TEMA 5

Vectores multidimensionales

Emma Rollón
erollon@cs.upc.edu

Departamento de Ciencias de la Computación

Índice

Introducción

Un vector multidimensional es un vector cuyos elementos son, a la vez, un vector.

Por ejemplo:

- un vector cuyos elementos son vectores de int sería:

```
vector <vector <int> >                  // hay 2 dimensiones
//  |          |
//  |          +--- segunda dimensión o vector interno
//  +------ primera dimensión o vector externo
```

- un vector cuyos elementos son vectores cuyos elementos son vectores de int sería:

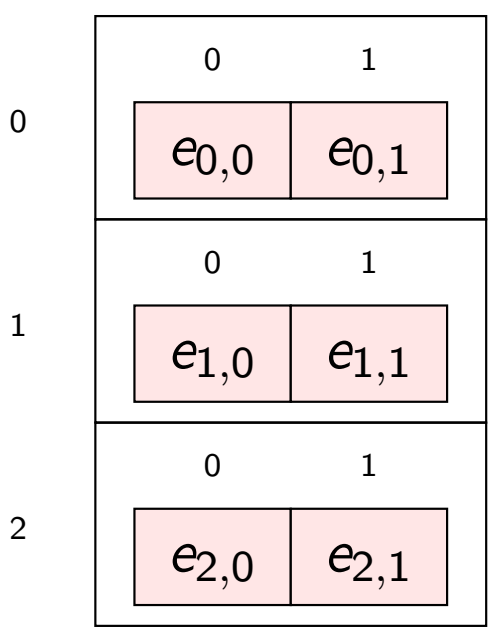
```
vector <vector <vector <int> > >  // hay 3 dimensiones
```

Cada vez que utilizamos la palabra vector decimos que tenemos una nueva dimensión.

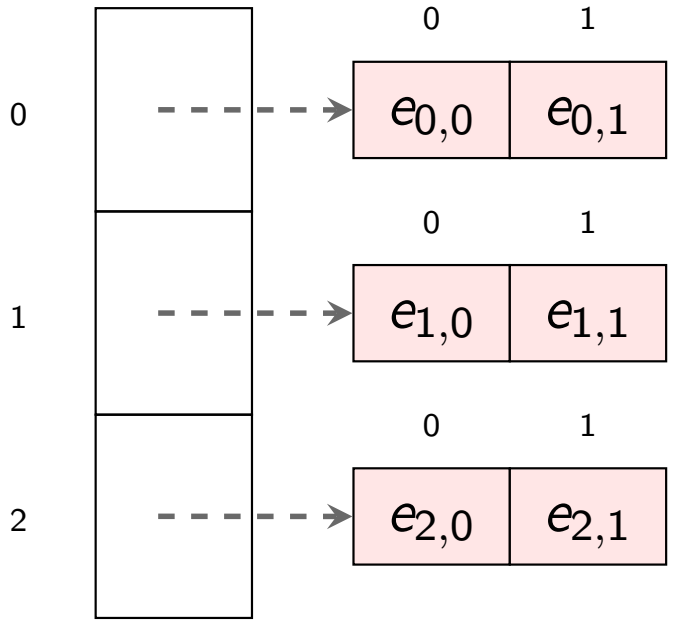
Introducción

Un vector cuyos elementos son un vector de un tipo simple se llama matriz. En otras palabras, un vector de 2 dimensiones es una matriz.

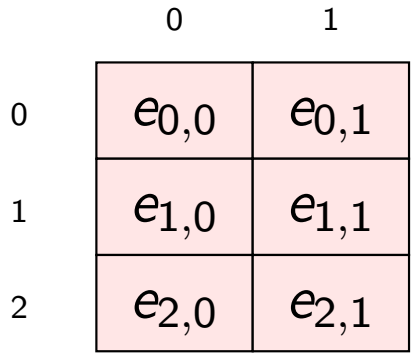
¿Cómo lo representamos gráficamente?



Opción 1



Opción 2



Opción 3

Declaración de una matriz

Sintaxis:

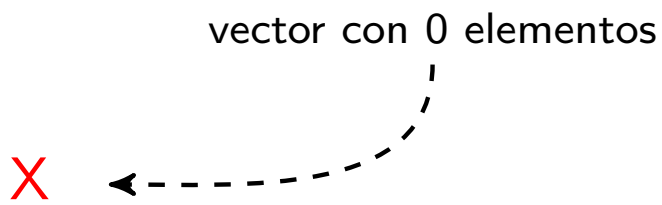
```
#include <vector>          // Biblioteca necesaria

vector< vector <T> > nombre_matriz(S, I);
```

- T: tipo de datos simple
- S: tamaño del vector externo (por defecto es 0)
- I: inicialización del vector interno (si no se indica será un vector con 0 elementos)

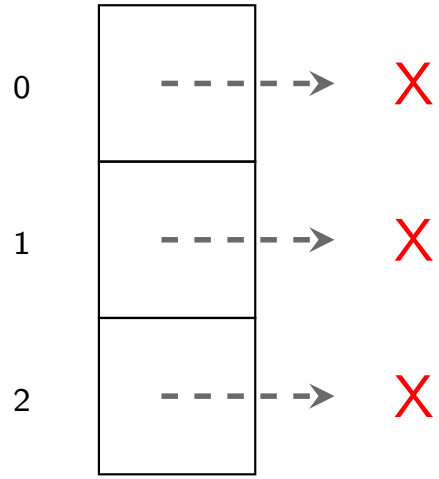
Los parámetros S, I son opcionales:

- Si sólo indicas uno, será S.
- Si indicas dos, serán S, I.

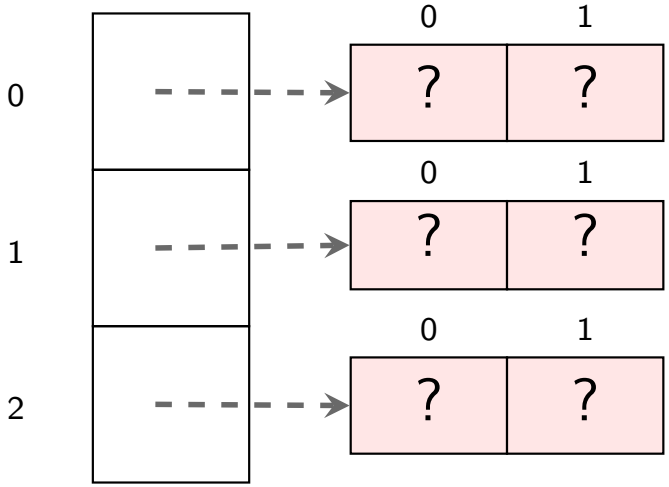
Declaración de una matriz: ejemplos



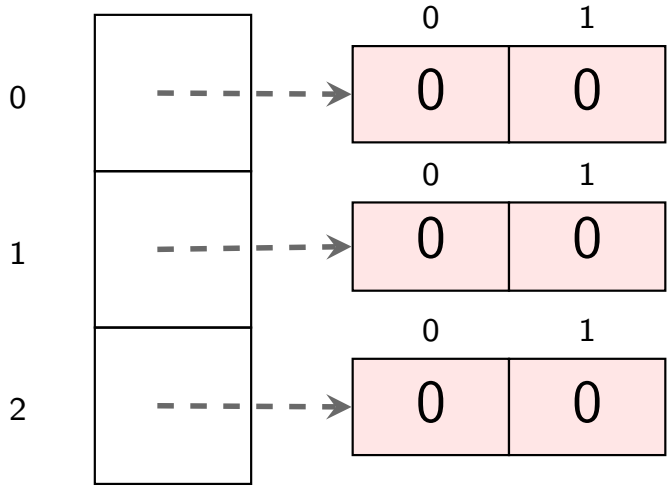
vector <vector <int>> mat;



vector <vector <int>> mat(3);



vector <vector <int>> mat(3, vector<int>(2));



vector<vector<int>> mat(3, vector<int>(2, 0))

Declaración de una matriz: simplificación

Vamos a definir alias para los vectores que representan filas y para el vector que representa la matriz gracias a la instrucción typedef.

```
typedef vector <int> Fila;
typedef vector <Fila> Matriz;
```

Las declaraciones anteriores quedarían como:

- Matriz con 0 filas y 0 columnas:

```
Matriz mat;
```

- Matriz con 3 filas con 0 columnas cada una:

```
Matriz mat(3);
```

- Matriz con 3 filas con 2 columnas cada una en donde el valor de los elementos se desconoce:

```
Matriz mat(3, Fila(2));
```

- Matriz con 3 filas con 2 columnas cada una en donde el valor de todos los elementos es 0:

```
Matriz mat(3, Fila(2, 0));
```

Uso de matrices: acceso

Siempre se accede por índice, en cada una de sus dimensiones:

```
nombre_matriz[E1][E2]
```

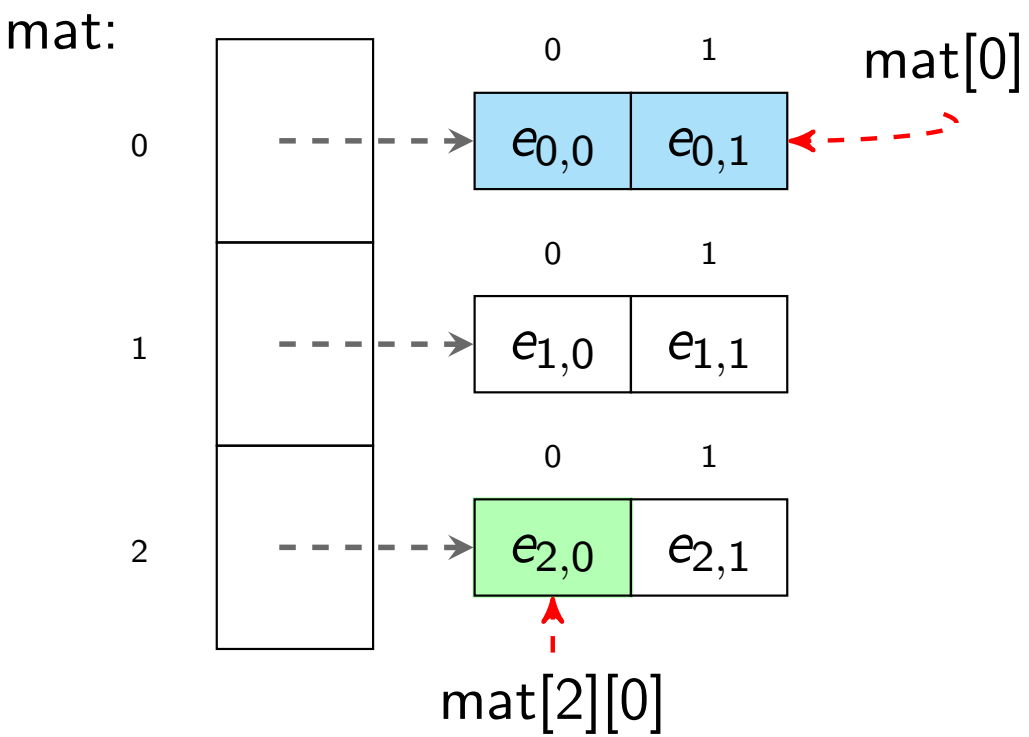
- E1, E2: expresión entera.
- Se accede a la posición E1 del vector externo nombre_matriz (fila E1) y a la posición E2 del correspondiente vector interno (columna E2). Sirve tanto para consultar como para actualizar.

¡Alerta!

Si se accede a una posición no válida: container with out-of-bounds index.

Uso de matrices: acceso

```
int main() {
    vector <vector <int> > mat(3, vector <int> (2, 1));
    vector <int> v = mat[0];      // copio un vector interno en v
    int x = mat[2][0];            // obtengo un elemento del vector interno
    mat[2][0] = 10;               // actualizo elemento del vector interno
    cout << x << " " << mat[2][0] << endl;
}
```



mat:

	0	1
0	$e_{0,0}$	$e_{0,1}$
1	$e_{1,0}$	$e_{1,1}$
2	$e_{2,0}$	$e_{2,1}$

Uso de matrices: saber número de elementos

Número de elementos del vector externo:

```
int n = nombre_matriz.size();
```

- Nos dará el número de filas de la matriz.

Número de elementos de algún vector interno:

```
int m = nombre_matriz[índice].size();
```

- Nos dará el número de columnas de la matriz.
- El índice tiene que ser válido.

```
int main() {
    vector <vector <int> > mat(3, vector <int> (2, 1));
    int n = mat.size();
    int m = mat[0].size();
    cout << n << " " << m << endl;     // escribe 3 2
}
```

Uso de matrices: lectura/escritura de matrices

```
vector <vector <int> > mat;
cin >> mat;     // ERROR de compilación
cout << mat;    // ERROR de compilación
```

Recuerda que cin/cout sólo están definidos sobre tipos de datos simples.

¿Entonces cómo leemos y escribimos una matriz?

Lo haremos haciendo un recorrido en donde visitaremos todos los elementos de la matriz y los iremos leyendo/escribiendo. Vamos a distinguir dos casos:

- Leer/escribir matriz rectangular (todas las filas tienen el mismo número de columnas).
- Leer/escribir matriz irregular (cada fila tiene un número determinado de columnas).

Uso de matrices: lectura/escritura matriz rectangular

```
// Pre: en la entrada nos dan dos enteros que son número de filas f y columnas c
//      de la matriz. A continuación nos dan f líneas, cada una con c enteros
// Post: lee la matriz que se nos da en la entrada, hace una cierta tarea, y la escribe.
int main() {
    int f, c;
    cin >> f >> c;

    // crear matriz
    vector <vector <int> > mat(f, vector <int>(c));

    // inicializar la matriz con los valores de la entrada
    for (int i = 0; i < f; ++i) {
        for (int j = 0; j < c; ++j) cin >> mat[i][j];
    }

    // se realiza la tarea correspondiente
    // ...

    // escribir el contenido de la matriz
    for (int i = 0; i < f; ++i) {
        for (int j = 0; j < c; ++j) cout << mat[i][j] << " ";
        cout << endl;
    }
}
```

Uso de matrices: lectura/escritura matriz irregular

```
// Pre: en la entrada nos dan un entero que son número de filas f. A continuación
//      nos dan f líneas, cada una empieza con un número c que es el número de
//      columnas de esa fila y, a continuación, c enteros que son los
//      elementos de esa fila.
// Post: lee la matriz que se nos da en la entrada, hace una cierta tarea, y la escribe.
int main() {
    int f;
    cin >> f;

    // crear matriz
    vector <vector <int> > mat(f);

    // inicializar la matriz con los valores de la entrada
    for (int i = 0; i < f; ++i) {
        int c;
        cin >> c;
        mat[i] = vector <int>(c);
        for (int j = 0; j < c; ++j) cin >> mat[i][j];
    }

    // se realiza la tarea correspondiente
    // ...

    // escribir el contenido de la matriz
    for (int i = 0; i < f; ++i) {
        int c = mat[i].size();
        for (int j = 0; j < c; ++j) cout << mat[i][j] << " ";
        cout << endl;
    }
}
```

Paso de parámetros

Por valor:

- Todo lo que se ha explicado para un vector unidimensional aplica para los multidimensionales. Es decir, utilizaremos un paso por valor simulado: paso por referencia constante.
- Nada nuevo.

```
int main() {
    Matriz mat(3, Fila(4, 1));
    mi_accion(mat);
}
```

```
void mi_accion(const Matriz& m) {
    ...
}
```

Por referencia:

- Todo lo que se ha explicado para un vector unidimensional aplica para los multidimensionales.
- Nada nuevo.

```
int main() {
    Matriz mat(3, Fila(4, 1));
    mi_accion(mat);
}
```

```
void mi_accion(Matriz& m) {
    ...
    ... m[i][j] = 10;
    ...
}
```

Paso de parámetros: por referencia constante (ejemplo)

Acción que escribe el contenido de una matriz rectangular/irregular:

```
typedef vector<int> Fila;
typedef vector<Fila> Matriz;

// Pre: m es válida, rectangular y no es vacía
// Post: escribe el contenido de m
void escribir_matriz(const Matriz& m) {
    int f = m.size();
    int c = m[0].size();      // el acceso a m[0] es correcto porque m no es vacía
    for (int i = 0; i < f; ++i) {
        for (int j = 0; j < c; ++j) cout << " " << m[i][j];
        cout << endl;
    }
}

// Pre: en la entrada nos dan los datos necesarios para inicializar una matriz de enteros
// Post: lee la matriz que se nos da en la entrada, hace una cierta tarea, y la escribe.
int main() {
    int f, c;
    cin >> f >> c;
    Matriz mat(f, Fila(c));
    // lectura de la matriz
    ...
    // tarea sobre la matriz
    ...
    // escritura de la matriz
    escribir_matriz(mat);
}
```

Paso de parámetros: por referencia (ejemplo)

La lectura de una matriz rectangular/irregular se puede encapsular como una acción. A continuación te damos una opción posible:

```
typedef vector<int> Fila;
typedef vector<Fila> Matriz;

// Pre: m es rectangular y no es vacía;
//   en la entrada nos dan los elementos fila a fila de m
// Post: m está inicializada con los datos de la entrada
void leer_matriz(Matriz& m) {
    int f = m.size();
    int c = m[0].size();      // el acceso a m[0] es correcto porque m no es vacía
    for (int i = 0; i < f; ++i) {
        for (int j = 0; j < c; ++j) cin >> m[i][j];
    }
}

// Pre: en la entrada nos dan dos enteros que son número de filas f (mayor a 0)
//    y columnas c de la matriz. A continuación nos dan f líneas, cada una con c enteros
// Post: lee la matriz que se nos da en la entrada, hace una cierta tarea, y la escribe.
int main() {
    int f, c;
    cin >> f >> c;
    Matriz mat(f, Fila(c));
    leer_matriz(mat);
    // ...
}
```

Retorno de una función

Una función puede retornar un valor de cualquier tipo de datos. En particular, puede retornar un vector multidimensional.

Ejemplo: la lectura de una matriz rectangular/irregular se puede encapsular como una función. A continuación te damos una opción posible:

```
// Pre: en la entrada nos dan dos enteros que son número de filas f (mayor a 0)
//    y columnas c de la matriz. A continuación nos dan f líneas, cada una con c enteros
// Post: m está inicializada con los datos de la entrada
Matriz leer_matriz() {
    int f, c;
    cin >> f >> c;
    Matriz m(f, Fila(c));
    for (int i = 0; i < f; ++i) {
        for (int j = 0; j < c; ++j) cin >> m[i][j];
    }
    return m;
}

// Pre: en la entrada nos dan dos enteros que son número de filas f (mayor a 0)
//     y columnas c de la matriz. A continuación nos dan f líneas, cada una con c enteros
// Post: lee la matriz que se nos da en la entrada, hace una cierta tarea, y la escribe.
int main() {
    Matriz mat = leer_matriz();
    // ...
}
```

Esquemas algorítimicos y orden de visita de los elementos

Recuerda que:

- Si necesariamente tenemos que visitar todos los elementos, será un recorrido.
- Si queremos comprobar una cierta condición sobre la matriz, será una búsqueda.

En ambos esquemas, podemos visitar los elementos de una matriz siguiendo diferentes órdenes. Los más utilizados son:

- Visitar los elementos por filas.
  - Es el que hemos utilizado para leer/escribir matrices.
- Visitar los elementos por columnas.

A continuación encontrarás la solución a diferentes ejercicios de recorrido y búsqueda sobre matrices cuyos valores no tienen un orden específico.

Consulta el material docente de la web de la asignatura para ver un ejemplo de búsqueda en una matriz con valores ordenados.

Sumar uno a todos los elementos de una matriz

$$\begin{pmatrix} 3 & 2 & 4 \\ 1 & 6 & 8 \end{pmatrix} \xrightarrow{\text{sumar 1 a todos los elementos}} \begin{pmatrix} 4 & 3 & 5 \\ 2 & 7 & 9 \end{pmatrix}$$

Solución 1: Recorrer la matriz por filas

- Por cada valor i que corresponde con un índice válido de fila:
- Por cada valor j que corresponde con un índice válido de columna:
- Sumar 1 al elemento mat[i][j]

Código:

```
// Pre: mat es no vacía y válida
// Post: a los elementos originales de mat se ha sumado 1
void suma_uno(Matriz& mat) {
    int n = mat.size();
    int m = mat[0].size();        // acceso correcto porque mat no es vacía
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j <  m; ++j) ++mat[i][j];
    }
}
```

Sumar uno a todos los elementos de una matriz

Generación de índices:

$$\begin{pmatrix} 3 & 2 & 4 \\ 1 & 6 & 8 \end{pmatrix} = \begin{pmatrix} mat[0][0] & mat[0][1] & mat[0][2] \\ mat[1][0] & mat[1][1] & mat[1][2] \end{pmatrix}$$

- Cuando i es 0, j toma los valores 0, 1, 2. Por lo tanto, se accede (en este orden) a las posiciones:

  $$mat[0][0] \qquad mat[0][1] \qquad mat[0][2]$$

  Estas posiciones son las de la primera fila.

- Cuando i es 1, j toma los valores 0, 1, 2. Por lo tanto, se accede (en este orden) a las posiciones:

  $$mat[1][0] \qquad mat[1][1] \qquad mat[1][2]$$

  Estas posiciones son las de la segunda fila.

Sumar uno a todos los elementos de una matriz

Solución 2: Recorrer la matriz por columnas

- Por cada valor j que corresponde con un índice válido de columna:
- Por cada valor i que corresponde con un índice válido de fila:
- Sumar 1 al elemento mat[i][j]

Código:

```
// Pre: mat es no vacía y válida
// Post: a los elementos originales de mat se ha sumado 1
void suma_uno(Matriz& mat) {
    int n = mat.size();
    int m = mat[0].size();        // acceso correcto porque mat no es vacía
    for (int j = 0; j < m; ++j) {
        for (int i = 0; i <  n; ++i) ++mat[i][j];
    }
}
```

Sumar uno a todos los elementos de una matriz

Generación de índices:

$$\begin{pmatrix} 3 & 2 & 4 \\ 1 & 6 & 8 \end{pmatrix} = \begin{pmatrix} mat[0][0] & mat[0][1] & mat[0][2] \\ mat[1][0] & mat[1][1] & mat[1][2] \end{pmatrix}$$

- Cuando j es 0, i toma los valores 0, 1. Por lo tanto, se accede (en este orden) a las posiciones:

  $mat[0][0]$ $mat[1][0]$ primera columna

- Cuando j es 1, i toma los valores 0, 1. Por lo tanto, se accede (en este orden) a las posiciones:

  $mat[0][1]$ $mat[1][1]$ segunda columna

- Cuando j es 2, i toma los valores 0, 1. Por lo tanto, se accede (en este orden) a las posiciones:

  $mat[0][2]$ $mat[1][2]$ tercera columna

Suma de matrices

Escribe una función tal que dadas dos matrices mat1 y mat2 con las mismas dimensiones, retorne una matriz que represente su suma.

Ejemplo:

$$\underbrace{\begin{pmatrix} 3 & 2 & 4 \\ 1 & 6 & 8 \end{pmatrix}}_{mat1} + \underbrace{\begin{pmatrix} 4 & 1 & 6 \\ 9 & 8 & 4 \end{pmatrix}}_{mat2} = \underbrace{\begin{pmatrix} 7 & 3 & 10 \\ 10 & 14 & 12 \end{pmatrix}}_{suma}$$

Su representación con un vector bi-dimensional es:

	0	1	2
0	3	2	4
1	1	6	8

$+$

	0	1	2
0	4	1	6
1	9	8	4

$=$

	0	1	2
0	7	3	10
1	10	14	12

Suma de matrices

Pasos:

- f y c son el número de filas y columnas respectivamente que tendrá la matriz resultado
- Declaramos la matriz resultado suma
- Por cada valor i de 0 a f - 1 (índice válido de fila en suma):
- Por cada valor j de 0 a c - 1 (índice válido de columna en suma):
- Inicializar suma[i][j] con mat1[i][j] + mat2[i][j]
- Retornar suma

```
typedef vector <vector <int> > Matriz;

// Pre: mat1, mat2 matrices no vacías y válidas con las mismas dimensiones
// Post: retorna mat1 + mat2
Matriz suma_matrices(const Matriz& mat1, const Matriz& mat2) {
    int f = mat1.size();
    int c = mat1[0].size();          // acceso correcto porque mat1 es no vacía
    Matriz suma(f, vector<int>(c));
    for (int i = 0; i < f; ++i) {
        for (int j = 0; j < c; ++j) suma[i][j] = mat1[i][j] + mat2[i][j];
    }
    return suma
}
```

Matriz transpuesta

Escribe una función tal que dada una matriz, retorne su transpuesta.

$$mat = \begin{pmatrix} 3 & 2 & 4 \\ 1 & 6 & 8 \end{pmatrix} \qquad mat^T = \begin{pmatrix} 3 & 1 \\ 2 & 6 \\ 4 & 8 \end{pmatrix}$$

La representación de estas dos matrices será:

mat:

	0	1	2
0	3	2	4
1	1	6	8

mat^T:

	0	1
0	3	1
1	2	6
2	4	8

Una solución es recorrer la matriz mat por columnas, ya que cada una de esas columnas será una fila de la transpuesta.

Matriz transpuesta

```
typedef vector <vector <int> > Matriz;

// Pre: mat es una matriz rectangular no vacía y válida
// Post: retorna la matriz transpuesta de mat
Matriz transpuesta(const Matriz& mat) {
    int f  = mat.size();
    int c = mat[0].size();      // correcto (mat no es vacía)
    Matriz t (c, vector <int> (f));
    for (int j = 0; j < c; ++j) {
        for (int i = 0; i < f; ++i) t[j][i] = mat[i][j];
    }
    return t;
}
```

Fíjate que:

- i está asociado a la primera dimensión de mat, pero a la segunda en t
- j está asociado a la segunda dimensión de mat, pero a la primera en t

Matriz simétrica

Escribe una función tal que dada una matriz cuadrada, retorne true si es simétrica, false en caso contrario.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 3 & 1 & 4 & 3 \\ 1 & 6 & 8 & 10 \\ 4 & 8 & 9 & 2 \\ 3 & 10 & 1 & 2 \end{pmatrix} \end{array} \rightarrow \textit{false}$$

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 3 & 1 & 4 & 3 \\ 1 & 6 & 8 & 10 \\ 4 & 8 & 9 & 1 \\ 3 & 10 & 1 & 2 \end{pmatrix} \end{array} \rightarrow \textit{true}$$

Sabemos que una matriz es simétrica si mat[i][j] == mat[j][i] para todos los valores i, j.

Vamos a pensar qué valores de i, j tenemos que visitar para comprobar que una matriz mat es simétrica o no.

Matriz simétrica

$$mat: \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 3 & 1 & 4 & 3 \\ 1 & 6 & 8 & 10 \\ 4 & 8 & 9 & 1 \\ 3 & 10 & 1 & 2 \end{pmatrix} \end{array}$$

$\mathrm{i} = 0$
$$\begin{cases} j = 0: & \cancel{mat[0][0] == mat[0][0]} \\ j = 1: & mat[0][1] == mat[1][0] \\ j = 2: & mat[0][2] == mat[2][0] \\ j = 3: & mat[0][3] == mat[3][0] \end{cases}$$

$\mathrm{i} = 1$
$$\begin{cases} j = 0: & \cancel{mat[1][0] == mat[0][1]} \\ j = 1: & \cancel{mat[1][1] == mat[1][1]} \\ j = 2: & mat[1][2] == mat[2][1] \\ j = 3: & mat[1][3] == mat[3][1] \end{cases}$$

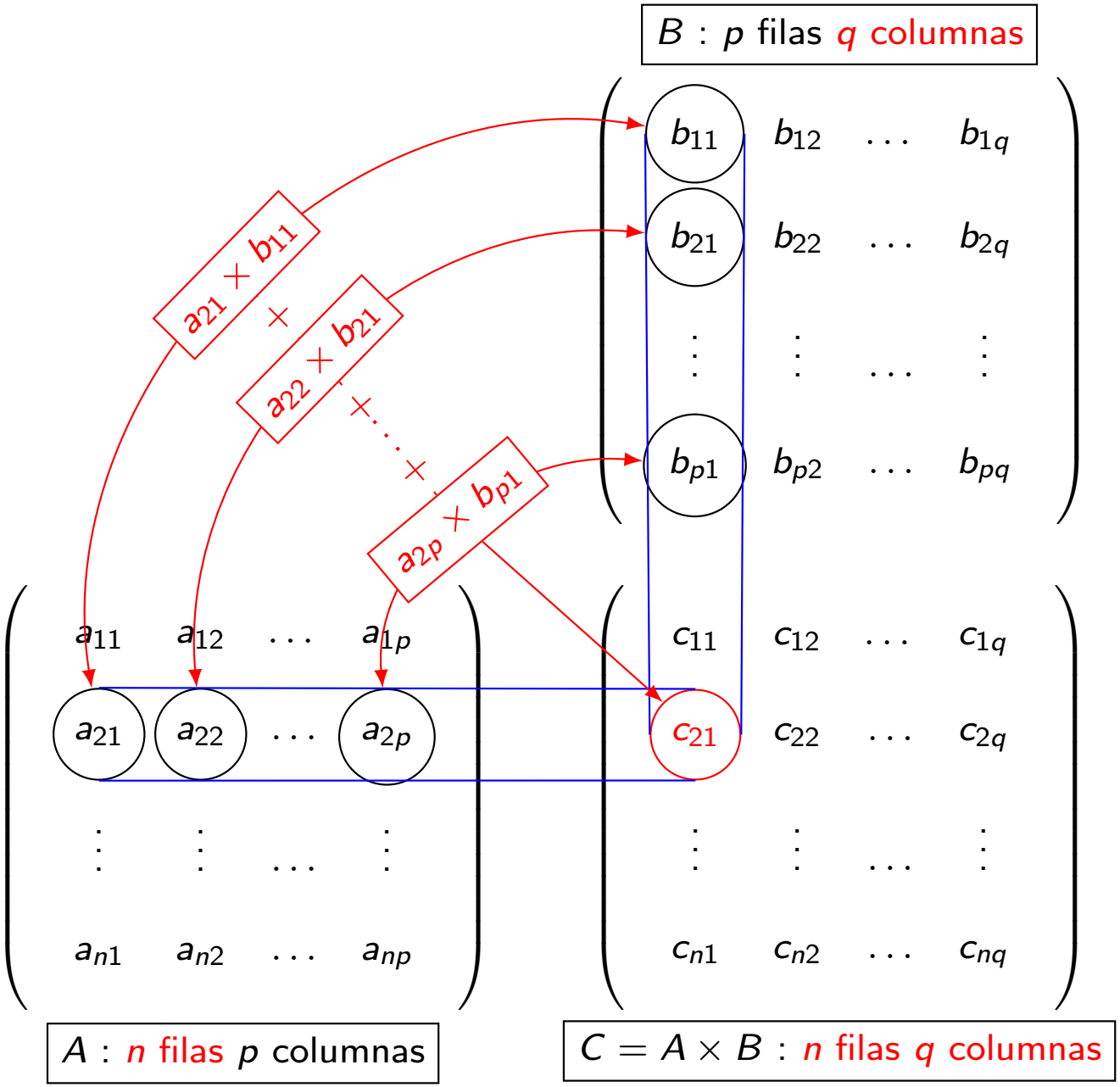
$\mathrm{i} = 2$
$$\begin{cases} j = 0: & \cancel{mat[2][0] == mat[0][2]} \\ j = 1: & \cancel{mat[2][1] == mat[1][2]} \\ j = 2: & \cancel{mat[2][2] == mat[2][2]} \\ j = 3: & mat[2][3] == mat[3][2] \end{cases}$$

$i = 3$
$$\begin{cases} j = 0: & \cancel{mat[3][0] == mat[0][3]} \\ j = 1: & \cancel{mat[3][1] == mat[1][3]} \\ j = 2: & \cancel{mat[3][2] == mat[2][3]} \\ j = 3: & \cancel{mat[3][3] == mat[3][3]} \end{cases}$$

Matriz simétrica

```
// Pre: mat es una matriz cuadrada válida
// Post: retorna true si la matriz es simétrica, false si no.
bool simetrica(const Matriz& mat) {
    int n = mat.size();    // es una matriz de n*n (cuadrada)
    for (int i = 0; i < n - 1; ++i) {
        for (int j = i + 1; j <  n; ++j) {
            if (mat[i][j] != mat[j][i]) return false;
        }
    }
    return true;
}
```

Producto de matrices



Producto de matrices

```
// Pre: A, B matrices no vacías con dimensiones n*p y p*q
// Post: A * B que tendrá dimensión n*q
Matriz producto_matrices(const Matriz& A, const Matriz& B) {
    int n = A.size();
    int p = B.size();        // equivalente: int p = A[0].size();
    int q = B[0].size();
    Matriz C(n, vector <int>(q, 0));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < q; ++j) {
            // calcular el valor del elemento C[i][j]
            // en la declaración lo hemos inicializado a 0
            for (int k = 0; k < p; ++k) {
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
            }
        }
    }
    return C;
```

Para mantener la concordancia con el dibujo y la notación matemática hemos utilizado nombres de variables en mayúscula. Sin embargo, recuerda que los nombres de variables tienen que ser en minúsculas.