ECE 421 Assignment 1 Part 2

Winter2019_Group4:

Nathan Klapstein (1449872)

Tony Qian (1396109)

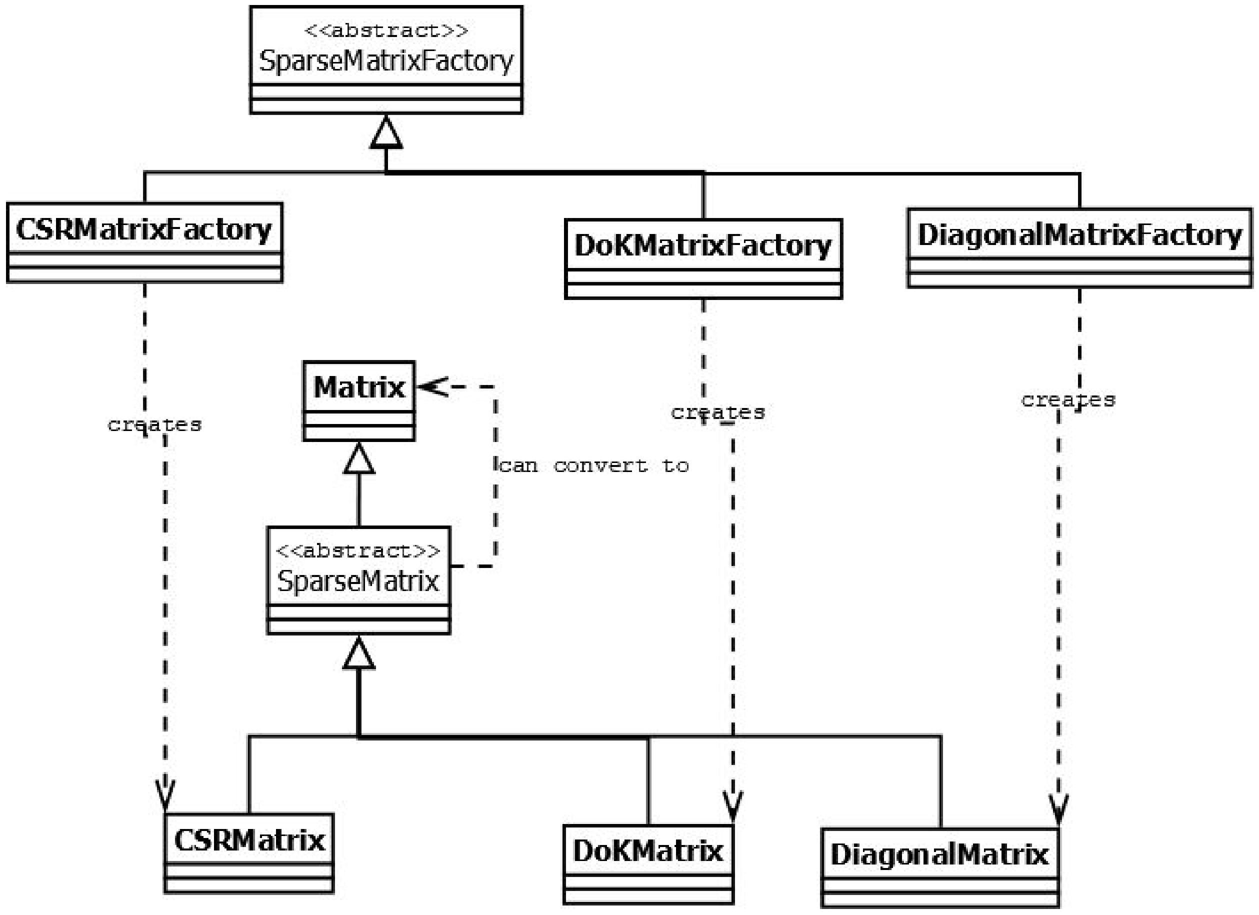
Thomas Lorincz (1461567)

Zach Drever (1446384)

Part 2: Design Proposal and Contracts

Class Diagram

Below is a class diagram that displays the abstract factory architecture that we plan to implement. Matrix is the Ruby standard Matrix class. We subclass Matrix so that our SparseMatrix can be used anywhere a Matrix can be used in Ruby.



Functionality (Design Contracts)

SparseMatrixFactory (Abstract Factory)

An abstract factory class that defines the factory methods that all concrete sparse matrix factory classes inherit.

Methods

(public) static build(*args)

args[0] is an array of arrays or a matrix, args[1] is the optional type.

(private) def build_from_array(array, type='csr')

If the first argument to build(*args) is an array, this method will be used by a sparse matrix factory to produce a SparseMatrix object from the array of arrays. This will call the appropriate SparseMatrix subclass constructor.

(private) def build_from_matrix(matrix, type='csr')

If the first argument to build(*args) is a Ruby Matrix, this method will be used by a sparse matrix factory to produce a SparseMatrix object from the matrix. This will call the appropriate SparseMatrix subclass constructor.

SparseMatrix (Abstract Superclass)

An abstract sparse matrix superclass with empty methods. All other sparse matrices inherit the method stubs of this class. In this way, assertions can be made about what functionality must be present in all sparse matrices.

A SparseMatrix is a subclass of the Ruby standard library Matrix (matrix.rb). Our plan is to implement method stubs for all of the most common matrix methods that can benefit from sparse matrix representation. Methods that can not serve to be improved by sparse matrix representation will be converted to a Ruby standard matrix through the to_matrix() method so that the superclass method can be called. After the superclass method is called, the resulting Matrix will be converted back to a SparseMatrix object and returned.

Class Invariants

-
  ```
  @nonzero_values.each do |value|  {
          assert(value.is_a?(Numeric))
  }
  ```

Fields

- **@column_count** - The number of columns in the matrix.
- **@column_size** - The number of rows in the matrix.
- **@rows** - The array of arrays representation of the matrix. This field will not store the full representation and will instead point to an accessor method that computes the array of arrays (this is to preserve the efficient storage of SparseMatrix instances).
- **@nonzero_values** - An array of the nonzero values stored in the SparseMatrix.

Methods

add(addend)

- Purpose: Return a new SparseMatrix that is the result of the matrix element-wise addition of the SparseMatrix with the addend.
- Pre
  - addend.is_a?(Matrix) || addend.is_a?(Numeric)
- Post
  - @nonzero_values.each_with_index do |val, i| {
    assert_equal(val, **old** @nonzero_values[i] + addend)
    }

subtract(subtractor)

- Purpose: Return a new SparseMatrix that is the result of element-wise subtraction of the SparseMatrix with the subtractor.
- Pre
  - subtractor.is_a?(Matrix) || subtractor.is_a?(Numeric)
- Post
  - @nonzero_values.each_with_index do |val, i| {
    assert_equal(val, **old** @nonzero_values[i] - subtractor)
    }

divide(divisor)

- Return a new sparse matrix that is result of the matrix element by element division of the self SparseMatrix with the parameter passed SparseMatrix.
- Pre
  - divisor.is_a?(Matrix) || divisor.is_a?(Numeric)
- Post
  - @nonzero_values.each_with_index do |val, i| {
    assert_equal(val, **old** @nonzero_values[i] / divisor)
    }

multiply(multiplier)

- Purpose: Return a new sparse matrix that is result of the matrix element by element multiplication of the self SparseMatrix with the parameter passed Matrix.
- Pre
  - multiplier.is_a?(Matrix) || multiplier.is_a?(numeric)
- Post
  - @nonzero_values.each_with_index do |val, i| {
    assert_equal(val, **old** @nonzero_values[i] * multiplier)
    }

power(exponent)

- Purpose: Return a new SparseMatrix that is the result of the matrix power operation
- Pre
  - exponent.is_a?(Matrix) || exponent.is_a?(Numeric)
- Post
  - @nonzero_values.each_with_index do |val, i| {
    assert_equal(val, **old** @nonzero_values[i] ** exponent)

}

***(matrix)**

- Purpose: Return a new SparseMatrix that is result of the matrix dot product of the self SparseMatrix with the parameter passed SparseMatrix.
- Pre
  - matrix.is_a?(Matrix)
  - assert_equal(@column_count, matrix.column_size)
- Post
  - assert_equal(@column_size, **old** @column_count)
  - assert_equal(@column_count, matrix.column_size)

to_matrix():

- Purpose: Return a copy of the self SparseMatrix as a Ruby standard library matrix.rb Matrix
- Pre
  - None
- Post
  - assert_equal(**new** Matrix.column_count, @column_count)
  - assert_equal(**new** Matrix.column_size, @column_size)
  - assert_equal(**new** Matrix.to_array, **old** SparseMatrix.to_array)
  - new Matrix.each_with_index do |row, i|| {
            row.each_with_index do |value, j| {
    assert_equal(**new** Matrix.element(i, j), **old** SparseMatrix.element(i, j))
            }
    }

transpose()

- Purpose: Returns the transpose of the SparseMatrix that calls it.
- Pre
  - None
- Post
  - assert_equal(@column_count, transpose.column_size)
  - assert_equal(@column_size, transpose.column_count)

  -
    ```
    @column_count.times do |i| {
      @column_size.times do |j| {
        assert_equal( element(i, j), transpose.element(j, i) )
      }
    }
    ```
  - assert_equal(Matrix.transpose, SparseMatrix.transpose.to_matrix)

==(matrix)

- Purpose: Returns true if and only if the two matrices contain equal elements.
- Pre
  - matrix.is_a?(Matrix)
- Post

        - return matrix.each_with_index do |val, i| {
          assert((val == nonzero_values[i]) || (val == 0))
          }

determinant()

- Purpose: Return the determinant of this matrix
- Pre
    - matrix.is_a?(Matrix)
- Post
    - return matrix.each_with_index do |val, i| {
      assert((val == nonzero_values[i]) || (val == 0))
      }

CSRMatrix (Subclass of SparseMatrix)

A SparseMatrix subclass that uses the Compressed Sparse Row (CSR) storage scheme to improve efficiency in storing zero values.

Fields

- **@column_count** - The number of columns in the matrix.
- **@column_size** - The number of rows in the matrix.
- **@rows** - The array of arrays representation of the matrix. This field will not store the full representation and will instead point to an accessor method that computes the array of arrays (this is to preserve the efficient storage of SparseMatrix instances).
- **@nonzero_values** - An array of the nonzero values stored in the SparseMatrix.
- **@ia_array** - Each element of this array keeps track of the number of nonzero elements per row
- **@ja_array** - Each element of this array keeps track of the column of the corresponding nonzero element in @nonzero_values.

Methods

def initialize(rows, column_count = rows[0].size)

- Purpose: Construct a new SparseMatrix from an array of rows (arrays) that uses the CSR storage scheme.
- Pre:
    - rows.is_a?(Array)
    - rows.size > 0
    - column_count > 0 || rows[0].size > 0
- Post:
    - @nonzero_values.length == number of nonzero values in rows && @nonzero_values elements = nonzero elements in rows

    - @ia_array = rows.length + 1 && @ia_array[0] = [0] && @ia_array[i] = @ia_array[i-1] + number of nonzero elements on ith row
    - @ja_array.size == @nonzero_values.size && @ja_array[i] is the column index of element in @nonzero_values[i]
    - assert_equal(@column_count, column_count)
    - assert_equal(@column_size, rows.length)

DoKMatrix (Subclass of SparseMatrix)

A SparseMatrix subclass that utilizes Dictionary of Keys (DoK) to improve efficiency in storing zero values.

Class Invariants

Fields

- **@column_count -** The number of columns in the matrix.
- **@column_size** - The number of rows in the matrix.
- **@rows** - The array of arrays representation of the matrix. This field will not store the full representation and will instead point to an accessor method that computes the array of arrays (this is to preserve the efficient storage of SparseMatrix instances).
- **@nonzero_values** - An array of the nonzero values stored in the SparseMatrix.
- **@dict** - Ruby Hash that stores all nonzero values of the DoKMatrix.

Methods

def initialize(rows, column_count = rows[0].size)

- Purpose: Construct a new SparseMatrix from an array of rows (arrays) into a Ruby Hash, mapping keys:(row, column) => values
- Pre
    - rows.is_a?(Array)
    - rows.length > 0
    - column_count > 0 || rows[0].size > 0
- Post
    - @nonzero_values.size == number of nonzero values in rows && @nonzero_values elements = nonzero elements in rows
    - @dict.length == number of non_zero elements in rows && @dict.fetch((i,j)) == rows[i][j]
    - assert_equal(@column_count, column_count)
    - assert_equal(@column_size, rows.length)

DiagonalMatrix (Subclass of SparseMatrix)

A SparseMatrix subclass that only contains non-zero values within the main diagonal. With this knowledge heavy optimizations can be done.

Fields

- **@column_count** - The number of columns in the matrix.
- **@column_size** - The number of rows in the matrix.
- **@rows** - The array of arrays representation of the matrix. This field will not store the full representation and will instead point to an accessor method that computes the array of arrays (this is to preserve the efficient storage of SparseMatrix instances).
- **@nonzero_values** - An array of the nonzero values stored in the SparseMatrix.
- **@bandwidth** - The number of diagonals that the diagonal matrix spans (tridiagonal matrices have a bandwidth of 1)

Methods

def initialize(rows, column_count = rows[0].size)

- Purpose: Construct a new SparseMatrix from an array of rows (arrays) where there are only elements stored along the main diagonal (rows[i][i]). Variable bandwidth supported, asymmetrical band matrices not supported.
- Pre:
  - rows.is_a?(Array)
  - rows.length > 0
  - column_count > 0 || rows[0].size > 0
- Post:
  - @nonzero_values.size == number of nonzero values in rows && @nonzero_values elements = nonzero elements in rows
  - @bandwidth = max(abs(j - i) where rows[i][j] != 0)
  - @matrix.length = @column_size && @matrix[0].length = 2*@bandwidth + 1 && @matrix[i] = rows[i - @bandwidth, i + @bandwidth]
  - assert_equal(@column_count, column_count)
  - assert_equal(@column_size, rows.length)

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