

Sparse matrix

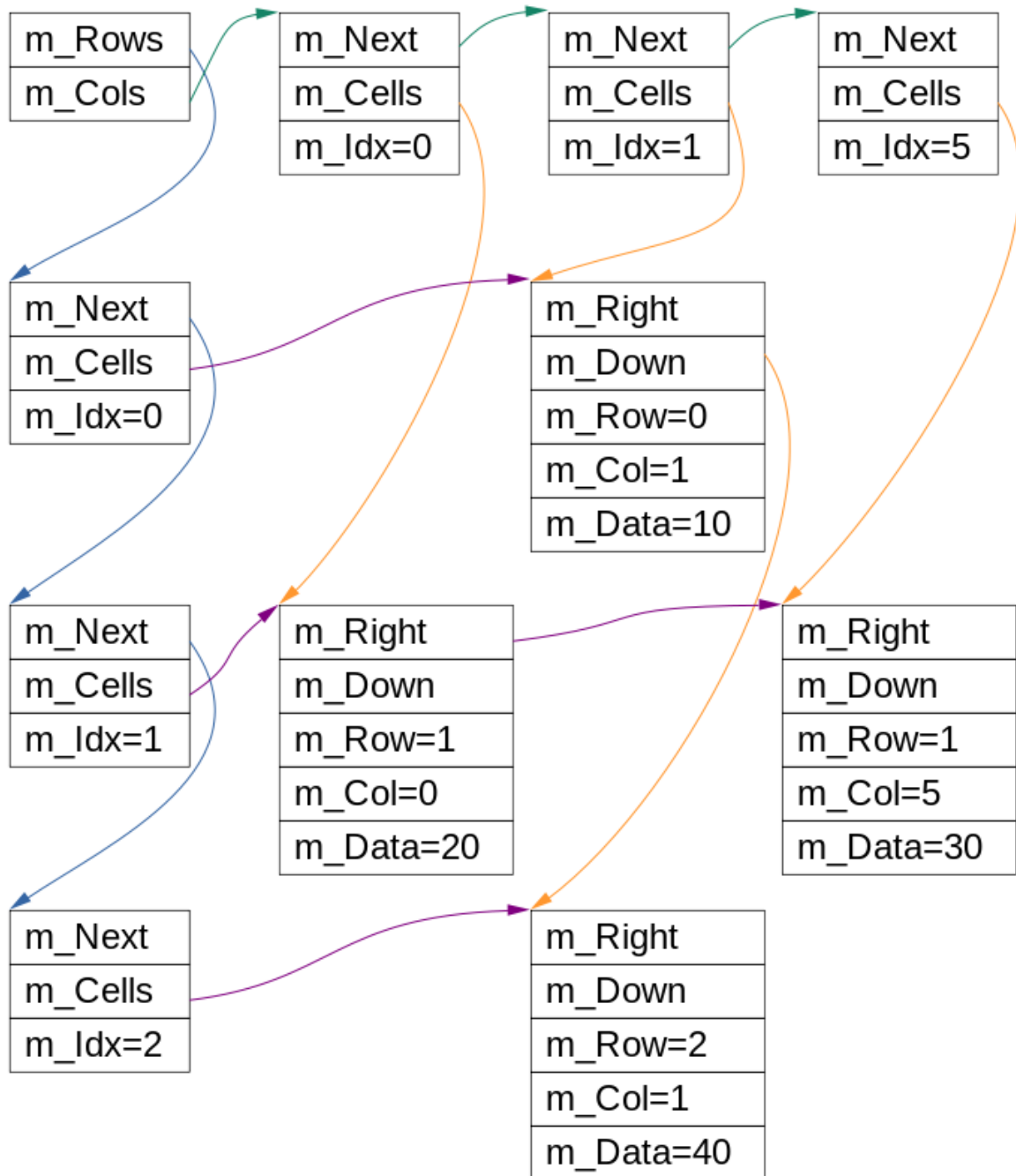
Submission deadline:	2024-01-01 11:59:59	1194001.899 sec
Late submission with malus:	2024-01-01 23:59:59 (Late submission malus: 100.0000 %)	
Evaluation:	5.1946	
Max. assessment:	5.0000 (Without bonus points)	
Submissions:	2 / 20 Free retries + 10 Penalized retries (-10 % penalty each retry)	
Advices:	1 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)	

The task is to implement a set of functions to handle sparse matrices. A sparse matrix is a matrix where there is set only a small fraction of its values. The remaining values are not set (and are usually considered zero). Since majority of the elements is not set, it makes sense to represent the matrix in a form more efficient than a 2D array. We will use linked lists in this homework.

The matrix itself is represented by structure TSPARSEMATRIX. The structure holds two pointer that point to the single linked lists with the description of the existing (non-empty) rows and columns; the rows and columns are described in structure TROWCOL. Both rows and columns are stored in an increasing row (column) index. Further, the TROWCOL structure contains a pointer m_Cells, this pointer points to the elements in the corresponding matrix row or column.

Structure TCELL describes a single element of the matrix. The structure holds the position (row and column), element value and two pointers: the next element in the same row and the next element in the same column. The elements in these lists are again stored in an ascending order (increasing value of column or row, respectively). The figure below depicts the structure with elements:

```
m[0, 1] = 10
m[1, 0] = 20
m[1, 5] = 30
m[2, 1] = 40
```



TSPARSEMATRIX

a data structure defined in the testing environment. Your implementation must use the structure, however, it must not modify it in any way. The structure represents a sparse matrix. The fields are:

- **m_Rows** - pointer to the linked list that describes the existing matrix rows,
- **m_Cols** - pointer to the linked list that describes the existing matrix columns.

TROWCOL

a data structure defined in the testing environment. Your implementation must use the structure, however, it must not modify it in any way. The structure is used to implement a linked list that describes the existing matrix rows and columns. The fields are:

- **m_Next** - next row (column) of the matrix, `nullptr` for the last row (column),
- **m_Idx** - row or column index,
- **m_Cells** - a linked list of elements that form one row (column) of the matrix. Since linked list of **TROWCOL** only list existing rows (columns), the field **m_Cells** is never `nullptr`.

TCELL

a data structure defined in the testing environment. Your implementation must use the structure, however, it must not modify it in any way. The structure is used to implement one element of the matrix. The fields:

- `m_Right` - pointer to the next element in the same row (with higher column index) or `nullptr` for the last element of the row,
- `m_Down` - pointer to the next element in the same column (with higher row index) or `nullptr` for the last element of the column,
- `m_Row` - row index,
- `m_Col` - column index,
- `m_Value` - the value of the element.

`initMatrix (m)`

the function initializes an empty matrix - it sets both row and column pointers to `nullptr`.

`addSetCell (m, row, col, value)`

the function adds n element to the matrix. The element is located at (row, col) and its value is set to value. If there already was an element at (row, col), then the function simply replaces the stored value. On the other hand, the function needs to create the element and connect it into the linked lists if the position (row, col) was not set yet.

`removeCell (m, row, col)`

the function remove the matrix element at(row, col). If the element did not exist, the function returns false and leaves the matrix unchanged. If the element existed, the function removes the element and updates the affected linked lists. Caution, you may need to update the description of the existing rows/columns when removing the last defined element in a row/column.

`freeMatrix (m)`

the function frees the memory allocated by the matrix.

Submit a source file with the implementation of the above functions. Further, the source file must include your auxiliary functions which are called from the required functions. Your functions will be called from the testing environment, thus, it is important to adhere to the required interface. Use the attached sample code as a basis for your development, complete the required functions and add your required auxiliary functions. There is an example main with some tests in the attached code. These test cases will be used in the basic test. Please note the header files as well as main is nested in a conditional compile block (`#ifdef/#endif`). Please keep these conditional compile block in place. They are present to simplify the development. When compiling on your computer, the headers and main will be present as usual. On the other hand, the header and main will "disappear" when compiled by Progtest. Thus, your testing main will not interfere with the testing environment's main.

Your function will be executed in a limited environment. There are limits on both time and memory. The exact limits are shown in the test log of the reference. A reasonable implementation of the naive algorithm shall pass both limits without any problems.

Advice:

- Download the attached sample code and use it as a base for your development.
- The main function in your program may be modified (e.g. a new test may be included). The conditional compile block must remain, however.
- There is macro `assert` used in the example main function. If the value passed to `assert` is nonzero (true), the macro does nothing. On the other hand, if the parameter is zero, the macro stops the execution and reports the line, where the test did not match (and shall be fixed). Thus, the program ends silently when your implementation passes all the tests correctly.
- This program cannot be used for code review.

Sample data:

Download

Submit:

Browse...

No file selected.

Submit

☒ Reference

• Evaluator: computer

- Program compiled
- Test 'Zakladni test podle ukazky': success
 - result: 100.00 %, required: 100.00 %
 - Total run time: 0.000 s (limit: 4.000 s)
 - Mandatory test success, evaluation: 100.00 %
- Test 'Test meznich hodnot': success
 - result: 100.00 %, required: 50.00 %
 - Total run time: 0.000 s (limit: 4.000 s)
 - Mandatory test success, evaluation: 100.00 %
- Test 'Test nahodnymi daty': success
 - result: 100.00 %, required: 50.00 %
 - Total run time: 0.699 s (limit: 4.000 s)
 - Mandatory test success, evaluation: 100.00 %
- Test 'Test nahodnymi daty + mem debugger': success
 - result: 100.00 %, required: 50.00 %
 - Total run time: 0.122 s (limit: 2.000 s)
 - Mandatory test success, evaluation: 100.00 %
- Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00 * 1.00)
- Total percent: 100.00 %
- Early submission bonus: 0.50
- Total points: 1.00 * (5.00 + 0.50) = 5.50

		Total	Average	Maximum	Function name
SW metrics:	Functions:	9	--	--	--
	Lines of code:	137	15.22 ± 10.21	35	removeCell
	Cyclomatic complexity:	39	4.33 ± 3.68	12	removeCell

2

2023-12-17 19:18:24

Download

Submission status:

Evaluated

Evaluation:

5.1946

• Evaluator: computer

◦ Program compiled

◦ Test 'Basic test with example input data': success

▪ result: 100.00 %, required: 100.00 %

▪ Total run time: 0.000 s (limit: 4.000 s)

▪ Mandatory test success, evaluation: 100.00 %

◦ Test 'Borderline test': success

▪ result: 100.00 %, required: 50.00 %

▪ Total run time: 0.000 s (limit: 4.000 s)

▪ Mandatory test success, evaluation: 100.00 %

◦ Test 'Random test': success

▪ result: 97.33 %, required: 50.00 %

▪ Total run time: 0.310 s (limit: 4.000 s)

▪ Mandatory test success, evaluation: 97.33 %

▪ Failed (invalid output) [Unlock advice (31 B)]

▪ Failed (invalid output) [Unlock advice (27 B)]

▪ Failed (invalid output) [Unlock advice (27 B)]

▪ Failed (invalid output) [Unlock advice (27 B)]

▪ Failed (invalid output) [Unlock advice (24 B)]

▪ Failed (invalid output) [Unlock advice (24 B)]

▪ Failed (invalid output) [Unlock advice (24 B)]

▪ Failed (invalid output) [Unlock advice (33 B)]

◦ Test 'Random test + mem debugger': success

▪ result: 97.04 %, required: 50.00 %

▪ Total run time: 0.078 s (limit: 2.000 s)


▪ Mandatory test success, evaluation: 97.04 %

▪ Failed (invalid output) [Unlock advice (30 B)]

▪ Failed (invalid output) [Unlock advice (30 B)]

▪ Failed (invalid output) [Unlock advice (30 B)]

▪ Failed (invalid output) [Unlock advice (30 B)]

▪  Failed (invalid output)

row 2 too long

▪ Failed (invalid output) [Unlock advice (33 B)]

▪ Failed (invalid output) [Unlock advice (22 B)]

▪ Failed (invalid output) [Unlock advice (33 B)]

◦ Overall ratio: 94.45 % (= 1.00 * 1.00 * 0.97 * 0.97)

• Total percent: 94.45 %

• Early submission bonus: 0.50

• Total points: 0.94 * (5.00 + 0.50) = 5.19

SW metrics:

Functions:

5

--

-- --

Lines of code:

384

76.80 ± 68.90

200

main

Cyclomatic complexity:

194

38.80 ± 54.74

147

main

1	2023-12-17 18:37:12	Download
Submission status: Evaluated		
Evaluation: 0.0000		
<ul style="list-style-type: none">• Evaluator: computer<ul style="list-style-type: none">◦ Program compiled◦ Test 'Basic test with example input data': success<ul style="list-style-type: none">▪ result: 100.00 %, required: 100.00 %▪ Total run time: 0.000 s (limit: 4.000 s)▪ Mandatory test success, evaluation: 100.00 %◦ Test 'Borderline test': Abnormal program termination (Segmentation fault/Bus error/Memory limit exceeded/Stack limit exceeded)		

- Total run time: 0.008 s (limit: 4.000 s)
- Mandatory test failed, evaluation: 0.00 %
- Overall ratio: 0.00 % (= 1.00 * 0.00)
- Total percent: 0.00 %
- Early submission bonus: 0.50
- Total points: 0.00 * (5.00 + 0.50) = 0.00

		Total	Average	Maximum	Function name
SW metrics:	Functions:	5	--	--	--
	Lines of code:	384	76.80 ± 68.90	200	main
	Cyclomatic complexity:	197	39.40 ± 54.55	147	main