- The **ADT Matrix** is a container that represents a two-dimensional array.

- Each element has a unique position, determined by two indexes: its line and column.

- The domain of the ADT Matrix: $\mathcal{MAT} = \{mat | \text{mat is a matrix with elements of the type TElem}\}$

- init(mat, nrL, nrC)
  - **descr:** creates a new matrix with a given number of lines and columns
  - **pre:** $nrL \in N^*$ and $nrC \in N^*$
  - **post:** $mat \in \mathcal{MAT}$, mat is a matrix with nrL lines and nrC columns
  - **throws:** an exception if nrL or nrC is negative or zero
- nrLines(mat)
  - **descr:** returns the number of lines of the matrix
  - **pre:** $mat \in \mathcal{MAT}$
  - **post:** $nrLines \leftarrow$ returns the number of lines from mat
- nrCols(mat)
  - **descr:** returns the number of columns of the matrix
  - **pre:** $mat \in \mathcal{MAT}$
  - **post:** $nrCols \leftarrow$ returns the number of columns from mat

- element(mat, i, j)
  - **descr:** returns the element from a given position from the matrix (assume 1-based indexing)
  - **pre:** $mat \in \mathcal{MAT}$, $1 \leq i \leq nrLines$, $1 \leq j \leq nrColumns$
  - **post:** $element \leftarrow$ the element from line i and column j
  - **throws:** an exception if the position (i, j) is not valid (less than 1 or greater than nrLines/nrColumns)
- modify(mat, i, j, val)
  - **descr:** sets the element from a given position to a given value (assume 1-based indexing)
  - **pre:** $mat \in \mathcal{MAT}$, $1 \leq i \leq nrLines$, $1 \leq j \leq nrColumns$, $val \in TElem$
  - **post:** the value from position (i, j) is set to val. $modify \leftarrow$ the old value from position (i, j)
  - **throws:** an exception if position (i, j) is not valid (less than 1 or greater than nrLine/nrColumns)

SPARSE MATRIX

- We can memorize (line, column, value) triples, where value is different from 0 (or 0_{TElem}). For efficiency, we memorize the elements sorted by the (line, column) pairs (if the lines are different we order by line, if they are equal we order by column) - R1.
- When we have a Sparse Matrix (i.e., we keep only the values different from 0), for the modify operation we have four different cases, based on the value of the element currently at the given position (let's call it *current_value*) and the new value that we want to put on that position (let's call it *new_value*).
  - $current\_value = 0$ and $new\_value = 0 \Rightarrow$ do nothing
  - $current\_value = 0$ and $new\_value \neq 0 \Rightarrow$ insert in the data structure
  - $current\_value \neq 0$ and $new\_value = 0 \Rightarrow$ remove from the data structure
  - $current\_value \neq 0$ and $new\_value \neq 0 \Rightarrow$ just change the value in the data structure

- We can see that in the previous representation there are many consecutive elements which have the same value in the line array. The array containing this information could be compressed, in the following way:
  - Keep the *Col* and *Value* arrays as in the previous representation.
  - For the lines, have an array of number of lines + 1 element, in which at position i we have the position from the *Col* array where the sequence of elements from line *i* begins.
  - Thus, elements from line *i* are in the *Col* and *Value* arrays between the positions [*Line*[i], *Line*[i+1]).
- This is called **compressed sparse line representation**.
- **Obs:** In order for this representation to work, in the *Col* and *Value* arrays the elements have to be stored by rows (first elements of the first row, then elements of second row, etc.)
- In a similar manner, we can define **compressed sparse column representation**:
  - We need two arrays *Lines* and *Values* for the non-zero elements, in which first the elements of the first column are stored, than elements from the second column, etc.
  - We need an array with nrColumns + 1 elements, in which at position i we have the position from the *Lines* array where the sequence of elements from column *i* begins.
  - Thus, elements from column *i* are in the *Lines* and *Value* arrays between the positions [*Col*[i], *Col*[i+1]).