Nelson Penn: Summary of Undergraduate Research

I began with building out implementations of matrix and polynomial template classes. These had to be compatible, which was non-trivial, especially because I was having problems with “double casting”: as with every mathematical field of numbers, vectors, etc., both the matrix and polynomial classes had to have a special number 1 and a special number 0. Thus, the matrix class had to be able to initialize a type T to 1 or 0. However, this was made difficult by the fact that the polynomial class was also a template class. The compiler can do one implicit cast at a time, and no more. So it was struggling with casting the 1 or 0 to the type used in the polynomial class (e.g. mpz\_class), and then casting that to a polynomial class. To solve this I had to provide special implicit casts from number literals straight to polynomials.

I added certain functions necessary for preparation for the testing of the project on counting embedded graphs (and for the generating function I had been using to test my classes). For example, I added the Hadamard product, the derivative, Taylor series expansion for two divided polynomials, classic polynomial long division, and some nice operator overloading for all the mathematical operations as well as printing. I hear that students in a separate section of CS 321 made similar matrix/polynomial implementations to this, but the polynomial class implementation was new (and very exciting, actually) to me, so the implementation took some time.

Finally, I also installed and configured the GMP library for infinite precision in my classes. This was quite difficult­. After several failed, indefinitely stopped, and messy installations as well as errant configurations, I got the library to be able to be used in my program. After that it was a matter of going through the GMP documentation to figure out how to either use the current interface or make my own so as to make it easy on myself to use the library. The documentation was not incredibly helpful, and for a while, I was scared this might be the largest part of the project so far; however, I found a very helpful Stack Overflow post that led me to using the mpz\_class and mpq\_class classes, rather than the mpz\_t, mpq\_t, etc. types. This saved a lot of work, because it takes a lot more understanding of GMP’s specific implementation to be able to use the mpz\_t and mpq\_t types, seeing as they are strange, single-element arrays- but aren’t actually used as arrays. Not only that, but they cannot be passed around between functions. This would have meant I would have either had to figure out how to make my own interface, or rework absolutely everything. But the mpz\_class and mpq\_class (etc., the last letter in mpx is usually the name for the set of numbers: integer, rational, complex, ...) act exactly like you would want the infinite precision to work: exactly like any old primitive type in C++, with appropriate casts. They also have helpful functions, such as mpq\_class∷canonicalize, which simplifies fractions for you. After finding these classes, I was able to merely replace all occurrences of “long” in my program with “mpz\_class” or “mpq\_class” and it worked perfectly! This is what I have done up to now.

The next steps I need to take are mostly learning ones. What I need most next is to understand what exactly the graph embeddings are, and the general basis of the 1996 project. I have done some research in the past weeks on the graph embeddings; however, I have been unable so far to grasp what graph embeddings are in a concrete sense. I have found a lot of general information, but not many examples or specifics. I need to continue my search and find a resource that will give me concrete examples so that I can count these embeddings of a small graph onto a manifold (? I believe this project was counting embeddings into a 2-manifold or a manifold of dimension 2, I am not entirely sure). Also, I will provide an implementation of the method for counting these embeddings as presented in the paper.