FASE I. COMPRENSIÓN DEL MATERIAL PROPORCIONADO Y DESARROLLO DEL CONSTRUCTOR DE LA MATRIZ DISPERSA POR COLUMNAS

- Descargar los ficheros fuente.
- Compilarlos y ejecutarlos:

```
$ g++ -g main_sparse_matrix_t.cpp -o
main_sparse_matrix_t

$ ./main_sparse_matrix_t < data_sparse_matrix_t.txt
```

- Examinar **todos** los ficheros de cabecera identificando los constructores, el destructor, los métodos para acceder a los atributos, los métodos para lectura desde teclado y escritura a pantalla, e identificar los elementos privados y públicos, tanto atributos como métodos. Identificar la sintaxis de las plantillas. Establecer dónde se efectúa la sobrecarga de operadores, determinar qué operadores han sido sobrecargados.
- Mirar la clase **`matrix_t.hpp`**, y sobre todo los nuevos *getters* (vistos en la clase de problemas):

```
vector_t<T> get_row(const int) const; // copia una fila completa
vector_t<T> get_col(const int) const; // copia una columna
completa
```

- Mirar la clase **`sparse_matrix_t`** que se implementa fundamentalmente con la siguiente estructura que permite un vector de vectores dispersos desarrollados con la última versión (v4) que usa la clase **`dll_t`**:

```
typedef AED::vector_t<AED::sparse_vector_t>
vector_sparse_vector_t;
```

- En la misma clase, mirar los atributos:

```
vector_sparse_vector_t sm_; // vector de vectores dispersos
int  m_;                    // filas de la matriz original
int  n_;                    // columnas de la matriz original
bool by_rows_;              // indica si la matriz se ha creado
                            // por filas (true) o por columnas
(false)
```

- Los siguiente gráficos (PDF) explican la clase **`sparse_matrix_t`** con dos matrices del fichero **`data_sparse_matrix_t.txt`**.

- **Deben desarrollar la parte del constructor de `sparse_matrix_t` que construya la matriz dispersa por columas** (`by_rows == false`). Recuerden que el fichero fuente ya trae implementada la parte de creación de la matriz dispersa por **filas**:

```
sparse_matrix_t::sparse_matrix_t(const matrix_t<double>& M, bool
conf):
sm_(),
m_(M.get_m()),
n_(M.get_n()),
by_rows_(conf)
{
  if (by_rows_) {
    sm_.resize(m_);
    for (int i = 1; i <= M.get_m(); ++i) {
      sparse_vector_t sv(M.get_row(i));
      sm_[i - 1] = sv;
    }
  }
  else {
    // FASE I
    // configuración por columnas
  }
}
```

- **IMPORTANTE**: al igual que los `sparse_vector_t`, las `sparse_matrix_t` empiezan sus **índices en 0**, y no en 1 como en las `matrix_t`. Tengan **CUIDADO** con esta característica al desarrollar los algoritmos que se les piden.

FASE II: DESARROLLO DEL PRODUCTO ESCALAR ENTRE UN `sparse_vector_t` Y UNA COLUMNA DE `matrix_t`

Se debe desarrollar el siguiente método de la clase `sparse_vector_t`:

```
double scal_prod(const matrix_t<double>& M, int j);
```

que implementa el producto escalar entre un vector disperso invocante y una columna `j` de la matriz densa `M` pasada como parámetro.

Nota: si lo piensan bien, este método puede hacerse en una sola línea de código.

FASE III: DESARROLLO DEL PRODUCTO DE MATRICES ENTRE UNA `sparse_matrix_t` Y UNA `matrix_t`

Adicionalmente, debe llevarse a cabo un método de la clase `sparse_matrix_t` para la multiplicación de una matriz dispersa invocante por una matriz densa `A` que se pasa como referencia constante, y el **resultado, que será una matriz densa**, debe almacenarse sobre un **parámetro** `B` que se pasa como referencia. El método tendrá la siguiente cabecera:

```
void sparse_matrix_t::multiply(const matrix_t<double>& A,
matrix_t<double>& B)
```

Nota: este método puede desarrollarse en 4 líneas de código (sin contar las que ya se han puesto en el fichero fuente proporcionado) aprovechando el método realizado en la Fase III.

Evaluación

La calificación de la práctica dependerá de la calidad y eficiencia de la misma.

- Si la práctica funciona correctamente, así como la modificación propuesta por el profesor, y ha concluido la Fase I: **hasta 5**
- Si la práctica funciona correctamente, así como la modificación propuesta por el profesor, y ha concluido la Fase II: **hasta 7**
- Si además de lo anterior, se ha efectuado la Fase III: **hasta 10**

Una vez presentada la modificación de la práctica (y no antes) **deben ser enviados solo los ficheros cpp y hpp** a través del enlace de esta tarea al campus virtual.

Profesorado responsable de las tutorías, entregas y evaluaciones

(Se especificará en la semana de tutorización de la práctica, del 13-04 al 17-04).

Resultado esperado

Si ejecutan la práctica con todas las fases desarrolladas, la salida esperado por pantalla sería:

```
v1= 8: [ 0 0 3.4 0 5.6 0 0 8.9 ]

sv1= 8(3): [ (2:3.4) (4:5.6) (7:8.9) ]

M1= 8x2
0 2.1
3.4 0
5.6 0
7.8 0
0 10.1
11.2 0
13.4 0
0 0

M2= 3x8
0 2.3 0 0 5.6 6.7 0 8.9
0 0 0 0 0 0 0 0
9.1 0 0 12.1 0 0 15.6 0

SM1= 8x2
BY_COLS
```

```
[0] -> 8(5): [ (1:3.4) (2:5.6) (3:7.8) (5:11.2) (6:13.4) ]
[1] -> 8(2): [ (0:2.1) (4:10.1) ]

sv1 * M1(·, 1)= 19.04
sv1 * M1(·, 2)= 56.56

SM2= 3x8
BY_ROWS
[0] -> 8(4): [ (1:2.3) (4:5.6) (5:6.7) (7:8.9) ]
[1] -> 8(0): [ ]
[2] -> 8(3): [ (0:9.1) (3:12.1) (6:15.6) ]

M3 = SM2 * M1
3x2
82.86 56.56
0 0
303.42 19.11
```