Matrices

Curso de Estructura de datos y algoritmos 1 – Universidad ORT Uruguay

Conceptos generales

1. ¿Qué son las matrices?

Al igual que los vectores, las matrices permiten almacenar datos del mismo tipo. Sin embargo, las matrices son vectores de dos dimensiones.
Se podría referir un a una matriz como **un vector** que contiene **vectores**, con la restricción que los vectores que están adentro de ese vector son del mismo largo.

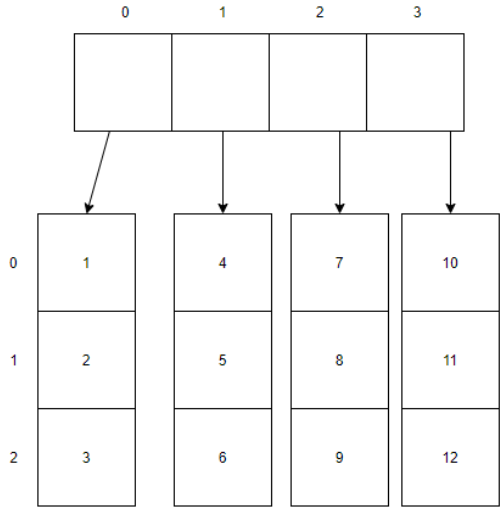
Por ejemplo:
[[1,2,3] , [1,2] , [1,2,3,4]] No es una matriz, porque los vectores internos no tienen el mismo largo.

[[1,2,3] , [4, 5, 6], [7,8,9], [10,11,12]] Esto si es una matriz, ya que todos los vectores internos tienen el mismo largo.

Esta restricción, permite representar a la matriz como el formato de una tabla:

1	2	3
4	5	6
7	8	9
10	11	12

Sin embargo, otra representación válida sería:



La matriz, al ser un vector de vectores, podría representarse de esta forma porque, utilizando el ejemplo del diagrama, si obtengo el primer elemento del vector, obtendría otro vector: [1,2,3]

Dependiendo del contexto, podemos utilizar cualquiera de las representaciones. Por ejemplo, la primera representación es conveniente cuando queremos obtener o visualizar datos de una matriz.
La segunda representación podría ser conveniente para crear la matriz dinámicamente.

2. ¿Cómo inicializo una matriz?

Al igual que los vectores, las matrices pueden tener largos estáticos o dinámicos.

A continuación, se muestra un ejemplo de la utilización de una matriz con largo estático:

```
#define FILA 5
#define COLUMNA 4

int main(){
   int matriz[FILA][COLUMNA];
        int cont = 1;
        for (int i = 0; i < FILA; i++) {
             for (int j = 0; j < COLUMNA; j++) {
                  matriz[i][j] = cont++;
             }
        }

        for (int i = 0; i < FILA; i++) {
             for (int j = 0; j < COLUMNA; j++) {
                  cout << matriz[i][j] << " ";
             }
             cout << endl;
        }
}
```

Para crear matrices dinámicamente, es conveniente usar la segunda representación de matrices que mencionamos anteriormente como referencia.

Primero es necesario reservar memoria para el vector que contendrá los demás vectores.
Para seguir con el mismo ejemplo anterior, este vector tendrá el largo de la fila, pero esto arbitrario, podría ser el de la columna. La única diferencia está en como se van llenando los datos y como se recorre la matriz: primero todas las filas de izquierda a derecha (el caso del ejemplo anterior) o primero todas las columnas de arriba hacia abajo.

```
int main(){
   int filas = 5;
   int columnas = 4;

   int** matriz = new int* [filas];
}
```

A la declaración de la matriz se le agregan dos ***** ya que, al ser un vector de vectores, es un puntero a otro puntero.

Como vimos anteriormente, la sintaxis de una variable de tipo puntero es: <***tipo***>* <***nombre_variable***>;
En este caso <***tipo***> es ***int**** por lo que también es un puntero.
Entonces cada elemento de la matriz, guardará un ***int****, por eso es *new* ***int**** .

Una vez que tenemos reservada la memoria para el vector que contendrá a los demás vectores, hay que reservar memoria para cada vector interno:

```
int main(){
   int filas = 5;
   int columnas = 4;

   int** matriz = new int* [filas];
   for(int i=0; i<filas; i++){
        matriz[i] = new int[columnas];
   }
}
```

En **cada** posición reservamos ***columnas*** lugares de memoria.

Para asignar datos y recorrerlos, el código sería el siguiente:

```
int main() {

   int filas = 4;
   int columnas = 5;

   int** matriz = new int* [filas];
   int cont = 1;

   for (int i = 0; i < filas; i++) {
        matriz[i] = new int[columnas];
        for (int j = 0; j < columnas; j++) {
             matriz[i][j] = cont++;
        }
   }

   for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
             cout << matriz[i][j] << " ";
        }
        cout << endl;
   }
}
```

RECORDAR: *SI RESERVO, LIBERO*
En otras palabras, si utilicé ***new*** en algún momento hay que utilizar ***delete***.
Como la matriz es un vector de vectores, hay que liberar la memoria de cada vector interno, y luego liberar memoria del vector que los contiene:

```
   for (int i = 0; i < filas; i++) {
        delete[] matriz[i];
   }

   delete[] matriz;
```

Ejercicios

1. Implementar una función *esDiagonalPrincipal* que, dada una matriz de booleanos, retorne verdadero si y solo si la diagonal principal tiene valores verdaderos **y** el resto de la matriz valores falsos.

   Ejemplo:

   i.
   [[T , F , F],
   [F , T , F],
   [F , F , T]]

   Verdadero

   ii.
   [[T , T ,T],
   [T , T , T],
   [T, T , T]]

   Falso

```
//PRE: Recibe una matriz de booleanos con la cantidad de
//     filas y columnas
//POS: True ⇔ Diagonal Principal tiene valores T y el resto
//     de la matriz F
bool esDiagonalPrincipal(bool **mat, int filas, int columnas);
```

2. Implementar un procedimiento imprimirFilCol que, dada una matriz de enteros y un entero, imprima por pantalla la fila y columna de ese entero en la matriz.
   Ejemplo:
   elem = 6
   [[1 , 2 , 3],
   [4 , 5 , 6],
   [7 , 8 , 9]]

   Imprime: "Fila: 2 – Columna: 3"

```
//PRE: 'mat' no contiene repetidos
//POS: Imprime por pantalla fila y columna de 'elem'
void imprimirFilCol (int **mat, int filas, int columnas, int elem);
```

3. Implementar una función *obtenerOrdenados* que, dada una matriz de caracteres, retorne un vector de enteros con las posiciones de las filas que tienen los caracteres ordenados según la tabla ASCII.
   Ejemplo:
   [[‘a’ , ‘b’ , ‘c’ , ‘a’],
   [‘a’ , ‘b’ , ‘c’ , ‘q’],
   [‘a’ , ‘b’ , ‘a’ , b’],
   [‘a’ , ‘a’, ‘b’ , ‘c’]]

   Debería retornar:
   [1, 3]

```
//PRE: Recibe una matriz de caracteres
//POS: Retorna un vector con las posiciones de las filas
//     cuyos caracteres está ordenados según la tablas ASCII
int* borrarAdjuntos (char ** mat, int filas, int columnas);
```

4. Implementar una función *borrarAdjuntos* que, dada una matriz de enteros distintos de 0 y dos enteros *f* y *c* (uno representando la posición de la fila y el otro la posición de la columna), que remplace por 0 todos los valores adjuntos a la posición *[f][c]* y que sean igual a *mat[f][c].* Debe retornar una **nueva matriz.**

   Ejemplo:
   f = 0
   c = 1

   [[1 , **2** , 3 , 4 , 5 , 6]
   [3 , **2** , 6 ,4, 2 , 4]
   [**2**, **2** , **2** , 5 , 6 , 2]]

   Debería retornar:

   [[1 , **0** , 3 , 4 , 5 , 6]
   [3 , **0** , 6 ,4, 2 , 4]
   [**0**, **0** , **0** , 5 , 6 , 2]]

```
//PRE: Recibe una matriz de enteros distintos de 0.
//     f >= 0
//     c >= 0
//POS: Retorna una nueva matriz con adjuntos a mat[f][c] y
//     valores igual a mat[f][c] remplazados por 0
int** borrarAdjuntos (int ** mat, int filas, int columnas, int f, int c);
```