NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DTSD Lab03

dloTspmatrix: distributed Sparse Matrix processing on resource-constrained IoT devices

v1.5

Pedro Maló

pmm@fct.unl.pt

Sparse Matrices

Fundamentals, Applications, Supports

What is a Sparse Matrix ?

"A matrix is sparse if many of its coefficients are zero. The interest in sparsity arises because its exploitation can lead to enormous computational savings and because many large matrix problems that occur in practice are sparse." [1]

- A sparse matrix is a matrix comprised of **mostly zero/null values**
- Sparse matrices are distinct from matrices with mostly non-zero/non-null values, which are referred to as **dense matrices**
- The **sparsity of a matrix can be quantified** = the number of zero/null values divided by the total number of elements in the matrix

$$\begin{pmatrix} 1.0 & 0 & 5.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3.0 & 0 & 0 & 0 & 0 & 11.0 & 0 \\ 0 & 0 & 0 & 0 & 9.0 & 0 & 0 & 0 \\ 0 & 0 & 6.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7.0 & 0 & 0 & 0 & 0 \\ 2.0 & 0 & 0 & 0 & 0 & 10.0 & 0 & 0 \\ 0 & 0 & 0 & 8.0 & 0 & 0 & 0 & 0 \\ 0 & 4.0 & 0 & 0 & 0 & 0 & 0 & 12.0 \end{pmatrix}$$

[1] Page 1, Direct Methods for Sparse Matrices, Second Edition, 2017 (https://amzn.to/2DcsQVU)

Why Sparse Matrices ?

- **Understanding:** easily compressible/visualisable due to not storing the zero/null elements especially in very large matrices

- **Storage**: lesser non-zero/non-null elements than zeros/null => lesser memory needs to be used to store only those elements

- **Computing time**: Computing time can be saved by logically designing a data structure traversing only non-zero/non-null elements

Applications of Sparse Matrices

- Electric, Electronic and Digital systems analysis & simulation
  - https://www.researchgate.net/publication/265811571_Sparse_Matrix_Methods_for_Circuit_Simulation_Problems
- Sensing, Signal processing, Computer vision, Image/Video processing, etc.
  - https://www.sciencedirect.com/topics/engineering/sparse-signal
- Natural Language Processing (NLP)
  - https://medium.com/@shachiakyaagba_41915/the-sparse-matrix-with-nlp-77901216c649
- Recommender systems
  - https://towardsdatascience.com/why-we-use-sparse-matrices-for-recommender-systems-2ccc9ab698a4
- Machine learning / Artificial Intelligence
  - https://dziganto.github.io/Sparse-Matrices-For-Efficient-Machine-Learning/
- Spreadsheet applications
  - Microsoft Excel spreedsheets are sparse matrices
- Many others …

Sparse Matrices representations

Support efficient modification

- **Dictionary of Keys (DOK)**
  - A dictionary is used where a row and column index is mapped to a value
- **List of Lists (LOL)**
  - Each row of the matrix is stored as a list, with each sub-list containing the column index and the value
- **Coordinate List (COO)**
  - A list of tuples is stored with each tuple containing the row index, column index, and the value

Support efficient operations

- **Compressed Sparse Row (CSR)**
  - The sparse matrix is represented using three one-dimensional arrays for the non-zero values, the extents of the rows, and the column indexes
- **Compressed Sparse Column (CSC)**
  - The same as the Compressed Sparse Row method except the column indices are compressed and read first before the row indices

Sparse Matrices file formats & datasets

- **Rutherford-Boeing (RB)** format – text
  - The most popular mechanism for text-file exchange of sparse matrix data
  - https://math.nist.gov/MatrixMarket/formats.html#hb
  - http://sparse-files.engr.tamu.edu/files/DOC/rb.pdf

- **Matrix Market (MM)** format – text
  - The native exchange format for the Matrix Market
  - https://math.nist.gov/MatrixMarket/formats.html#MMformat

- **Coordinate Text File** format – text
  - Simple and portable method to exchange sparse matrices
  - https://math.nist.gov/MatrixMarket/formats.html#coord

- **HDF5 (Hierarchical Data Format)** – text
  - set of file formats designed to store and organize large amounts of data
  - https://www.hdfgroup.org

Datasets:

- **SuiteSparse Matrix Collection**
  - https://sparse.tamu.edu

dIoTspmatrix

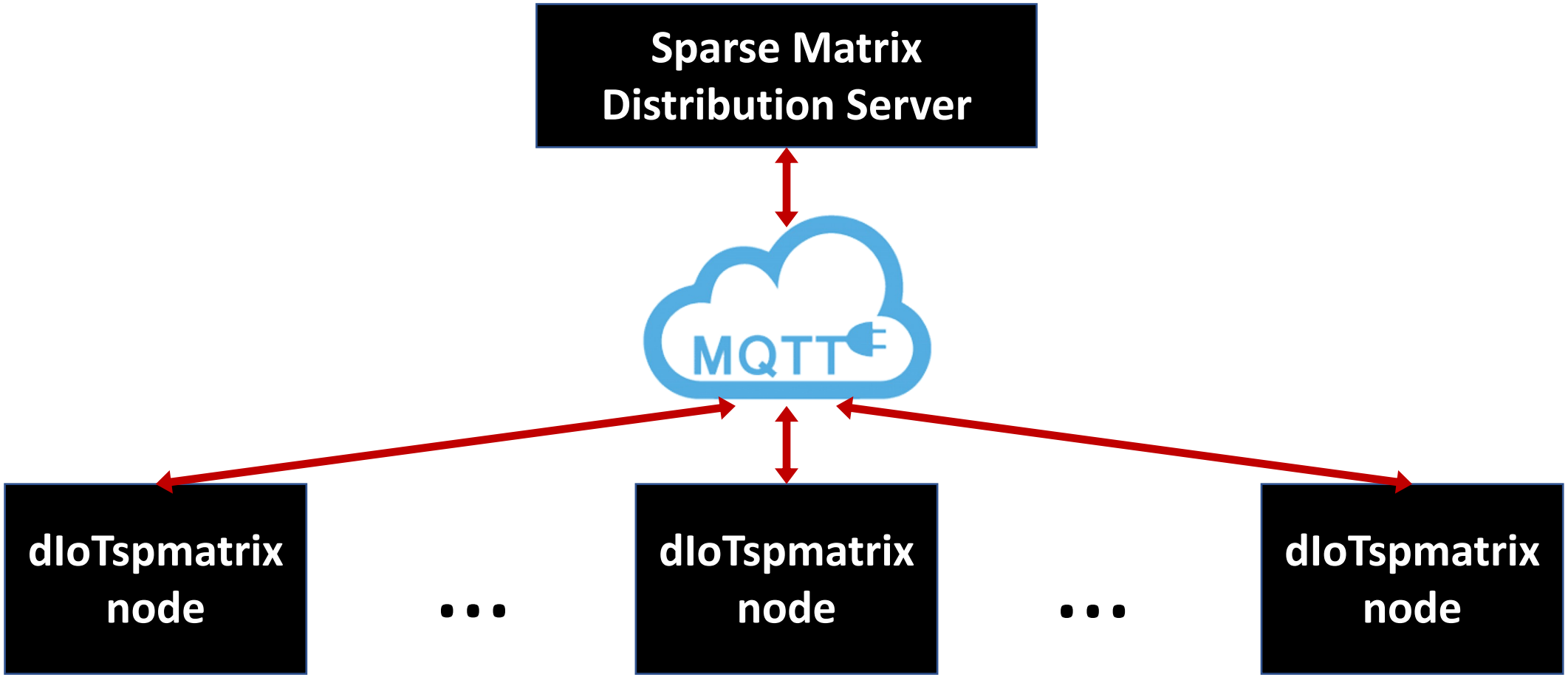
distributed Sparse Matrix processing on resource-constrained IoT devices

dIoTspmatrix digital system

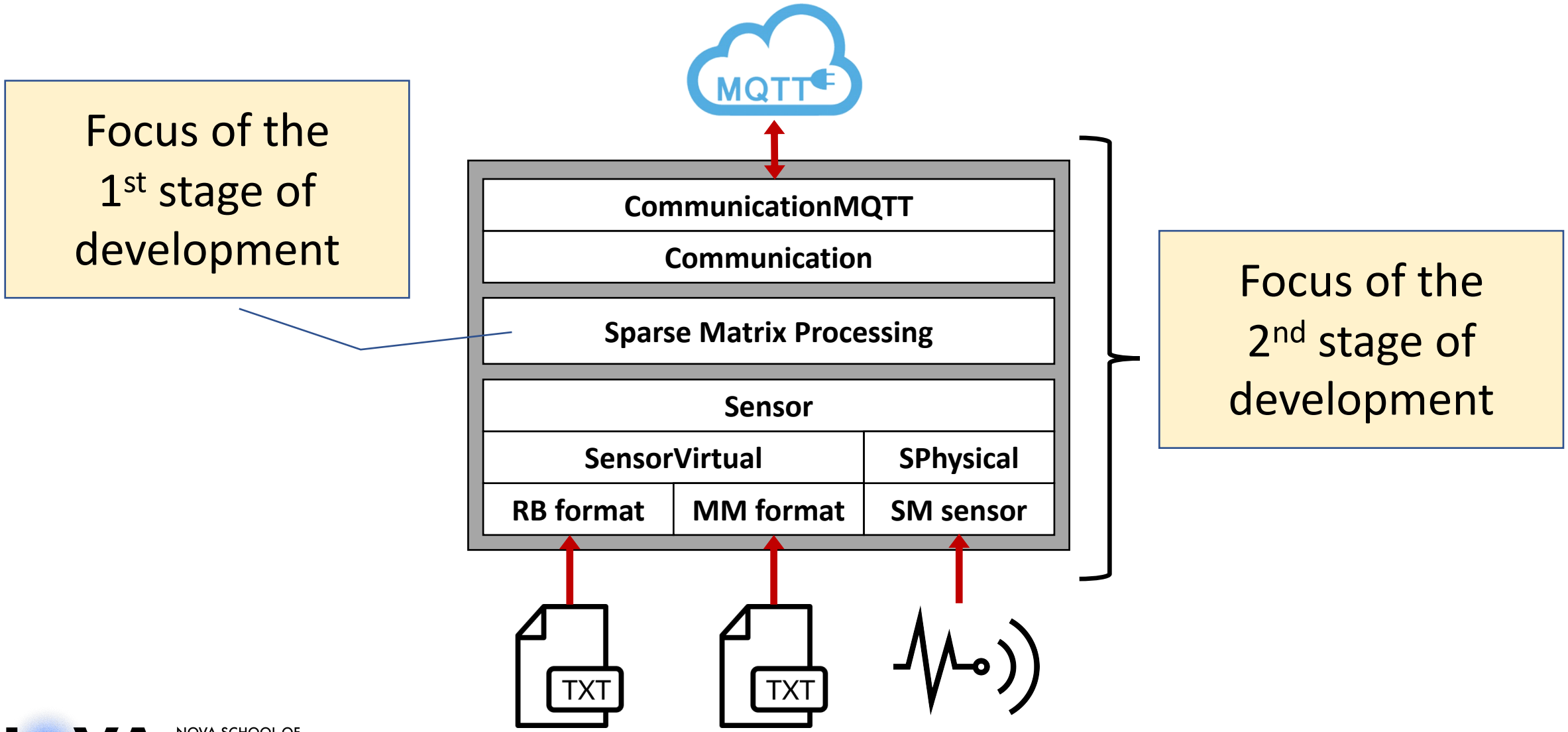
dIoTspmatrix digital system: distributed Sparse Matrix processing on resource-constrained IoT devices

- Distributed processing: distributed processing of operations sparse matrices among several processing nodes
- Matrix and Sparse Matrix operations: compute typicall operations of Matrices (e.g., transpose, add, multiply, etc.) and Sparse Matrices (e.g., eye, compress, etc.)
- Resource-constrained devices: execute Sparce Matrices operations on resource-constrained devices (limited memory, limited performance, limited power)
- Internet-of-Things: make use IoT-based communications (e.g. MQTT)

dIoTspmatrix System View



dIoTspmatrix Node: Logical Architecture



dIoTspmatrix: Development approach

1st stage: BASIC dIoTspmatrix (not for evaluation)

- Implementation using DOK (Dictionary Of Keys) and basic set sparse matrix operations
- Programming Goal: **Basic Python**
  - Individual assignment (by each student)
  - Procedural Programming with Python
  - Basic software testing using pytest
- Programming Approach: **Code-first, test-last**
  - A series of public tests will be made available to students
  - A series of private tests are executed periodically and the results are made available to the the students (private tests source code is not made available)

2nd stage: FULL dIoTspmatrix (evaluation)

- Implementation using additional sparse matrix representations and a wide set of operations
- Programming Goal: **Advanced Python**
  - Group assignment (2~3 students per group)
  - Object-Oriented Programming Python
  - Advanced testing with pytest & unittest
- Program approach: **Test-first, code-after**
  - Initial set of public tests provided for initial software development
  - New public tests made available periodically to students for software code refactoring
  - Batch of exhaustive private tests will be executed at the end of the assignment

1st stage: BASIC dloTspmatrix

(not for evaluation)

Specification: position (position.py) [1/3]

- A spare matrix element is a value identified by a position
- A position is represented by row and column, non-negative integers
- The following operations manipulate positions:
  - **Create position**: position_create
  - **Check if is position**: position_is
  - **Get the row of a position**: position_row
  - **Get the column of a position**: position_col
  - **Compare if two positions are equal**: position_equal
  - **Represent a position as text**: position_str

Note1: It is up to the developer to select the most adequate ADT (Abstract Data Type) for representing positions

Note2: The above-mentioned operations enforce an abstraction barrier to access a position ADT, i.e., should always be used for interfacing the position ADT

Specification: position (position.py) [2/3]

- **def position_create(row: int, col: int) -> position**
  - Create a position based on the *row* and a *column* set as input parameters
  - If arguments are invalid => raise exception ValueError with message ‘position_create: invalid arguments’

- **def position_is(pos: position) -> bool**
  - Validate if the input parameter *pos* is a valid position, returning True if yes and False if otherwise

- **def position_row(pos: position) -> int**
  - Retrieve the row associated with the position *pos* passed as parameter to the function
  - If arguments are invalid => raise exception ValueError with message ‘position_row: invalid arguments’

Specification: position (position.py) [3/3]

- **def position_col(pos: position) -> int**
  - Retrieve the column associated with the position *pos* passed as parameter to the function
  - If arguments are invalid => raise exception ValueError with message ‘position_col: invalid arguments’

- **def position_equal(pos1: position, pos2: position) -> bool**
  - Compare if the two positions (*pos1* and *pos2*) are equal, returning True if yes and False if otherwise
  - If arguments are invalid => raise exception ValueError with message ‘position_equal: invalid arguments’

- **def position_str(pos: position) -> str**
  - Get the representation of the position *pos* as a text string with the format ‘(row, column)’
  - If arguments are invalid => raise exception ValueError with message ‘position_str: invalid arguments’

Specification: spmatrix (spmatrix.py) [1/4]

- **A sparse matrix has a zero/null value and non-zero/non-null elements**
  - **A sparse matrix element is a value identified by a position**
    - **An element position is represented by the position ADT (Abstract Data Type)**
    - **An element value is a floating-point number**
  - **A sparse matrix has a zero/null that is a floating-point number**
- The following operations manipulate *spmatrix* sparse matrices:
  - Create a sparse matrix; Check if is sparse matrix
  - Get the zero/null of a sparse matrix; Set a new zero/null to a sparse matrix
  - Get a value from a sparse matrix; Set a value to the sparse matrix
  - Create a copy of a sparse matrix; Get the dimension of a sparse matrix
  - Get the sparsity of a sparse matrix; Represent a sparse matrix as text;
  - Get a row from a sparse matrix; Get a column from a sparse matrix; Get the diagonal of a sparse matrix
- **The sparse matrix representation (ADT, Abstract Data Type) to be used in the 1st stage of development is DOK (Dictionary Of Keys)**

Specification: spmatrix (spmatrix.py) [2/4]

- **def spmatrix_create(zero: float = 0) -> spmatrix**
  - Create a new sparse matrix with *zero* parameter as zero/null – the default zero of a sparse matrix is 0
  - The input parameter zero is a float but can also be inputted as int as : integers ∋ real numbers
  - If arguments are invalid => raise exception ValueError with message “spmatrix_create: invalid arguments”

- **def spmatrix_is(mat: spmatrix) -> bool**
  - Validate if the input parameter *mat* is a valid sparse matrix, returning True if yes and False if otherwise

- **def spmatrix_zero_get(mat: spmatrix) -> float**
  - Get the zero/null of the sparse matrix *mat* passed as input parameter
  - If arguments are invalid => raise exception ValueError with message “spmatrix_zero_get: invalid arguments”

- **def spmatrix_zero_set(mat: spmatrix, zero: float)**
  - Change the zero/null element of the sparse matrix *mat* to *zero* value passed as input parameter
  - The input parameter zero is a float but can also be inputted as int as : integers ∋ real numbers
  - All the existing elements in the sparse matrix *mat* which are equal to the new zero/null are to be removed
  - If arguments are invalid => raise exception ValueError with message “spmatrix_zero_set: invalid arguments”

Specification: spmatrix (spmatrix.py) [3/4]

- **def spmatrix_value_get(mat: spmatrix, pos: position) -> float**
  - Retrieve the value from the sparse matrix *mat* from the given position *pos* passed as input parameter
  - If arguments are invalid => raise exception ValueError with message “spmatrix_value_get: invalid arguments”

- **def spmatrix_value_set(mat: spmatrix, pos: position, val: float)**
  - Set a new value to the sparse matrix *mat* from at given position *pos* with value *val* passed as input parameters
  - If arguments are invalid => raise exception ValueError with message “spmatrix_value_set: invalid arguments”

- **def spmatrix_copy(mat: spmatrix) -> spmatrix**
  - Create a copy of the sparse matrix *mat* passed as input parameter
  - If arguments are invalid => raise exception ValueError with message “spmatrix_copy: invalid arguments”

- **def spmatrix_dim(mat: spmatrix) -> [tuple[position, position], ()]**
  - Get the dimension of the sparse matrix *mat* passed as input parameter as a tuple with a pair of positions containing the minimum and maximum coordinates of the sparse matrix elements, or an empty tuple if the sparse matrix contains no elements
  - If arguments are invalid => raise exception ValueError with message “spmatrix_dim: invalid arguments”

Specification: spmatrix (spmatrix.py) [4/4]

- **def spmatrix_sparsity(mat: spmatrix) -> float**
  - Get the sparsity of the sparse matrix *mat* as the score of zero/null elements divided by the total number of elements (dimension)
  - The sparsity of an empty sparse matrix is 1 as all elements in the sparse matrix are indeed zero/null
  - If arguments are invalid => raise exception ValueError with message “spmatrix_sparsity: invalid arguments”
- **def spmatrix_str(mat: spmatrix, format: str) -> str**
  - Get the representation of the sparse matrix *mat* as a text string, one line of text per matrix line and for the overall matrix dimension
  - The values of the elements of the sparse matrix are represented with the formatting defined in the *format* input parameter (e.g., ‘%.1f’)
  - If arguments are invalid => raise exception ValueError with message ‘spmatrix_str: invalid arguments’
- **def spmatrix_row(mat: spmatrix, row: int) -> spmatrix**
  - Retrieve the row (*row* number passed as parameter) of the sparse matrix *mat* as a new sparse matrix
  - If arguments are invalid => raise exception ValueError with message ‘spmatrix_row: invalid arguments’
- **def spmatrix_col(mat: spmatrix, col: int) -> spmatrix**
  - Retrieve the column (*column* number passed as parameter) of the sparse matrix *mat* as a new sparse matrix
  - If arguments are invalid => raise exception ValueError with message ‘spmatrix_column: invalid arguments’
- **def spmatrix_diagonal(mat: spmatrix) -> [spmatrix, ...]**
  - Retrieve the diagonal of the sparse matrix *mat* as a new sparse matrix considering the sparse matrix dimension
  - If arguments are invalid => raise exception ValueError with message ‘spmatrix_diagonal: invalid arguments’
  - If sparse matrix *mat* is not square => raise exception ValueError with message ’spmatrix_diagonal: matrix not square’

Testing: position public tests (with pytest)

```
import pytest


from position import *


def test_position_create_valid_1_2():
   assert position_create(1,2) is not None


def test_position_create_valid_1000_0():
   assert position_create(1000,0) is not None


def test_position_create_invalid_args_minus1():
   try:
      position_create(-1, -1)
   except ValueError as error:
      assert str(error) == 'position_create: invalid arguments'


def test_position_create_invalid_args_emptytuple():
   try:
      position_create((), ())
   except ValueError as error:
      assert str(error) == 'position_create: invalid arguments'


def test_position_is_true():
   assert position_is(position_create(1, 2)) is True


def test_position_is_false():
   assert position_is(1.2) is False


def test_position_row():
   assert position_row(position_create(1, 2)) == 1


def test_position_col():
   assert position_col(position_create(1, 2)) == 2


def test_position_equal_same():
   assert position_equal(position_create(1, 2), position_create(1, 2))
is True


def test_position_equal_different():
   assert position_equal(position_create(1, 2), position_create(2, 1))
is False


def test_position_str():
   assert position_str(position_create(1, 2)) == '(1, 2)'


if __name__ == '__main__':
   pytest.main()
```

Testing: spmatrix public tests (with pytest) [1/2]

```
import pytest


from spmatrix import *


def test_spmatrix_create_with_zero_as_default():
    assert spmatrix_create() is not None


def test_spmatrix_create_with_zero_as_1_0():
    assert spmatrix_create(1.0) is not None


def test_spmatrix_is_of_empty_matrix():
    assert spmatrix_is(spmatrix_create()) is True


def test_spmatrix_is_of_non_matrix_float_1():
    assert spmatrix_is(1.0) is False


def test_spmatrix_get_of_empty_matrix_with_zero_as_default():
    assert spmatrix_zero_get(spmatrix_create()) == 0.0


def test_spmatrix_get_of_empty_matrix_with_zero_as_2():
    assert spmatrix_zero_get(spmatrix_create(2)) == 2.0


def test_spmatrix_copy_of_empty_matrix_with_zero_as_2():
    assert spmatrix_zero_get(spmatrix_copy(spmatrix_create(2)))==2.0
```

```
def test_spmatrix_is_after_spmatrix_copy_after_spmatrix_create():
    assert spmatrix_is(spmatrix_copy(spmatrix_create())) is True


def test_spmatrix_value_set_and_spmatrix_value_get():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,2), 12.5)
    assert spmatrix_value_get(mat, position_create(1,2)) == 12.5
    spmatrix_value_set(mat, position_create(2,1), 5.0)
    assert spmatrix_value_get(mat, position_create(2,1)) == 5.0


def
test_spmatrix_value_get_after_replacing_value_with_spmatrix_value_get(
):
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,2), 12.5)
    assert spmatrix_value_get(mat, position_create(1,2)) == 12.5
    spmatrix_value_set(mat, position_create(1,2), 5.0)
    assert spmatrix_value_get(mat, position_create(1,2)) == 5.0


def test_spmatrix_dim_of_empty_matrix():
    assert spmatrix_dim(spmatrix_create()) == ()
```

Testing: spmatrix public tests (with pytest) [2/2]

```
def test_spmatrix_dim_of_matrix_with_one_element():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,2), 5)
    dim = spmatrix_dim(mat)
    assert position_str(dim[0]) == '(1, 2)'
    assert position_str(dim[1]) == '(1, 2)'


def test_spmatrix_sparsity_of_m2x2_diagonal_matrix():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,1), 12.5)
    spmatrix_value_set(mat, position_create(2,2), 5.0)
    assert spmatrix_sparsity(mat) == 0.5


def
test_spmatrix_sparsity_of_matrix_with_1_element_after_element_removal_
using_spmatrix_zero_set():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,2), 12.5)
    spmatrix_value_set(mat, position_create(2,1), 5.0)
    spmatrix_zero_set(mat, 12.5)
    assert spmatrix_sparsity(mat) == 0.0


def test_spmatrix_str_of_m2x2_diagonal():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,1), 12.5)
    spmatrix_value_set(mat, position_create(2,2), 5.0)
    assert spmatrix_str(mat, "%.1f") == '12.5 0.0\n0.0 5.0'
```

```
def test_spmatrix_row_m2x2_diagonal():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,1), 12.5)
    spmatrix_value_set(mat, position_create(2,2), 5.0)
    mat_row = spmatrix_create()
    spmatrix_value_set(mat_row, position_create(1,1), 12.5)
    assert spmatrix_row(mat, 1) == mat_row


def test_spmatrix_col_m2x2_diagonal():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,1), 12.5)
    spmatrix_value_set(mat, position_create(2,2), 5.0)
    mat_col = spmatrix_create()
    spmatrix_value_set(mat_col, position_create(2,2), 5.0)
    assert spmatrix_col(mat, 2) == mat_col


def test_spmatrix_diagonal_m2x2_diagonal_zero():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,1), 12.5)
    spmatrix_value_set(mat, position_create(2,2), 5.0)
    assert spmatrix_diagonal(mat) == mat


def test_spmatrix_diagonal_m2x2_anti_diagonal_zero():
    mat = spmatrix_create()
    spmatrix_value_set(mat, position_create(1,2), 12.5)
    spmatrix_value_set(mat, position_create(2,1), 5.0)
    mat_diagonal = spmatrix_create()
    assert spmatrix_diagonal(mat) == mat_diagonal


if __name__ == '__main__':
    pytest.main()
```

Let's start (programming & learning) !

Workspace preparation:

- **Check if git is installed in your system** - Open terminal and type git – if not install, install git https://git-scm.com
- **CONFIGURE YOUR GIT IDENTITY:**
  - git config --global user.name "John Doe"
  - git config --global user.email johndoe@example.com
- **Open GitHub account** & Create a **private** repository **dloTspmatrix_[student_number]**
- Add **Prof. Pedro Maló (Github id: pmnmalo)** as Collaborator to the new repository
- **Configure your programming environment** (VSCode, PyCharm) to the Github repository
- **MAKE SURE YOU ARE USING PYTHON 3.9 (or superior version)**

Programming / Coding:

- Think about the ADT (Abstract Data Type) for representing a position
- Code and test (using public tests) the position related operations
- Define additional own tests for comprehensive testing of position related operations
- Think about the DOK ADT (Abstract Data Type) for representing sparse matrices
- Code and test the spmatrix related operations
- Define additional own tests for comprehensive testing of spmatrix related operations
- Document your code with docstrings (https://peps.python.org/pep-0257/)

The “Imitation Game” – rules

- **You are not allowed to use any external python modules !!!**
  - The assignment is fairly simple and can be fully done with native python features – the goal is to learn and using external modules is “cheating”
- **The private tests execute around midnight, every day !!!**
  - Individual tests’ execution are limited to 60 seconds – this is to make sure that execution ends for any one function that is taking long time to execute.
- **The private tests (done in pytest) execute on python3.9 !!!**
  - Beware differences of python3.9+ and previous versions – setup environment to python3.9+
- **The “imitation game” is the name of our game of testing**
  - You are playing against the test “machine” that is learning from your mistakes/errors – making tests harder and more comprehensive and you progress
  - Game “imitation game” (aka Turing test) in honour to Alan Turing’s system to test a machine's ability to exhibit intelligent behaviour equivalent to, or indistinguishable from, that of a human.
  - You may recall the “The Imitation Game” movie (premiered in 2014) about Alan Turing and how he built a machine that cracked intercepted Nazi messages coded with the Enigma device
    - *“Turing’s work inspired generations of researchers into what scientists called ‘Turing machines’ – Today, we call them computers”*

position (position.py): an initial just-start push…

```
position = tuple[int, int]


def position_create(row: int, col: int) -> position:
    """
    Create a position
    :param row: row of position
    :param col: column of position
    :return: position
    """
    if not (type(row) is int and row >= 0) or not (type(col) is int and col >= 0):
        raise ValueError('position_create: invalid arguments')
    return row, col
```