Teréz A. Várkonyi **1. assignment/0. task** 11th February 2019
NEPTUNCODE
nick@inf.elte.hu
Group 0

Task

Implement the diagonal matrix type which contains integers. These are square matrices that can contain nonzero entries only in their diagonal. Don't store the zero entries. Store only the entries that can be nonzero in a sequence. Implement as methods: getting and setting the entry located at index (i, j), adding and multiplying two matrices, reading and printing the matrix (in a square shape).

Diagonal matrix type

Set of values

$Diag(n) = \{ a \in \mathbb{Z}^{n \times n} \mid \forall i,j \in [1..n]: i \neq j \rightarrow a[i,j]=0 \}$

Operations

1. Getting an entry
Getting the entry of the *i*th column and *j*th row ($i,j \in [1..n]$): $e:=a[i,j]$.

Formally: $A : Diag(n) \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$
$a \quad i \quad j \quad e$
$Pre = (a=a' \wedge i=i' \wedge j=j' \wedge i,j \in [1..n])$
$Post = (Pre \wedge e=a[i,j])$

This operation needs any action only if $i=j$, otherwise the output is zero.

2. Setting an entry
Setting the entry of the *i*th column and *jth* row ($i,j \in [1..n]$): $a[i,j]:=e$. Entries outside the diagonal cannot be modified ($i=j$).

Formally: $A = Diag(n) \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$
$a \quad i \quad j \quad e$
$Pre = (e=e' \wedge a=a' \wedge i=i' \wedge j=j' \wedge i,j \in [1..n] \wedge i=j)$
$Post = (e=e' \wedge i=i' \wedge j=j' \wedge a[i,j]=e \wedge \forall k,l \in [1..n]: (k \neq i \vee l \neq j) \rightarrow a[k,l]=a'[k,l])$

This operation needs any action only if $i=j$, otherwise it gives an error if we want to modify a zero entry.

3. Sum
Sum of two matrices: $c:=a+b$. The matrices have the same size.

Formally: $A = Diag(n) \times Diag(n) \times Diag(n)$
$a \quad b \quad c$
$Pre = (a=a' \wedge b=b')$
$Post = (Pre \wedge \forall i,j \in [1..n]: c[i,j]= a[i,j] + b[i,j])$

In case of diagonal matrices there is an easier version:

$\forall i \in [1..n]$: $c[i,i] = a[i,i] + b[i,i]$ és $\forall i,j \in [1..n]$: $i \neq j \rightarrow c[i,j]=0$.

4. Multiplication

Multiplication of two matrices: *c:=a*b*. The matrices have the same size.

Formally: $A = \underset{a}{Diag(n)} \times \underset{b}{Diag(n)} \times \underset{c}{Diag(n)}$

$Pre = (a=a' \wedge b=b')$

$Post = (Pre \wedge \forall i,j \in [1..n]: c[i,j] = \Sigma_{k=1..n} a[i,k] * b[k,j])$

In case of diagonal matrices there is an easier version:

$\forall i \in [1..n]$: $c[i,i] = a[i,i]*b[i,i]$ és $\forall i,j \in [1..n]$: $i \neq j \rightarrow c[i,j]=0$.

Representation

Only the diagonal of the $n \times n$ matrix has to be strored.

$$a = \begin{matrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ 0 & 0 & a_{33} & \dots & 0 \\ 0 & 0 & 0 & \dots & a_{nn} \end{matrix} \quad \leftrightarrow \quad v = < a_{11} \quad a_{22} \quad a_{33} \quad a_{nn} >$$

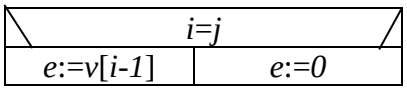
Only a one-dimension array (v) is needed, with the help of which any entry of the diagonal matrix can be get:

$$a[i,j] = \begin{cases} v[i] & if \quad i = j \\ 0 & if \quad i \neq j \end{cases}$$

Implementation[1]

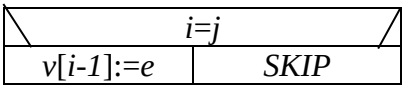
1. Getting an entry

Getting the entry of the *i*th column and *jth* row ($i,j \in [1..n]$) $e:=a[i,j]$ where the matrix is represented by $v, 1 \le i \le n$, and n stands for the size of the matrix can be implemented as



2. Setting an entry

Setting the entry of the *i*th column and *jth* row ($i,j \in [1..n]$) $a[i,j]:=e$ where the matrix is represented by $v, 1 \le i \le n$, and n stands for the size of the matrix can be implemented as



3. Sum

The sum of matrices *a* and *b* (represented by arrays *t* and *u*) goes to matrix *c* (represented by array *u*), where all of the arrays have to have the same size.

$$\forall i \in [0..n-1]: u[i]:= v[i] + t[i]$$

4. Multiplication

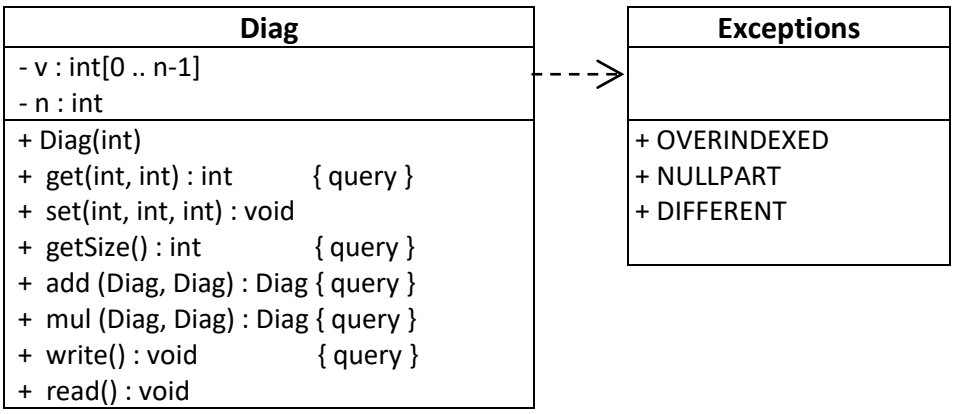
The product of matrices *a* and *b* (represented by arrays *t* and *u*) goes to matrix *c* (represented by array *u*), where all of the arrays have to have the same size.

$$\forall i \in [0..n-1]: u[i]:= v[i] * t[i]$$

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

Class

The diagonal matrix type is worked out as a class. The size of the matrix can be set through the constructor. At the operations, the matrices have to have the same size, otherwise the program should throw an exception.



The entries in the diagonal can be represented as a vector<int> or as a dynamic array. In the latter case the constructor allocates the dynamic array an the destructor frees the memory (a copy constructor and an assignment operator is needed, too).

To override the getter and the setter of an entry, operator () is overridden in two ways. Operators sum and product are created as friend functions.

The class is extended by read and write methods, worked out via operators.

For error handling, four exceptions are defined. `OVERINDEXED` is thrown when wrong indexes are given as inputs in the getter and the setter. `NULLPART` is raised when an entry outside the diagonal should be overwritten. `DIFFERENT` exception is given in case of operation (sum, product) with matrices of different size. `INVALID` exception is thrown, when (e.g. in the reading operator) negative number is given for the size of the matrix.

Testing

Testing the operations (black box testing)

1) Creating, reading, and writing matrices of different size.
   a) 0, 1, 2, 5-size matrix
2) Getting and setting an entry
   a) Getting and setting an entry in the diagonal
   b) Getting and setting an entry outside the diagonal
   c) Illegal index, indexing a 0-size matrix
3) Copy constructor
   a) Creating matrix *b* based on matrix *a*, comparing the entries of the two matrices. Then, changing one of the matrices and comparing the entries of the two matrices.
4) Assignment operator
   a) Executing command *b=a* for matrices *a* and *b* (with and without same size), comparing the entries of the two matrices. Then, changing one of the matrices and comparing the entries of the two matrices.
   b) Executing command *c=b=a* for matrices *a, b,* and *c* (with and without same size), comparing the entries of the three matrices. Then, changing one of the matrices and comparing the entries of the three matrices.
   c) Executing command *a=a* for matrix *a*.
5) Sum of two matrices, command *c:=a+b*.
   a) With matrices of different size (size of *a* and *b* differs, size of *c* and *a* differs)
   b) Checking the commutativity (a + b == b + a)
   c) Checking the associativity (a + b + c == (a + b) + c == a + (b + c))
   d) Checking the neutral element (a + 0 == a, where 0 is the null matrix)
6) Multiplication of two matrices, command *c:=a*b*.
   a) With matrices of different size (size of *a* and *b* differs, size of *c* and *a* differs)
   b) Checking the commutativity (a * b == b * a)
   c) Checking the associativity (a * b * c == (a * b) * c == a * (b * c))
   d) Checking the neutral element (a * 0 == 0, where 0 is the null matrix)
   e) Checking the identity element (a * 1 == a, where 1 is the identity matrix)

Testing based on the code (white box testing)

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