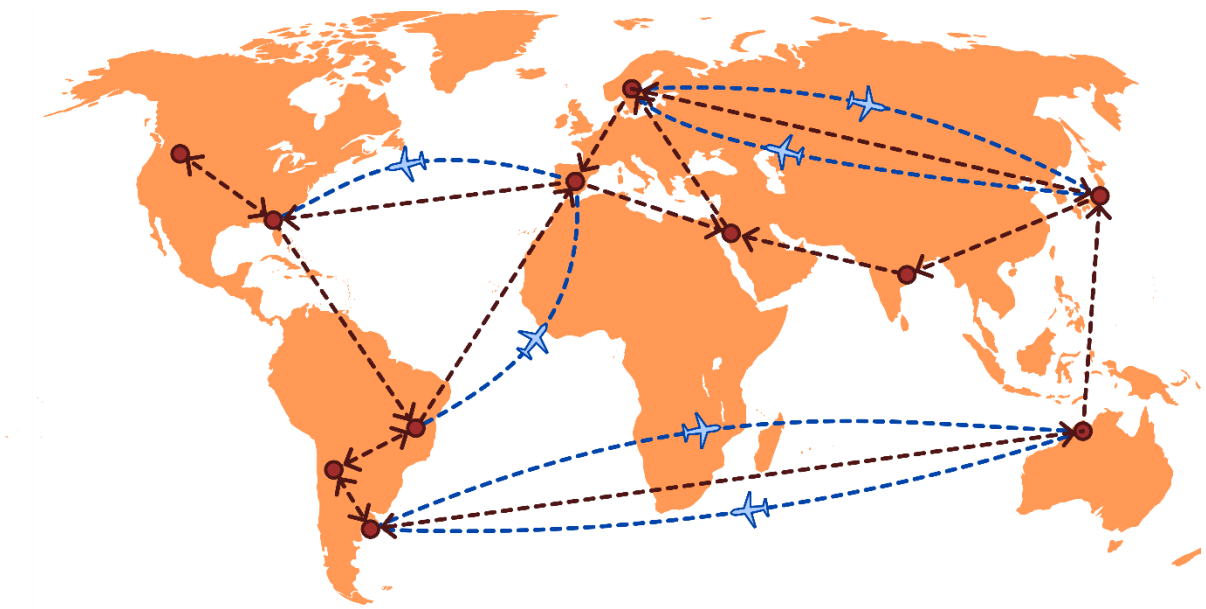
EVALUACION	Obligatorio	**GRUPO**	Todos	**FECHA**	Marzo 2025 V1
MATERIA	Algoritmos y Estructuras de Datos 2				
CARRERA	Analista Programador – Analista en Tecnologías de la Información				
CONDICIONES	**- Puntaje máximo:** 35 puntos **- Puntaje mínimo:** - **- Fecha de entrega:** 05/06/25 hasta las 21:00 horas en gestion.ort.edu.uy (máx. 40Mb en formato zip o rar) **Uso de material de apoyo y/o consulta** Inteligencia Artificial Generativa - Seguir las pautas de los docentes: Se deben seguir las instrucciones específicas de los docentes sobre cómo utilizar la IA en cada curso. - Citar correctamente las fuentes y usos de IA: Siempre que se utilice una herramienta de IA para generar contenido, se debe citar adecuadamente la fuente y la forma en que se utilizó. - Verificar el contenido generado por la IA: No todo el contenido generado por la IA es correcto o preciso. Es esencial que los estudiantes verifiquen la información antes de usarla. - Ser responsables con el uso de la IA: Conocer los riesgos y desafíos, como la creación de "alucinaciones", los peligros para la privacidad, las cuestiones de propiedad intelectual, los sesgos inherentes y la producción de contenido falso. - En caso de existir dudas sobre la autoría, plagio o uso no atribuido de IAG, el docente tendrá la opción de convocar al equipo de obligatorio a una defensa específica e individual sobre el tema. **IMPORTANTE:** 1) Inscribirse 2) Formar grupos de hasta 2 personas del mismo dictado 3) Subir el trabajo a Gestión antes de la hora indicada (ver hoja al final del documento: "RECORDATORIO") Aquellos de ustedes que presenten alguna dificultad con su inscripción o tengan inconvenientes técnicos, por favor contactarse con el Coordinador o Coordinación adjunta **antes de las 20:00hs.** del día de la entrega, a través de los mails alamon@ort.edu.uy y gervaz@ort.edu.uy o telefónicamente al 29021505 – int. 1156 u 1138				

IMPORTANTE: Verificar que se haya recibido el correo electrónico corroborando la correcta entrega en gestión, y verificar que haya quedado entregada la última versión del obligatorio de forma correcta.

No se considerarán entregas fuera de fecha.

Introducción

Se desea implementar un programa para resolver la alta demanda de las operaciones de una agencia de viajes cumpliendo con las cotas de tiempo requeridas para cada operación. Se necesita poder ingresar y consultar información sobre los viajeros, cuidades, conexiones y vuelos.



Todas las operaciones deberán devolver una instancia de la clase Retorno. Dicha clase contiene:

- Un resultado, que especifica si la operación se pudo realizar correctamente (OK), o si ocurrió algún error (según el número de error).
- Un valor entero, para las operaciones que retornen un número entero.
- Un valor String, para las operaciones que retornen un String, o un valor más complejo, por ejemplo, un listado, el cual será formateado según lo indicado en los ejemplos.

```
public class Retorno {
	public enum Resultado {OK, ERROR_1, ERROR_2, ERROR_3, ERROR_4, ERROR_5,
ERROR_6, ERROR_7, ERROR_8, NO_IMPLEMENTADA};
	public int valorEntero;
	public String valorString;
	public Resultado resultado;
}
```

Se provee: una interfaz llamada **Sistema**, la cual no podrá ser modificada en ningún sentido, y una clase **ImplementacionSistema** que la implementa, donde su equipo deberá completar la implementación de las operaciones solicitadas.

Consideraciones:

- La clase **ImplementacionSistema** NO PODRÁ SER UN SINGLETON. Debe ser una clase **instanciable**.
- Pueden definirse tipos de datos (clases) auxiliares.
- Se provee un proyecto base que debe ser utilizado, utilizar el proyecto es obligatorio.

Funcionalidades

01.- Inicializar Sistema

```
Retorno inicializarSistema (int maxCiudades);
```

Descripción: Inicializa las estructuras necesarias para representar el sistema especificado, capaz de registrar como máximo la cantidad `maxCiudades` de ciudades diferentes en el sistema.

Restricción de eficiencia: no tiene.

Retornos posibles	
OK	El sistema pudo ser inicializado exitosamente.
ERROR	1. Si `maxCiudades` es menor o igual a 4.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

02.- Registrar viajero

```
Retorno registrarViajero(String cedula, String nombre, String correo, int edad, Categoria categoria);
```

Descripción: Registra el viajero con sus datos, la cédula y el correo son únicos.
Restricción de eficiencia: Esta operación deberá realizarse en orden O(log n) promedio, siendo n la cantidad total de viajeros.

Retornos posibles	
OK	El viajero fue registrado exitosamente.
ERROR	1. Si alguno de los parámetros es vacío o null. 2. Si la cédula no es una cédula con formato válido. 3. Si el correo no tiene el formato válido. 4. Si la edad no está en el rango válido [0 ~ 139] límites incluidos. 5. Si ya existe un viajero registrado con esa cédula. 6. Si ya existe un viajero registrado con ese correo.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Restricción: (Investigación) Se requiere el uso de expresiones regulares para lograr validar el formato de la cédula y el correo. Para la cédula el formato a validar es: N.NNN.NNN-N o NNN.NNN-N (No es necesario realizar la validación con el dígito verificador de la cédula)

Las categorías posibles son:

- Platino
- Frecuente
- Estándar

Ejemplos:

Resultado	Operación
OK	registrarViajero("1.914.689-5","Guillermo","guille@ort.edu.uy",35, Categoria.*ESTANDAR*)
ERROR_1	registrarViajero("", "Guillermo", "guille@ort.edu.uy", 35, Categoria.*ESTANDAR*);
ERROR_2	registrarViajero("1.91.4.689-5", "Guillermo", "guille@ort.edu.uy", 35, Categoria.*ESTANDAR*);
ERROR_3	registrarViajero("1.914.689-5", "Guillermo", "guilleort.edu.uy", 35, Categoria.*ESTANDAR*);
ERROR_4	registrarViajero("1.914.689-5","Guillermo","guille@gmail.com",-5, Categoria.*ESTANDAR*)

03.- Buscar Viajero por cédula

```
Retorno buscarViajeroPorCedula(String cedula);
```

Descripción: Retorna en valorString los datos del viajero con el formato "cedula;nombre;correo;edad;categoria". Además, en el campo valorEntero del objeto retorno deberá devolver la cantidad de elementos que recorrió durante la búsqueda en la estructura utilizada.

Restricción de eficiencia: Esta operación deberá realizarse en orden O(log n) promedio, siendo n la cantidad total de viajeros.

Retornos posibles	
OK	Si el viajero se encontró. Retorna en valorString los datos del viajero. Retorna en valorEntero la cantidad de elementos recorridos durante la búsqueda.
ERROR	1. Si la cédula es vacía o null. 2. Si la cédula no tiene formato válido. 3. Si no existe un viajero registrado con esa cédula.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valorString: cedula;nombre;correo;edad;categoría.

Por ejemplo, valorString del retorno en una consulta válida:

```
1.914.689-5;Guillermo;guille@ort.edu.uy,35,Estándar
```

04.- Buscar Viajero por correo

```
Retorno buscarViajeroPorCorreo(String correo);
```

Descripción: Retorna en valorString los datos del viajero con el formato "cedula;nombre;correo;edad;categoría". Además, en el campo valorEntero del objeto retorno, deberá devolver la cantidad de elementos que recorrió durante la búsqueda en la estructura utilizada.

Restricción de eficiencia: Esta operación deberá realizarse en orden O(log n) promedio, siendo n la cantidad total de viajeros.

Retornos posibles	
OK	Si el usuario se encontró. Retorna en valorString los datos del viajero. Retorna en valorEntero la cantidad de elementos recorridos durante la búsqueda.
ERROR	1. Si el correo es vacío o null. 2. Si el correo no tiene formato válido. 3. Si no existe un viajero con ese correo.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valorString: cedula;nombre;correo;edad;categoría.

Por ejemplo, valorString del retorno en una consulta válida:

```
1.914.689-5;Guillermo;guille@ort.edu.uy,35,Estándar
```

05.- Listar viajeros por cedula ascendente

```
Retorno listarViajerosPorCedulaAscendente();
```

Descripción: Retorna en valorString los datos de todos los viajeros registrados, ordenados por cédula en forma creciente separados por un |.
Restricción de eficiencia: Esta operación deberá realizarse en orden O(n), siendo n la cantidad total de viajeros.

Retornos posibles	
OK	Retornando el listado de viajeros en valorString.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:
cédula1;nombre1;correo1;edad1;categoría1|cédula2;nombre2;correo2;edad2;categoria2

Nota: Se debe cumplir que cédula1 es numéricamente menor a cédula2.

Por ejemplo:
```
433.018-4;Ana;ana@ort.edu.uy;82;Estándar|1.914.689-5;Guille;guille@ort.edu.uy,35,Estándar
```

06.- Listar viajeros por cedula descendente

```
Retorno listarViajerosPorCedulaDescendente();
```

Descripción: Retorna en valorString los datos de todos los viajeros registrados, ordenados por cédula en forma decreciente.
Restricción de eficiencia: Esta operación deberá realizarse en orden O(n), siendo n la cantidad total de viajeros.

Retornos posibles	
OK	Retornando el listado de viajeros en valorString.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:
cédula1;nombre1;correo1;edad1;categoría1|cédula2;nombre2;correo2;edad2;categoria2

Nota: Se debe cumplir que cédula1 es numéricamente mayor a cédula2.

Por ejemplo:
```
1.914.689-5;Guille;guille@ort.edu.uy,35,Estándar|433.018-4;Ana;ana@ort.edu.uy;82;Estándar
```

07.- Listar viajeros por correo ascendente

```
Retorno listarViajerosPorCorreoAscendente();
```

Descripción: Retorna en valorString los datos de todos los viajeros registrados, ordenados por correo en forma creciente.
Restricción de eficiencia: Esta operación deberá realizarse en orden O(n), siendo n la cantidad total de viajeros.

Retornos posibles	
OK	Retornando el listado de viajeros en valorString.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:
cédula1;nombre1;correo1;edad1;categoría1|cédula2;nombre2;correo2;edad2;categoria2

Nota: Se debe cumplir que correo1 es lexicográficamente menor a correo2.

Por ejemplo:
```
433.018-4;Ana;ana@ort.edu.uy;82;Estándar|1.914.689-5;Guille;guille@ort.edu.uy,35,Estándar
```

08.- Listar viajeros por categoría

```
Retorno listarViajerosPorCategoría(Categoria unaCategoria);
```

Descripción: Retorna en valorString los datos de todos los viajeros registrados con esa categoría, ordenados de forma creciente por cédula.
Restricción de eficiencia: Esta operación deberá realizarse en orden O(k), siendo k la cantidad de viajeros con dicha categoría.

Retornos posibles	
OK	Se pudo listar los viajeros que pertenecen a esa categoría correctamente.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:
cédula1;nombre1;correo1;edad1;categoría|cédula2;nombre2;correo2;edad2;categoria

Nota: Se debe cumplir que cédula1 es numéricamente menor a cédula2.

Por ejemplo:
```
433.018-4;Ana;ana@ort.edu.uy;82;Estándar|1.914.689-5;Guille;guille@ort.edu.uy,35,Estándar
```

09.- Listar viajeros ascendente de un rango de edades

```
Retorno listarViajerosDeUnRangoAscendente(int rango);
```

Descripción: Retorna en valorString los datos de todos los viajeros registrados cuya edad esté en el rango indicado, ordenados por cédula en forma creciente. Los rangos han sido previamente definidos y se detallan más adelante.
Restricción de eficiencia: Esta operación deberá realizarse en orden O(k), siendo k la cantidad de viajeros con edad en el rango indicado.

Retornos posibles	
OK	Retornando el listado de viajeros en valorString.
ERROR	1. Si rango es menor a 0. 2. Si rango es mayor a 13.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:
cédula1;nombre1;correo1;edad1;categoría|cédula2;nombre2;correo2;edad2;categoria

Nota: Se debe cumplir que cédula1 es numéricamente menor a cédula2.

Por ejemplo, para rango=3:
`1.104.441-1;Ana;ana@ort.edu.uy;32;Estándar|1.914.689-5;Guille;guille@ort.edu.uy,38,Estándar`

Los rangos están predefinidos:

- Rango = 0: De 0 a 9 años
- Rango = 1: De 10 a 19 años
- Rango = 2: De 20 a 29 años
- Rango = 3: De 30 a 39 años
- Rango = 4: De 40 a 49 años
- Rango = 5: De 50 a 59 años
- Rango = 6: De 60 a 69 años
- Rango = 7: De 70 a 79 años
- Rango = 8: De 80 a 89 años
- Rango = 9: De 90 a 99 años
- Rango = 10: De 100 a 109 años
- Rango = 11: De 110 a 119 años
- Rango = 12: De 120 a 129 años
- Rango = 13: De 130 a 139 años

10.- Registrar ciudad

```
Retorno registrarCiudad(String codigo, String nombre);
```

Descripción: Registra una ciudad en el sistema con el código y nombre indicado. El código es el identificador único, el código y nombre no pueden ser vacíos ni null. Esta operación no tiene restricciones de eficiencia.

Retornos posibles	
OK	Si la ciudad fue registrada exitosamente.
ERROR	1. Si en el sistema ya hay registrados *maxCuidades.* 2. Si código o nombre son vacíos o null. 3. Si ya existe una ciudad con ese código.
NO_IMPLEMENTADA	Cuando aún no se implementó.

11.- Registrar Conexión

```
Retorno registrarConexion(String codigoCiudadOrigen, String
codigoCiudadDestino);
```

Descripción: Registra una conexión en el sistema desde la ciudad `codigoCiudadOrigen` a la cuidad `codigoCiudadDestino`. Esta operación no tiene restricciones de eficiencia.

Retornos posibles	
OK	Si la conexión fue registrada exitosamente.
ERROR	1. Si alguno de los parámetros es vacío o null. 2. Si no existe la ciudad de origen. 3. Si no existe la ciudad de destino. 4. Si ya existe una conexión entre el origen y el destino.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Nota: Se considera que las conexiones no son navegables en ambos sentidos. O sea que, si existe una conexión entre las ciudades A y B, puede no existir de B a A.

12.- Registrar vuelo

```
Retorno registrarVuelo(String codigoCiudadOrigen, String
codigoCiudadDestino,String codigoDeVuelo, double combustible, double minutos,
double costoEnDolares, TipoVuelo tipoDeVuelo);
```

Descripción: Registra un nuevo vuelo en el sistema, debe existir la conexión entre las ciudades de origen y el destino. Puede haber varios vuelos entre las ciudades de origen y el destino. El identificador de un vuelo "`codigoDeVuelo`" es único dentro de la conexión entre la ciudad de origen y destino, ósea, en una conexión no puede haber dos vuelos con el mismo código. Esta operación no tiene restricciones de eficiencia.

Retornos posibles	
OK	Si el vuelo se registró exitosamente.
ERROR	1. Si alguno de los parámetros double es menor o igual a 0. 2. Si alguno de los parámetros String es vacío o null. 3. Si no existe la ciudad de origen. 4. Si no existe la ciudad de destino. 5. Si no existe una conexión entre origen y destino. 6. Si ya existe un vuelo con ese código en esa conexión.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Los tipos de vuelo posibles son:

- Comercial
- Privado

13.- Actualizar vuelo

```
Retorno actualizarVuelo(String codigoCiudadOrigen, String codigoCiudadDestino,
String codigoDeVuelo, double combustible, double minutos, double costoEnDolares,
TipoVuelo tipoDeVuelo);
```

Descripción: Actualiza un vuelo previamente ingresado al sistema, debe existir un vuelo con ese código en la conexión entre ciudad origen y ciudad. Los valores que se pueden actualizar son: combustible, minutos, costoEnDolares y tipoDeVuelo. Esta operación no tiene restricciones de eficiencia.

Retornos posibles	
OK	Si el vuelo se actualizó exitosamente.
ERROR	1. Si alguno de los parámetros double es menor o igual a 0. 2. Si alguno de los parámetros String es vacío o null. 3. Si no existe la ciudad origen. 4. Si no existe la ciudad destino. 5. Si no existe una conexión entre origen y destino. 6. Si no existe un vuelo con ese código en esa conexión.
NO_IMPLEMENTADA	Cuando aún no se implementó.

14.- Ciudades por cantidad de escalas

```
Retorno listadoCiudadesCantDeEscalas(String codigoCiudadOrigen, int cantidad);
```

Descripción: Dada una ciudad de origen se debe retornar en el valorString los datos de las ciudades (ordenadas por código creciente) a las que se pueda llegar realizando hasta la cantidad de escalas indicada por parámetro. Esta operación no tiene restricciones de eficiencia.

Retornos posibles	
OK	Retorna en valorString los datos de las ciudades a las que se pueda llegar con hasta “cantidad” de escalas ordenadas de forma creciente por código.
ERROR	1. Si la cantidad es menor que cero. 2. Si el código es vacío o null. 3. Si la ciudad no está registrada en el sistema.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:

```
codigoCiudad1;nombreCiudad1|codigoCiudad2;nombreCiudad2
```

Nota: Se debe cumplir que `codigoCiudad1` es lexicográficamente menor a `codigoCiudad2`.

15. Viaje de costo mínimo en minutos

```
Retorno viajeCostoMinimoMinutos(String codigoCiudadOrigen, String
codigoCiudadDestino, TipoVueloPermitido tipoVueloPermitido);
```

Descripción: Retorna el camino más corto en minutos que podría realizar un viajero para ir de la ciudad origen a la ciudad destino. Esta operación no tiene restricciones de eficiencia.

Retornos posibles	
OK	Si el camino pudo ser calculado exitosamente. Retorna en valorEntero la cantidad total minutos del camino. Retorna en valorString el camino desde la ciudad origen a la ciudad destino incluidas.
ERROR	1. Si alguno de los parámetros es vacío o null. 2. Si no existe la ciudad origen. 3. Si no existe la ciudad destino. 4. Si no hay **camino** entre el origen y el destino.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Los tipos de vuelos permitidos son: Comercial, Privado o Ambos.

Formato de retorno del valor String:

```
codigoCiudadOrigen;nombreCiudadOrigen|codigoCiudad1;nombreCiudad1|codigoCiudad2
;nombreCiudad2|codigoCiudadDestino;nombreCiudadDestino
```

16. Viaje de costo mínimo en dólares

```
Retorno viajeCostoMinimoDolares(String codigoCiudadOrigen, String
codigoCiudadDestino, TipoVueloPermitido tipoVueloPermitido);
```

Descripción: Retorna el camino más corto en dólares (monto acumulado más barato) que podría realizar un viajero para ir de la ciudad origen a la ciudad destino. Esta operación no tiene restricciones de eficiencia.

Retornos posibles	
OK	Si el camino pudo ser calculado exitosamente. Retorna en valorEntero el costo total en dólares del camino. Retorna en valorString el camino desde la ciudad origen a la ciudad destino incluidas.
ERROR	1. Si alguno de los parámetros es vacío o null. 2. Si no existe la ciudad origen, 3. Si no existe la ciudad destino. 4. Si no hay **camino** entre el origen y el destino.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Los tipos de vuelos permitidos son: Comercial, Privado o Ambos.

Formato de retorno del valor String:

```
codigoCiudadOrigen;nombreCiudadOrigen|codigoCiudad1;nombreCiudad1|codigoCiudad2
;nombreCiudad2|codigoCiudadDestino;nombreCiudadDestino
```

Información importante

Se deberán **respetar los formatos de retorno** dados para las operaciones que devuelven datos.

Está **terminantemente prohibido** el uso de clases de Java tales como **ArrayList, HashMap, el uso de estas clases implica la pérdida de todos los puntos de la operación en la que se utilizaron**.

Ninguna de las operaciones deben imprimir **nada** en consola.

El sistema no debe requerir ningún tipo de interacción con el usuario por consola.

Es obligación del estudiante mantenerse al tanto de las aclaraciones que se realicen en clase o a través del foro de aulas.

Es necesaria una selección adecuada de las estructuras para modelar el problema y la eficiencia en cada una de las operaciones. Deberá aplicar la metodología vista en el curso, resolver el obligatorio (incluso pasando las pruebas funcionales) sin cumplir los órdenes puede implicar la pérdida del total de los puntos de la operación.

Información importante

El proyecto será implementado en lenguaje JAVA en el ide IntelliJ IDEA, ya se encuentra disponible en el sitio de la materia en aulas.ort.edu.uy. La interfaz Sistema no se puede modificar. El uso del proyecto provisto junto a la letra es obligatorio.

El proyecto entregado debe compilar y ejecutar correctamente en IntelliJ IDEA. De no ser así implica la pérdida de todos los puntos del obligatorio. Si no se logra implementar una operación se debe dejar en un estado consistente o como “No implementada”.

No se contestarán dudas sobre el obligatorio en las 48 horas previas a la entrega.

El equipo debe entregar un conjunto de pruebas unitarias para evidenciar una correcta estrategia de testing aplicada al proyecto.

Una semana antes de la entrega la cátedra dejará disponible un conjunto de pruebas (no exhaustivas) para todas las operaciones para que puedan ajustar las desviaciones con las pruebas propias.

El ABB utilizado se debe implementar de forma genérica, como también Pila, Lista y Cola si se utilizan.

El Grafo utilizado no es necesario implementarlo de forma genérica.

Ninguno de los listados (que se cargan en el valorString) termina en |, ósea al agregar el último elemento no se debe agregar el |.

Defensa: la defensa del obligatorio será en fecha y formato a confirmar por el docente. Se publicará en aulas la fecha y horario específico con antelación. La no asistencia a la defensa implica la pérdida de todos los puntos.

Considere que superar las pruebas no garantiza que la operación se haya resuelto correctamente.

Entrega (Se debe subir un único zip o rar con):

- Proyecto con las operaciones implementadas respetando las buenas prácticas y estándares vistos en el curso.
    - La implementación debe respetar los órdenes solicitados.
    - Incluir en la parte superior de la clase ImplementacionSistema los nombres y números de estudiante del equipo.
- Documentación: Entregar un documento PDF con una carátula con información de la materia, evaluación y del equipo. En el cuerpo del documento agregar una descripción de las estructuras utilizadas y una justificación del cumplimiento de los órdenes solicitados.

RECORDATORIO: IMPORTANTE PARA LA ENTREGA

- **Obligatorios**

La entrega de los obligatorios será en formato digital online, a excepción de algunas materias que se entregarán en Bedelía y en ese caso recibirá información específica en el dictado de la misma.

Los principales aspectos a destacar sobre la **entrega online de obligatorios** son:

1. Ingresá al sistema de Gestión.
2. En el menú, seleccioná el ítem "Evaluaciones" y la instancia de evaluación correspondiente, que figura bajo el título "Inscripto".
3. Para iniciar la entrega hacé clic en el ícono:
4. Ingresá el número de estudiante de cada uno de los integrantes y hacé clic en "Agregar". El sistema confirmará que los integrantes estén inscriptos al obligatorio y, de ser así, mostrará el nombre y la fotografía de cada uno de ellos. Una vez agregados todos los integrantes, hacé clic en "Crear equipo".

   Cualquier integrante podrá:

   • Modificar la integración del equipo.

   • Subir el archivo de la entrega.

5. Seleccioná el archivo que deseás entregar. Verificá el nombre del archivo que aparecerá en la pantalla y hacé clic en "Subir" para iniciar la entrega. Cada equipo (hasta 2 estudiantes) debe entregar **un único archivo en formato zip o rar** (los documentos de texto deben ser pdf, y deben ir dentro del zip o rar). El archivo a subir debe tener **un tamaño máximo de 40mb**

   Cuando el archivo quede subido, se mostrará el nombre generado por el sistema (1), el tamaño y la fecha en que fue subido.

6. El sistema enviará un e-mail a todos los integrantes del equipo informando los detalles del archivo entregado y confirmando que la entrega fue realizada correctamente.
7. Podés cerrar la pestaña de entrega y continuar utilizando Gestión o salir del sistema.
8. La **hora tope para subir el archivo será las 21:00** del día fijado para la entrega.
9. La entrega se podrá realizar desde cualquier lugar (ej. hogar del estudiante, laboratorios de la Universidad, etc).
10. Aquellos de ustedes que presenten alguna dificultad con su inscripción o tengan inconvenientes técnicos, por favor contactarse con la Coordinadora o Coordinación adjunta antes de las 20:00hs. del día de la entrega, a través de los mails, alamon@ort.edu.uy, ó gervaz@ort.edu.uy o telefónicamente al 29021505 - int 1156 u 1138