The transpose of a matrix is an operation that takes the rows of a matrix and makes them the columns in the transposed matrix. Effectively, this just means the element at entry gets moved to coordinates in the transposed matrix.

Interestingly, the transpose operation works even on a block matrix. This means that if we have a matrix partitioned into blocks , , , and (where , , and are matrices themselves) as such:

Then to find the transpose of , we can transpose the block matrix entries:

And then transpose each individual block matrix to get the final answer:

This result generalizes for any block-partitioning of a matrix.

Suppose we represented a block matrix in C as an array of pointers, each pointer representing a particular block, where the pointer points to the elements belonging to that block. (Remember multi-level arrays!)

For example,

int block1[4] = {1, 2,

3, 4}

int block2[4] = {1, 4,

8, 4}

int block3[4] = {5, 2,

3, 9}

int block4[4] = {8, 8,

5, 4}

int\* matrix[4] = {block1, block2,

block3, block4}

Would represent this matrix :

Write a function that takes an array of blocks and finds the transpose of the matrix it represents, with tiling.

The idea is to first transpose the matrix of blocks, and then transpose each individual block. For each level of transposing, perform tiling. (You should end up with a loop nested 8 levels deep)

Assume that the matrix is partitioned into equal-sized square blocks, and that the matrix of blocks is also square.

Use the following method signature :

void transpose(int\*\* src, int\*\* dest, int n, int m, int B, int L);

Where

* src is the input matrix
* dest is where the transposed matrix should be stored
* n is the number of rows and columns in the matrix of blocks
* m is the number of rows and columns in each individual block
* B is the tiling factor for the first level of transposing
* L is the tiling factor for the second level of transposing

You can use this function to print a matrix represented in this form and test your output:

void printArray(int\*\* array, int n, int m){

for(int i = 0; i < n; i++){

for(int j = 0; j < m; j++){

for(int k = 0; k < n; k++){

for(int l = 0; l < m; l++){

printf("%d ", array[i\*n + k][j\*m + l]);

}

}

printf("\n");

}

}

}

**Solution :**

void transpose(int\*\* src, int\*\* dest, int n, int m, int B, int L){

for(int i = 0; i < n; i += B){

for(int j = 0; j < n; j += B){

for(int i1 = i; i1 < i + B && i1 < n; i1++){

for(int j1 = j; j1 < j + B && j1 < n; j1++){

//transpose the blocks

dest[n\*i1 + j1] = src[n\*j1 + i1];

//prepare to transpose within the block

int\* src\_sub = (int\*) malloc(m \* m \* sizeof(int));

int\* dest\_sub = dest[n\*i1 + j1];

memcpy(src\_sub, dest[n\*i1 + j1], m \* m \* sizeof(int));

for(int a = 0; a < m; a += L){

for(int b = 0; b < m; b += L){

for(int a1 = a; a1 < a + L && a1 < m; a1++){

for(int b1 = b; b1 < b + L && b1 < m; b1++){

//transpose each element

dest\_sub[a1\*m + b1] = src\_sub[b1\*m + a1];

}

}

}

}

}

}

}

}

}