

COSC 3311 Software Design

Assignment 1: Sparse Matrices

Due: Thursday, January 27, 5:30pm

1 Main Points

1.1 Learning objectives

- Eiffel programming with arrays and singly linked lists
- Abstract data types, classes and objects
- Specification of interfaces
- Loop and class invariants

1.2 To hand in

Hand in the following items as a package in order given in the following.

1. Cover page
2. A listing of the file `MatrixElement.e` – Section 3.1.1
3. A listing of the file `SparseMatrix.e` – Section 3.1.2
4. Pseudocode documentation for Sparse Matrix features – Section 3.2
5. Minimal output test program – Section 3.3

2 Assignment Overview

A sparse matrix is a matrix with entries having a zero (or null) value predominating. Rather than taking up memory storing the zeros, only nonzero entries are stored. For example in Figure 1 there are only four matrix elements stored (represented with the squares containing the values 3, -1, 6 and 10) whereas the 8x10 matrix contains 80 elements (see Figure 2 for a standard 2-d matrix).

In addition to the user data (the data values for each matrix element) there is a need to store **metadata** – data about the user data. For the sparse matrix, metadata consists of the following.

1. The row and column header lists – implemented as one dimensional arrays for this assignment. There is one singly linked list for each row, and for each column.
2. The maximum number of rows and columns (one number for both) – maximum size of the array.
3. The actual number of rows and the actual number of columns for the matrix;
4. The maximum row with a nonzero element and the maximum column with a nonzero element.

Each matrix element appears in a singly linked list of elements in the same row, and a singly linked list of elements in the same column.

Column header list

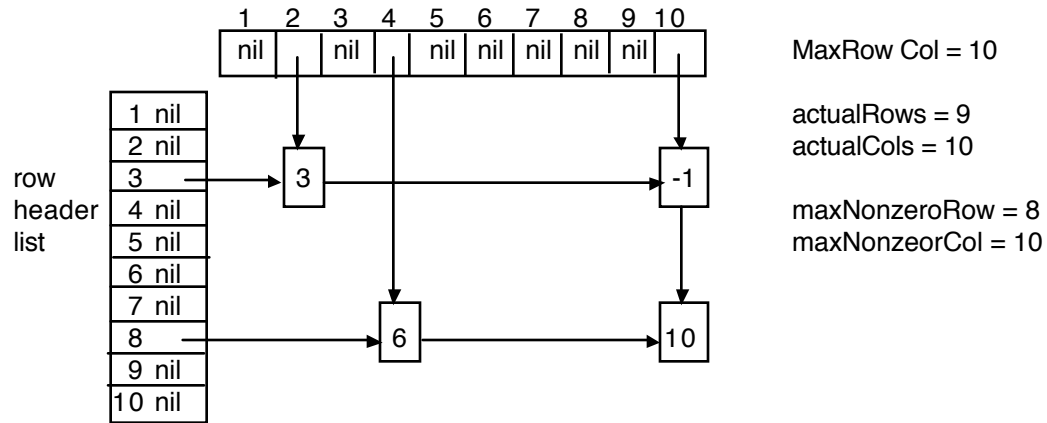


Figure 1: Example of an 9x10 sparse matrix with 4 nonzero elements at $\langle 3,2 \rangle$, $\langle 3,10 \rangle$, $\langle 8,4 \rangle$ and $\langle 8,10 \rangle$. All other elements are assumed to be zero. Maximum size of the matrix is 10x10. Compare with Figure 2.

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	3	0	0	0	0	0	0	0	-1
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	6	0	0	0	0	0	10
9	0	0	0	0	0	0	0	0	0	0

Figure 2: The standard way of storing the 9x10 matrix shown in Figure 1. All zero elements take up space. Note that a non zero element always appears in the maxNonzero Row and maxNonzeroCol.

Each matrix element (see Figure 3) contains, in addition to the user data, metadata consisting of the row and column position for the element (must be explicitly stored as physical location cannot be used; for example, the -1 is physically the second element in row 3 but logically it is in column 10) and pointers to the next row and element and the next column element. The pointers are required to link elements in the same row and column so they can be easily found.

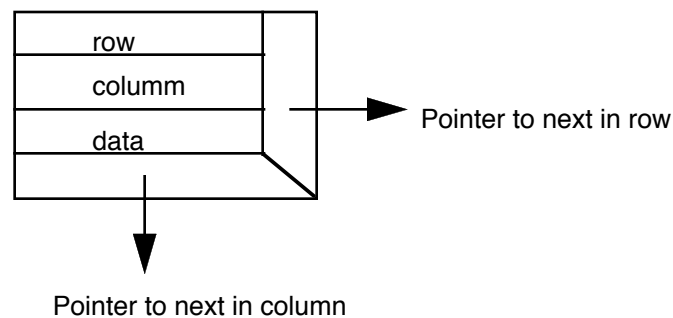


Figure 3: Shows the fields in a typical sparse matrix element.

3 Tasks

3.1 Create interfaces and implementation

You must program your own list features. While a singly linked list class could be used for either the rows or the columns, it cannot easily be used for both, thus leading to an asymmetric implementation. It is better to treat rows and columns symmetrically – in the same way. You will probably need to have some private support features to simplify the implementation.

3.1.1 *MatrixElement*

Create and document the file `MatrixElement.e`. It has the following features. They are straight forward write features to the fields of a matrix element. No get features are required

```
setRow(row)           setColumn(col)           setData(data)
setNextRow(nextRowElem) setNextColumn(nextColElem)
```

3.1.2 *SparseMatrix*

Create and document the file `SparseMatrix.e`. It has the following features. You will also need to add additional read features to get appropriate metadata for the test program (Section 3.3).

```
insertElement(row, col, data)
display
sum(sparseMatrix)
```

InsertElement

`InsertElement` modifies the sparse matrix structure. If there already exists a matrix element at `<row, col>` then data replaces the existing value. Recall that zero elements are not stored, thus replacing an existing value with zero is equivalent to deleting the element from the matrix. Recall that by definition the maximum row and column for a particular matrix contain nonzero data values, thus inserting a data value may increase or decrease the maximum row or column metadata for the matrix.

Display

`Display` prints the matrix in the following format. For a matrix with no rows or columns, print the message “The matrix is empty”. This version of `display` works for matrices up to 15 columns wide.

An example display for a 3x4 matrix. Keep integers to 3 digits for positive values and 2 digits for negative values.

	1	2	3	4
1	**	**	-1	**
2	**	1	2	**
3	**	**	**	6

Sum

`Sum` is a function to be used as in the following expression – a new matrix is created, neither `matrix1` nor `matrix2` are to be modified

```
newMatrix := matrix1.sum(matrix2)
```

It is the equivalent of the following mathematical expression.

```
newMatrix = matrix1 + matrix2
```

The basis of the sum algorithm is the merging of the elements from both matrices. Think of the elements of each matrix as being logically a single sequence by considering the rows as being appended to each other (alternately the columns could be appended). Then do a merge using the row and column indices. If the locations differ, then a copy of the earlier element is put into the sum. If the locations are the same, then the data values are summed and the element put into the sum. Use the insertElement feature to add the new elements to the sum matrix.

3.2 Pseudocode algorithm description

Consider the algorithms for the sparse matrix features insertElement, display and sum.

1. Document the algorithm using pseudocode for each feature with appropriate overview, comments, conditions, loop invariants and example diagrams (can be drawn by hand).
2. Include class invariants, for example only row headers with nonnull pointers point to nonzero elements, the maximum row contains a nonzero element, etc.

Aside from diagrams, documentation should be typed. An ASCII file is acceptable.

3.3 Test implementation

Create and document a minimal output test program to test SparseMatrix. The Eiffel file should contain appropriate comments to help the reader follow and understand what you are doing. As necessary add hand annotations and diagrams.