Exercise 1 - Sparse Matrix

A sparse matrix is a matrix in which most of the elements are zero. Sparse matrices are often used in numerical analysis, especially for finite element methods. Usually sparse matrices work in two steps 1. The matrix is in uncompressed format, here it is cheap to add an element (change the sparsity pattern of the matrix), however it is expensive to do operations like matrix-vector multiplication 2. You compress the matrix, at this point you cannot change the sparsity pattern of the matrix, but it becomes much cheaper to do matrix-vector multiplication

Step 1 - The abstract class

There are many different ways to implement as sparse matrix. In this laboratory we will show a couple of naive implementations that rely on the STL. For this reason our starting point is the definition of an abstract class where we declare all the methods that we want to have. Namely:

- a getter for number of rows
- a getter for number of cols
- a getter for number of number of non-zero
- a print function
- a `vmult` method that implements matrix-vector multiplication (the dot product of vector with each of the rows of the matrix)
- an access method to read and write the elements of the matrix

In order to make the matrix more flexible use templates to make it usable with different types, just like with `std::vector`s.

Step 2 - Matrix made with maps

After having defined our interface we can really implement our sparse matrix class. In this step the aim is to code a `MapMatrix` sparse matrix, in which each row is represented by a map and all the maps are collected in a `std::vector`. This represent a sparse matrix before compression. For debugging we suggest the following compiler flags

```
g++ hint.cpp -std=c++20 -O0 -g -Wall -Wextra -pedantic -fsanitize=address
```

Namely, the last flag will check that you do not exceed the container's memory bounds. Once you are confident you code is correct you can use the following flags:

```
g++ hint.cpp -std=c++20 -O3 -Wall -Wextra -pedantic -march=native -ffast-math
```

This ensures an aggressive optimization of the code tailored for your specific CPU architecture. This makes your code much faster, but rarely may cause issues due to the modified semantics.

Step 3 - Benchmark and test code

Finally we want to implement some utilities to benchmark the performances of our class and test if the implementation is correct. Namely we implement:

- a function to fill a matrix and given the final size n. Namely it puts on the main diagonal -2 and 1 on the diagonal above and below the main one

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -2 & 1 \\ 0 & \dots & 0 & 0 & 1 & -2 \\ \end{bmatrix}$$

- a function `bool eq(const Vector &lhs, const Vector &rhs)` that checks if two vectors are equal

Then we will check that `matrix.print(std::cout)` effectively prints the expected tri-diagonal matrix that we expect and that the matrix-vector multiplication is correct using the test

$$A \begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ n-1 \\ n \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -n \end{bmatrix}$$

Homework

What happens if instead of using a `std::map` we use `std::unordered_map`? How do you explain this results?

Step 4 - COO format

Now we want to implement the more famous Coordinate list format. COO is a compressed format that stores a list of (row, column, value) tuples. The entries are sorted first by row index and then by column index, to improve random access times. To find the elements in a sorted vector exploit `std::lower_bound`. Since we want to be able to get a COO matrix only from an uncompressed matrix, we will employ the two following techniques: 1. Define the constructor of the `CooMatrix` as `private` 2. Define the `MapMatrix` class as a `friend` of `CooMatrix`

Homework - STL

Fill the code in `main.cpp` to complete four small assignment about the STL. The code will compare your solution against the correct one and tell you if it is correct.