Assignment 1
Group 1
Part 1

January 30, 2018

Bianca Angotti (bangotti 1417422)
Andrew McKernan (amckerna 1432175)
Arjun Kalburgi (kalburgi 1388774)

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

- A sparse matrix is a matrix where most of the elements are 0. It is “sparsely” populated. There can be different degrees of sparsity, but to be identified as a sparse matrix this degree would have to be relatively high. Because of its sparsity, sparse matrices can be more efficient during operations and more concise in terms of memory usage. Since much of the matrix is 0, much of the matrix operation doesn’t have to be computed and only elements and their indices need to be stored.

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

- Users and use cases of sparse matrices would be good sources to learn about what the features of a sparse matrix should be. Matrices representing weaker image masks or terrain maps of a relatively flat land would be sparse matrices. So it’s also possible that packages and programs used to handle matrices in these fields can be good sources to derive features. An analogy would be a car manufacturer looking at their customer base, past models, as well as what other manufacturers have done when making similar vehicles to decide on a design for their latest model of car.

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

- Mathematicians working with sparse linear algebra
    - Require that operations done using the array be optimised as much as possible
    - Require that regular array operations can be done on sparse arrays
- Computer engineers and data scientists working with and storing data in the form of an array
    - Require that space taken to store the matrix is optimized
    - Require that sparse matrices can be differentiated from regular matrices (whether or not you care the data is 0 - doing operations with 0)

What is a tri-diagonal matrix?

- “In linear algebra, a tridiagonal matrix is a band matrix that has nonzero elements only on the main diagonal, the first diagonal below this, and the first diagonal above the main diagonal.” (Wikipedia)
- A TriDiagonal matrix is a band matrix with non-zero elements only in the main diagonal, and the first diagonals below and above this line. Note that the matrix is square (nxn).

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

- A tri-diagonal matrix is a sparse matrix, with a particular pattern.
- Understanding the structure of a sparse matrix allows for algorithms to make them more efficient. Tri-diagonal's pattern is an understanding of a sparse matrix.

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

- Tri-diagonal matrices are important because they allow for algorithms to make them more efficient. Considering the patterns when designing the sparse matrix package is important for making a better program for the use cases.
- If the tri-diagonal matrix pattern is found, the sparse matrix package should use operations and storage functions that are optimized for the tri-diagonal pattern.

Can complex numbers exist in matrices?

Complex number can exist in matrices. For example, this could a matrix like the following:

$$\begin{pmatrix} 1 & 2i \\ -2i & 1 \end{pmatrix}$$

They can also be represented in the form of a matrix. For example:

$$a + ib \mapsto \begin{bmatrix} a & -b \\ b & a \end{bmatrix}$$

What is a good data representation for a sparse matrix?

- There are two good data representations for sparse matrices, triplet representation and linked representation.
- Triplet representation considers only non-zero values of the matrix by holding them in a table with three columns. The first two columns of the table is for the value's row and column in the matrix, and the final column is for the value itself.
- Linked representation uses a linked list and is much more complicated. The linked list has two different types of nodes, the header node and the element node, which have different structures. The header nodes represent indexes and have 3 values associated with them, the index value, and pointers to elements to the immediate right and below. The element nodes represent values of the matrix and have 5 values associated with them, the row and column of the value, the value and pointers to the elements to the immediate right and below/above.

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

Storage should be ~O(km), where k << N and m is any arbitrary type defined in your design.
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
Provide a detailed analysis to demonstrate that your initial abstract design means these requirements.

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

- A good data representation for a sparse matrix would be a Hash in Ruby. The insert time into a Hash is nearly O(1), since all that is needed to insert a key-value pair is to hash the key and insert it with its corresponding value. Searching is also much faster here. Storage would also be O(km), since we only need to store the k non-zero elements in the Hash, which should be << N. We can easily define a type that can represents the coordinates of an item in the array that will serve as the Hash key, and store the value of that space with it.

Design Patterns are common tricks which normally enshrine good practice
Explain the design patterns: Delegate and Abstract Factory
Explain how you would implement these two patterns in Ruby
Are these patterns applicable to this problem? Explain your answer! (HINT: The answer is probably yes) If the answer is yes, your implementation must reflect this.

- **Design patterns**
  - **Delegate:** The delegation pattern allows object composition decouple tasks and literally “delegate” the work to another class. This can thereby hide complexity from the user of the class and delegate the work of a method to the actual worker itself. For example, a House class could have a increaseHeat() or decreaseHeat() method, but within those methods, they would actually call the furnace or air conditioning systems to use their methods.
  - **Abstract Factory:** This pattern allows us to efficiently encapsulate individual factories with a common theme, but with an abstract class rather than a concrete one. For example, an abstract House class can be created with methods pertaining to a house, but system will have various concrete versions of a House such as a MansionHouse or MobileHouse.

- **Implementation in Ruby**
  - To implement the delegate pattern in Ruby, first make the delegate class. Then make the functional classes that the delegate class will use (as variables in the delegate class). Any other classes that need to use the functional classes will use them through the delegate class.
  - To implement the abstract factory pattern, first make the factory class which is an abstract class with an abstract method. Classes then implement this class and define the function in their own, unique way.

- **Applicability**
  - Yes, these patterns are applicable to this problem. The Abstract Factory method is applicable because it allows us to create different types of sparse matrices in an object oriented fashion - if a sparse matrix containing a certain type of data needs a special consideration, we can easily create a factory method for it rather than redefining the entire definition of the sparse matrix.
  - Similarly, the Delegate design pattern allows us to abstract much of our data processing to other more qualified classes. If a unique type of data is being modified

or inserted into the sparse matrix, rather than writing our own code to handle this case ourselves, we can delegate it elsewhere to code that is hopefully better written, well tested, and more accurate.

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

- Our sparse matrix package follows the factory design pattern. This way the different implementations of the sparse matrix can share code from the factory class.
- The sparse matrix as implemented by a hash will implement the Hash package
  Hash package: https://docs.ruby-lang.org/en/2.0.0/Hash.html
- We decided to inherit the Matrix class into our TriDiagonal and Sparse matrix classes. These classes are both Matrices, they just implement certain functionalities differently so that they can optimize certain operations. It is because of this that we decided to use inheritance.
  Matrix package: https://docs.ruby-lang.org/en/2.0.0/Matrix.html

Design by Contract assumes that programmers understand at least basic testing theory. Explain with examples using sparse matrices:
Equivalence Partitioning
Boundary Value Analysis
Statement and Branch Code Coverage.
Provide a second set of examples where the input is non-numeric; e.g. testing a database of people, where people are defined by name, address, telephone number, occupation, and a list of pastimes.

- **Equivalence Partitioning**
  - Partition the input space by what will give similar outputs. Each partition will go through similar processes within the function this testing by partition means you test the proper functionality of the program completely, with a minimum number of test cases.
  - **Numeric:** Assuming the same database example from the question, using equivalence partitioning to test phone numbers would be simple. We can make partitions of valid/invalid digits. Another example would involve equivalence classes of square matrices vs rectangular
  - **Non-numeric:** Lots of complexity is added for non-numeric tests, but a breakdown of equivalence classes could be grouped by the definitions of people. Hence tests for matrix addition, for example, could be grouped for equivalence classes by name, address, telephone number, occupation and a list of pastimes.
- **Boundary Value Analysis**
  - Boundary value analysis focuses on the edge cases of a test, ensuring that the extreme inputs do not break the system. The boundary value analysis exists at the edges of input equivalence classes. There can be closed and open boundaries, where boundary points may or may not belong to the sub-domain.
  - **Numeric:** Building upon the last database example with phone numbers, the maximum phone number and minimum number could be tested. Such as 0 and 9999999999 depending on the maximum telephone number set by the system. Then

the adding of two matrices could be another test that ensures this boundary of phone numbers is not broken.
    - **Non-numeric:** For non numeric boundary value analysis, the boundary ASCII values could be tested in a system that processes ASCII characters to ensure that no unexpected behaviour is observed at these boundaries.
- **Statement and Branch Code Coverage**
    - Statement coverage is a requirement of testing meaning that all lines of code must have been executed at some point during the test suite. It does not require that all possible paths through the code are taken, and does not imply branch coverage.
    - Branch coverage is a requirement of testing meaning that all possible logical "paths" that can be taken through the code are taken at least once. A state diagram is often used to determine all the possible paths through a program.
    - **Numeric:** For statement coverage, a numeric example would be testing a few matrices of various sizes, ensuring that there is proper looping to store the matrix in the sparse matrix format.
    - **Non-numeric:** For non-numeric branch coverage, an example would be inputting multiple strings into a program meant to test a password's strength, including strings with all lowercase words (weak passwords), strings with numbers and letters (medium passwords) and strings with capital letters, numbers, and special characters (strong passwords).

Is iteration a good technique for sparse matrix manipulation? Is "custom" iteration required for this problem?

- Custom iteration would be required for sparse matrix manipulation. Iterating through each element of the sparse matrix would defeat the purpose of using a sparse matrix. By utilizing a sparse matrix, we can optimise looking through the matrix since we know that most of the elements are zeroes. The "custom" iteration that should be used here is iterating through the non-zero elements of the matrix, and mapping each element to their proper coordinates in the matrix.

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

- A matrix could be created that has a size <= 0.
    - Handled by checking size before creation and acting accordingly
- An attempt to access an item out of range of the matrix index could be made.
    - Handled by checking the size of the matrix against the intended index and acting accordingly
- Illegal data types could be added into the matrix that are different from the matrix elements.
    - Check that the data being inserted is of a valid type, both on edit or on creation

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?

- The type of data or structure of data being stored in the sparse matrix.
    Ex. Strings, integers, hex for colors, arrays, objects, etc.
- The dimensions of the matrix.

Ex. 2x2 matrix, 5x100 matrix, etc.

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

- Correctness is important, since when we convert a regular matrix into a sparse matrix, we do not want to lose any data quality that is present in the old array.
- Portability is also valuable so that the consumers are able to utilize the program across various platforms.
- Flexibility is important as additional functionality and operations can be added to the program without breaking the system.
- Reusability is important because the customer is unknown. The ability to convert our sparse matrix package for use by different customers (possibly different data types) is important.
- It is integral to have an efficient sparse matrix package. Without creating optimizations to utilize the properties of sparse matrices, the package is essentially useless as it treats the matrices as any other, executing operations over useless zero values.

How do we generalize 2-D matrices to N-D matrices, where n > 2. Sounds like an extensible design?

- We can extend the design to generalize matrices to higher dimensions by adding to our data representations. For triplet representation, we'd add more columns for the additional dimensions and for linked representation, we'd add more header nodes and room for more indices in all nodes (n=3 should have room for 3 indices in the nodes).