**TRAN QUOC TUAN – UG2F20 ---------------------------------------------------------------** 17th March 2024

[Ug2f20@inf.elte.hu](mailto:Ug2f20@inf.elte.hu)

Group 10

**TASK:**

Implement the block matrix type which contains integers. These are square matrices that can contain nonzero entries only in two blocks on their main diagonal. Let the size of the first and second blocks be b1 and b2, where 1≤b1,b2≤n-1 and b1+b2=n (in the example, b1=2 and b2=4). Don't store the zero entries. Store only the entries that can be nonzero in a sequence or two smaller matrices. Implement as methods: getting the entry located at index (i, j), adding and multiplying two matrices, and printing the matrix (in a square shape).

**BLOCK MATRIX TYPE:**

**Set of values:**

BM(n) = { a∈ ℤ^(n× n) ⎪ ∀i,j∈[1..n]: i > b1 ∧ i < n ∧ j < b1 ∨ i <= b1 ∧ j >= b1 ∧ j < n: a[i,j]=0}

**Operation:**

1.Get entry:

Getting the entry of the ith column and jth row

Formally:

A : BM(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 only need action if (i < b1 and j < b1) or (i >= b1 and j >= b1), otherwise output is 0.

2.Set entry:

Setting the entry of the ith column and jth row (i,j∈[1..n]): a[i,j]:=e. Entries outside the diagonal blocks cannot be modified.

Formally:

A : BM(n) × ℤ × ℤ × ℤ

a i j e

Pre = ( e=e’ ∧ a=a’ ∧ i=i’ ∧ j=j’ ∧ i,j∈[1..n] ∧ i < b1 ∧ j < b1 ∨ i >= b1 ∧ j >= b1  
∧ b1=b1’ ∧ b2=b2’ ∧ n =n’ ∧ n=b1+b2)

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] )

This operation needs any action only if (i < b1 and j < b1) or (i >= b1 and j >= b1), 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 = BM(n) × BM(n) × BM(n)

a b c

Pre = ( a=a’ ∧ b=b’)

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

4. Multiplication

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

Formally: A = Diag(n) × Diag(n) × Diag(n)

a b c

Pre = ( a=a’ ∧ b=b’)

Post = ( Pre ∧ ∀i,j∈[1..n]: c[i,j]= Σ(k=1..n) {a[i,k] \* b[k,j]} )

**Representation:**

Only the diagonal block of the n×n has to be stored:  
**a = a11… a1m 0 0 … 0**

**a12… a2m 0 0 … 0**

**0 … 0 a3(m+1) a3(m+2) … a3n**

**0 … 0 a4(m+1) a4(m+2) … a4n**

**…**

**0 … 0 ak(m+1) ak(m+2) … akn**

↔

**v = < a11… a1m a12… a2m a3(m+1) a3(m+2) … a3n a4(m+1) a4(m+2) … a4n ak(m+1) ak(m+2) …akn >**

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

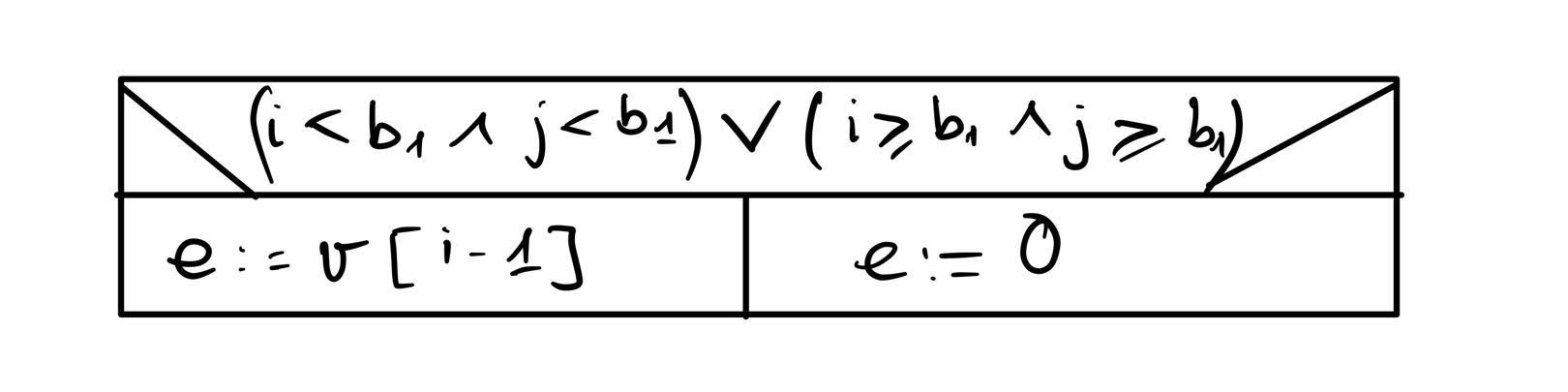
a[𝑖, 𝑗] = {𝑣[𝑖] 𝑖𝑓 (i > b1 ∧ j < b1) ∨ (i <= b1 ∧ j >= b1)}

{=0, otherwise}

**Implementation:**

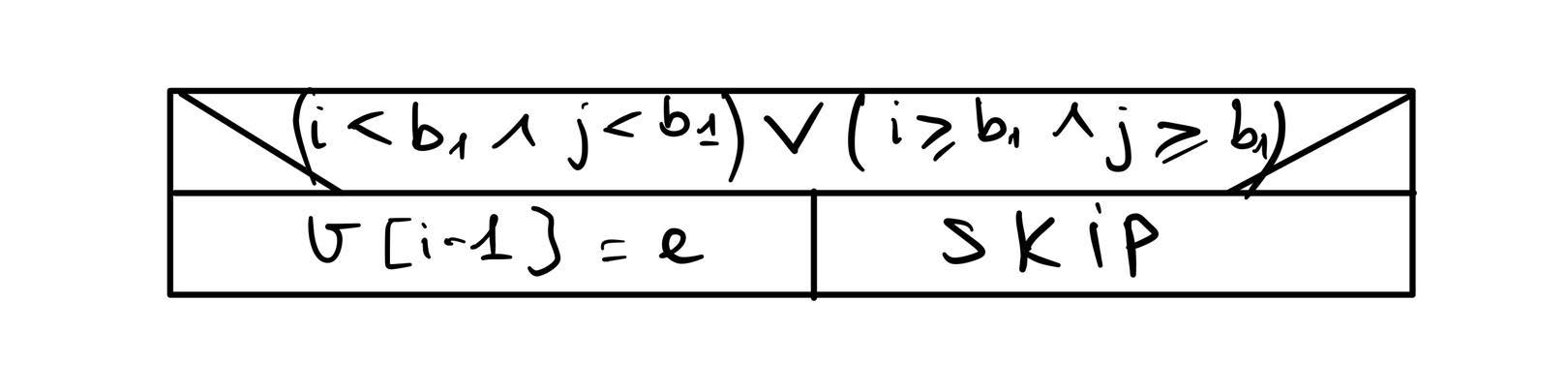
**1. Get value**

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.Set value**

Setting the entry of the ith 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



**3. Sum**

The sum of matrices a and b (represented by arrays t and v) 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]

**4. Multiplication**

The product of matrices a and b (represented by arrays t and v) 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:**

**1. Test Create Matrix:**

Purpose: Tests the creation of matrices of different sizes.

Inputs:

Expected block sizes:

b1=2,b2=3

Expected matrix size:

n=5

Actions: Creates a block matrix with the specified sizes.

Assertions: Verifies that the created matrix has the expected block sizes and overall size.

**2. Test Get Entry:**

Purpose: Tests the functionality of the GetEntry method.

Inputs: Block matrix with b1=2, b2=1, n=3.

Actions: Fills the matrix with values using the MatrixFiller method. Retrieves entries at various positions.

Assertions: Checks if the retrieved entries match the expected values.

**3. Test Summation:**

Purpose: Tests the addition of two block matrices.

Inputs: Two block matrices with b1=2,b2=1,n=3.

Actions: Fills the matrices with values using the MatrixFiller method. Performs matrix addition.

Assertions: Verifies if the resulting matrix entries match the expected values.

**4. Test Multiply:**

Purpose: Tests the multiplication of two block matrices.

Inputs: Two block matrices with b1=2,b2=2,n=4.

Actions: Fills the matrices with values using the MatrixFiller method. Performs matrix multiplication.

Assertions: Checks if the resulting matrix entries match the expected values.

**ADDITIONAL NOTES:**

**MatrixFiller Method Description:**

Purpose: The MatrixFiller method is designed to populate a block matrix with specific values for testing purposes. It facilitates the creation of test scenarios where different matrix entries are predefined to evaluate various functionalities of the block matrix implementation.

Parameters: matrix: The block matrix object to be populated with values. offset: An integer value used to offset the generated values. This allows for differentiating between entries in the matrix.

Functionality: The MatrixFiller method iterates through the elements of the given block matrix and assigns values based on the specified logic: For elements within the first block (0 ≤ i, j < b1), values are assigned based on the formula: i \* b1 + j + offset. For elements within the second block (b1 ≤ i, j < n), values are assigned based on the formula: (i - b1) \* b2 + (j - b1) + offset.

Example: Consider a block matrix with b1 = 2, b2 = 1, and n = 3. With an offset of 3, the MatrixFiller method would populate the matrix as follows:

| 3 4 0 |

| 5 6 0 |

| 0 0 3 |

Usage: The MatrixFiller method is utilized within test methods to prepare block matrices with predetermined values, enabling comprehensive testing of functionalities such as entry retrieval, summation, and multiplication.