**Mahammad Isayev**

VF36R7

[vf36r7@inf.elte.hu](mailto:vf36r7@inf.elte.hu) Group 1

**1. assignment/2. task**

17th March 2021

# Task

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

# Nonzero diagonals matrix type

## Set of values

*Matrix*(*n*) = { *a* ℤn×n  *i,j*[*1*..*n*]: *i*≠*j and i + j* ≠ size - 1→ *a*[*i,j*]=*0* }

## Operations

1. *Getting an entry*

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

Formally: *A* : *Matrix*(*n*) × ℤ × ℤ × ℤ

*a i j e*

*Pre* = ( *a*=*a’*  *i*=*i’*  *j*=*j’*  *i,j*[*1*..*n*] )

*Post =* ( *Pre*  *e*=*a*[*i,j*] )

This operation needs one condition if *i* not equal to j and i + j not equal to size – 1, output becomes 0.

1. *Setting an entry*

Setting the entry of the *i*th column and *jth* row (*i,j*[*1*..*n*]): *a*[*i,j*]:=*e*. Entries outside the diagonal scannot be modified (*i*=*j and i + j* = size - 1).

Formally: *A* = *Matrix*(*n*) × ℤ × ℤ × ℤ

*a i j e*

*Pre* = ( *e*=*e’*  *a*=*a’*  *i*=*i’*  *j*=*j’*  *i,j*[*1*..*n*]  *i*=*j* )

*Post* = (*e*=*e’*  *i*=*i’*  *j*=*j’*  *a*[*i,j*]=*e*  *k,l*[*1*..*n*]: (*k*≠*i*  *l*≠*j*)→ *a*[*k,l*]=*a’*[*k,l*] )

1. *Sum*

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

Formally: *A* = *Matrix*(*n*) × *Matrix*(*n*) × *Matrix*(*n*)

*a b c Pre* = ( *a*=*a’*  *b*=*b’*)

*Post* = ( *Pre*  *i,j*[*1*..*n*]: *c*[*i*,*j*]= *a*[*i*,*j*] + *b*[*i*,*j*] )

In case of diagonal matrices there is an easier version:

*i*[*1*..*n*]: *c*[*i*,*i*]= *a*[*i*,*i*] + *b*[*i*,*i*] and *i,j*[*1*..*n*]: *i*≠*j and i + j* ≠ size - 1→ *c*[*i,j*]=*0*.

1. *Multiplication*

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

Formally: *A* = *Matrix*(*n*) × *Matrix*(*n*) × *Matrix*(*n*)

*a b c Pre =* ( *a*=*a’*  *b*=*b’*)

*Post* = ( *Pre*  *i,j*[*1*..*n*]: *c*[*i*,*j*]= *k=1..n a*[*i*,*k*] \* *b*[*k*,*j*])

In case of diagonal matrices there is an easier version:

*i*[*1*..*n*]: *c*[*i*,*i*]= *a*[*i*,*i*]\**b*[*i*,*i*] and *i,j*[*1*..*n*]: *i*≠*j* → *c*[*i,j*]=*0*.

## Representation

Only the diagonals of the *n×n* matrix has to be stored.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *a11* | *0* | *0* | *… ar* |  |
| *0* | *a22* | *a23* | *… 0* | *v* = < *a11 a22 a33 a23 a32 a44 ann ar>* |
| *a = 0* | *a32* | *a33 … 0* | |
| *a44* | *0* | *0* | *… ann* |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

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] = {𝑣[𝑖] 𝑖𝑓 𝑖 = 𝑗}

0 𝑖𝑓 𝑖 ≠ 𝑗

## Implementation1

1. *Getting an entry*

Getting the entry of the *i*th 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

|  |  |
| --- | --- |
| *i*=*j* | |
| *e*:=*v*[*i-1*] | *e*:=*0* |

1. *Setting an entry*

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

|  |  |
| --- | --- |
| *i*=*j* | |
| *v*[*i-1*]:=*e* | *SKIP* |

1. *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.

*i*[*0*..*n-1*]: *u*[*i*]:= *v*[*i*] + *t*[*i*]

1. *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.

*i*[*0*..*n-1*]: *u*[*i*]:= *v*[*i*] \* *t*[*i*]

# Testing

Testing the operations (black box testing)

1. Creating, reading, and writing matrices of different size.
   1. 0, 1, 2, 5-size matrix
2. Getting and setting an entry
   1. Getting and setting an entry in the diagonal
   2. Getting and setting an entry outside the diagonal
   3. Illegal index, indexing a 0-size matrix
3. Copy constructor
   1. 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
   1. 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.
   2. 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.
   3. Executing command *a=a* for matrix *a*.
5. Sum of two matrices, command *c*:=*a*+*b*.
   1. With matrices of different size (size of *a* and *b* differs, size of *c* and *a* differs)
   2. Checking the commutativity (a + b == b + a)
   3. Checking the associativity (a + b + c == (a + b) + c == a + (b + c))
   4. Checking the neutral element (a + 0 == a, where 0 is the null matrix)
6. Multiplication of two matrices, command *c*:=*a*\**b*.
   1. With matrices of different size (size of *a* and *b* differs, size of *c* and *a* differs)
   2. Checking the commutativity (a \* b == b \* a)
   3. Checking the associativity (a \* b \* c == (a \* b) \* c == a \* (b \* c))
   4. Checking the neutral element (a \* 0 == 0, where 0 is the null matrix)
   5. 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.