Chyong Artur **1. assignment/1. task** 24th March 2023
XOG5D9
xog5d9@inf.elte.hu
Group 11

Task

Implement the chessboard matrix type which contains integers. In these matrices, every second entry is zero. The entries that can be nonzero are located like the same colored squares on a chessboard, with indices (1, 1), (1, 3), (1, 5), ..., (2, 2), (2, 4), ... The zero entries are on the indices (1, 2), (1, 4), ..., (2, 1), (2, 3), ... Store only the entries that can be nonzero in row-major order in a sequence. Don't store the zero entries. Implement as methods: getting the entry located at index (i, j), adding, and multiplying two matrices, and printing the matrix (in a shape of m by n).

Chessboard matrix type

Set of values

CBMatrix (n) = { a ∈ $\mathbb{Z}$ n×n | ∀i,j∈[1..n]: i+j mod 2 <> 0 → a[i,j]=0 }

Operations

1. Getting the entry at index (i, j)
Gets the element at the given index from the matrix.

Formally: A = a : CBMatrix(n), i : $\mathbb{Z}$, j : $\mathbb{Z}$, e : $\mathbb{Z}$
Pre = (a = a` ∧ i=i' ∧ j=j' ∧ i,j∈[1..n])
Post = (Pre ∧ e=a[i,j])

This method will output the non-zero value only if sum of i and j is even, otherwise the output is zero.

2. Addition of two matrices
Adds two same-size matrices.

Formally: A = a : CBMatrix(n), b : CBMatrix(n), c : CBMatrix(n)
Pre = (a = a` ∧ b = b`)
Post = (Pre ∧ ∀i,j∈[1..n]: c[i,j] = a[i,j] + b[i,j])

This method will sum two same size matrices and generate a new matrix with a result.

3. Multiplication of two matrices
Multiplies two same-size matrices.

Formally: A = a : CBMatrix(n), b : CBMatrix(n), c : CBMatrix(n)
Pre = (a = a` ∧ b = b`)
Post = (Pre ∧ ∀i,j∈[1..n]: c[i,j] = $\Sigma_{k=1..n}$ a[i,k] * b[k,j])

This method will multiply two same size matrices and generate a new matrix with a result.

4. Printing the matrix (in a shape of m by n)
ToString method provides the string of a matrix in a shape of m by n.

Formally: A = a : CBMatrix(n), s : S
Pre = (a = a`)
Post = $\forall i,j \in [1..n] : s = \bigoplus a[i,j]$ + "newLine"
where newLine = "\n"

This operation will go through all the values and concatenate them to the result string. Once a row ends, a new line is added to represent the rows and columns.

Representation

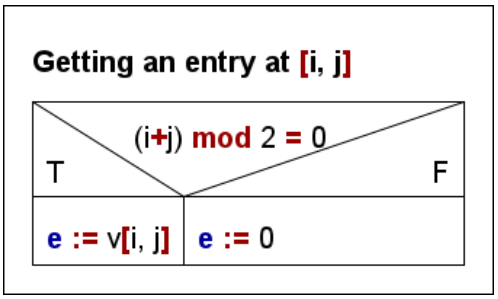
The chess board matrix is represented by a vector of integer values.

$$a = \begin{matrix} a11 & 0 & a12 \\ 0 & a22 & 0 \\ a31 & 0 & a_{nn} \end{matrix} \quad <-> \quad v = \langle a11, a12, a22, a31, ann \rangle$$

Implementation[1]

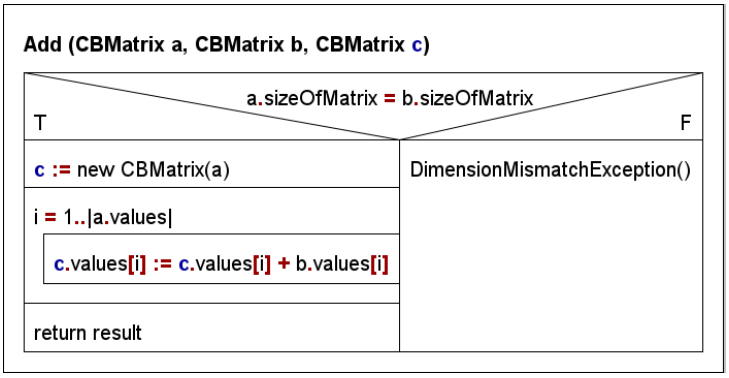
1. Getting the entry at index (i, j)

Getting the entry of the ith column and jth row (i,j∈[1..n]) e := a[i,j] where the matrix is represented by v, 1≤i≤n, and n stands for the size of the matrix can be implemented as



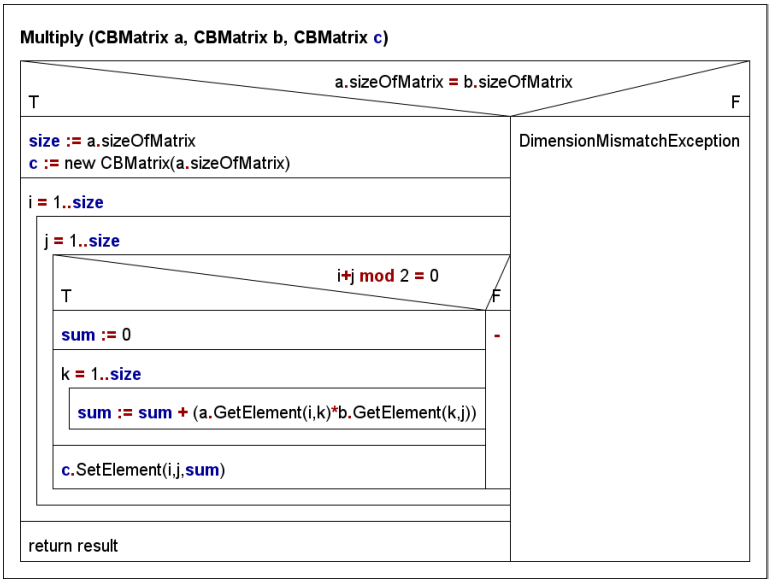
2. Addition of two matrices

The sum of matrices a and b (represented by lists x and y) goes to matrix c (represented by list z), where all the lists have to have the same size.



3. Multiplication of two matrices

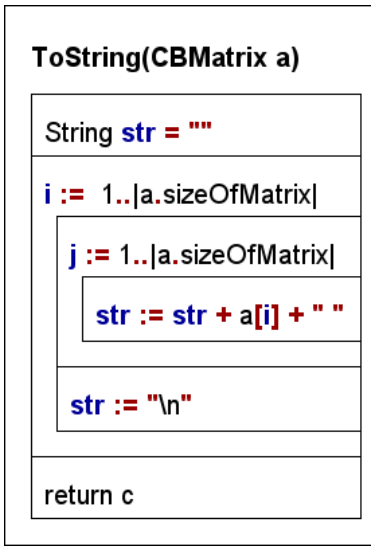
The product of matrices a and b (represented by lists x and y) goes to matrix c (represented by list z), where all the lists have to have the same size.



[1] To implement an operation, a program has to be given (not necessarily structogram).

4. Printing the matrix (in a shape of m by n)

The method will return the string in the format of m by n with the proper paddings.



Testing

Testing the operations (black box testing)

1) Constructor:
   a) Constructor with an odd integer as an argument. 3 => Size of the matrix is 3. Number of values is counted with a formula for odd size matrices which is 5 in the current case.
   b) Constructor with an even integer as an argument. 4 => Size of the matrix is 4. Number of values is counted with a formula for even size matrices which is 8 in the current case.
   c) Empty constructor. => Size of a matrix is 3. There are 5 values from 1 to 5 in the list of the matrix.
   d) Constructor with a list of values (number of which is enough for a full matrix). { 16, 9, 8, 15, 6 } => Size of a matrix is 3. Number of values is 5.
   e) Constructor with a list of values (number of which is not enough for a full matrix). { 16, 9, 8, 15, 6 } => Exception InvalidVectorException because there are not enough values in the list to create a full matrix.
   f) Constructor with another matrix as an argument. new CBMatrix() => Size of the matrix is 3. Number of values is 5. If we change the entry in the matrix-argument, it won't change in the new matrix because it's not a reference but a copy.
2) Get element:
   a) Matrix with empty constructor:
      - GetElement(0, 0) – very first value => 1.
      - GetElement(2, 2) – very last value => 5.
      - GetElement(1, 0) – zero value cell => 0.
      - GetElement(5, 5) – outside of the matrix, indices exceed the actual size => Exception InvalidIndexException.
   b) SetElement(1, 1, 100); GetElement(1, 1) – after interaction with SetElement() => 100.
3) Sum of matrices:
   a) Sum of matrix with empty constructor and matrix with 3 as a parameter in the constructor:
      - GetElement(0, 0) – 1 + 1 => 2.
      - GetElement(2, 2) – 1 + 5 => 6.
      - GetElement(1, 0) – 0 + 0 => 0.
   b) Sum of matrix with empty constructor and matrix with 5 as a parameter in the constructor => Exception DimensionMismatchException.
4) Product of matrices:
   a) Product of matrix with empty constructor and matrix with 3 as a parameter in the constructor:
      - GetElement(0, 0) – 1*1 + 0*0 + 1*2 => 3.
      - GetElement(2, 2) – 4*1 + 0*0 + 5*1 => 9.
      - GetElement(1, 0) – 0*1 + 3*0 + 0*1 => 0.
   b) Product of matrix with empty constructor and matrix with 5 as a parameter in the constructor => Exception DimensionMismatchException.
5) Set element:
   a) Matrix with empty constructor:
      - SetElement(0, 0, 100); GetElement(0, 0) – very first value => 100.
      - SetElement(2, 2, 100); GetElement(2, 2) – very last value => 100.
      - SetElement(1, 0, 100) – zero value cell => Exception ZeroCellException.
      - SetElement(5, 5, 100) – outside of the matrix, indices exceed the actual size => Exception InvalidIndexException.

Testing based on the code (white box testing)

1. Generating and catching exceptions.
2. Creating an extreme-size matrix (-1, 0, 1, 1000).