1. Create a Sage/Cocalc account.
   
   (a) Start the Chrome browser.
   (b) Go to http://cocalc.com
   (c) Login. You should see an existing Project for our class. Click on that.
   (d) Click “New”, then “Worksheets”, then call it s11.

   Our idea was to define a “graph poset” consisting of some subgraphs of a given graph \( g \), use mobius inversion to get a series formula for the value of some graph invariant of interest (for us the independence number), truncate that series and get an estimate. Ideally, this would be a good estimate for a small number of terms but of course the value of the entire sum will be the value (by the Mobius Inversion Theorem).

   In today’s Lab we’ll investigate posets connected to vector spaces. The main idea here is that the cycles of a graph form a vector space. So they have a basis (of cycles), and all the other cycles are linear combinations of these (where the vector sum is the symmetric difference of the edge sets, and the field is \( \mathbb{Z}_2 \)–or \( \mathbb{F}_2 \) or \( \text{GF}(2) \), etc).

   The cycle rank (or cyclotomic number, etc) of a connected graph is the the number of edges of the graph minus the number of edges of a spanning tree (so \( e - n + 1 \)). You can argue that this is the cardinality of a minimum basis of the cycle space—and thus the dimension of this space.

   We’ll use the following graph (the “spaceship”) as our main example today:

![Graph Diagram]

   You should notice the two triangles, the square, the 3 cycles you can make as the symmetric difference of pairs of 2 of these, the big cycle all the way around and the empty cycle (no edges). The 2 triangles and the square are one basis for the cycle space (but not the only one).
2. All the functions mentioned here are in the file `posets5.sage` in your Handouts folder. So you might start by copying that to your main directory and loading that: `load('posets5.sage')`.

3. Run `cycle_rank(spaceship)` to find the cycle rank.

4. Run: `spaceship.cycle_basis()` to get a cycle basis for this graph.

5. We’ll want the cycles edges as a list or set of individual edges (so that we can find symmetric differences, etc). Run: `get_basis_edges(spaceship)`.

**Eulerian Subgraph Poset.**

6. We can now build our poset of eulerian subgraphs with inclusion by finding all linear combinations of the basis cycles. (There are several auxiliary functions needed, but we won’t look much at these details). Run: `make_eulerian_poset(spaceship)`. Note that we get 9 elements: the $2^3 = 8$ graphs we get as linear combinations of the 3 basis elements—together with the non-eulerian parent graph.

7. Let’s give this poset a name. Run: `Eship = make_eulerian_poset(spaceship)`.

8. To view the graphs we have, run:

   ```python
   for h in Eship:
       h.show()
   ```

9. To get a (bad) visualization of the Hasse diagram run: `Eship.show()`.

10. To see the values of the built-in mobius function, run:

    ```python
    for h in Eship:
        for g in Eship:
            if is_subgraph(h,g):
                print h.edges(labels=False)
                print g.edges(labels=False)
                print Eship.moebius_function(h,g)
    ```

11. Now let’s check that the function we get from the Mobius Inversion Theorem indeed calculated the independence numbers correctly:

    ```python
    for h in Eship:
        print h.edges(labels=False)
        print independence_number(h), mobius_F(Eship, independence_number, h)
    ```

12. Now let’s check the Taylor series approximations of the independence number:

    ```python
    for i in [1..spaceship.order()]:
        print i, truncated_mobius_F(Eship, independence_number, spaceship, i)
    ```
Cycle Unions Poset.

13. We can now build a poset of subgraphs of all unions of basis cycles with inclusion. Run: `make_cycle_union_poset(spaceship)`. Note that we get 7 elements: the \(2^3 = 8 - 1\) graphs we get from non-empty unions of the edges of the 3 basis cycles. This time the parent graph is already included.

14. Let’s give this poset a name. Run: `Cship = make_cycle_union_poset(spaceship)`.

15. To view the graphs we have, run:

```python
for h in Cship:
    h.show()
```

16. To see the values of the built-in moebius function, run:

```python
for h in Cship:
    for g in Cship:
        if is_subgraph(h,g):
            print h.edges(labels=False)
            print g.edges(labels=False)
            print Cship.moebius_function(h,g)
```

17. Now let’s check that the function we get from the Mobius Inversion Theorem indeed calculated the independence numbers correctly:

```python
for h in Cship:
    print h.edges(labels=False)
    print independence_number(h), mobius_F(Cship, independence_number, h)
```

18. Now let’s check the Taylor series approximations of the independence number:

```python
for i in [1..spaceship.order()]:
    print i, truncated_mobius_F(Cship, independence_number, spaceship, i)
```
Vector Spaces Poset.

We can now do computational experiments to check that our theoretical work on the poset of subspaces of a vector space agrees with practice. In particular we can check that the mobius formula we derived agrees with the one coming straight from the definition.

19. First lets make the vector space of dimension 2 over the field $\mathbb{F}_3$ (or GF(3)). Run: 
$$V=\text{VectorSpace}(\text{GF}(3),2).$$

20. While we entered the dimension we can also recover it: $V.\text{dimension}()$.

21. We can also get a basis. Run: $V.\text{basis}()$.

22. While we entered the field, we can also recover that. Run: $V.\text{base_field}()$.

23. Now let’s create the poset of all subspaces of this vector space. The secret is that Sage will generate all the subspaces; we just need to collect them and define an appropriate relation. Run: $VS=\text{make_vector_space_poset}(3,2)$.

24. Now let’s see what the elements are and count them:

```python
count = 0
for V in VS:
    print count, V
    count += 1
```

25. We can also try to view the Hasse diagram of this poset. Run: $\text{view}(VS)$.

26. Now let’s test our mobius function!

```python
for V1 in VS:
    for V2 in VS:
        if is_vector_subspace(V1,V2):
            print V1
            print V2
            print VS.moebius_function(V1,V2), vector_space_mobius(V1,V2)
```

How can we use this for our graph investigations?!?!!?