ECE 421: Assignment 1
Sparse Matrix Library

Group Member Name	Student ID
Riley Dixon	1425251
Aaron Espiritu	1462698
Jayden Kohlman	1458293
Shang Chen	1432808

1) What is a sparse matrix and what features should it possess?

A sparse matrix is a matrix whose elements are mostly zeroes. A feature that a sparse matrix possesses is the “sparsity” which describes the relative presence of zero-elements compared to total elements in the matrix. It should also be easily compressible as well as having quicker computations due to the zero elements being omitted during computations.

2) What sources of information should you consult to derive features? Do analogies exist? If so with what?

To determine basic functionality, we would consult existing matrix libraries and implement the relevant functions and operations for sparse matrices. Sparse matrices should be able to do regular matrix operations.

3) Who is likely to be the user of a sparse matrix package? What features are they likely to demand?

Sparse matrices are frequently used in solving partial differential equations as well as in machine learning applications. Practically any field that relies heavily on linear algebra operations will deal with using sparse matrices. For instance, Computer Graphics uses a lot of matrix multiplication. Services that use information retrieval (Amazon’s Recommended For You system) also use sparse matrices.

https://github.com/amzn/amazon-dsstne/blob/master/FAQ.md

4) What is a tri-diagonal matrix?

A tridiagonal matrix is a square matrix with nonzero elements along and immediately adjacent to the diagonal. All other elements in the matrix are zero.

5) What is the relationship between a tri-diagonal matrix and a generic sparse matrix?

A matrix is considered a sparse matrix when it has more zero elements than non-zero elements in the matrix. Given a tridiagonal matrix with length “n”, it will be considered sparse for lengths n >= 6.

6) Are tri-diagonal matrices important? And should they impact your design? If so, how?

Yes, they will be important to our sparse matrix library and we should consider them in our design. If we know that a matrix is a tri-diagonal matrix, we could store the diagonals into 3 separate lists. Linear algebra computations are typically easier to compute for diagonal matrices, so we should consider identifying these matrices in our programs and adjust our methods accordingly.

7) What is a good data representation for a sparse matrix?

A good data representation for a sparse matrix would be one with a high sparsity. In other words, sparse matrices have the main advantage of reducing storage space by assigning positions to non-zero elements and ignoring the zero elements. Higher sparsity means less storage space is being used and computations become simpler. A good data representation is one where computations can be executed efficiently.

8) Assume that you have a customer for your sparse matrix package. The customer states that their primary requirements as: for a N x N matrix with m non-zero entries.

o Storage should be ~O(km), where k << N and m is any arbitrary type defined in your design.

o Adding the m+1 value into the matrix should have an execution time of ~O(p) where the execution time of all method calls in standard Ruby container classes is considered to have a unit value and p << m ideally p=1.

In this scenario, what is a good data representation for a sparse matrix?

If the client is concerned about inserting new values into the matrix quickly, then the Dictionary Of Keys (DOK) data representation is ideal for inserting on average O(1) time. The storage requirements follow a complexity of O(3m) plus the memory required to setup the dictionary container. For each nonzero element, the row & column locations are used as the keys and the value is the non-zero element itself.

9) Design Patterns are common tricks which normally enshrine good practice

- **o Explain the design patterns: Delegate and Abstract Factory.**
- **o Explain how you would approach implementing these two patterns in Ruby.**
- **o Are these patterns applicable to this problem? Explain your answer! (HINT: The answer is yes)**

Delegate - A request is **delegated** to a helper object which maintains the same context as the original object.

Abstract Factory - This pattern implements a factory which is used to create other factories.

10) What implementation approach are you using (reuse class, modify class, inherit from class, compose with class, build new standalone class); justify your selection.

Compared to the conventional matrix, the nature of sparse matrices means that the representation of the matrix may be completely different from conventional matrix representation in order to enable optimization of a lot of computational methods. This means that inheritance, for example over a default Matrix library may not be a good idea since most of the methods inherited will not work. For this reason, it may be a better idea to write a new standalone class, composed with other helper libraries as it allows us for maximum flexibility.

11) Are iterators a good technique for sparse matrix manipulation? Are "custom" iterators required for this problem? (HINT: The answer is yes)

Yes because default iterators work on one-dimensional lists and depending on the data structure used to represent a sparse matrix, it may not be suitable. Since iterating over matrices are a huge component to many of the default operations on matrix, it makes sense to write a custom iterator to optimize this process. For example, instead of iterating through every element, we can opt to only iterate through non-zero elements.

12) What exceptions can occur during the processing of sparse matrices? And how should the system handle them?

Sparse matrices should have the same constraints for processing as regular matrices. For instance, the dot product between two matrices with incompatible sizes should be impossible and throw an exception.

We will assume that any and all matrices being used for our sparse matrix library are sparse. If the user decides to execute operations with dense matrices our system will not throw an exception, but rather process and store the dense matrix as a sparse matrix.

13) What information does the system require to create a sparse matrix object? Remember you are building for a set of unknown customers – what will they want?

Information depends on the chosen storage structure for representing the sparse matrix. For example, Internally the CSR method requires that we only know the values of all non-zero entities, the column index of each non-zero entity, and the number of non-zero entities of every row + the number of non-zero entities for the row before it.

Other information that the system needs to know includes that *data type* contained in the sparse matrix object. It goes that some users may want representations with ints, and some users may want to store floats. It is important to know this information as depending on the data type, we can save a lot of memory usage.

Depending on the complexity of the library, we may also want to require the storage structure used to represent the Sparse matrix. This is because different storage methods yield different benefits so a user may choose to select their storage method based on their needs.

14) What are the important quality characteristics of a sparse matrix package? Reusability? Efficiency? Efficiency of what?

Storage efficiency is important for representing a sparse matrix. It goes that a sparse matrix is a matrix whereby its density is < 0.5. This means that most of the values in the matrix, are zeroes, and therefore it can be redundant to store them. A good storage structure such as CSR CRS can significantly reduce the amount of data needed to fully represent the matrix - and therefore the amount of memory needed. This is especially important in applications where the size of the sparse matrix is large where naive storage methods (representing every value in the matrix) will simply break the computer.

Computational Efficiency is also important because due to the number of zeroes in a sparse matrix, it means that a lot of computations can be simplified down (such as X+0 = X, or X*0 = 0). Any library that does not simplify its computations will be at a significant disadvantage because *other libraries will have done it*, and therefore will be slower and computationally more expensive to run in comparison.