Data Structures and Algorithms for Engineers

Programming Assignment 4: Sparse Matrix

Task Description:

1. Load two sparse matrices from an input file.
2. Perform addition, subtraction, and multiplication on the matrices.

Note: Read the instructions carefully before you attempt to write the solution.

Instructions

1) Download the code and sample data for this assignment from this location.
   Unzip this file to a folder on the VM(or similar environment). Organize the code and the sample input into the following locations:
   */home/dsa/*sparse_matrix/code/src/
   */home/dsa/*sparse_matrix /code/bin/
   */home/dsa/*sparse_matrix /sample_inputs/

2) Implement code to:
   a) Read a sparse matrix from a file. The format of the file will be:

      ```
      rows=8433
      cols=3180
      (0, 381, -694)
      (0, 128, -838)
      (0, 639, 857)
      (0, 165, -933)
      (0, 1350, -89)
      ```

      The first row gives the number of rows. The second row gives the number of columns. From the third row onwards, there is one entry in parenthesis with row, column, and the integer value separated by commas. All other values in the matrix will be zero by default. For example, in the given sample, the number of rows is 8433, the number of columns is 3180. Row 0 and column 381 has the value -694. Row 0 and column 128 has the value -838, and so on.

   b) Your goal is to implement a data structure that optimizes both memory and run time while storing such large matrices. The code needs be implemented in the following functions of SparseMatrix.h and SparseMatrix.cpp:

      ```
      SparseMatrix(char* matrixFilePath)
      SparseMatrix(int numRows, int numCols)
      getElement(int currRow, int currCol)
      setElement(int currRow, int currCol, int value)
      ```

   c) You are free to define other data structures, classes, and helper functions in SparseMatrix.h and SparseMatrix.cpp. You cannot use std::template packages. You can reuse code, or classes that YOU have written. You may also reuse codes that have been given in class. However, you must document and cite the sources for such codes. Furthermore, you must inform the TAs of the codes you have reused from class-based examples. Ensure that all your code or classes are in these two files and you are not creating any other .h or .cpp files for submission.

d) Your code will be tested to perform operations like addition, subtraction, and multiplication. These operations are already implemented in the code shared with you. They make a call to the four functions given in point b) above. Refer to lines 47 to 58 of homework.cpp to see how your code will be tested.

e) A few samples of input files and result files are given in zip file.

f) Your code must handle following variations in the input file:
   i) If the file has any whitespaces on some lines, these should be ignored.
   ii) If the file has the wrong format i.e., different kind of parenthesis, or floating point values, the code must throw an std::invalid_argument("Input file has wrong format") error and stop.

Submissions:

1) Files SparseMatrix.h and SparseMatrix.cpp must be zipped into a single file "Assignment4.zip". You must NOT create any other source files. These are the only two source files you will submit.
2) Submit zip file on gradescope and canvas.
3) Any number of submissions are allowed up to the due date and time. Your last submission will be used for grading.

Grading method:

1) If your code runs and generates an output file in the correct format for each input file, you get submission points.
2) If your method generates correct results for each test file, you get points for correctness.
3) We will review your source code to examine the internal documentation and award points for proper use of meaningful internal documentation.
4) We will measure the memory consumption in Bytes (submitted_memory). We will take the memory used by our implementation (our_memory). Memory score will be based on the ratio: (our_time /submitted_time) * max score for memory. If your memory usage is less than our method, you get more points. The maximum points you can get here is "max score for memory".
5) We will measure the run time in milliseconds (submitted_time). We will take the time used by our implementation (our_time). Your run-time score will be based on the ratio: (our_time /submitted_time) * max score for run-time. The maximum points you can get here is "max score for run-time".
6) See table below for max score on run-time and memory.

Table 1: Marking Rubrics

Item	**Criteria**	**Points Awarded**
Output: Does the code run? Are the output files generated with the filenames and formats as specified in the problem?	Points obtained	
	Max Points	***5***
Correctness: Is the right output generated?	Points obtained	

	Max Points	***13***
Quality internal documentation. Does documentation aid code understandability?	Points obtained	
	Max Points	***6***
Timing	Time taken to run (ms)	
	Comparison with other students (divide by min of others)	
	Max. Timing score	***13***
Memory	Memory consumed (Bytes)	
	Comparison with other students (divide by min of others)	
	Max. Memory score	***13***
	Total points obtained	
	Total for Task	**50**