ECE 421 Assignment 1 Part 1

Winter2019_Group4:

Nathan Klapstein (1449872)

Tony Qian (1396109)

Thomas Lorincz (1461567)

Zach Drever (1446384)

Table of Contents

Abbreviations

CSR Compressed Sparse Row
DoK Dictionary of Keys

Introduction

Sparse matrices are matrices that mostly contain zero values. Due to the heavy repetition of zero values within such matrices, they can be implemented to require less storage and be optimized for some faster matrix arithmetic operations. However, the programming language Ruby contains no native package to handle such matrices. Thus, within this report a requirements discussion is made to develop a sparse matrix library in Ruby.

There are many sparse matrix representations that allow for memory compression and computational efficiencies, and each has their own strengths and weaknesses. Some of these representations include Compressed Sparse Row (CSR), Dictionary of Keys (DoK), Diagonal matrices, Block Sparse Matrices or Linked List Matrices. For most of these alternative matrix data structures, optimizations are typically only realized for sufficiently sparse matrices due to the need to store matrix meta-data (entry indices). For example, every non-zero value in a DoK matrix representation requires storage of the (row, column) coordinates as well as the value. This means that memory compression is only attained when the frequency of non-zero values in the matrix is less than 0.33. Just like many software problems, selection of the right data structure is essential to realizing optimality.

Part 1: Design Questions

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

A sparse matrix is any matrix where more than half of its entries are zero values. Since arithmetic operations on zero produce zero or identity values, optimizations can be made for arithmetic operations performed on sparse matrices.

Sparse matrices should be more spatially efficient to store in computers than matrices stored in a dense fashion because values of zero are assumed and do not need to be explicitly stored. It is also expected that sparse matrices should be able to compute resultant matrices faster than regular dense matrices because many of the results of arithmetic will be zero or unchanged (e.g. adding zero, multiplying by zero, etc.).

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

To learn more about sparse matrix features and they are used in computing, computer science textbooks and online lecture slides can be consulted for theory. As well, there are sparse matrix libraries in other programming languages like Python that can help to inform us about what operations users would expect our library to support.

Ruby's standard Matrix class be used as a baseline for expected features for Ruby matrices. This way, Ruby users who are familiar with working with dense matrices in Ruby will find a familiar interface in our library.

An analogy to describe the advantages of sparse matrices could be drawing a black-and-white image. If you are drawing on white paper, you only need to draw the black parts of the image. It would waste time and drawing material to draw the white parts (and for little gain). The same way, it doesn't make sense to store and process zero values in a computer. These values can be assumed in storage and arithmetic with zero produces predictable results.

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

Mathematicians, statisticians and data analysts/scientists are the most likely users of this package. In general, most non-scientific developers typically would not have a need for a sparse matrix library. Those that are attempting to leverage the sparse matrix library would certainly demand optimized matrix operations and storage for sparse matrices. Developers that are familiar with the standard library matrix.rb would likely assume conformity with their API, since sparse matrices are matrices.

4. **What is a tridiagonal matrix?**

A tridiagonal matrix is a matrix that has only nonzero elements on the main diagonal, the diagonal above it, and the diagonal below it.

5. **What is the relationship between a tridiagonal matrix and a generic sparse matrix?**

Tridiagonal matrices are a subset/subclass of generic sparse matrices. Similar to sparse matrices, when they are sufficiently sized, they can benefit from a custom storage scheme that only keeps track of nonzero entries.

6. **Are tridiagonal matrices important? And should they impact your design? If so, how?**

Not really. They have interesting mathematical properties, but are mostly interesting because of their efficient storage and optimizable computations. However, all diagonal matrices share these benefits, making tridiagonal less worthwhile to implement as a standalone class.

We do not intend to focus on tridiagonal matrices specifically in our design. Instead, we are going to define a generic diagonal matrix class where the bandwidth of the main diagonal can be specified.

7. **What is a good data representation for a sparse matrix?**

The answer to this question depends on what the user is looking to do with the sparse matrix. In general, Compressed Sparse Row (CSR) representation would work well because matrices are most often used for operations like the dot product. However, if the user is not interested in matrix arithmetic, other storage options like Dictionary of Keys (DoK) would be more favourable because accessing subsets of their entries and inserting new entries are more optimized than CSR implementations.

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

A sparse matrix that uses DoK storage satisfies these design constraints. Storage is O(km) where k=2. Insertion is O(p) where p is the unit value of dictionary insertion in Ruby.

9. **Design Patterns are common tricks which normally enshrine good practice.**

    9.1. **Explain the design patterns: Delegate and Abstract Factory.**

Delegate pattern allows objects to achieve code reuse similar to inheritance. In this pattern, requests made to a particular object are “delegated” to a helper object. This can be done by passing the original object to the method called in the helper object, as per the explicit pattern. Simple inheritance from a superclass is also considered a form of delegation.

Abstract factory is essentially a factory for factories. Abstract factory interfaces allow us to create factories for related objects, without explicitly declaring their class. These created factories are able to generate objects adhering to the classic factory pattern.

The classic factory pattern should encompass the boilerplate or baseline work for constructing a sparse matrix that is common for all sparse matrices.

    9.2. **Explain how you would approach implementing these two patterns in Ruby**

For the delegation pattern we could make use of an all-encompassing superclass that wraps all classes created by our abstract factory. We could store our reusable, shared methods

in this superclass which could be accessed from all objects created from its subclasses. We could also control singleton objects in our library which contain methods that take in a object as an argument, and perform actions that are useful for a wide variety of classes.

9.3. Are these patterns applicable to this problem? Explain your answer!

Yes, we will likely be using different classes to store our matrices.

Abstract Factory pattern will allow us to easily build objects that are suitable for each different type or classification of sparse matrix.

The delegation pattern comes into use when we need to perform actions for a certain type of matrix, where the the functionality is used in other matrices as well. This could be done via explicit delegation to a polymorphic singleton object containing methods useful to many classes. It could also be done with an all-encompassing superclass that is a parent to all classes served by our abstract factory.

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

We will be using the “inherit from class” implementation approach to create a sparse matrix implementation. The ruby standard library matrix.rb Matrix class will be the superclass that acts as the basis for the sparse matrix implementation. This was chosen, as the majority of users of a sparse matrix would appreciate the ability of such a object to be a direct drop-in replacement of ruby’s built in implementation. Under the hood the sparse matrix behaviour may be different, however externally, its behavior should be exactly the same as a normal matrix. As such the contracts defined for a sparse matrix are exactly the same for the external requirements.

Additionally using the standard library as a basis helps avoid code duplication, and reinventing the wheel. As such using the expected behavior of the built-in matrix implementation avoid re-thinking requirements specification.

Also using the Ruby standard library matrix.rb Matrix as a superclass allows us to use it as a testing oracle. Thus, using the comparisons between the built-in Matrix class and the sparse matrix implementation as a test component.

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

Yes, custom data structures benefit substantially from having custom iterators. Custom iterators are written to be optimized based on the underlying storage scheme of the objects that they iterate over.

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

Sparse matrices will require that their their entries are all Numeric types. Values that are not Numeric types are checked during object construction and will throw an error.

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?**

Customers familiar with Ruby Matrices will likely assume that similar constructors would be available for sparse matrices. The Ruby standard library supports construction of matrices by passing arrays of arrays, construction of specialty matrices, and construction through the elementwise API .build(). Construction of sparse matrices through all of these methods should be possible. As well, construction of sparse matrices by passing normal ruby matrices should also be possible.

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

Most mathematics utility libraries focus on speed and require that data given to them follows a rigid format (e.g. numpy requiring conversion to its Array object for most uses). This leads to a focus on Efficiency over Reusability.

Efficiency of the compactness/compression of a sparse matrix is important within a sparse matrix package. Otherwise, traditional matrix libraries could be used in place instead.

References

General mathematical definition of a sparse matrix
https://en.wikipedia.org/wiki/Sparse_matrix

Ruby delegation patterns
http://radar.oreilly.com/2014/02/delegation-patterns-in-ruby.html

SciPy sparse matrices documentation
https://docs.scipy.org/doc/scipy/reference/sparse.html

Numpy mathematical python package
http://www.numpy.org/

Ruby standard library matrix definition
https://github.com/ruby/matrix
https://ruby-doc.org/stdlib-2.5.1/libdoc/matrix/rdoc/Matrix.html

A paper on efficient CSR matrix operations:
https://www.mcs.anl.gov/papers/P5007-0813_1.pdf