

Assignment 2 Technical Report

2171039 Chaewon Lee

Code 1

This code transposes the matrix B and converts two sparse matrixes into dense matrixes.

First, in line 21, the code defines sparse matrix B. The structure of the sparse matrix is as same as the one in the lecture 3 file. Then, in the next line, the code defines Bt, which would be the transpose matrix of B. At first, Bt has the same value as B.

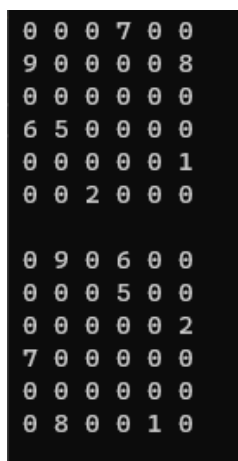
Between the original matrix and transpose matrix, there is an characteristic that the row and column value of each element of sparse matrix is opposite. Therefore, from line 24 to 29, this code swaps the value of row and column of every element in Bt.

And, after the swap, the elements of Bt should be sorted. So, from line 31 to 38, I used selection sort to sort this matrix. By using selection sort, we can guarantee that when there are more than two elements that has the same row value, then their order would not be changed from their original order. After that, rows would be ordered in a descending order, and when the elements have the same row, their columns would also be ordered in a descending order.

In line 40 and 41, code declares two dense matrixes, b and bt. Their size is ROWS x COLS, and inside the matrix, their value is all 0. Then, in line 43 to 50, code checks all the elements inside B and Bt, then assigns the value part of the element to the row and column of the element. This converts sparse matrix B and Bt into dense matrix.

Finally, from line 52 to line 65, the code prints the value of two dense matrices. We can check that the program is working properly.

- Result



```
0 0 0 7 0 0
9 0 0 0 0 8
0 0 0 0 0 0
6 5 0 0 0 0
0 0 0 0 0 1
0 0 2 0 0 0

0 9 0 6 0 0
0 0 0 5 0 0
0 0 0 0 0 2
7 0 0 0 0 0
0 0 0 0 0 0
0 8 0 0 1 0
```

Code 2

This code dynamically allocates the memory to triple pointer A and B and makes N x N x N size 3d array. Then, it adds the value of A to B, and frees the memory of A and B.

First, the code defines two matrices, A and B by function `mem_alloc_3D_double`.

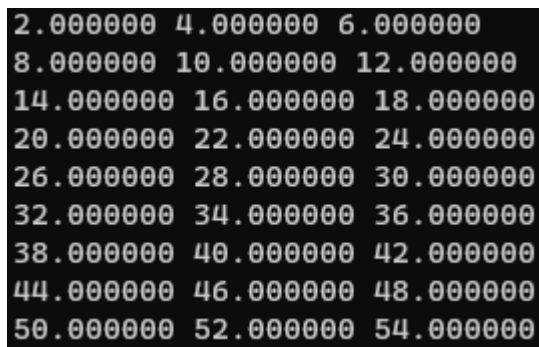
Function `mem_alloc_3d_double` first declares `***double mat` and set it NULL. And it allocates the memory of `(**double) * N` to `mat`, then `(*double) * N` to every `mat[i]`, and `(double) * N` to every `mat[i][j]`. Finally, it returns `mat`.

From line 48 to 56, code assigns the value inside A and B with nested loop. As the loop operates, the int value `cnt` increases. Therefore, the value inside the matrix would be like this: 1 2 3 4 5 6 ...

In line 59, this calls the function `addition_3D`. This function gets two `double***` as an argument. And, by triple nested loop, this adds the value of `A[i][j][k]` to `B[i][j][k]`. Since array variable is a pointer, we can change the value inside of it by directly using them as an argument.

From line 61 to 68, the code prints the value of B. By the result of print, we can sure that the program is working properly. Finally, we should free the memory that we dynamically allocated. To do this, we must free the memory from the small one to the big one. So, first, with nested loop, the code frees `A[i][j]` and `B[i][j]`, then `A[i]` and `B[i]` with loop, and finally, A and B.

- Result



```
2.000000 4.000000 6.000000
8.000000 10.000000 12.000000
14.000000 16.000000 18.000000
20.000000 22.000000 24.000000
26.000000 28.000000 30.000000
32.000000 34.000000 36.000000
38.000000 40.000000 42.000000
44.000000 46.000000 48.000000
50.000000 52.000000 54.000000
```

Explanation: The value inside A and B are both 1, 2, 3, 4, ... as I explained before.

Therefore, the result of $A + B$ would be 2, 3, 6, 8, ... as you can see from the screenshot above.