Facultad de Ingeniería
UNER

Informe sobre la implementación y análisis de algoritmos de ordenamiento

Algoritmos y Estructura de Datos.

Integrantes:

- Zapata Mariana Gabriela.
- Weimer Valentin.
- Kerbs Javier.

2° Cuatrimestre, 2025

Problema 2: Juego de Cartas “Guerra”

1. Introducción.

El segundo ejercicio consistió en simular el juego cartes guerra con el fin de implementar algoritmos como la clase mazo, utilizando una lista doblemente enlazada para almacenar objetos de tipo carta, aplicando conceptos de estructuras de datos dinámicas como el manejo de insercion y extraccion en listas, como también el manejo de excepciones al intentar interactuar con mazos vacíos.

El objetivo fue **aplicar los conceptos de la materia para resolver el problema planteado**, comprobando que el algoritmo funciona correctamente con los datos de prueba proporcionados o generados aleatoriamente.

2. Solución implementada.

Clase Mazo.

- **Estructura interna:**
  Se utiliza la ListaDobleEnlazada del Problema 1 para almacenar los objetos Carta. Esto permite operaciones eficientes de inserción y extracción tanto al inicio como al final del mazo.

- **Operaciones implementadas:**

    - poner_carta_arriba(carta): Inserta la carta al inicio del mazo (para repartir cartas al jugador).

    - poner_carta_abajo(carta): Inserta la carta al final del mazo (para cuando el jugador gana cartas).

    - sacar_carta_arriba(mostrar=False): Extrae la carta del inicio del mazo. Si el mazo está vacío, lanza la excepción DequeEmptyError.

    - __len__(): Devuelve la cantidad de cartas en el mazo.

    - __str__(): Representa el mazo como cadena de cartas visibles.

- **Manejo de errores:**
  La extracción de cartas de un mazo vacío está protegida mediante la excepción DequeEmptyError, garantizando que el juego no falle ante esta situación.

- **Compatibilidad con el test y JuegoGuerra:**
  La firma de los métodos coincide exactamente con los requerimientos de los tests proporcionados, asegurando que todas las pruebas unitarias pasen correctamente.

3. Resultados.

Se realizaron pruebas automáticas con los tests provistos:

1. **TestMazo**
    - poner_carta_arriba y sacar_carta_arriba funcionan correctamente.
    - poner_carta_abajo y extracción del inicio funcionan en orden esperado (FIFO/cola).
2. **TestJuegoGuerra**
    - Las partidas de ejemplo con random_seed devuelven el ganador correcto y el número de turnos esperado.
    - Casos de empate y guerras se manejan correctamente.
    - No se modifica ningún test ni la lógica del juego.

Conclusión de pruebas: cumple con los objetivos planteados en el enunciado, genera la salida esperada para todos los casos de prueba y si corresponde, se muestran gráficas o tablas con los datos procesados.

4. Análisis de complejidad.

El análisis de complejidad de las operaciones principales es el siguiente:

- poner_carta_arriba: O(1), ya que solo se modifica el puntero al inicio del mazo.
- poner_carta_abajo: O(1), solo se modifica el puntero al final del mazo.
- sacar_carta_arriba: O(1), se elimina el nodo del inicio sin recorrer la lista.
- __len__: O(1), se devuelve un contador interno que mantiene el número de cartas.
- __str__: O(n), donde n es el número de cartas en el mazo, porque se recorre toda la lista para generar la representación en cadena.

Justificación:

Se utiliza una lista doblemente enlazada, por lo que insertar o eliminar al inicio o final es constante. Recorrer todas las cartas para imprimir el mazo requiere tiempo lineal.

5. Conclusión general.

- La clase Mazo permite administrar cartas de manera eficiente usando una **Lista Doble Enlazada**.

- Todas las operaciones cumplen con los requerimientos de los tests de la cátedra y de la clase JuegoGuerra.

- El manejo de errores con DequeEmptyError garantiza que no se produzcan fallas por extracción de mazo vacío.

- La implementación es eficiente y escalable, manteniendo complejidad O(1) para la mayoría de las operaciones críticas del juego.