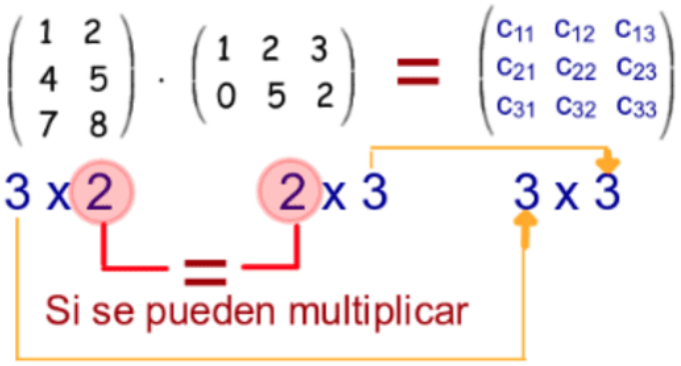
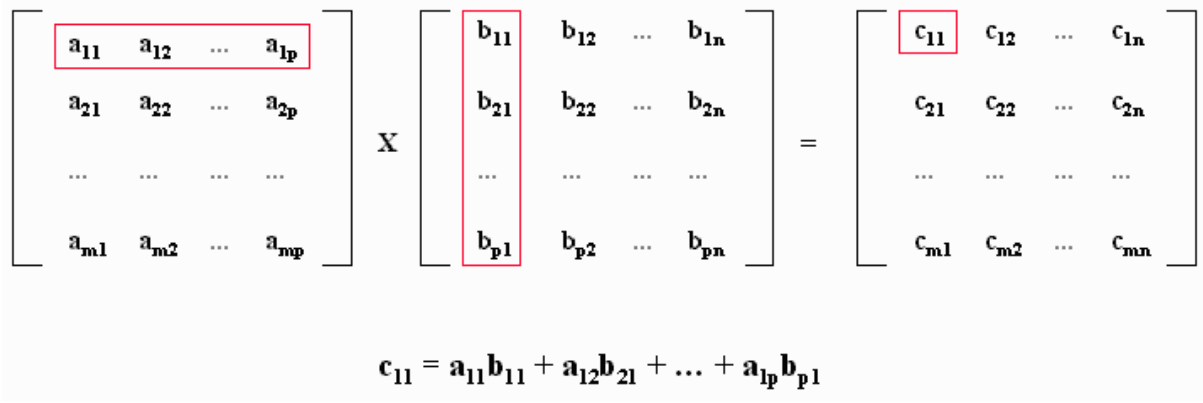
Ejercicio 1.
En la clase Program, crea los métodos correspondientes a las siguientes operaciones con matrices, si no se puede realizar la operación se lanza una excepción:

- Suma de dos matrices: se suma cada elemento en la misma posición de las dos matrices. Dos matrices se pueden sumar si tienen el mismo número de filas y columnas.
- Traspuesta de una matriz: las filas pasan a ser columnas.
- Multiplicación de dos matrices: Para que dos matrices se puedan multiplicar debe ocurrir que el número de filas de la segunda matriz sea igual que el de columnas de la primera matriz.





$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + \ldots + a_{1p}b_{p1}$$

Ejercicio 2.
Crea una lista de enteros, añádele 100 números enteros generados aleatoriamente entre 1 y 10 ambos inclusive. Luego, muestra por pantalla si contiene todos los números entre el 1 y el 10 ambos incluidos. Posteriormente, elimina todas las ocurrencias de números pares. Finalmente muestra por pantalla el contenido de la lista.

Ejercicio 3.
Crea una cola de enteros, añádele 100 números enteros generados aleatoriamente entre 1 y 10 ambos inclusive. Saca de la cola números hasta que haya un número par al principio de la cola. Muestra por pantalla el valor que hay al principio de la cola.

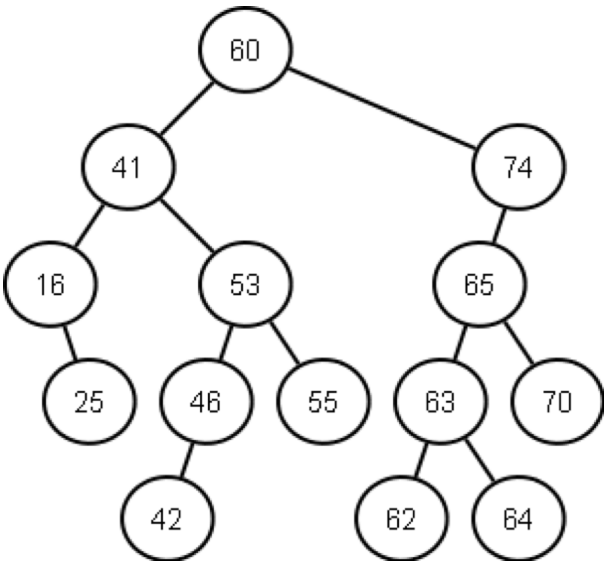
Ejercicio 4.
Crea una pila de enteros, añádele 100 números enteros generados aleatoriamente entre 1 y 10 ambos inclusive. Saca de la pila números hasta que haya un número par encima de la pila. Muestra por pantalla el valor que hay encima de la pila.

Ejercicio 5.
Crea un diccionario en el cual las claves van a ser palabras y los valores sus definiciones. Crea el diccionario con 5 términos. Luego, pide al usuario una palabra, si esta está en el diccionario se muestra su definición, en caso contrario se notifica al usuario.

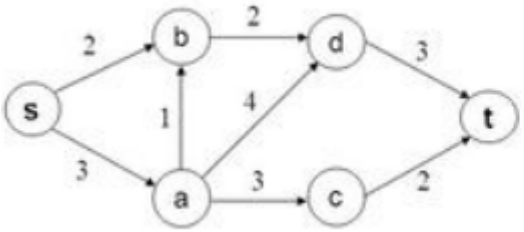
Ejercicio 6.
Si U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} es el conjunto universal y A = {1, 4, 7, 10},
B = {1, 2, 3, 4, 5}, C = {2, 4, 6, 8}, calcula los siguientes conjuntos:
(Notación complementario ', por ejemplo, A' sería el complementario de A)
a) $A \cup B$
b) $A - B$
c) A'
d) U'
e) $B \cap U$
f) $B' \cap (C - A)$
g) $A \cap B' \Delta C$
h) $B \cap C$
i) $A \cup \varnothing$
j) $A \cap (B \cup C)$
k) $(A \cap B) \cup C$
l) $(A \cap B) - C$
m) $(A \cup B) - (C - B)$

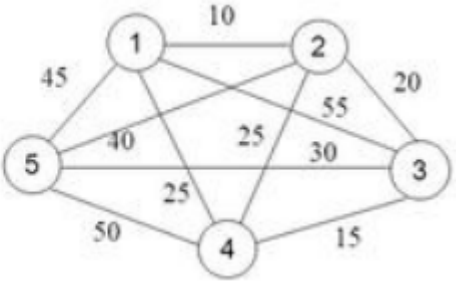
Ejercicio 7.
Teniendo en cuenta el árbol que se muestra a continuación:

- Indica el nodo raíz.
- Indica una de sus hojas.
- Traza un camino para llegar desde el 41 hasta el 42.
- Calcula el nivel del nodo 70.
- Calcula la altura del árbol.
- Calcula la profundidad del nodo 25.
- Haz el recorrido en preorden, enorden y posorden.

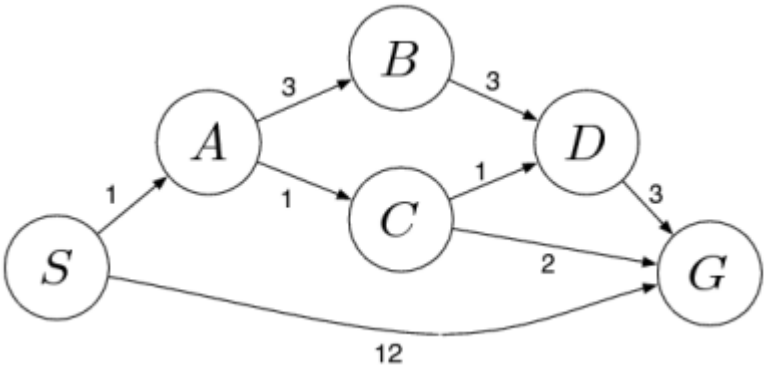


Ejercicio 8.
Pasa a tabla cada uno de los siguientes grafos:





Ejercicio 9.
Calcula el camino con menos coste para ir desde S hasta G.



Ejercicio 10.
Implementa desde cero una Cola (FIFO) utilizando como base un array. Para ello crea una clase genérica llamada Cola que tendrá las siguientes características, todos los métodos son públicos:

- Un array de T como atributo donde se irá guardando los elementos de la colección.
- Un número entero como atributo para llevar la cuenta de cuántos elementos tiene la colección.

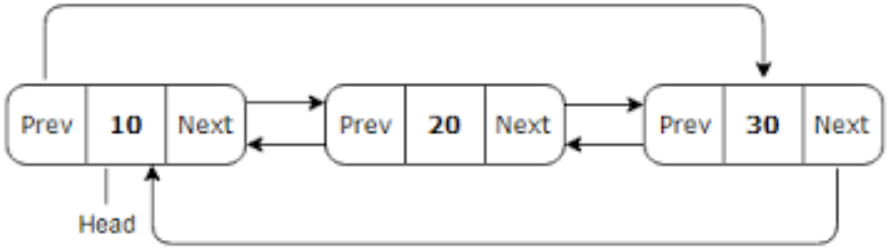
- Count: propiedad de lectura pública que devuelve la cantidad de elementos que hay en la colección.
- Un constructor por defecto en el que se crea el array con un tamaño inicial de 10.
- Un constructor que recibe una colección por parámetro (IEnumerable<T>) y añade cada uno de los elementos a la colección.
- IsEmpty(): devuelve un booleano indicando si la colección está vacía (true) o no (false).
- Enqueue(T): introduce el elemento pasado por parámetro al final de la cola, si no cabe nada más en el array, se duplicará la capacidad del mismo.
- Dequeue(): elimina el primer elemento de la cola y lo devuelve. Para eliminar el primer elemento, desplaza todos los elementos del array una posición a la izquierda. Si la colección está vacía lanza la excepción InvalidOperationException.
- Peek(): devuelve el primer elemento de la cola sin eliminarlo. Si la colección está vacía lanza la excepción InvalidOperationException.
- Contains(T): devuelve verdadero si la colección contiene un elemento igual al que se le pasa por parámetro, en caso contrario devuelve falso. Ten en cuenta que la colección puede contener valores nulos.
- Clear(): limpia la colección eliminando todos los elementos.
- Implementa la interfaz IEnumerable<T> y su correspondiente iterador respetando el orden FIFO.

Ejercicio 11.
Implementa desde cero una Pila (LIFO) utilizando como base la lógica de la lista enlazada. Para ello crea una clase genérica llamada Pila que tendrá las siguientes características, todos los métodos son públicos:

- Una clase privada llamada Nodo que servirá para guardar el valor del elemento y la referencia del nodo siguiente.
- Un número entero como atributo para llevar la cuenta de cuántos elementos tiene la colección.
- Un Nodo (correspondiente a la implementación de la lista enlazada) como atributo que apuntará al último elemento insertado en la pila (la cima de la pila).
- Count: propiedad de lectura pública que devuelve la cantidad de elementos que hay en la colección.
- Un constructor por defecto que crea la pila vacía.
- Un constructor que recibe una colección por parámetro (IEnumerable<T>) y añade cada uno de los elementos a la colección.
- IsEmpty(): devuelve un booleano indicando si la colección está vacía (true) o no (false).
- Push(T): introduce el elemento pasado por parámetro encima de la pila.
- Pop(): elimina el elemento que está encima de la pila y lo devuelve. Si la colección está vacía lanza la excepción InvalidOperationException.
- Peek(): devuelve el elemento que está encima de la pila sin eliminarlo. Si la colección está vacía lanza la excepción InvalidOperationException.
- Contains(T): devuelve verdadero si la colección contiene un elemento igual al que se le pasa por parámetro, en caso contrario devuelve falso. Ten en cuenta que la colección puede contener valores nulos.
- Clear(): limpia la colección eliminando todos los elementos.

- Implementa la interfaz IEnumerable<T> y su correspondiente iterador respetando el orden LIFO.

Ejercicio 12.
Implementa desde cero una Lista doblemente enlazada (cada nodo sabe quién es su anterior y posterior). Si solo hay un nodo, que es la cabecera, su anterior y siguiente será ese mismo nodo.



Para ello crea una clase genérica llamada ListaDobleEnlazada que tendrá las siguientes característica, todos los métodos son públicos:

- Una clase privada llamada Nodo que servirá para guardar el valor del elemento y las referencias del nodo anterior y siguiente.
- Un Nodo como atributo que apuntará al primer elemento de la lista.
- Count: propiedad de lectura pública que devuelve la cantidad de elementos que hay en la colección.
- Un constructor por defecto en el que crea la colección completamente vacía.
- Un constructor que recibe una colección por parámetro (IEnumerable<T>) y añade cada uno de los elementos a la colección.
- IsEmpty(): devuelve un booleano indicando si la colección está vacía (true) o no (false).
- Add(T): introduce el elemento pasado por parámetro al final de la lista.
- Remove(T): elimina el primer nodo con el mismo valor que el pasado por parámetro.
- Contains(T): devuelve verdadero si la colección contiene un elemento igual al que se le pasa por parámetro, en caso contrario devuelve falso.
- Clear(): limpia la colección eliminando todos los elementos.
- Implementa la interfaz IEnumerable<T> y su correspondiente iterador respetando el orden FIFO.

Ejercicio 13.
Implementa desde cero un Conjunto utilizando como base la implementación de lista que viene ya hecha con el lenguaje (System.Colletions.Generic.List). Para ello crea una clase genérica llamada Conjunto que tendrá las siguientes características, todos los métodos son públicos:

- Una List como atributo, en ella se irán guardando los elementos.
- Count: propiedad de lectura pública que devuelve la cantidad de elementos que hay en la colección.
- Un constructor por defecto en el que crea la colección completamente vacía.
- Un constructor que recibe una colección por parámetro (IEnumerable<T>) y añade cada uno de los elementos a la colección.

- IsEmpty(): devuelve un booleano indicando si la colección está vacía (true) o no (false).
- Add(T): añade el elemento pasado por parámetro a la colección (se recuerda que en un conjunto no puede haber elementos repetidos).
- Remove(T): elimina el elemento pasado por parámetro de la colección.
- Contains(T): devuelve verdadero si la colección contiene un elemento igual al que se le pasa por parámetro, en caso contrario devuelve falso.
- Clear(): limpia la colección eliminando todos los elementos.
- Implementa la interfaz IEnumerable<T> y en el método GetEnumerator que devuelva el iterador del atributo que es una lista.

Ejercicio 14.
Crea una lista o array con 30 números del 1 al 20 (ambos incluidos) elegidos aleatoriamente y realiza las siguientes consultas y operaciones, mostrando por pantalla el resultado de las mismas:

1. El último número de la colección.
2. Los números pares.
3. Los números que tengan menos de 2 cifras.
4. Los números que no terminen en 5.
5. El número más alto y más bajo.
6. La media aritmética.
7. La media cuadrática (la raíz cuadrada de la media aritmética de los cuadrados de los números).
8. Los números ordenados de mayor a menor sin repeticiones.
9. La mitad de cada número sin decimales.
10. El logaritmo en base 2 de cada número con 2 decimales.
11. La moda (el número que más veces se repita).
12. Los números mayores a la mitad de la media aritmética.
13. El primer número mayor que 10.
14. Los números que están entre la posición 10 y 20.
15. Comprobar si hay algún número mayor que 10.
16. Comprobar que todos los números están en el intervalo [1, 20].
17. Comprobar que el número 5 está en la colección.
18. Los números menos el conjunto { 1, 3, 5, 7, 11, 13, 17, 19, 23, 29 }.

Reto 99% imposible.
Hacer el ejercicio del diamante en una sola sentencia usando LINQ. El programa pedirá un número al usuario (solo ReadLine, no se muestra por pantalla nada) y mostrará el diamante con la altura correspondiente. Crea el proyecto sin clase Program ni Main (como hacíamos al principio del curso). Solo podrá haber un ; y nada de declarar métodos en todo el código. Se recuerda que las listas tienen un método ForEach al estilo LINQ.
Este tutorial puede ser de ayuda: https://www.youtube.com/watch?v=hvL1339luv0