Abstract

Due to the high usage, of matrix manipulation by engineers, and that most of these matrices are sparsely populated. In the addition to the fact that ruby has no library for a sparse matrix. The objective of this project was to create a sparse matrix package in ruby.

Introduction

- What is a sparse matrix and what features should it possess?
    - A sparse matrix is a matrix whose elements are mainly zero. Namely there's a majority of elements with a value of zero. Features it should possess are addition, subtraction, multiplication, and division with other matrices or values. It will also have other matrix functions like determinate, inverse, and transpose.
- What sources of information should you consult to derive features? Do analogies exist? If so with what?
    - Some sources consulted are Wikipedia. As far as we can think of, there isn't any analogies we can think of.
- Who is likely to be the user of a sparse matrix package? What features are they likely to demand?
    - Users of a sparse matrix package might be programmers, engineers, mathematician, scientist. The most likely features may be the determinate, transpose, and inverse functions.
- What is a tri-diagonal matrix?
    - A matrix where on a diagonal band of three lines, non-zero numbers are only on the lines.
    - Properties of a tri-diagonal matrix with the assumption that it is a square:
        -Number of non-zero = n + 2(n - 1) = 3n - 2
        -Number of zeros = n^2 - non-zero = n^2 - n - 2(n -1) = n^2 -3n + 2
- What is the relationship between a tri-diagonal matrix and a generic sparse matrix?
    - Assuming the matrices are nxn and using the above equations and definition of a sparse matrix
        - # of zero > # of non-zero
        - n^2 -3n + 2 > 3n -2
        - n^2 -6n + 4 > 0
        - the roots of the left are (3 +/- sqrt(5))
        - therefore, a tri-diagonal matrix becomes a generic sparse matrix when the size is greater than 6
- Are tri-diagonal matrices important? And should they impact your design? If so, how?
    - Yes, they are important. They are used to solve linear system of equations. They can be used to solve partial differential equations as well.

- It impacts the design because if the size is greater than 6, we then know it is a subset of a sparse matrix
- If two tri-diagonal matrices of the same size and are a sparse matrix and are added or subtracted together, the result is always a sparse matrix. Avoiding any kind post condition checks.

Design

- What is a good data representation for a sparse matrix?
  - Methods that could be used to represent the data is compressed sparse row (CSR) and compressed sparse columns (CSC)
- 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 good data representation might be to use a hash to store all the non-zero values. Using the value's position as the key and the value as the value. It will meet the requirements for execution and adding values to the matrix. The only down side to it compared to a CSR/CSC format is the storage space.
- Explain the design patterns: Delegate and Abstract Factory
  - Using composition to avoid code reuse. This requires essentially two things: first, incorporating other objects and classes in a class. This is called composition. Second, responsibility is passed from the original class, to methods of the classes and objects that are used in composition of the original class. This is called delegation.
  - Basically, a tool for making a group of related objects (using factories) but without making concrete instances itself. The Group of related objects would be concrete factories that could instantiate concrete objects.
- Explain how you would approach implementing these two patterns in Ruby
  - We can introduce a form of multiple inheritance with mixin modules. Mixin is a feature of ruby where you use the 'include' keyword to easily include definitions of all the methods of that module in that class. The classes can mixin a series of modules for composition by including them. The modules would handle the delegated responsibilities.
  - We would use the inheritance operator '<'. To mimic the abstract aspect, we can have the parent abstract class raise exceptions in the methods that would be inherited in the concrete factory subclasses

- Are these patterns applicable to this problem? Explain your answer! (HINT: The answer is yes)
    - The Delegate pattern could be used transfer responsibilities. For example: tri-diagonal matrices with dimensions NxN where N larger than 5, could delate responsibility to sparse matrix class that would be used in the tri-diagonal class with composition
    - The Abstract Factory pattern can be used to create generic matrices. The concrete factories and classes that would be instantiated could be Dense matrices, sparse matrices, tri-diagonal matrices
- What implementation approach are you using (reuse class, modify class, inherit from class, compose with class, build new standalone class); justify your selection.
    - The implementation approach that will be used is composition. The defined matrix class in ruby will be used in the implementation of the sparse matrix. Some operations will have a matrix object to speed up the computation. A new standalone sparse matrix class will be implemented that uses the matrix class.
- Is iteration a good technique for sparse matrix manipulation? Is “custom” iteration required for this problem?
    - Yes, iteration would be a required technique for sparse matrix manipulation when adding, subtracting, multiplying, or division with values.
    - Custom iteration methods would be required for other matrix operations such as adding, subtracting, multiply, division, determinate, inverse, and transpose.
- What exceptions can occur during the processing of sparse matrices? And how should the system handle them?
    - a sparse matrix could potentially break the definition of a sparse matrix after operations such as addition or subtraction. The post condition of the operation will detect that the conditions for a sparse matrix has been broken. It will then return the result as a matrix class instead of a sparse matrix class. If the object that is being assigned the result is a sparse matrix it will then check its invariants to determine if they have been broken or not. If they are broken, the invariant method of the sparse matrix will then throw an error indicating that the invariants have been broken.
- 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 to create the sparse matrix would be the size of the matrix, and the values inside of the matrix.
- What are the important quality characteristics of a sparse matrix package? Reusability? Efficiency? Efficiency of what?
    - A sparse matrix package should be reusable as it should be able to be be used in a general matrix class that has a condition that is under a sparse matrix's

domain. Or it should be able to be modified to work as a dense matrix or any other matrix types.
    - The package should be efficient in that it should not take more resources than a simple matrix generally would. Nor should it take more time to run any operation than the simple matrix would.
    - The package should be usable. It shouldn't require a manual on how to use all the functions. It should be logical and consistent on how the functions are used.
    - The package should be reliable. It shouldn't return any incorrect return values.
    - The package should also be Extendable. It shouldn't be difficult to implement new functions into it that are required yet missing.
- How do we generalize 2-D matrices to n-D matrices, where n > 2 – um, sounds like an extensible design?
    - We will use another system for compressing n- D dimensional matrices that is not as efficient as CSR. Essentially it will use tuples for every non-zero values, along with its position in in each dimension of the matrices. Example: for a 3x3x3 or 3-D matrices. (Note it is depicted as 3 pages of 3x3 , 2-D matrices) [1 0 0] [0 8 3] [0 2 8] [0 2 0] [0 2 0] [1 0 0] [4 0 0] [0 0 6] [0 0 0]
      The tuples would be: (1,0,0,0) (2,1,1,0) (4,0,2,0) (8,1,0,1) (3,2,0,1) (2,1,1,1) (6,2,2,1) (2,1,0,2) (8,2,0,2) (1,0,1,2)

    - The heuristics are as follows:. While it may not be as efficient as CSR, as shown above, it is extensible to n dimensions.

Conclusion

Through some thorough thinking with the questions above, It can be said that it is possible to design a sparse matrix package in ruby.