Sparse Matrix Design Document

The implementation of this Sparse Matrix Library is inspired by the Abstract Factory Pattern. Using the *Factoryproducer* class, (here our abstract factory), we can return a Csrfactory Object or DOKfactory Object using the method *get_factory()*.

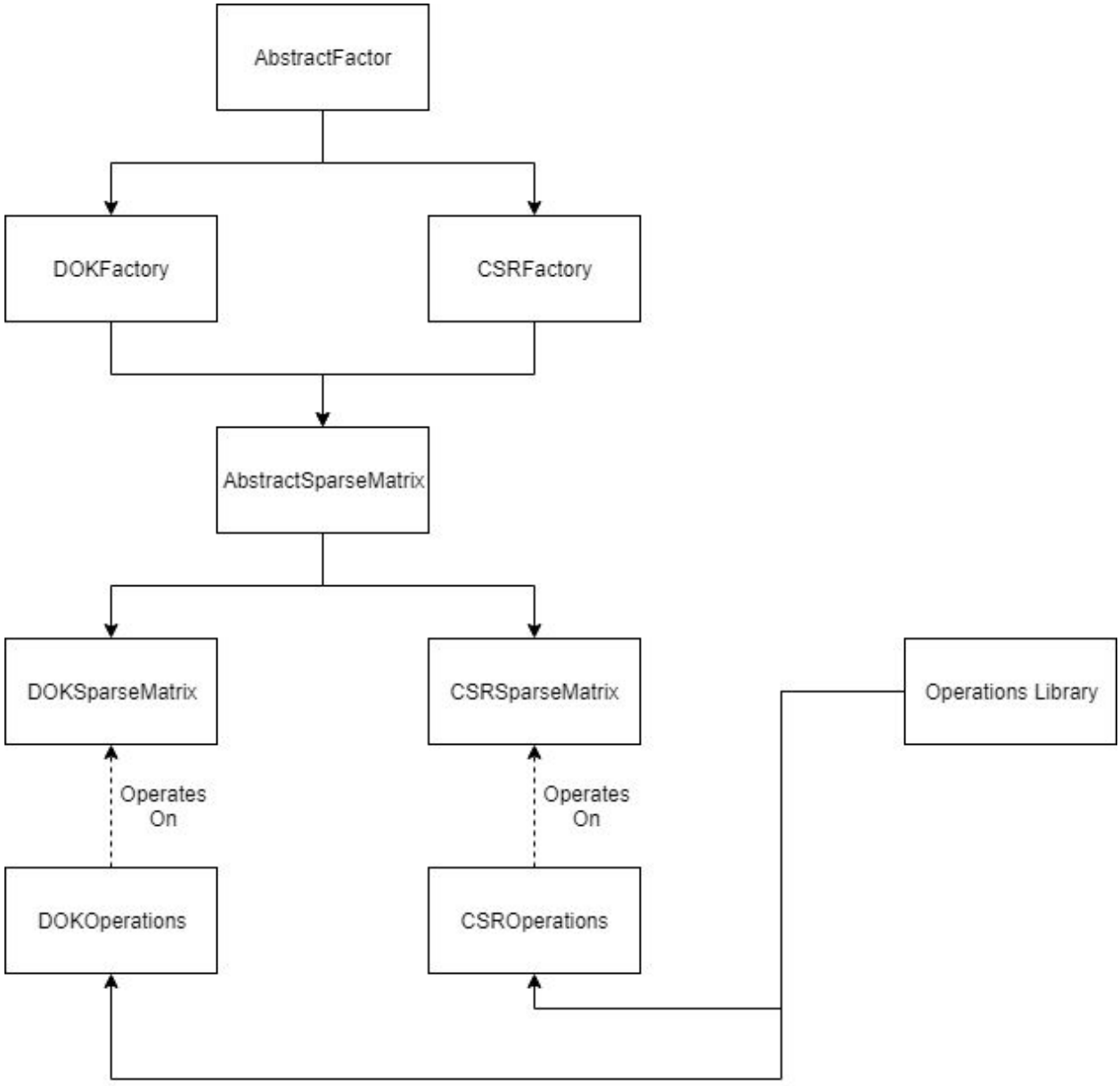
There are multiple ways to describe a sparse matrix. The Dictionary of Keys (DOK) method for example uses the positional coordinates of the element in the matrix as the key with the element itself being the value in the dictionary. Another format is Compressed Sparse Row (CSR) or more commonly known as the Yale format, stores 3 seperate arrays. The first array contains the values of each element in row-major format. The second array keeps a tally of how many non-zero elements have appeared since the end of the row. The third and last array keeps track of the column of each element. These are the two methods used in our library. However, in order make our library more expandable in future versions we will implement the Abstract Factory design method.

The Abstract Factory design will pass different factories that can construct the various different types of sparse matrices. Doing it in this method allows the type of matrix that will be constructed by simply changing the parameter passed to the factory with no other underlying code have to be changed. Each factory will then return its corresponding sparse matrix object. The Factory will be able to return a SparseMatrix object from a variety of inputs. These inputs could be a blank matrix, a 2D-array, a Matrix object, another matrix of the same type, or creating the identity matrix for that type of SparseMatrix. This also allows the factory to be expanded on to create matrices from other objects (e.g. create a DOK matrix from a CSR matrix or vice-versa).

Each sparse matrix object will be a subtype of an abstract class SparseMatrix. Each SparseMatrix subclass should implement methods to get and set elements in the matrix. Additional implemented methods can return if the matrix is of a special type (identity, tridiagonal, sparse, non-sparse, zero, etc.) as well as a function that returns the sparsity of the matrix. Each matrix class however should not be responsible for performing any kind of mathematical operation. This will be left to a separate Operations class that will perform the various mathematical operations. If time permits, an Iterator will be constructed to iterate over the elements in each SparseMatrix, which will include information such as the element value and its position in the matrix.

The various matrix operations that will be implemented will be as follows:

Scalar Operations: Addition, Subtraction, Multiplication, Division, Exponentiation
Matrix Operations: Addition, Subtraction, Multiplication, Division
Taking the inverse of the Matrix
Transposing the Matrix
Finding the determinant of the matrix


AbstractFactor
DOKFactory
CSRFactory
AbstractSparseMatrix
DOKSparseMatrix
CSRSparseMatrix
Operations Library
Operates On
Operates On
DOKOperations
CSROperations

Modified Design Document

Our final product did not change a whole lot from what was promised in the final product. However, we did not have the opportunity to implement the methods that identified the special types of matrices (zero, identity, sparse, etc). Some internal changes were made in an effort to increase performance.

Aside from the assert statements that were implemented as part of our contracts, we did not do any other formal testing as part of this assignment.

Between the four of us, there are not any known errors to report besides the missing functions.