Introduction

This document is a required write-up for the open problem given by the Human Analysis Lab, MSU. This is a draft solution. It reduces the space complexity to represent special matrices proposed by the problem and outlines an Abstract Data Type (ADT) for the solution. The proposed ADT is implemented by python.

Note that, the solution is a draft solution and numerical precision is not maintained. If the solution is convincing more precise implementation with proof can be provided.

Representative Matrices

The problem states that a matrix A of size $\mathbb{R}^{nd\times nd}$ is composed of n^2 non-overlapping blocks of matrices of size $d \times d$, where each such matrix $D_{ij}$is a diagonal matrix, i.e., entries outside the main diagonal are all zero. Since matrix A is a sparse matrix, it can be represented with some efficient data structure. Here, I've proposed such a data structure with the necessary interfaces that implement essential matrix operations efficiently. For the sake of easy description, let's call matrix A base matrix. I represent the base matrix A with d representative matrices, a_k where $k = 1, 2, 3, \ldots, d$, each of size $\mathbb{R}^{n\times n}$. $a_k[i][j]$ is defined as follows:

$$a_k[i][j] = D_{ij}[k][k]$$

To illustrate, let's set some random values to A and represent it with corresponding representative matrices.

$$A = \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix} & \begin{bmatrix} -3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 9 \end{bmatrix} \\ \begin{bmatrix} 2 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 13 \end{bmatrix} & \begin{bmatrix} -17 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & -5 \end{bmatrix} \end{bmatrix}^{1}$$

Now, considering the indices of matrices of each D_{ij} independently and applying the above-mentioned relation, the representative matrices for A are given bellow:

$$a_1 = \begin{bmatrix} 1 & -3 \\ 2 & -17 \end{bmatrix}$$

$$a_2 = \begin{bmatrix} 4 & 7 \\ 7 & 9 \end{bmatrix}$$

$$a_3 = \begin{bmatrix} 5 & 9 \\ 13 & -5 \end{bmatrix}$$

[1] The brackets within matrix are only to the correspondence between entries of A with corresponding D_{ij} . It has no other mathematical notions.

Elementary binary matrix operations

Elementary binary matrix operations such as addition, subtraction, multiplication (dot product operations) between two base matrices, for example, A and B will be carried out through their representative matrices. To illustrate the idea let's carefully examine the following examples.

Suppose,

$$B = \begin{bmatrix} 2 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -5 & 0 \\ 0 & 0 & 1 & 0 & 0 & -17 \\ -2 & 0 & 0 & 1 & 0 & 0 \\ 0 & -9 & 0 & 0 & 9 & 0 \\ 0 & 0 & -11 & 0 & 0 & 15 \end{bmatrix}$$

The representative matrices of B are as follows,

$$b_1 = \begin{bmatrix} 2 & -1 \\ -2 & 1 \end{bmatrix}$$

$$b_2 = \begin{bmatrix} 1 & -5 \\ -9 & 9 \end{bmatrix}$$

$$b_3 = \begin{bmatrix} 1 & 17 \\ -11 & 15 \end{bmatrix}$$

To illustrate an example, let's perform a multiplication operation between A and B which produces the output matrix Z.

$$Z = AB = \begin{bmatrix} 1 & 0 & 0 & -3 & 0 & 0 \\ 0 & 4 & 0 & 0 & 7 & 0 \\ 0 & 0 & 5 & 0 & 0 & 9 \\ 2 & 0 & 0 & -17 & 0 & 0 \\ 0 & 7 & 0 & 0 & 9 & 0 \\ 0 & 0 & 13 & 0 & 0 & -5 \end{bmatrix} \times \begin{bmatrix} 2 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -5 & 0 \\ 0 & 0 & 1 & 0 & 0 & -17 \\ -2 & 0 & 0 & 1 & 0 & 0 \\ 0 & -9 & 0 & 0 & 9 & 0 \\ 0 & 0 & -11 & 0 & 0 & 15 \end{bmatrix}$$

$$Z = \begin{bmatrix} 8 & 0 & 0 & -4 & 0 & 0 \\ 0 & -59 & 0 & 0 & 43 & 0 \\ 0 & 0 & -94 & 0 & 0 & 50 \\ 38 & 0 & 0 & -19 & 0 & 0 \\ 0 & -74 & 0 & 0 & 46 & 0 \\ 0 & 0 & 68 & 0 & 0 & -296 \end{bmatrix}$$

Instead of performing these operations over these base matrices, let's perform them with their associated representative matrices. **The main idea is to compute the non-zero entries correctly and store them systematically.** Let's define z_1, z_2, z_3 as the representative matrices of Z and they are computed as follows:

$$z_1 = a_1 \times b_1,$$

$$z_2 = a_2 \times b_2,$$

$$z_3 = a_3 \times b_3$$

So the entries are,

$$z_1 = \begin{bmatrix} 1 & -3 \\ 2 & -17 \end{bmatrix} \times \begin{bmatrix} 2 & -1 \\ -2 & 1 \end{bmatrix} = \begin{bmatrix} 8 & -4 \\ 38 & 19 \end{bmatrix}$$

$$z_2 = \begin{bmatrix} 4 & 7 \\ 7 & 9 \end{bmatrix} \times \begin{bmatrix} 1 & -5 \\ -9 & 9 \end{bmatrix} = \begin{bmatrix} -59 & 43 \\ -74 & 46 \end{bmatrix}$$

and,

$$z_3 = \begin{bmatrix} 5 & 9 \\ 13 & -5 \end{bmatrix} \times \begin{bmatrix} 1 & 17 \\ -11 & 15 \end{bmatrix} = \begin{bmatrix} -94 & 50 \\ 68 & -296 \end{bmatrix}$$

Note that, the non-zero entries are computed correctly and their relative positions are preserved (which is shown with different colors). This property of performing operations using representative matrices are true[2] for all other binary operations i.e. addition, subtraction etc.

Mathematical Generalization

Binary operations

Suppose A and B are two matrices of size $\mathbb{R}^{nd \times nd}$ and their respective representative matrices are a_k and b_k, respectively, where $k = 1, 2, 3, \ldots, d$. Now for $Z = A \circledast B$ where $\circledast \in \{+, -, \times\}$, the entries of representative matrices z_k is computed as:

$$z_k = a_k \circledast b_k$$

It is guaranteed that the entries are computed correctly as they preserve the rows of the matrices A and B and, relative positions for all entries are preserved as well.

Index Transformations

Suppose r and c indicate the row and column of a base matrix A of size $\mathbb{R}^{nd \times nd}$ and a_k for $k = 1, 2, 3, \ldots, d$ are the representative matrices of A, then,

$$A[r][c] = a_k[i][j] \Rightarrow r = k + (i-1)d \text{ and } c = k + (j-1)d$$

The reverse transformation can also be found by some elementary manipulation.

Unary operations

Like binary operations, unary operations is also carried out through the representative matrices. The inverse operation is illustrated here. Other functionalities such as finding determinants and rank can be computed using almost same techniques.

The most practical and stable way for finding inverse of a matrix is by LUP decomposition. In our case, each representative matrix is decomposed first and then its inverse is found by forward substitution and

[2] The proof can be given. One important hint that the representative matrices preserve the elements of the row and their relative positions.

back substitution. All these methods are well known methods in linear algebra, hence not discussed here. The inverse of each representative matrix contains the no-zero elements of A^{-1}. Reconstruction of A^{-1} is possible from the above-mentioned index transformation.

Notes on finding the determinant

Computing determinant of a matrix A is efficient if its LUP decomposition is available. We can use the following well established rules:

$$\det(X) = \det(P) * \det(L) * \det(U)$$

This operation is not included in the main implementation but can be easily added.

Abstract Data Type CompositionMatrix

I define an Abstract Data Type CompositionMatrix that includes a list of representation matrices of a base matrix. Its interfaces are given bellow:

__add__(self, other): add two CompositionMatrix and return a CompositionMatrix that contains the elements of the summation of related two base matrix

__sub__(self, other): similar as __add__ but returns the result of subtraction

__mul__(self, other): returns CompositionMatrix of dot products

__invert__(self): returns CompositionMatrix of the inverse of a matrix.

transform(self): returns base matrix from its representation matrices.

Implementation

The CompositionMatrix ADT is implemented in python class CompositionMatrix[3]. Two supporting modules lup[4] and elementary[5] are also added. The module lup contains all sufficient functions required for LUP decompositions where elementary module contains the elementary matrix operations. All these implementations are implemented by raw python functionalities. For better performance numpy module can be used.

Complexity Analysis

Space Complexity

With this solution a base matrix A of size $\mathbb{R}^{nd \times nd}$ can be represented with d representative matrices each of size $\mathbb{R}^{n \times n}$, thus the space complexity is $O(dn^2)$.

[3] See solution.py
[4] See lup.py
[5] See elementary.py

Time Complexities

Operations	Operation with base Matrices	Operation with representative matrices
Addition	$O(n^2d^2)$	$O(dn^2)$
Subtraction	$O(n^2d^2)$	$O(dn^2)$
Multiplication[6]	$O(n^3d^3)$	$O(dn^3)$
Inversion	$O(n^3d^3)$	$O(dn^3)$

[6] If Strassen's fast multiplication algorithm is used the complexity can be reduced to $O(dn^{2.8})$.