Problemas

Tema 2: Algoritmos sobre matrices. (Segunda Parte)

Vectores/matrices comunes o implementados usando contenedores.

Trabajo con Matrices (1)

- Dada la siguiente matriz realice las operaciones que se le indican.

```
const int m = 3;
int C[m][m] = {{5, 1, 3},
               {1, 8, 2},
               {3, 2, 5}};
```

- Impleméntese el método bool `is_symmetric(void)`, el cual devuelve **verdadero** si la matriz es simétrica, o **falso** en caso contrario. Es decir, devuelve verdadero si para todo par (i,j) existe también un par (j,i) igual.

Matriz simétrica

$$\begin{bmatrix} 5 & 1 & 3 \\ 1 & 8 & 2 \\ 3 & 2 & 5 \end{bmatrix}$$

par (i, j) = par (j, i)

Ejemplo previo

```
cout << "Elementos de la submatriz
triangular inferior: " << endl;
for (int i = 0; i < m; i++) {
    for (int j = 0; j <= i; j++)
        cout << B[i][j] << " ";
    cout << endl;
}
```

Ejemplo previo

```
cout << "Elementos de la submatriz
triangular inferior sin diagonal: " <<
endl;
for (int i = 1; i < m; i++) {
    for (int j = 0; j <= i - 1; j++)
        cout << B[i][j] << " ";
    cout << endl;
}
```

Matriz simétrica (1)

```
bool is_symmetric(void) {
    bool is_sym = true;
    for (int i = 1; i < m; i++) {
        for (int j = 0; j <= i - 1; j++) {
            if (C[i][j] != C[j][i])
                is_sym = false;
        }
    }
    return is_sym;
}
```

Matriz simétrica (2)

```
bool is_symmetric(void) {
    bool is_sym = true;
    for (int i = 1; is_sym && (i < m); i++) {
        for (int j = 0; is_sym && (j <= i - 1); j++) {
            if (C[i][j] != C[j][i])
                is_sym = false;
        }
    }
    return is_sym;
}
```

Matriz simétrica (3)

```
bool is_symmetric(void) {
    bool is_sym = true;
    int i = 1;
    while (is_sym && (i < m)) {
        int j = 0;
        while (is_sym && (j <= i - 1)) {
            if (C[i][j] != C[j][i])
                is_sym = false;
            j++;
        }
        i++;
    }
    return is_sym;
}
```

Trabajo con Matrices (2)

- Dada la siguiente matriz realice las operaciones que se le indican.

```
const int m = 3;
int C[m][m] = {{5, 1, 3},
               {-1, 8, 2},
               {-3, -2, 5}};
```

- Impleméntese el método `bool is_antisymmetric(void)`, el cual devuelve **verdadero** si la matriz es antisimétrica, o **falso** en caso contrario. Es decir, devuelve verdadero si para todo par (i,j) existe también un par (j,i) igual pero de signo contrario.

Matriz antisimétrica

$$\begin{bmatrix} 5 & 1 & 3 \\ -1 & 8 & 2 \\ -3 & -2 & 5 \end{bmatrix}$$

par (i, j) = - par (j, i)

Matriz antisimétrica (1)

```
bool is_antisymmetric(void) {
    bool is_asym = true;
    for (int i = 1; is_asym && (i < m); i++) {
        for (int j = 0; is_asym && (j <= i - 1); j++) {
            if (abs(C[i][j]) != abs(C[j][i]) || C[i][j] == C[j][i])
                is_asym = false;
        }
    }
    return is_asym;
}
```

¿Soluciona este código el problema?

Matriz antisimétrica (2)

```
bool is_antisymmetric(void) {
    bool is_asym = true;
    for (int i = 1; is_asym && (i < m); i++) {
        for (int j = 0; is_asym && (j <= i - 1); j++) {
            if (C[i][j] != -C[j][i])
                is_asym = false;
        }
    }
    return is_asym;
}
```

Trabajo con vectores dispersos

Dada los siguientes atributos para representar vectores dispersos:

```
double* val_;  // valor
int*    inx_;  // índice
int     nz_;   // número de elementos <> 0
int     n_;    // tamaño
```

a) Impleméntese el algoritmo para el producto escalar entre dos vectores dispersos **sv1** y **sv2**.

b) Impleméntese el algoritmo para la suma entre dos vectores dispersos **sv1** y **sv2**.

Ejemplo previo
Representación de vectores dispersos

```
// vector normal con 6 elementos
n_: 6
0.0000   0.0000   5.0000   6.0000   0.0000   4.0000
     0        1        2        3        4        5

// vector disperso (sparse vector)
n_: 6
nz_: 3
val_: 5.0000   6.0000   4.0000
inx_:      2        3        5
```

Ejemplo previo

Producto escalar entre un vector denso y un vector disperso

```
double producto = 0;
for (int i = 0; i < nz_; i++)
  producto += val_[i] * v[inx_[i]];
```

a) Producto escalar entre dos vectores dispersos (1)

¿Soluciona este código el problema?

```
double producto = 0;
for (int i = 0; i < min(sv1.nz_, sv2.nz_);
i++)
  producto += sv1.val_[i] * sv2.val_[i];
```

a) Producto escalar entre dos vectores dispersos (2)

```
double producto = 0;
for (int i = 0, j = 0; (i < sv1.nz_) && (j < sv2.nz_);)
  if (sv1.inx_[i] == sv1.inx_[j])
    producto += sv1.val_[i++] * sv2.val_[j++];
  else
    if (sv1.inx_[i] < sv2.inx_[j])
       i++;
    else
       j++;
```

b) Suma entre dos vectores dispersos (1)

¿Soluciona este código el problema?

```
sparse_vector_t suma(max(sv1.n_,sv2.n_));
int k = 0;
for (int i = 0, j = 0; (i < sv1.nz_) && (j < sv2.nz_);)
  if (sv1.inx_[i] == sv1.inx_[j])
    suma[k++] = sv1.val_[i++] + sv2.val_[j++];
  else
    if (sv1.inx_[i] < sv2.inx_[j])
       suma[k++] = sv1.val_[i++];
    else
       suma[k++] = sv2.val_[j++];
suma.nz_ = k;
```

b) Suma entre dos vectores dispersos (2)

¿Soluciona este código el problema?

```
sparse_vector_t suma(max(sv1.n_,sv2.n_));
int k = 0;
for (int i = 0, j = 0; (i < sv1.nz_) && (j < sv2.nz_);)
  if (sv1.inx_[i] == sv1.inx_[j]) {
    double s = sv1.val_[i++] + sv2.val_[j++];
    if (!igual(s, 0.0))
      suma[k++] = s;
  } else
    if (sv1.inx_[i] < sv2.inx_[j])
       suma[k++] = sv1.val_[i++];
    else
       suma[k++] = sv2.val_[j++];
suma.nz_ = k;
```

b) Suma entre dos vectores dispersos (3)

```
sparse_vector_t suma(max(sv1.n_,sv2.n_));
int i = 0, j = 0, k = 0;
while ((i < sv1.nz_) && (j < sv2.nz_))
  if (sv1.inx_[i] == sv1.inx_[j]) {
    double s = sv1.val_[i++] + sv2.val_[j++];
    if (!igual(s, 0.0))
      suma[k++] = s;
  } else
    if (sv1.inx_[i] < sv2.inx_[j])
       suma[k++] = sv1.val_[i++];
    else
       suma[k++] = sv2.val_[j++];
```

```
while (i < sv1.nz_)
  suma[k++] = sv1.val_[i++];
while (j < sv2.nz_)
  suma[k++] = sv2.val_[j++];
suma.nz_ = k;
```

Ejercicios de Estudio Individual

a) Calcular la sumatoria de los elementos de la diagonal principal de una matriz.
b) Calcular la media de los elementos de la diagonal secundaria de una matriz.
c) Mostrar los elementos y la sumatoria de la submatriz triangular inferior de una matriz, sin incluir la diagonal principal.
d) Mostrar los elementos y la sumatoria de la submatriz triangular superior de una matriz, sin incluir la diagonal principal.
e) Algoritmo para la suma entre un vector denso **v** y uno disperso **sv**.
f) Algoritmo para el cálculo del módulo de un vector disperso.