1 Sparse Matrix Multiplication

The algorithmic task in this question is to multiply two $n \times n$ sparse matrices A and B, to obtain a result $C \leftarrow AB$. By sparse, we mean that almost all of the entries in A and B are 0.

Each matrix is given as a list of triples (i, j, x). We have $A[i][j] = x$ if (i, j, x) is in the list, and $A[i][j] = 0$ otherwise.

Assume that we index beginning in the top left corner, and the matrices are 0-indexed (for example, this means that the top left corner of A is $A[0, 0]$, and the top right corner of A is $A[0, n-1]$). The triples can, of course, be given in any order. However, we assume that the entries of A are listed in column-major order: entries are listed one column after the other, with entries within a column listed row-by-row. Similarly, the entries of B are listed in row-major order: one row at a time; entries within a row sorted by column.

For example, assume we are computing the product

$$\underset{A}{\begin{bmatrix} 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 4 & 0 & 1 \end{bmatrix}} \underset{B}{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 \\ 0 & 7 & 6 & 0 \\ 0 & 0 & 3 & 0 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 0 & 28 & 24 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 70 & 60 & 0 \\ 36 & 0 & 3 & 0 \end{bmatrix}}$$

A would be represented using the triples $(3, 1, 4), (0, 2, 4), (2, 2, 10), (3, 3, 1)$ and B would be represented as $(1, 0, 9), (2, 1, 7), (2, 2, 6), (3, 2, 3)$. The solution to this instance would be

$$C = (3, 0, 36), (0, 1, 28), (0, 2, 24), (2, 1, 70), (2, 2, 60), (3, 2, 3).$$

Consider an algorithm that goes through each nonempty column of A and each nonempty row of B, performing all relevant multiplications and storing the result in C. We assume that A and B are both $n \times n$ matrices. Recall that matrix multiplication can be defined as $C[i][j] = \sum_k A[i][k] \cdot B[k][j]$; we use this notation in our pseudocode for clarity.

Note that this algorithm leaves C as a list of possibly-duplicate tuples. A full matrix-multiplication algorithm would further process C to remove entries that correspond to the same cell (i.e. if $(1, 1, 3)$ and $(1, 1, 5)$ were both in C, we would want to add them to obtain $(1, 1, 8)$). In Algorithm 1 we ignore this step for simplicity.

Implement

Algorithm 1 Sparse Matrix Multiplication

```
for k = 1 to n do                                   ▷ Column k of A; row k of B
    for i = 1 to n do  ▷ These loops iterate over every pair of nonzero items in this row/column
        for j = 1 to n do
            c ← A[i][k] · B[k][j]
            Append (i, j, c) to C
```

1. Implement the above sparse matrix multiplication algorithm efficiently.

By efficient, we mean that your running time should be proportional to the number of non-zero entries in the output (elementary products to be summed to entries in C); in particular it should not be $\Theta(n^3)$ for sufficiently sparse matrices.

A and B are given as lists of triples, one triple per line. The program should take five arguments: n (the size of the matrices), the number of entries in A, the name of the file containing A, the number of entries in B, and finally the name of the file containing B. The first CodeJudge test, which corresponds to the above example, has arguments

```
4 4 A.txt 4 B.txt
```

We have given exercises on CodeJudge to allow you to verify your implementation.

Questions to Discuss

2. Give an example where C *does not* need a final summation step (all entries are for distinct positions in C). Give an example where C *does* need a final summation step.
3. Design and carry out an experiment to compare the performance of your sparse matrix multiplication code with a dense matrix multiplication implementation.