1. Log in to your Sage Cloud account.
   
   (a) Start Firefox or Chrome browser.
   
   (b) Go to http://cloud.sagemath.com
   
   (c) Click “Sign In”.
   
   (d) Click project Classroom Worksheets.
   
   (e) Click “New”, call it c33, then click “Sage Worksheet”.

2. (Ramanujan) 2, 9, 16, etc. can be written (uniquely) as the sum of 2 cubes (1^3 + 1^3, 1^3 + 2^3, 2^3 + 2^3, etc.). Find the smallest integer which can be written as the sum of 2 cubes in 2 different ways.

Now we will define a new graph function for computing the hard-to-compute stability number of a graph. The stability number of a graph is the largest number of vertices in the graph that have no edges between them. So if \texttt{g=graphs.PetersenGraph()}, its stability number is 4.

The naive (and inefficient) way to find a largest stable set in a graph is to test every subset of vertices, check if it is stable, and then keep track of the largest one you’ve seen up to that point. The next big idea for finding a maximum stable set was due to Tarjan and Trojanowski in the 1970s: they noted that each vertex \( v \) of a graph is either in a maximum stable set \textit{or} it is not. And, if \( v \) is in a maximum stable set then none of the vertices it is touching (that it is adjacent to is), called the neighbors of \( v \), can be in that set.

Finding the neighbors of a vertex \( v \) in a graph \( g \) is a built-in graph method: \texttt{g.neighbors(v)}

3. Let \texttt{g=graphs.PetersenGraph()}. Find the neighbors of vertex 0 in \( g \). Use \texttt{g.show()} to check.

So our problem of finding a maximum stable set in a graph can be reduced to the problem of finding a maximum stable set in two smaller subgraphs: (1) the graph formed by removing vertex \( v \) and (2) the graph formed by removing \( v \) and its neighbors. In this case, we assume that \( v \) is in the maximum stable set.

4. So we need to be able to form these graphs. Let \( g \) be a graph with vertex set \( V \). Let \( S \) be any subset of \( V \). Then you can find the graph formed by \( S \) together with all the edges that are between vertices of \( S \) in the original graph \( g \) with the command \texttt{g.subgraph(S)}. This is a new graph. We can give it a name, say \( h \) by \texttt{h=g.subgraph(S)}.
5. Let’s see the graphs that need to be formed when we apply the Tarjan-Trojanowski idea to vertex 0 of the Petersen graph. We’ll need to form two sets $S_1$ and $S_2$ and the corresponding graphs. $S_1$ is all the vertices of $g$ except 0 and $S_2$ is all the vertices of $g$ except 0 and its neighbors.

Try $S_1=g$.vertices(), $S_1.remove(0)$. Now evaluate $S_1$ to see this set. Then try $h=g$.subgraph($S_1$) and then $h$.show().

6. Now removing $v$ and its neighbors will require more work:

\[ S_2=g$.vertices() \\
S2.remove(0) \\
for w in g.neighbors(0): \\
    S2.remove(w) \]

Evaluate $S_2$ to see this set. Now try $h=g$.subgraph($S2$) and $h$.show().

7. To simplify things in the future, we should write a function to remove a vertex and its neighbors from a graph and produce a new graph with $v$ and its neighbors removed.

\[
def remove_vertex_and_neighbors(g,v):
    S2=g$.vertices() \\
    S2.remove(v) \\
    for w in g.neighbors(v): \\
        S2.remove(w) \\
    return g$.subgraph(S2) \]

Try $remove_vertex_and_neighbors(g,0)$. How come it didn’t do anything???

8. Now we are ready to write our new maximum stable set function. We will need two new vertex sets $S_1$ and $S_2$. There is a subtle issue to worry about. Can a function change one of its input parameters and how should we address this in out code?

So we should get a clear understanding of how a function affect or changes the objects that is is using as its parameters. Consider the following function. It takes a list $L$ as input and adds 9 to it. Let’s see if $L$ gets changed.

\[
def change_list(L):
    return L+[9] \]


9. Let’s also get a clearer look at what happens to $V$ and the graph’s vertices when we $pop()$ a vertex off of the end of $V$. Try: $g$.vertices(), then $V = g$.vertices(), then $v = V.pop()$, then $V$, and finally $g$.vertices().