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 s08.

Problems

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 the value (by the Mobius Inversion Theorem).

But we’ve had some problems for the graph posets we’re defined:

   (a) Even for a relatively small graph, the number of possible small subgraphs grows very fast. So for the Petersen graph with 15 edges, and a graph poset of subgraphs built up from those edges, constructing the poset with just subgraphs with up to 5 of the edges means at least \((\binom{15}{5}) = 3003\) edge subsets and \((\binom{15}{5})^2\) entries to compute in making the zeta function table (and maybe inverting that matrix depending on how Sage computes values of the mobius function).
   
   (b) So we need to find smaller but still useful graph posets—if we are to use these ideas for practical computations. But how?
   
   (c) We’ve also seen that in the case of Inclusion-Exclusion and the Euler totient function, nice results were a consequence of finding a closed-form formula (not directly in terms of \( F \)) for the \( G \) function— which in turn was the result of finding a relevant partition. Can we find a relevant partition in the case where \( F \) is the independence number of a graph? That alone would be new and of theoretical value—even if turns out to not help our practical computations.

In today’s Lab we’ll try two ideas just to reduce the number of graphs in these posets—next week we’ll implement the mobius function for acyclic graph posets used in the Babic and Trinajstic paper. The first idea is that for connected graphs there are typically fewer (much fewer) vertices than edges. So if we consider subsets of vertices there will at least be less terms in our mobius function series. Then if we take the corresponding induced subgraph we don’t ever have to concern ourselves directly with edges—we get whatever the induced edges are. So here’s a relation we can use for induced subgraph posets:
2. So we can simplify our relation by just checking if one vertex set is a subset of another. Since (we assume) they are both subsets of the same parent set that’s all we have to check to see if one is an induced subgraph of another.

```python
def is_induced_subgraph(g1, g2):
    v1 = g1.vertices()
    v2 = g2.vertices()
    if not Set(v1).issubset(Set(v2)):
        return False
    else:
        return True
```

3. All of today’s code including the last function are in “posets3.sage” in your CoCalc project Handouts folder. See what’s in there. It includes immutable copies of \( p_3 \), the Petersen graph, \( c_5 \) and the BuckyBall graph (which I confusingly called \( c_60 \)—what the chemists call it). Load that file.

4. Try: \( P=\text{make}_m\text{vertex}\text{poset}(p_3, 2) \). This will create a poset \( P \) whose elements are all 2 vertex subgraphs of \( p_3 \). It doesn’t require that they are connected. Since they are induced, this actually captures some “long range” independence structure. Evaluate (RUN) \( P \) to see what you have.

5. To see the graphs themselves run:

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

6. We know that subsets of any non-trivial set get very large very fast. You might wonder: even if we just take a selections of the produced subsets does Sage produce ALL the subsets and THEN make a selection. Fortunately the Subset constructor is a \textit{generator}: it only produces the sets one at a time as requested by the user. Let’s see this in action. There are \( \binom{10000}{2} \) 2-element subsets of a 10000-element set. Run \( \text{binomial}(10000,2) \) to see what this number is.

7. Now run \( \text{Subsets}(10000,2) \) to generate all 2-subsets of \( \{1,2,\ldots,10000\} \). See how fast the output is! There’s no way it stored all these sets. Try even bigger numbers than 10000.

8. Now let’s get randomly-generated elements. First give this collection the name \( S \). Run \( S=\text{Subsets}(10000,2) \). To get a random element run \( S.\text{random}_\text{element}() \). Try this a few times.

    induced subgraph posets and random subsets of vertices

9. We’ll want random subsets of our vertex set of a fixed number of vertices. That’s in the auxiliary function \( \text{random}_\text{vertex}_\text{sets} \). Try \( \text{random}_\text{vertex}_\text{sets}(\text{pete},2) \).
10. Now we can make posets with random selections of vertex sets up to a specified number. Let’s make a poset for $p3$ with random vertex sets up to 2 vertices. Run \( P = \text{make_m_vertex_random_poset}(p3, 2) \). Again to see what we have run:

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

11. We can try our (theoretical) value of the independence number from the mobius $F$ function. Run: \( \text{mobius}_F(P, \text{independence_number}, p3) \).

12. Let’s continue with some of our other graphs. Let’s do the Petersen graph. Run: \( P2 = \text{make_m_vertex_random_poset}(\text{pete}, 3) \).

13. And check the theoretic value \( \text{mobius}_F(P2, \text{independence_number}, \text{pete}) \).

14. Let’s do $c60$. Run \( P3 = \text{make_m_vertex_random_poset}(c60, 3) \). Check the theoretical value of the independence number.

15. We can do this with any graph. Let’s make a random graph $r$ with 100 vertices and edge probability $\frac{1}{2}$. Run: \( r = \text{graphs.RandomGNP}(100, 0.5) \). Find the true independence number of your graph with: \( \text{independence_number}(r) \) (results will vary!)

16. Now make a graph poset. Run: \( P5 = \text{make_m_vertex_random_poset}(r, 50) \) with vertex subsets with as many as 50 vertices.

17. The previous graphs were made immutable (hashable) when we originally defined them. To use $r$ for our poset codes we need an immutable copy. Run: \( \text{mobius}_F(P5, \text{independence_number}, r.\text{copy}(\text{immutable=True})) \).

18. Recall that our “truncated mobius” function removes the parent graph from the poset. We’ll now see different results. Run: \( \text{truncated_mobius}_F(P5, \text{independence_number}, r.\text{copy}(\text{immutable=True}), r) \).

19. We can now try our Taylor series functions using our random posets and our truncated mobius function. Try: \( \text{taylor_random}(\text{independence_number}, c60, 2) \). This example uses vertex sets with up to 2 vertices. Now increase the number of vertices.

20. Try this with various of our graphs (the corresponding posets get built automatically) and various choices for numbers of vertices.