are 0 soldier left.

How many soldiers are there?