



# Facultad de UNER Ingeniería

## Trabajo Práctico N°1

### Algoritmos y Estructuras de Datos

Profesores: Dr. Javier Eduardo Diaz Zamboni, Nicolet, Jonathan

Integrantes: Luis Alberto Rodriguez, Luis Diego Rodriguez

## LISTA DOBLE ENLAZADA

- Clase Nodo: Definimos una clase llamada **Nodo** que tiene tres atributos: **dato**, **siguiente**, y **anterior**.
  - Al atributo **dato** le asignamos para que almacenará el valor del nodo.
  - A los atributos **siguiente** y **anterior** le asignamos para que sirvan de punteros en la cual apuntan al siguiente nodo y al nodo anterior.
  - En el constructor **\_\_init\_\_**, inicializa el nodo con el dato proporcionado y establece los punteros como None.
- Clase Lista Doble Enlazada: Definimos una clase llamada **ListaDobleEnlazada** que representa la lista doblemente enlazada, en la cual un elemento de la lista estará representado por un nodo.
  - Le asignamos atributos como **cabeza**, **cola** y **tamanio** para mantener el estado de la lista.
  - Incluimos los métodos **\_\_iter\_\_** y **\_\_next\_\_** para permitir que la lista sea iterable. Estos métodos son fundamentales para recorrer los elementos de la lista.
  - Implementamos un método también llamado **tamanio** que nos permite obtener el tamaño de la lista doblemente enlazada. Este método devuelve el valor almacenado en el atributo **tamanio**, que mantiene un seguimiento de la cantidad de nodos en la lista.
  - Implementamos métodos para agregar nodos al principio (**agregar\_al\_inicio**) y al final (**agregar\_al\_final**) de la lista, los cuales nos permite ir agregando elementos a nuestra lista.
  - Implementamos un método para insertar un nodo en una posición específica. (**insertar**)
  - Implementamos un método para copiar la lista en una nueva lista (**copiar**).
  - Implementamos un método para extraer un nodo en una posición específica (**extraer**), el cual nos permite extraer elementos en cualquier posición de la lista.
  - Implementamos un método para invertir el orden de los nodos de la lista (**invertir**).
  - Implementamos un método que tiene la capacidad de concatenar la lista con otra lista (concatenar).

- Sobrecargamos al operador “+” para facilitar la concatenación de listas.
- Implementa un método para ordenar la lista utilizando el algoritmo de ordenamiento rápido (**ordenar**).



La gráfica es una muestra del comportamiento del rendimiento del método ordenar de la Clase ListaDobleEnlazada que contiene el algoritmo de ordenamiento rápido o quicksort.

## JUEGO DE GUERRA

- Clase Cartas: Definimos una clase llamada **Cartas** que contiene tres atributos: **valor**, **palo**, **boca\_abajo**.
  - Definimos los valores de las cartas desde ('2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A') y los palos (♠, ♥, ♦, ♣).
  - Le asignamos al atributo **valor** que representara los valores de la carta y al atributo **palo** le asignamos el palo correspondiente.
  - Asignamos al atributo **boca\_abajo** para que sea un atributo booleano que indica si la carta está boca abajo o boca arriba. Inicialmente, todas las cartas están boca abajo.
- Clase Mazo: Definimos la clase **Mazo** que inicializa un mazo de cartas, las baraja y lo distribuye entre dos jugadores usando una lista doblemente enlazada. La clase contiene 3 atributos: mazo, **jugador\_1** y **jugador\_2**.
  - Le asignamos al atributo mazo una lista vacía de python.
  - A los atributos tanto **jugador\_1** como **jugador\_2** le asignamos para que sean una Lista Doble Enlazada.
  - Dentro de la clase implementamos instancias de cartas combinando los valores y los palos. Por ejemplo, se crearán cartas como 2♠, 3♥, Q♦, etc. Estas se agregaran a lista mazo (Las 52 cartas que tendrá el mazo).
  - Agregamos para que opcionalmente, se pueda proporcionar una semilla para la generación aleatoria, que se utilizará para barajar el mazo.
  - Implementamos un método que nos permite barajar las cartas que tiene la lista mazo. El método utiliza la función random.sample para mezclar las cartas de manera aleatoria. (**barajador**) dentro de este método, tenemos otro método que recorre la lista de cartas y agrega cada carta a una **ListaDobleEnlazada** que convierte la lista de cartas “mazo” en un objeto de **ListaDobleEnlazada** (**pasador**).
  - Implementamos un método dentro del método barajador que toma el mazo de cartas y lo reparte en partes iguales entre dos jugadores (26 cartas cada uno), que se extraen del

mazo y se agregan a sus respectivos mazos de jugadores, que también son representados como listas doblemente enlazadas. (**repartir**).

- Implementamos un método que permite colocar una carta específica en la parte superior del mazo. (**poner\_arriba**).
- Implementamos un método que permite colocar una carta específica en la parte inferior del mazo. (**poner\_abajo**).
- Implementamos un método que permite sacar una carta de la parte superior del mazo. extrae la primera carta de la parte superior del mazo. (**sacar\_arriba**).

- Clase JuegoGuerra: Definimos la clase JuegoGuerra que representa el juego de Guerra.

- Implementamos un constructor **\_\_init\_\_**, donde se inicializan los siguientes atributos:  
**turnos**: Un contador de turnos. **max\_turnos**: Número máximo de turnos permitidos antes de declarar un empate. **mazo**: Se crea una instancia de la clase Mazo para crear y repartir las cartas a los jugadores. **mazo\_jugador1** y **mazo\_jugador2**: Mazos de cartas para el jugador 1 y el jugador 2. **mesa**: Lista doblemente enlazada que representa la mesa donde se colocarán las cartas durante un turno.
- Implementamos un método llamado **jugar\_turno** que permite jugar un turno del juego. Durante un turno:
  - Se verifica si se ha alcanzado el límite de turnos (**max\_turnos**) para evitar juegos infinitos.
  - Se verifica si uno de los jugadores se queda sin **cartas** en su **mazo**; en ese caso, se declara al otro jugador como ganador.
  - Se extraen las **cartas** superiores de los mazos de ambos jugadores, se revelan y se muestran.
  - Se comparan las **cartas** para determinar el ganador del turno:
  - El jugador con la carta de mayor valor (según la lista valores) gana el turno y se queda las cartas que están en la **mesa** y las agrega a el **mazo del jugador**.

- En caso de empate, se inicia una "guerra".
- Se ocultan las cartas en la mesa nuevamente.
- Implementamos un método llamado **guerra** que permite manejar una "guerra" en el juego:
  - Se extraen tres cartas adicionales de los mazos de cada jugador y se colocan en la **mesa**.
  - Se muestran las **cartas** en la **mesa** durante la guerra.
  - Se compara una quinta **carta** de cada jugador para determinar el ganador.
  - Las cartas ganadas se agregan al **mazo** del jugador ganador.
  - Si hay otro empate, se inicia otra **guerra** recursivamente.
- Implementamos un método llamado **jugar** que permite jugar una partida completa:
  - Se ejecuta un bucle mientras el número de turnos no supere el límite (**max\_turnos**).
  - Si un jugador se queda sin cartas en su **mazo**, se finaliza la partida.
  - Se llama al método **jugar\_turno** en cada iteración del bucle.
  - Al final de la partida, se verifica el resultado y se imprime quién ganó o si la partida terminó en empate.

## ORDENAMIENTO EXTERNO

- Función **mezclar\_bloques**: Implementamos esta función que se encarga de mezclar bloques de líneas ordenadas de varios archivos de entrada en un archivo de salida. Su funcionamiento lo podemos resumir de la siguiente manera:
  - Abre los archivos de entrada en modo lectura.
  - Abre el archivo de salida en modo escritura.
  - Inicializa una lista llamada **valores\_actuales** para mantener los valores leídos de cada archivo.
  - Lee la primera línea de cada archivo y almacena los valores en **valores\_actuales**.
  - Mientras haya valores en **valores\_actuales**, encuentra el valor mínimo, lo escribe en el archivo de salida y actualiza los valores leídos.
  - Cierra todos los archivos de entrada y el archivo de salida.
- Función **ordenar\_bloque** y **particion**: Estas funciones que implementamos ejecutan el algoritmo de ordenación ordenamiento rápido en una lista. La función **ordenar\_bloque** se utiliza para ordenar un bloque de líneas. **particion** es una función auxiliar que realiza la partición de la lista en ordenamiento rápido.
- Función **ordenar\_archivo\_grande**: Esta función que implementamos principalmente coordina todo el proceso de ordenación de un archivo grande:
  - Obtiene el directorio temporal.
  - Inicializa una lista llamada **archivos\_temporales** para mantener los nombres de los archivos temporales generados.
  - Abre el archivo de entrada en modo lectura y procesa el archivo en bloques.
  - Para cada bloque, filtra líneas vacías o no numéricas, ordena el bloque utilizando **ordenar\_bloque**, y escribe las líneas ordenadas en un archivo temporal.
  - Luego, mezcla los archivos temporales en un solo archivo ordenado final utilizando la función **mezclar\_bloques**.

- Los archivos temporales se eliminan a medida que se mezclan.
- Finalmente, el archivo de salida ordenado se renombra y se devuelve como resultado.