Programación Orientada a Objetos

Grado en Ingeniería Informática

Examen de evaluación continua

Curso 2020-21 **Jueves 15 de abril de 2021**

Apellidos: Centeno Ariza **Nombre:** Juan Mariano

Una matriz dispersa es aquella en la que la mayoría de las posiciones están vacías o almacenan elementos nulos. En lugar de almacenar la matriz en un espacio proporcional a su tamaño, se puede conseguir una reducción de espacio considerable almacenándola en un espacio proporcional al número de valores no nulos.

Un objeto de la clase `MatrizDispersa` representa una *matriz bidimensional dispersa* de valores de tipo `double`, la mayoría de los cuales son 0. Una matriz tiene tres atributos, las dos dimensiones (m y n) y un vector (*val*) en el que se guardan los valores que no son 0 junto a sus coordenadas —en cada posición del vector se encuentra una terna (f, c, v) donde v es el valor correspondiente a la fila f y columna c de la matriz. Las ternas se almacenan en orden creciente de filas y columnas.

Ejemplo:

$$\begin{pmatrix} 0{,}0 & 7{,}5 & 0{,}0 & 0{,}0 \\ 0{,}0 & 0{,}0 & 0{,}0 & 0{,}0 \\ 18{,}2 & 0{,}0 & 86{,}37 & 0{,}0 \\ 0{,}0 & 0{,}0 & 0{,}0 & 10{,}25 \\ 0{,}0 & 0{,}0 & 61{,}05 & 0{,}0 \end{pmatrix}$$

$m = 5$
$n = 4$
$val = \{ \{0,\ 1,\ 7.5\},\ \{2,\ 0,\ 18.2\},\ \{2,\ 2,\ 86.37\},$
$\{3,\ 3,\ 10.25\},\ \{4,\ 2,\ 61.05\} \}$

La clase `MatrizDispersa` deberá proporcionar métodos públicos para realizar al menos las siguientes operaciones:

- Crear una matriz nula de dimensiones dadas, por omisión 1×1. No se permitirán conversiones implícitas.
- `asignar()`: asignar un nuevo valor en una posición dada de la matriz.
- `valor()`: leer el valor de una posición dada.
- `filas()`, `columnas()` y `n_valores()`: obtener las dimensiones de la matriz y el número de valores distintos de 0, respectivamente.

Se incluirá, así mismo, el siguiente método privado:

- `buscar()`: comprobar y devolver si una posición de la matriz contiene un valor distinto de 0 (**true**) o no (**false**). Si es distinto de 0, además devuelve por parámetro de salida el índice de este elemento dentro del vector *val*; en caso contrario, devuelve por dicho parámetro el índice del siguiente valor distinto de 0 o `n_valores()` si tal índice no existe.

1. Completa la siguiente definición de la clase `MatrizDispersa` escribiendo únicamente las declaraciones de los miembros requeridos según la especificación anterior. **1,0 p**

```
class MatrizDispersa {
public:
  //...
private:
  struct terna {
    size_t f, c;
    double v;
  };
  size_t m, n;
  std::vector<terna> val;
};
```

2. Define una sobrecarga del operador $<$ para comparar dos objetos de tipo `terna`: $t_1 < t_2$ si y solo si t_1 precede a t_2 siguiendo el orden creciente de filas y columnas. A continuación, utiliza este operador para implementar el método privado *buscar*. **1,5 p**

3. Define en el exterior de la clase `MatrizDispersa` los métodos públicos declarados anteriormente, siguiendo estas pautas: **3,0 p**

   - En las funciones que lo requieran se deberá validar que las coordenadas están dentro del rango de las dimensiones de la matriz y lanzar una excepción estándar de tipo `out_of_range` si no es así.
   - Utiliza los métodos `insert` y `erase` de la clase `vector`. El primero recibe dos parámetros, una posición dada por un iterador y el elemento a insertar. El segundo sólo recibe un iterador que indica la posición del elemento a suprimir.

     Ejemplos:

     ```
     vector<int> v{1, 3, 5, 7, 9};
     v.erase(v.begin());          // v = {3, 5, 7, 9}
     v.erase(v.begin()+3);        // v = {3, 5, 7}
     v.erase(v.end()-1);          // v = {3, 5}
     v.insert(v.begin(), 2);      // v = {2, 3, 5}
     v.insert(v.begin()+2, 4);    // v = {2, 3, 4, 5}
     ```

   - Todo el código se debe escribir tan claro y conciso como sea posible, evitando el uso innecesario de variables, asignaciones u otras instrucciones.

4. Incorpora a `MatrizDispersa` un método para construir una matriz a partir de una lista de inicializadores de tipo `terna` dados en cualquier orden, la cual incluirá al final el elemento de la fila y columna últimas, tenga el valor 0 o no. Las dimensiones de la matriz se deducirán de esta última terna de la lista, que no se almacenará si su valor es 0. Trata de reutilizar la operación *asignar*. Por ejemplo, la matriz del ejemplo inicial se podrá construir como sigue: **1,0 p**

   ```
   MatrizDispersa A { {0, 1, 7.5}, {2, 0, 18.2}, {2, 2, 86.37},
                      {3, 3, 10.25}, {4, 2, 61.05}, {4, 3, 0.0} };
   ```

5. Escribe un fragmento de código en el que se intente crear la matriz dispersa A y seguidamente actualizar el valor $A(8,6)$ a 0. Se deben capturar las posibles excepciones lanzadas e imprimir un mensaje explicativo por el flujo de salida estándar de error. **0,5 p**

$$A = \begin{pmatrix} 0{,}0 & 7{,}5 & 0{,}0 & 0{,}0 \\ 0{,}0 & 0{,}0 & 0{,}0 & 0{,}0 \\ 18{,}2 & 0{,}0 & 86{,}37 & 0{,}0 \\ 0{,}0 & 0{,}0 & 0{,}0 & 10{,}25 \\ 0{,}0 & 0{,}0 & 61{,}05 & 0{,}0 \end{pmatrix}$$

6. Implementa el destructor de la clase `MatrizDispersa`, salvo que pienses que no es necesario, en cuyo caso explica la causa. **0,5 p**

7. Implementa una eficiente función no miembro para intercambiar dos matrices dispersas evitando las copias de objetos. Añade a la clase `MatrizDispersa` cualquier método que consideres necesario y no esté definido. **1,0 p**

8. Escribe la declaración de una sobrecarga del método *valor* que permita tanto leer como actualizar el valor de una posición dada de la matriz. Explica si podría causar algún problema incluir este método en la clase `MatrizDispersa` y, en tal caso, pon un ejemplo. **1,5 p**